МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Навчальний тренажер для практичної роботи "Фізичне моделювання бази даних" із дисципліни "Бази даних та інформаційні системи"»
здобувача групи ІН - 91 Зулюкова Івана Юрійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Іван ЗУЛЮКОВ
(підпис)

Керівник,
старший викладач,
кандидат технічних наук Борис КУЗІКОВ _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Зулюкова Івана Юрійовича

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для фізичного моделювання баз даних. 3) Розробка навчального тренажеру.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для фізичного моделювання баз даних*		
3	*Розробка навчального тренажеру*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 51 ст., 24 рис., 2 табл., 21 використане джерело.

Обґрунтування актуальності теми роботи – фізичне моделювання баз даних є важливим етапом проектування бази даних та дозволяє значно полегшити пошук та виправлення помилок проектування. Нехтування цим етапом може призвести до великої кількості міграцій для збереження створених даних. Додаток попередньо використаний для виконання практичної роботи "Фізичне моделювання бази даних" застарілий та не працює на більшості сучасних систем. Сучасні альтернативи переважно комерційні або мають обмежений функціонал вимушуючи писати скрипт та малювати діаграму окремо та вручну.

Об’єкт дослідження – фізичне моделювання схеми баз даних

Мета роботи – створення додатку для автоматичного створення діаграм на основі інформації про базу даних.

Методи дослідження – метод зворотньої розробки та інструмент моделювання баз даних

Результат – створено додаток що має можливість аналізувати як код створення бази даних написаний на діалекті Oracle, MS SQL Server, Sybase, PostgreSQL, MySQL, MariaDB, DB2, H2, HSQLDB, Derby або SQLite так і вже розгорнуту базу даних в усіх вище зазначених СУБД окрім Sybase та створювати на її основі діаграму в нотації Баркера або нотації «Вороняча лапка» що зберігається в одному форматі зі списку: PNG, SVG, DOT, XDOT, PLAIN, PLAIN_EXT, PS, PS2(PDF), JSON(для XDOT), JSON0(для DOT).

Рекомендації щодо використання – при запуску додатку без аргументів або з аргументом -h або --help додаток надає допоміжне повідомлення зі стандартизованими інструкціями використання. Додаток потребує щоб ключові слова DELETE та UPDATE синтаксису [ON DELETE referential_action] або [ON UPDATE referential_action] були в верхньому регістрі. Бажано вказувати лише ім’я вихідного файлу адже в цьому випадку розширення буде згенеровано автоматично на основі обраного формату.

SQL, ERD, JAVA, DOT, GRAPHVIZ, НОТАЦІЯ БАРКЕРА, НОТАЦІЯ «ВОРОНЯЧА ЛАПКА», JSQLPARSER, GRAPHVIZ-JAVA, PICOCLI

ЗМІСТ

ВСТУП

Проектування є дуже важливим етапом створення баз даних. Задачею цього етапу є якомога більше наближення первісно розгорнутої бази даних до її кінцевого вигляду для зменшення кількості міграцій. Більш того якісно спроектована база даних значно полегшує процес внесення змін що призводить до значного зменшення кількості втраченого часу.

Типовою формою документування структури бази даних є діаграми «сутність-зв'язок»(Entity-Relationship). Діаграма «сутність-зв’язок» – це модель даних що дозволяє описувати сутності та зв’язки між ними за допомогою блок-схем узагальнених певною нотацією. В реляційній базі даних кожен рядок кожної таблиці є екземпляром сутності. При цьому зовнішні ключі є покажчиками зв’язків між цими сутностями.

Задачею ER діаграм є концептуальне представлення бази даних для полегшення сприйняття та знаходження способів можливого покращення структури бази даних.

Додаток CASE Studio 2 що використовувався для виконання практичної роботи "Фізичне моделювання бази даних" із дисципліни "Бази даних та інформаційні системи" не працює в операційній системі Windows 10 що стало актуальною проблемою під час переходу на дистанційне навчання. Переважна більшість альтернатив є комерційними продуктами. Альтернативи з відкритим кодом є дуже неточними та потребують як створення діаграми так і написання коду вручну.

Об’єктом дослідження було обрано фізичне моделювання баз даних.

Метою цієї роботи було створити додаток для автоматичного створення діаграм на основі інформації про базу даних для більшості з найбільш популярних СУБД.

Для досягнення поставленої мети роботи сформульовано наступні задачі дослідження:

1. Провести огляд існуючих засобів інжинірингa та ре-інжинірингa реляційних баз даних
2. Проаналізувати нотації документування схеми бази даних
3. Спроектувати додаток документування структури бази даних.
4. Провести імплементацію додатку у відповідності до сформульованих вимог.
5. Провести тестування, проаналізувати сильні та слабкі сторони, сформулювати можливі шляхи покращення роботи додатку.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Засоби проектування баз даних

Найлогічнішою заміною CASE Studio 2 [1] була б нова його версія що була перейменована в Toad Data Modeler [2] але окрім того що вона є комерційним продуктом її та подібний продукт під назвою Erwin Data Modeler [3] неможливо придбати чи навіть завантажити пробну версію перебуваючи в Україні. Спричинено це блокуванням доступу українських IP адрес до власних сервісів компанією Quest Software. Нижче наведено приклад діаграми створеної в CASE Studio 2 (див. рис 1.1) використовуючи застарілу операційну систему.

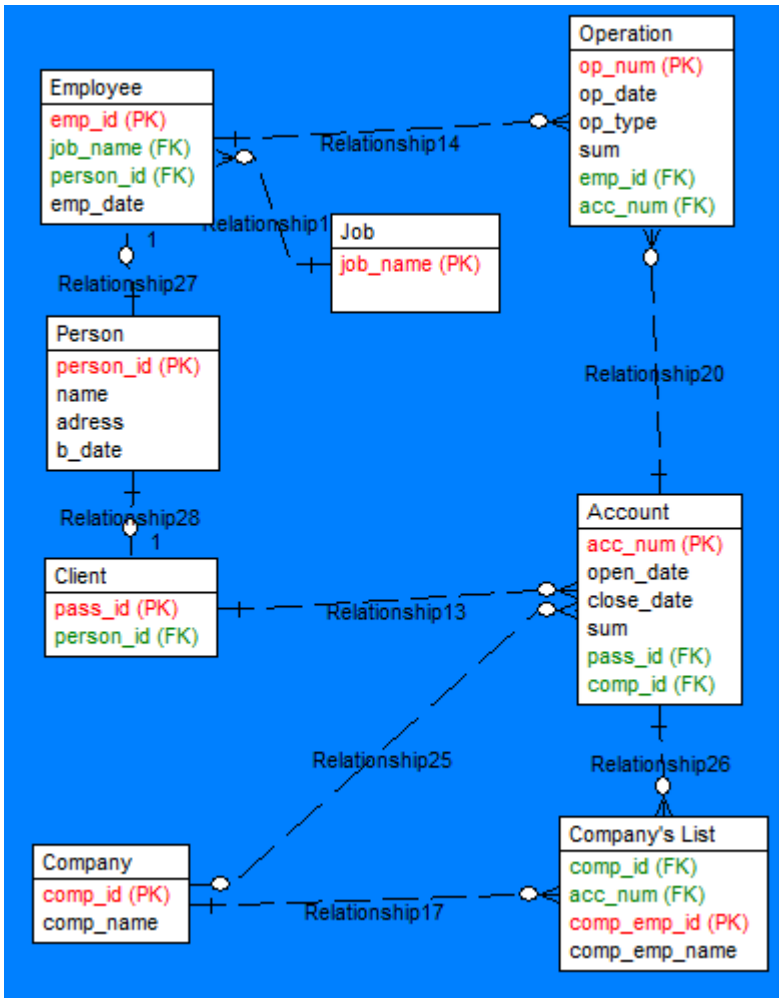

Рисунок 1.1 Приклад діаграми створеної в CASE Studio 2

Окрім цього існують додатки Lucidchart [4] та draw.io [5] що мають обмежений функціонал та дозволяють лише моделювання діаграм без можливості імпорту або експорту баз даних при використанні безкоштовної версії без сторонніх модифікацій.

Додаток dbdiagram.io [6] дозволяє проектувати бази даних використовуючи власну мову розмітки DBML що схожа з SQL та іншими мовами кодування діаграм. Також цей додаток дозволяє імпортувати DDL в діалектах MySQL, PostgreSQL та SQL Server що є доволі обмеженим вибором. Більш того підтримка цих діалектів є неповною через що наприклад використання типу Double precision в діалекті PostgreSQL є неможливим. Нижче наведено приклад діаграми згенерованої за допомогою додатку dbdiagram.io (див. рис 1.2).

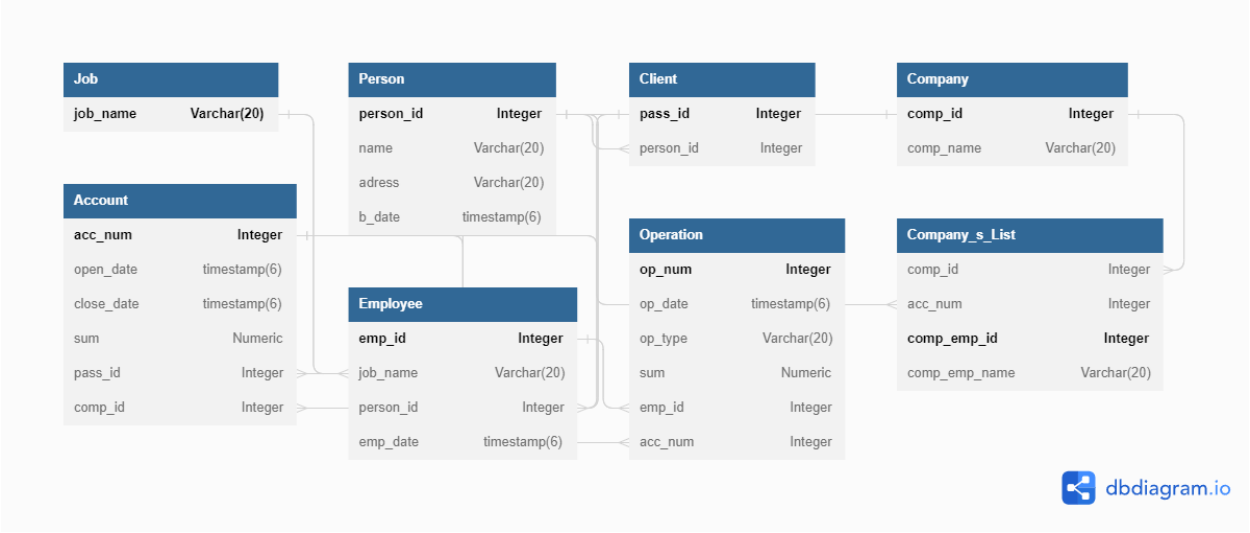

Рисунок 1.2 Приклад діаграми згенерованої dbdiagram.io

Додатки SchemaCrawler [7] та SchemaSpy [8] використовуються для реінжинірінга реляційних баз даних та є доволі схожими між собою. Обидва з них безкоштовні та мають відкритий код. Окрім цього обидва є найближчими до розв’язання поставленої задачі але на жаль обидва розраховані на зворотну розробку вже розгорнутих баз даних що значно зменшує гнучкість при проектуванні через потребу в розгортанні скрипту після кожної його зміни.

Більш того SchemaSpy потребує завантаження драйверів систем управління базами даних вручну. Нижче наведені приклади діаграм згенерованих за допомогою додатків SchemaCrawler (див. рис 1.3 та 1.4) та SchemaSpy (див. рис 1.5).

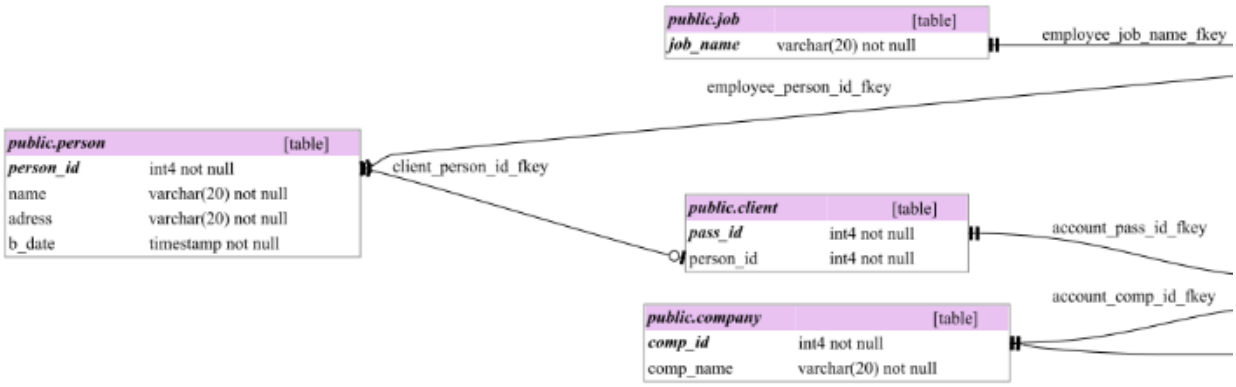

Рисунок 1.3 Приклад діаграми згенерованої SchemaCrawler(частина 1)

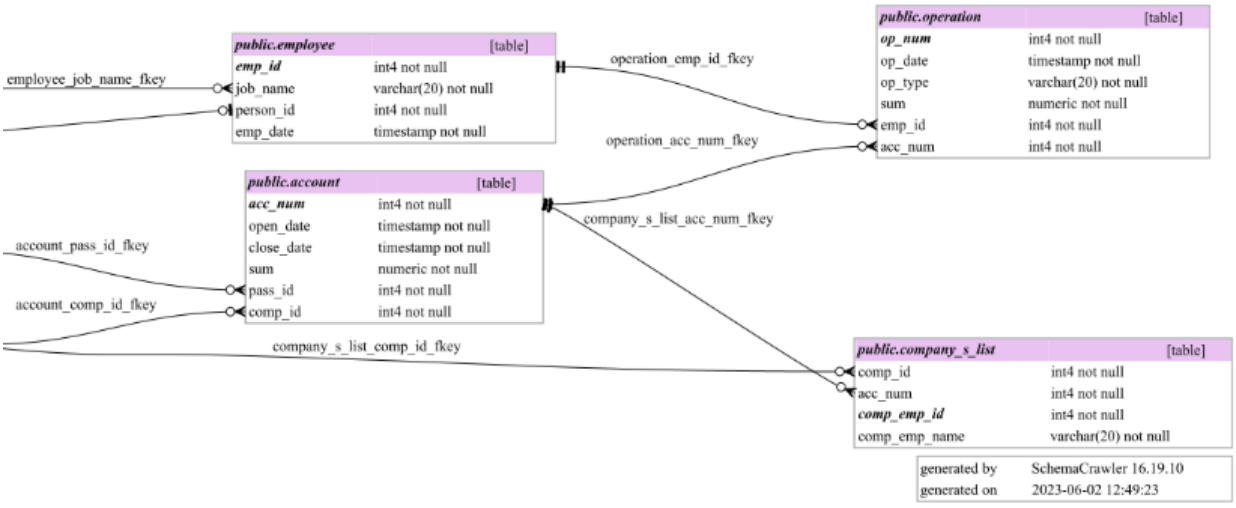

Рисунок 1.4 Приклад діаграми згенерованої SchemaCrawler(частина 2)

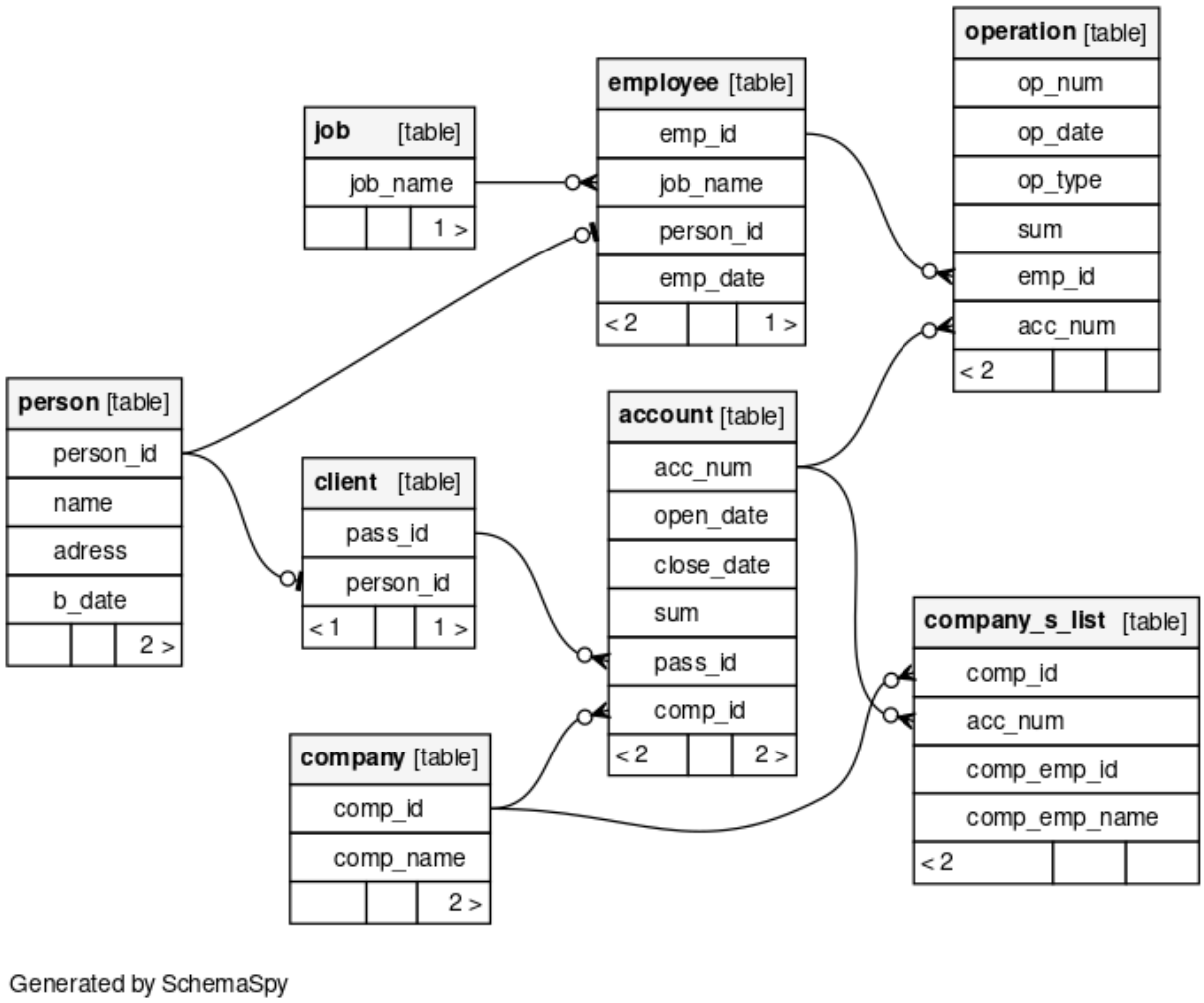

Рисунок 1.5 Приклад діаграми згенерованої SchemaSpy

1.2 ERD нотації

Існує велика кількість ERD нотацій але для проектування баз даних зазвичай використовують такі нотації як: «Вороняча лапка», Баркера, IDEF1X та UML.

«Вороняча лапка»

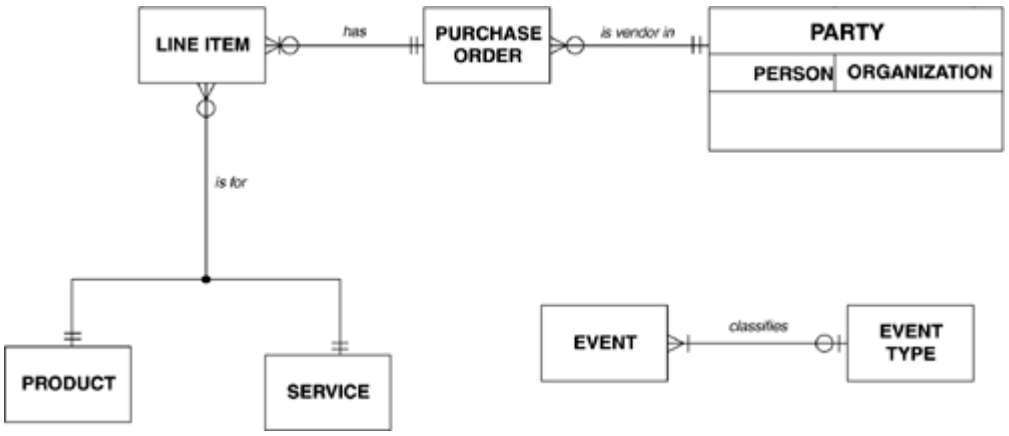

Рисунок 1.6 Приклад діаграми в нотації "Вороняча лапка" [9]

Назва нотації "Вороняча лапка" (див. рис 1.6) походить від вигляду стрілки що використовується для позначення участі в зв’язку багатьох екземплярів сутності що є одним з двох значень властивості кардинальності.

Інше значення кардинальності вказує на участь одного екземпляру сутності та позначається одинарною горизонтальною рискою.

Другою властивістю зв’язку є модальність. Ці дві властивості поєднуються та утворюють чотири види стрілок що характеризують нотацію «вороняча лапка». Зв’язок може бути обов’язковим що позначається горизонтальною рискою або необов’язковим що позначається колом.

Ще одною характеристикою є сила залежності. Залежність є сильною якщо первинний ключ залежної сутності містить первинний ключ сутності від якої вона залежить. Позначається це суцільною лінією. В іншому випадку залежність є слабкою що позначається штрихованою лінією.

Для інших характеристик визначеного стандарту немає але первинний ключ досить часто позначають зіркою(*). В цій роботі було вирішено використовувати інший популярний стандарт позначень оснований на перших літерах назв обмежень: PK – первинний ключ, FK – зовнішній ключ, U – обмеження унікальності та NN – заборона нульових значень.

Нотація Баркера

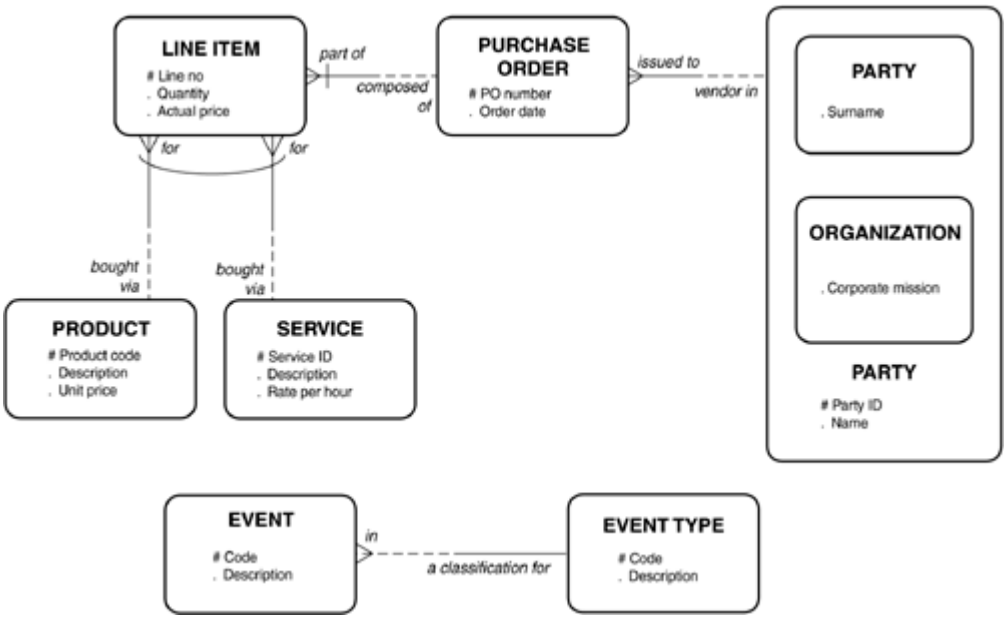

Рисунок 1.7 Приклад діаграми в нотації Баркера [9]

В нотації Баркера (див. рис 1.7) обов’язковий зв’язок позначається суцільною лінією а необов’язковий – штриховою.

В цій роботі вважається що екземпляр сутності завжди може існувати без залежних від нього екземплярів сутності.

Участь в зв’язку багатьох екземплярів сутності позначається такою ж стрілкою як і в нотації «вороняча лапка» тоді як участь одного екземпляру сутності позначається відсутністю стрілки.

Також до стрілки може бути додана горизонтальна риска якщо залежність є сильною що виглядає аналогічно обов’язковому зв’язку нотації «вороняча лапка».

Окрім цього ця нотація використовує зірку(*) для позначення унікального ідентифікатору та решітку(#) для позначення заборони на нульові значення. У випадку якщо заборона на нульові значення відсутня рядок позначається колом(O).

IDEF1X

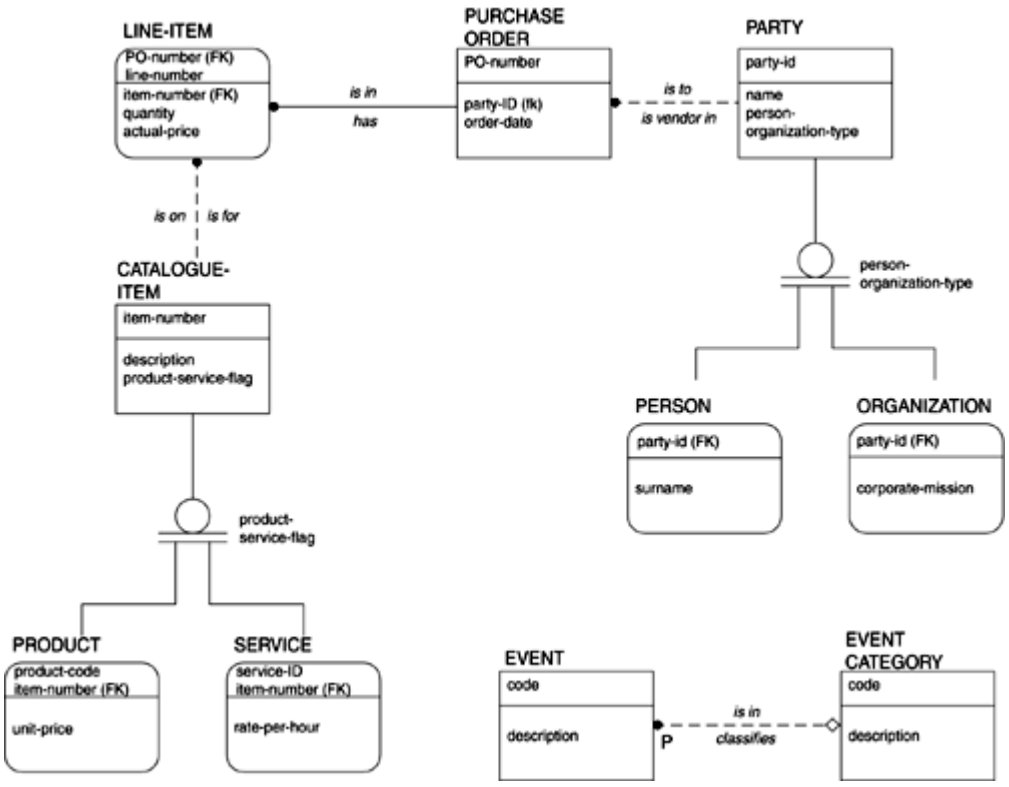

Рисунок 1.8 Приклад діаграми в нотації IDEF1X [9]

В нотації IDEF1X (див. рис 1.8) сутності поділяються на залежні та незалежні. Незалежні позначаються прямокутниками з гострими кутами а залежні – з заокругленими.

Кожна сутність розділяється на дві частини. У верхній вказуються первинні ключі тоді як в нижній вказуються всі інші атрибути. Зовнішні ключі позначаються FK. Ключі що не є первинними але унікально ідентифікують екземпляр сутності називаються альтернативними та позначаються AK.

Стрілка в цій нотації завжди має вигляд крапки. Властивості кардинальності та модальності поєднані в можливу кількість залежних елементів що позначається над стрілкою:

– Відсутність позначки свідчить про довільну кількість.
– Позначка P свідчить про обов’язкову наявність хоча б одного елементу(1..* в нотації UML) та відповідає забороні на нульові значення.

– Позначка Z свідчить про наявність не більше ніж одного елементу(0..1 в нотації UML) та відповідає обмеженню унікальності.
– В разі комбінації двох обмежень вказується кількість залежних елементів позначкою 1.

UML

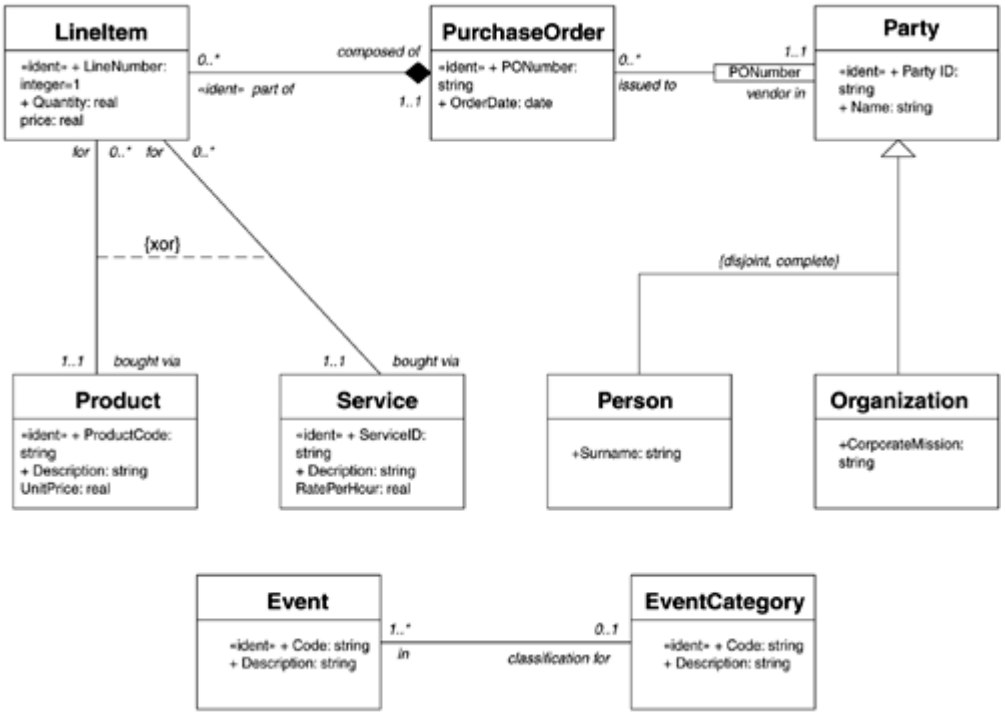

Рисунок 1.9 Приклад діаграми в нотації UML [9]

При моделюванні баз даних в нотації UML (див. рис 1.9) використовуються такі типи відношень як: асоціація, наслідування, агрегація та композиція.

Відношення асоціації показує що екземпляри сутності пов’язані з екземплярами іншої сутності. Використовується коли відношення не можна конкретизувати до іншого типу. Буває унарним та бінарним. Для позначення унарної асоціації використовується суцільна лінія з відкритою стрілкою.

Для позначення бінарної асоціації використовується суцільна лінія без стрілки. Також в асоціації може брати участь більше двох сутностей.

Відношення наслідування показує що сутність є більш конкретною формою іншої. Позначається суцільною лінією з порожньою трикутною стрілкою.

Відношення агрегації показує що сутність є частиною цілого але може існувати незалежно від цілого. Відповідає слабкій залежності в інших нотаціях. Позначається суцільною лінією зі стрілкою у вигляді порожнього ромбу.

Відношення композиції показує що сутність є частиною цілого та не може існувати незалежно від цілого. Відповідає сильній залежності в інших нотаціях. Позначається суцільною лінією зі стрілкою у вигляді заповненого ромбу.

Властивість модальності є поєднаною з властивістю кардинальності в можливу кількість залежних елементів що позначається над стрілкою:

- Позначка 0..* свідчить про довільну кількість.
- Позначка 1..* свідчить про обов’язкову наявність хоча б одного елементу та відповідає забороні на нульові значення.
- Позначка 0..1 свідчить про наявність не більше ніж одного елементу та відповідає обмеженню унікальності.
- Позначка 1 свідчить про наявність одного та тільки одного елементу та відповідає комбінації двох обмежень.

Ця нотація переважно використовується для об’єктно-орієнтованого моделювання через що її використання для моделювання реляційних баз даних є сумнівним на думку частини експертів.

Результати аналізу

Нижче наведено порівняння нотацій у вигляді таблиці (див. рис 1.10).

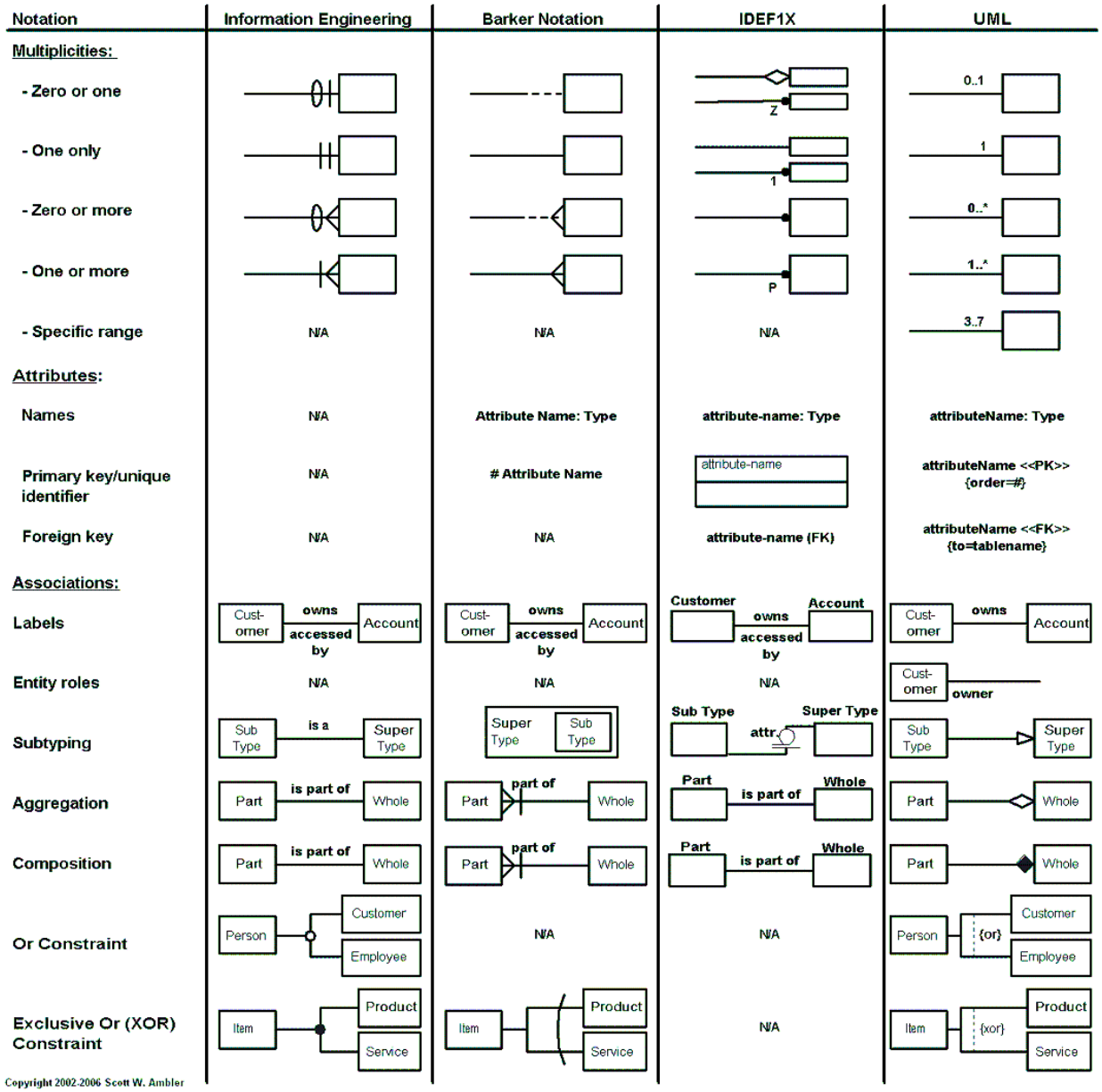

Рисунок 1.10 Порівняння нотацій документування схеми бази даних [10]

Окрім особливостей кожної нотації існують певні особливості фізичного моделювання що є спільними для усіх нотацій.

По перше кожному зовнішньому ключу завжди відповідає лише один первинний ключ через що для створення зв’язку багато-до-багатьох завжди потрібно створювати проміжну таблицю (див. рис 1.11).

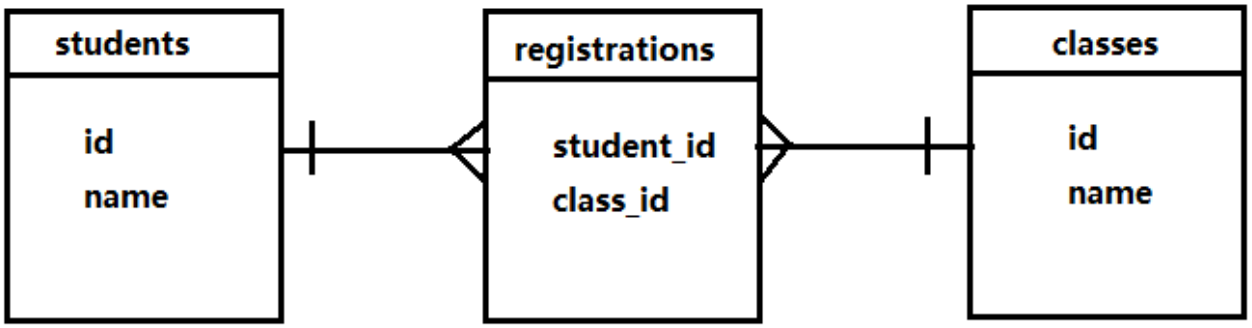

Рисунок 1.11 Приклад реалізації зв’язку багато-до-багатьох

По друге неможливо забезпечити існування екземпляру сутності залежного від створюваного екземпляру сутності без створення додаткових обмежень що не є рекомендованою практикою через виникнення ситуації взаємоблокування адже зовнішній ключ може набувати тільки тих ненульових значень яким відповідають значення первинного ключа.

Таким чином при фізичному моделюванні бази даних зв’язок первинного-до-зовнішнього ключа є одним-до-одного коли на зовнішній ключ накладено обмеження унікальності та обов’язковим-до-необов’язкового коли на зовнішній ключ накладена заборона на нульові значення.

1.3 Постановка задачі дослідження

Метою додатку, що розробляється є побудова ER-діаграми. Джерелом даних є SQL скрипт у текстовому форматі, що містить DDL запити, або наперед створена база даних. Спираючись на результатах аналізу відомих аналогів ключовими вимогами до додатку є:

1. Додаток повинен працювати на Windows 10 64bit та MacOS X
2. Додаток повинен мати підтримку діалектів MySQL, PostgreSQL, Oracle, SQL Server та SQLite
3. Додаток повинен мати підтримку СУБД MySQL, PostgreSQL, Oracle, SQL Server та SQLite
4. Додаток повинен надавати результаті у форматах PNG та DOT
5. Додаток повинен підтримувати нотації Баркера та «Вороняча лапка»

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Проектування додатку

Першим етапом рішення поставленої задачі є створення моделі в якій буде зберігатись інформація про таблиці бази даних та відношення між ними. Об'єкти цієї моделі створюються та заповнюються окремими контролерами залежно від того звідки була отримана інформація про базу даних. Заповнені об'єкти моделі використовуються представленням для формування діаграми. Головний клас обирає потрібний контролер на основі отриманих аргументів та за допомогою представлення формує діаграму в обраній нотації що потім записується в зазначений файл в зазначеному форматі.

На основі плану було створено діаграму класів додатку (див. рис 2.1).

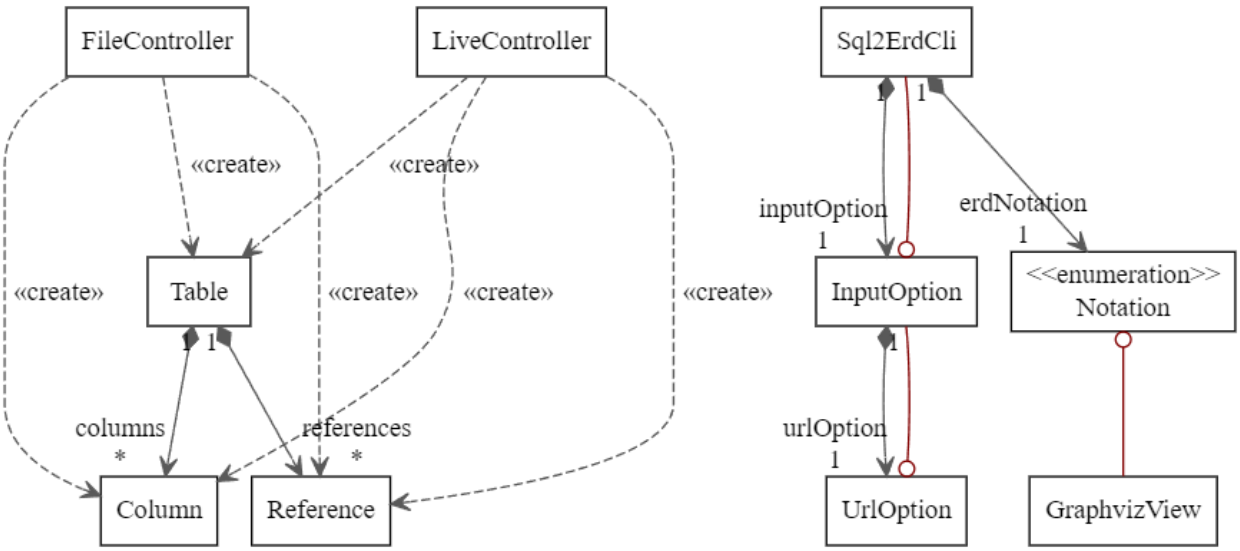

Рисунок 2.1 Діаграма класів та зв'язків між ними

2.2 SQL Парсинг

Під час огляду наявних шляхів парсингу SQL коду було вирішено використати сторонню бібліотеку для розширення списку підтримуваних діалектів. В результаті пошуків було обрано чотири найкращі кандидати.

Zql

Zql [11] є простим SQL парсером згенерованим за допомогою JavaCC. Перший реліз було випущено 10 березня 1998 року. Більшість покращень відбулася в перші чотири роки після релізу після чого робота над цією бібліотекою була зупинена. У вересні 2010-го року бібліотека була відкрита для публічного використання. В 2014-ому році відбулася міграція проекту зі sourceforge до github та maven.

На жаль ця бібліотека не підійшла для рішення поставленої задачі через відсутність підтримки DDL запитів. Більш того ця бібліотека є доволі застарілою та наразі не є рекомендована до використання.

jOOQ

jOOQ [12] є рекурсивним акронімом що розшифровується як jOOQ Object Oriented Querying. Як можна зрозуміти з назви головним призначенням jOOQ є об’єктно орієнтоване створення запитів але jOOQ також здатен генерувати об’єктно орієнтовану модель на основі бази даних та парсити запити мови SQL.

На жаль для поставленої задачі генерація коду неможлива адже згенерована модель буде відповідати лише конкретній базі даних. Більш того буде згенерована модель що може містити дані про рядки коли потрібна метамодель що буде містити дані про таблиці та їх стовпці.

jOOQ SQL парсер дозволяє рішення поставлених задач але на жаль jOOQ є комерційним продуктом. Існує версія з відкритим кодом але вона працює лише з Java 17 або новіше та підтримує лише останні версії баз даних з відкритим кодом. Такі популярні бази даних як Oracle або SQL Server не підтримуються що значно обмежує функціонал результуючого додатку. Інші версії також мають 30 денний пробний період чого недостатньо навіть для розробки додатку.

General SQL Parser

General SQL Parser [13] є широко використовуваним парсером з інтуїтивно зрозумілим синтаксисом. Zql та JSQLParser рекомендують [11][14] GSQLParser як альтернативу.

На жаль General SQL Parser також є комерційним продуктом. Існує 90 денна пробна версія але обмеження часу функціонування додатку до трьох місяців не враховуючи час на розробку є недоцільно.

JSQLParser

JSQLParser [14] подібно до Zql використовує код згенерований JavaCC але на відміну від Zql є набагато сучаснішим та має підтримку DDL запитів. Також він підтримує широку низку діалектів включаючи Oracle та SQL Server що не підтримуються jOOQ.

Єдиною проблемою JSQLParser є дуже скудна документація але відкритість коду дозволяє вирішити цю проблему шляхом аналізу коду вручну.

Враховуючи специфіку поставленого завдання можна зробити висновок, що JSQLParser є ідеальним варіантом парсеру, спираючись на таблицю 2.1 порівняння бібліотек для парсингу SQL скриптів.

Таблиця 2.1 Порівняння бібліотек для парсингу SQL скриптів

Назва	Шлях розповсюдження	Підтримка	Підтримувані діалекти
Zql	безкоштовний продукт з відкритим кодом	завершилася в 2002-ому році	не підтримує DDL
jOOQ	комерційний продукт з версією з обмеженим функціоналом	продовжується	безкоштовна версія підтримує лише MySQL, PostgreSQL та SQLite з мінімальних вимог

Продовження табл. 2.1

Назва	Шлях розповсюдження	Підтримка	Підтримувані діалекти
General SQL Parser	комерційний продукт з пробною версією на 90 днів	продовжується	підтримує всі діалекти з вимог окрім SQLite
JSQLParser	безкоштовний продукт з відкритим кодом	продовжується	підтримує всі діалекти з вимог та Sybase, MariaDB, DB2, H2, HSQLDB, Derby

2.3 Зворотна розробка

Java SQL

Бібліотека java.sql є вбудованою в JDK бібліотекою що містить інтерфейси програмної взаємодії з базами даних. При цьому реалізують ці інтерфейси постачальники СУБД у вигляді драйверів що дозволяє написання коду що може взаємодіяти з будь-якою СУБД за наявності драйверу для неї. За обрання відповідного драйверу відповідає клас DriverManager.

В цій роботі було використано драйвери СУБД відповідні підтримуваним JSQLParser діалектам окрім драйверу СУБД Sybase через його пропраетарність.

За зворотну розробку в цій бібліотеці відповідає інтерфейс DatabaseMetaData [15] що отримує всі метадані з бази даних та надає доступ до них за допомогою методів.

2.4 Візуалізація діаграм

JGraphX

JGraph та JGraphX [16] є нативними Java бібліотеками візуалізації діаграм. JGraph відповідає версіям 1-5 тоді як JGraphX відповідає версії 6. На жаль розробка була завершена в 2020 році що робить даний інструмент

застарілим порівняно з іншими. До того ж JGraphX є відсутнім в Maven та не має власної через що не був використаний для вирішення поставленої задачі.

Graphviz

Graphviz [17] є програмою для візуалізації діаграм що використовує власну мову під назвою dot. Існує два шляхи використання Graphviz зсередини Java коду:

– надсилання команди до встановленого разом з пакетом Graphviz додатку командного рядку під назвою dot
– використання реалізації Graphviz на мові Javascript під назвою viz.js за допомогою Javascript двигуна

graphviz-java-api

graphviz-java-api [18] є бібліотекою що спрощує доступ до додатку dot. Була створена в 2003-ому році. Розробка була завершена в 2016-ому році через що ця бібліотека є доволі застарілою. Ще одною проблемою є вимога наявності встановленого на систему користувача пакету Graphviz. Також відсутній об’єктно-орієнтований інтерфейс через що створення коду на мові dot необхідно виконувати вручну.

graphviz-java

Бібліотека graphviz-java [19] вирішує майже всі проблеми graphviz-java-api. Розробка graphviz-java була завершена в 2021 році що робить цю бібліотеку значно сучаснішою. Існує підтримка як додатку dot так і бібліотеки viz.js що забезпечує велику гнучкість. Наявний об’єктно-орієнтований інтерфейс.

PlantUML

Потенційною альтернативою Graphviz є бібліотека PlantUML [20] що створена на основі пакету Graphviz але розширює його функціонал. Використовує власну мову розмітки під тією самою назвою що і бібліотека. На жаль також вимагає встановлення пакету Graphviz та не має об’єктно-орієнтованого інтерфейсу але покращує вигляд діаграм (див. рис 2.2).

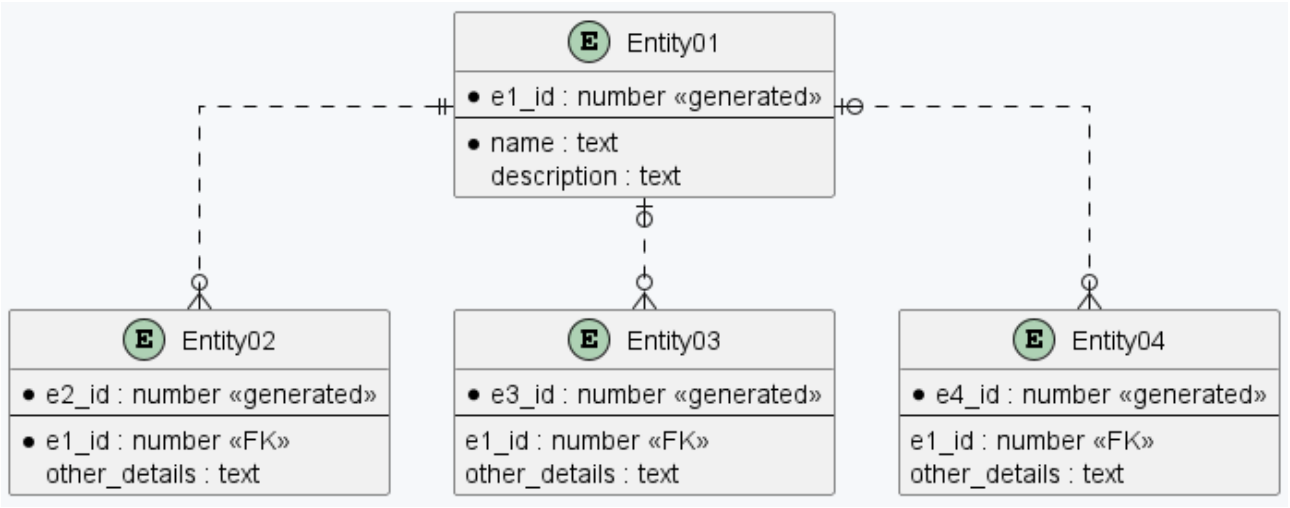

Рисунок 2.2 Приклад діаграми згенерованої за допомогою PlantUML

Враховуючи специфіку поставленого завдання можна зробити висновок, що graphviz-java є ідеальним варіантом граферу, спираючись на таблицю 2.2 порівняння бібліотек для візуалізації діаграм.

Таблиця 2.2 Порівняння бібліотек для візуалізації діаграм

Назва	Об’єктно-орієнтований інтерфейс	Підтримка	Коментарі
JGraphX	наявний	завершилася в 2020-ому році	відсутня в Maven, не має власної мови розмітки
graphviz-java-api	відсутній	завершилася в 2016-ому році	потребує встановлення Graphviz
graphviz-java	наявний	завершилася в 2021-ому році	наявна підтримка viz.js
PlantUML	відсутній	продовжується	потребує встановлення Graphviz, покращує вигляд діаграм

2.5 Додаткові інструменти

Picocli

Бібліотека picocli [21] дозволяє значно покращити досвід користувача при використанні консольних додатків. Picocli використовує анотації для позначення потрібних параметрів та дозволяє генерувати стандартизовані інструкції використання подібні до використовуваних в популярних консольних додатках. Також на відміну від написання консольного інтерфейсу вручну бібліотека Picocli робить можливим компіляцію додатку в спеціалізований під операційну систему виконавчий файл що може значно покращити швидкодію додатку.

На основі огляду було вибрано такі інструменти для реалізації додатку:

- JSQLParser для парсингу скриптів через велику кількість підтримуваних діалектів та відкритий код
- Java SQL для зворотної розробки через велику кількість підтримуваних СУБД та вбудованість в SDK
- graphviz-java для генерації діаграм через наявність об’єктно-орієнтованого інтерфейсу та відкритого коду
- Picocli для стандартизації інтерфейсу командного рядку порівняно з іншими популярними додатками командного рядку

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

Вхідні дані було сформовано на основі діаграми створеної в CASE Studio 2 під час вивчення дисципліни "Бази даних та інформаційні системи". Для демонстрації результатів виконання додатку було обрано СУБД PostgreSQL та відповідний діалект. Код створення бази даних наведено в Додатку А.

3.2 Опис програмної реалізації

Клас Table

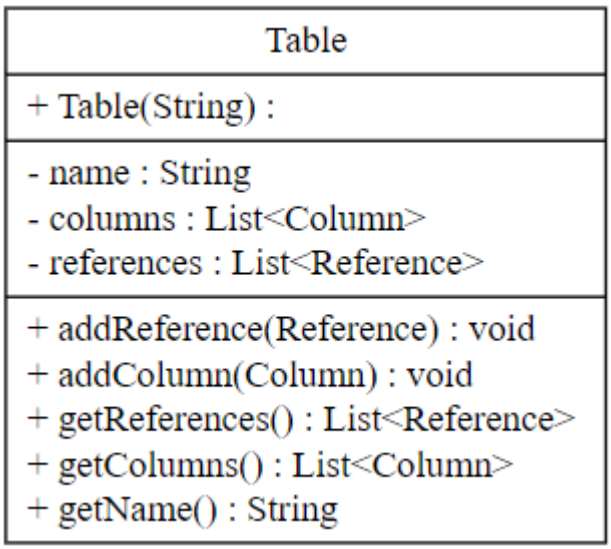

Рисунок 3.1 UML діаграма класу Table

Відповідає таблиці та містить таку інформацію про таблицю як (див. рис 3.1):

– Назва таблиці в полі name
– Список стовпців в полі columns
– Список відношень в полі references

Клас Column

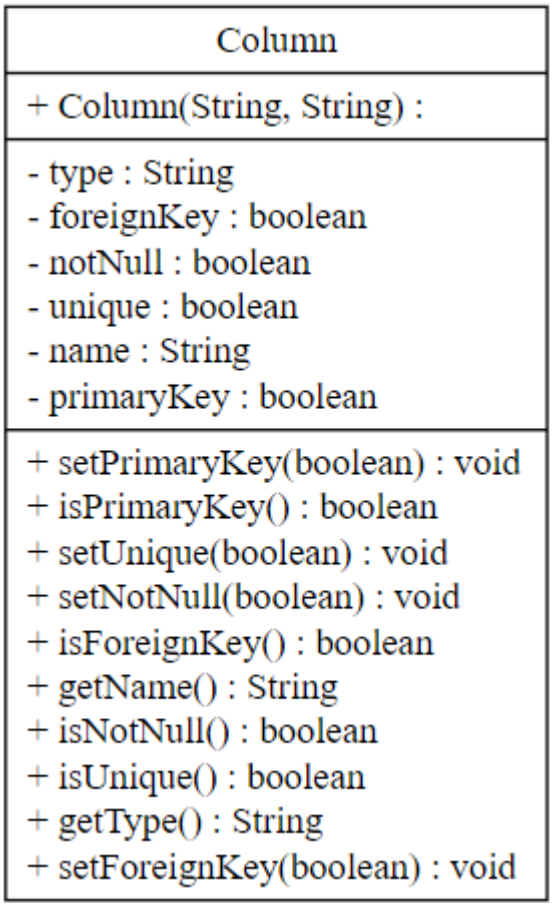

Рисунок 3.2 UML діаграма класу Column

Відповідає стовпцю та містить таку інформацію про стовпець як (див. рис 3.2):

– Назва стовпця в полі name
– Тип стовпця в полі type
– Наявність обмеження NOT NULL в полі notNull
– Наявність обмеження UNIQUE в полі unique
– Чи є стовпець частиною первинного ключа? в полі primaryKey
– Чи є стовпець частиною зовнішнього ключа? в полі foreignKey

Клас Reference

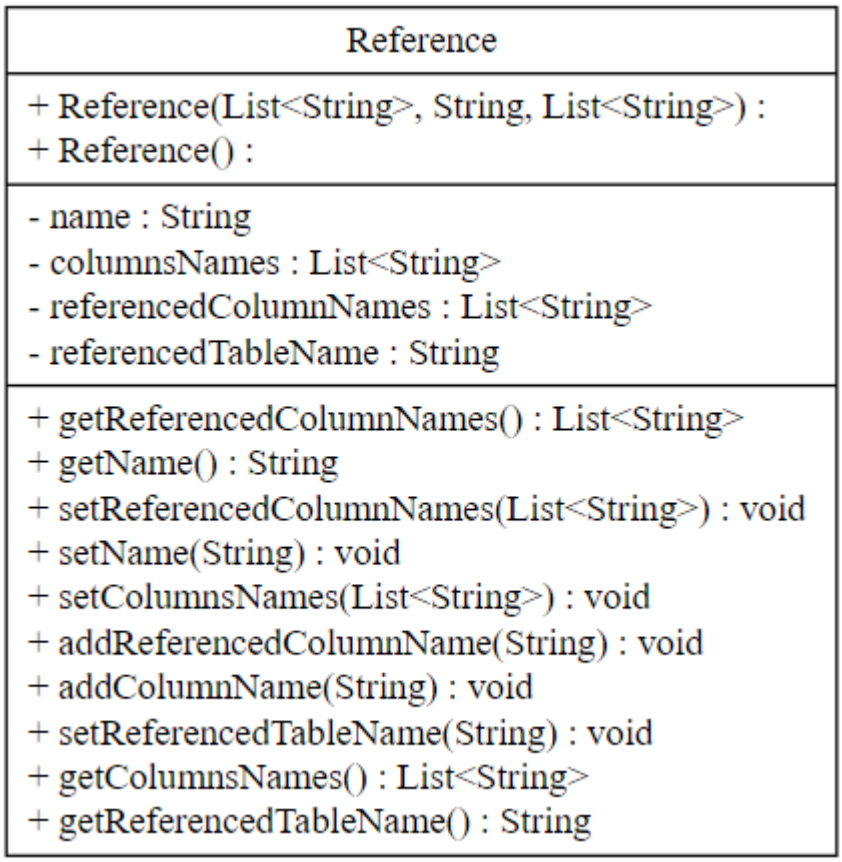

Рисунок 3.3 UML діаграма класу Reference

Відповідає відношенню та містить таку інформацію про відношення як (див. рис 3.3):

– Назва відношення в полі name (може бути null)
– Список зовнішніх ключів в полі columnsNames
– Назва таблиці від якої залежить поточна в полі referencedTableName
– Список первинних ключів з якими пов’язані зовнішні в полі referencedColumnNames

Клас FileController

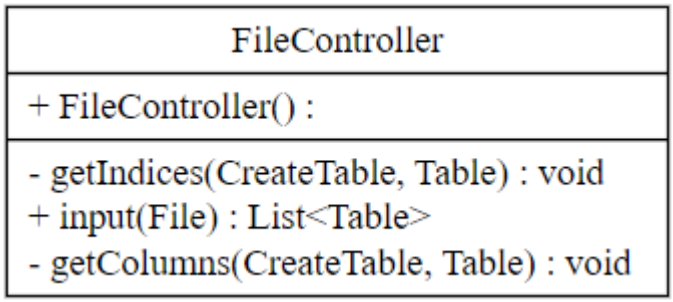

Рисунок 3.4 UML діаграма класу FileController

Даний клас (див. рис 3.4) відповідає за парсинг скрипту написаного на мові SQL в список об’єктів класу Table. Аналіз відкритого коду бібліотеки JSQLParser показав що вона спочатку розбиває скрипт на запити яким відповідає клас Statement. Далі кожен запит конкретизується у відповідний йому клас. Запитам створення таблиць відповідає клас CreateTable. Потім запити створення таблиць аналізуються порядково. Кожен рядок або визначає стовпець та відповідає класу ColumnDefinition або визначає обмеження та відповідає класу Index. Клас Index також може бути конкретизований до класу ForeignKeyIndex якщо обмеження визначає відношення.

Таким чином клас має три методи:

Метод input є основним методом класу. В якості вхідних даних отримує файл. Зчитує з нього запити та ітерує по ним. Якщо запит є запитом створення таблиці створює об’єкт класу Table з отриманою з запиту назвою. Потім за допомогою допоміжних методів додає до таблиці список стовпців та список відношень. Далі додає отриману таблицю до списку та в кінці в якості вихідних даних повертає його.

Допоміжний метод getColumns отримує інформацію про стовпці зі списку рядків визначень стовпців що зберігається в об’єкті класу CreateTable. Інформація про кожен стовпець отримується з об’єктів класу ColumnDefinition. Назва та тип стовпця отримуються напряму. Всі інші

обмеження потрапляють в список рядків та потребують ручної обробки. В разі якщо стовпець позначений як первинний ключ він також позначається як унікальний та ненульовий. В обмеженнях визначених у визначенні стовпця можна вказати відношення лише до одного стовпця і то не у всіх діалектах.

Допоміжний метод getIndices отримує інформацію про обмеження унікальності накладені на стовпці та відношення між таблицями. В разі якщо обмеження вказує на існування відношення кожен стовпець що бере участь у відношенні позначається як зовнішній ключ. В іншому випадку якщо обмеження є обмеженням унікальності та накладено на лише один стовпець то той позначається як унікальний. У випадку якщо обмеження унікальності накладено на декілька стовпців позначка унікальності не встановлюється адже унікальною є комбінація стовпців а не кожен зі стовпців. На жаль відобразити унікальність комбінацій стовпців немає можливості тому інформація про це обмеження відкидається.

Ретельний аналіз коду JSQLParser дозволив знайти помилку парсингу. Під час аналізу відношень відбувається аналіз дій у випадку видалення або оновлення первинного ключа. В синтаксисі SQL ці дії позначаються “ON DELETE…” та “ON UPDATE…”. В JSQLParser цим діям відповідають значення enum. Проблема виникає при спробі парсингу ключових слів delete та update у нижньому регістрі. На момент закінчення практики розробника JSQLParser було повідомлено про проблему та було знайдено рішення. На жаль це рішення буде впроваджене лише в JSQLParser 5.0. До того моменту варто запобігати використанні ключових слів delete та update у нижньому регістрі при написанні скрипту.

Повний код класу наведено в Додатку Б.

Клас LiveController

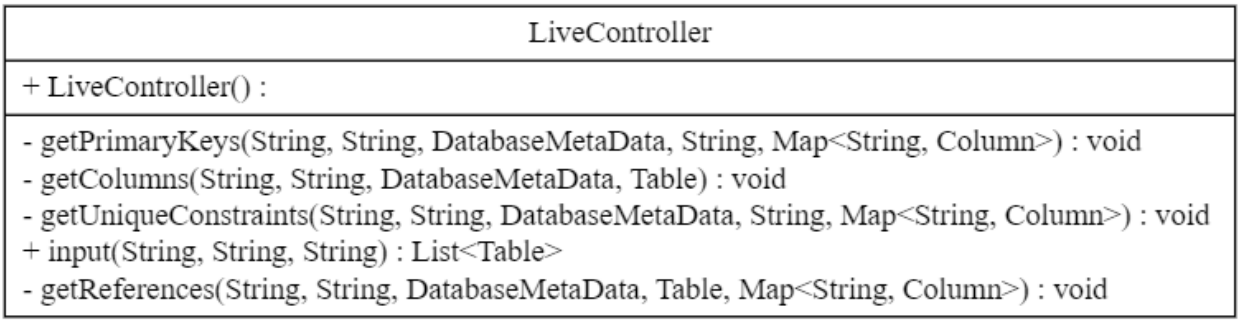

Рисунок 3.5 UML діаграма класу LiveController

Даний клас (див. рис 3.5) відповідає за отримання метаданих з розгорнутої бази даних та створення на їх основі списку об’єктів класу Table. Метадані отримуються з об’єкта класу DatabaseMetaData що отримується з об’єкта класу Connection що отримується за допомогою класу DriverManager що аналізує надане йому посилання та на його основі обирає драйвер відповідний базі даних. Самі метадані отримуються за допомогою різноманітних методів що повертають об’єкт класу ResultSet. Об’єкти класу ResultSet зазвичай використовуються для зберігання даних отриманих в результаті виконання запитів та є подібними до таблиць де значення зберігаються в рядках розділених на стовпці.

Аналогічно FileController метод input є основним методом класу. В якості вхідних даних отримує посилання на базу даних (що повинно містити дані для авторизації), назву каталогу та назву схеми до якої належить потрібна нам база даних.

Далі отримується таблиця з метаданими про всі таблиці в цій базі даних. З кожного рядка отримується назва таблиці за якою отримуються таблиця стовпців, таблиця обмежень унікальності, таблиця первинних ключів та таблиця імпортованих ключів що містить інформацію про залежні ключі та первинні ключі від яких вони залежать.

Метод getColumns заповнює список об’єктів класу Column в об’єкті класу Table. При цьому тип та розмір кожного стовпця отримуються окремо та конкатенуються. У випадку якщо розмір стовпця не був вказаний при створенні бази даних система управління базами даних встановлює його або в значення за замовчуванням або в максимальне можливе. У випадку цілочисельних значень максимально можливий розмір є доволі довгим що погіршує вигляд діаграми через що було вирішено його не вказувати. Також цей метод встановлює чи може певний стовпець містити нульові значення.

Метод getUniqueConstraints встановлює обмеження унікальності на стовпцях. Варто зазначити що в таблиці ResultSet унікальними також вважаються всі первинні ключі та всі стовпці що належать до складних обмежень унікальності. Це створює проблему адже унікальною для складних обмежень є комбінація стовпців а не кожен стовпець що до неї належить. Це в свою чергу створює некоректні позначення кардинальності адже у випадку якщо один з стовпців на які накладено обмеження унікальності є зовнішнім ключем він може набувати певного значення декілька разів за умови що інша частина обмеження змінюється. Також немає сенсу накладати обмеження унікальності на окремі частини складного первинного ключа адже це робить цю частину унікальним ідентифікатором що прибирає потребу у складеному первинному ключі. На жаль інформація про ці обмеження зберігаються у вигляді рядків таблиці без ідентифікації групування цих обмежень через що була здійснена спроба ідентифікації цих складних обмежень за їх ім’ям. У випадку якщо ім’я не вказано система управління базою даних зазвичай генерує його автоматично. Якщо воно нульове подальша логіка ігнорується. Для обліку використовується хеш-набір. Встановлення обмеження відбувається для стовпця якщо імені обмеження немає в хеш-наборі. У випадку якщо ім’я є в хеш наборі обмеження знімається з попереднього стовпця.

Метод getPrimaryKeys встановлює чи є стовпець первинним ключем для кожного стовпця таблиці.

Метод getReferences встановлює чи є стовпець зовнішнім ключем для кожного стовпця таблиці та додає запис в список зв'язків таблиці. На відміну від ResultSet що містив обмеження унікальності в даному ResultSet є стовпець KEY_SEQ в якому міститься порядковий номер даного зовнішнього ключа в обмеженні. Таким чином якщо порядковий номер попереднього та поточного зовнішнього ключа дорівнює одиниці ми можемо стверджувати що попередній зв'язок містив лише один зовнішній ключ. В іншому випадку ми додаємо зовнішні ключі до списку всередині одного зв'язку.

Повний код класу наведено в Додатку В.

Клас GraphvizView

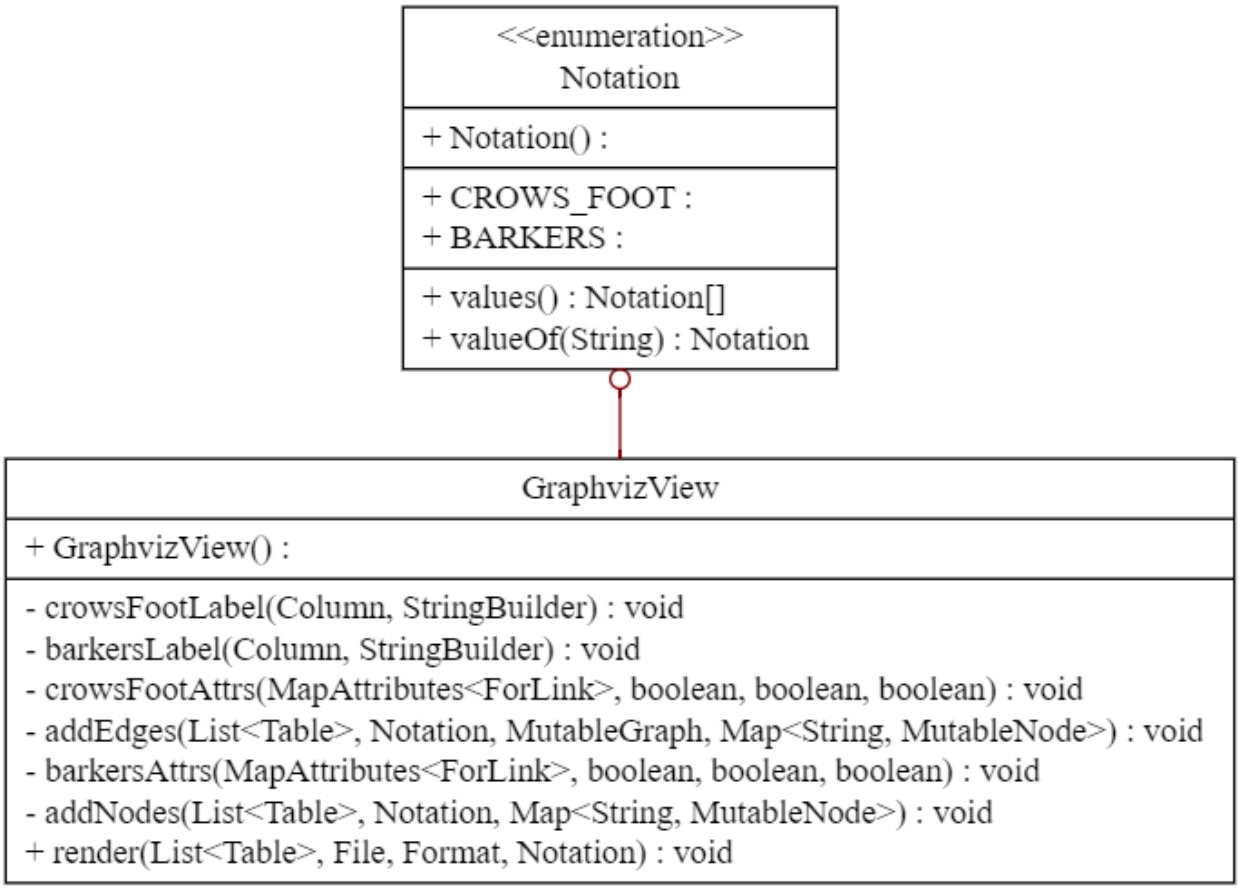

Рисунок 3.6 UML діаграма класу GraphvizView

Даний клас (див. рис 3.6) здійснює створення dot коду на основі отриманої моделі та рендеринг цього коду у вигляді діаграми. Головний метод render приймає список таблиць, файл для вихідних даних, формат вихідних даних та обрану нотацію.

Реалізовані нотації «вороняча лапка» та Баркера. Форматом вихідних даних може бути як формат зображення(PNG або SVG) так і текстовий формат(DOT, PLAIN, JSON). Повний список форматів можна побачити в допоміжному повідомленні при виконанні додатку без вхідних параметрів або з параметром -h чи --help. Також при вказанні вхідних параметрів рекомендується вказувати лише назву вихідного файлу. В цьому випадку розширення буде згенеровано автоматично.

Спочатку ініціалізується граф з глобальними параметрами графу, вузлів та ребер. Два допоміжних методи використовуються для додання вузлів та додання ребер. Кожен з цих допоміжних методів також використовує по допоміжному методу на кожну нотацію.

Далі вказується вибір Javascript двигуна. Бібліотека java-graphviz підтримує три двигуна та може працювати напряму з Graphviz при доступності команди dot в консолі системи користувача. На жаль рендеринг за допомогою команди dot працює некоректно порівняно з Javascript імплементацією а саме не відбувається рендеринг складних стрілок. Синтаксис складання позначок на кінці стрілок вказаний в офіційній документації Graphviz що теоретично має свідчити про однаковий функціонал в обох імплементаціях. До двигунів підтримуваних graphviz-java належать:

– Java імплементація двигуна V8 під назвою J2V8
– Двигун GraalJS з пакету розробки GraalVM
– Двигун Nashorn з пакетів розробки Java до 15-ої версії виключно

Останній двигун є найповільнішим та вважається застарілим починаючи з 11-ої версії JDK. Двигун GraalJS потребує компіляції та виконання за допомогою інструментів GraalVM для досягнення максимальної швидкодії та

довго прогрівається. Двигун V8 має відкритий код, був розроблений Google та використовується в Node.js та Chrome. Його Java імплементація J2V8 хоч і є застарілою через відмову розробників в підтримці будь-яких операційних систем окрім Android починаючи з 2017 року але є найшвидшим та найбільш надійним з трьох варіантів. До того ж J2V8 підтримує Windows 7-10 (64bit та 32bit), Mac OS X та Linux (64bit).

Повний код класу наведено в Додатку Г.

Клас Sql2ErdCli

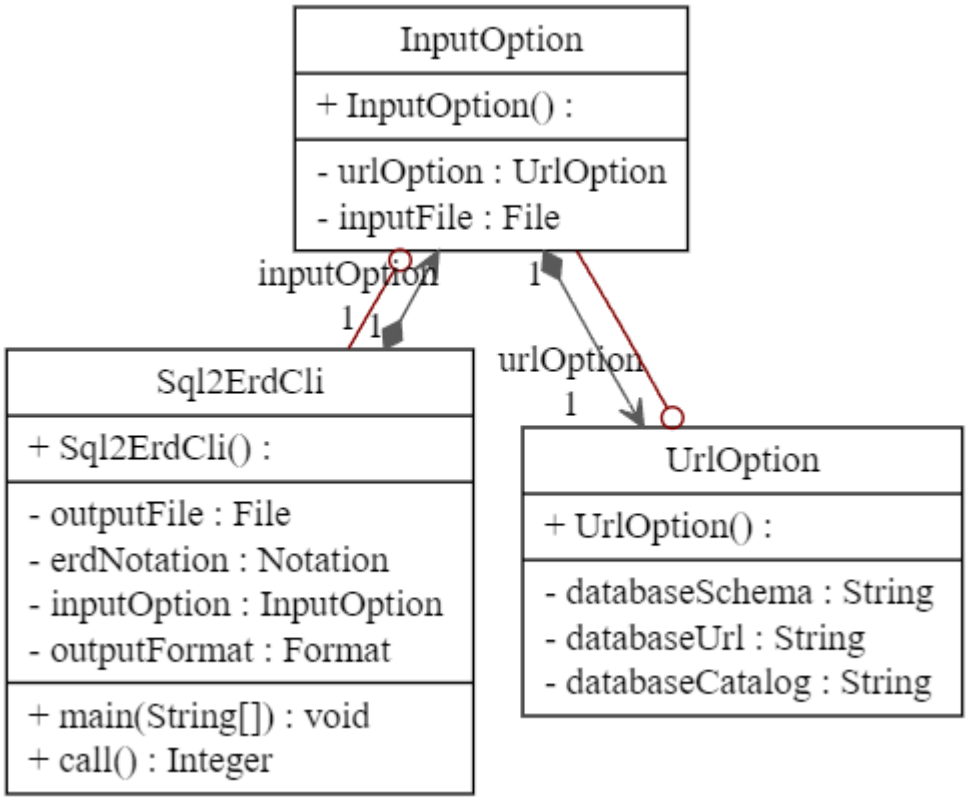

Рисунок 3.7 UML діаграма класу Sql2ErdCli

Цей клас(див. рис 3.7) є головним класом додатку адже містить метод main. Цей метод містить лише один рядок коду тоді як головна логіка міститься в методі call.

Отримання значень полів з аргументів командного рядку виконує бібліотека Picocli що налаштовується за допомогою анотацій. Для визначення

групи аргументів ця бібліотека потребує створення внутрішніх класів. Якщо можуть бути задані всі аргументи групи то параметр анотації exclusive встановлюється в стан false. У іншому випадку тільки один аргумент з групи може бути вказаний. Обов’язковість вказання групи визначається значенням параметру анотації multiplicity.

Клас InputOption є обов’язковою ексклюзивною групою та відповідає чи будуть дані зчитані з файлу чи за посиланням на базу даних.

Клас UrlOption є обов’язковою неексклюзивною групою та відповідає за можливі уточнення щодо посилання на базу даних що не мають сенсу при зчитуванні даних з файлу.

Поле outputFormat може містити лише значення з перелічуваного типу даних отриманого з бібліотеки graphviz-java. Поле erdNotation може містити лише значення з перелічуваного типу даних отриманого з класу GraphvizView. Значення що приймаються цими полями можна побачити в допоміжному повідомленні що відображається при запуску додатку без аргументів або з аргументом -h або --help. При цьому обов’язково дотримуватись зазначеного регістру.

Повний код класу наведено в Додатку Ґ.

3.3 Аналіз результатів

Після написання та перевірки коду додаток було скомпільовано у файл sql2erd.jar . Цей файл було виконано з командного рядку за допомогою команди

```
java -jar sql2erd.jar -o crow_file -f PNG -i test.sql -n CROWS_FOOT
```

В результаті було отримано діаграму в нотації «вороняча лапка» (див. рис 3.8).

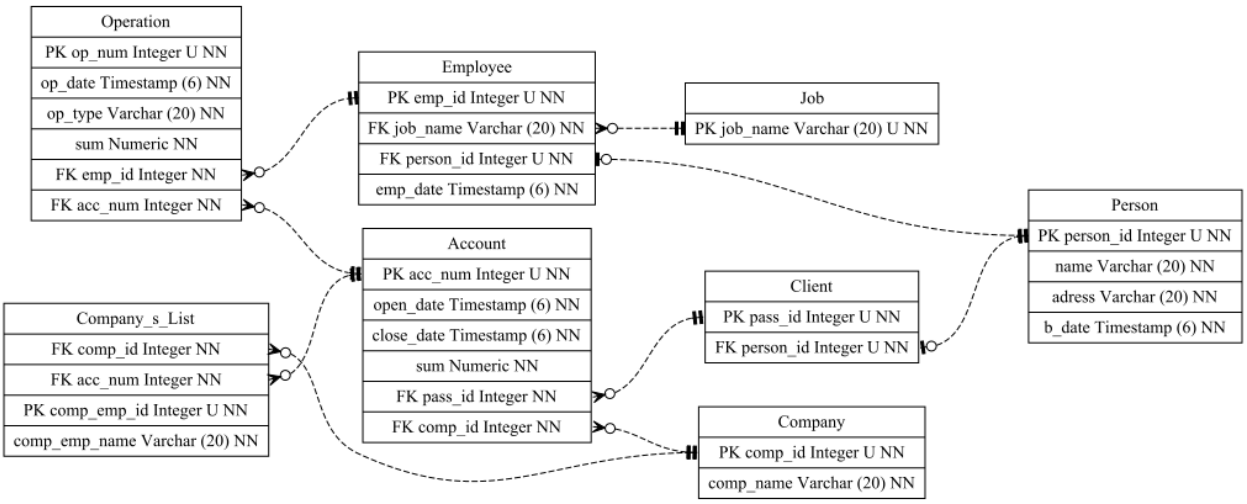

Рисунок 3.8 Діаграма на основі скрипту з файлу в нотації "Вороняча лапка"

Після цього команду було змінено та отримано діаграму в нотації Баркера (див. рис 3.9).

```
java -jar sql2erd.jar -o barker_file -f PNG -i test.sql -n BARKERS
```

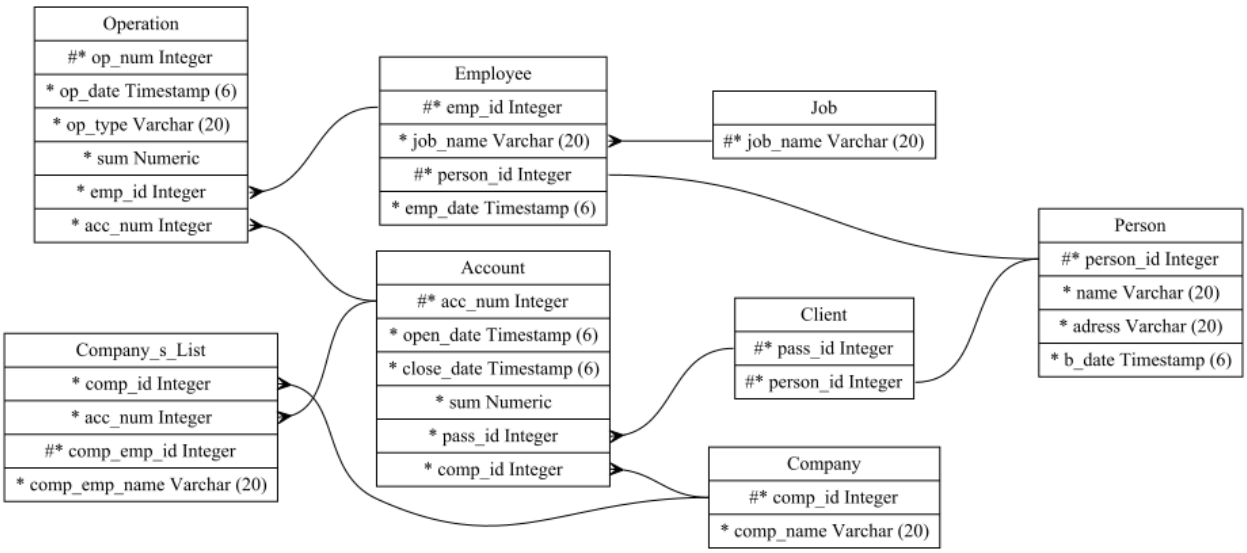

Рисунок 3.9 Діаграма на основі скрипту з файлу в нотації Баркера

Використаний скрипт було розгорнуто в системі управління базами даних PostgreSQL. На цей раз додаток було виконано за допомогою команди

```
java -jar sql2erd.jar -o crow_live -f PNG -u
jdbc:postgresql://localhost:5432/postgres?user=postgres"&"password=test -n
CROWS_FOOT
```

та отримано дещо іншу діаграму (див. рис 3.10) через автоматичну генерацію назв відношень СУБД PostgreSQL.

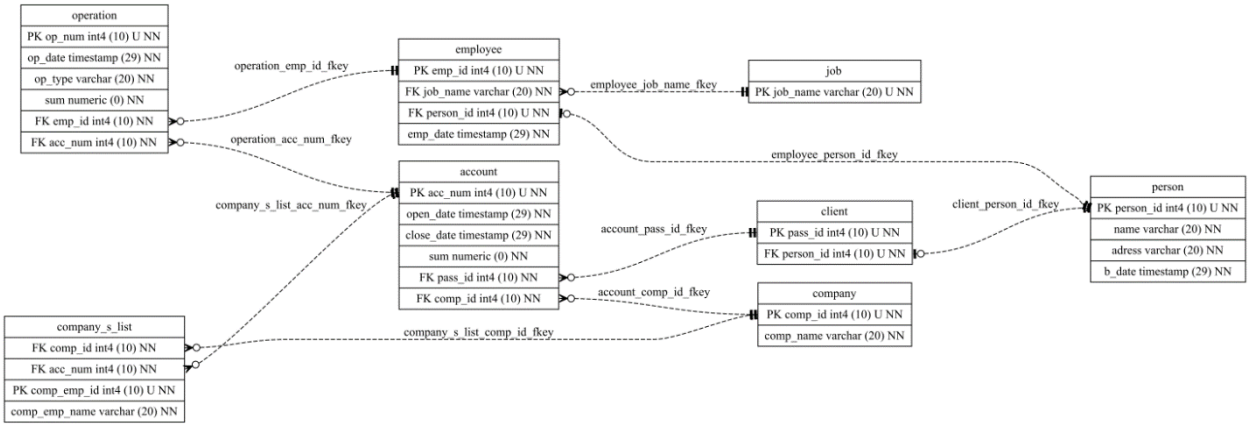

Рисунок 3.10 Діаграма на основі розгорнутої бази даних в нотації "Вороняча лапка"

Також було згенеровано діаграму в нотації Баркера (див. рис 3.11).

```
java -jar sql2erd.jar -o barker_live -f PNG -u
jdbc:postgresql://localhost:5432/postgres?user=postgres"&"password=test -n
BARKERS
```

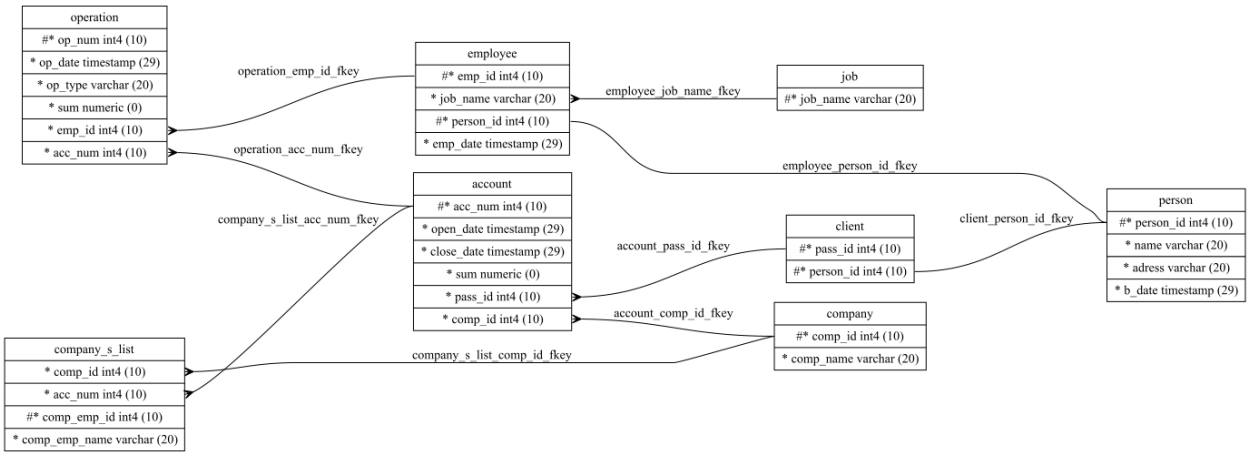

Рисунок 3.11 Діаграма на основі розгорнутої бази даних в нотації Баркера

Порівняно з альтернативними додатками створений додаток надає більшу гнучкість при виборі операційної системи, джерела інформації про базу даних та нотації кінцевого результату .

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було:

1. Проведено огляд існуючих засобів інжинірингa та ре-інжинірингa реляційних баз даних. Були аналізовані додатки CASE Studio 2, Toad Data Modeler, Erwin Data Modeler, Lucidchart, draw.io, dbdiagram.io, SchemaCrawler та SchemaSpy. Аналіз показав недостатній функціонал наведених додатків.
2. Проаналізовано нотації документування схеми бази даних. Проаналізовані були нотації «Вороняча лапка», Баркера, IDEF1X та UML. За результатами аналізу було вирішено поставити вимогу до розроблюваного додатку на підтримку нотацій Баркера та «Вороняча лапка».
3. Спроектовано додаток документування структури бази даних. Було обрано патерн Модель-Представлення-Контролер та створені класи для кожної частини патерну разом з головним класом додатку. На основі створених класів та відношень між ними було створено діаграму класів (див. рис 2.1).
4. Проведено імплементацію додатку у відповідності до сформульованих вимог. Створений додаток має можливість аналізувати як код створення бази даних написаний на діалекті Oracle, MS SQL Server, Sybase, PostgreSQL, MySQL, MariaDB, DB2, H2, HSQLDB, Derby або SQLite так і вже розгорнуту базу даних в усіх вище зазначених СУБД окрім Sybase та створювати на її основі діаграму в нотації Баркера або нотації «Вороняча лапка» що зберігається в одному форматі зі списку: PNG, SVG, DOT, XDOT, PLAIN, PLAIN_EXT, PS, PS2(PDF), JSON(для XDOT), JSON0(для DOT).
5. Проведено тестування на основі сформованих вхідних даних для файлу зі скриптом створення бази даних та розгорнутої в СУБД бази даних в обох нотаціях. За результатами тестування було підтверджено переваги над оглянутими альтернативами а саме: можливість аналізу файлу з SQL скриптом, більша кількість підтримуваних СУБД, більша кількість підтримуваних нотацій та кросплатформенність. З недоліків додаток не

дозволяє створювати діаграму вручну та генерувати скрипт створення бази даних на її основі, не підтримує мови розмітки DBML, PlantUML та Mermaid, має повільнішу швидкодію та певні недоліки діаграм такі як злиття стрілок та відсутність підтримки IDEF1X нотації через неможливість додання назви над вузлом.

Для подальшого покращення роботи додатку можливо:

1. Додання підтримки більшої кількості нотацій
2. Додання підтримки більшої кількості СУБД та діалектів
3. Додання підтримки більшої кількості мов розмітки
4. Перехід з Record-based вузлів на HTML-like вузли для покращення вигляду та розширення можливостей генерації діаграм
5. Оптимізація коду для покращення швидкодії
6. Вчасне оновлення бібліотек та участь в усуненні багів
7. Модифікація бібліотеки graphviz-java для використання бібліотеки Javet замість застарілої версії бібліотеки J2V8
8. Додання можливості генерації SQL коду на основі коду мов розмітки

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Contributors to Wikimedia projects. Toad data modeler. Wikipedia. URL: https://en.wikipedia.org/wiki/Toad_Data_Modeler (дата звернення: 03.06.2023).
2. Quest Software. Toad data modeler. Toad World. URL: https://www.toadworld.com/products/toad-data-modeler (дата звернення: 03.06.2023).
3. Quest Software. Erwin data modeler. erwin. URL: https://www.erwin.com/register/129426 (дата звернення: 03.06.2023).
4. Lucid Software Inc. Intelligent diagramming. Lucidchart. URL: https://www.lucidchart.com/pages (дата звернення: 03.06.2023).
5. JGraph Ltd. Draw.io. draw.io. URL: https://www.drawio.com (дата звернення: 03.06.2023).
6. Holistics Software. Database relationship diagrams design tool. dbdiagram.io. URL: https://dbdiagram.io/home (дата звернення: 03.06.2023).
7. Fatehi S. Database schema discovery and comprehension tool. SchemaCrawler. URL: https://www.schemacrawler.com (дата звернення: 03.06.2023).
8. Currier J. Database documentation. SchemaSpy. URL: https://schemaspy.org (дата звернення: 03.06.2023).
9. Hay D. C. Requirements analysis: from business views to architecture : навч. посіб. New Jersey : Prentice Hall, 2002. 343-389 с.
10. Ambler S. W. Data modeling 101. The Agile Data (AD) Method. URL: https://agiledata.org/essays/dataModeling101.html (дата звернення: 03.06.2023).
11. Gibello P.-Y. Java SQL parser. Zql. URL: https://zql.sourceforge.net (дата звернення: 03.06.2023).
12. Data Geekery GmbH. JOOQ object oriented querying. jOOQ. URL: https://www.jooq.org (дата звернення: 03.06.2023).

13. Gudu Software. General SQL parser. SQL Parser. URL: https://www.sqlparser.com/sql-parser-java.php (дата звернення: 03.06.2023).
14. manticore projects Co. Ltd. JSQLParser. GitHub. URL: https://github.com/JSQLParser/JSqlParser (дата звернення: 03.06.2023).
15. Oracle. DatabaseMetaData. Java SE Documentation. URL: https://docs.oracle.com/javase/8/docs/api/java/sql/DatabaseMetaData.html (дата звернення: 03.06.2023).
16. JGraph Ltd. JGraphX. GitHub. URL: https://github.com/jgraph/jgraphx (дата звернення: 03.06.2023).
17. The Graphviz Authors. Graphviz. Graphviz. URL: https://graphviz.org (дата звернення: 03.06.2023).
18. Laci J. Graphviz-java-api. GitHub. URL: https://github.com/jabbalaci/graphviz-java-api (дата звернення: 03.06.2023).
19. Niederhauser S. Graphviz-java. GitHub. URL: https://github.com/nidi3/graphviz-java (дата звернення: 03.06.2023).
20. Roques A. Entity Relationship diagram syntax and features. PlantUML. URL: https://plantuml.com/ie-diagram (дата звернення: 03.06.2023).
21. Popma R. A mighty tiny command line interface. picocli. URL: https://picocli.info (дата звернення: 03.06.2023).

ДОДАТОК А КОД СТВОРЕННЯ ТЕСТОВОЇ БАЗИ ДАНИХ

```
Create table Job (
job_name Varchar (20) NOT NULL ,
primary key (job_name)
);

Create table Person (
person_id Integer NOT NULL ,
name Varchar (20) NOT NULL ,
adress Varchar (20) NOT NULL ,
b_date Timestamp(6) NOT NULL ,
primary key (person_id)
);

Create table Client (
pass_id Integer NOT NULL ,
person_id Integer NOT NULL UNIQUE,
primary key (pass_id) ,
foreign key (person_id) references Person (person_id)
);

Create table Company (
comp_id Integer NOT NULL ,
comp_name Varchar (20) NOT NULL ,
primary key (comp_id)
);

Create table Account (
acc_num Integer NOT NULL ,
open_date Timestamp(6) NOT NULL ,
close_date Timestamp(6) NOT NULL ,
sum Numeric NOT NULL ,
pass_id Integer NOT NULL ,
comp_id Integer NOT NULL ,
primary key (acc_num) ,
foreign key (pass_id) references Client (pass_id) ,
foreign key (comp_id) references Company (comp_id)
);

Create table Employee (
emp_id Integer NOT NULL ,
job_name Varchar (20) NOT NULL ,
person_id Integer NOT NULL UNIQUE,
emp_date Timestamp(6) NOT NULL ,
primary key (emp_id) ,
foreign key (job_name) references Job (job_name) ,
foreign key (person_id) references Person (person_id)
);
```

```
Create table Operation (
op_num Integer NOT NULL ,
op_date Timestamp(6) NOT NULL ,
op_type Varchar (20) NOT NULL ,
sum Numeric NOT NULL ,
emp_id Integer NOT NULL ,
acc_num Integer NOT NULL ,
primary key (op_num) ,
foreign key (emp_id) references Employee (emp_id) ,
foreign key (acc_num) references Account (acc_num)
);

Create table Company_s_List (
comp_id Integer NOT NULL ,
acc_num Integer NOT NULL ,
comp_emp_id Integer NOT NULL ,
comp_emp_name Varchar (20) NOT NULL ,
primary key (comp_emp_id) ,
foreign key (acc_num) references Account (acc_num) ,
foreign key (comp_id) references Company (comp_id)
);
```

ДОДАТОК Б КОД КЛАСУ FILECONTROLLER

```
package edu.sumdu.dbis.sql2erd.controller;

import edu.sumdu.dbis.sql2erd.model.Column;
import edu.sumdu.dbis.sql2erd.model.Reference;
import edu.sumdu.dbis.sql2erd.model.Table;
import net.sf.jsqlparser.JSQLParserException;
import net.sf.jsqlparser.parser.CCJSqlParserUtil;
import net.sf.jsqlparser.statement.Statement;
import net.sf.jsqlparser.statement.create.table.ColumnDefinition;
import net.sf.jsqlparser.statement.create.table.CreateTable;
import net.sf.jsqlparser.statement.create.table.ForeignKeyIndex;
import net.sf.jsqlparser.statement.create.table.Index;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

public class FileController {
    public static List<Table> input(File input) throws IOException, JSQLParserException {
        List<Table> tables = new ArrayList<>();
        List<Statement> statements =
CCJSqlParserUtil.parseStatements(Files.readString(input.toPath())).getStatements();
        for (Statement statement : statements) {
            if (statement instanceof CreateTable) {
                CreateTable createTable = (CreateTable) statement;
                Table table = new Table(createTable.getTable().getName());
                getColumns(createTable, table);
                getIndices(createTable, table);
                tables.add(table);
            }
        }
        return tables;
    }

    private static void getColumns(CreateTable createTable, Table table) {
        List<ColumnDefinition> columnDefinitions = createTable.getColumnDefinitions();
        for (ColumnDefinition columnDefinition : columnDefinitions) {
            Column column = new Column(columnDefinition.getColumnName(),
columnDefinition.getColDataType().toString());
            List<String> columnSpecs = columnDefinition.getColumnSpecs();
            if (columnSpecs != null) {
                for (String columnSpec : columnSpecs) {
                    if (columnSpec.equalsIgnoreCase("PRIMARY") &&
                            columnSpecs.get(columnSpecs.indexOf(columnSpec) +
1).equalsIgnoreCase("KEY")) {
                        column.setPrimaryKey(true);
                        column.setNotNull(true);
                        column.setUnique(true);
                    } else if (columnSpec.equalsIgnoreCase("NOT") &&
                            columnSpecs.get(columnSpecs.indexOf(columnSpec) +
1).equalsIgnoreCase("NULL")) {
                        column.setNotNull(true);
                    } else if (columnSpec.equalsIgnoreCase("UNIQUE")) {
                        column.setUnique(true);
                    } else if (columnSpec.equalsIgnoreCase("REFERENCES")) {
                        column.setForeignKey(true);
                        String referencedColumnName =
columnSpecs.get(columnSpecs.indexOf(columnSpec) + 2);
                        Reference reference = new
Reference(Collections.singletonList(column.getName()),
                                columnSpecs.get(columnSpecs.indexOf(columnSpec) + 1),
                                Collections.singletonList(referencedColumnName.substring(1,
referencedColumnName.length() - 1)));
                        if (columnSpecs.indexOf(columnSpec) - 2 >= 0) {
                            if (columnSpecs.get(columnSpecs.indexOf(columnSpec) -
2).equalsIgnoreCase("CONSTRAINT")) {
                                reference.setName(columnSpecs.get(columnSpecs.indexOf(columnSpec)
- 1));
```

```
                            }
                        }
                        table.addReference(reference);
                    }
                }
            }
            table.addColumn(column);
        }
    }

    private static void getIndices(CreateTable createTable, Table table) {
        List<Index> indices = createTable.getIndexes();
        if (indices != null) {
            for (Index index : indices) {
                String indexName = index.getName();
                Map<String, Column> columnMap =
table.getColumns().stream().collect(Collectors.toMap(Column::getName, Function.identity()));
                if (index instanceof ForeignKeyIndex) {
                    ForeignKeyIndex fkIndex = (ForeignKeyIndex) index;
                    for (String columnName : fkIndex.getColumnsNames()) {
                        columnMap.get(columnName).setForeignKey(true);
                    }
                    Reference reference = new Reference(fkIndex.getColumnsNames(),
fkIndex.getTable().getName(), fkIndex.getReferencedColumnNames());
                    reference.setName(indexName);
                    table.addReference(reference);
                } else {
                    List<String> columnNames = index.getColumnsNames();
                    for (String columnName : columnNames) {
                        Column column = columnMap.get(columnName);
                        if (columnNames.size() == 1) {
                            if (index.getType().equalsIgnoreCase("UNIQUE")) {
                                column.setUnique(true);
                            }
                        }
                        if (index.getType().equalsIgnoreCase("PRIMARY KEY")) {
                            column.setPrimaryKey(true);
                            column.setNotNull(true);
                            if (columnNames.size() == 1) {
                                column.setUnique(true);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

ДОДАТОК В КОД КЛАСУ LIVECONTROLLER

```
package edu.sumdu.dbis.sql2erd.controller;

import edu.sumdu.dbis.sql2erd.model.Column;
import edu.sumdu.dbis.sql2erd.model.Reference;
import edu.sumdu.dbis.sql2erd.model.Table;

import java.sql.*;
import java.util.*;
import java.util.function.Function;
import java.util.stream.Collectors;

public class LiveController {
    public static List<Table> input(String url, String catalogName, String schemaName) throws
SQLException {
        List<Table> tableList = new ArrayList<>();
        Connection connection = DriverManager.getConnection(url);
        DatabaseMetaData metaData = connection.getMetaData();
        ResultSet tables = metaData.getTables(catalogName, schemaName, null, new
String[]{"TABLE"});
        while (tables.next()) {
            String tableName = tables.getString("TABLE_NAME");
            Table table = new Table(tableName);
            getColumns(catalogName, schemaName, metaData, table);
            Map<String, Column> columnMap =
table.getColumns().stream().collect(Collectors.toMap(Column::getName, Function.identity()));
            getUniqueConstraints(catalogName, schemaName, metaData, tableName, columnMap);
            getPrimaryKeys(catalogName, schemaName, metaData, tableName, columnMap);
            getReferences(catalogName, schemaName, metaData, table, columnMap);
            tableList.add(table);
        }
        return tableList;
    }

    private static void getColumns(String catalogName, String schemaName, DatabaseMetaData
metaData, Table table) throws SQLException {
        ResultSet columns = metaData.getColumns(catalogName, schemaName, table.getName(), null);
        while (columns.next()) {
            Column column;
            int colSize = columns.getInt("COLUMN_SIZE");
            if (colSize == Integer.MAX_VALUE)
                column = new Column(columns.getString("COLUMN_NAME"),
columns.getString("TYPE_NAME"));
            else column = new Column(columns.getString("COLUMN_NAME"),
                    columns.getString("TYPE_NAME") + " (" + colSize + ")");
            if (columns.getString("IS_NULLABLE").equals("NO")) {
                column.setNotNull(true);
            }
            table.addColumn(column);
        }
    }

    private static void getUniqueConstraints(String catalogName, String schemaName,
DatabaseMetaData metaData, String tableName, Map<String, Column> columnMap) throws SQLException {
        ResultSet indices = metaData.getIndexInfo(catalogName, schemaName, tableName, true,
false);
        String prevColumnName = null;
        String currIndexName;
        String currColumnName;
        Set<String> indexHashSet = new HashSet<>();
        while (indices.next()) {
            currIndexName = indices.getString("INDEX_NAME");
            currColumnName = indices.getString("COLUMN_NAME");
            if (currIndexName != null) {
                if (!indexHashSet.contains(currIndexName)) {
                    indexHashSet.add(currIndexName);
                    columnMap.get(currColumnName).setUnique(true);
                    prevColumnName = currColumnName;
                } else columnMap.get(prevColumnName).setUnique(false);
            } else columnMap.get(currColumnName).setUnique(true);
        }
    }

    private static void getPrimaryKeys(String catalogName, String schemaName, DatabaseMetaData
metaData, String tableName, Map<String, Column> columnMap) throws SQLException {
```

```
        ResultSet primaryKeys = metaData.getPrimaryKeys(catalogName, schemaName, tableName);
        while (primaryKeys.next()) {
            columnMap.get(primaryKeys.getString("COLUMN_NAME")).setPrimaryKey(true);
        }
    }

    private static void getReferences(String catalogName, String schemaName, DatabaseMetaData
metaData, Table table, Map<String, Column> columnMap) throws SQLException {
        ResultSet foreignKeys = metaData.getImportedKeys(catalogName, schemaName,
table.getName());
        Reference reference = null;
        String currReferencedTableName;
        String prevReferencedTableName = null;
        while (foreignKeys.next()) {
            currReferencedTableName = foreignKeys.getString("PKTABLE_NAME");
            if (currReferencedTableName.equals(prevReferencedTableName)) {
                if (foreignKeys.getShort("KEY_SEQ") == 1) {
                    table.addReference(reference);
                    reference = new Reference();
                    reference.setName(foreignKeys.getString("FK_NAME"));
                    reference.setReferencedTableName(currReferencedTableName);
                }
            } else {
                if (reference != null) {
                    table.addReference(reference);
                }
                reference = new Reference();
                reference.setName(foreignKeys.getString("FK_NAME"));
                reference.setReferencedTableName(currReferencedTableName);
            }
            columnMap.get(foreignKeys.getString("FKCOLUMN_NAME")).setForeignKey(true);
            reference.addColumnName(foreignKeys.getString("FKCOLUMN_NAME"));
            reference.addReferencedColumnName(foreignKeys.getString("PKCOLUMN_NAME"));
            prevReferencedTableName = currReferencedTableName;
        }
        if (reference != null) {
            table.addReference(reference);
        }
    }
}
```

ДОДАТОК Г КОД КЛАСУ GRAPHVIZVIEW

```
package edu.sumdu.dbis.sql2erd.view;

import edu.sumdu.dbis.sql2erd.model.Column;
import edu.sumdu.dbis.sql2erd.model.Reference;
import edu.sumdu.dbis.sql2erd.model.Table;
import guru.nidi.graphviz.attribute.ForLink;
import guru.nidi.graphviz.attribute.MapAttributes;
import guru.nidi.graphviz.attribute.Rank;
import guru.nidi.graphviz.attribute.Records;
import guru.nidi.graphviz.engine.Format;
import guru.nidi.graphviz.engine.Graphviz;
import guru.nidi.graphviz.engine.GraphvizV8Engine;
import guru.nidi.graphviz.model.Link;
import guru.nidi.graphviz.model.MutableGraph;
import guru.nidi.graphviz.model.MutableNode;

import java.io.File;
import java.io.IOException;
import java.util.*;
import java.util.function.Function;
import java.util.stream.Collectors;

import static guru.nidi.graphviz.attribute.Records.rec;
import static guru.nidi.graphviz.model.Factory.*;

public class GraphvizView {
    public static void render(List<Table> tables, File outputFile, Format outputFormat, Notation
erdNotation) throws IOException {
        MutableGraph mg = mutGraph().setDirected(true)
                .graphAttrs().add(Rank.dir(Rank.RankDir.LEFT_TO_RIGHT))
                .nodeAttrs().add("shape", "record")
                .linkAttrs().add("dir", "both")
                .linkAttrs().add("minlen", 2);
        Map<String, MutableNode> mutableNodeMap = new HashMap<>();
        addNodes(tables, erdNotation, mutableNodeMap);
        addEdges(tables, erdNotation, mg, mutableNodeMap);
        Graphviz.useEngine(new GraphvizV8Engine());
        Graphviz.fromGraph(mg).render(outputFormat).toFile(outputFile);
    }

    private static void addNodes(List<Table> tables, Notation erdNotation, Map<String,
MutableNode> mutableNodeMap) {
        for (Table table : tables) {
            MutableNode mn = mutNode(table.getName());
            mutableNodeMap.put(table.getName(), mn);
            List<String> recs = new ArrayList<>();
            recs.add(table.getName());
            for (Column column : table.getColumns()) {
                StringBuilder sb = new StringBuilder();
                if (Objects.equals(erdNotation, Notation.BARKERS)) barkersLabel(column, sb);
                else crowsFootLabel(column, sb);
                recs.add(rec(column.getName(), sb.toString()));
            }
            mn.add(Records.of(recs.toArray(new String[0])));
        }
    }

    private static void addEdges(List<Table> tables, Notation erdNotation, MutableGraph mg,
Map<String, MutableNode> mutableNodeMap) {
        for (Table table : tables) {
            String tableName = table.getName();
            for (Reference reference : table.getReferences()) {
                String referencedTableName = reference.getReferencedTableName();
                MutableNode referencedNode = mutableNodeMap.get(referencedTableName);
                String fkColumnName = reference.getColumnsNames().get(0);
                String pkColumnName = reference.getReferencedColumnNames().get(0);
                Link link = between(port(fkColumnName), referencedNode.port(pkColumnName));
                MapAttributes<ForLink> linkAttributes = new MapAttributes<>();
                if (reference.getName() != null) linkAttributes.add("label",
reference.getName());
                boolean cardinality = reference.getColumnsNames().stream()
                        .anyMatch(cn ->
table.getColumns().stream().collect(Collectors.toMap(Column::getName,
Function.identity())).get(cn).isUnique());
```

```
                boolean modality = reference.getColumnsNames().stream()
                        .allMatch(cn ->
table.getColumns().stream().collect(Collectors.toMap(Column::getName,
Function.identity())).get(cn).isNotNull());
                boolean identifying = reference.getColumnsNames().stream()
                        .anyMatch(cn ->
table.getColumns().stream().collect(Collectors.toMap(Column::getName,
Function.identity())).get(cn).isPrimaryKey());
                if (Objects.equals(erdNotation, Notation.BARKERS))
                    barkersAttrs(linkAttributes, cardinality, modality, identifying);
                else crowsFootAttrs(linkAttributes, cardinality, modality, identifying);
                link.add(linkAttributes);
                mutableNodeMap.get(tableName).addLink(link);
            }
            mutableNodeMap.get(tableName).addTo(mg);
        }
    }


    private static void crowsFootLabel(Column column, StringBuilder sb) {
        if (column.isPrimaryKey() && column.isForeignKey()) sb.append("PFK");
        else if (column.isPrimaryKey()) sb.append("PK");
        else if (column.isForeignKey()) sb.append("FK");
        sb.append(" ").append(column.getName());
        sb.append(" ").append(column.getType());
        if (column.isUnique()) sb.append(" ").append("U");
        if (column.isNotNull()) sb.append(" ").append("NN");
    }

    private static void crowsFootAttrs(MapAttributes<ForLink> linkAttributes, boolean
cardinality, boolean modality, boolean fkPrimary) {
        linkAttributes.add("arrowtail", cardinality ? "teeodot" : "crowodot");
        linkAttributes.add("arrowhead", modality ? "teetee" : "teeodot");
        linkAttributes.add("style", fkPrimary ? "solid" : "dashed");
    }


    private static void barkersLabel(Column column, StringBuilder sb) {
        if (column.isUnique() || column.isPrimaryKey()) sb.append("#");
        if (column.isNotNull() || column.isPrimaryKey()) sb.append("*");
        else sb.append("O");
        sb.append(" ").append(column.getName());
        sb.append(" ").append(column.getType());
    }

    private static void barkersAttrs(MapAttributes<ForLink> linkAttributes, boolean cardinality,
boolean modality, boolean fkPrimary) {
        linkAttributes.add("arrowhead", "none");
        String arrowtail = cardinality ? "none" : "crow";
        if (fkPrimary) arrowtail += "tee";
        linkAttributes.add("arrowtail", arrowtail);
        linkAttributes.add("style", modality ? "solid" : "dashed");
    }

    public enum Notation {
        CROWS_FOOT,
        BARKERS
    }
}
```

ДОДАТОК Ґ КОД КЛАСУ SQL2ERDCLI

```
package edu.sumdu.dbis.sql2erd;

import edu.sumdu.dbis.sql2erd.controller.FileController;
import edu.sumdu.dbis.sql2erd.controller.LiveController;
import edu.sumdu.dbis.sql2erd.model.Table;
import edu.sumdu.dbis.sql2erd.view.GraphvizView;
import guru.nidi.graphviz.engine.Format;
import picocli.CommandLine;

import java.io.File;
import java.util.List;
import java.util.concurrent.Callable;

@CommandLine.Command(name = "sql2erd.jar", mixinStandardHelpOptions = true, version = "1.0")
public class Sql2ErdCli implements Callable<Integer> {
    @CommandLine.ArgGroup(multiplicity = "1")
    private InputOption inputOption;
    @CommandLine.Option(names = {"-o", "--output"}, required = true, description = "Output file
name")
    private File outputFile;

    @CommandLine.Option(names = {"-f", "--format"}, required = true, description = "Output
format: ${COMPLETION-CANDIDATES}")
    private Format outputFormat;

    @CommandLine.Option(names = {"-n", "--notation"}, defaultValue = "CROWS_FOOT", description =
"ERD notation: ${COMPLETION-CANDIDATES} (default: ${DEFAULT-VALUE})")
    private GraphvizView.Notation erdNotation;

    public static void main(String[] args) {
        System.exit(new CommandLine(new Sql2ErdCli()).setUsageHelpAutoWidth(true).execute(args));
    }

    @Override
    public Integer call() throws Exception {
        List<Table> tables;
        if (inputOption.inputFile != null) {
            tables = FileController.input(inputOption.inputFile);
        } else {
            tables = LiveController.input(inputOption.urlOption.databaseUrl,
                    inputOption.urlOption.databaseCatalog,
                    inputOption.urlOption.databaseSchema);
        }
        GraphvizView.render(tables, outputFile, outputFormat, erdNotation);
        return 0;
    }

    static class InputOption {
        @CommandLine.ArgGroup(exclusive = false, multiplicity = "1")
        private UrlOption urlOption;
        @CommandLine.Option(names = {"-i", "--input"}, paramLabel = "<inputFile.sql>",
description = "Input SQL file")
        private File inputFile;

        static class UrlOption {
            @CommandLine.Option(names = {"-u", "--url"}, required = true, paramLabel = "<url>",
description = "Database URL")
            private String databaseUrl;
            @CommandLine.Option(names = {"-c", "--catalog"}, paramLabel = "<catalogName>",
description = "Catalog(database) name")
            private String databaseCatalog;
            @CommandLine.Option(names = {"-s", "--schema"}, paramLabel = "<schemaName>",
description = "Schema(namespace) name")
            private String databaseSchema;
        }
    }
}
```