Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

**КУРСОВА РОБОТА**

з дисципліни «Курсова робота з розподілених баз даних»

на тему:

«Налаштування розподіленої баз даних та віртуальних таблиць»

Студентки групи ТВ-01 мп

Юрченко Богдани Олегівни

Київ - 2021

# ЗМІСТ

[ЗМІСТ 2](#_Toc1)

[АНОТАЦІЯ 3](#_Toc2)

[ВСТУП 4](#_Toc3)

[1. ТЕОРЕТИЧНІ ВІДОМОСТІ 5](#_Toc4)

[1.1. Використання розподілених баз даних в реальних системах 5](#_Toc5)

[2. Розробка схеми бази даних 8](#_Toc6)

[1.1. Опис предметної області 8](#_Toc7)

[2.2. Use-case таблиці 8](#_Toc8)

[2.3. Use-case діаграми 10](#_Toc9)

[3. Налаштування реплікації 15](#_Toc10)

[3.1. Теоретичні дані про реплікацію та вибір RDBMS 15](#_Toc11)

[3.2. Кроки, виконані, для налаштування репліцікаї 16](#_Toc12)

[4. Розробка додатку для роботи із базою данних 20](#_Toc13)

[4.1. Використані технології 20](#_Toc14)

[4.2. Опис розробленого додатку 21](#_Toc15)

[4.3. Скріншоти готової програми: 22](#_Toc16)

[4.4. Елементи вихідного коду 24](#_Toc17)

[5. Налаштування прав користувачів 27](#_Toc18)

[6. Налаштування віртувальної таблиці 30](#_Toc19)

[6.1. Кроки налаштування віртуальної таблиці 31](#_Toc20)

[ВИСНОВКИ 37](#_Toc21)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 38](#_Toc22)

[ДОДАТОК 1 ВИКОРИСТАНІ СКРИПТИ 39](#_Toc23)

# АНОТАЦІЯ

В роботі описаний процес налаштування розподіленої бази даних. А саме розробка схеми бази даних, налаштування реплікації. Розробки додатку для роботи із базою даних.

А також налаштування віртуальних таблиць в базі даних.

В записці приведені використані команди для налаштування.

А також обгрунтовується вибір RDBMS.

# ВСТУП

Наразі зростає кількість технологічних та економічних розробок, які впливають на спосіб використання систем баз даних. Зберігання даних, видобуток даних, географічні та інші інформаційні системи, напр. в Інтернеті додали нові вимоги щодо моделювання та обробки даних. Як наслідок, багато нових складних типів даних, таких як просторові, часові, аудіо- та відеодані, були введені в програмне забезпечення системи управління базами даних. Цей процес був підтриманий технічним прогресом, який був зумовлений високими вимогами комерційних програм баз даних.

Розподілені бази даних в основному надають нам переваги розподілених обчислень у домені управління базами даних. В основному, ми можемо визначити розподілену базу даних як сукупність безлічі взаємопов’язаних баз даних, розподілених по комп’ютерній мережі, та розподілену систему управління базами даних як програмну систему, яка в основному управляє розподіленою базою даних, роблячи розподіл прозорим для користувача.

Таким чином розподілені бази даних є важливим архітектурним рішенням, яке може покращити систему.

.

# 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

## 1.1. Використання розподілених баз даних в реальних системах

Багато додатків розподілено за природою. Контейнерна транспортна компанія може мати декілька відділень або філій в одній і тій же або різних країнах. Як компанії з цим справляються? Більшість компаній будь-якого розміру мають цю проблему (або перевагу) і вирішують її шляхом розподіленої обробки даних. Уявіть, чи можете ви отримати програму, яка обслуговує цю транспортну компанію Acme Container. Користувачі локальних додатків можуть виконувати свою локальну роботу над своїми локальними структурами даних на локальних сайтах. Глобальні або батьківські користувачі програми можуть мати доступ до локальних даних для зведених або контрольних функцій, таких як пошук вантажів.

Однією з характеристик локальної бази даних у цій компанії є те, що більшість користувачів та джерел даних та процесів розташовані на локальному сайті. Це дозволяє автономну або напівавтономну діяльність. Тобто це дозволяє місцевому сайту контролювати власні дані, не передаючи свої дані про доставку в будь-яке місце. Ця автономія дозволяє розподіл функцій і місцевий контроль над локальними даними. Коли материнській компанії потрібні локальні дані для вирішення проблем або відстеження, вона може легко дістатися до них за допомогою взаємозв’язків мережі.

Розподілена концепція забезпечує більшу надійність та доступність. Надійність визначається як ймовірність того, що програма буде запущена та підключена в будь-який момент часу. Доступність, з іншого боку, можна визначити як ймовірність того, що система буде постійно доступною протягом певного періоду часу. Надійність та доступність на місцевому рівні забезпечують незалежність процесів, якщо центральне місце знижується або втрачає зв'язок, що було основною вадою централізованої структури. Коли потрібні подальша надійність, доступність та незалежність, це може бути забезпечено тиражуванням даних на інших сайтах, щоб процес міг бути зроблений на іншому сайті, якщо основний сайт не працює.

Однією з головних перешкод, яку потрібно було подолати, була необхідність у можливості обміну певними даними. У більшості випадків це клас даних, який ми часто називаємо посилальними даними або даними обмеження домену. Це дані, які є загальними для перевірки та перекладу в розподіленій базі даних. Отримуючи доступ до загальних довідкових даних та обмінюючись ними, це додає ділової цілісності загальному розподіленому додатку. Коли ці загальні довідкові дані обмінюються у всій програмі, немає необхідності узгоджувати між веб-сайтами або централізованим сайтом. Звичайно, існують причини для обміну нереференційними даними лише для читання, а також для того, щоб розповсюджена дія оновлення та вставки. Це просто потрібно спланувати та контролювати за допомогою коду програми або програмного забезпечення постачальника.

Розподілені СУБД деяких постачальників дозволяють обмінюватися даними від сайту до сайту, дозволяючи при цьому автономний контроль над локальним сайтом. Це дозволяє здійснювати локальний контроль та обробку, але також дозволяє обмежену форму контролю централізованим або батьківським сайтом. Правильно встановлюючи та визначаючи параметри в середовищі, можна керувати обсягом обміну даними та віддаленим доступом до спільних даних, щоб запобігти незбалансованому або погано розподіленому навантаженню доступу на будь-якій машині локального сайту.

Обмежений контроль над обміном даними також можна встановити, щоб дозволити розповсюдження змін програмного забезпечення DDBMS на віддалених сайтах у центральному сценарії push to local. Програмне забезпечення можна було виштовхнути з центрального місця, витягнути та застосувати на місцевому рівні. Це дозволяє плавніше розподіляти програмне забезпечення та узгоджувати протокол, щоб зробити все безперебійно.

Однією із переваг, яка сприяла зростанню розподілених баз даних, є покращена продуктивність. Коли розповсюджується велика база даних, вона стає (по суті) купою менших. Кожна менша база даних, зберігаючи загальні характеристики та структуру інших компонентів, працює у своєму власному середовищі та на власному обладнанні, програмному забезпеченні та навантаженні транзакцій. Це призведе до того, що локальні бази даних матимуть набагато кращу продуктивність для запитів та доступів, що виконуються в локальній базі даних, ніж якщо б вони були у великій базі даних. Крім того, транзакції, що передбачають доступ до більш ніж одного сайту, можуть скористатися перевагами та паралельно запускати потоки процесів, тим самим скорочуючи час проходження транзакції. Потім окремі результати поєднуються, щоб дати остаточну відповідь.

Використання розподіленої бази даних також пропонує можливість локальної настройки. Коли розподілена база даних розроблена належним чином, це забезпечує незалежність продуктивності. Наприклад, якщо з якихось причин один конкретний розподілений сайт мав апаратне, програмне забезпечення або навіть обсяг даних, його можна проаналізувати, виправити, протестувати та реалізувати, не впливаючи на ефективність програми будь-якого іншого сайту та користувачів.

# 2. Розробка схеми бази даних

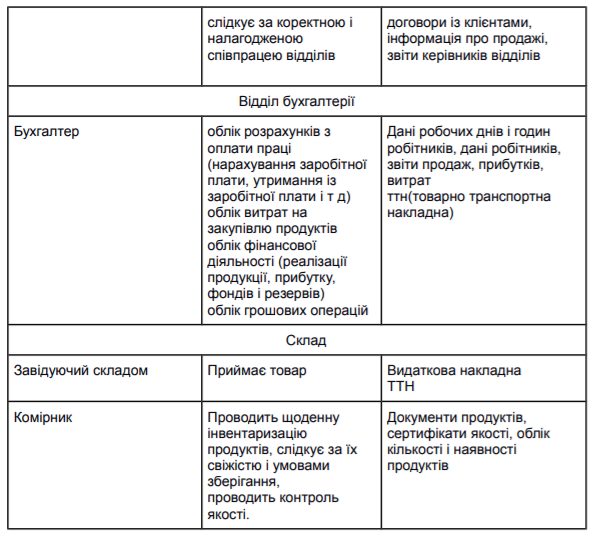
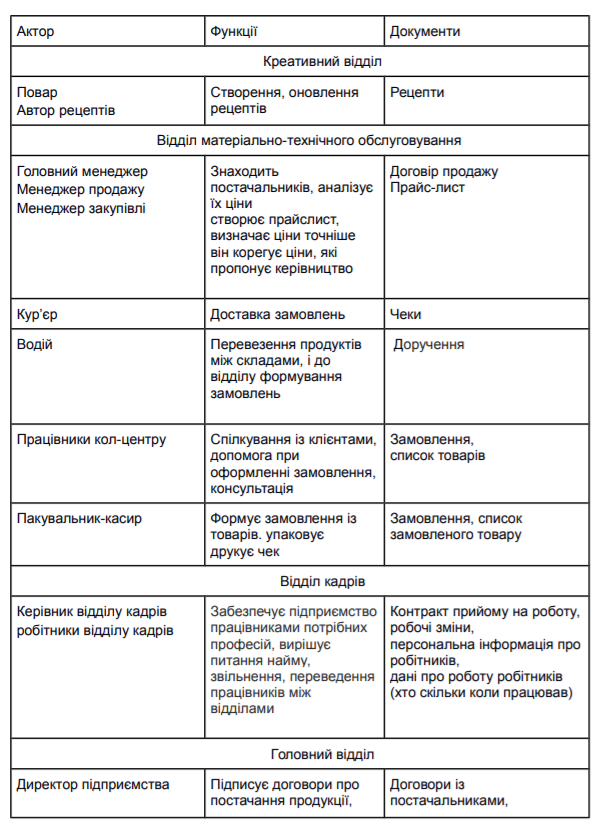
## 1.1. Опис предметної області

Сервіс доставки продуктів харчування для самостійного приготування Надає клієнтам наступні послуги: клієнт замовляє собі сніданок, обід або вечерю, і кур'єр привозить усі необхідні інгредієнти, а найголовніше - у достатній і необхідній кількості для вказаних у замовленні блюд для вказаної кількості осіб, а також детальний рецепт для приготування на власній кухні клієнта. Дане підприємство має територіально два відділи - у Києві (головний) і у Полтаві. Відповідно два склади продукції. Продукцією виступають товари харчування. У Києві знаходиться основний склад, куди товари приходять від постачальників, у другому складі зберігається не повна множина товарів, відповідно і частка від повного меню. Товари на склад відвозяться з основного складу. Підприємство виступає однією юридичною особою, тож має одного директора, що міститься у штабі головного офісу в Києві. Кожна частина підприємства має відділи: матеріально-технічного обслуговування, бухгалтерії, креативний, кадровий, головний і склад.

## 2.2. Use-case таблиці

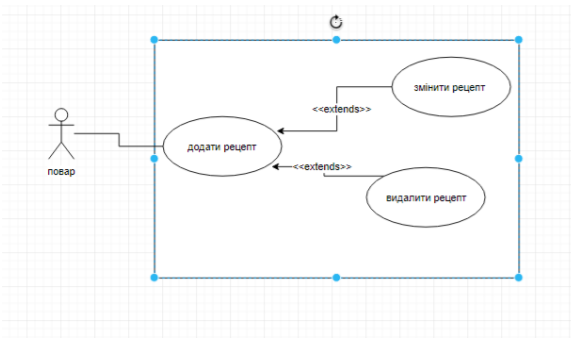
Існує багато різних діаграм UML, які слугують різним цілям (як ви можете бачити з дерева діаграм UML вище). Ви можете описати ці деталі в інших типах діаграм UML та документах, а також зв’язати їх із випадками використання.

Варіанти використання представляють лише функціональні вимоги системи. Інші вимоги, такі як ділові правила, вимоги до якості обслуговування та обмеження впровадження, повинні бути представлені окремо, знову ж таки, з іншими діаграмами UML.

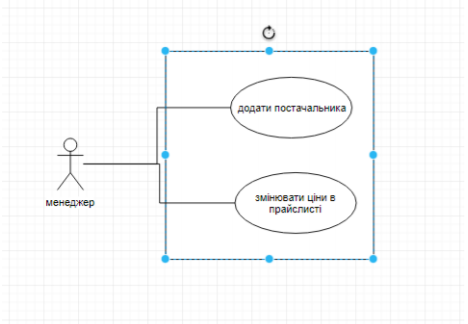


## 2.3. Use-case діаграми

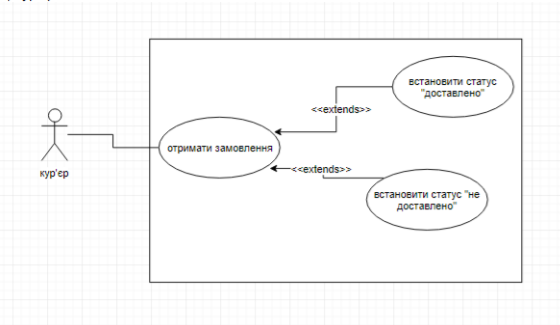
Повар :



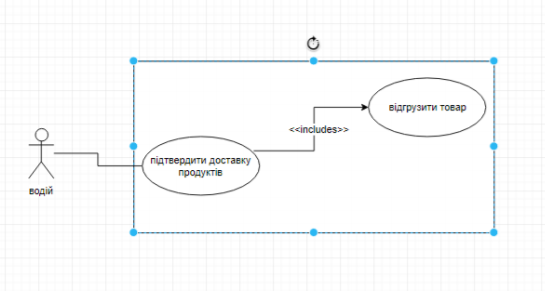
Менеджер



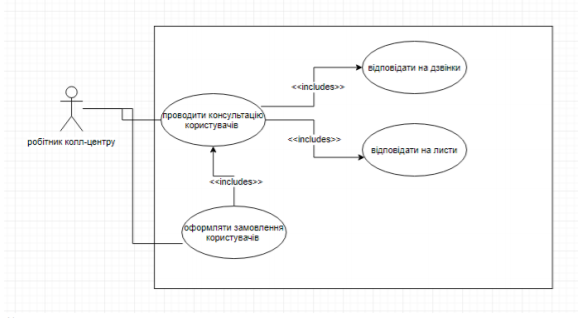
Кур’єр



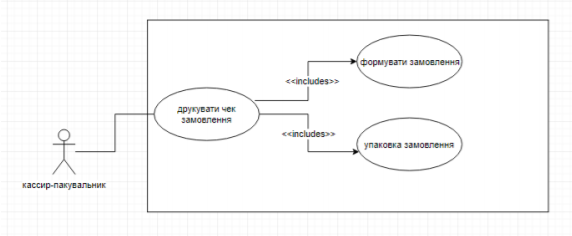
Водій



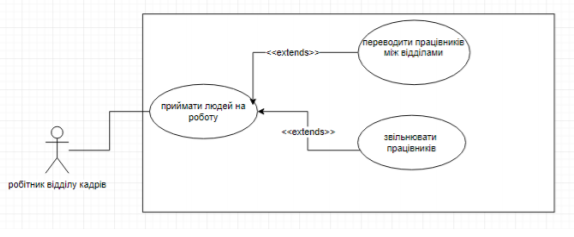
Робітник колл-центру



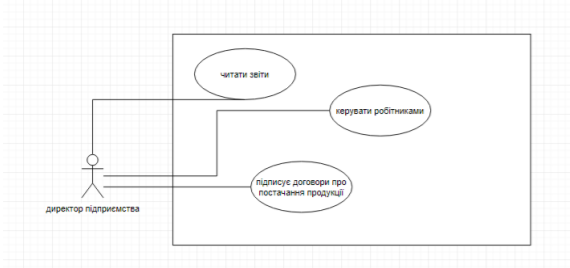
Кассир-пакувальник



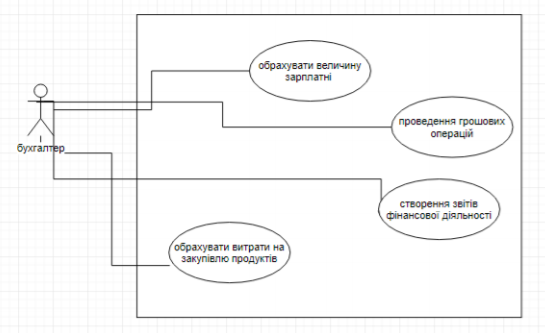
Робітник відділу кадрів



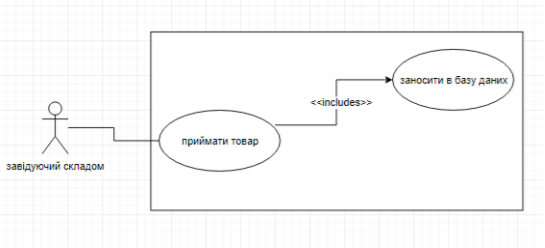
Директор підприємства



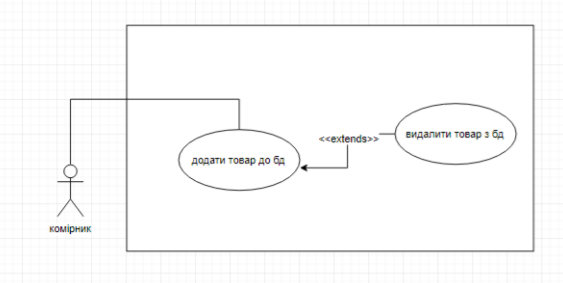
Бухгалтер



Завідуючий складом



Комірник



# 3. Налаштування реплікації

В роботі була налаштована реплікація між двома інстансами бд MySQL.

## 3.1. Теоретичні дані про реплікацію та вибір RDBMS

Реплікація дозволяє копіювати дані з одного сервера баз даних MySQL (відомого як джерело) на один або більше серверів баз даних MySQL (відомі як репліки). Реплікація за замовчуванням асинхронна; копії не потрібно постійно підключати, щоб отримувати оновлення від джерела. Залежно від конфігурації можна копіювати всі бази даних, вибрані бази даних або навіть вибрані таблиці в базі даних.

До переваг реплікації в MySQL належать:

* Розширені рішення - розподіл навантаження між кількома репліками для підвищення продуктивності. У цьому середовищі всі записи та оновлення повинні відбуватися на вихідному сервері. Однак читання можуть відбуватися на одній або декількох копіях. Ця модель може покращити продуктивність записів (оскільки джерело присвячене оновленням), одночасно різко збільшуючи швидкість читання у зростаючій кількості копій.
* Захист даних - оскільки репліка може призупинити процес реплікації, можна запустити служби резервного копіювання на репліці, не пошкоджуючи відповідні вихідні дані.
* Аналітика - дані в реальному часі можна створювати у джерелі, тоді як аналіз інформації може здійснюватися на репліці, не впливаючи на роботу джерела.
* Розподіл даних на великі відстані - за допомогою реплікації можна створити локальну копію даних для використання віддаленим сайтом без постійного доступу до джерела.

MySQL 8.0 підтримує різні методи реплікації. Традиційний метод заснований на реплікації подій із двійкового журналу джерела і вимагає синхронізації файлів журналу та позицій у них між джерелом та реплікою. Новіший метод, заснований на глобальних ідентифікаторах транзакцій (GTID), є транзакційним, і тому не вимагає роботи з файлами журналів або позиціями в цих файлах, що значно спрощує багато загальних завдань реплікації. Реплікація з використанням GTID гарантує узгодженість між джерелом та реплікою, якщо всі транзакції, здійснені у джерелі, також були застосовані до репліки.

## 3.2. Кроки, виконані, для налаштування репліцікаї

Крок 1 - налаштування Мастера :

Треба відредагувати конфігурації серверу : etc/mysql/my.cnf

bind-address = 127.0.0.1

bind-address = 12.34.56.789

server-id = 1

log\_bin = /var/log/mysql/mysql-bin.log

binlog\_do\_db = newdatabase

Збергти конфіги і рестратануть сервер .

sudo service mysql restart

Тепер треба налаштувати права для слейва :

mysql -u root -p

GRANT REPLICATION SLAVE ON \*.\* TO 'slave\_user'@'%' IDENTIFIED BY 'password';

FLUSH PRIVILEGES;

USE newdatabase;

FLUSH TABLES WITH READ LOCK;

SHOW MASTER STATUS;

Якщо результать запиту приблизно такий, то все працює

mysql> SHOW MASTER STATUS;

+------------------+----------+--------------+------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

+------------------+----------+--------------+------------------+

| mysql-bin.000001 | 107 | newdatabase | |

+------------------+----------+--------------+------------------+

1 row in set (0.00 sec)

Тепер треба експортувати базу даних, використовуючи mysqldump .

mysqldump -u root -p --opt newdatabase > newdatabase.sql

Тепер треба розблокувати базу даних

UNLOCK TABLES;QUIT;

Конфігурація майстера закінчена

Крок 2 - налаштування Слейва.

Залогуватися в слейва і створити базу даних

CREATE DATABASE newdatabase;EXIT;

Імпортувати базу даних, експортовану із майстра

mysql -u root -p newdatabase < /path/to/newdatabase.sql

Треба відредагувати конфігурацію слейва так само як і майстра :

sudo nano /etc/mysql/my.cnf

server-id = 2

relay-log = /var/log/mysql/mysql-relay-bin.loglog\_bin = /var/log/mysql/mysql-bin.logbinlog\_do\_db = newdatabase

Тепер треба ребутнуть слейв :

sudo service mysql restart

Наступний крок енейблить реплікацію :

CHANGE MASTER TO MASTER\_HOST='12.34.56.789',MASTER\_USER='slave\_user', MASTER\_PASSWORD='password', MASTER\_LOG\_FILE='mysql-bin.000001', MASTER\_LOG\_POS= 107;

Ця команда виконує кілька речей одночасно:

1. Він позначає поточний сервер як підлеглий нашого головного сервера.

2. Він надає серверу правильні дані для входу

3. Нарешті, це дозволяє веденому серверу знати, з чого почати реплікацію; головний файл журналу та позиція журналу походять від цифр, які ми записали раніше.

Тепер можна активувати слейва командою

START SLAVE;

# **4. Розробка додатку для роботи із базою данних**

В рамках роботи був розроблений клієнстський додаток для роботи із базою даних. Додаток є десктоп програмою і графічним інтерфейсом. Він дозволяє переглядати наявні таблиці в системі, переглядати записи в таблицях а також редагувати записи і сторювати нові.

## 4.1. Використані технології

В розробці використовувались такі технології: як JavaFX і Hibernate .

Платформа JavaFX являє собою набір графічних та медіа пакетів (бібліотек Java), які дозволяють програмістам проектувати, створювати , тестувати та розгортати RIA (Rich Client Applications) додатки, які фактично являють собою програми із графічним інтерфейсом, які виглядають та працюють однаково на всіх платформах та пристроях. Типова Java програма є клієнт-серверним додатком, що працює як на сервері (де зосереджена бізнес-логіка додатку) та на клієнтських пристроях, які підключаються до сервера за допомогою мережі Інтернет. Платформа JavaFX дозволяє проектувати високопродуктивні інтерфейси користувача на клієнтських додатках, які окрім базових графічних компонентів, включають відео, звук, анімацію.

Hibernate - це безкоштовний Java фреймворк з відкритим вихідним кодом, що представляє собою інструмент об'єктно-реляційного відображення (object-relational mapping - ORM). Його основним завданням є перетворення даних реляціоннной БД в об'єктно-орієнтовані моделі і назад. Крім цього фреймворк також надає іструменти для автоматичної побудови запитів, пошуку та вилучення даних.

Метою Hibernate є звільнення розробника від 95-ти відсотків виконання завдань щодо забезпечення зберігання об'єктів в реляційній базі даних. Hibernate може бути не кращим рішенням для додатків, в яких бізнес логіка побудована на виклик збережених процедур при роботі з БД. Він буде більш корисний для додатків, в яких обробка даних виконується, скажімо так, на середньому рівні, з використанням об'єктно-орієнтованих моделей.

Hibernate може бути з легкістю впроваджений на будь-якій стадії проекту, його можна використовувати як в процесі проектування класів і таблиць, так і для роботи з уже існуючою базою даних. Hibernate має природну модель програмування: фреймворк дозволяє розробляти класи слідуючи особливостям об'єктно-орієнтованої мови: успадкування, поліморфізму, а також, асоціацій і колекцій Java;

Також його плюсом є висока продуктивність: Hibernate підтримує «ледачу» ініціалізацію і безліч стратегій вибірок. Він не вимагає наявності спеціальних таблиць або полів в базі даних і генерує більшу частину SQL коду під час ініціалізації системи, а не під час виконання. Hibernate зазвичай забезпечує чудову продуктивність в порівнянні з прямим JDBC-кодом.

Також надійність і масштабованість: Hibernate добре відомий своїми відмінними якістю і стабільністю, перевіреними використанням десятками тисяч Java-розробників. Він був розроблений, щоб працювати в кластері сервера додатків і надавати високу масштабованість архітектури також Hibernate легко налаштовується і розширюється.

До недоліків бібліотеки можна віднести наступне:

1. деяка складність у використанні, особливо на початку роботи, зважаючи на велику кількість властивостей і налаштувань;
2. погано підходить для вибірок об'ємних даних і побудови складних нетривіальних запитів;
3. дуже мало грамотних прикладів і документації по використанню і тонкощам при роботі з фреймворком.\

## 4.2. Опис розробленого додатку

У ході роботи був розроблений клієнтський додаток для роботи із базою даних. Був реалізований простий інтерфейс, що складається із 3 панелей, а саме списку таблиць, таблиці, і панелі для додавання чи редагування записів.   
Реалізовані усі операції CRUD : можна додавати нові записи, видаляти, змінювати і продивлятися.

## 4.3. Скріншоти готової програми:

Перегладаємо таблицю “Статуси” і створюємо новий запис

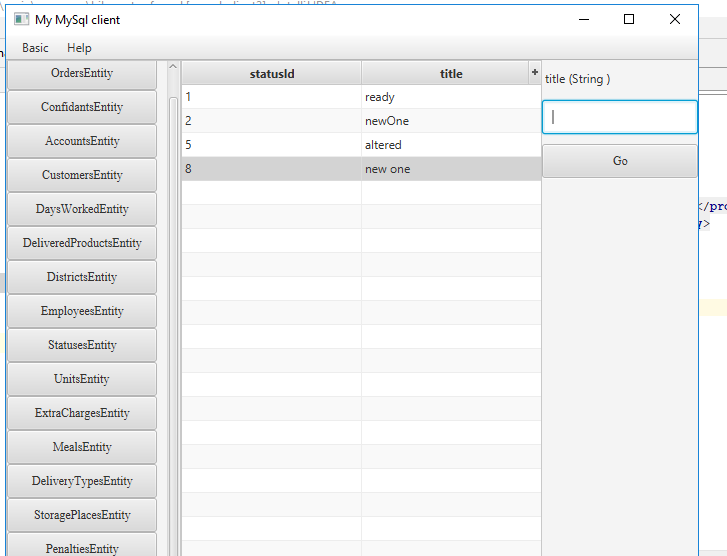


Рисунок 4.1 - Таблиця “Статуси”

Редагуємо запис під номером 5

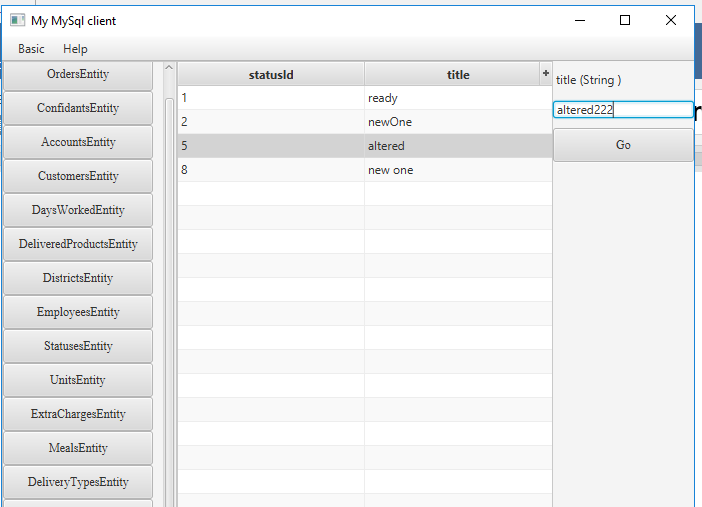


Рисунок 4.2 - Редагування запису

Видалили запис під номером 5

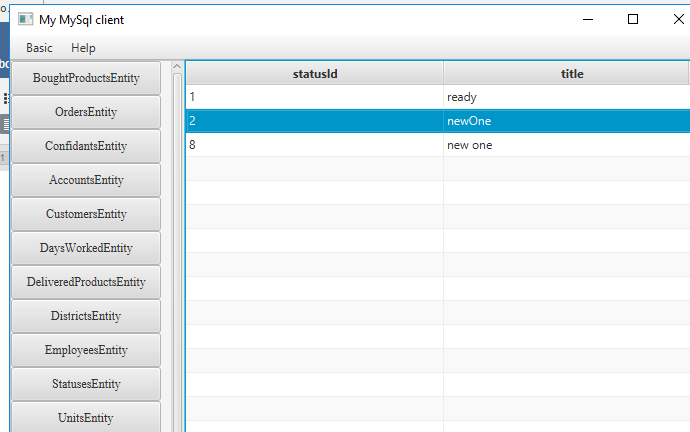


Рисунок 4.3. - Видалення запису

Іще одна таблиця

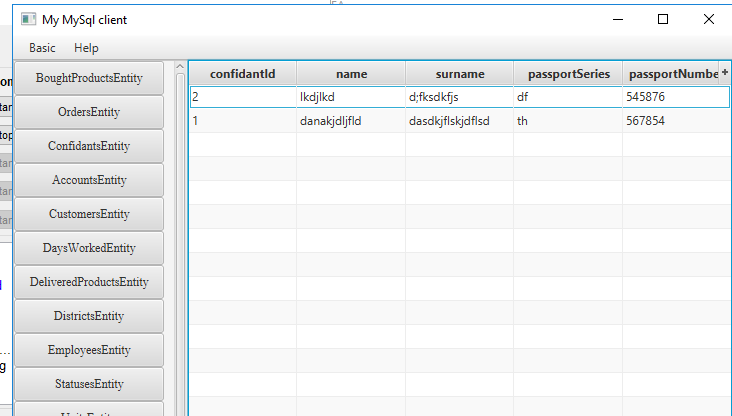


Рисунок 4.4 - Таблиця робітників

Відсортована за ідентифікатором

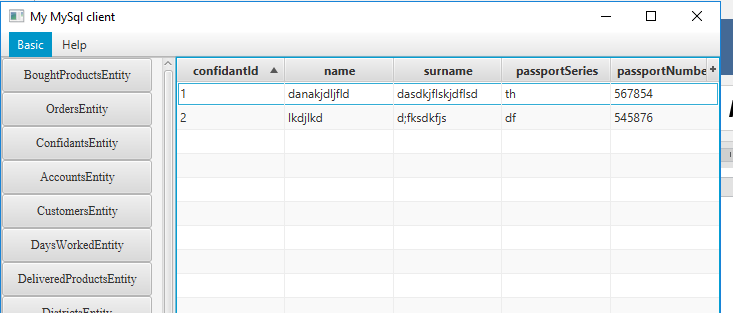


Рисунок 4.5 - Сортування таблиці

## 4.4. Елементи вихідного коду

Метод для створення панелі із полями для вводу для створення нового запису. В методі використовується рефлексія для того, аби дізнатися поля класу і для кожного із полей створити поле на формі .

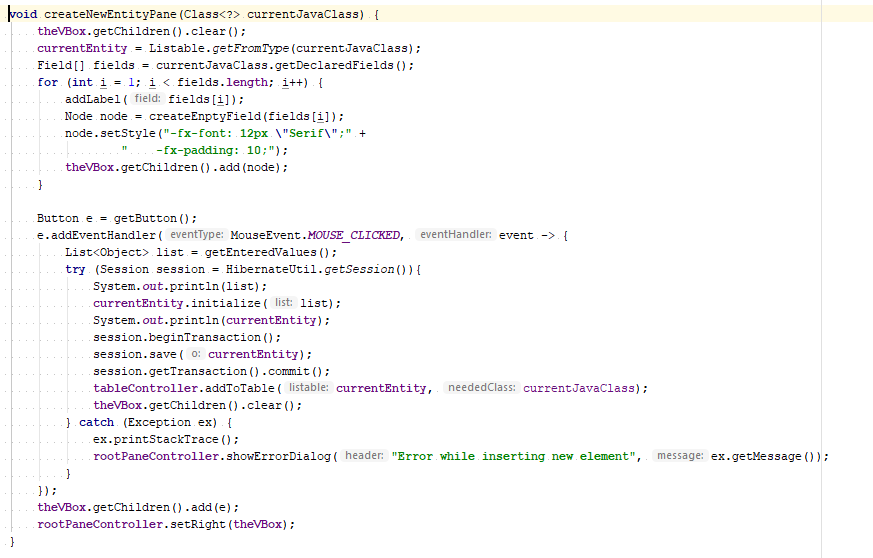


Рисунок 4.6 - Метод для створення панелі із полями

Метод для створення панелі для редагування запису

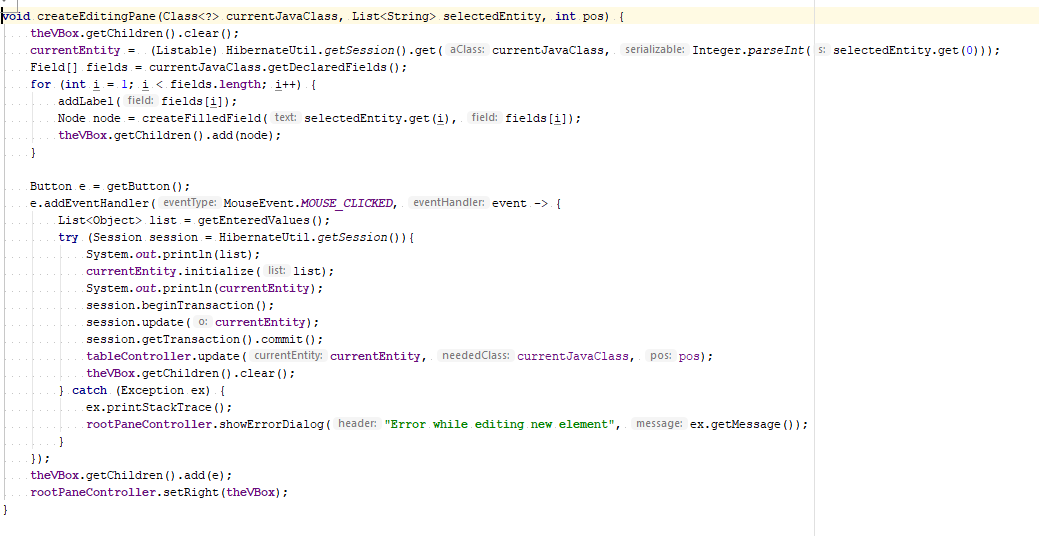


Рисунок 4.7 - Метод для створення панелі для редагування запису

Метод заповнення таблиці.

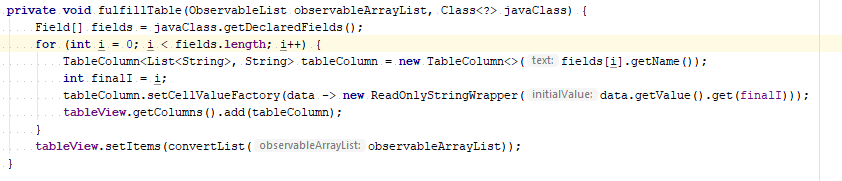


Рисунок 4.8 - Метод заповнення таблиці

Метод для видалення запису із бази даних

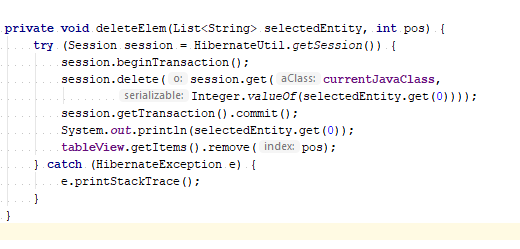
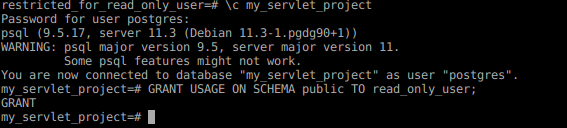
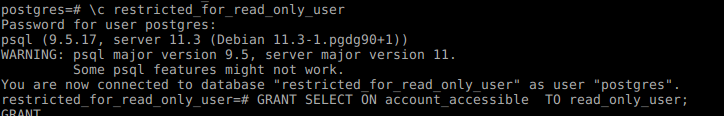
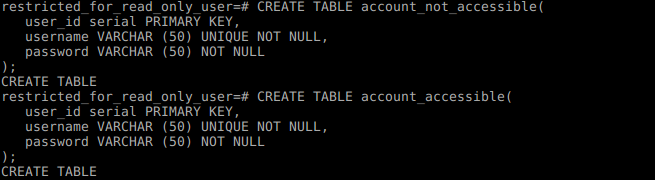


Рисунок 4.9 - Метод для видалення запису із бази даних

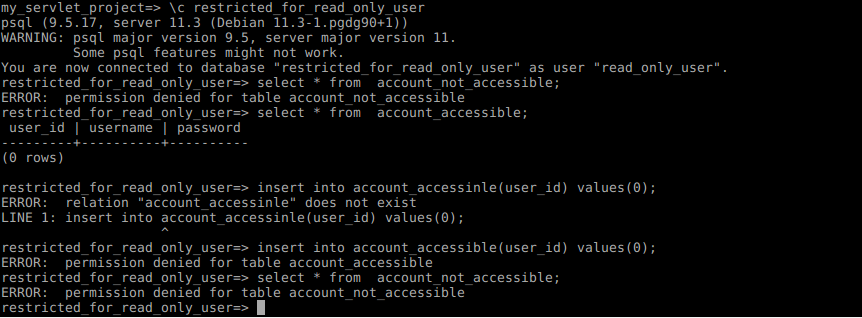
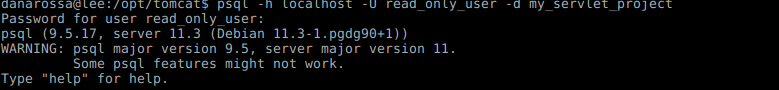
# 5. Налаштування прав користувачів

В роботі була використана база даних - PostgreSQL.

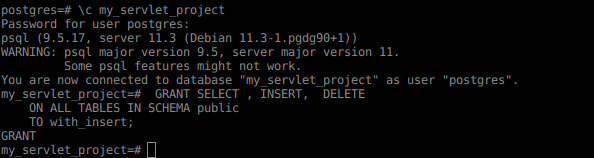
1)створюємо базу даних



2) логінимось створеним юзером



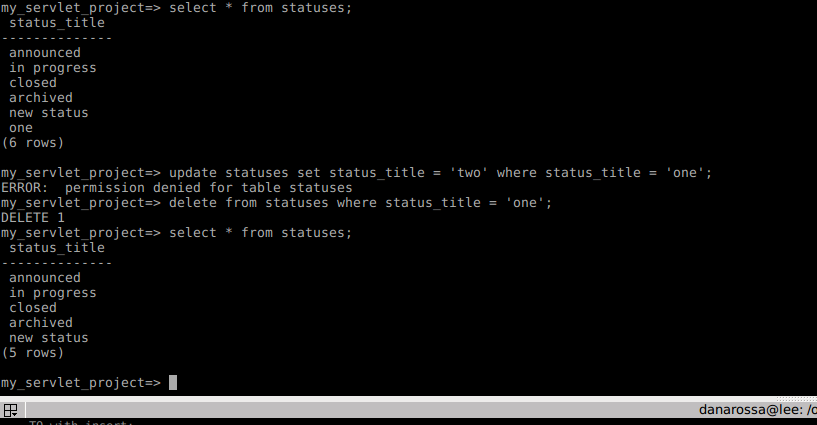
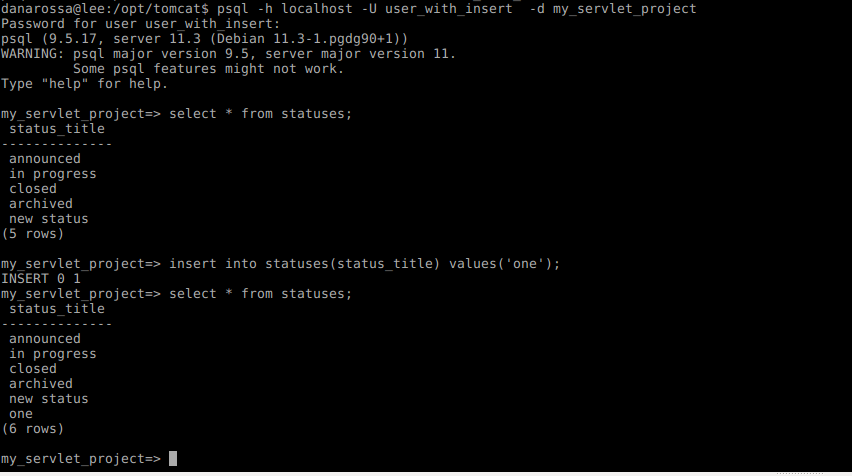
3) Грантуємо права на інсерт, селект, дилит в базі даних для ролі.



4) створюємо юзера із цією роллю



5)логінимось під цим юзером



# 6. Налаштування віртувальної таблиці

PostgreSQL реалізує частини специфікації SQL / MED, дозволяючи отримувати доступ до даних, що знаходяться за межами PostgreSQL, використовуючи звичайні запити SQL. Такі дані називаються іноземними даними (foreign data).

Доступ до іноземних даних здійснюється за допомогою зовнішньої обгортки даних. Іноземна обгортка даних (foreign data wrapper)- це бібліотека, яка може взаємодіяти із зовнішнім джерелом даних, приховуючи деталі підключення до джерела даних та отримання даних з нього. Є деякі закордонні обгортки даних, доступні як модулі contrib. Інші види закордонних обтікачів даних можуть бути представлені як сторонні продукти. .

Щоб отримати доступ до іноземних даних, потрібно створити чужий серверний об’єкт, який визначає спосіб підключення до певного зовнішнього джерела даних відповідно до набору опцій, що використовуються його підтримує зовнішньою обгорткою даних. Потім потрібно створити одну або кілька зовнішніх таблиць, які визначають структуру віддалених даних. Зовнішня таблиця може використовуватися в запитах так само, як звичайна таблиця, але зовнішня таблиця не має сховища на сервері PostgreSQL. Кожного разу, коли він використовується, PostgreSQL просить зовнішню обгортку даних отримати дані із зовнішнього джерела або передати дані зовнішньому джерелу у випадку команд оновлення.

Для доступу до віддалених даних може знадобитися автентифікація до зовнішнього джерела даних. Ця інформація може бути надана за допомогою зіставлення користувачів, яке може надати додаткові дані, такі як імена користувачів та паролі на основі поточної ролі PostgreSQL.

Модуль dblink передбачений для легкого підключення до інших баз даних або на одному хості бази даних, або на віддаленому хості.

dblink призначений для користувачів баз даних для виконання коротких спеціальних запитів в інших базах даних. dblink не призначений як заміна зовнішніх таблиць або адміністративних інструментів, таких як gptransfer.

## 6.1. Кроки налаштування віртуальної таблиці

Спочатку треба налаштувати 2 інстанси PostgeSQL та створити деякі таблиці в них.

Два контейнери були створені за допомогою docker-compose із таким файлом конфігурації :

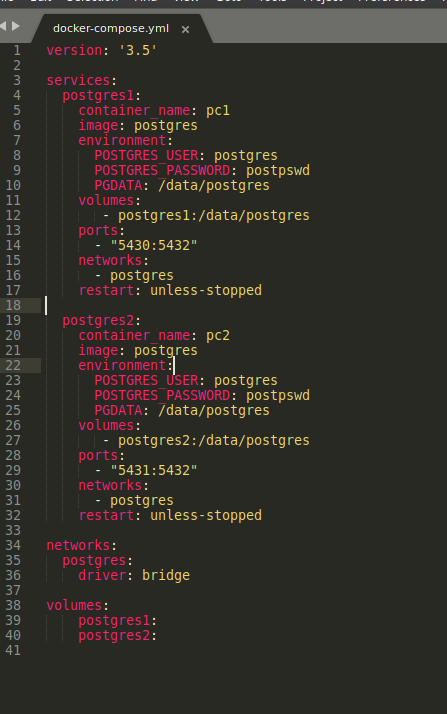


Рисунок 6.1 - Конфігурація 2 контейнерів із PostgreSQL

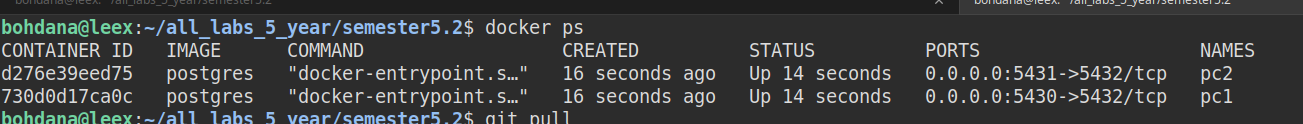


Рисунок 6.2 - Запущені контейнери

Після цього в базах даних були створені таблиці.

В таблицю були вставлені деякі записи.

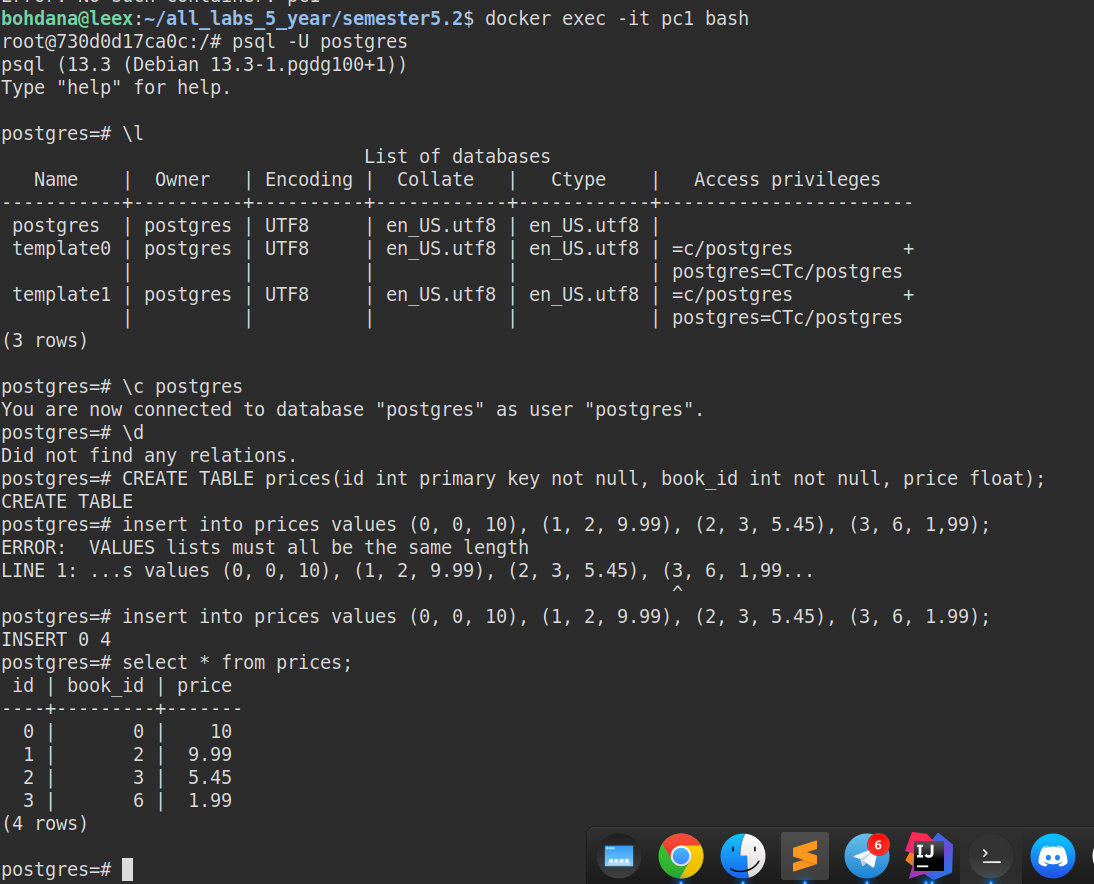


Рисунок 6.3 - Створення таблиці і заповнення даними в контейнері 1

В іншому контейнері була створена аналогічна таблиця і були вставлені деякі записи :

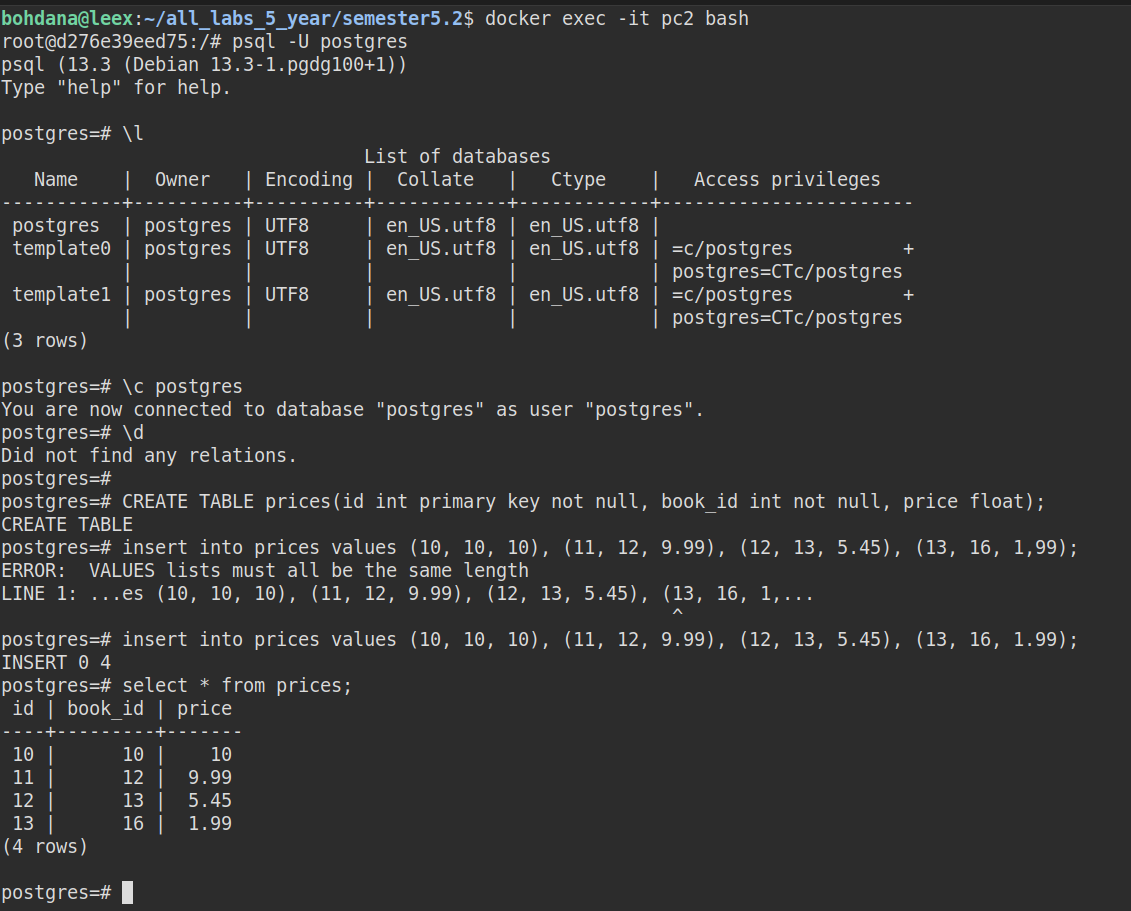


Рисунок 6.4 - Створення таблиці і заповнення даними в контейнері 2

Після цього створюємо коннекшн від одного хоста до іншого :

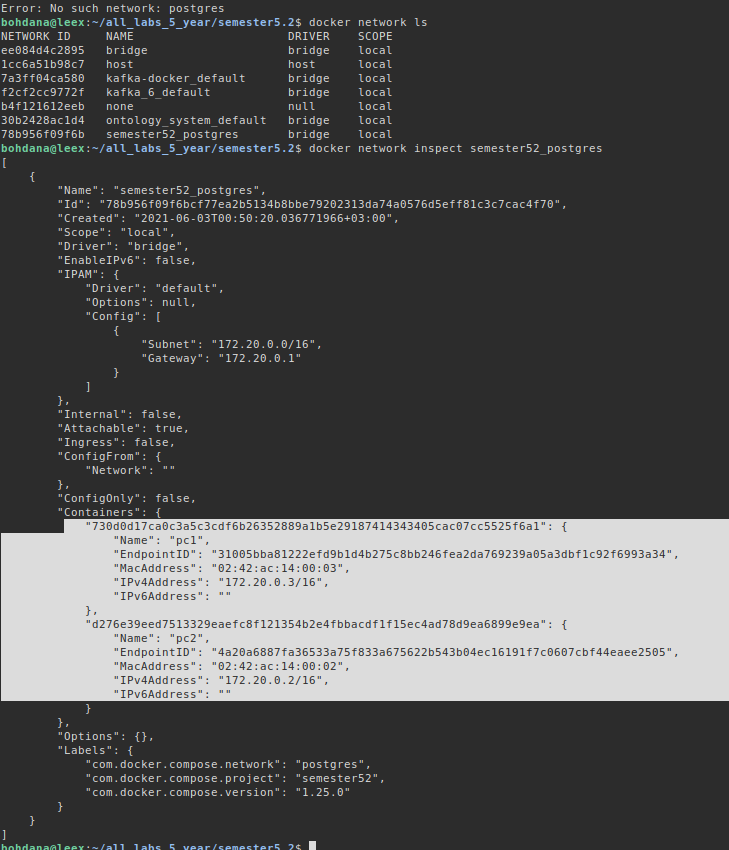


Рисунок 6.5 - Визначення IP-адрес всенедині віртуальної мережі

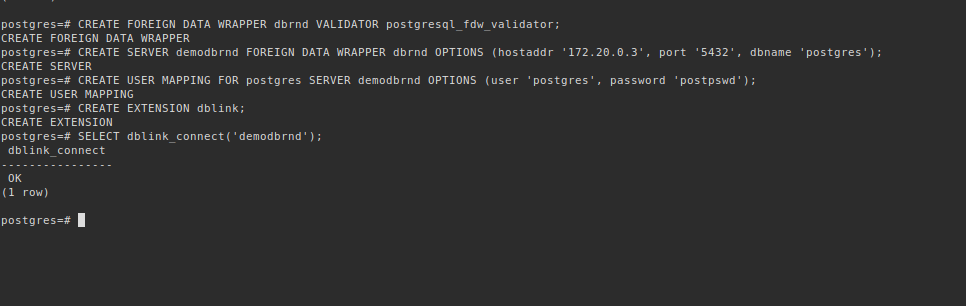


Рисунок 6.6 - Створення лінки від 2 контейнеру до першого

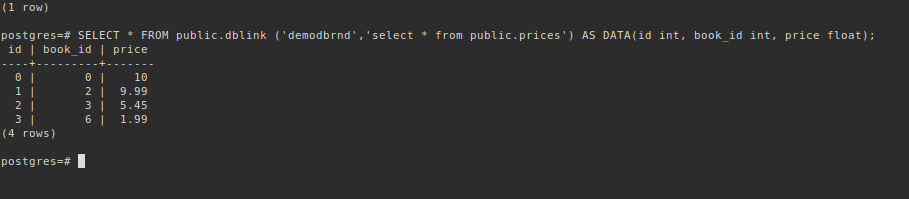


Рисунок 6.7 - Результат селекту в другому контейнері із першого, використовуючи коннекшн

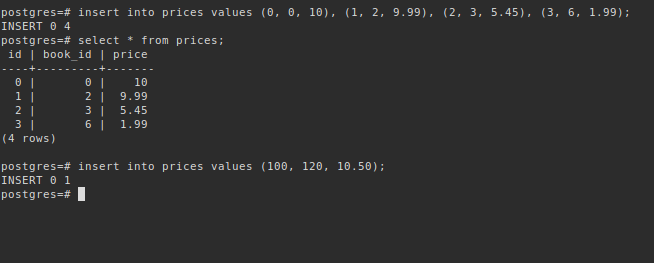


Рисунок 6.8 - Додавання нового запису в таблицю в контейнері 1

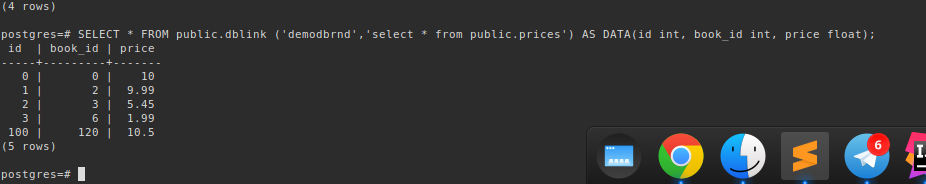


Рисунок 6.9 - Новий запис відабражається в контейнері 2

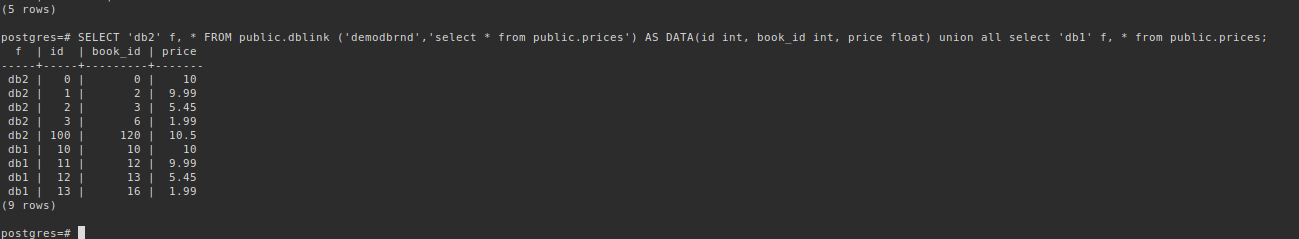


Рисунок 6.10 - Створення запиту на селект із 2 контейнерів і результат

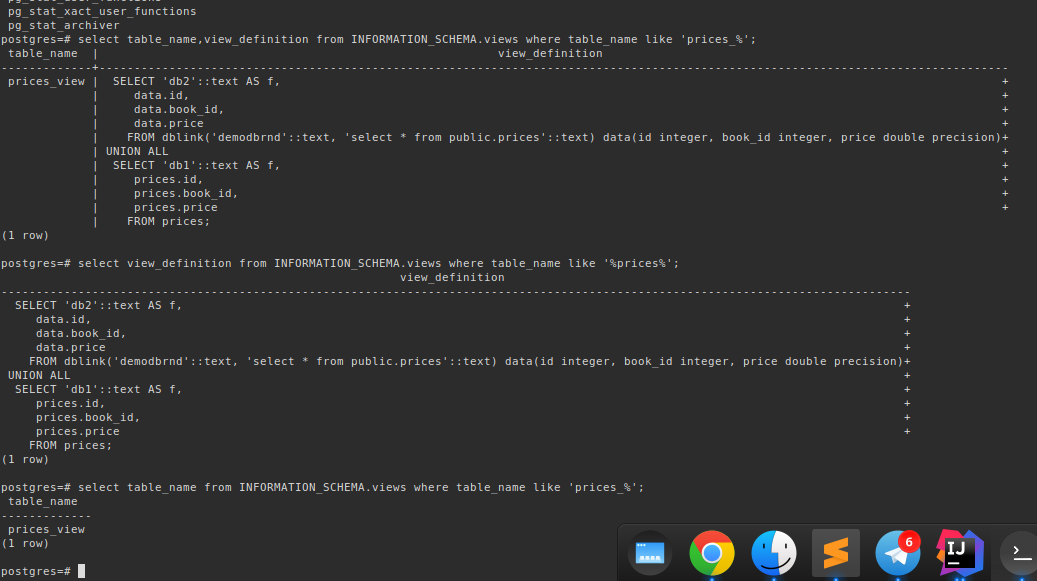


Рисунок 6.11 - Створення вью використовуючи коннекшн і запит на селект із 2 контейнерів

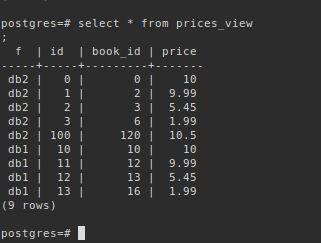


Рисунок 6.12 - Результат селекту із вью

# ВИСНОВКИ

Під час написання курсової роботи було розроблено схему розподіленої бази даних. Було налаштовано реплікацію та рівні доступу до даних для різних груп користувачів. Також було розроблено клієнстський додаток для роботи із базою даних.

Було налаштовано віртуальну таблицю в DBMS PostgreSQL за допомогою екстеншену dblink. Модуль dblink передбачений для легкого підключення до інших баз даних або на одному хості бази даних, або на віддаленому хості.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Obe, Regina; Hsu, Leo (July 8, 2012). PostgreSQL: Up and Running. O'Reilly. ISBN 978-1-4493-2633-3.
2. Matthew, Neil; Stones, Richard (April 6, 2005). Beginning Databases with PostgreSQL (second ed.). Apress. p. 664. ISBN 1-59059-478-9. Archived from the original on April 9, 2009. Retrieved April 28, 2009.
3. Gupta, Swati & Saroha, Kuntal & Bhawna, & Tech, M & Scholar, Pdmce & Bahadurgarh,. (2011). Fundamental Research of Distributed Database. International Journal of Computer Science and Management Studies. 11.
4. The Postgres95. User Manual. A. Yu and J. Chen. University of California. Berkeley, California. Sept. 5, 1995.

# ДОДАТОК 1 ВИКОРИСТАНІ СКРИПТИ

docker-compose up

**---------- pc1**

docker run -p 54320:5432 --name postgres1 -e POSTGRES\_PASSWORD=postpswd -d postgres

docker exec -it postgres1 bash

psql -U postgres

\l

\c postgres

\d

CREATE TABLE prices(id int primary key not null, book\_id int not null, price float);

insert into prices values (0, 0, 10), (1, 2, 9.99), (2, 3, 5.45), (3, 6, 1.99);

select \* from prices;

**----------- pc2**

docker exec -it pc2 bash

psql -U postgres

\l

\c postgres

\d

CREATE TABLE prices(id int primary key not null, book\_id int not null, price float);

insert into prices values (10, 10, 10), (11, 12, 9.99), (12, 13, 5.45), (13, 16, 1.99);

select \* from prices;

CREATE FOREIGN DATA WRAPPER dbrnd VALIDATOR postgresql\_fdw\_validator;

CREATE SERVER demodbrnd FOREIGN DATA WRAPPER dbrnd OPTIONS (hostaddr '172.20.0.3', port '5432', dbname 'postgres');

CREATE USER MAPPING FOR postgres SERVER demodbrnd OPTIONS (user 'postgres', password 'postpswd');

CREATE EXTENSION dblink;

SELECT dblink\_connect('demodbrnd');--OK

SELECT \* FROM public.dblink ('demodbrnd','select \* from public.prices') AS DATA(id int, book\_id int, price float);

SELECT 'db2' f, \* FROM public.dblink ('demodbrnd','select \* from public.prices') AS DATA(id int, book\_id int, price float) union all select 'db1' f, \* from public.prices;

CREATE VIEW prices\_view AS

SELECT 'db2' f, \* FROM public.dblink ('demodbrnd','select \* from public.prices') AS DATA(id int, book\_id int, price float) union all select 'db1' f, \* from public.prices;

select table\_name from INFORMATION\_SCHEMA.views where table\_name like 'prices%';

select view\_definition from INFORMATION\_SCHEMA.views where table\_name like '%prices%';

select table\_name,view\_definition from INFORMATION\_SCHEMA.views where table\_name like 'prices%';