**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Інститут **ІКНІ**

Кафедра **ПЗ**

****

**ЗВІТ**

До лабораторної роботи №3

**На тему:** «Команди додавання, оновлення та видалення даних»

**З дисципліни:** «Бази даних»

|  |
| --- |
| **Лектор:**  доцент каф. ПЗ.  Білоіваненко М.В. |
| **Виконала:**  Студентка групи ПЗ-33  Винокурова К. Г. |
| **Прийняв:**  доцент каф. ПЗ.  Білоіваненко М.В. |
| « » 2024р  ∑= |
|  |

Львів – 2024

**Тема роботи:** Команди додавання, оновлення та видалення даних

**Мета роботи**

1. Ознайомлення із способами маніпуляції даних реляційних СУБД.

2. Синтаксис команди внесення нових рядків за наборами значень або за запитом до іншої таблиці.

3. Конструювання, виконання та аналіз помилок SQL-команд INSERT, UPDATE, DELETE, TRUNCATE.

4. Використання значення за замовчуванням при модифікації даних.

5. Методи генерації унікальних значень для первинних ключів.

6. Аналіз роботи механізму підтримки цілісності посилань.

**Завдання**

1. Створення та виконання декількох тестових запитів на внесення, оновлення та видалення рядків таблиць БД.

2. Аналіз результату додавання рядків із порушенням вимог цілісності.

3. Аналіз результату оновлення рядків невідповідними типами даних.

4. Аналіз результату видалення рядків із основної таблиці, до якої встановлено каскадне видалення за зовнішнім ключем.

5. Розробка запиту на дублювання рядку із заміною значення.

6. Розробка запиту на перенесення даних із іншої таблиці.

7. Розробка запиту, якій одночасно додає рядки і в основну, і в зв’язану таблиці, використовуючи автоматично створене значення ключа.

8. Виконання запитів заміни значень первинного і зовнішнього ключів.

9. Виконання запитів модифікації із використанням спеціальних слів DEFAULT та NULL у якості значення.

10.Виконання запитів модифікації із застосуванням вбудованої функції у відповідності до типу даних колонок.

11.Розробка програми для генерації коректно зв’язаних випадкових даних обраної ПО для заповнення всіх наявних таблиць БД.

12.Очищення таблиць та пакетне виконання команд завантаження демонстраційних даних.

**Хід виконання**

1. Скріни екрану із тестовими SQL-запитами, результатами виконання та новим вмістом таблиць.

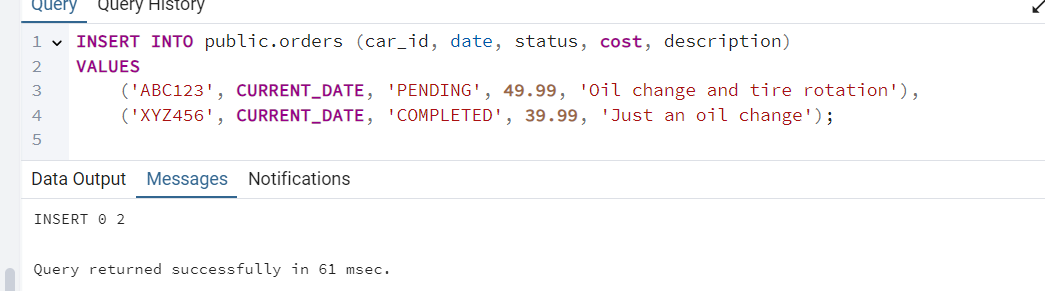


Рис.1. Операція вставки даних

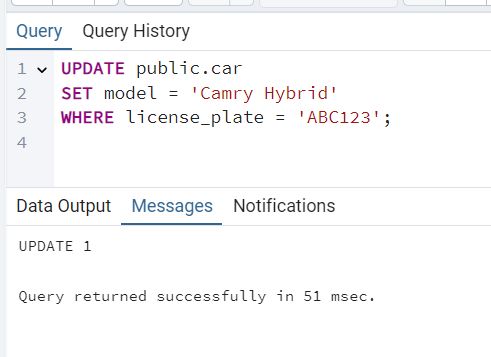
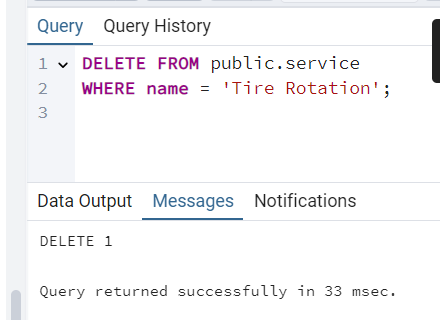


Рис.2. Запит на оновлення

  
Рис.3. Запит на видалення

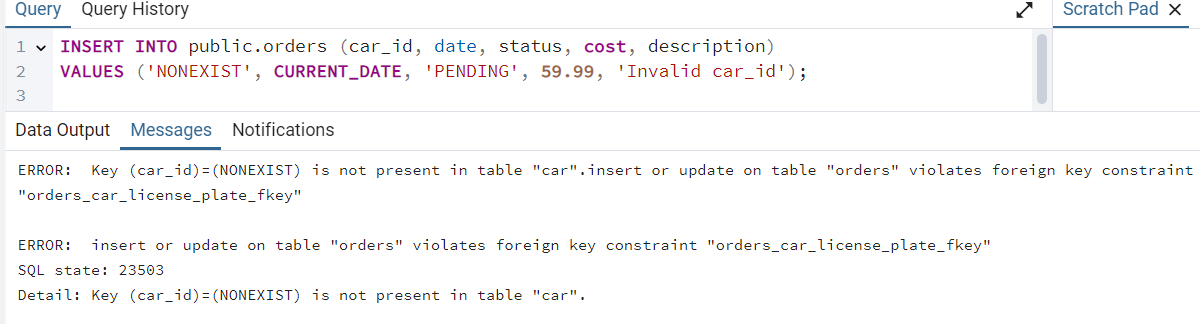
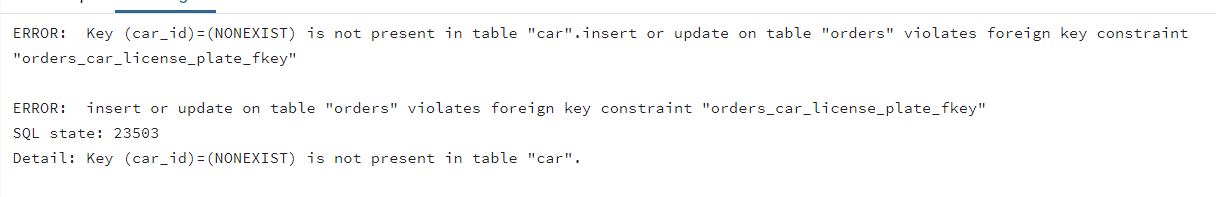


Рис.4. Спроба вставити замовлення для автомобіля, якого не існує в таблиці car (license\_plate)

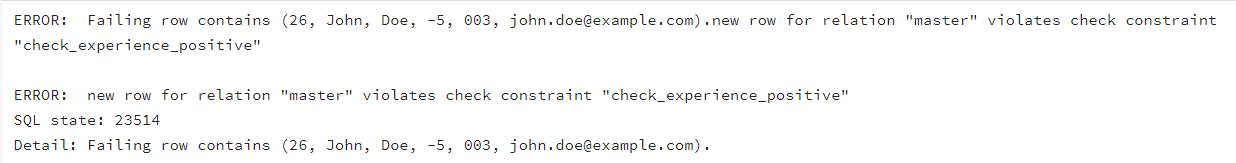


Рис.5. Вставка з порушенням CHECK обмеження

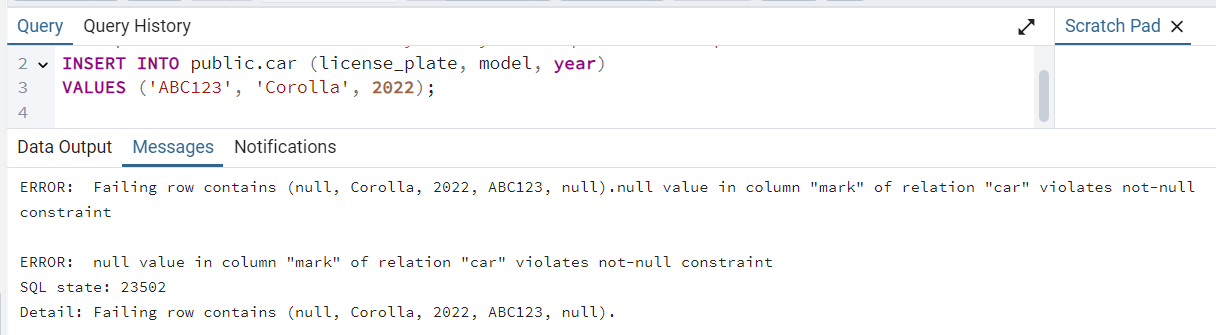
2. Текст і коментар деяких помилок, що генеруються СУБД при спробі порушення цілісності та невідповідності типу даних.



**Очікуваний результат:** Помилка через порушення зовнішнього ключа на car\_id, так як автомобіля з license\_plate = 'NONEXIST' не існує.



**Очікуваний результат:** Помилка через порушення обмеження CHECK на поле experience, оскільки значення має бути більше за 0.



**Очікуваний результат:** Помилка через порушення унікальності. Ліцензійний номер 'ABC123' вже існує в таблиці car.

3. Код запитів, які виконують перенесення даних із 1 іншої таблиці, одночасне внесення до 2 зв’язаних таблиць.



Рис.9. Вставка даних до двох зв’язних таблиць



Рис.10. Створення та заповнення таблиці, з якої будем переносити данні

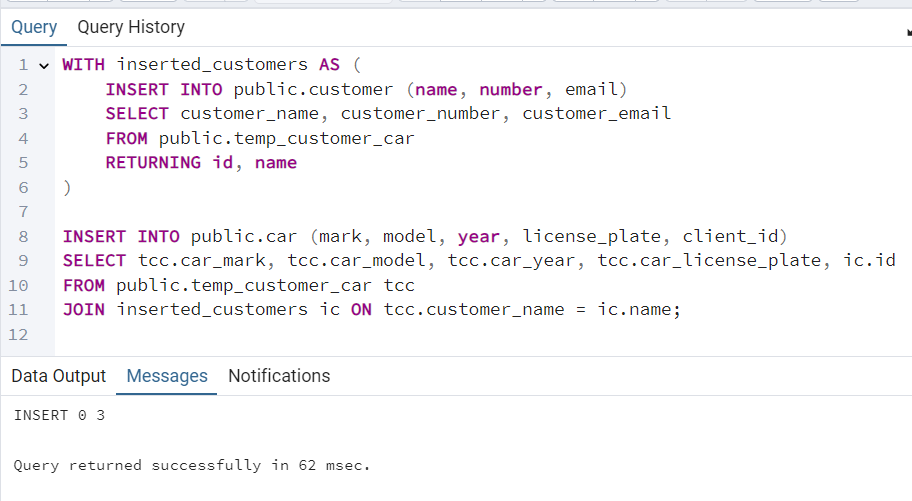


Рис.11. Перенесення даних з тимчасової таблиці в таблиці customer та car

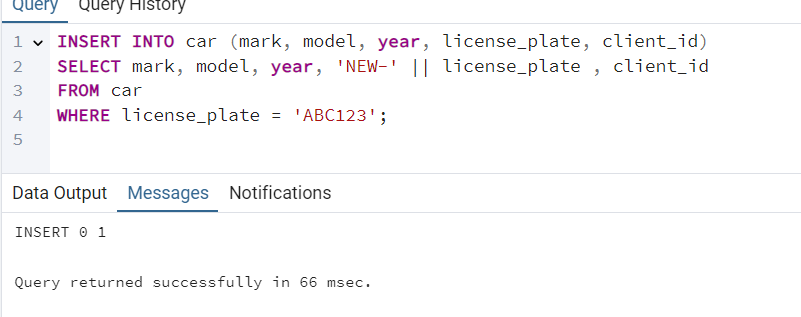


Рис.12. Дублювання рядка із заміною значення

4. Опис дій СУБД при зміні значень первинних та зовнішніх ключів.

У базах даних із реляційною структурою первинні та зовнішні ключі виконують важливу функцію забезпечення цілісності даних і правильних зв'язків між таблицями. Дії, які виконує СУБД при зміні цих ключів, можуть мати серйозні наслідки для даних у базі.

**1. Первинний ключ (Primary Key):**

**Зміна первинного ключа:**

* СУБД зазвичай не дозволяє змінювати значення первинного ключа напряму, оскільки це може порушити зв'язки з іншими таблицями через зовнішні ключі.
* При спробі змінити значення первинного ключа, зазвичай СУБД повертає помилку або потребує спеціальних умов для виконання такого запиту.

**Видалення рядка з первинним ключем:**

* Якщо рядок із первинним ключем видаляється, СУБД перевіряє, чи є на цей ключ посилання у зовнішніх таблицях.
* Якщо на цей первинний ключ посилається зовнішній ключ іншої таблиці, застосовуються визначені правила, такі як ON DELETE NO ACTION, ON DELETE CASCADE, ON DELETE SET NULL, або ON DELETE RESTRICT.

**2. Зовнішній ключ (Foreign Key):**

**Зміна значення зовнішнього ключа:**

* При зміні значення зовнішнього ключа СУБД перевіряє, чи таке нове значення існує в таблиці, на яку він посилається.
* Якщо значення первинного ключа, на яке посилається зовнішній ключ, не існує, СУБД повертає помилку про порушення цілісності.

**Дії при зміні або видаленні рядка з первинним ключем:**

* Якщо в таблиці з первинним ключем видаляється рядок, на який посилаються рядки в іншій таблиці, СУБД виконує дію відповідно до правил, які були встановлені під час створення зовнішнього ключа.

**3. Дії за правилами зовнішнього ключа:**

При створенні зовнішнього ключа можуть бути визначені правила, які регулюють, що робити при видаленні або зміні первинного ключа:

* **ON DELETE CASCADE:**
  + Якщо рядок із первинним ключем видаляється, всі рядки в інших таблицях, які посилаються на цей ключ, автоматично видаляються.
  + Це корисно, коли видалення об'єкта повинно призводити до автоматичного видалення всіх пов'язаних даних.
* **ON DELETE SET NULL:**
  + Якщо рядок із первинним ключем видаляється, зовнішній ключ у всіх залежних рядках встановлюється в NULL.
  + Це використовується, коли можна залишити рядки без зв'язку з іншими таблицями після видалення даних.
* **ON DELETE NO ACTION:**
  + Видалення рядка з первинним ключем заборонено, якщо на нього є посилання. СУБД поверне помилку, вказуючи на порушення цілісності.
  + Це стандартна дія за замовчуванням, якщо жодне інше правило не визначено.
* **ON DELETE RESTRICT:**
  + СУБД не дозволяє видалити рядок із первинним ключем, якщо на нього є залежні посилання, подібно до NO ACTION. Однак ця дія може бути активною безпосередньо під час виконання запиту, а не після завершення транзакції.

**Код розробленої програми**

package org.example**;**import java.io.BufferedWriter**;**import java.io.FileWriter**;**import java.io.IOException**;**import java.util.ArrayList**;**import java.util.List**;**import java.util.Random**;**import com.github.javafaker.Faker**;**public class DataGenerator {  
  
 private static final Random *random* = new Random()**;** private static final Faker *faker* = new Faker()**;** private static List<Integer> *customerIds* = new ArrayList<>()**;** private static List<String> *licensePlates* = new ArrayList<>()**;** private static List<Integer> *serviceIds* = new ArrayList<>()**;** private static List<Integer> *masterIds* = new ArrayList<>()**;** private static List<Integer> *orderIds* = new ArrayList<>()**;** private static List<Skills> *skills* = new ArrayList<>()**;** private static BufferedWriter *writer***;** public static void main(String[] args) {  
 try {  
 *writer* = new BufferedWriter(new FileWriter("bebe.txt"))**;** *generateClearAndResetSequenceSQL*()**;** *generateCustomers*(**10**)**;** *generateCars*(**10**)**;** *generateServices*(**10**)**;** *generateMasters*(**10**)**;** *generateSkills*(**20**)**;** *generateOrders*(**10**)**;** *generateServiceItems*(**10**)**;** } catch (IOException e) {  
 throw new RuntimeException(e)**;** } finally {  
 if (*writer* != null) {  
 try {  
 *writer*.close()**;** } catch (IOException e) {  
 e.printStackTrace()**;** }  
 }  
 }  
 }  
  
 private static void generateClearAndResetSequenceSQL() throws IOException {  
 String[] tablesWithAutoIncrement = {"customer"**,** "master"**,** "orders"**,** "service"}**;** String[] tables = {"customer"**,** "car"**,** "orders"**,** "service"**,** "master"**,** "archive"**,** "service\_item"**,** "archive"**,** "skills"}**;** for (String table: tables){  
 *writer*.write(String.*format*("DELETE FROM %s;\n"**,** table))**;** }  
  
 for (String table : tablesWithAutoIncrement) {  
 *writer*.write(String.*format*("ALTER SEQUENCE %s\_id\_seq RESTART WITH 1;\n"**,** table))**;** }  
  
 }  
  
 public static void generateCustomers(int numberOfCustomers) throws IOException {  
 for (int i = **0;** i < numberOfCustomers**;** i++) {  
 String fullName**;** do {  
 fullName = *faker*.name().fullName()**;** }while (fullName.contains("'"))**;** String phoneNumber = *generatePhoneNumber*()**;** String email = *faker*.internet().emailAddress()**;** String sql = String.*format*("INSERT INTO customer (full\_name, phone\_number, email) VALUES ('%s', '%s', '%s') RETURNING id;"**,** fullName**,** phoneNumber**,** email)**;** *writer*.write(sql)**;** *writer*.newLine()**;** int generatedCustomerId = i + **1;** *customerIds*.add(generatedCustomerId)**;** }  
 }  
  
 private static String generatePhoneNumber() {  
 return "+380" + *faker*.number().digits(**9**)**;** }  
  
 public static void generateCars(int numberOfCars) throws IOException {  
 for (int i = **0;** i < numberOfCars**;** i++) {  
 String mark = *faker*.company().name()**;** String model = *faker*.commerce().productName()**;** int year = *faker*.number().numberBetween(**1990, 2024**)**;** String licensePlate = *faker*.bothify("??###??")**;** int randomCustomerId = *customerIds*.get(*random*.nextInt(*customerIds*.size()))**;** String sql = String.*format*("INSERT INTO car (mark, model, year, license\_plate, client\_id) VALUES ('%s', '%s', %d, '%s', %d);"**,** mark**,** model**,** year**,** licensePlate**,** randomCustomerId)**;** *writer*.write(sql)**;** *writer*.newLine()**;** *licensePlates*.add(licensePlate)**;** }  
 }  
  
 public static void generateServices(int numberOfServices) throws IOException {  
 for (int i = **0;** i < numberOfServices**;** i++) {  
 String name = *faker*.commerce().productName()**;** double price = *faker*.number().randomDouble(**2, 100, 1000**)**;** String description = *faker*.lorem().sentence(**10**)**;** String type = String.*valueOf*(*getRandomSpecialization*())**;** String sql = String.*format*("INSERT INTO service (name, price, description, type) VALUES ('%s', %.2f, '%s', '%s') RETURNING id;"**,** name**,** price**,** description**,** type)**;** *writer*.write(sql)**;** *writer*.newLine()**;** int generatedServiceId = i + **1;** *serviceIds*.add(generatedServiceId)**;** }  
 }  
  
 public static void generateMasters(int numberOfMasters) throws IOException {  
 for (int i = **0;** i < numberOfMasters**;** i++) {  
 String name**;** do {  
 name = *faker*.name().firstName()**;** }while (name.contains("'"))**;** String surname**;** do {  
 surname = *faker*.name().firstName()**;** }while (surname.contains("'"))**;** String phoneNumber = *generatePhoneNumber*()**;** String email = *faker*.internet().emailAddress()**;** Random random = new Random()**;** Integer exp = random.nextInt(**20**) + **1;** String sql = String.*format*("INSERT INTO master (name, surname, phone\_number, email, experience) VALUES ('%s', '%s', '%s', '%s', '%d') RETURNING id;"**,** name**,** surname**,** phoneNumber**,** email**,** exp)**;** *writer*.write(sql)**;** *writer*.newLine()**;** int generatedMasterId = i + **1;** *masterIds*.add(generatedMasterId)**;** }  
 }  
  
 public static void generateServiceItems(int numberOfServiceItems) throws IOException {  
 for (int i = **0;** i < numberOfServiceItems**;** i++) {  
 int randomOrderId = *orderIds*.get(*random*.nextInt(*orderIds*.size()))**;** int randomId = *random*.nextInt(*skills*.size())**;** int randomServiceId = *skills*.get(randomId).serviceId**;** int randomMasterId = *skills*.get(randomId).masterId**;** String comment = *faker*.lorem().sentence(**5**)**;** String sql = String.*format*("INSERT INTO service\_item (order\_id, service\_id, master\_id, comment) VALUES (%d, %d, %d, '%s');"**,** randomOrderId**,** randomServiceId**,** randomMasterId**,** comment)**;** *writer*.write(sql)**;** *writer*.newLine()**;** }  
 }  
  
 public static void generateOrders(int numberOfOrders) throws IOException {  
 for (int i = **0;** i < numberOfOrders**;** i++) {  
 String carId = *licensePlates*.get(*random*.nextInt(*licensePlates*.size()))**;** String date = *faker*.date().past(**30,** java.util.concurrent.TimeUnit.*DAYS*).toString()**;** String status = *random*.nextBoolean() ? "COMPLETED" : "PENDING"**;** double cost = *faker*.number().randomDouble(**2, 100, 1000**)**;** String description = *faker*.lorem().sentence(**5**)**;** String sql = String.*format*("INSERT INTO orders (car\_id, date, status, cost, description) VALUES ('%s', '%s', '%s', %.2f, '%s') RETURNING id;"**,** carId**,** date**,** status**,** cost**,** description)**;** *writer*.write(sql)**;** *writer*.newLine()**;** int generatedOrderId = i + **1;** *orderIds*.add(generatedOrderId)**;** }  
 }  
  
 public static void generateSkills(int numberOfSkills) throws IOException {  
 for (int i = **0;** i < numberOfSkills**;** i++) {  
 int randomMasterId**;** int randomServiceId**;** do {  
 randomMasterId = *masterIds*.get(*random*.nextInt(*masterIds*.size()))**;** randomServiceId = *serviceIds*.get(*random*.nextInt(*serviceIds*.size()))**;** } while (*skills*.contains(new Skills(randomMasterId**,** randomServiceId)))**;** String sql = String.*format*("INSERT INTO skills (master\_id, service\_id) VALUES (%d, %d);"**,** randomMasterId**,** randomServiceId)**;** *writer*.write(sql)**;** *writer*.newLine()**;** *skills*.add(new Skills(randomMasterId**,** randomServiceId))**;** }  
 }  
  
  
 private static SpecializationEnum getRandomSpecialization() {  
 SpecializationEnum[] specializations = SpecializationEnum.*values*()**;** return specializations[*random*.nextInt(specializations.length)]**;** }  
  
}

**Результат виконання програми:**

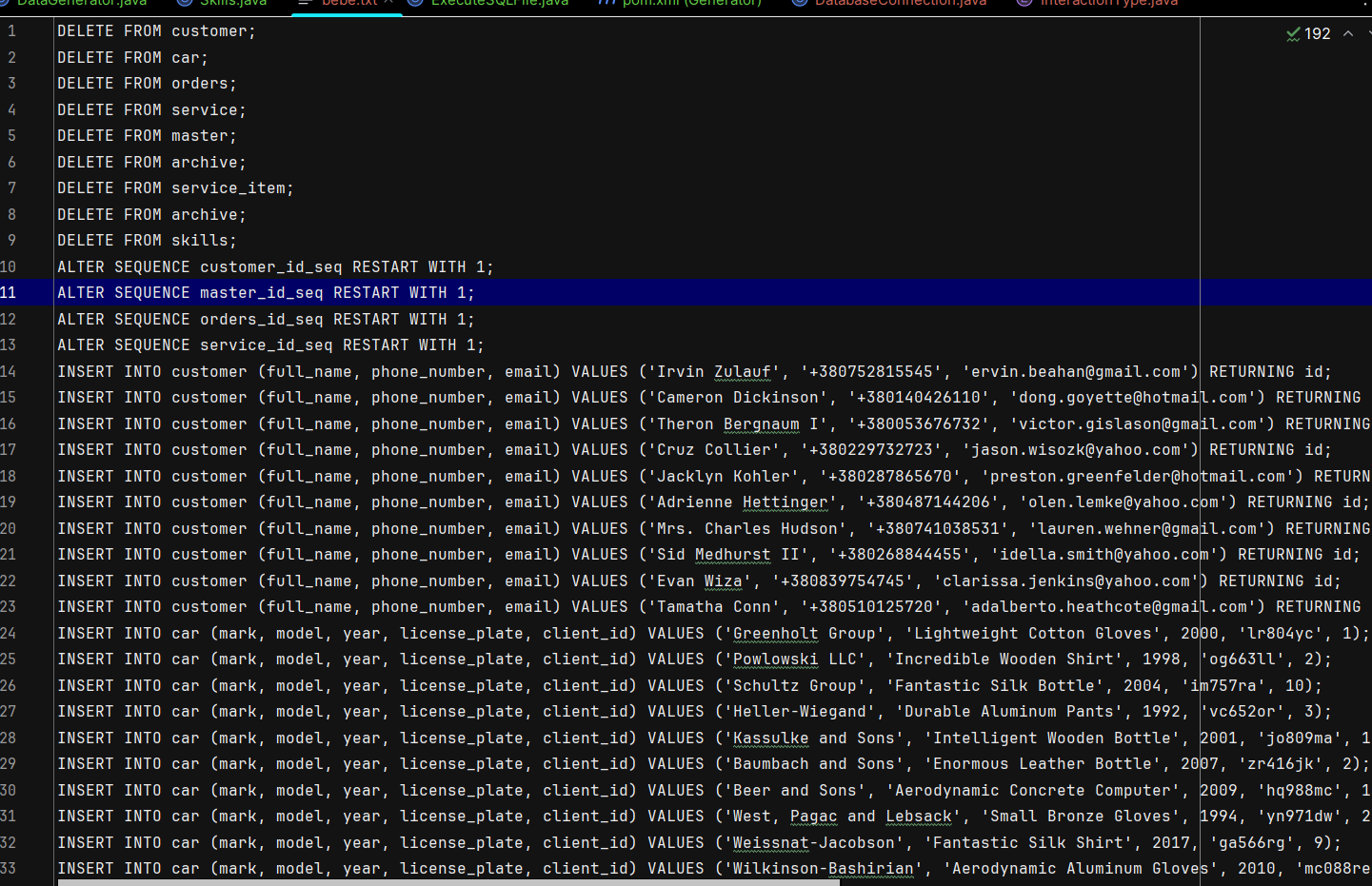


Рис.12. Згенеровані скрипти

**Висновки**

У ході виконання даної лабораторної роботи було розглянуто команди додавання, оновлення та видалення даних, а також детально вивчено механізми підтримки цілісності та унікальності інформації в базах даних.

Основні результати:

1. **Ознайомлення з командами INSERT, UPDATE, DELETE та TRUNCATE**: Я навчилася використовувати ці команди для внесення, редагування та видалення даних у таблицях бази даних, а також зрозуміла, як вони впливають на структуру та зв’язки між таблицями.
2. **Додавання нових рядків**: Було виконано вставку нових записів як за заданими значеннями, так і через запити до інших таблиць, що дозволило більш ефективно працювати з великими обсягами даних та перенесенням інформації.
3. **Оновлення та видалення даних**: Під час оновлення записів я зіткнулася з проблемами, пов’язаними з невідповідними типами даних та порушеннями цілісності зовнішніх ключів. Це дозволило зрозуміти важливість ретельної перевірки цілісності даних та налаштування каскадного видалення.
4. **Використання значень за замовчуванням**: Я застосовувала значення за замовчуванням (DEFAULT та NULL), що значно полегшило роботу з таблицями, особливо коли не всі поля мають бути заповнені вручну.
5. **Генерація унікальних значень для первинних ключів**: Було досліджено методи генерації унікальних значень для первинних ключів за допомогою автоінкременту та секвенсів. Це забезпечило унікальність кожного запису та уникнення конфліктів у таблицях.
6. **Механізми забезпечення цілісності даних**: Особливу увагу було приділено підтримці референтної цілісності через налаштування зовнішніх ключів, що дозволило уникати конфліктів при оновленні або видаленні даних у зв'язаних таблицях.
7. **Автоматична генерація випадкових даних**: Я розробила програму для генерації випадкових даних, яка допомогла заповнити всі таблиці бази коректно зв’язаними даними. Це показало важливість автоматизації при роботі з великими обсягами даних.