

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 2

з дисципліни “Бази даних”

# тема “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

|  |  |
| --- | --- |
| Виконав  Студент II курсу  групи КП-03  Павлюшин Максим Юрійович  варіант №12 |  |

Київ 2021

**Мета роботи**

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних

PostgreSQL.

**Постановка завдання**

1. Реалізувати функції внесення, редагування та вилучення даних у

таблицях бази даних, створених у лабораторній роботі No1, засобами

консольного інтерфейсу.

2. Передбачити автоматичне пакетне генерування «рандомізованих»

даних у базі.

3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та

більше сутностей одночасно: для числових атрибутів – у рамках

діапазону, для рядкових – як шаблон функції LIKE оператора SELECT

SQL, для логічного типу – значення True/False, для дат – у рамках

діапазону дат.

4. Програмний код виконати згідно шаблону MVC

(модель-подання-контролер).

**Вимоги до інструментарію**

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.6-3.7

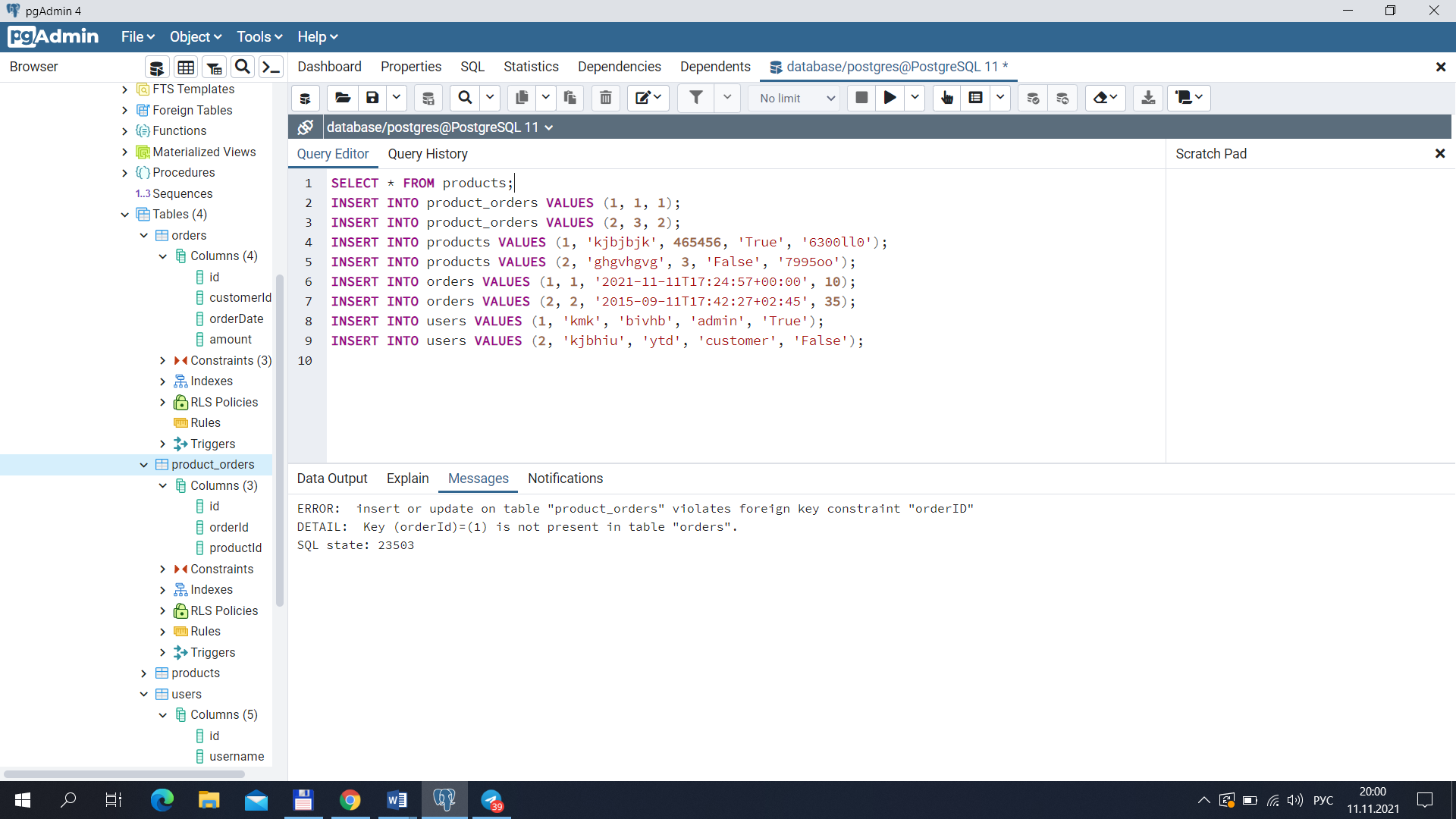
Середовище розробки програмного забезпечення – PyCharm Community Edition

2020.

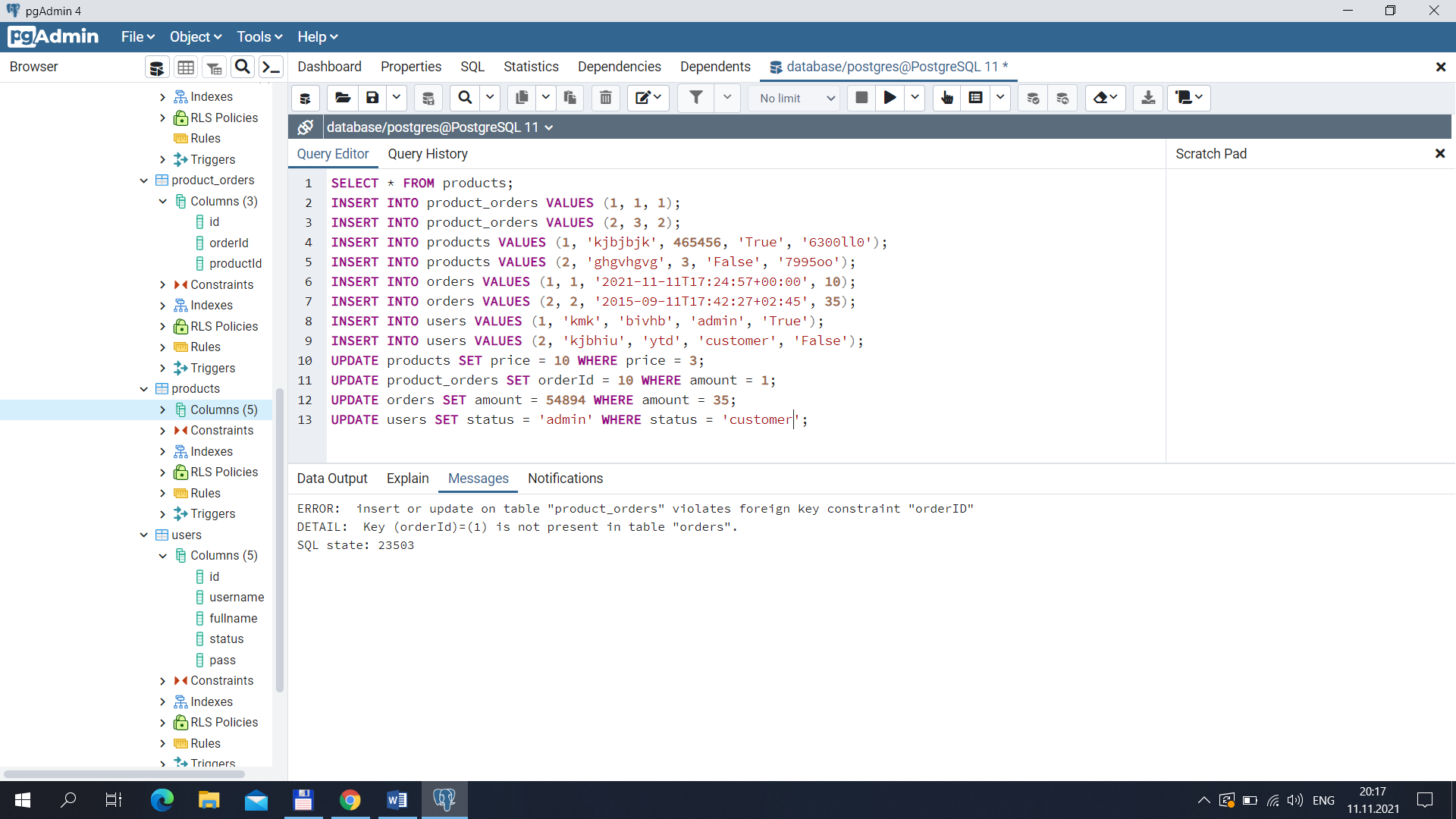
**Результат роботи**

1.

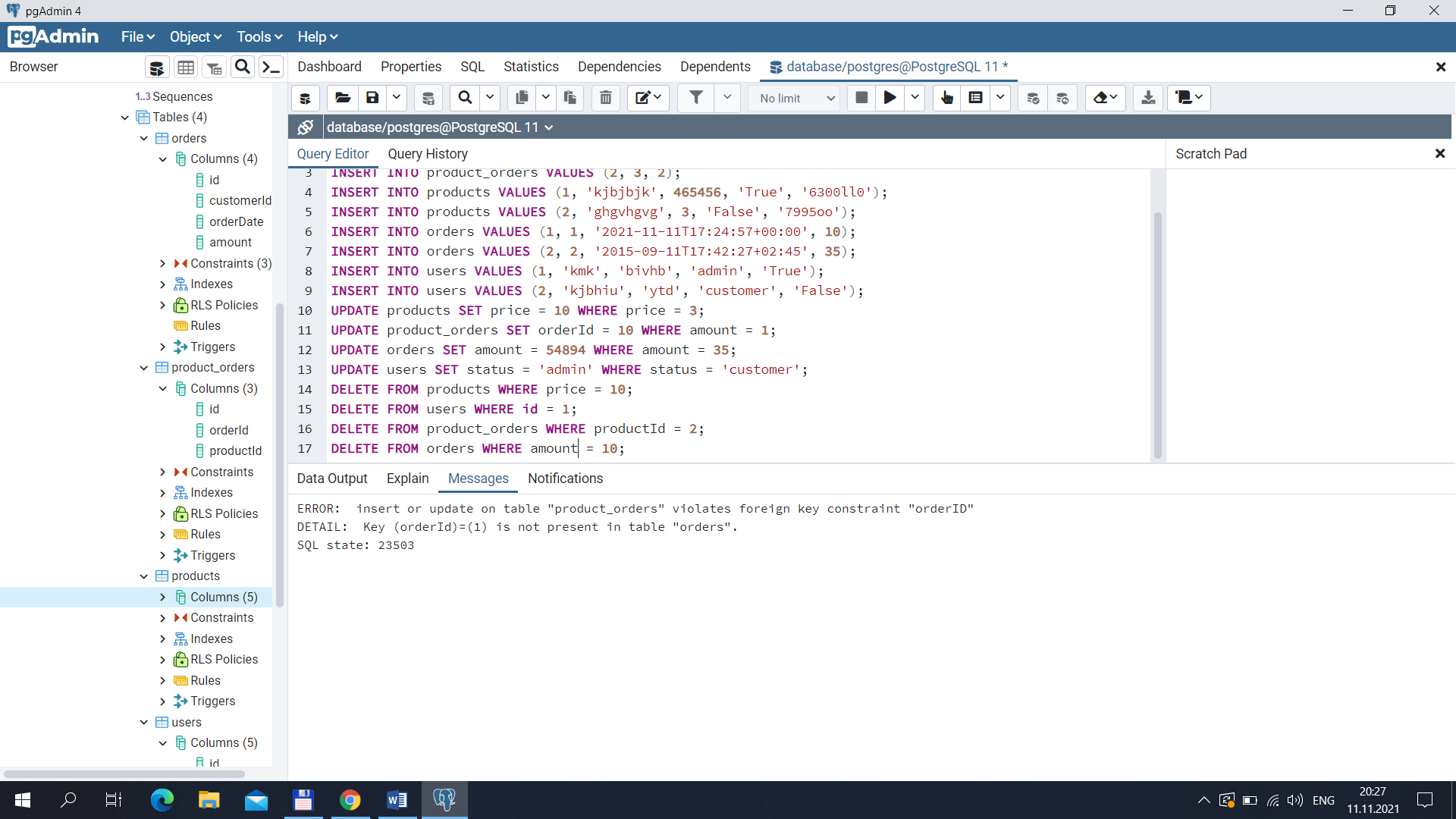
Внесення даних



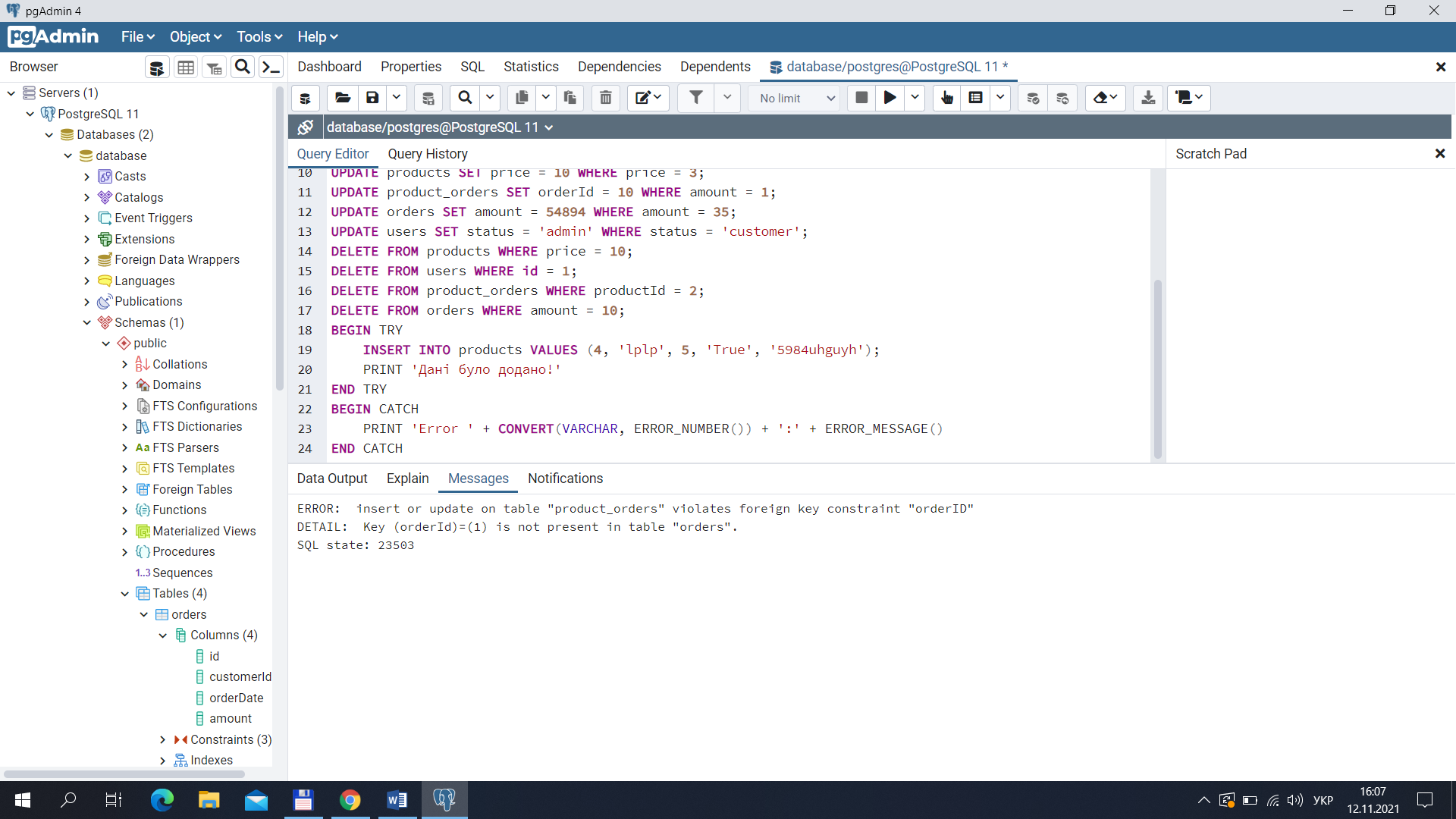
Редагування даних

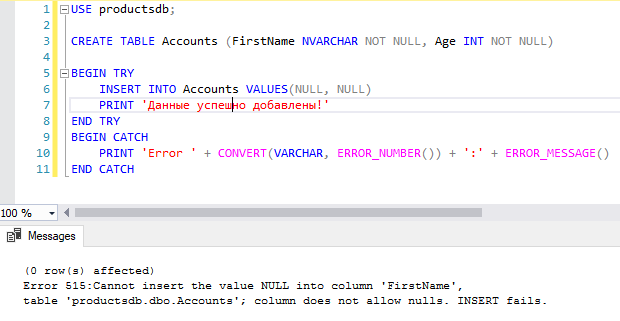


Видалення даних

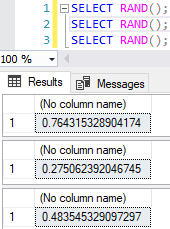


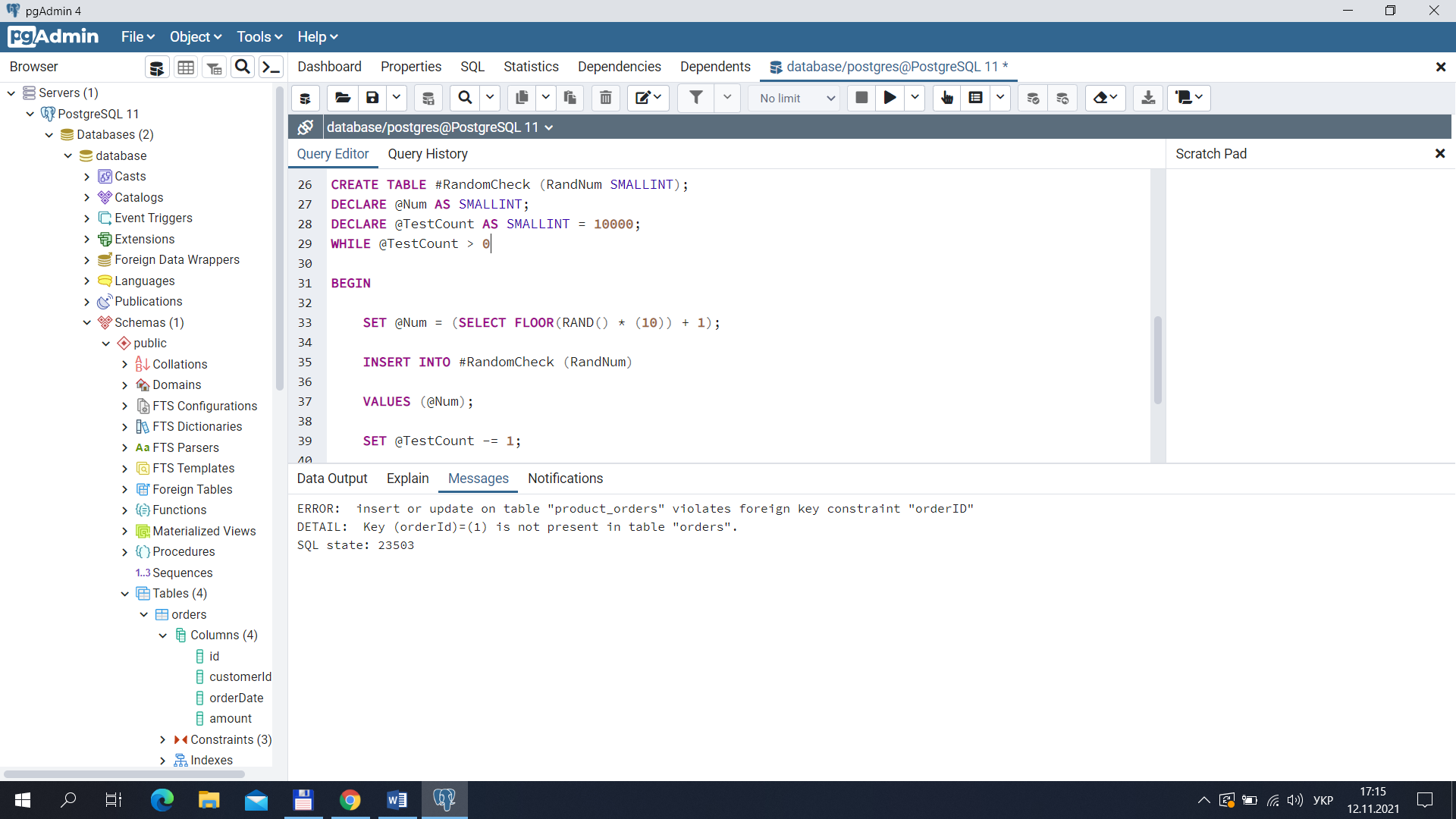
Приклад перехоплення помилок

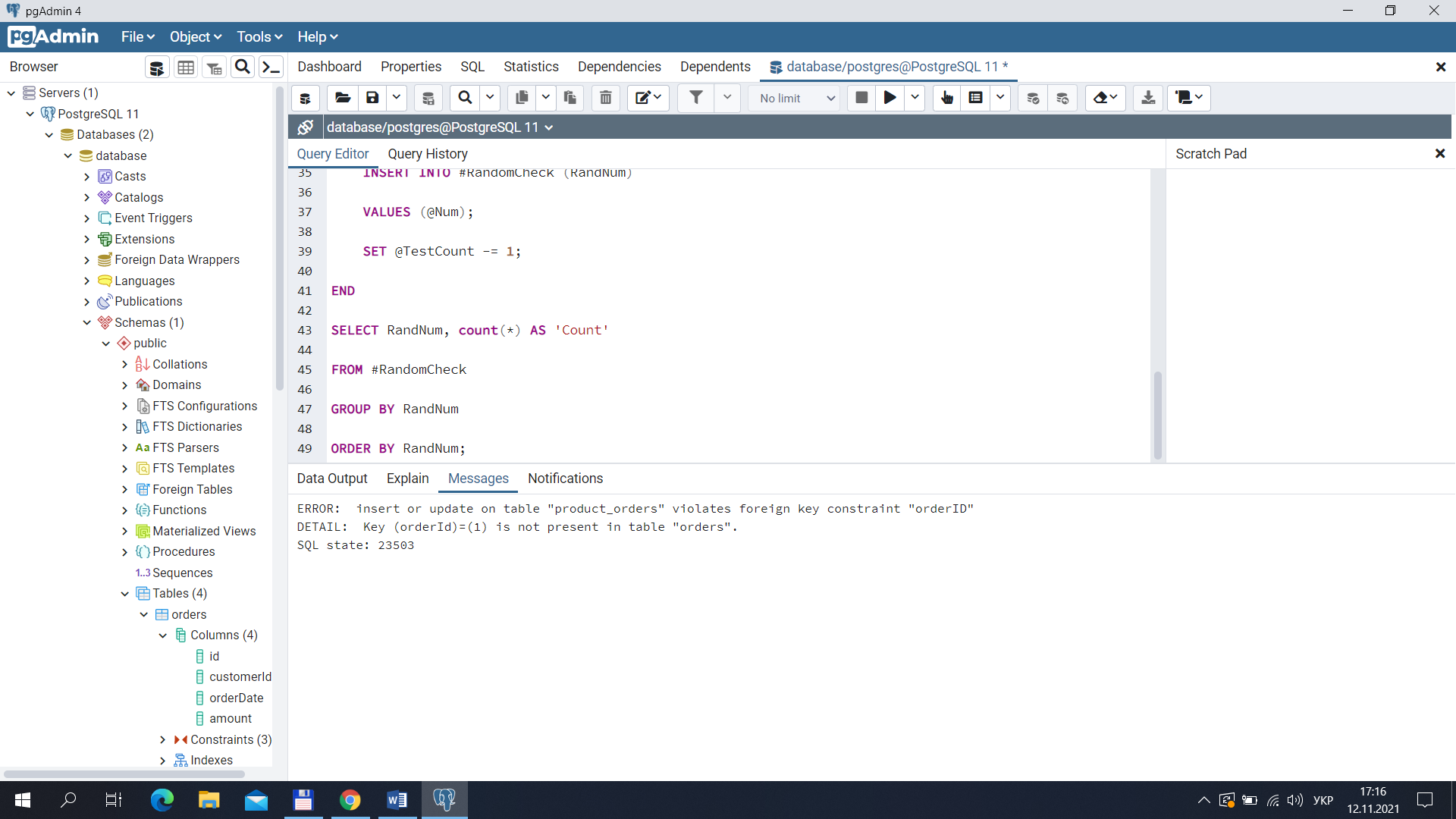


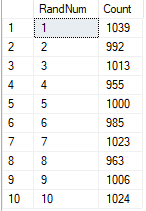


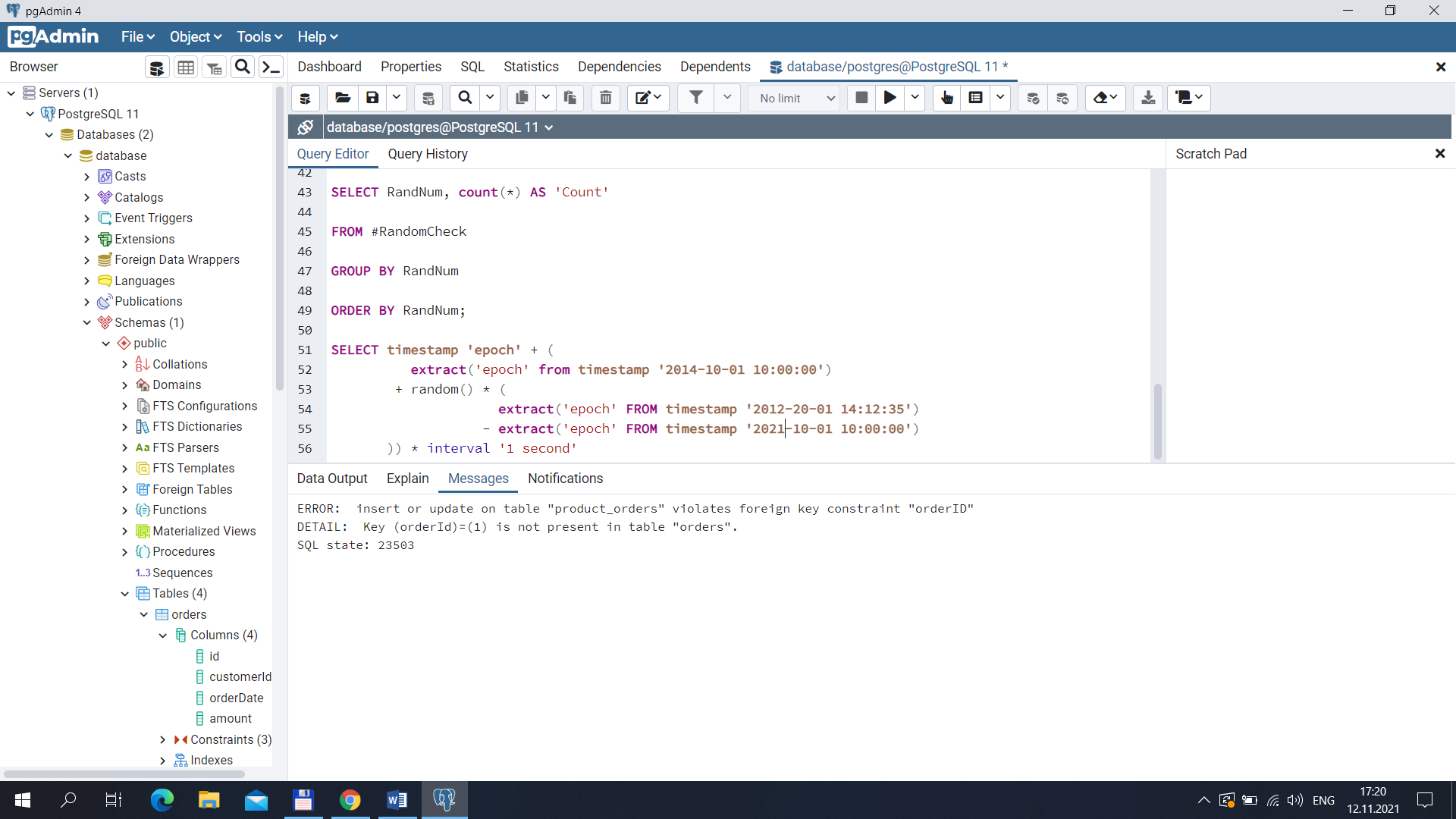
2.Автоматичне генерування даних різних типів у базі



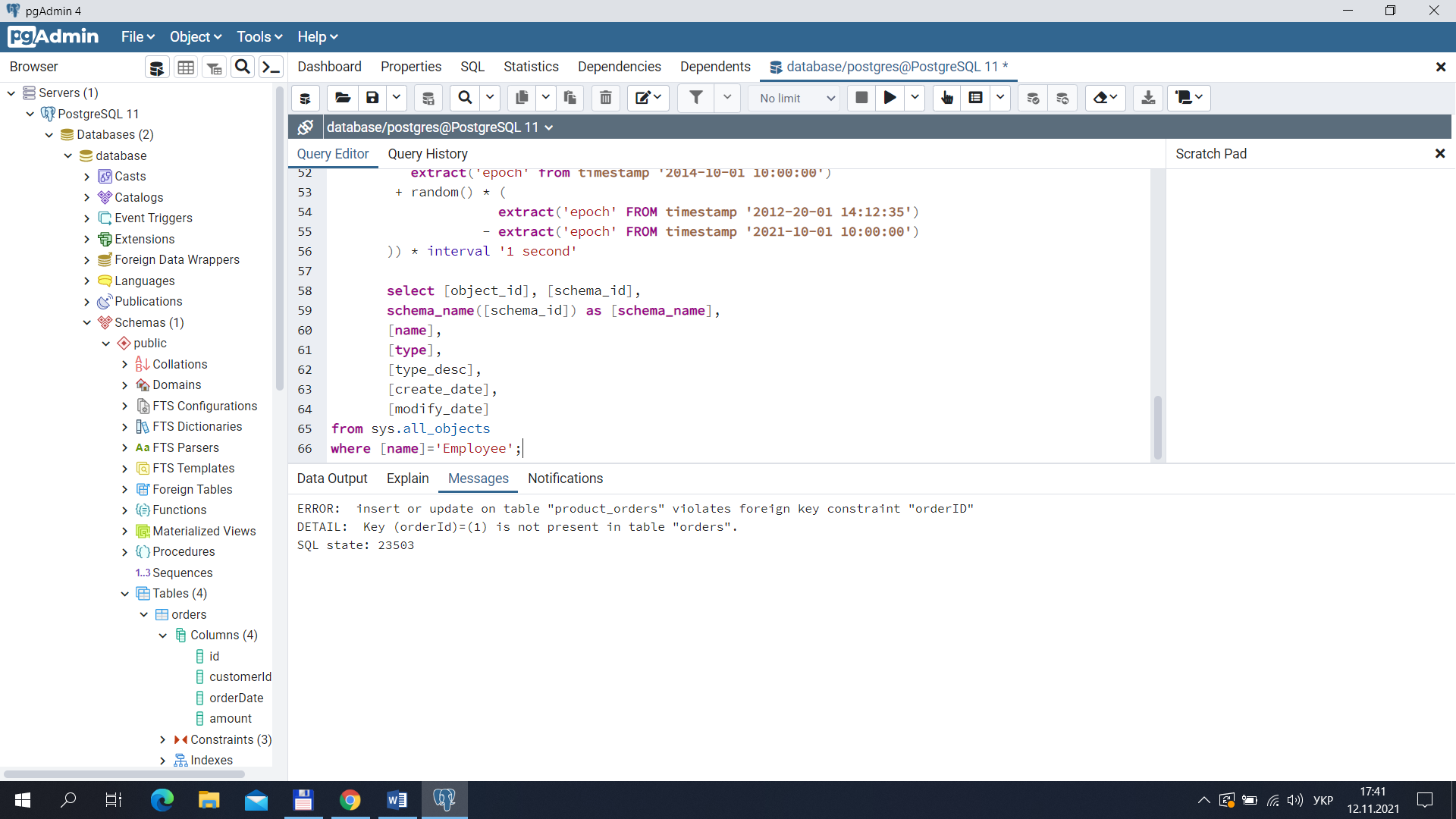




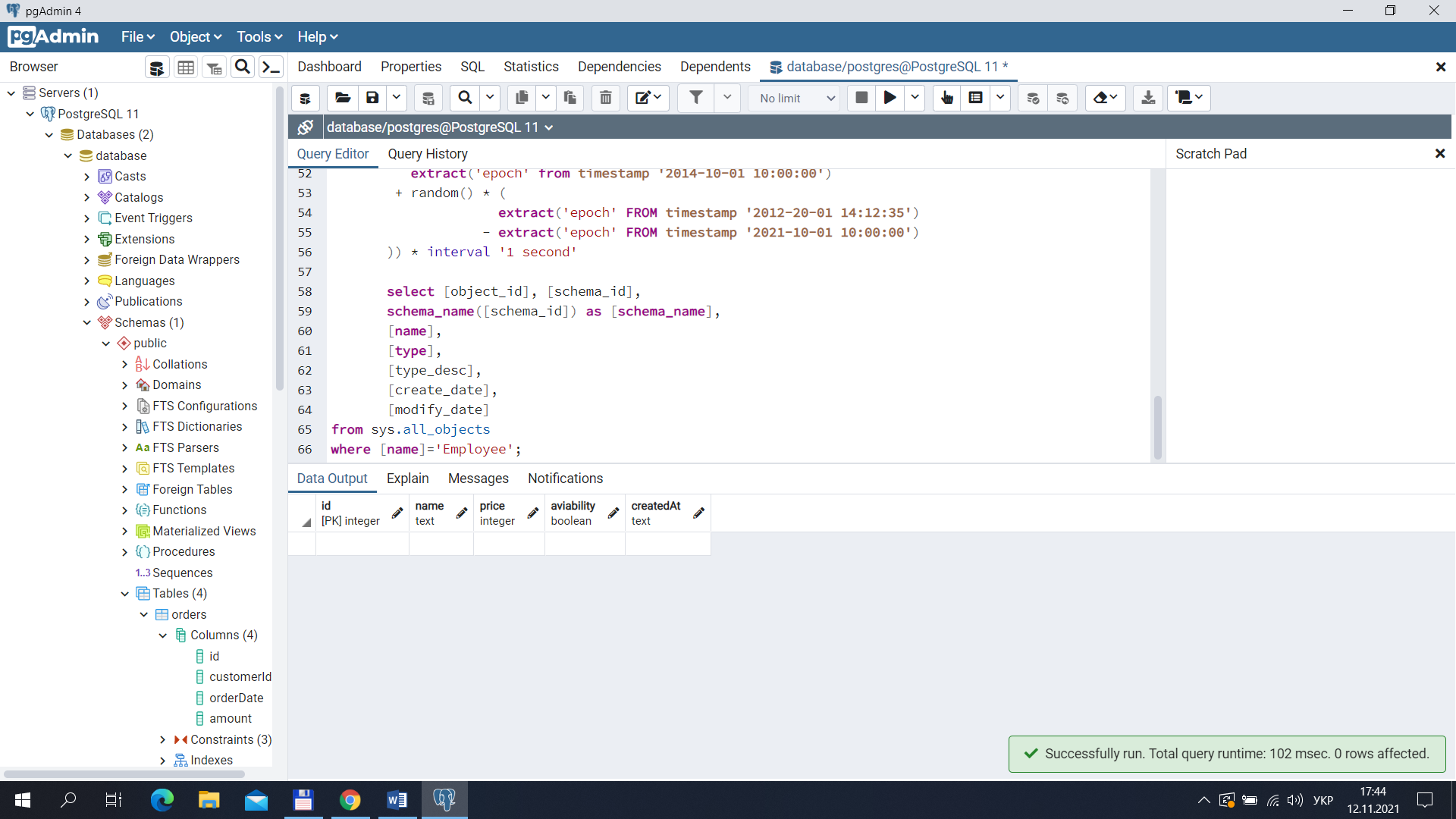


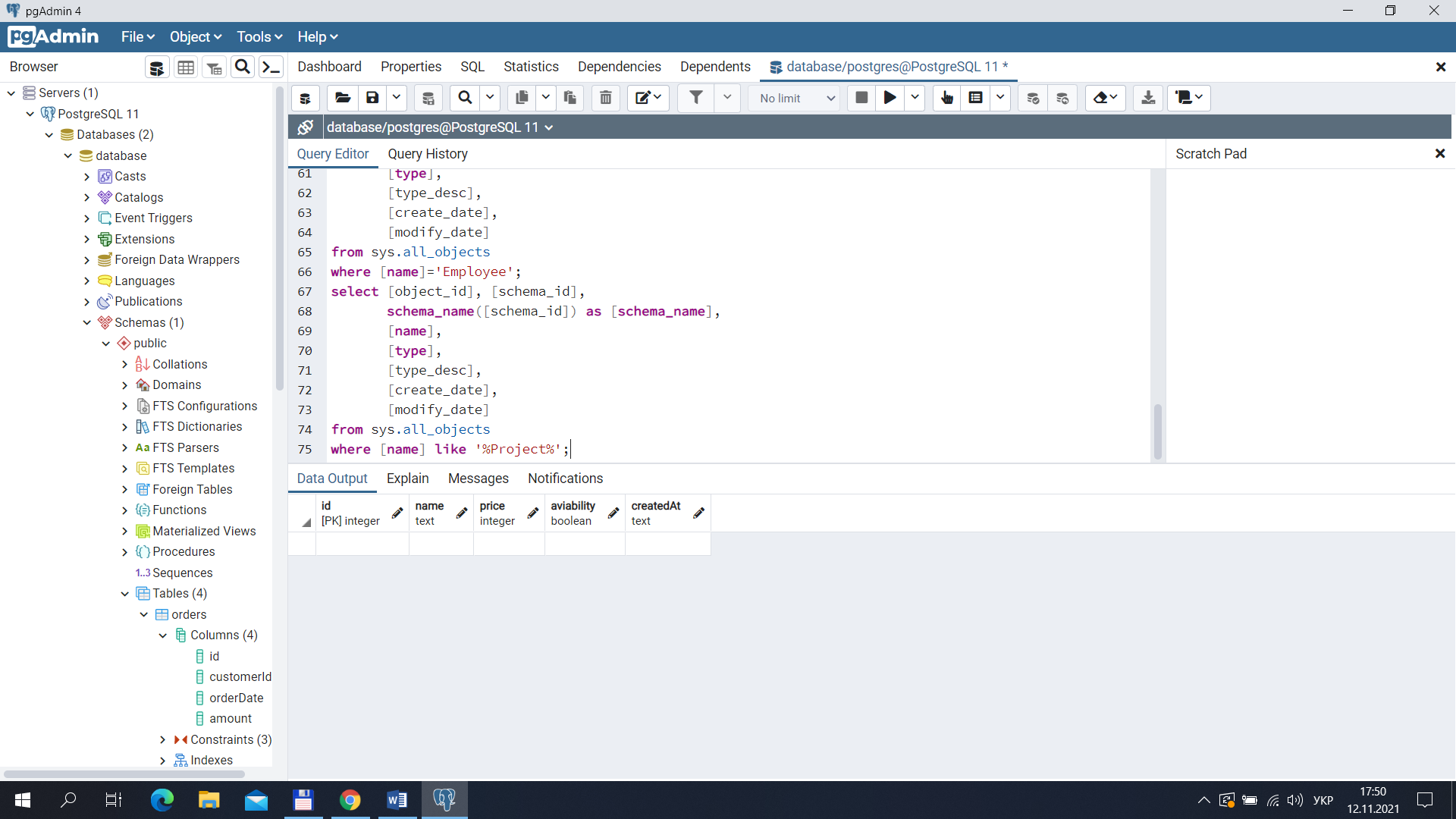


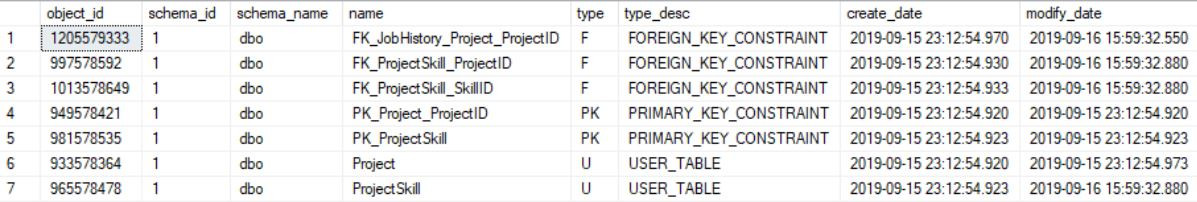
3.Реалізація пошуку

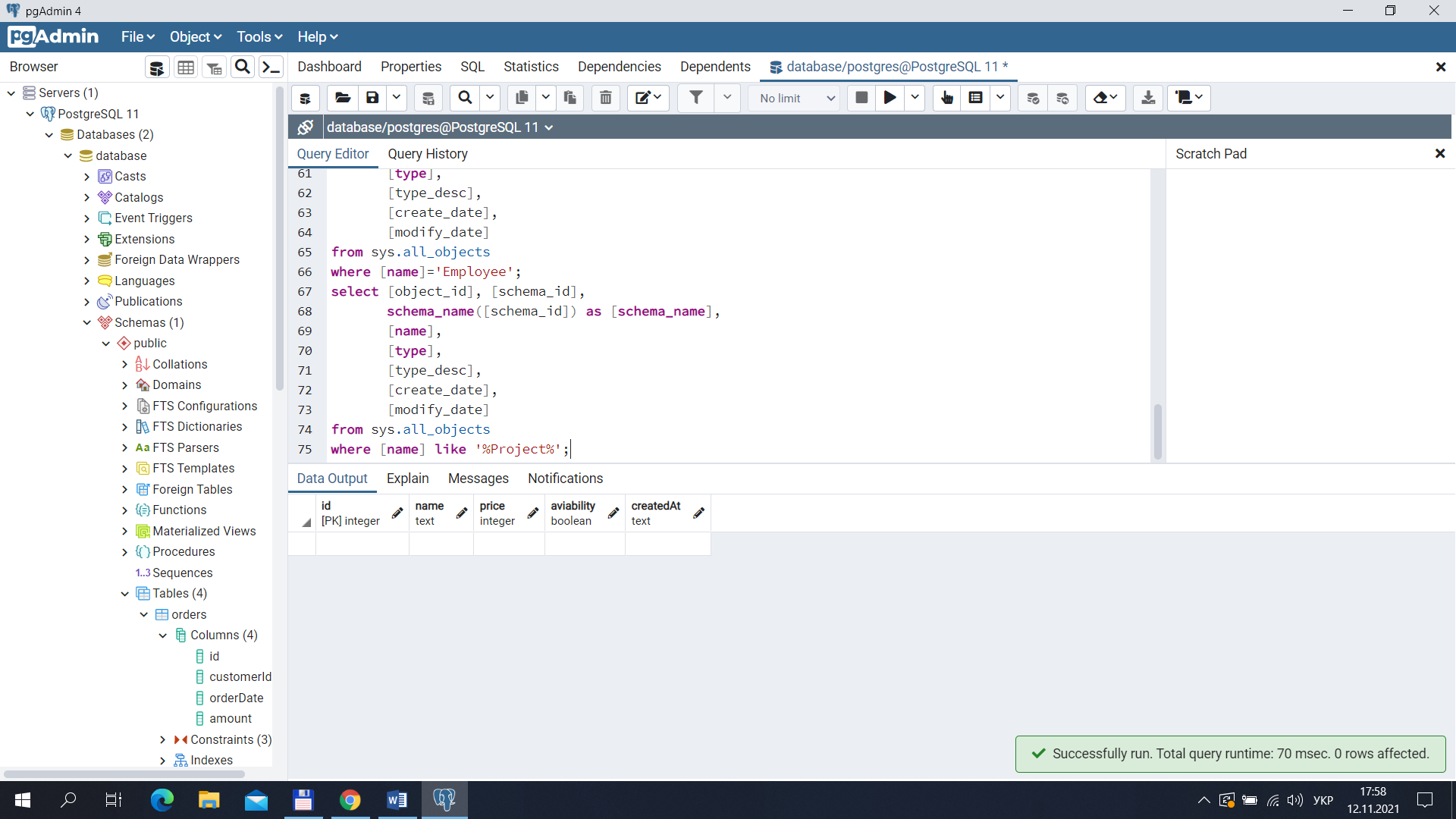












4.Програмний код

|  |
| --- |
| Program1.py |
| class Controller(object):   def \_\_init\_\_(self, model, view):  self.model = model  self.view = view   def show\_items(self, bullet\_points=False):  items = self.model.read\_items()  item\_type = self.model.item\_type  if bullet\_points:  self.view.show\_bullet\_point\_list(item\_type, items)  else:  self.view.show\_number\_point\_list(item\_type, items)   def show\_item(self, item\_name):  try:  item = self.model.read\_item(item\_name)  item\_type = self.model.item\_type  self.view.show\_item(item\_type, item\_name, item)  except mvc\_exc.ItemNotStored as e:  self.view.display\_missing\_item\_error(item\_name, e)   def insert\_item(self, name, price, quantity):  assert price > 0, 'price must be greater than 0'  assert quantity >= 0, 'quantity must be greater than or equal to 0'  item\_type = self.model.item\_type  try:  self.model.create\_item(name, price, quantity)  self.view.display\_item\_stored(name, item\_type)  except mvc\_exc.ItemAlreadyStored as e:  self.view.display\_item\_already\_stored\_error(name, item\_type, e)   def update\_item(self, name, price, quantity):  assert price > 0, 'price must be greater than 0'  assert quantity >= 0, 'quantity must be greater than or equal to 0'  item\_type = self.model.item\_type   try:  older = self.model.read\_item(name)  self.model.update\_item(name, price, quantity)  self.view.display\_item\_updated(  name, older['price'], older['quantity'], price, quantity)  except mvc\_exc.ItemNotStored as e:  self.view.display\_item\_not\_yet\_stored\_error(name, item\_type, e)  # if the item is not yet stored and we performed an update, we have  # 2 options: do nothing or call insert\_item to add it.  # self.insert\_item(name, price, quantity)   def update\_item\_type(self, new\_item\_type):  old\_item\_type = self.model.item\_type  self.model.item\_type = new\_item\_type  self.view.display\_change\_item\_type(old\_item\_type, new\_item\_type)   def delete\_item(self, name):  item\_type = self.model.item\_type  try:  self.model.delete\_item(name)  self.view.display\_item\_deletion(name)  except mvc\_exc.ItemNotStored as e:  self.view.display\_item\_not\_yet\_stored\_error(name, item\_type, e)  def update\_item(name, price, quantity):  global items  # Python 3.x removed tuple parameters unpacking (PEP 3113), so we have to do it manually (i\_x is a tuple, idxs\_items is a list of tuples)  idxs\_items = list(  filter(lambda i\_x: i\_x[1]['name'] == name, enumerate(items)))  if idxs\_items:  i, item\_to\_update = idxs\_items[0][0], idxs\_items[0][1]  items[i] = {'name': name, 'price': price, 'quantity': quantity}  else:  raise mvc\_exc.ItemNotStored(  'Can\'t update "{}" because it\'s not stored'.format(name))   def delete\_item(name):  global items  # Python 3.x removed tuple parameters unpacking (PEP 3113), so we have to do it manually (i\_x is a tuple, idxs\_items is a list of tuples)  idxs\_items = list(  filter(lambda i\_x: i\_x[1]['name'] == name, enumerate(items)))  if idxs\_items:  i, item\_to\_delete = idxs\_items[0][0], idxs\_items[0][1]  del items[i]  else:  raise mvc\_exc.ItemNotStored(  'Can\'t delete "{}" because it\'s not stored'.format(name)) |

**Висновки**

Виконавши дану лабораторну роботу ми навчилися програмувати прикладні додатки баз даних PostgreSQL. Ми реалізували функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, засобами консольного інтерфейсу, передбачили автоматичне пакетне генерування «рандомізованих» даних у базі, забезпечили реалізацію пошуку