

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра програмного забезпечення та комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

Виконала: студентка ІII курсу

ФПМ групи КП-72

Городченко Анна Володимирівна

Перевірив:

Петрашенко А.В.

Київ – 2019

**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Мета роботи:* здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних

*Завдання* роботи полягає у наступному:

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

*Функціональні вимоги:*

1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

*Вимоги до інтерфейсу користувача:*

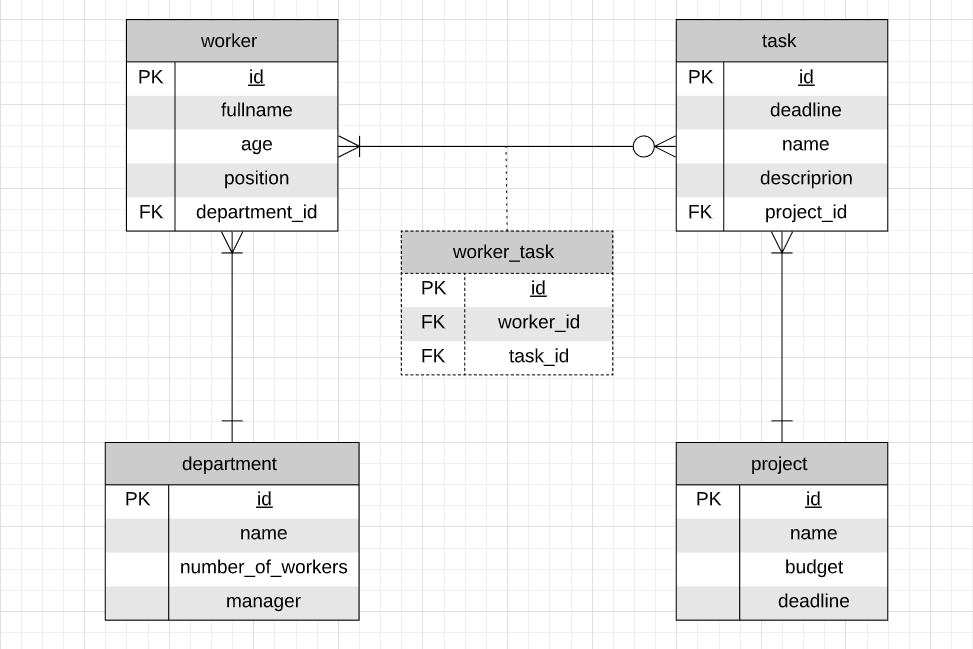
1. Використовувати консольний інтерфейс користувача.

**Варіант 8**

Пошук за атрибутами має відбуватися по переліченню строк та логічному типу.

Повнотекстовий пошук за цілою фразою або для випадку, коли вказане слово не входить в документ.

**Нормалізована модель даних**

****

Таблиця Department знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Department знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Department знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Worker знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Worker знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Worker знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Task знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Task знаходиться у 2 НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Task знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Project знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Project знаходиться у 2 НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинного ключа Id. Таблиця Project знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

Таблиця Worker\_Task знаходиться у 1 нормальній формі , бо всі поля мають тільки атомарні значення. Таблиця Worker\_Task знаходиться у 2НФ, бо знаходиться у 1НФ та кожний її неключовий атрибут функціонально повністю залежить від первинних ключів worker\_id, task\_id. Таблиця Worker\_Task знаходиться у 3НФ , бо знаходиться у 2НФ та кожний її атрибут нетранзитивно залежить від первинного ключа.

**Опис програми**

Програма створена за патерном MVC (Model-View-Controller). Складається відповідно з модулів model , view та controller.

У модулі model реалізовані функції , що здійснюють SQL запити до Бази Даних.

У модулі view реалізовані функції виводу даних з таблиць.

У класі Controller реалізовані функції для відповідних меню та допоміжні функції.

**Опис структури меню програми**

 Меню програми можна розглядати як її концептуальну модель

**Лістинг програми**

from controller import Controller

Controller().show\_init\_menu()

**Model**

import psycopg2

class **Model**:

def **\_\_init\_\_**(self):

try:

self.connection = psycopg2.connect(host="localhost", port="5432",

database='db\_lab2', user='goroanya', password='goroanya99')

self.cursor = self.connection.cursor()

except (Exception, psycopg2.Error) as error:

**print**("Помилка при з'єднанні з PostgreSQL", error)

def **get\_col\_names**(self):

return [d[0] for d in self.cursor.description]

def **create\_db**(self):

f = **open**("create\_db.txt", "r")

self.cursor.execute(f.read())

self.connection.commit()

def **get**(self, tname, condition):

try:

query = f'SELECT \* FROM {tname}'

if condition:

query += ' WHERE ' + condition

self.cursor.execute(query)

return self.get\_col\_names(), self.cursor.fetchall()

finally:

self.connection.commit()

def **insert**(self, tname, columns, values):

try:

query = f'INSERT INTO {tname} ({columns}) VALUES ({values});'

self.cursor.execute(query)

finally:

self.connection.commit()

def **delete**(self, tname, condition):

try:

query = f'DELETE FROM {tname} WHERE {condition};'

self.cursor.execute(query)

finally:

self.connection.commit()

def **update**(self, tname, condition, statement):

try:

query = f'UPDATE {tname} SET {statement} WHERE {condition}'

self.cursor.execute(query)

finally:

self.connection.commit()

def **search\_task\_by\_worker\_position**(self, positions):

try:

query = f'''

SELECT \* from task

WHERE id in(

SELECT task\_id FROM worker\_task

JOIN worker on worker\_task.worker\_id=worker.id

WHERE LOWER(position) in ({positions.lower()})

);'''

self.cursor.execute(query)

return self.get\_col\_names(), self.cursor.fetchall()

finally:

self.connection.commit()

def **search\_worker\_by\_task\_is\_done**(self, is\_done):

try:

query = f'''

SELECT \* from worker

WHERE id in(

SELECT worker\_id FROM worker\_task

JOIN task on task.id=worker\_task.task\_id

WHERE isDone={is\_done});'''

self.cursor.execute(query)

return self.get\_col\_names(), self.cursor.fetchall()

finally:

self.connection.commit()

def **fts\_without\_word**(self, word):

query = f'''

select project\_name, task\_name, task\_description from (

select

p.name as project\_name,

task.name as task\_name,

task.description as task\_description,

to\_tsvector(p.name) ||

to\_tsvector(task.name) ||

to\_tsvector(task.description) as document,

to\_tsquery('!{word}') as query

from task

join project p on task.project\_id = p.id

) search

where document @@ query;

'''

try:

self.cursor.execute(query)

return self.get\_col\_names(), self.cursor.fetchall()

finally:

self.connection.commit()

def **fts\_phrase**(self, phrase):

query = f'''

select

ts\_headline(project\_name, query, 'StartSel=\033[94m, StopSel=\033[0m') as project\_name,

ts\_headline(task\_name, query, 'StartSel=\033[94m, StopSel=\033[0m') as task\_name,

ts\_headline(task\_description, query, 'StartSel=\033[94m, StopSel=\033[0m') as task\_description

from (

select

p.name as project\_name,

task.name as task\_name,

task.description as task\_description,

to\_tsvector(p.name) ||

to\_tsvector(task.name) ||

to\_tsvector(task.description) as document,

phraseto\_tsquery('{phrase}') as query

from task

join project p on task.project\_id = p.id

) search

where document @@ query;

'''

try:

self.cursor.execute(query)

return self.get\_col\_names(), self.cursor.fetchall()

finally:

self.connection.commit()

def **fillTaskByRandomData**(self):

sql = """

CREATE OR REPLACE FUNCTION randomDepartments()

RETURNS void AS $$

DECLARE

step integer := 0;

BEGIN

LOOP EXIT WHEN step > 10;

INSERT INTO department (name, number\_of\_workers, manager)

VALUES (

substring(md5(random()::text), 1, 10),

(random() \* (50 - 1) + 1)::integer,

substring(md5(random()::text), 1, 15)

);

step := step + 1;

END LOOP ;

END;

$$ LANGUAGE PLPGSQL;

SELECT randomDepartments();

"""

try:

self.cursor.execute(sql)

finally:

self.connection.commit()

**View**

from consolemenu import \*

from consolemenu.items import \*

class **View**:

def **print**(self, data):

columns, rows = data

lineLen = 30 \* **len**(columns)

self.printSeparator(lineLen)

self.printRow(columns)

self.printSeparator(lineLen)

for row in rows:

self.printRow(row)

self.printSeparator(lineLen)

def **printRow**(self, row):

for col in row:

**print**(str(col).rjust(26, ' ') + ' |', end='')

**print**('')

def **printSeparator**(self, length):

**print**('-' \* length)

Controller

from consolemenu import SelectionMenu

from model import Model

from view import View

TABLES\_NAMES = ['task', 'worker', 'project', 'department', 'worker\_task']

TABLES = {

'task': ['id', 'deadline', 'name', 'description', 'project\_id', 'isdone'],

'worker': ['id', 'fullname', 'age', 'position', 'department\_id'],

'department': ['id', 'name', 'number\_of\_workers', 'manager'],

'project': ['id', 'name', 'budget', 'deadline'],

'worker\_task': ['id', 'worker\_id', 'task\_id']

}

def getInput(msg, tableName=''):

print(msg)

if tableName:

print(' | '.join(TABLES[tableName]), end='\n\n')

return input()

def getInsertInput(msg, tableName):

print(msg)

print(' | '.join(TABLES[tableName]), end='\n\n')

return input(), input()

def pressEnter():

input()

class Controller:

def \_\_init\_\_(self):

*self*.model = Model()

*self*.view = View()

def show\_init\_menu(self, msg=''):

selectionMenu = SelectionMenu(

TABLES\_NAMES + ['Find text where word is not included', 'Find text by full phrase',

'Fill table "department" by random data (10 items)'], title='Select the table to work with | command:', subtitle=msg)

selectionMenu.show()

index = selectionMenu.selected\_option

if index < len(TABLES\_NAMES):

tableName = TABLES\_NAMES[index]

*self*.show\_entity\_menu(tableName)

elif index == 5:

*self*.fts\_without\_word()

elif index == 6:

*self*.fts\_phrase()

elif index == 7:

*self*.fillByRandom()

else:

print('Bye, have a beautiful day!')

def show\_entity\_menu(self, tableName, msg=''):

options = ['Get', 'Delete', 'Update', 'Insert']

functions = [*self*.get, *self*.delete, *self*.update, *self*.insert]

if tableName == 'task':

options.append('Search task by worker positions')

functions.append(*self*.search\_task\_by\_worker\_position)

elif tableName == 'worker':

options.append('Search worker by his task is done')

functions.append(*self*.search\_worker\_by\_task\_is\_done)

selectionMenu = SelectionMenu(options, f'Name of table: {tableName}',

exit\_option\_text='Back', subtitle=msg)

selectionMenu.show()

try:

function = functions[selectionMenu.selected\_option]

function(tableName)

except IndexError:

*self*.show\_init\_menu()

def get(self, tableName):

try:

condition = getInput(

f'GET {tableName}\nEnter condition (SQL) or leave empty:', tableName)

data = *self*.model.get(tableName, condition)

*self*.view.print(data)

pressEnter()

*self*.show\_entity\_menu(tableName)

except Exception as err:

*self*.show\_entity\_menu(tableName, str(err))

def insert(self, tableName):

try:

columns, values = getInsertInput(

f"INSERT {tableName}\nEnter colums divided with commas, then do the same for values in format: ['value1', 'value2', ...]", tableName)

*self*.model.insert(tableName, columns, values)

*self*.show\_entity\_menu(tableName, 'Insert is successful!')

except Exception as err:

*self*.show\_entity\_menu(tableName, str(err))

def delete(self, tableName):

try:

condition = getInput(

f'DELETE {tableName}\n Enter condition (SQL):', tableName)

*self*.model.delete(tableName, condition)

*self*.show\_entity\_menu(tableName, 'Delete is successful')

except Exception as err:

*self*.show\_entity\_menu(tableName, str(err))

def update(self, tableName):

try:

condition = getInput(

f'UPDATE {tableName}\nEnter condition (SQL):', tableName)

statement = getInput(

"Enter SQL statement in format [<key>='<value>']", tableName)

*self*.model.update(tableName, condition, statement)

*self*.show\_entity\_menu(tableName, 'Update is successful')

except Exception as err:

*self*.show\_entity\_menu(tableName, str(err))

def search\_task\_by\_worker\_position(self, tableName):

try:

positions = getInput(

'Search task where worker\'s position are: \nEnter positions divided with commas:')

data = *self*.model.search\_task\_by\_worker\_position(positions)

*self*.view.print(data)

pressEnter()

*self*.show\_entity\_menu(tableName)

except Exception as err:

*self*.show\_entity\_menu(tableName, str(err))

def search\_worker\_by\_task\_is\_done(self, tableName):

try:

is\_done = getInput('Search workers that have done them tasks.\nIs task done?:').lower() in [

'true', 't', 'yes', 'y', '+']

data = *self*.model.search\_worker\_by\_task\_is\_done(is\_done)

*self*.view.print(data)

pressEnter()

*self*.show\_entity\_menu(tableName)

except Exception as err:

*self*.show\_entity\_menu(tableName, str(err))

def fts\_without\_word(self):

try:

word = getInput('Enter word:')

data = *self*.model.fts\_without\_word(word)

*self*.view.print(data)

pressEnter()

*self*.show\_init\_menu()

except Exception as err:

*self*.show\_init\_menu(str(err))

def fts\_phrase(self):

try:

phrase = getInput('Enter phrase:')

data = *self*.model.fts\_phrase(phrase)

*self*.view.print(data)

pressEnter()

*self*.show\_init\_menu()

except Exception as err:

*self*.show\_init\_menu(str(err))

def fillByRandom(self):

try:

*self*.model.fillTaskByRandomData()

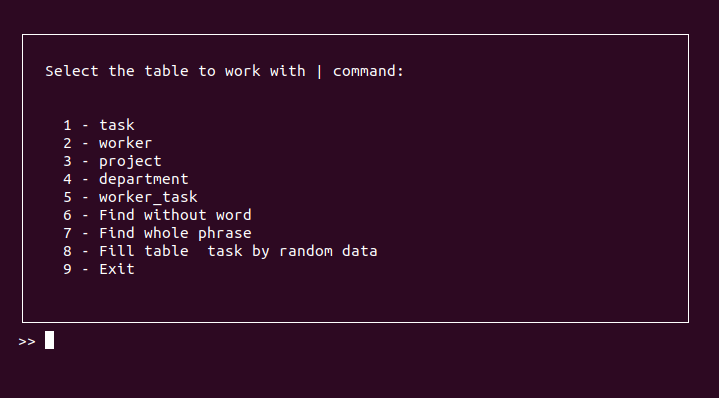
*self*.show\_init\_menu('Generated successfully')

except Exception as err:

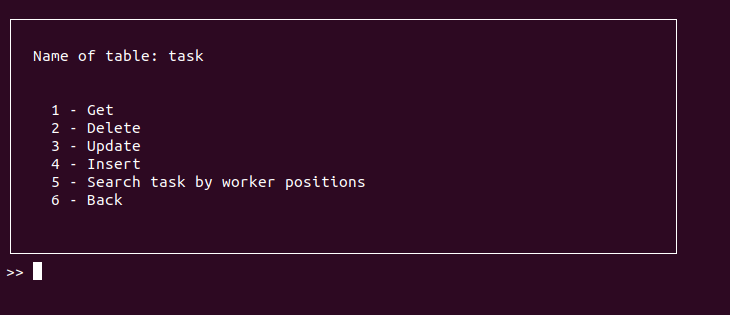
*self*.show\_init\_menu(str(err))

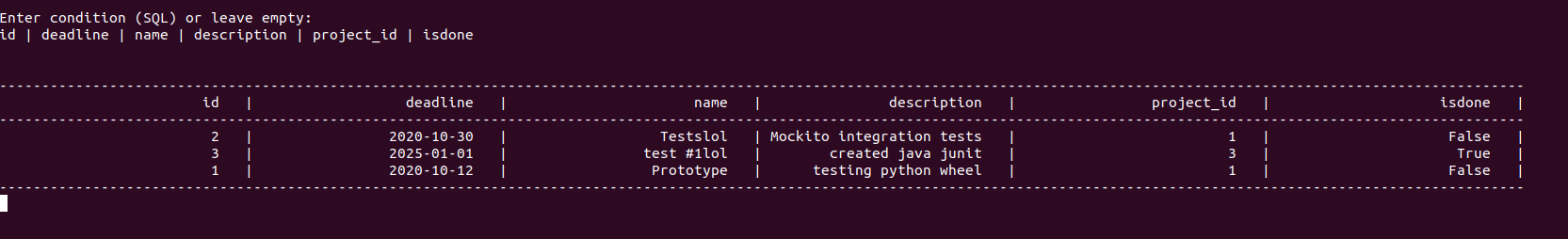
**Скріншот результатів виконання операції вилучення**

**INITIAL MENU**

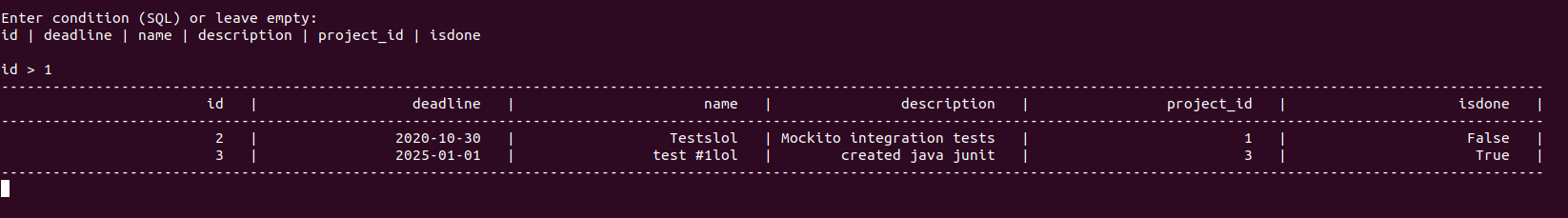
**ENTITY MENU**

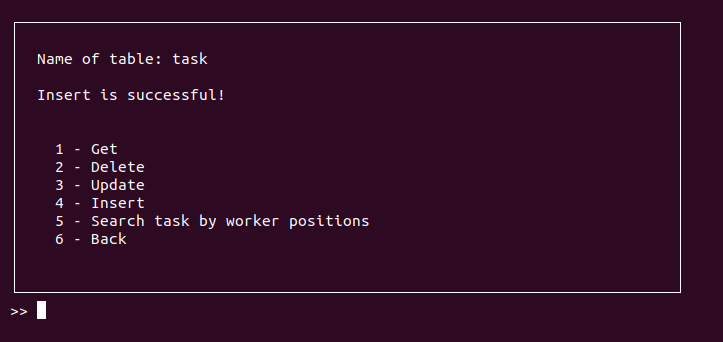
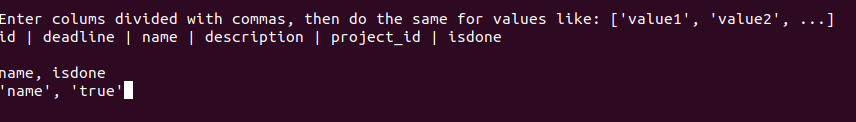
**(task)**

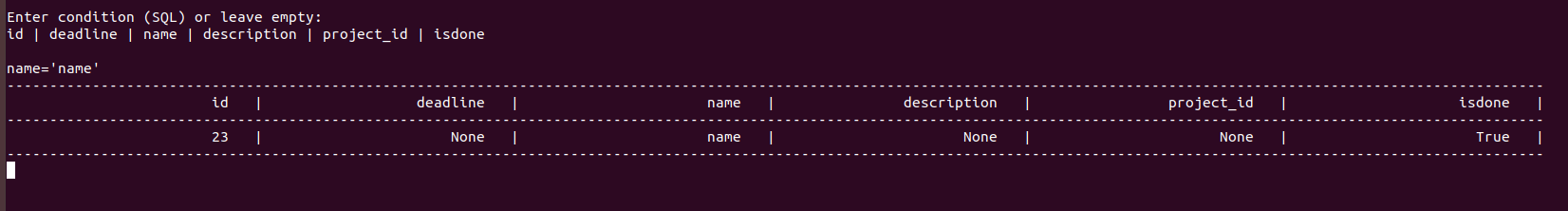
**GET**

**(task)**

**(task де id >1)**

**INSERT**

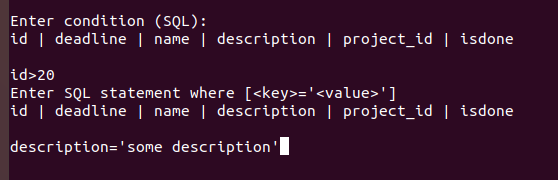
**Операція вставки успішна.**

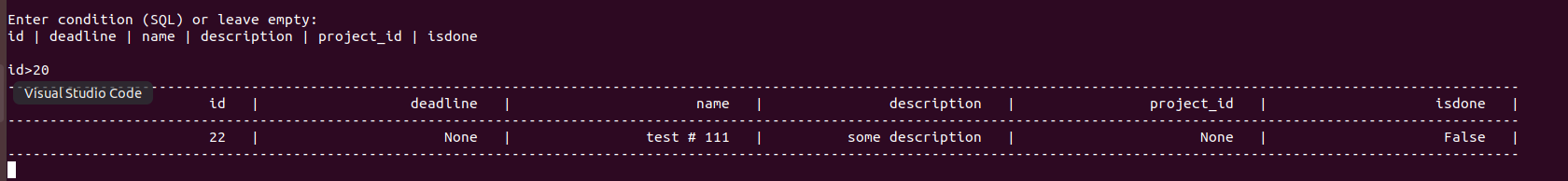
****

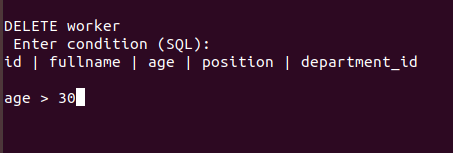
**Новий вставлений запис.**

**UPDATE**

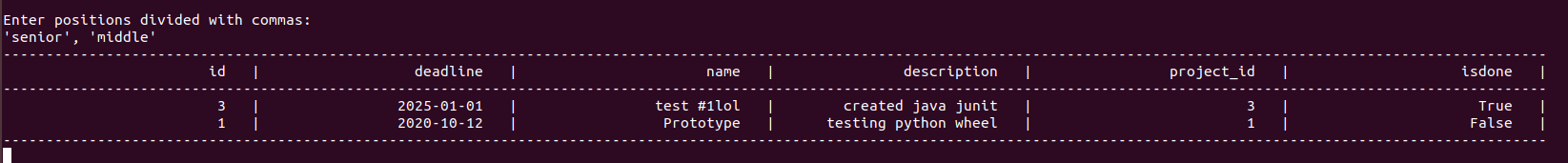
**(task де id > 20)**

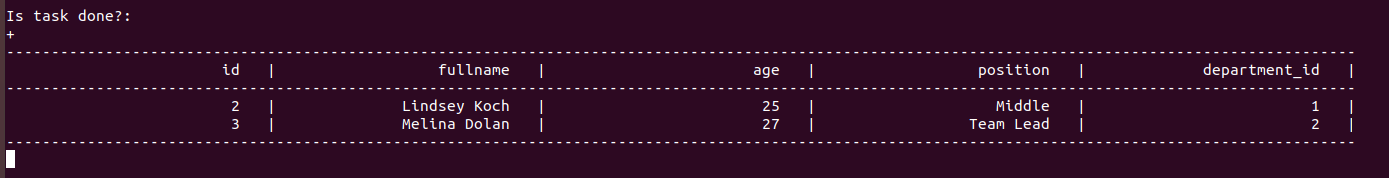
****

**DELETE**

****

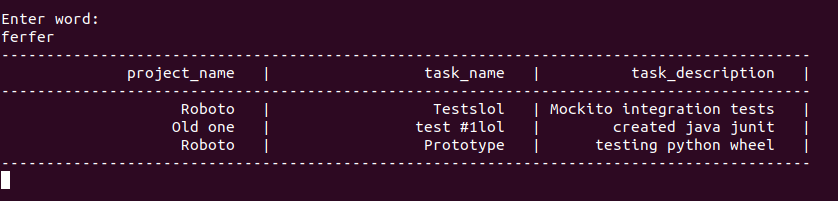
**Знайти всі завдання, де працівник, який його виконує ,має посаду з перелічення:**

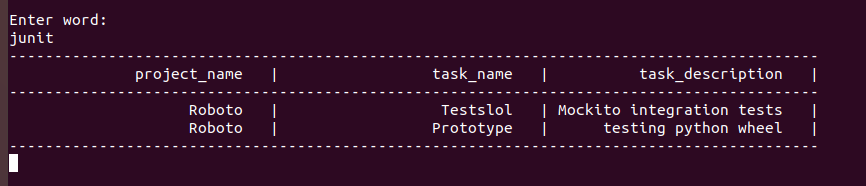
**Знайти всіх працівників,які виконали свої завдання:**

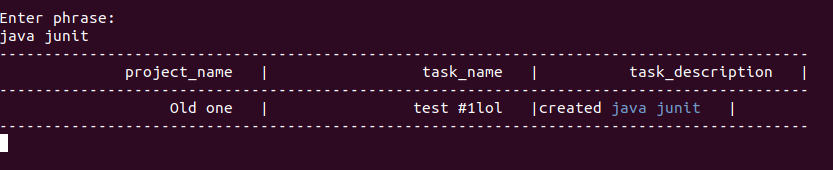
****

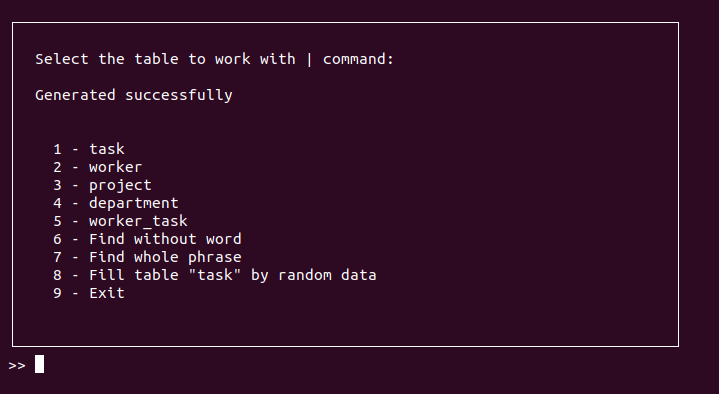
**Повнотектовий пошук**

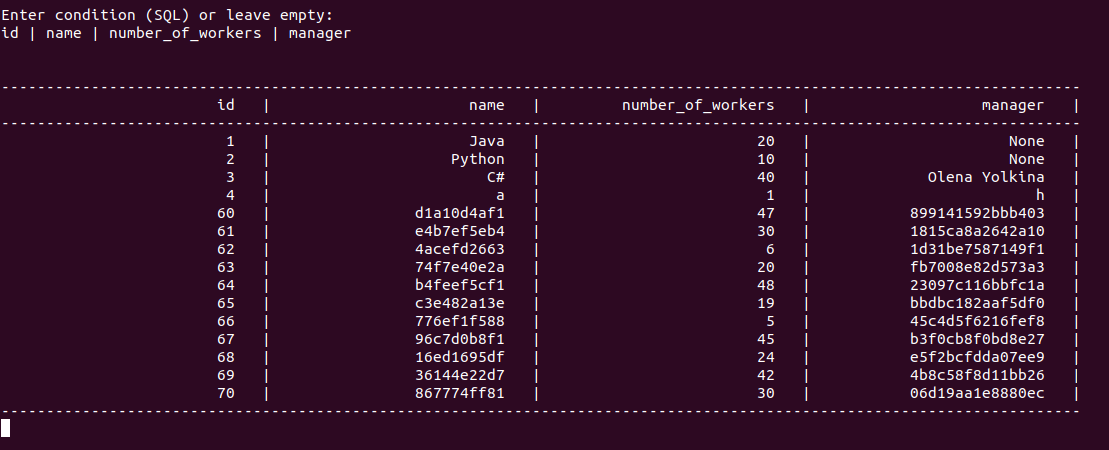
**1) Слово яке не входить**

****

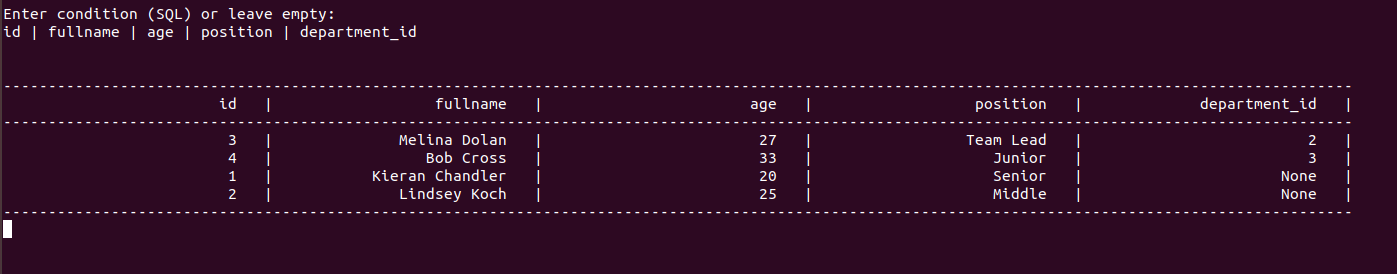
**2) Повна фраза**

****

**Заповнення таблиці рандомізованими значеннями**

****

**Видалення таблиці department , на яку посилається таблиця worker**

**Піля видалення видалення department з id = 1 , у таблиці worker у 1 та 2 працівника department\_id став null. Так сталось тому,що при створенні таблиці було вказано FOREIGN KEY ... ON DELETE SET NULL**