

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:*

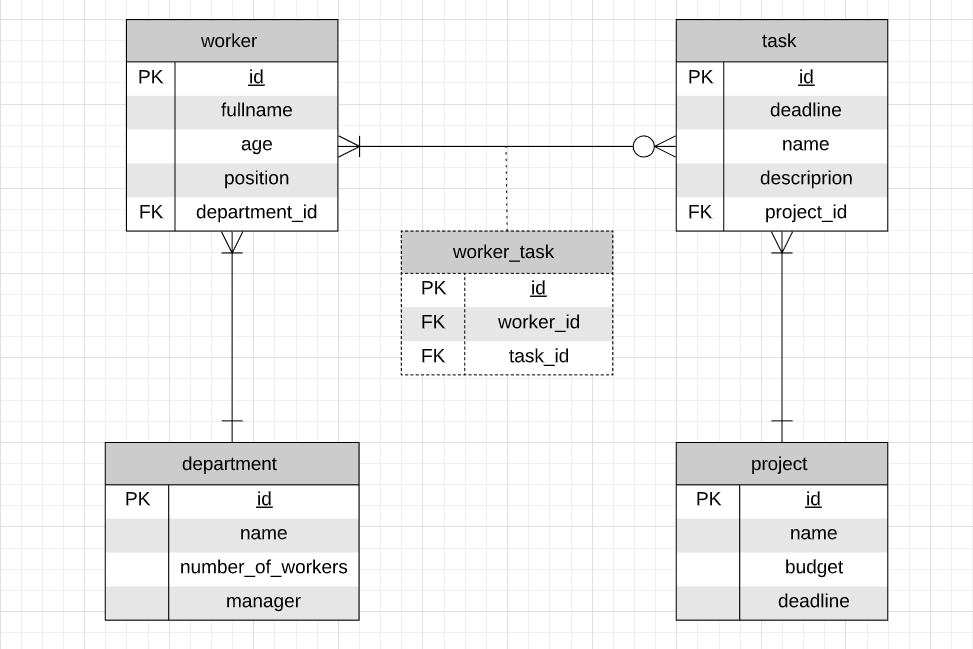
1. Використовувати консольний інтерфейс користувача.

**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()

**Controller**

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)

**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:

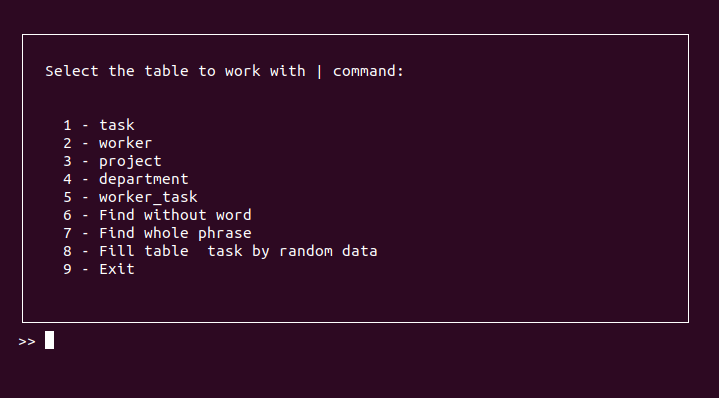
self.cursor.execute(sql)

finally:

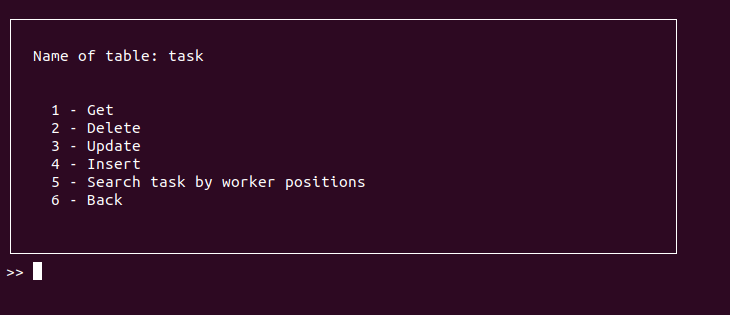
self.connection.commit()

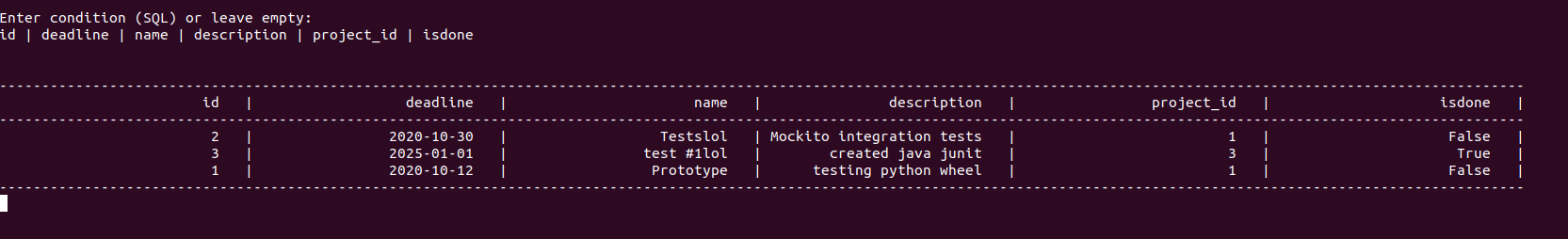
**Скріншот результатів виконання операції вилучення**

**INITIAL MENU**

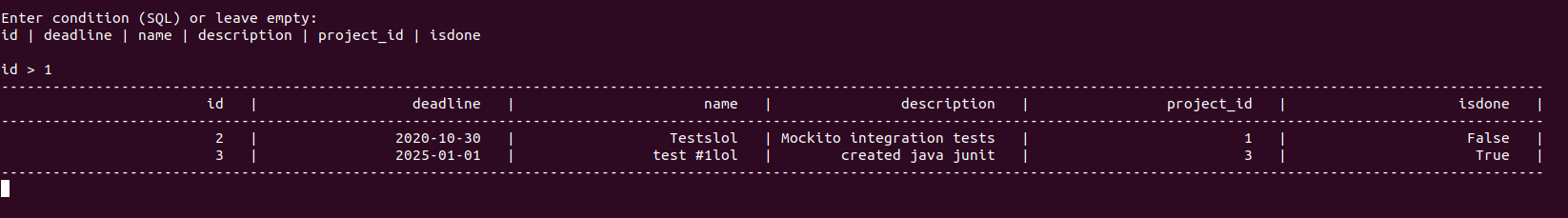
**ENTITY MENU**

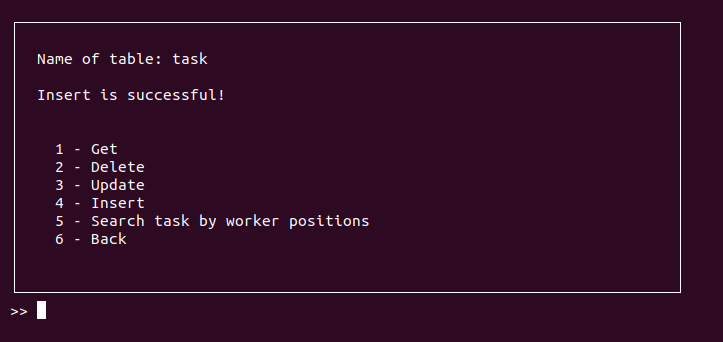
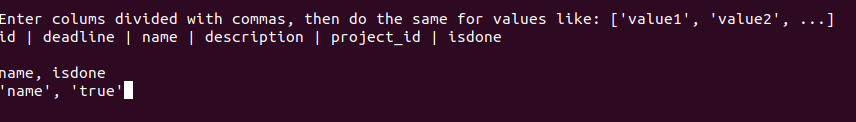
**(task)**

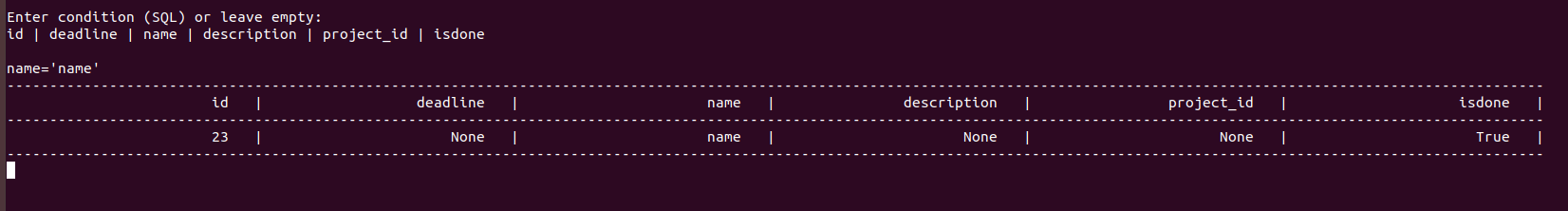
**GET**

**(task)**

**(task де id >1)**

**INSERT**

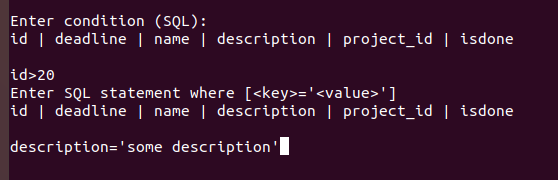
**Операція вставки успішна.**

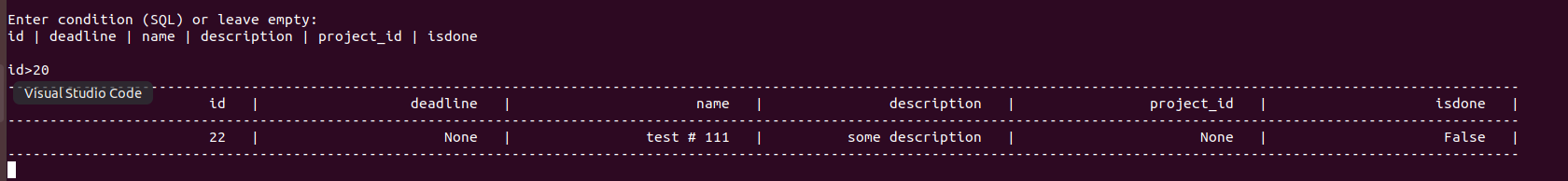
****

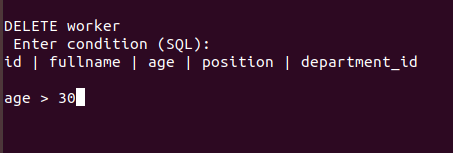
**Новий вставлений запис.**

**UPDATE**

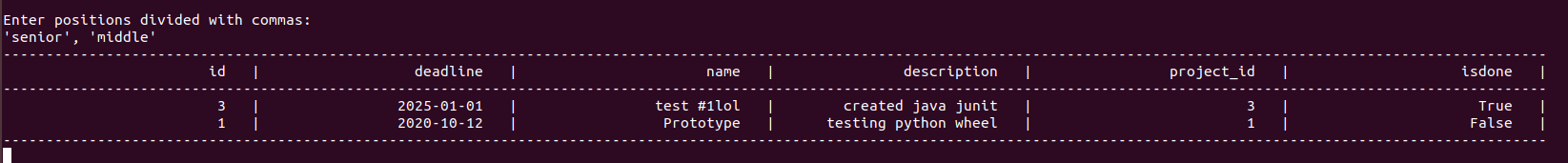
**(task де id > 20)**

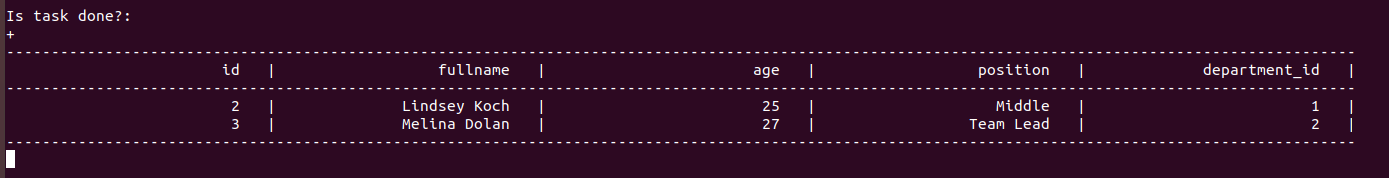
****

**DELETE**

****

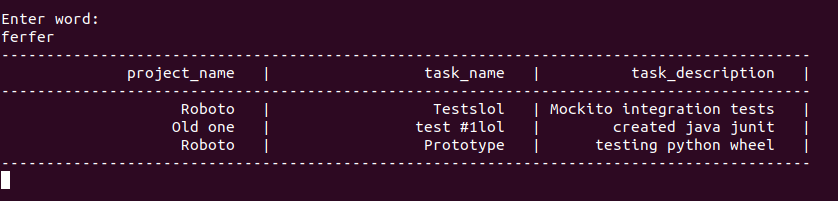
**Знайти всі завдання, де працівник, який його виконує ,має посаду з перелічення:**

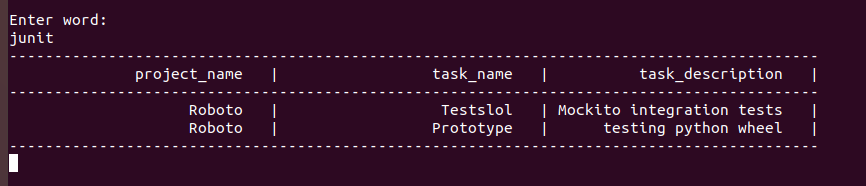
**Знайти всіх працівників,які виконали свої завдання:**

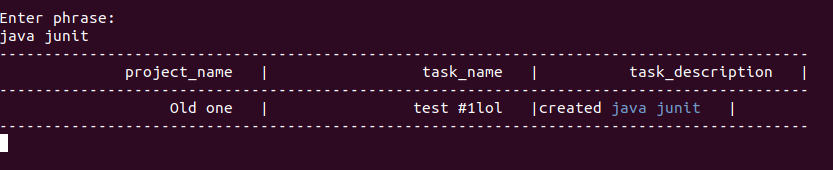
****

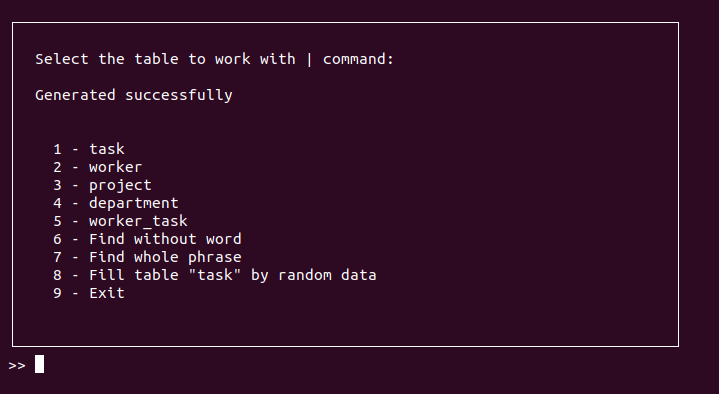
**Повнотектовий пошук**

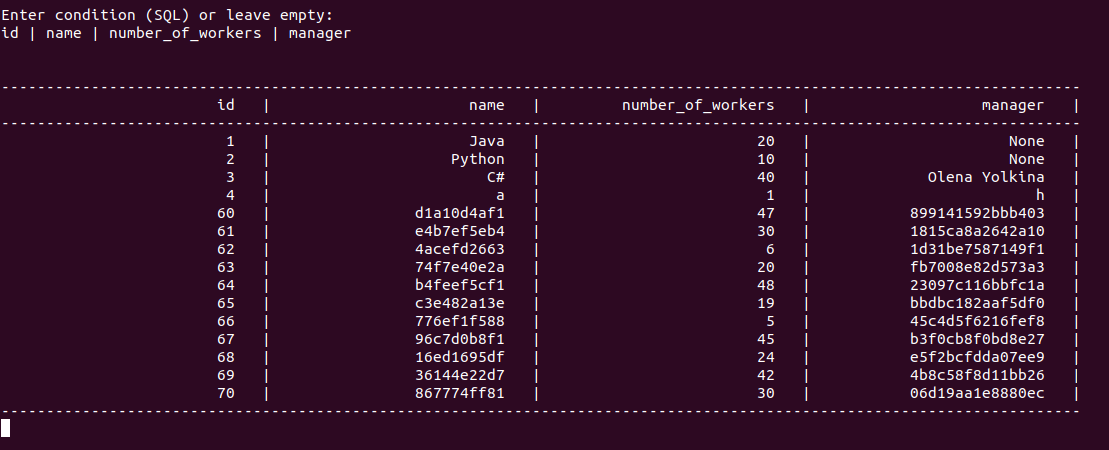
**1) Слово яке не входить**

****

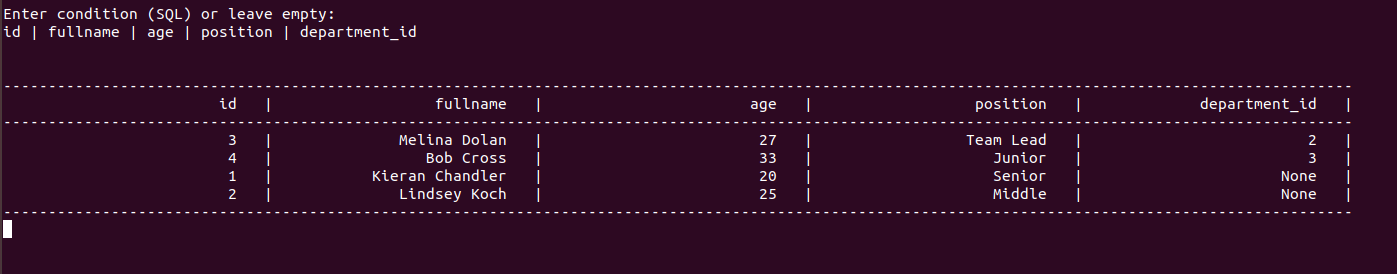
**2) Повна фраза**

****

**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**