Redovisning av Python-kod

1. Översikt

Den här Python-koden är utvecklad för att interagera med en SQL Server-databas som hanterar en bokhandel. Applikationen möjliggör för användaren att söka efter boktitlar och visa lagerstatus för böcker i olika butiker. Användaren kan utföra både exakt och wildcard-baserad sökning för att hitta relevanta boktitlar.

2. Funktionalitet

Koden består av flera funktioner som tillsammans hanterar anslutningen till databasen, användarens interaktion och visningen av sökresultat:

- **Skapa anslutning till databasen (`create_engine_connection`)**:
  Koden läser anslutningsuppgifter från en `.ini`-fil och skapar en anslutning till SQL Server-databasen. Detta gör det möjligt för applikationen att kommunicera med databasen för att hämta och manipulera data.
- **Hämta användarinput (`get_user_input`)**:
  Användaren uppmanas att ange en boktitel att söka efter och kan välja mellan exakt matchning eller wildcard-baserad sökning. Funktionen kan också ge hjälpmeddelanden om hur man använder wildcard-sökning.
- **Utföra sökfrågan (`execute_search_query`)**:
  Denna funktion skapar och kör en SQL-fråga baserat på användarens input. Frågan söker efter boktitlar i databasen och hämtar information om titeln, författaren, språket, priset och lagerstatusen i olika butiker.
- **Visa resultat (`print_books`)**:
  Sökeresultaten visas i en tabellform med hjälp av `PrettyTable`, vilket gör det lätt för användaren att läsa och förstå informationen.
- **Huvudfunktion (`main`)**:
  Funktionen hanterar programmets huvudflöde. Den initierar anslutningen till databasen, tar emot användarens input, utför sökningen och presenterar resultaten. Efter varje sökning får användaren möjlighet att utföra ytterligare sökningar eller avsluta programmet.

3. Kodstruktur

```
from sqlalchemy import create_engine, text
import pyodbc
from prettytable import PrettyTable
import configparser

def create_engine_connection():
    # Laddar konfigurationer från en .ini-fil
```

```
    config = configparser.ConfigParser()
    config.read('database.ini')

    # Hämtar anslutningsuppgifter från konfigurationsfilen
    database_type = config['DATABASE']['Type']
    server = config['DATABASE']['Server']
    database = config['DATABASE']['Database']
    driver = config['DATABASE']['Driver']
    trusted_connection = config['DATABASE']['Trusted_Connection']

    # Skapar en anslutningssträng baserat på ovanstående uppgifter och
returnerar en SQLAlchemy "engine"
    connection_string =
f'{database_type}://@{server}/{database}?driver={driver}&trusted_connec
tion={trusted_connection}'
    return create_engine(connection_string)

def get_user_input():
    # Interagerar med användaren för att få söksträng eller visa
hjälpinstruktioner
    print("Ange en boktitel att söka efter (eller skriv 'hjälp' för att
lära dig mer): ")
    keyword = input("Sök efter boktitel: ")
    if keyword.lower() == 'hjälp':
        print("\nWildcard-sökning...")
        print("Använd '%' som wildcard för att ersätta en eller flera
tecken.")
        print("Exempel: '%Harry' hittar alla titlar som slutar med
'Harry'.")
        print("'Potter%' hittar alla titlar som börjar med 'Potter'.")
        print("'%Harry Potter%' hittar alla titlar som innehåller
'Harry Potter'.\n")
        return None, None  # Returnerar None för att signalera att
hjälp har visats
    exact_match = input("Sök på exakt matchning? (ja/nej): ").lower()
    exact_match = 'ja' in exact_match
    return keyword, exact_match

def execute_search_query(engine, keyword, exact_match):
    # Definierar SQL-fråga för att hitta böcker baserat på titel med
möjlighet att ange exakt matchning
    query = """
```

```
    SELECT B.Titel, F.Förnamn + ' ' + F.Efternamn AS Författare,
B.Språk, B.Pris, LS.ButikID, LS.Antal
    FROM Böcker B
    JOIN Författare F ON B.FörfattareID = F.ID
    JOIN LagerSaldo LS ON B.ISBN13 = LS.ISBN
    WHERE B.Titel LIKE :keyword
    """
    keyword = f"%{keyword}%" if not exact_match else
keyword.replace('%', '')
    with engine.connect() as connection:
        result = connection.execute(text(query), {"keyword": keyword})
        books = result.mappings().all()
    return books

def print_books(books):
    # Visar sökresultat i en tabell för tydlig presentation
    if not books:
        print("Inga böcker matchar din sökning.")
        return

    book_table = PrettyTable()
    book_table.field_names = ["Titel", "Författare", "Språk", "Pris",
"ButikID", "Antal i Lager"]

    for book in books:
        book_table.add_row([book['Titel'], book['Författare'],
book['Språk'], f"{book['Pris']} kr", book['ButikID'], book['Antal']])

    print(book_table)

def main():
    # Huvudfunktion som styr programmets flöde
    engine = create_engine_connection()
    while True:
        keyword, exact_match = get_user_input()
        if keyword is None:
            continue  # Skip till nästa iteration om användaren begär
hjälp
        books = execute_search_query(engine, keyword, exact_match)
        print_books(books)

        # Frågar användaren om att fortsätta eller avsluta
        while True:
```

```
            continue_search = input("Vill du göra en annan sökning?
(ja/nej): ").lower().strip()
            if continue_search in ['ja', 'nej']:
                break
            print("Ange endast 'ja' eller 'nej'.")

        if continue_search == 'nej':
            break  # Avslutar loopen om användaren svarar 'nej'

if __name__ == "__main__":
    main()
```

4. Hur koden uppfyller uppgiftskraven

- **Sökfunktionalitet:** Koden tillåter användaren att söka efter boktitlar baserat på en angiven sökterm. Användaren kan välja mellan exakt matchning eller wildcard-sökning för att göra sökningen mer flexibel.
- **Visning av lagerstatus:** Efter att ha hittat relevanta boktitlar, visar koden lagerstatus för dessa böcker i olika butiker, vilket ger användaren en tydlig översikt över tillgängligheten av böckerna.
- **Flexibilitet och användarvänlighet:** Genom att erbjuda både exakt och wildcard-baserad sökning, samt möjligheten att visa hjälpmeddelanden, är koden användarvänlig och flexibel för olika sökbehov.

5. Krav och beroenden

För att köra denna kod måste följande Python-paket vara installerade, som anges i `requirements.txt`:

- **`SQLAlchemy`:**
  Används för att hantera databasen och köra SQL-frågor på ett enkelt och effektivt sätt. Det fungerar som en bro mellan Python-koden och SQL Server-databasen.
- **`pyodbc`:**
  Används som en ODBC-drivrutin för att ansluta Python till SQL Server. Detta paket gör det möjligt för `SQLAlchemy` att kommunicera med databasen.
- **`PrettyTable`:**
  Används för att formatera och visa sökresultat i en tydlig och lättläst tabell i terminalen, vilket förbättrar användarupplevelsen.
- **`configparser`:**
  Används för att läsa konfigurationsfiler (`.ini`-filer) som innehåller anslutningsuppgifter till databasen, vilket gör det enkelt att hantera och ändra inställningar utan att modifiera koden.

6. Användning av konfigurationsfil (`bokhandel.ini`)

Koden använder en konfigurationsfil (`bokhandel.ini`) för att läsa in anslutningsinställningar till databasen, vilket gör det enkelt att ändra dessa inställningar utan att behöva modifiera själva koden.