TECH STARTER

1.6.1. Gruppenaufgabe Adressbuch

Use Case

Stell dir vor, du möchtest all deine Kontakte an einem zentralen Ort speichern und verwalten. Mit einem digitalen Adressbuch kannst du Namen, Telefonnummern, E-Mail-Adressen und Wohnadressen effizient organisieren. Dieses Projekt hilft dir dabei, ein solches Adressbuch zu erstellen, indem du Python zur Kommunikation mit einer externen Datenbank verwendest.

Aufgabenbeschreibung

- Datenbank-VM einrichten:
  - Installiere MySQL oder PostgreSQL auf einer separaten VM.
  - Erstelle eine Datenbank namens `adressbuch`.
  - Definiere eine Tabelle `contacts` mit den Feldern:
    - `id` (Primärschlüssel, automatisch inkrementierend)
    - `name` (VARCHAR)
    - `phone` (VARCHAR)
    - `email` (VARCHAR)
    - `adresse` (VARCHAR)

- Python-VM einrichten:
  - Installiere die benötigten Python-Bibliotheken (`mysql-connector-python` für MySQL oder `psycopg2` für PostgreSQL).
  - Stelle sicher, dass Python auf dem neuesten Stand ist.

- Funktionen implementieren:
  - Kontakt hinzufügen
  - Kontakt anzeigen
  - Kontakt aktualisieren
  - Kontakt löschen

- Benutzeroberfläche:
  - Erstelle ein einfaches Menü in der Konsole, um dem Benutzer Optionen anzubieten.

Tipps

- Datenbankverbindung:
  - Stelle sicher, dass die VM mit dem Python-Skript Zugriff auf die Datenbank-VM hat.
  - Verwende Umgebungsvariablen oder eine Konfigurationsdatei für die Datenbankanmeldeinformationen.

- Sicherheit:
  - Vermeide SQL-Injection, indem du Parameter in SQL-Abfragen sicher übergibst.

- Beispiel: Verwende `cursor.execute(sql, values)` statt `cursor.execute(sql % values)`.

- Fehlerbehandlung:
  - Implementiere Try-Except-Blöcke, um Verbindungs- oder Abfragefehler zu handhaben.
  - Gib dem Benutzer klare Fehlermeldungen aus.

Code Snippets

1. Verbindung zur Datenbank herstellen:

```
import mysql.connector

def create_connection():
    connection = mysql.connector.connect(
        host='DB_VM_IP',
        user='dein_benutzername',
        passwort='dein_passwort',
        database='contacts'
    )
    return connetion
```

Achtung: Stelle sicher, dass du die korrekte IP-Adresse der Datenbank-VM einträgst und die Firewall entsprechend konfiguriert ist.

2. Neuen Kontakt hinzufügen:

```
def add_contact(name, nummer, email, adresse):
    connection = create_connection()
    cursor = connection.cursor()
    sql = "INSERT INTO contects (name, phonenumber, email, adresses)
VALUES (%s, %s, %s, %s, %s)"
    values = (name, phone, email, adresse)
    cursor.execute(sql, values)
    connection.commit()
    cursor.close()
    connection.close()
```

3. Kontakte anzeigen:

```
def view_contacts():
    connection = create_connection()
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM kontakte")
    result = cursor.fetchall()
    for row in result:
        print(f"ID: {row[1]}, Name: {row[2]}, Telefonnummer: {row[3]},
E-Mail: {row[4]}, Adresse: {row[5]}")
    cursor.close()
    connection.close()
```

Allgemeine Hinweise für alle Projekte

- Netzwerkverbindung zwischen VMs:
- Stelle sicher, dass die Python-VM die Datenbank-VM über das Netzwerk erreichen kann.

- Datenbank-Berechtigungen:
- Erstelle einen Datenbankbenutzer mit den notwendigen Berechtigungen (SELECT, INSERT, UPDATE, DELETE).
- **Kleine Falle:** Wenn du die Standard-Benutzeranmeldeinformationen verwendest, könnte das Sicherheitsrisiken mit sich bringen.

- Dokumentation:
- Kommentiere deinen Code ausführlich.
- Erstelle eine README-Datei mit Installationsanweisungen und einer Beschreibung, wie das Programm verwendet wird.

- Versionskontrolle:
- Verwende Git, um Änderungen an deinem Code zu verfolgen.
- Kleine Falle: Vergiss nicht, sensible Informationen wie Passwörter oder geheime Schlüssel aus dem Repository auszuschließen.

Abschließender Hinweis:

Achte darauf, jeden Schritt gründlich zu lesen und zu verstehen, bevor du ihn ausführst oder kopierst. Manchmal können kleine Details einen großen Unterschied machen. Viel Spaß beim Programmieren!