LU06f - DAO-Klassen und Datenmodelle

DAO steht für „Data Access Object“. Ein DAO ist ein Objekt, das eine Schnittstelle zu einer Datenbank bereitstellt. In Flask können DAO-Klassen verwendet werden, um die Logik für den Datenbankzugriff zu kapseln. Zusätzlich zu DAO-Klassen können Datenmodelle wie die `ShoppingItem`-Klasse verwendet werden, um die Datenstruktur zu repräsentieren.

ShoppingItem-Klasse

Die `ShoppingItem`-Klasse repräsentiert die Datenstruktur eines Einkaufsartikels. Sie enthält Attribute wie `item_id`, `item_name` und `quantity`. Anstatt manuell eine `__init__`-Methode und andere spezielle Methoden zu schreiben, können wir die `@dataclass`-Dekorator verwenden, um eine einfachere und sauberere Repräsentation unserer `ShoppingItem`-Klasse zu erhalten. Dies ist besonders nützlich, wenn die Klasse hauptsächlich zur Speicherung von Daten verwendet wird.

```
from dataclasses import dataclass

@dataclass
class ShoppingItem:
    item_id: int
    item_name: str
    quantity: int
```

Beispiel für eine DAO-Klasse

Nachfolgend finden Sie ein Beispiel für eine DAO-Klasse, die CRUD-Operationen (Create, Read, Update, Delete) für eine „shopping_items“-Tabelle durchführt und die `ShoppingItem`-Klasse verwendet.

```
class ShoppingItemDao:
    def __init__(self, db_file):
        self.conn = sqlite3.connect(db_file, check_same_thread=False)
        self.cursor = self.conn.cursor()

    def create_table(self):
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS shopping_items
(item_id INTEGER PRIMARY KEY, item_name TEXT, quantity INT)''')
        self.conn.commit()

    def add_item(self, item):
        self.cursor.execute("INSERT INTO shopping_items (item_name,
quantity) VALUES (?, ?)", (item.item_name, item.quantity))
        self.conn.commit()

    def get_item(self, item_id):
        self.cursor.execute("SELECT * FROM shopping_items WHERE item_id =
```

```
?", (item_id,))
        row = self.cursor.fetchone()
        if row:
            return ShoppingItem(row[0], row[1], row[2])
        return None

    def get_all_items(self):
        self.cursor.execute("SELECT * FROM shopping_items")
        rows = self.cursor.fetchall()
        return [ShoppingItem(row[0], row[1], row[2]) for row in rows]

    def update_item(self, item):
        self.cursor.execute("UPDATE shopping_items SET item_name = ?,
quantity = ? WHERE item_id = ?", (item.item_name, item.quantity,
item.item_id))
        if self.cursor.rowcount > 0:
            self.conn.commit()
            return True
        return False

    def delete_item(self, item_id):
        self.cursor.execute("DELETE FROM shopping_items WHERE item_id = ?",
(item_id,))
        if self.cursor.rowcount > 0:
            self.conn.commit()
            return True
        return False

    def close(self):
        self.conn.close()
```

Beispiel zur Verwendung der ShoppingItemDao-Klasse

Nach der Definition der `ShoppingItemDao`- und `ShoppingItem`-Klassen können wir sie in unserem Code verwenden, um Einkaufsartikel in der Datenbank zu verwalten.

```
# DAO-Instanz erstellen und Tabelle erstellen
dao = ShoppingItemDao('shopping_example.db')
dao.create_table()

# Neuen Einkaufsartikel hinzufügen
new_item = ShoppingItem(None, 'Apple', 5)
dao.add_item(new_item)

# Einkaufsartikel abrufen
retrieved_item = dao.get_item(1)
if retrieved_item:
    print(f"Item ID: {retrieved_item.item_id}, Item Name:
{retrieved_item.item_name}, Quantity: {retrieved_item.quantity}")
```

```
# Einkaufsartikel aktualisieren
retrieved_item.quantity = 7
dao.update_item(retrieved_item)

# Einkaufsartikel löschen
dao.delete_item(1)

# Verbindung schließen
dao.close()
```

M323-LU06