

Anforderungen

Es soll eine Software entwickelt werden, welche einen grafischen Zugriff auf eine Datenbank zulässt. Hierbei sollen die wichtigsten Datenbankfunktionen abgedeckt werden. Das ganze Projekt soll in der Skriptsprache Python realisiert werden.

1 Beschreibung

Für dieses Projekt haben wir uns das Thema Lagerverwaltungssoftware gewählt. Hierbei handelt es sich lediglich um die Lagerverwaltung, sprich das erstellen der zu verbuchenden Belege oder ähnliches ist nicht integriert. Hierbei haben wir folgende Funktionen implementiert.

- [Laden der Belege](#)
- [Listenauswahl](#)
 - [OpenNewWindowBeleg](#)
 - [DrawNewPositionContent](#) - [SavePositions](#) - [SaveWAPositions](#)
- [Inventur](#)
 - [DrawNewIVContent](#)
 - [SaveIV](#)
- [Inventur Druck](#)
- [Hilfsmethoden](#)

2 Hauptfunktionen

2.1 Laden der Belege

Die Belege werden im Wareneingang, Warenausgang und zur Inventur ähnlich vorbereitet. Im Anschluss folgt das Beispiel für den Wareneingang.

```
def Wareneingang():
    mainfenster.title("Easy Log | Wareneingang") # Titel des Hauptfensters ändern
    mdBelege = []
    listbox.delete(0,"end") # alle alten Einträge in der Liste
    löschen
    mdBelege = ReadDatafromDB('mdbelege') # alle nuene Belege Laden
    for row in mdBelege:
        if row[1] != 'WE': # schauen das nur Wareneingänge in der
        Liste bleiben
            mdBelege.remove(row)
    for beleg in mdBelege:
        if beleg.__len__() > 0:
            listbox.insert("end",beleg) # Belege der listbox anfügen
```

Dies ist die Vorbereitungsmethode um alle Wareneingangsbelege aus der Datenbank zu laden und in der Optionsliste anzuzeigen. Hierzu wird die Hilfsmethode `ReadDatafromDB(table_name)` benutzt. Nun werden alle Belege, welche nicht "WE" sind aus dieser Liste entfernt und die restlichen werden in die Auswahlliste hinzugefügt.

2.2 Listenauswahl

Hierbei wird auf das „OnClick“-Event der Listbox reagiert und dann entsprechend die Methode `OpenNewWindowBeleg()` aufgerufen. Dabei wird als Parameter der selektierte Beleg übergeben.

```
listbox.bind('<Double-1>', lambda x :  
OpenNewWindowBeleg(listbox.selection_get().split()))
```

Information zum Vorgehen: Durch den Lambda -ausdruck wird auf das entsprechende Event die selbst definierte Methode gebunden/ zugeordnet.

2.2.1 OpenNewWindowBeleg

In dieser Methode wird nun anhand des übergebenen Belegs die entsprechenden Belegpositionen geladen. Nun wird anhand des übergebenen Beleg ein neues Fenster erstellt, sollte der Beleg nun ein Wareneingang „WE“ sein so wird das Fenster für einen Wareneingang erstellt. Dies muss so umgesetzt werden, da unterschieden werden muss welche Funktionen die Entsprechenden Buttons auf dem Fenster haben. Außerdem muss beim erstellen der Fensters schon definiert werden, daher wird hier entsprechend der Belegart das Fenster erzeugt.

```
def OpenNewWindowBeleg(beleg):  
    positionen = LoadPositions(beleg) # In dieser Methode wird die Hilfsmethode  
    ReadDatafromDB(mdbelegepositionen) aufgerufen  
    if beleg[1] == "WE": # anhand der übergebenen Belegart wird das Fenster  
    dynamisch genneriert  
        if positionen.__len__() > 0:  
            window = tk.Tk()  
            window.title("Positionseingabe")  
            window.geometry("400x400")  
            DarwNewPositionContent(window,positionen,0)  
        else:  
            print("Keine Positionen vorhanden")  
    else:  
        if positionen.__len__() > 0:  
            window = tk.Tk()  
            window.title("Positionseingabe")  
            window.geometry("400x400")  
            DarwNewWAPositionContent(window,positionen,0)
```

```

else:
    print("Keine Positionen vorhanden")

```

Information zu DrawNewPositionContent: Hier wird das Fenster selbst übergeben, die Positionen welcher der Beleg hat und die Start Position, von welcher aus die Positionen gezählt werden. Wichtig ist dass das Fenster selbst übergeben wird, denn diese wird in der Methode dynamisch mit neuem Inhalt gefüllt.

2.2.2 DrawNewPositionContent

In dieser Methode wird der Inhalt des Fenster neu gezeichnet und neu definiert. Daher muss in der Methode vorher das Fenster Objekt übergeben werden, damit für die Nächsten Button aufrufe, das gleiche Fenster verwendet werden kann. Und so lediglich der Inhalt des Fenster ausgetauscht werden kann.

```

### Nur entscheidender Code welcher für weitere Logik relevant ist ###
menge_txt.insert("end",positionen[i][4])
if positionen.__len__()-1 > i:
    next_button = tk.Button(window, text="Next",command= lambda :
LoadNewPosition(window,positionen,i,menge_txt.get("1.0","end")), height= 5,
width=10)
    save_button = tk.Button(window,text="Save",command=lambda:
SavePositions(window,positionen), height= 5, width=10)
else:
    next_button = tk.Button(window, text="Next",command= lambda :
SaveAtLastPositions(window,positionen,i,menge_txt.get("1.0","end").strip()),
height= 5, width=10)
    save_button = tk.Button(window,text="Save",command=lambda:
SavePositions(window,positionen), height= 5, width=10)

```

- Wie man im Codeausschnitt erkennen kann, wird hier wieder mit dem Lambda-Ausdruck gearbeitet , welcher dem command des Buttons eine Methode mit Parametern zuweisen kann, statt einer Methode, welche keine Parameter entgegen nimmt.
- Des Weiteren wird unterschieden, ob wir uns an der letzten Position befinden oder in einer vorherigen, denn sobald die letzte Position erreicht wurde, muss der Command für den Next-Button geändert werden und auf eine andere Funktion zu verweisen. Denn an der letzten Position müssen alle Eingaben verarbeitet werden.
- Durch den Save-Button kann das bearbeiten einer Buchung unterbrochen werden, alle bis dahin bearbeiten Positionen werden dann gebucht.

Hinweis: Für den Warenausgang ist es genau gleich wie für den Wareneingang. Es werden zwei unterschiedliche Methoden genutzt um es in der Erstellung einfacher zu handhaben.

2.2.2.1 SavePositions

Hier werden als Parameter alle Positionen aus dem Beleg übergeben, auserdem wird auch das Fenster-Objekt übergeben. Des Weiteren werden die bearbeiteten Position und Belege aus der Datenbank entfernt, damit diese nicht mehrfach gebucht werden können. Dafür wird die Hilfsmethode [DeleteFromDB\(positionen\)](#) verwendet.

```
def SavePositions(window,positionen):
    for pos in positionen:
        con = mysql.connector.connect(user='root',
        password='*****',host='localhost',database='dbo')
        cursor = con.cursor()
        cursor.execute("INSERT INTO lagerplaetze (`Artikel`,`Menge`) VALUES ( %s ,
%s)", (pos[3] , pos[4])) # %s dient als Parameter
        cursor.close()
        con.commit()
        con.disconnect()
        con.close()
        DeleteFromDB(positionen)
        window.destroy()
```

Es wird für jede einzelne Position der Artikel und die Menge des Artikels in die Datenbank geschrieben. Hierbei muss geachtet werden, das `con.commit()` ausgeführt wird. Dies sorgt dafür, das die Änderungen welche durch den Cursor ausgeführt werden auch in die Datenbank übermittelt werden.

2.2.2.2 SaveWAPositions

Hierbei werden die Positionen welche in einem Warenausgang gebucht werden aus der Datenbank gebucht. Anhand der angeführten Matrix wird entschieden wie die Daten verbucht werden.

| Fall | Umsetzung Code | Datenbank Operation |
|-----------------------------|---------------------------------------|---|
| Lagermenge > gebuchte Menge | <code>if menge > pos[menge]</code> | Update lagerplaetze Set Menge = (menge-pos[menge]) |
| Lagermenge = gebuchte Menge | <code>elif menge == pos[menge]</code> | DELETE FROM lagerplaetze Where Artikel = pos[artikel] |
| Lagermenge < gebuchte Menge | <code>else</code> | print("Fehler ! nicht genügend Ware verfügbar") |

Dementsprechend werden die Datenbank Operationen der Hilfsmethoden aufgerufen ([UpdateDB\(\)](#) oder [DeleteDB\(\)](#))

2.3 Inventur

Über den Button Inventur kann eine Inventur über das gesamte Lager durchgeführt werden. Hierbei werden lediglich die Daten aller Lagerplätze aus der Datenbank geladen. Hier wird auch wieder die Hilfsmethode

`ReadDatafromDB()` verwendet. Auch hierbei wird nun ein neues sich dynamisch verändertes Fenster erstellt. Damit das Laden einer neuen Position und das Speichern der Inventur über den selben Button durchgeführt werden kann.

```
def PerformInventur():
    iv = ReadDatafromDB("lagerplaetze")
    if iv.__len__() > 0:          # Eine Inventur kann nur durchgeführt werden
        wenn sich Ware im Lager befindet
            window = tk.Tk()
            window.title("Inventur")
            window.geometry("400x400")
            DrawNewIVContent(window,iv,0) # "selber" Aufruf wie für Warenein/ausgang
            nur diesmal mit allen Artikeln welche an Lager liegen
            # In dieser Methode wird das Fenster selbst erzeugt. Und mit Komponenten
            gefüllt.
        else:
            print("Keine Lagerplätze vorhanden")
```

2.3.1 DrawNewIVContent

Hierbei werden die wesentlichen Bestandteile des Eingabefensters erstellt. Dies hat den Vorteil, dass die Methode mit jedem Button-Click aufgerufen werden kann. So können die Inhalte dynamisch ausgetauscht werden. Je nachdem an welcher Position man sich gerade befindet, können Artikelnummer und Menge dynamisch angepasst werden. So können auch die Funktionsaufrufe, welche hinter den einzelnen Buttons stehen, dynamisch ausgetauscht werden. z.B. letzte Position ist immer ein Save.

```
if iv.__len__()-1 > i:
    next_button = tk.Button(window, text="Next",command= lambda :
LoadNewIVPosition(window,iv,i,menge_txt.get("1.0","end")), height= 5, width=10)
else:
    next_button = tk.Button(window, text="Next",command= lambda :
SaveAtLastIVPositions(window,iv,i,menge_txt.get("1.0","end").strip()), height= 5,
width=10)
```

Wie oben im Code angegeben, kann so geschaut werden, welcher Command geladen werden muss. Ob es sich um eine neue Position handelt oder um das Speichern der Position.

2.3.2 SaveIV

Hierbei werden die Eingaben, welche während der Inventur gemacht wurden, verarbeitet. Denn nun wird jede Position in die Inventur-Tabelle geschrieben mit Artikelname, der Menge, welche es laut Datenbank sein soll, dem Wert, welcher gezahlt wurde, und die Differenz, welche sich aus den beiden Werten ergibt. Hierbei ist zu achten, da nur mit Integer gerechnet werden kann. Und so die Eingaben, welche durch den Nutzer getätigt

werden umgewandelt. Hier in diesem Fall verwenden wir einen **expliziten cast** verwendet, welcher natürlich bei falscher Nutzer Eingabe zu Fehlern führt.

```
def SaveIV(window,iv):
    for pos in iv:
        con = mysql.connector.connect(user='root',
password='*****',host='localhost',database='dbo')
        cursor = con.cursor()
        cursor.execute("INSERT INTO inventur
(`Artikel`,`MengeSoll`,`MengeIst`,`Diff`) VALUES (%s,%s,%s,%s)",
(pos[1],str(pos[2]),str(pos[3]),str(int(pos[3])-int(pos[2]))))
        con.commit()
        cursor.close()
        con.close()
    window.destroy()
```

2.4 Inventur Druck

Als Zusatz Funktion haben wir noch eine Druck-Report Funktion eingebunden. Über den Button "Inventur Print" kann ein Inventur Bericht über die letzte Inventur durchgeführt werden. Hierbei werden alle Daten der Inventur aus der Datenbank geladen und in eine HTML Seite eingebaut und ausgegeben. Hierbei fehlt noch der Optische Aspekt, da nur die reinen Daten als Tabelle gedruckt werden ohne Formatierung oder ähnliches.

```
liste = ReadDatafromDB("inventur")
with open("E:/Project/DB/Inventur.html",'w') as f:
    f.write("<html>")
    f.write("<table>")
    f.write("<tr><td>ID</td><td>Artikel</td><td>MengeSoll</td>
<td>MengeIst</td><td>Diff</td></tr>")
    for item in liste:
        f.write("<tr>")
        for pos in item:
            f.write("<td>"+str(pos)+"</td>")
        f.write("</tr>")
    f.write("</table>")
    f.write("</html>")
webbrowser.open_new_tab("file:///E:/Project/DB/Inventur.html")
```

3 Hilfsmethoden

Hierbei werden alle Methoden aufgelistet welche nicht im direkten Zusammenhang zum eigentlich Programm stehen. Sondern entsprechende Aufgaben für das Programm übernehmen, aber nicht in einem spezifischen

Kontext sondern generell für das gesamte Projekt.

3.1 ReadDatafromDB(table_name)

| Parameter / Rückgabe | Name | Datentyp |
|----------------------|------------|---------------------|
| Parameter | table_name | string |
| Rückgabe | liste | Array[List<string>] |

Diese Methode nimmt einen string Parameter entgegen. Dieser definiert von welcher Tabelle die abfrage gestartet werden soll. Hierbei wird ein simples `Select * From "table_name"` ausgeführt. Dies führt dazu, das man diese Methode überall dort verwenden kann wo man die Daten aus der Datenbank laden möchte. Man muss lediglich die Tabelle übergeben aus welcher man die Daten abfragen möchte. So kann eine gewisse Modularität für das Projekt gewährleistet werden. Des Weiteren hat diese Abfrage den Vorteil, das man auf den Index der Datenbank zugreifen kann und so den [primär Schlüssel](#) der Tabelle zur Verfügung hat. Diesen kann man dann für ein Update oder Delete übergeben und hat den entsprechenden Eintrag.

```
def ReadDatafromDB(table_name):
    liste = []
    con = mysql.connector.connect(user='root',
    password='*****',host='localhost',database='dbo')
    cursor = con.cursor()
    cursor.execute('SELECT * FROM %s' % table_name)
    result = cursor.fetchall()
    liste = [list(i) for i in result]
    cursor.close()
    con.close()
    return liste
```

3.2 DeleteFromDB(positionen)

| Parameter / Rückgabe | Name | Datentyp |
|----------------------|------------|---------------------|
| Parameter | positionen | Array[List<string>] |
| Rückgabe | void | void / null |

Diese Methode löscht die übergebenen Positionen aus der Datenbank und löscht den entsprechenden Beleg sobald diese abgeschlossen ist aus der Datenbank. Hierbei werden die zu löschenden Positionen übergeben. Diese werden dann durchlaufen und jede Position wird aus der Datenbank gelöscht. Des Weiteren wird dann im Anschluss der entsprechende Beleg auch aus der Datenbank entfernt, sodass nicht ein Beleg zweimal verbucht werden kann. Um die Datenintegrität zu wahren.

```
def DeleteFromDB(positionen):
    for pos in positionen:
        con = mysql.connector.connect(user='root',
```

```

password='*****',host='localhost',database='dbo')
    cursor = con.cursor()
    cursor.execute("DELETE FROM mdbelegepositionen WHERE id = "+ str(pos[0]))
    cursor.close()
    con.commit()
    con.disconnect()
    con.close()
    con = mysql.connector.connect(user='root',
password='*****',host='localhost',database='dbo')
    cursor = con.cursor()
    cursor.execute("DELETE FROM mdbelege WHERE Id = "+ str(pos[1]))
    cursor.close()
    con.commit()
    con.disconnect()
    con.close()

```

3.3 UpdateDB(cr,pos,i)

| Parameter / Rückgabe | Name | Datentyp |
|----------------------|------|--------------------|
| Parameter | cr | Position im Cursor |
| Parameter | pos | List<string> |
| Parameter | i | List<string> |
| Rückgabe | void | void / null |

Hierbei handelt es sich um die UpdateMethode bei einem Warenausgang, wenn mehr Ware an Lager ist als verschickt werden soll. Hierbei wird die Menge aus der Datenbank übergeben welche im Cursor steht. Die Position aus der Nutzereingabe und die Position aus der Datenbank selber. Hierbei wird dann die neue Menge errechnet welche sich aus **Datenbank Lagerbestand - Nutzereingabe** errechnet. Nun wird dann die neue Menge an die Position geschrieben welche aus der Übermethode übergeben wurde.

```

def UpdateDB(cr,pos,i):
    newmenge = int(cr[0]) - int(pos[4])
    con = mysql.connector.connect(user='root',
password='root',host='localhost',database='dbo')
    cursor = con.cursor()
    cursor.execute("Update lagerplaetze SET Menge = '"+ str(newmenge) +" ' WHERE id
= "+ str(i[0]))
    con.commit()
    cursor.close()
    con.disconnect()
    con.close()

```

Hinweis: Hierbei ist zu beachten, dass wenn es öfters den gleiche Artikel im Lager gibt, dies zu Problem im Warenausgang führen kann.

3.4 DeleteDB(cr,pos,i)

| Parameter / Rückgabe | Name | Datentyp |
|----------------------|------|--------------|
| Parameter | i | List<string> |
| Rückgabe | void | void / null |

Hierbei handelt es sich um eine Spezialmethode für den Warenausgang, da sobald die ausgehende Ware gleich der Ware im Lager ist, muss diese Position aus der Datenbank entfernen werden, damit diese nicht gebucht werden kann. Dafür wird nur die entsprechende Position übergeben welche gelöscht werden soll. Dazu wird ein einfaches DELETE verwendet: `DELETE FROM lagerplaetze WHERE id = "+ str(i[0])`. Dort wird die ID der Position übergeben wodurch die entsprechende Position gelöscht werden kann.

```
def DeleteDB(i):
    con = mysql.connector.connect(user='root',
    password='root',host='localhost',database='dbo')
    cursor = con.cursor()
    cursor.execute("DELETE FROM lagerplaetze WHERE id = "+ str(i[0]))
    con.commit()
    cursor.close()
    con.disconnect()
    con.close()
```

Hinweis: Auserdem ist zu beachten, dass wenn es öfters den gleiche Artikel im Lager gibt, dies zu Problem im Warenausgang führen kann.

4 Weiter Informationen

4.1 Explizites Casten

Von casten wird gesprochen sobald eine Typ-Konvertierung durchgeführt wird, sprich von einem Datentyp in einen andern Datentyp umgewandelt wird. Bei einem expliziten cast wird dann noch angegeben in welchen Datentyp der Wert gewandelt werden soll. Hier ein kleines Beispiel:

```
number = 123
string_number = "123"

print(type(number))
print(type(string_number))

##### AUSGABE #####
<class 'int'>
<class 'str'>
```

```
###_____CAST_____###
number = 123
string_number = "123"

number = str(number)           # expliziter cast in einen String
string_number = int(string_number) # expliziter cast in ein Integer

print(type(number))
print(type(string_number))

#### AUSGABE ####
<class 'str'>
<class 'int'>
```

Hierbei gibt es einige Vor- und Nachteile welche man beachten sollte wenn man einen expliziten Cast verwendet:

| Vorteil | Nachteil |
|--|---|
| Einfach und schnelle Umwandlung von Datentypen | Der Wert welcher gecasted werden soll muss dem Datentyp des Cast entsprechen. Ansonsten läuft der Cast auf einen Fehler dies kann z.B. ein Buchstabe in einem int cast sein |

4.2 Primär Schlüssel

Primär Schlüssel werden in Datnbanktabellen verwendet. Hierbei handelt es sich um einen eindeutigen Wert, welcher in der gesamten Tabelle nur einmal vorkommt. Dadurch können Datensätze eindeutig identifiziert werden. Auf diese eindeutigen Werte wird dann auch bei einem UPDATE oder DELETE verwiesen. Dadurch wird immer nur der entsprechende und nicht mehrere Datensätze angesprochen, was man Umständen nicht möchte.

| Name | Datentyp | Eigenschaft |
|---------|-------------|---|
| ID | INTEGER | Primär Schlüssel, Einzigartig, Automatisches Inkrementieren, Nicht Null |
| Artikel | VARCHAR(40) | Nicht Null |
| Menge | VARCHAR(40) | - |

Hinweis: Hierbei wäre ID nun ein Primär Schlüssel welcher nur einmal in der Tabelle vorkommen kann. Spricht ID = 1 gibt es exakt ein eiziges mal. Dadurch wird über die WHERE Clause mit ID = 1 immer nur der eine Datensatz getroffen.