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 die erstellen der zu verbuchenden Belege uä ist nicht integriert. Hierbei haben wir folgende Funktionen implementiert

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 landen 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 „OnDoubleClick"-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 entwspecehende Event die selbstdefenierte 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. Dies muss beim erstellen der Fensters schon defeniert werden, dahher wird hier entsrpechend 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 übergebn, die Positionen welcher der Beleg hat und die Start Position, von welcher aus die Positionen gezählt werden. Wichtig ist das das Fenster selber übergeben wird, da dies in der Methode sich der Inhalt des Fenster dynamisch neu erzeugt.

2.2.2 DrawNewPositionContetnt

In dieser Methode wird der Inhalt des Fenster neu gezeichnet und neu defeniert. Dahher muss in der Methode vorhher das Fenster Objekt übergeben werden damit für die Nächten Button aufrufe, das gleiche Fentser

besetehen kann, aber lediglich der Inhalt des Fensters neu gezeichnet 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 in Codeausschnitt erkenn kann, wird hier wirder mit dem Lambda-Ausdruck gearbeit , welcher dem command des Buttons eine Methode mit parametern zuweisen kann, statt einer Methode, welche keine Parameter entgegen nimmt.
- Desweitern wird unterschieden, ob wir uns an de letzten Position befinden oder in einer vorhherigen, 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 Postion müssen alle eingaben verarbeitet werden.
- Durch den Save-Button kann das bearbeiten einer Buchung unterbrochen werden, alle bis dahin bearbeiten Positonen werden dann gebucht.

Hinweis: Für den Warenausgang ist es genau gleich wie für den Wareneingang. Es werden zwei unterschidliche 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, desweitern wird auch das Fenster-Objekt übergeben. Desweitern werden die bearbieteten Position und belege aus der Datenbak 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 einzelene Position der Artikel und die Menge es Artikels in die Datenbank geschrieben. Hierbei muss geachtet werden, das `con.commit()` ausgeführt wird. Dies sorgt dafür, das die Änderungen welche durch den Cursor ausgeführet 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 angeführeten Matrix wird entschieden die Daten verbucht werden.

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

Dementsrpechend 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 ledichlich die Daten aller Lagerplätze aus der Datenbank geladen. Hier wird auch wieder die Hilfsmethode ReadDatafromDB() verwendet. Auch hierbei wird nun ein neues sich dynamisch veränderdes Fenster erstellt. In welchem man auch wie Wareneingabe und Warenausgang die anzegeigten Komponenten dynamisch Austauschen kann und so verschiedene Funktionen an verschiedenen Stellen bereitstellen kann.

```
def PerformInventur():
    iv = ReadDatafromDB("lagerplaetze")
    if iv.__len__() > 0:            # Eine Inventur kann nur duchgeführt werden
wenn sich ware im Lager befindet
        window = tk.Tk()
        window.title("Inventur")
        window.geometry("400x400")
        DarwNewIVContent(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 wesentlihen Bestandteile des eingabe Fensters erstellt. Dies hat den Vorteil, das die Methode mit jedem Button click aufgerufen werden kann. So können die Inhalte dynamisch ausgetauscht werden. Jenachedem an welcher Postionen man sich grade befindet können Artikelnummer und Menge dynamisch nagepasst werden. So könne 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 ob angegeben kann so geschaut werden, welceh Command geladen werden muss. Ob es sich um eine neue Position handelt oder um das speichern der Psotion.

2.3.2 SaveIV

Hierbei werden die Eingaben, welche wärende der Inventur gemacht wurden verarbeitet. Denn nun wird jede Postionen in die entsprechende Inventur Tabelle geschrieben mit Artikel der Menge welche es laut Datenbak sein soll, dem Wert welcher gezählt wurde und die Diefferenz 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 umgewandet werden. Hier in diesem fall verwenden wir einen expleziten 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 eine Inventur Bericht über die letzte Inventur durchgeführt werden. Hierbei werden alle Daten der Inventr aus der Datenbank geladen und in eine HTML seite eingbeaut 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 übernehemen, aber nicht in einem spezifischen kontext sondern generell für das gesatmmte 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. Diser defeniert von welcher tabelle die abfrage gestatet 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 modluarität für das Projekt gewährleistet werden. Desweitern hat diese Abfrage den Vorteil, das man auf den

Index der Datenbank zugreifen kann und so den primär Schlüssel der Tabelle zuverfügung 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 eingebenen Positionen aus der Datenbank und löscht die entsprechendne Belege solad diese abgeschlossen sind aus der Datendank. Hierbei werden die zu löschenenden Positioenn übergeben. Diese werden dann durchlaufen und jede Position wird aus der Datenbank gelöscht. Desweitern wird dann im anschluss der entsprechende Beleg auch aus der Datenbank entfernt, sodass nicht ein Beleg zweimal verbucht werden kann. Um die Datenintegrietät 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 Warenausgabg, wenn mehr Ware an Lager ist als verschickt werden soll. Hierbei wird die Mege 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 wir 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 Warenausgabg, da sobald die ausgehende Ware gleich der Ware im Lager ist muss diese Position aus der Datenbank entfenert werden, damit diese nicht gebucht werden kann. Hierfür wird nur die entsprechende Position übergeben welche glöscht werden soll.

Hierfür wird ein einfaches DELETE verwenndet: `DELETE FROM lagerplaetze WHERE id = "+ str(i[0])`. Dort wird die ID der Position übergeben wodurch die entsprechende Positon 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: Hierbei 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 gesprchen sobald eine Typ konvetierung durchgeführt wird, sprich von einem Datentyp in einen andern Datentyp übergewandelt wird. Bei einem explizieten 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)                # expleziter cast in einen String
string_number = int(string_number)  # expleziter 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 expleziten Cast verwendet:

Vorteil	Nachteil
Einach 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 zb ein Buchstabe in einem int cast sein

4.2 Primär Schlüssel

Von Primär Schlüsseln werden in Tabellen verwenndet. Hierbei handelt es sich um einen eindeutigen Wert, welcher in der gesammten Tabelle nur einmal vorkommt. Dadurch können Datensätze eindeutig identifiziert werden. Auf diese eindeutigen Werte wir dann auch bei einem UPDATE oder DELETE verwieden, da dadurch immer nur der eine entsprechende Datensatz angesprochen wird und nicht mehrere, was man unter 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 der Primär Schlüssel welcher nur einmal in der Tabelle vorkommen kann. Spricht ID = 1 gibt es exakt eineiziges mal. Dadurch wird über die WHERE Clause mit ID = 1 immer nur der eine Datensatz getroffen.