Nhat Nguyen

Assignment 3 – CSCI 4140

Contents

[Tools used: 3](#_Toc117104977)

[Create database 4](#_Toc117104978)

[Create an application and connect to the database 10](#_Toc117104979)

[Methods to query database 14](#_Toc117104980)

# Tools used and Git repo

* MySQL Work Bench Community was used to run a localhost server
* Bootstrap was used for UI
* Since I have little experience with backend on Java, I decided to use Python Flask framework for backend
* Git repository link: <https://github.com/nhatnguyen215/A3_CSCI4140>

# Create database

* Similar to assignment 1, database was created using forward engineering after the models are created. For this one though, I created 3 databases, the first 2 are similar to the database in assignment 1 and 2, for the third one for company Z, we just need to remove Parts table

Diagram

Description automatically generated

Diagram

Description automatically generated

* Below is the code generated from the EER diagram for database Z, the first 2 databases are similar to the first and second assignments, so I thought I won’t have to include it here:

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema Z\_db

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema Z\_db

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `Z\_db` DEFAULT CHARACTER SET utf8 ;

USE `Z\_db` ;

-- -----------------------------------------------------

-- Table `Z\_db`.`Z\_Clients207`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Z\_db`.`Z\_Clients207` (

`clientId207` INT NOT NULL,

`clientName207` VARCHAR(45) NULL,

`clientCity207` VARCHAR(45) NULL,

`clientPassword207` VARCHAR(45) NULL,

`moneyOwned207` VARCHAR(45) NULL,

PRIMARY KEY (`clientId207`),

UNIQUE INDEX `clientId207\_UNIQUE` (`clientId207` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Z\_db`.`Z\_POs207`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Z\_db`.`Z\_POs207` (

`poNo207` INT NOT NULL,

`clientCompID207` VARCHAR(45) NULL,

`dataOfPO207` DATE NULL,

`status207` VARCHAR(45) NULL,

`Clients207\_clientId207` INT NOT NULL,

PRIMARY KEY (`poNo207`, `Clients207\_clientId207`),

UNIQUE INDEX `poNo207\_UNIQUE` (`poNo207` ASC) VISIBLE,

INDEX `fk\_POs207\_Clients2071\_idx` (`Clients207\_clientId207` ASC) VISIBLE,

CONSTRAINT `fk\_POs207\_Clients2071`

FOREIGN KEY (`Clients207\_clientId207`)

REFERENCES `Z\_db`.`Z\_Clients207` (`clientId207`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Z\_db`.`Z\_Lines207`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Z\_db`.`Z\_Lines207` (

`POs207\_poNo207` INT NOT NULL,

`Parts207\_partNo007` INT NOT NULL,

`lineNo207` INT NOT NULL,

`qty207` INT NULL,

`priceOrdered207` DECIMAL NULL,

PRIMARY KEY (`POs207\_poNo207`, `Parts207\_partNo007`, `lineNo207`),

INDEX `fk\_POs207\_has\_Parts207\_POs207\_idx` (`POs207\_poNo207` ASC) VISIBLE,

CONSTRAINT `fk\_POs207\_has\_Parts207\_POs207`

FOREIGN KEY (`POs207\_poNo207`)

REFERENCES `Z\_db`.`Z\_POs207` (`poNo207`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

* Screenshots of table showed in MySQL Workbench with dump data:

Company X parts:

Graphical user interface, application

Description automatically generated

Company Y parts:

Graphical user interface, application

Description automatically generated

Clients in company Z:

Graphical user interface, application

Description automatically generated

Some other dump data in company Z:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, Word

Description automatically generated

# Create an application and connect to the database

The web application is run on a virtual environment using Python flask framework, the setup process can be found here: <https://flask.palletsprojects.com/en/2.2.x/installation/>

Firstly, I link the framework with the local database, with the following config:

Text

Description automatically generated

The web app will have different routes (pages) for each method we will be running (Look up parts, submit PO, list line, list PO):

Text

Description automatically generated

After that, I created html pages and link them to their route

A picture containing text

Description automatically generated

Home page:

Graphical user interface, text

Description automatically generated

Parts page that list all the parts available, with all parts provided by company X and Z, no duplicates:

Table

Description automatically generated with medium confidence

PO page where you can find your POs connected to your client ID:

Graphical user interface, text, application

Description automatically generated

Line page where you can look up your line based on PO number:

Chart

Description automatically generated with low confidence

# Methods to query database

All the methods were written in run.py file:

* On Part page, we want to query all the parts available at company X and Y, but we have to remove all the duplicates, function parts() was created for that:

Text

Description automatically generated

* When click on find part on the home page, I used get method to return all the information of parts table from the database:

Table

Description automatically generated with medium confidence

* On PO page, we want to submit a purchase order to company Z, and also insert a new line that connects part to that order, with the correct company we want to make the PO from, first we want to get information from customer:

Graphical user interface, text, application

Description automatically generated

* Submission is successful, since P2-2 is provided by both companies X and Y, however Y has lower price, so the line should mention the PO is to make with company Y

Graphical user interface, text, application, chat or text message

Description automatically generated

Company y price for P2-2 is 12:

Graphical user interface, text, application

Description automatically generated

Company X price for P2-2 is 13:

Graphical user interface, text, application

Description automatically generated

* After submitting the form, database has been updated with new PO and line:

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* To be able to submit PO and add line, we need to check if ClientID, PartNo match the ones in the system. We also have to check if the quantity the customer is ordering is greater than QOH or not from both company X and Y. If all the conditions are correct, we compare the price of the product from both companies and select the one with the lower price. Afterwards, we query and insert new PO and line:
* @app.route('/po', methods=['GET', 'POST'])
* def po():
* #SELECT list of clients
* cur = mysql.connection.cursor()
* cur.execute('''SELECT clientId207 FROM z\_db.z\_clients207''')
* clientIDList = cur.fetchall()
* print(clientIDList)
* cur.close()
* #SELECT list of part numbers
* cur = mysql.connection.cursor()
* cur.execute('''SELECT partNo007 FROM x\_db.x\_parts207''')
* partNoList\_x = cur.fetchall()
* cur.close()
* cur = mysql.connection.cursor()
* cur.execute('''SELECT partNo007 FROM y\_db.y\_parts207''')
* partNoList\_y = cur.fetchall()
* cur.close()
* if request.method == 'POST':
* compID = request.form['compID']
* clientID = request.form['clientID']
* partNo = request.form['partNo']
* qty = request.form['qty']
* status = 'Pending'
* poNo = 0
* date = datetime.date.today()
* #Check if clientID entered matches the ones in the system
* clientIDCheck = False
* for i in clientIDList:
* for j in i:
* if int(clientID) == int(j):
* clientIDCheck = True
* #Check if the partNo entered matches the ones in database X
* partNoCheck\_x = False
* for i in partNoList\_x:
* for j in i:
* if partNo == j:
* partNoCheck\_x = True
* #Check if the partNo entered matches the ones in database y
* partNoCheck\_y = False
* for i in partNoList\_y:
* for j in i:
* if partNo == j:
* partNoCheck\_y = True
* #Query to find qoh of part in X
* cur = mysql.connection.cursor()
* qohQuery = '''SELECT qoh207 FROM x\_db.x\_parts207 WHERE partNo007 = %s'''
* cur.execute(qohQuery, [partNo])
* qohList\_x = cur.fetchall()
* cur.close()
* #Query to find qoh of part in Y
* cur = mysql.connection.cursor()
* qohQuery = '''SELECT qoh207 FROM x\_db.x\_parts207 WHERE partNo007 = %s'''
* cur.execute(qohQuery, [partNo])
* qohList\_y = cur.fetchall()
* cur.close()
* #Check if qoh is less than quantity ordered or no in x database
* qohCheck\_x = False
* for i in qohList\_x:
* for j in i:
* if int(qty) < int(j):
* qohCheck\_x = True
* #Check if qoh is less than quantity ordered or no in y database
* qohCheck\_y = False
* for i in qohList\_y:
* for j in i:
* if int(qty) < int(j):
* qohCheck\_y = True
* #Function check if input is correct
* def checkPoValid(client\_ID, qoh, partNo):
* if client\_ID == True:
* if qoh == True and partNo == True:
* return True
* #Set company value to send a PO to
* if checkPoValid(clientIDCheck, qohCheck\_x, partNoCheck\_x):
* company = "X"
* if checkPoValid(clientIDCheck, qohCheck\_y, partNoCheck\_y):
* company = "Y"
* #if both company X and Y has provides the same part with sufficent qoh
* if checkPoValid(clientIDCheck, qohCheck\_x, partNoCheck\_x) and checkPoValid(clientIDCheck, qohCheck\_y, partNoCheck\_y):
* #Query to find price of entered part in x database
* cur = mysql.connection.cursor()
* priceQuery = '''SELECT currentPrice207 FROM x\_db.x\_parts207 WHERE partNo007 = %s'''
* cur.execute(priceQuery, [partNo])
* priceList\_x = cur.fetchall()
* cur.close()
* price\_x = 0
* for i in priceList\_x:
* for j in i:
* price\_x = j
* ##Query to find price of entered part in y database
* cur = mysql.connection.cursor()
* priceQuery = '''SELECT currentPrice207 FROM y\_db.y\_parts207 WHERE partNo007 = %s'''
* cur.execute(priceQuery, [partNo])
* priceList\_y = cur.fetchall()
* cur.close()
* price\_y = 0
* for i in priceList\_y:
* for j in i:
* price\_y = j
* #Lower price is used
* price = min(price\_x, price\_y)
* #Company value to insert in "lines" table
* if price\_x <= price\_y:
* company = "X"
* else:
* company = "Y"
* if checkPoValid(clientIDCheck, qohCheck\_x, partNoCheck\_x) or checkPoValid(clientIDCheck, qohCheck\_y, partNoCheck\_y):
* #Insert PO to Z database
* poQuery ="""INSERT INTO z\_db.z\_pos207 (clientCompID207, dataOfPO207, status207, Clients207\_clientId207)
* VALUES ( %s, %s, %s, %s)"""
* poValues = ( compID, date, status, clientID)
* cur = mysql.connection.cursor()
* cur.execute(poQuery, poValues)
* mysql.connection.commit()
* cur.close()
* #Find poNo of the line
* cur = mysql.connection.cursor()
* findPoNo = '''SELECT poNo207 FROM z\_db.z\_pos207 WHERE clientCompID207 = %s AND Clients207\_clientId207 = %s'''
* findPoNoValues = (compID, clientID)
* cur.execute(findPoNo, findPoNoValues)
* poNoList = cur.fetchall()
* poNo = poNoList[0]
* #Insert line
* lineQuery ="""INSERT INTO z\_db.z\_lines207 (POs207\_poNo207, Parts207\_partNo007, qty207, priceOrdered207, company\_207)
* VALUES ( %s, %s, %s, %s, %s)"""
* lineValues = (poNo, partNo, qty, price, company)
* cur = mysql.connection.cursor()
* cur.execute(lineQuery, lineValues)
* mysql.connection.commit()
* cur.close()
* return render\_template('successPoSubmit.html')
* else:
* return render\_template('inputError.html')
* return render\_template('po.html')
* For the last two pages line and poList, they have similar functionality. We want to SELECT data from a table based on a certain condition. On PoList page, we display all the POs that the clientID has:

(Before)

Graphical user interface, text, application

Description automatically generated

(After)

Graphical user interface, text, application

Description automatically generated

(Table in the database)

Graphical user interface, text, table

Description automatically generated with medium confidence

* Similarly, the Line pages does the same thing, but this time with PO number as input and line table as output:

(Before)

Graphical user interface, text, application

Description automatically generated

(After)

Chart, scatter chart

Description automatically generated

(Table in the database)

Graphical user interface, table

Description automatically generated with medium confidence

* Below is the method for poList and Line pages:
* @app.route('/poList', methods=['GET', 'POST'])
* def poList():
* globalData = ''
* if request.method == "POST":
* clientID = request.form.get('clientID')
* cur = mysql.connection.cursor()
* query = """SELECT \* FROM z\_db.z\_pos207 WHERE Clients207\_clientId207 = %s"""
* cur.execute(query, [clientID])
* data = cur.fetchall()
* globalData = data
* cur.close()
* return render\_template('poList.html', data=globalData)
* @app.route('/line', methods=['GET', 'POST'])
* def line():
* globalData = ''
* if request.method == "POST":
* poNo = request.form.get('poNum')
* cur = mysql.connection.cursor()
* query = """SELECT \* FROM z\_db.z\_lines207 WHERE POs207\_poNo207 = %s"""
* cur.execute(query, [poNo])
* data = cur.fetchall()
* globalData = data
* cur.close()
* return render\_template('line.html', data=globalData)