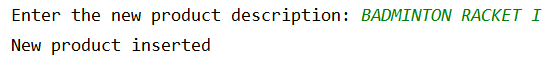
**WEEK 10 – PYTHON - MANIPULATING DATA PRACTICE QUESTIONS**

1. Prompt the user to enter a product description ‘BADMINTON RACKET I’ and insert a new row into the product table as follows:



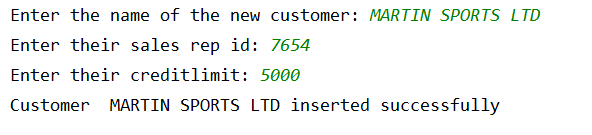
Solution

**import** sqlite3  
db = sqlite3.connect(**'u:/sqlite/teaching.db'**)  
cursor = db.cursor()  
prod\_desc = input(**"Enter the new product description: "**)  
sql\_insert = **"INSERT INTO product(descrip) \  
 VALUES (?)"**cursor.execute(sql\_insert, (prod\_desc,))  
print(**"Inserted new product"**)  
db.commit()

From SQLite, run the query SELECT \* FROM product; and check that the new product has been inserted and it has a prodid value.

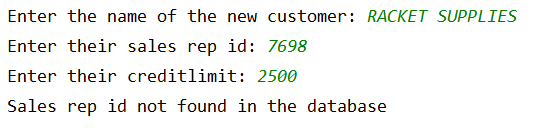
2. Prompt the user to enter a customer name, creditlimit and repid and insert a new row into the customer table. Check to ensure the repid entered is a valid empno of a salesman in the emp table.

Execution 1 testing the successful insert of a new customer



From SQLite, run the query SELECT \* FROM customer; and check that the new customer has been inserted and it has a custid value.

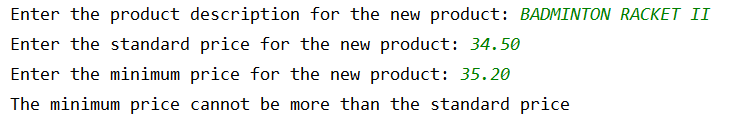
Execution 2 testing the error is displayed if the sales rep is not a SALESMAN in the emp table



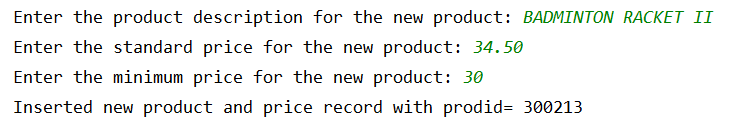
**import** sqlite3  
db = sqlite3.connect(**'c:/sqlite/teaching.db'**)  
cursor = db.cursor()  
*# enable foreign key constraints*cursor.execute(**"PRAGMA foreign\_keys=ON"**)  
cust\_name = input(**"Enter the name of the new customer: "**)  
cust\_repid = input(**"Enter their sales rep id: "**)  
cust\_creditlimit = int(input(**"Enter their creditlimit: "**))  
*# check if the sales rep id entered is a valid SALESMAN in the employee table*sql\_query = **"SELECT empno \  
 FROM emp \  
 WHERE empno=? \  
 AND job='SALESMAN'"**cursor.execute(sql\_query,(cust\_repid,))  
emp\_row = cursor.fetchone()  
**if** emp\_row **is not None**:  
 *# if the sales rep id exists, insert a new row into the customer table* sql\_insert = **"INSERT INTO customer (name, repid, creditlimit) \  
 VALUES (?,?,?)"** cursor.execute(sql\_insert,(cust\_name,cust\_repid,cust\_creditlimit))  
 *# commit the changes* db.commit()  
 print(**"Customer "**,cust\_name, **"inserted successfully"**)  
**else**:  
 *# if the sales rep id doesnt exist, print an error message* print (**"Sales rep id not found in the database"**)

1. Prompt the user to enter a product description, standard price and minimum price. If the minimum price is greater than the standard price, print an error message otherwise insert a new row into the product and price tables with the same prodid.

Execution 1 testing the error is displayed if the minimum price is greater than the standard price



Execution 2 testing that the new rows are added successfully if the minimum price is less than or equal to the standard price



Solution

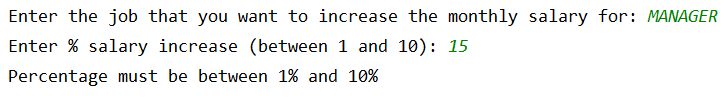
**import** sqlite3  
**import** datetime  
db = sqlite3.connect(**'c:/sqlite/teaching.db'**)  
cursor = db.cursor()  
*#get the next prodid from the sqlite\_sequence table*cursor.execute(**"SELECT seq+1 \  
 FROM sqlite\_sequence \  
 WHERE name='product'"**)  
next\_prodid = cursor.fetchone()  
*#prompt the user to enter a product description, standard price and minimum price for the new product*prod\_desc = input(**"Enter the product description for the new product: "**)  
std\_price = float(input(**"Enter the standard price for the new product: "**))  
min\_price = float(input(**"Enter the minimum price for the new product: "**))  
*#if the min price is invalid, print an error message***if** min\_price > std\_price:  
 print(**"The minimum price cannot be more than the standard price"**)  
**else**:  
 *#insert a new row into the product table for the new product* sql\_insert = **"INSERT INTO product(prodid, descrip) \  
 VALUES (?, ?)"** cursor.execute(sql\_insert, (next\_prodid[0], prod\_desc))  
 *#get todays date and convert it to the format YYYY-MM-DD* curr\_date = datetime.date.today().strftime(**"%Y-%m-%d"**)  
 *#insert a new row into the price table for the new product* sql\_insert = **"INSERT INTO price (prodid, startdate, stdprice, \  
 minprice, enddate) \  
 VALUES (?, ?, ?, ?, ?)"** cursor.execute(sql\_insert, (next\_prodid[0], curr\_date, std\_price, min\_price, **None**))  
 print(**"Inserted new product and price record with prodid="**, next\_prodid[0])  
 *#commit the changes* db.commit()

From SQLite, run the query SELECT \* FROM product; and SELECT \* from price; to check that the new product and price rows have been inserted with the same prodid value.

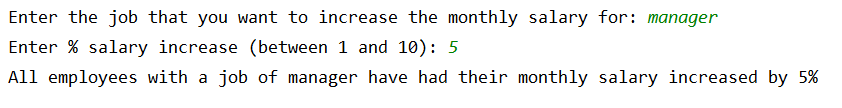
1. Prompt the user to enter a job type and a percentage between 1 and 10 and increase the monthly\_sal of all the employees with that job by the percentage entered.

In SQLite, execute the query SELECT \* FROM emp; to see the current monthly salaries.

Execution 1 to test when the percentage increase is outside the acceptable range



Execution 2 to test that the job can be entered in lowercase and the employees with the selected job have their monthly salary increased



In SQLite, execute the query SELECT \* FROM emp; again to see the monthly salaries of the right employees have been correctly increased.

Solution

**import** sqlite3  
**import** datetime  
db = sqlite3.connect(**'c:/sqlite/teaching.db'**)  
cursor = db.cursor()  
job\_to\_incr = input(**"Enter the job that you want to increase the monthly salary for: "**)  
perc\_incr = int(input(**"Enter % salary increase (between 1 and 10): "**))  
*#check the percentage increase is valid***if** perc\_incr >= 1 **and** perc\_incr <= 10:  
 sal\_multiplier = 1 + (perc\_incr/100)  
 *#increase the monthly salary of all employees with the selected job* sql\_update = **"UPDATE emp \  
 SET monthly\_sal = monthly\_sal\*(?) \  
 WHERE job=UPPER(?)"** cursor.execute(sql\_update,(sal\_multiplier, job\_to\_incr))  
 *#commit any changes* db.commit()  
 print(**"All employees with a job of {0} have had their monthly salary increased by {1}%"**.format(job\_to\_incr, perc\_incr))  
**else**:  
 print (**"Percentage must be between 1% and 10%"**)

1. Prompt the user to enter the name of an employee and delete the row for the employee with that name. If the employee you attempt to delete is a manager or salesman associated with a customer, the foreign key will prevent deletion so make sure you write an exception to handle this gracefully.

**import** sqlite3  
db = sqlite3.connect(**'c:/sqlite/teaching.db'**)  
cursor = db.cursor()  
*# enable foreign key constraints*cursor.execute(**"PRAGMA foreign\_keys=ON"**)  
*# start of code where exception needs to be handled***try**:  
 *# check if employee entered exists in the database* emp\_name = input(**"Enter the name of the employee to delete "**)  
 sql\_query = **"SELECT empno FROM emp WHERE \  
 UPPER(ename)=UPPER(?)"** cursor.execute(sql\_query,(emp\_name,))  
 emp\_row = cursor.fetchone()  
 empno = emp\_row[0]  
 *# if employee exists then try to delete them* **if** emp\_row **is not None**:  
 sql\_delete = **"DELETE FROM emp WHERE empno=(?)"** cursor.execute(sql\_delete,(empno,))  
 *# commit the changes* db.commit()print(**"Employee "**,emp\_name, **"deleted successfully"**)  
 **else**:  
 print (**"Employee entered not found in database"**)  
**except** db.Error:  
 print (**"Employee cannot be deleted as they are a manager or sales rep"**)

Test the code three times by entering an invalid employee, entering an employee who is a manager or sales rep and finally by entering a valid employee who is a CLERK.

From SQLite, run the query SELECT \* FROM emp and check that the employee has been successfully deleted.