Names: Thi Vo, Tram Nguyen, Jaeger Henderson

Database Design Final Project Final-Report

Introduction

Are all of your movie theaters closed because of a pandemic? Fear not! Our team has created a ticket and account management system for a movie theater. That way you can simulate gathering information for movie showings and buy your own tickets for those showings (Disclaimer: No actual movie showing is included).

Tickets are bought by the customers and sold by the theater employees. They have a row number for seating, price paid, type of payment, the customer who bought it (if they're a registered user), the employee who sold it, and the showtime (which references the date, time, movie, and auditorium). While multiple tickets can be bought at the same time, they will be separated by a unique ticketID. Registered customers and employees are users of our management system.

Entities

The types of entities are listed below along with their attributes are listed below:

1. Ticket {
    a. <u>TicketID</u>
    b. SellsTicket not null foreign key
    c. ShowID not null foreign key
    d. BuysTicket foreign key unique
    e. RowNumber not null
    f. Price not null
    g. PaymentMethod not null
    h. }

   FK: SellsTicket references Employee, ShowID references MovieShowtime, BuysTicket references Customer.
2. Auditorium {
    a. <u>AuditID</u>
    b. ScreenType (Regular 2D, 3D, IMAX) not null
    c. Capacity not null
    d. }
3. MovieShowtime {
    a. <u>ShowID</u>

    b. Date not null
    c. Time not null
    d. MovieID not null foreign key
    e. Assigned not null foreign key
    f. }

   FK: MovieID references Movie, Assigned references Auditorium.

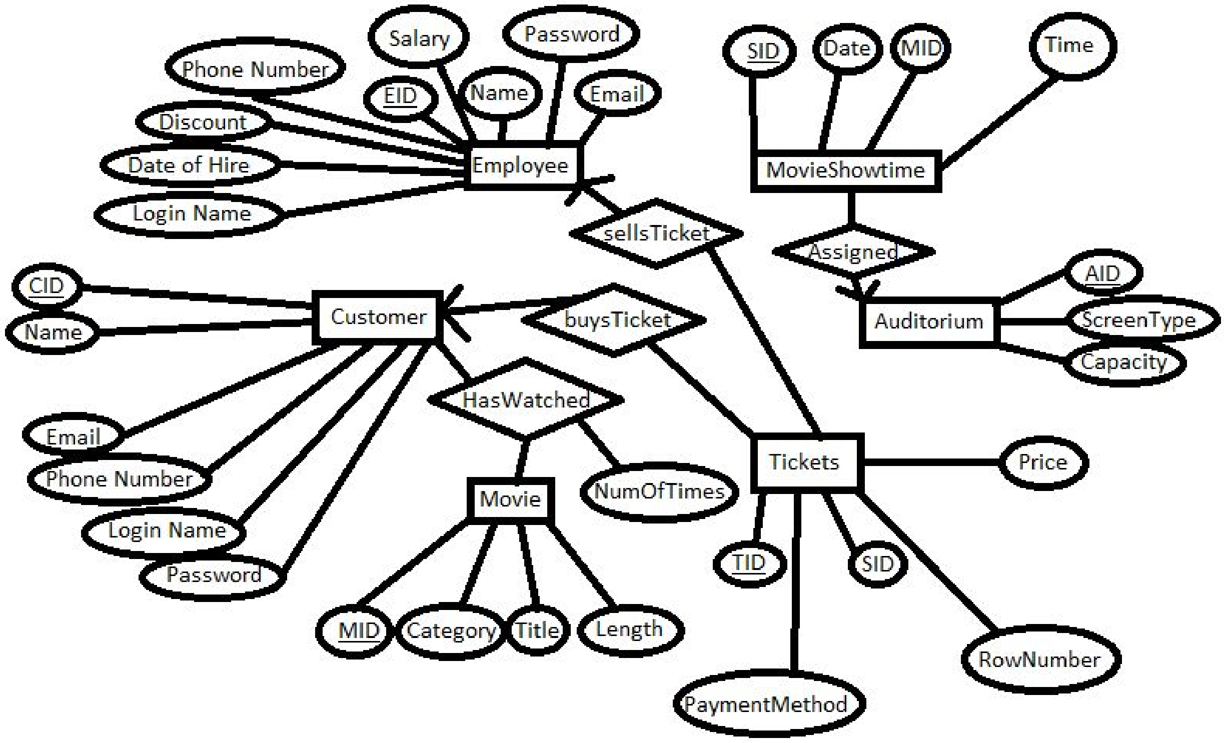
4. Movie {
    a. <u>MovieID</u>
    b. Title not null
    c. Category
    d. Length }
5. Customer {
    a. <u>CustID</u>
    b. Name not null
    c. Email not null unique
    d. PhoneNumber unique
    e. LoginName not null unique
    f. Password not null unique
    g. }
6. Employee {
    a. <u>EmpID</u>
    b. Name not null
    c. Email not null unique
    d. PhoneNumber not null unique
    e. Salary not null
    f. DateOfHire
    g. LoginName not null unique
    h. Password not null unique
    i. Discount
    j. }
7. Watched {
    a. <u>WatchID</u>
    b. CustID
    c. MovieID
    d. NumOfTimes
    e. }

   FK: CustID references Customer, MovieID references Movie

<u>Relations</u>

BuysTicket (Ticket, Customer) **Many-to-One Relationship**
SellsTicket (Ticket, Employee) **Many-to-One Relationship**
Assigned (MovieShowtime, Auditorium) **One-to-Many Relationship**
Watched (Customer, Movie) **Many-to-Many Relationship**

ER Diagram

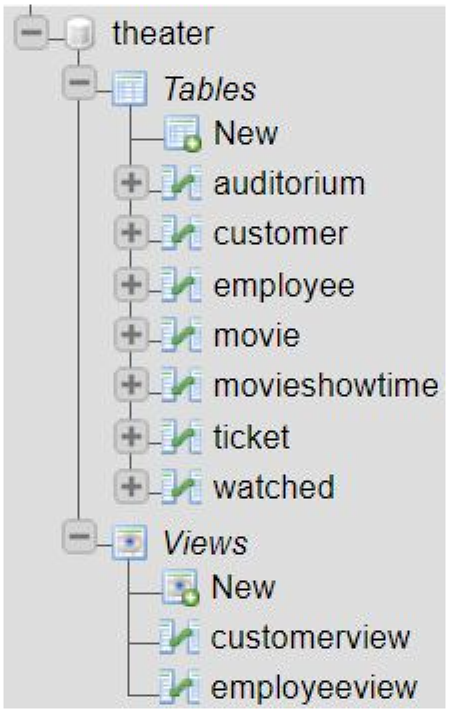


Our chosen platforms

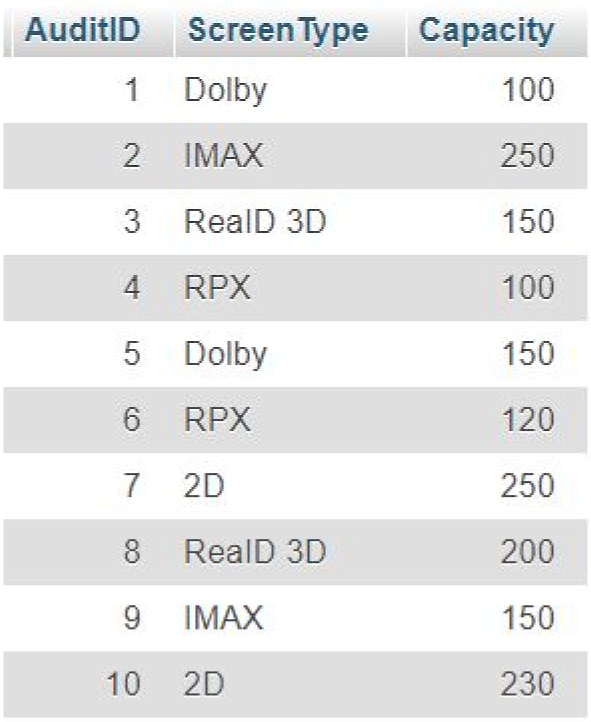
- Our DBMS is MySQL on phpMyAdmin.
- Python is the general programming language we're using to connect to the database.
- We have a text-based interface also using python.

phpMyAdmin Hosting

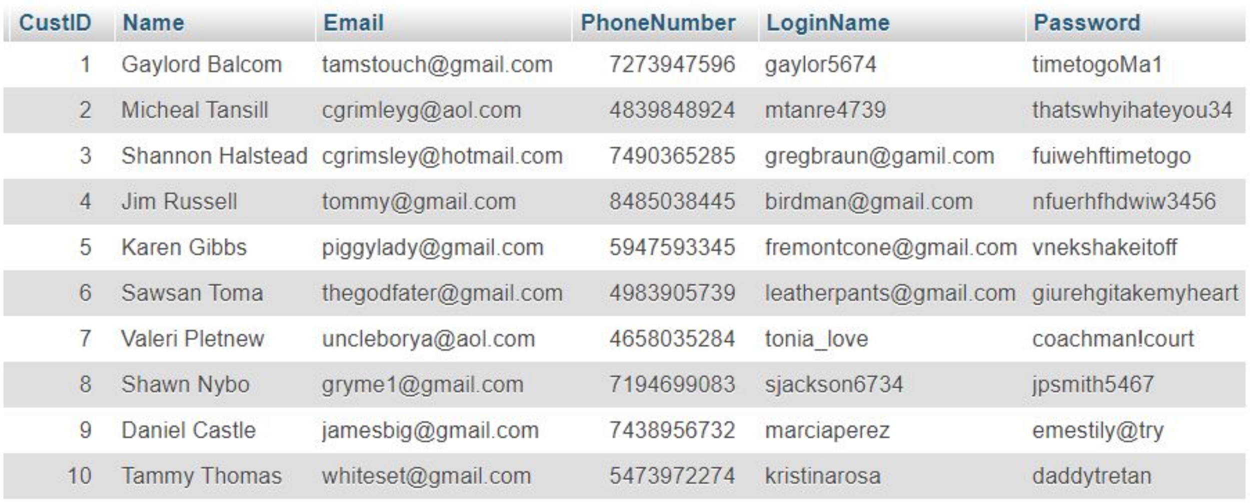
We're hosting our database called 'theater' in the image below via phpMyAdmin.



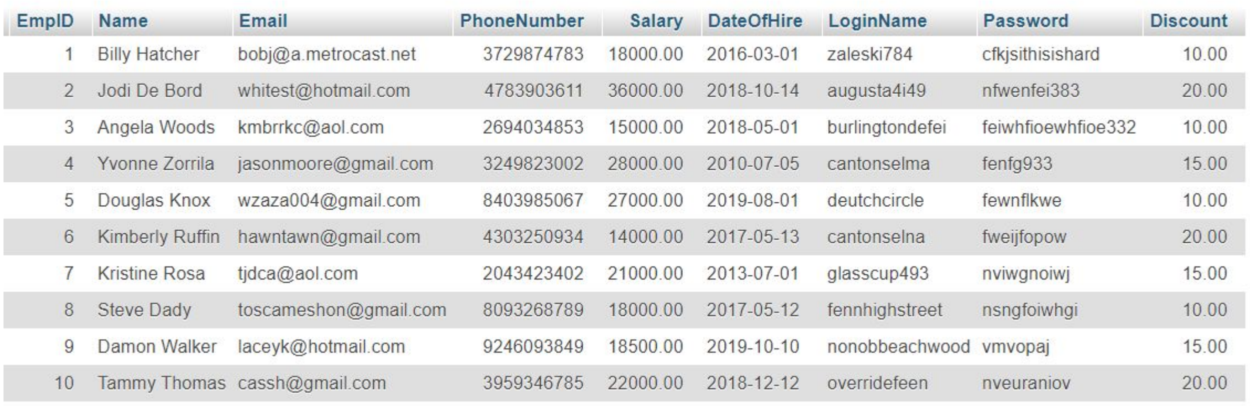
All of the tables and views from the ‘theater’ database are shown below. The order is Auditorium, Customer, Employee, Movie, MovieShowtime, Ticket, Watched.



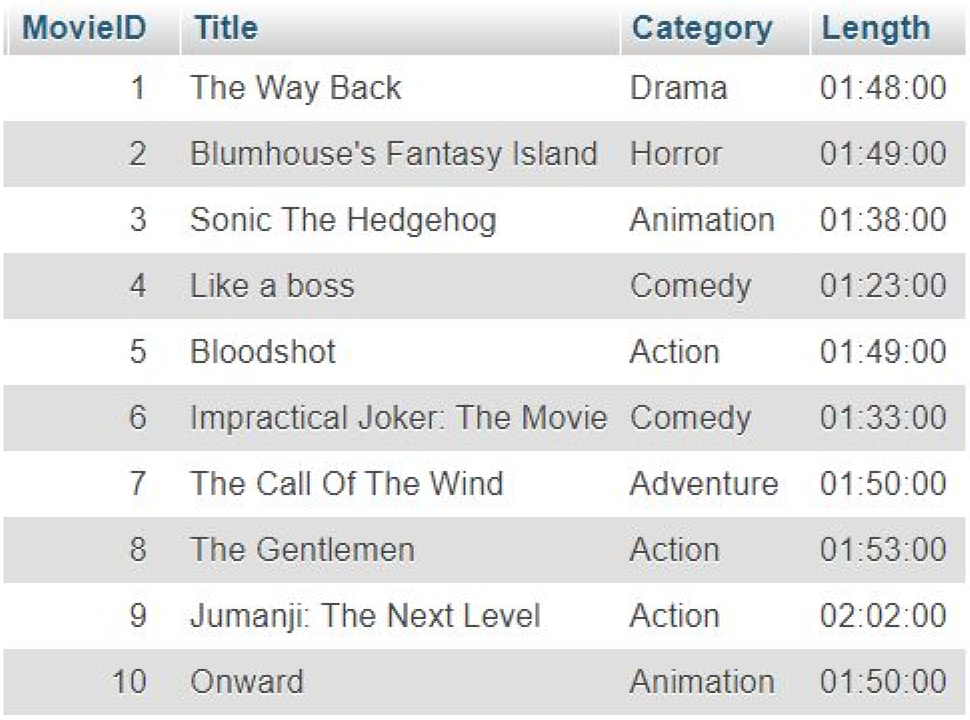
AuditID	ScreenType	Capacity
1	Dolby	100
2	IMAX	250
3	RealD 3D	150
4	RPX	100
5	Dolby	150
6	RPX	120
7	2D	250
8	RealD 3D	200
9	IMAX	150
10	2D	230



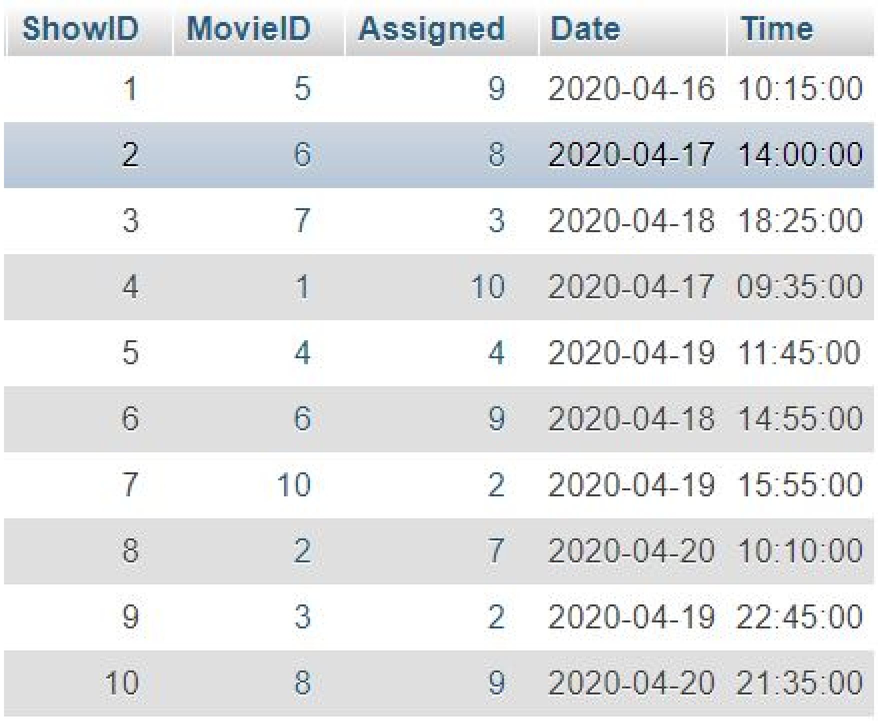
CustID	Name	Email	PhoneNumber	LoginName	Password
1	Gaylord Balcom	tamstouch@gmail.com	7273947596	gaylor5674	timetogoMa1
2	Micheal Tansill	cgrimleyg@aol.com	4839848924	mtanre4739	thatswhyihateyou34
3	Shannon Halstead	cgrimsley@hotmail.com	7490365285	gregbraun@gamil.com	fuiwehftimetogo
4	Jim Russell	tommy@gmail.com	8485038445	birdman@gmail.com	nfuerhfhdwiw3456
5	Karen Gibbs	piggylady@gmail.com	5947593345	fremontcone@gmail.com	vnekshakeitoff
6	Sawsan Toma	thegodfater@gmail.com	4983905739	leatherpants@gmail.com	giurehgitakemyheart
7	Valeri Pletnew	uncleborya@aol.com	4658035284	tonia_love	coachman!court
8	Shawn Nybo	gryme1@gmail.com	7194699083	sjackson6734	jpsmith5467
9	Daniel Castle	jamesbig@gmail.com	7438956732	marciaperez	emestily@try
10	Tammy Thomas	whiteset@gmail.com	5473972274	kristinarosa	daddytretan



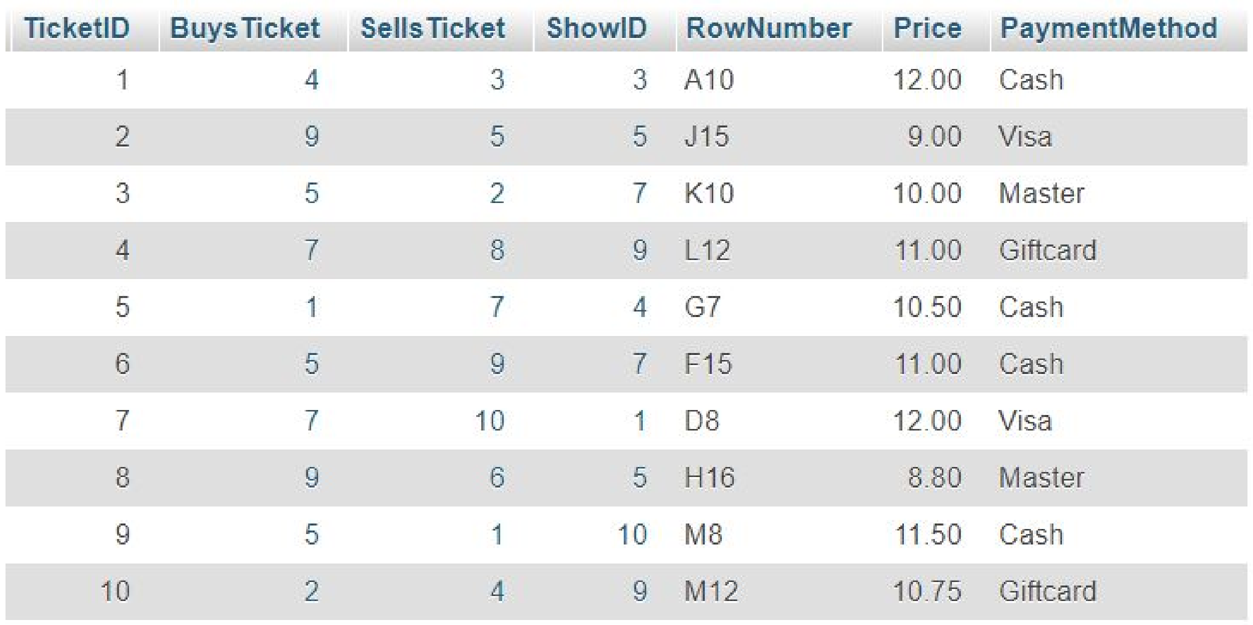
EmpID	Name	Email	PhoneNumber	Salary	DateOfHire	LoginName	Password	Discount
1	Billy Hatcher	bobj@a.metrocast.net	3729874783	18000.00	2016-03-01	zaleski784	cfkjsithisishard	10.00
2	Jodi De Bord	whitest@hotmail.com	4783903611	36000.00	2018-10-14	augusta4i49	nfwenfei383	20.00
3	Angela Woods	kmbrrkc@aol.com	2694034853	15000.00	2018-05-01	burlingtondefei	feiwhfioewhfioe332	10.00
4	Yvonne Zorrila	jasonmoore@gmail.com	3249823002	28000.00	2010-07-05	cantonselma	fenfg933	15.00
5	Douglas Knox	wzaza004@gmail.com	8403985067	27000.00	2019-08-01	deutchcircle	fewnflkwe	10.00
6	Kimberly Ruffin	hawntawn@gmail.com	4303250934	14000.00	2017-05-13	cantonselna	fweijfopow	20.00
7	Kristine Rosa	tjdca@aol.com	2043423402	21000.00	2013-07-01	glasscup493	nviwgnoiwj	15.00
8	Steve Dady	toscameshon@gmail.com	8093268789	18000.00	2017-05-12	fennhighstreet	nsngfoiwhgi	10.00
9	Damon Walker	laceyk@hotmail.com	9246093849	18500.00	2019-10-10	nonobbeachwood	vmvopaj	15.00
10	Tammy Thomas	cassh@gmail.com	3959346785	22000.00	2018-12-12	overridefeen	nveuraniov	20.00



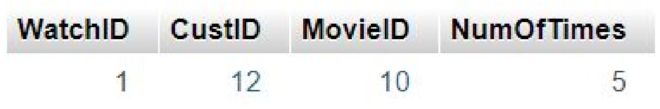
MovieID	Title	Category	Length
1	The Way Back	Drama	01:48:00
2	Blumhouse's Fantasy Island	Horror	01:49:00
3	Sonic The Hedgehog	Animation	01:38:00
4	Like a boss	Comedy	01:23:00
5	Bloodshot	Action	01:49:00
6	Impractical Joker: The Movie	Comedy	01:33:00
7	The Call Of The Wind	Adventure	01:50:00
8	The Gentlemen	Action	01:53:00
9	Jumanji: The Next Level	Action	02:02:00
10	Onward	Animation	01:50:00



ShowID	MovieID	Assigned	Date	Time
1	5	9	2020-04-16	10:15:00
2	6	8	2020-04-17	14:00:00
3	7	3	2020-04-18	18:25:00
4	1	10	2020-04-17	09:35:00
5	4	4	2020-04-19	11:45:00
6	6	9	2020-04-18	14:55:00
7	10	2	2020-04-19	15:55:00
8	2	7	2020-04-20	10:10:00
9	3	2	2020-04-19	22:45:00
10	8	9	2020-04-20	21:35:00



TicketID	BuysTicket	SellsTicket	ShowID	RowNumber	Price	PaymentMethod
1	4	3	3	A10	12.00	Cash
2	9	5	5	J15	9.00	Visa
3	5	2	7	K10	10.00	Master
4	7	8	9	L12	11.00	Giftcard
5	1	7	4	G7	10.50	Cash
6	5	9	7	F15	11.00	Cash
7	7	10	1	D8	12.00	Visa
8	9	6	5	H16	8.80	Master
9	5	1	10	M8	11.50	Cash
10	2	4	9	M12	10.75	Giftcard

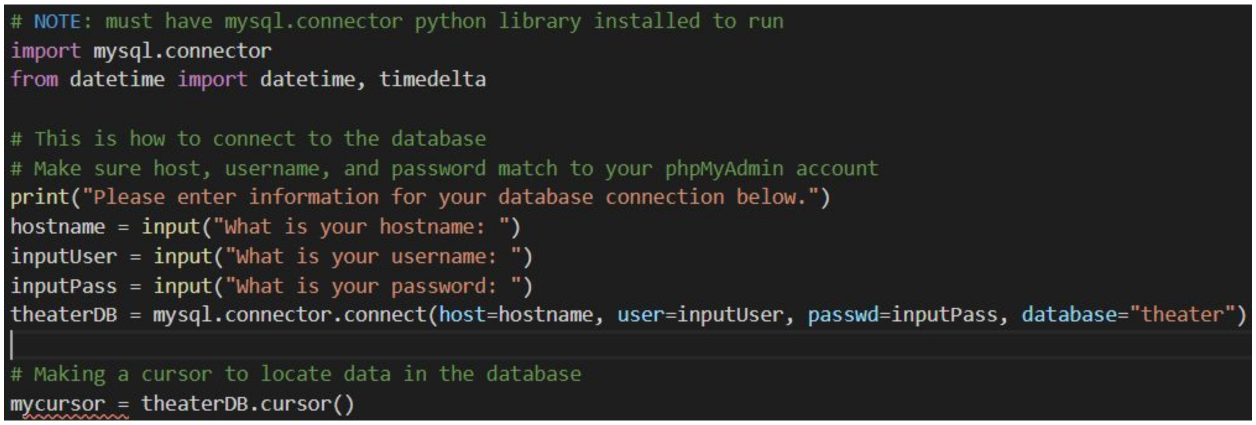


WatchID	CustID	MovieID	NumOfTimes
1	12	10	5

<u>Main-Menu Interface</u>

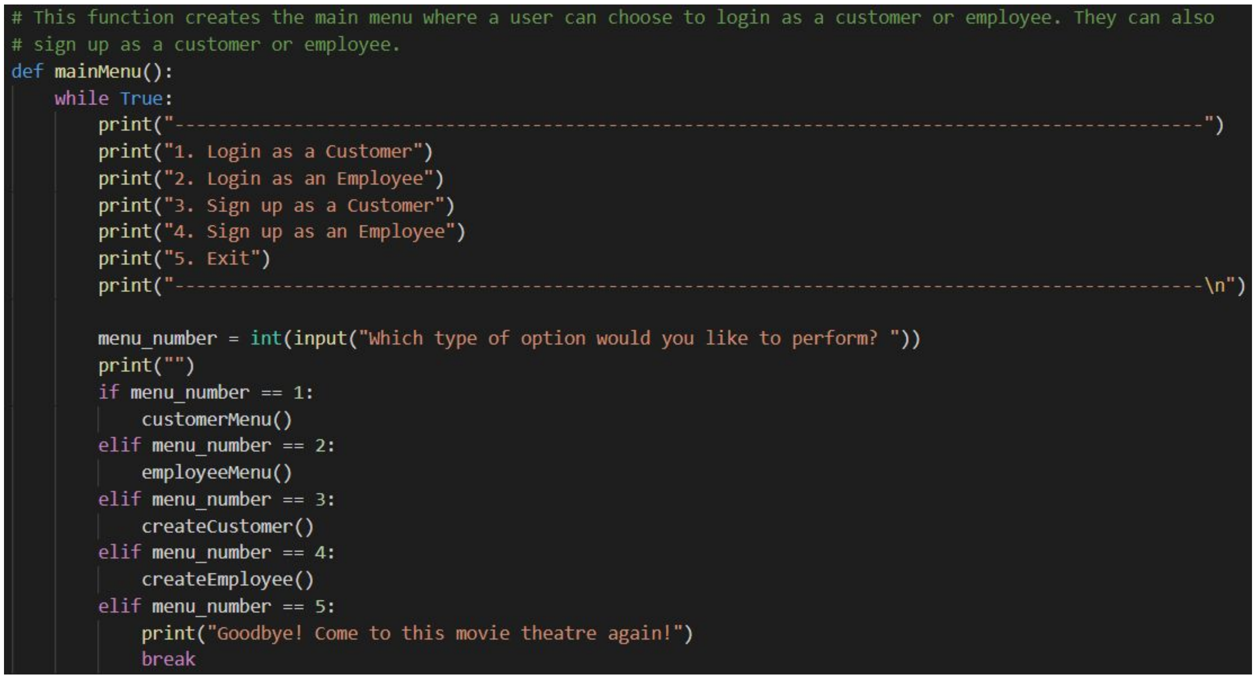
The interface for the queries was written in python. It shows how we connect to the MySQL database using a MySQL python connector. You are able to select your user's account on MySQL by entering your hostname, username, and password. The host name, username, and password must match a phpMyAdmin account. We use this connection for a cursor 'mycursor' that will perform SQL queries later on. You must install the MySQL python connector as well as our theater.sql file to use this interface then (shown below).

- Run the following in your command line: pip install mysql-connector-python

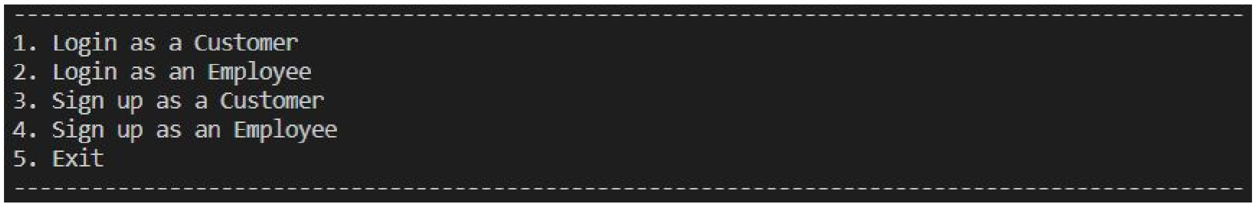


```python
# NOTE: must have mysql.connector python library installed to run
import mysql.connector
from datetime import datetime, timedelta

# This is how to connect to the database
# Make sure host, username, and password match to your phpMyAdmin account
print("Please enter information for your database connection below.")
hostname = input("What is your hostname: ")
inputUser = input("What is your username: ")
inputPass = input("What is your password: ")
theaterDB = mysql.connector.connect(host=hostname, user=inputUser, passwd=inputPass, database="theater")

# Making a cursor to locate data in the database
mycursor = theaterDB.cursor()
```

The next images below show the function and interface for our main menu. It allows customers or employees to login or sign-up using a switch-like statement (which isn't available in python, so we used an if-else statement). Inside the if-else statements are function-calls for the customer menu and employee menu (for logging-in), as well as the create queries for customers and employees (for signing-up).



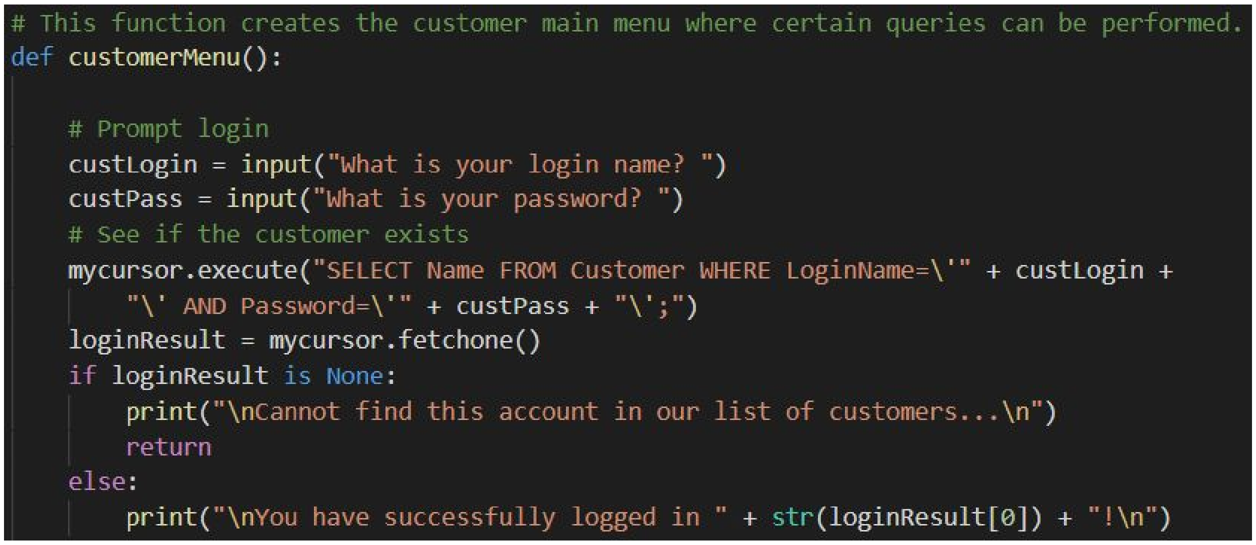
```python
# This function creates the main menu where a user can choose to login as a customer or employee. They can also
# sign up as a customer or employee.
def mainMenu():
    while True:
        print("-----------------------------------------------------------------------------------------------")
        print("1. Login as a Customer")
        print("2. Login as an Employee")
        print("3. Sign up as a Customer")
        print("4. Sign up as an Employee")
        print("5. Exit")
        print("-----------------------------------------------------------------------------------------------\n")

        menu_number = int(input("Which type of option would you like to perform? "))
        print("")
        if menu_number == 1:
            customerMenu()
        elif menu_number == 2:
            employeeMenu()
        elif menu_number == 3:
            createCustomer()
        elif menu_number == 4:
            createEmployee()
        elif menu_number == 5:
            print("Goodbye! Come to this movie theatre again!")
            break
```



```
---------------------------------------------------------------------------------------------
1. Login as a Customer
2. Login as an Employee
3. Sign up as a Customer
4. Sign up as an Employee
5. Exit
---------------------------------------------------------------------------------------------
```

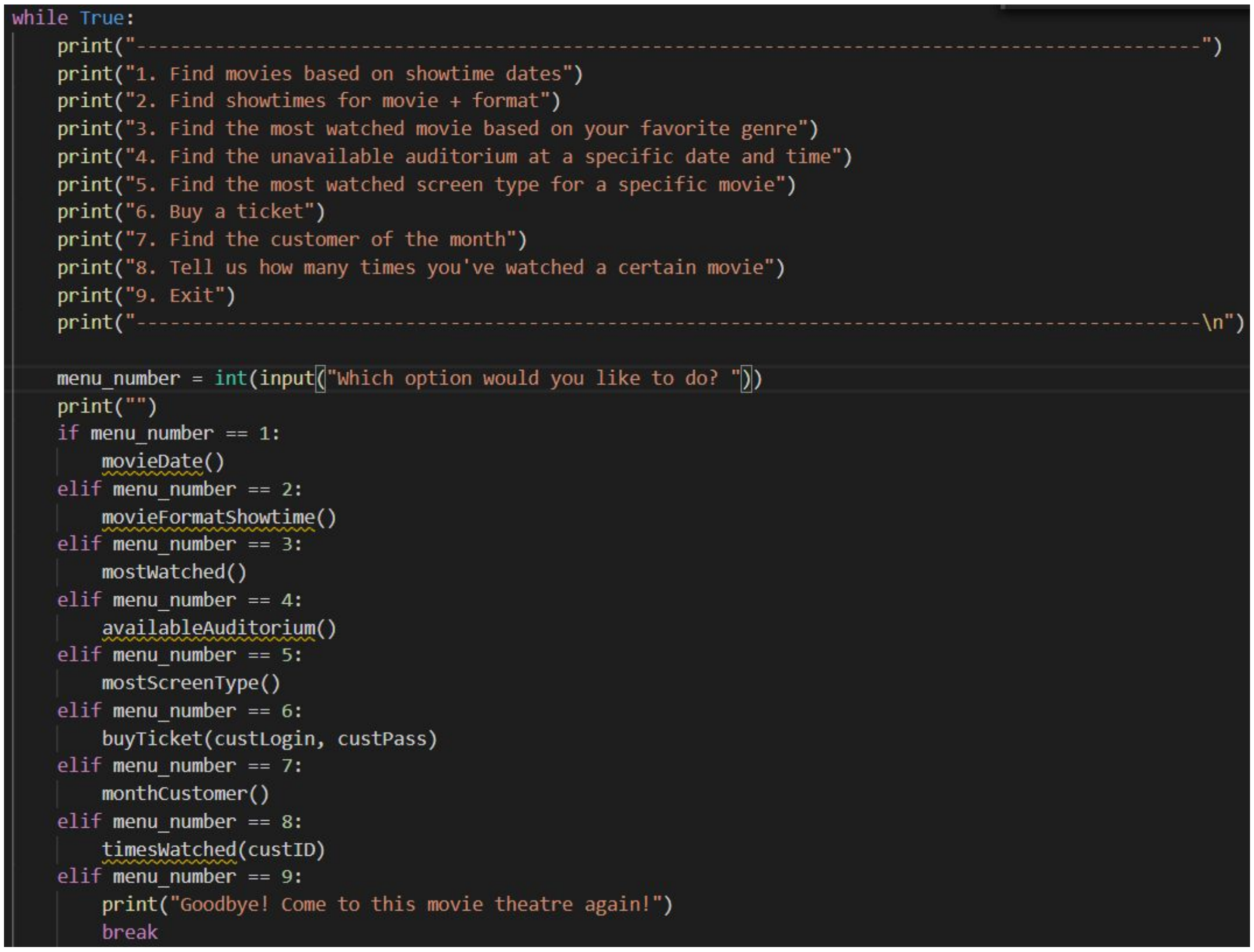
Customer / Employee Menu Interface

It is pretty important for both customers and employees to login. These login prompts occur in the customer/employee menu functions. Their interfaces are the exact same and their code is very similar. As seen in the python code below, an SQL query is used to match-make an inputted login name and password with the attributes from the associated tables. If the login is successful, we tell the user (their actual name) that. Nonetheless, a failed login will tell the user that and also exit out of the customer/employee menu, which leads back to the main menu function.

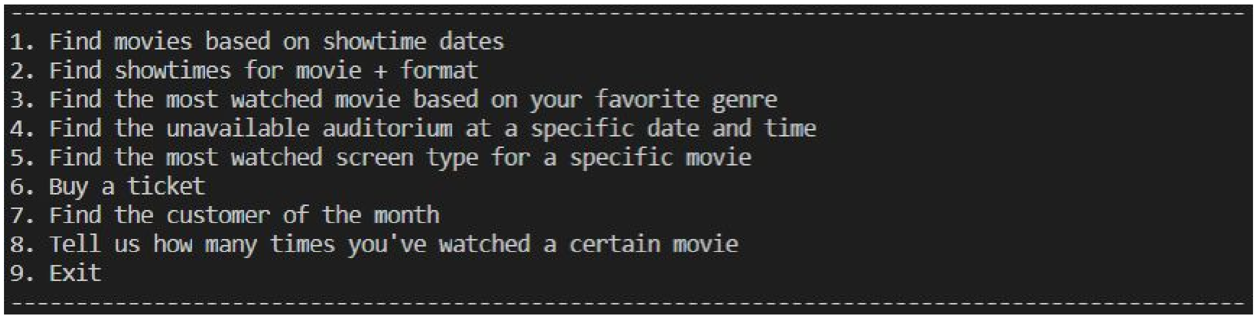


```python
# This function creates the customer main menu where certain queries can be performed.
def customerMenu():

    # Prompt login
    custLogin = input("What is your login name? ")
    custPass = input("What is your password? ")
    # See if the customer exists
    mycursor.execute("SELECT Name FROM Customer WHERE LoginName=\'" + custLogin +
        "\' AND Password=\'" + custPass + "\';")
    loginResult = mycursor.fetchone()
    if loginResult is None:
        print("\nCannot find this account in our list of customers...\n")
        return
    else:
        print("\nYou have successfully logged in " + str(loginResult[0]) + "!\n")
```

A successful login will lead to the next the rest of the customer menu where the customer can choose from a selection of 8 queries or exit back into the main menu as seen below. The employee menu behaves the same way with 2 queries instead.



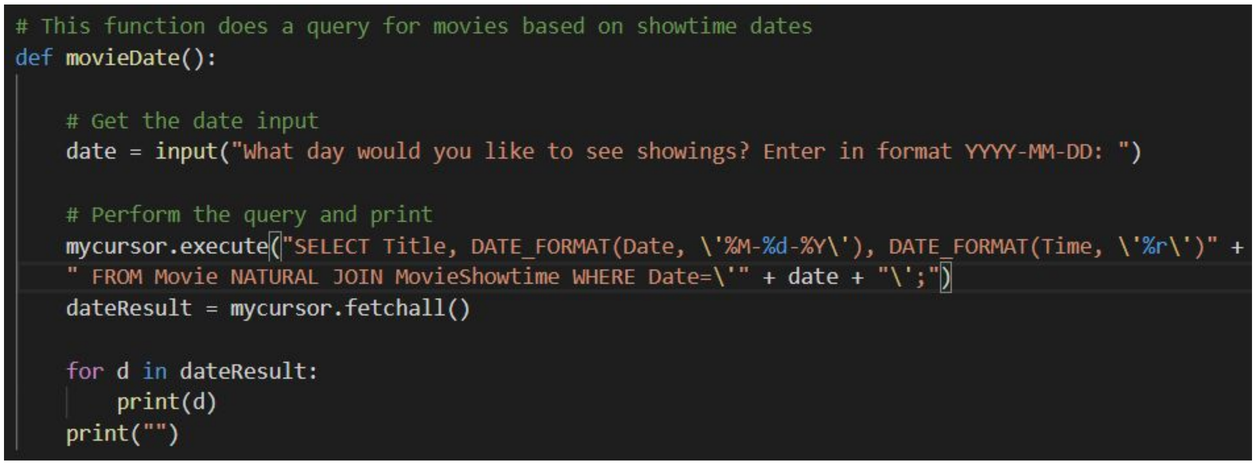
```
while True:
    print("---------------------------------------------------------------------------------------------")
    print("1. Find movies based on showtime dates")
    print("2. Find showtimes for movie + format")
    print("3. Find the most watched movie based on your favorite genre")
    print("4. Find the unavailable auditorium at a specific date and time")
    print("5. Find the most watched screen type for a specific movie")
    print("6. Buy a ticket")
    print("7. Find the customer of the month")
    print("8. Tell us how many times you've watched a certain movie")
    print("9. Exit")
    print("---------------------------------------------------------------------------------------------\n")

    menu_number = int(input("Which option would you like to do? "))
    print("")
    if menu_number == 1:
        movieDate()
    elif menu_number == 2:
        movieFormatShowtime()
    elif menu_number == 3:
        mostWatched()
    elif menu_number == 4:
        availableAuditorium()
    elif menu_number == 5:
        mostScreenType()
    elif menu_number == 6:
        buyTicket(custLogin, custPass)
    elif menu_number == 7:
        monthCustomer()
    elif menu_number == 8:
        timesWatched(custID)
    elif menu_number == 9:
        print("Goodbye! Come to this movie theatre again!")
        break
```



```
---------------------------------------------------------------------------------------------
1. Find movies based on showtime dates
2. Find showtimes for movie + format
3. Find the most watched movie based on your favorite genre
4. Find the unavailable auditorium at a specific date and time
5. Find the most watched screen type for a specific movie
6. Buy a ticket
7. Find the customer of the month
8. Tell us how many times you've watched a certain movie
9. Exit
---------------------------------------------------------------------------------------------
```

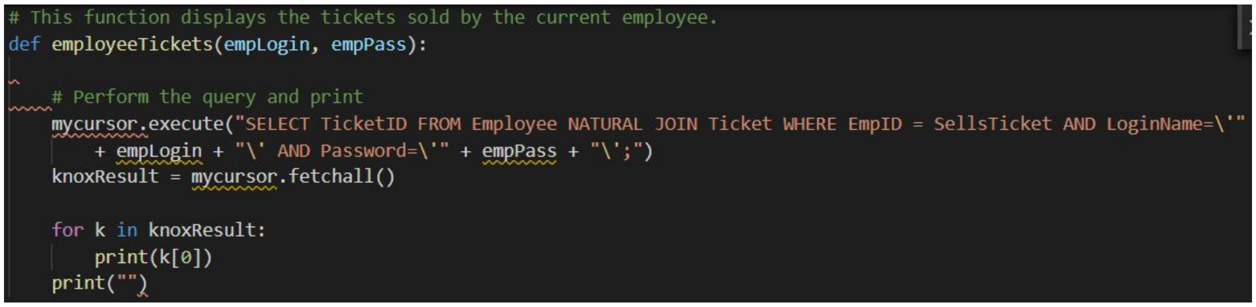
Main Queries

We have 9 queries so far to demonstrate the linkage between our text-based interface and database. The first query shows movie titles and their showtime dates and time (in a tuple) based on an input date from the user. The function below uses the given date input to perform the SQL query on the cursor ‘mycursor’.

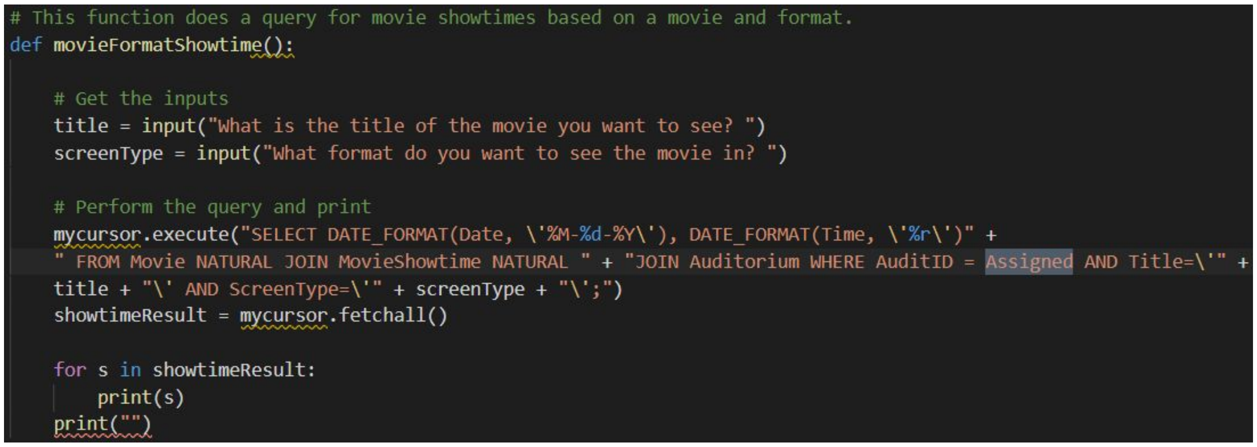


```python
# This function does a query for movies based on showtime dates
def movieDate():

    # Get the date input
    date = input("What day would you like to see showings? Enter in format YYYY-MM-DD: ")

    # Perform the query and print
    mycursor.execute("SELECT Title, DATE_FORMAT(Date, \'%M-%d-%Y\'), DATE_FORMAT(Time, \'%r\')" +
    " FROM Movie NATURAL JOIN MovieShowtime WHERE Date=\'" + date + "\';")
    dateResult = mycursor.fetchall()

    for d in dateResult:
        print(d)
    print("")
```

The next query just lists the TicketIDs sold by the current logged-in employee. The following query allows a user to select a movie and format, which will display the dates and times for each showtime that matches.

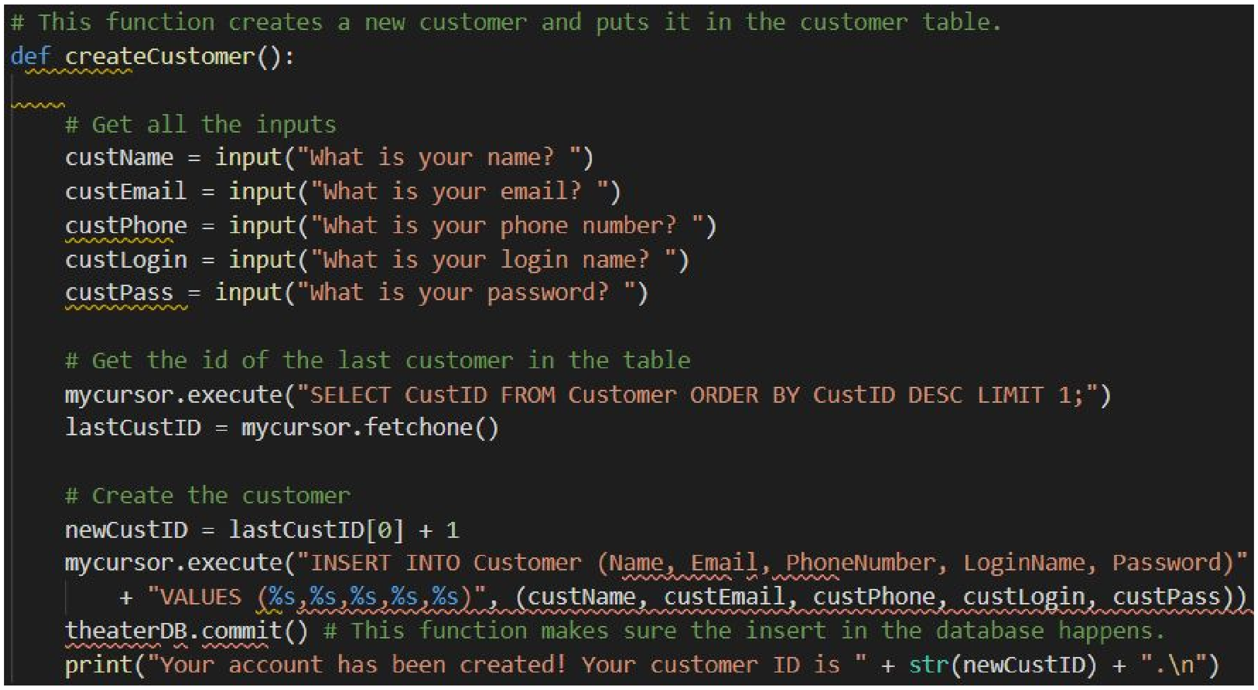


```python
# This function displays the tickets sold by the current employee.
def employeeTickets(empLogin, empPass):

    # Perform the query and print
    mycursor.execute("SELECT TicketID FROM Employee NATURAL JOIN Ticket WHERE EmpID = SellsTicket AND LoginName=\'"
        + empLogin + "\' AND Password=\'" + empPass + "\';")
    knoxResult = mycursor.fetchall()

    for k in knoxResult:
        print(k[0])
    print("")
```

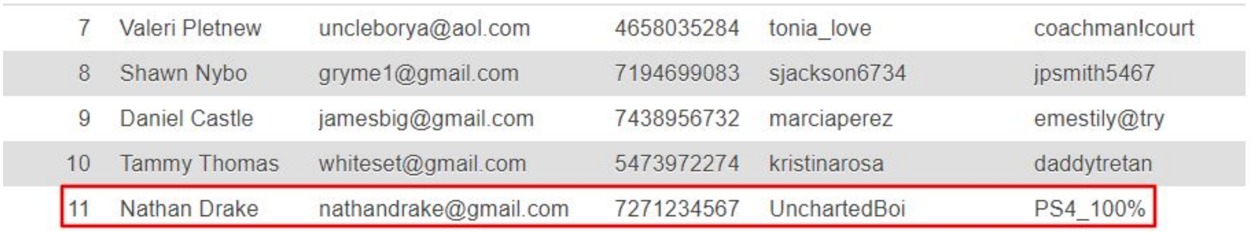


```python
# This function does a query for movie showtimes based on a movie and format.
def movieFormatShowtime():

    # Get the inputs
    title = input("What is the title of the movie you want to see? ")
    screenType = input("What format do you want to see the movie in? ")

    # Perform the query and print
    mycursor.execute("SELECT DATE_FORMAT(Date, \'%M-%d-%Y\'), DATE_FORMAT(Time, \'%r\')" +
    " FROM Movie NATURAL JOIN MovieShowtime NATURAL " + "JOIN Auditorium WHERE AuditID = Assigned AND Title=\'" +
    title + "\' AND ScreenType=\'" + screenType + "\';")
    showtimeResult = mycursor.fetchall()

    for s in showtimeResult:
        print(s)
    print("")
```

The next query creates a new customer based on inputs entered by the customer, which can be seen in the function below. We receive the last CustID in the customer's table as the new customer's ID will be one more than the last. We then perform an insert into the customer table and a commit on the database. An update to the MySQL database occurs in the following image.

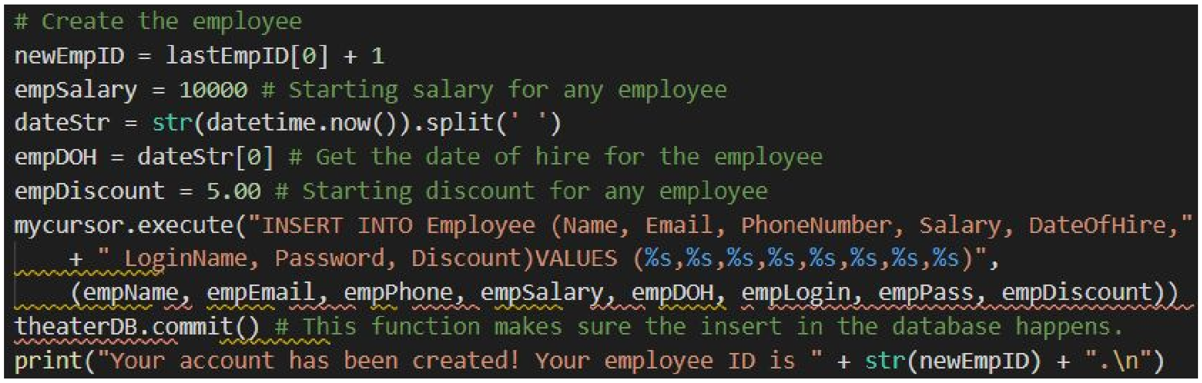


```python
# This function creates a new customer and puts it in the customer table.
def createCustomer():

    # Get all the inputs
    custName = input("What is your name? ")
    custEmail = input("What is your email? ")
    custPhone = input("What is your phone number? ")
    custLogin = input("What is your login name? ")
    custPass = input("What is your password? ")

    # Get the id of the last customer in the table
    mycursor.execute("SELECT CustID FROM Customer ORDER BY CustID DESC LIMIT 1;")
    lastCustID = mycursor.fetchone()

    # Create the customer
    newCustID = lastCustID[0] + 1
    mycursor.execute("INSERT INTO Customer (Name, Email, PhoneNumber, LoginName, Password)"
        + "VALUES (%s,%s,%s,%s,%s)", (custName, custEmail, custPhone, custLogin, custPass))
    theaterDB.commit() # This function makes sure the insert in the database happens.
    print("Your account has been created! Your customer ID is " + str(newCustID) + ".\n")
```



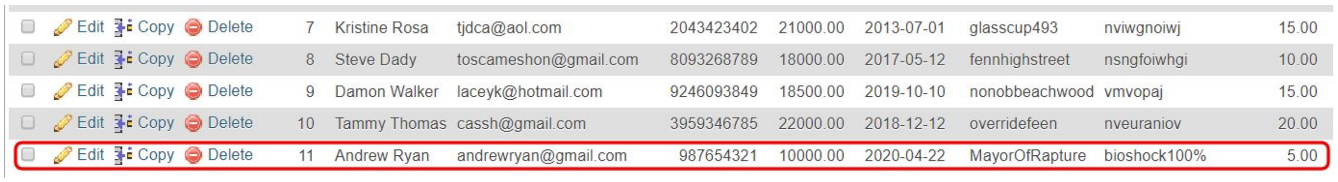
7	Valeri Pletnew	uncleborya@aol.com	4658035284	tonia_love	coachman!court
8	Shawn Nybo	gryme1@gmail.com	7194699083	sjackson6734	jpsmith5467
9	Daniel Castle	jamesbig@gmail.com	7438956732	marciaperez	emestily@try
10	Tammy Thomas	whiteset@gmail.com	5473972274	kristinarosa	daddytretan
11	Nathan Drake	nathandrake@gmail.com	7271234567	UnchartedBoi	PS4_100%

Of course it also made sense to implement a createEmployee() function for that query. It has the same interface as the createCustomer() function as it asks the exact same questions. However, the python code is a little different because three more attributes are required by the Employee table as seen below.



```python
# Create the employee
newEmpID = lastEmpID[0] + 1
empSalary = 10000 # Starting salary for any employee
dateStr = str(datetime.now()).split(' ')
empDOH = dateStr[0] # Get the date of hire for the employee
empDiscount = 5.00 # Starting discount for any employee
mycursor.execute("INSERT INTO Employee (Name, Email, PhoneNumber, Salary, DateOfHire,"
    + " LoginName, Password, Discount)VALUES (%s,%s,%s,%s,%s,%s,%s,%s)",
    (empName, empEmail, empPhone, empSalary, empDOH, empLogin, empPass, empDiscount))
theaterDB.commit() # This function makes sure the insert in the database happens.
print("Your account has been created! Your employee ID is " + str(newEmpID) + ".\n")
```

The new employee’s salary would start at 10000 annually and their discount would start at $5. Getting the date of hire was somewhat harder as you need a current timestamp. After importing datetime into the python file and making the right substring from the



Edit	Copy	Delete	7	Kristine Rosa	tjdca@aol.com	2043423402	21000.00	2013-07-01	glasscup493	nviwgnoiwj	15.00
Edit	Copy	Delete	8	Steve Dady	toscameshon@gmail.com	8093268789	18000.00	2017-05-12	fennhighstreet	nsngfoiwhgi	10.00
Edit	Copy	Delete	9	Damon Walker	laceyk@hotmail.com	9246093849	18500.00	2019-10-10	nonobbeachwood	vmvopaj	15.00
Edit	Copy	Delete	10	Tammy Thomas	cassh@gmail.com	3959346785	22000.00	2018-12-12	overridefeen	nveuraniov	20.00
Edit	Copy	Delete	11	Andrew Ryan	andrewryan@gmail.com	987654321	10000.00	2020-04-22	MayorOfRapture	bioshock100%	5.00

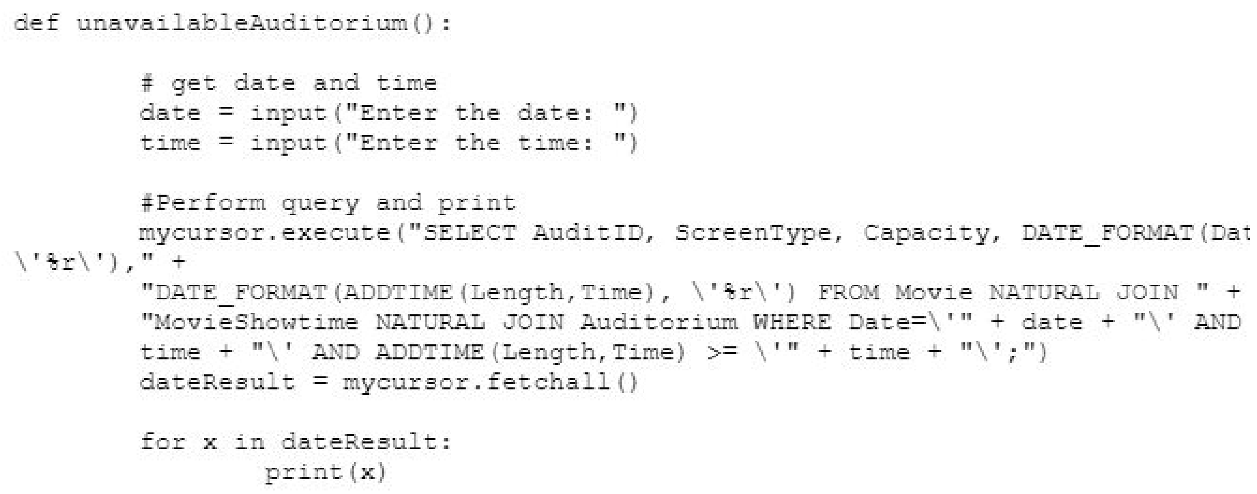
datetime object, we were able to get the date that the employee was created. As seen in the image above, the new employee was successfully inserted into the MySQL database.

The next query mostWatched() returns the most watched movie for whichever genre is specified by the user.

```
def mostWatched():

        # Get the genre
        genre = input("What is your favorite movie genre? ")
        # Perform the query and print
        mycursor.execute("SELECT Title FROM Ticket NATURAL JOIN Movie NATU
        genre + "\' GROUP BY Title ORDER BY COUNT(Title) DESC LIMIT 1;")
        genresResult = mycursor.fetchone()

        print("The most watched " + genre + " movie is ")
        for x in genresResult:
                print(x)
```

As you can read from the snippet of code above the method takes the user input into the variable genre, returns all the movies from the tickets where category matches the variable genre, groups by movie title, and returns the title of the movie that has been watched the most. This query would be valuable to find which movies are selling best, which genre is selling best, and to find max ticket sales for each genre.

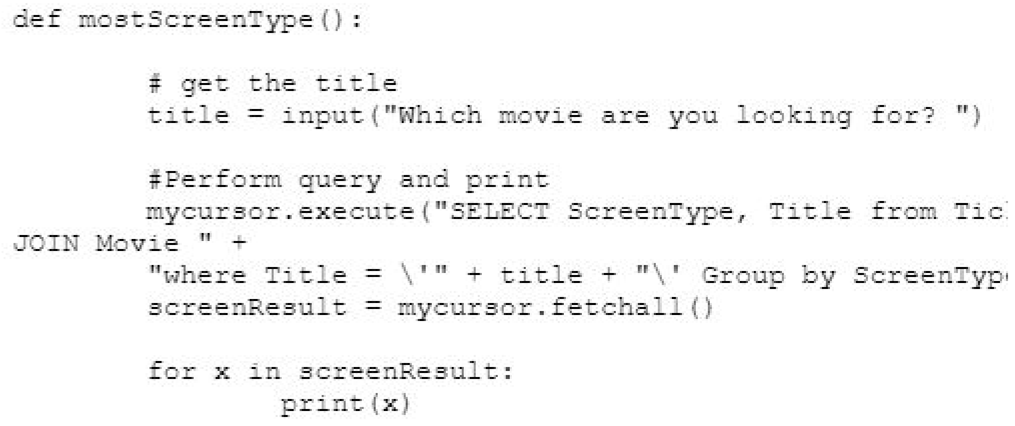
The next query unavailableAuditorium() finds all the unavailable auditoriums, that is the auditoriums that are currently in use for certain movies, for a user-inputted time and date.



```
def unavailableAuditorium():

        # get date and time
        date = input("Enter the date: ")
        time = input("Enter the time: ")

        #Perform query and print
        mycursor.execute("SELECT AuditID, ScreenType, Capacity, DATE_FORMAT(Dat
\'%r\')," +
        "DATE_FORMAT(ADDTIME(Length,Time), \'%r\') FROM Movie NATURAL JOIN " +
        "MovieShowtime NATURAL JOIN Auditorium WHERE Date=\'" + date + "\' AND
        time + "\' AND ADDTIME(Length,Time) >= \'" + time + "\';")
        dateResult = mycursor.fetchall()

        for x in dateResult:
                print(x)
```

This query stores the user input into the date and time queries, selects all the movies by joining the movies, showtimes, and auditoriums; and it selects those that have date attributes that match the user inputted date, and time attributes that fall within the interval of the movie's time interval. This query would be very useful for managing employees who are trying to schedule the various movies in the theater.

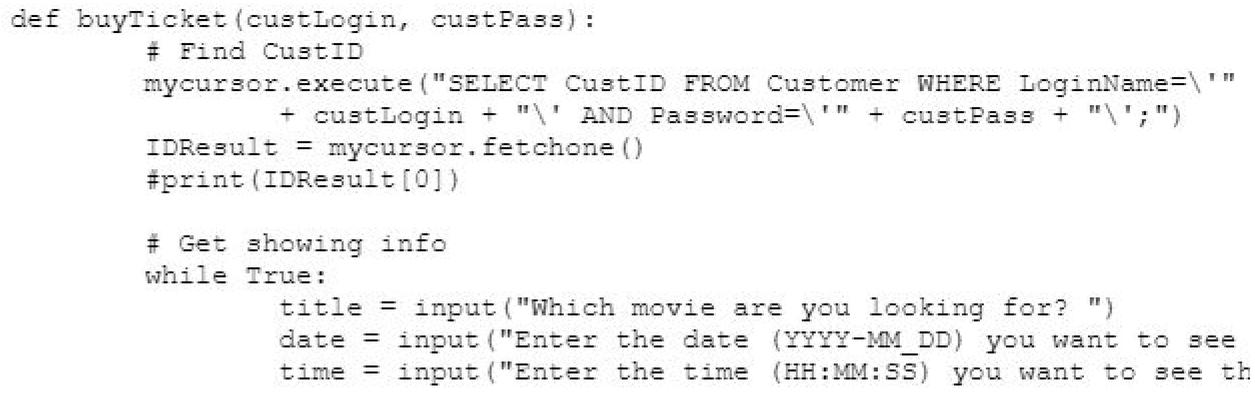
The next method mostScreenType() reads in the user specified movie, and returns the most commonly used screen type for that movie.



```
def mostScreenType():

        # get the title
        title = input("Which movie are you looking for? ")

        #Perform query and print
        mycursor.execute("SELECT ScreenType, Title from Tic
JOIN Movie " +
        "where Title = \'" + title + "\' Group by ScreenTyp
        screenResult = mycursor.fetchall()

        for x in screenResult:
                print(x)
```

The query reads in the title in the variable title, then selects the screen types and movie titles from all movies where the movie title matches the title variable, groups by screen type, and then returns the screen type with the most elements in the grouping. This query could be useful by movie companies who ask the theater which screen that their movie is being played on the most. This could be potentially useful to customers who want to know what screen they should view the movie on based on the most popular screen type.

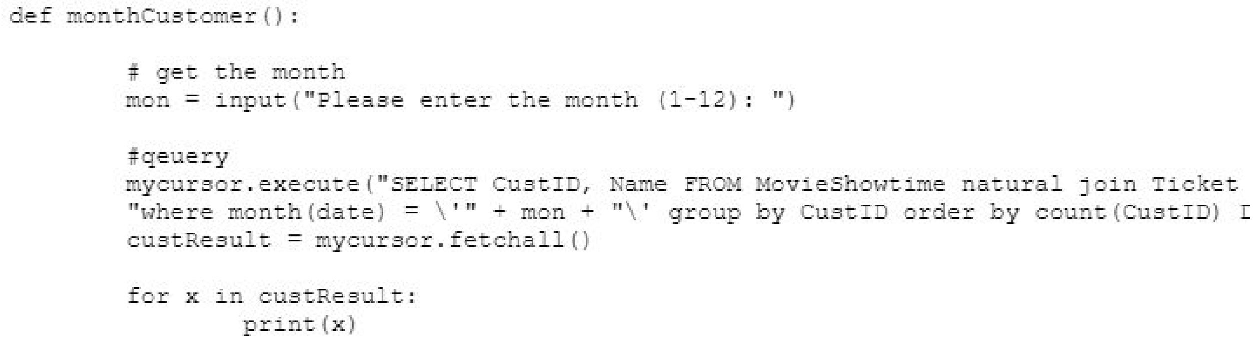
The next query buyTicket() uses the user inputted movie title, date, and time to buy a ticket for the user. Since this ticket needs to update the user table the code is very long and only a snippet for gathering the user input is shown below.



```
def buyTicket(custLogin, custPass):
        # Find CustID
        mycursor.execute("SELECT CustID FROM Customer WHERE LoginName=\'"
                + custLogin + "\' AND Password=\'" + custPass + "\';")
        IDResult = mycursor.fetchone()
        #print(IDResult[0])

        # Get showing info
        while True:
                title = input("Which movie are you looking for? ")
                date = input("Enter the date (YYYY-MM_DD) you want to see
                time = input("Enter the time (HH:MM:SS) you want to see th
```

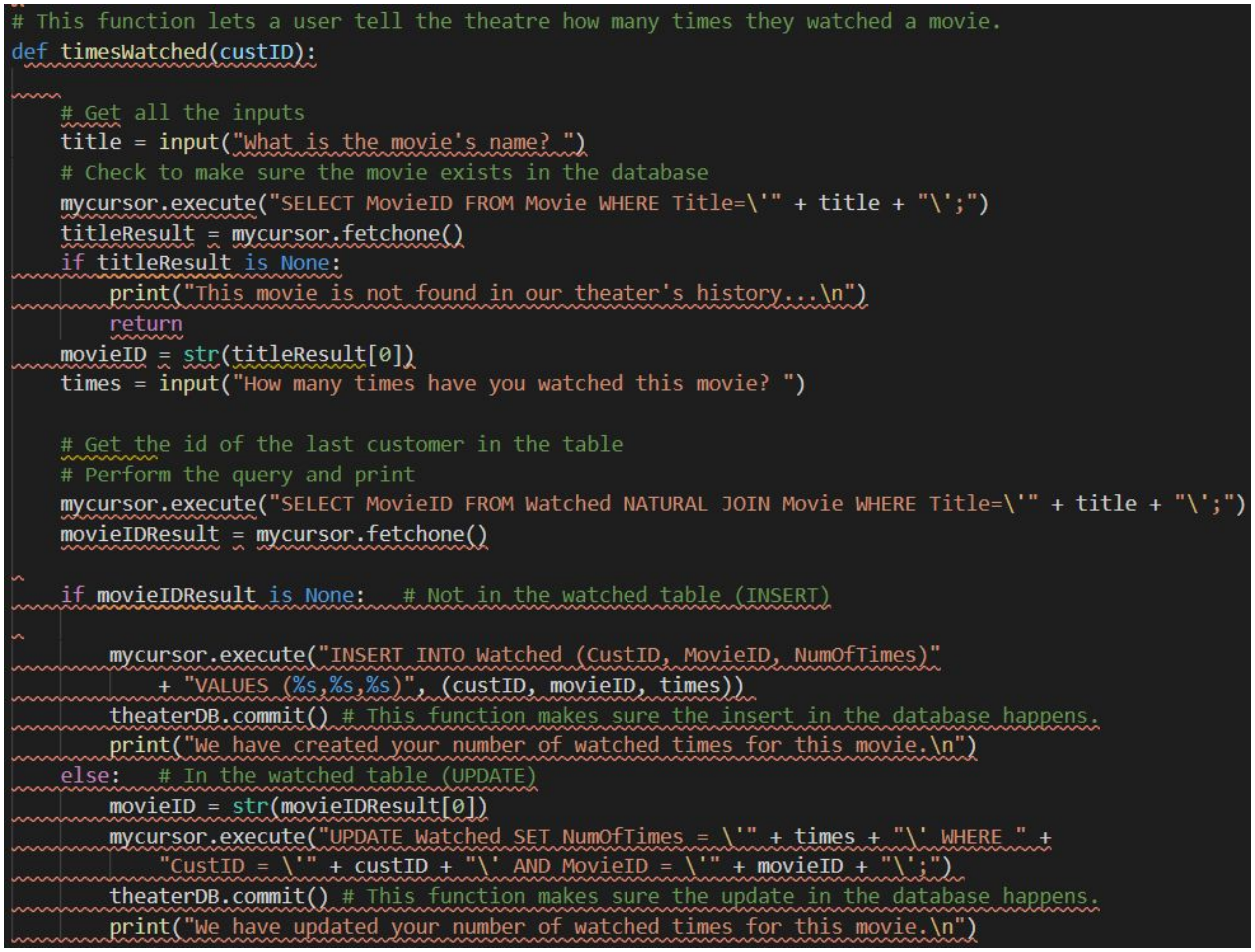
This method first finds if the movie is available at the given time. If a movie is playing at that time it goes on to the next step, but if the movie is not playing at that time it returns that no showing is available. Then the method will prompt the user for a seat number, if the seat is unavailable it will return unavailable and then ask for a new seat number. If no seats are available the program will return with the error of a full theater. If an available seat is found then the database makes a ticket ID for the ticket with the next available TID number, creates a new ticket, and inserts the ticket into the ticket table. This is very practical for a movie database system so that customers can purchase tickets and so that the database will update.

The final two queries monthCustomer() and monthEmployee() return the “customer of the month” and “employee of the month” based on whichever customers bought the most tickets and whichever employee sold the most tickets in the user inputed month (1-12).



```
def monthCustomer():

        # get the month
        mon = input("Please enter the month (1-12): ")

        #qeuery
        mycursor.execute("SELECT CustID, Name FROM MovieShowtime natural join Ticket
        "where month(date) = \'" + mon + "\' group by CustID order by count(CustID) I
        custResult = mycursor.fetchall()

        for x in custResult:
                print(x)
```

These methods are separate but are so similar I will talk about them together. The method monthCustomer() selects the tables movieShowtimes and Tickets, where the month equals the user inputted mon variable, groups by customer ID, and returns the customer ID, and name from the grouping with the largest number of tickets. The method monthEmployee() selects the tables movieShowtimes and Tickets, where the month equals the user inputted mon variable, groups by employee ID for who sold the ticket, and returns the employee ID, and name from the grouping with the largest number of tickets. This would be important for movie managers who wanted to congratulate the employee and customer who bought the most tickets.



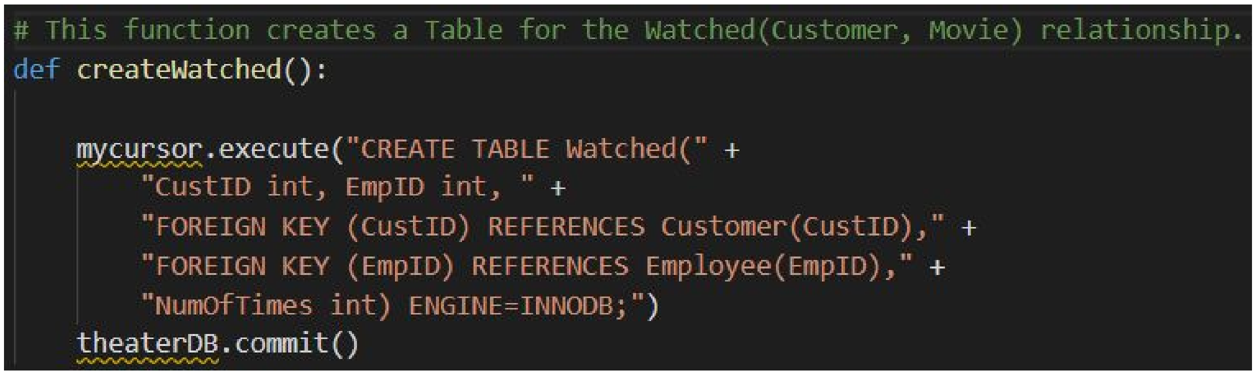
```
# This function lets a user tell the theatre how many times they watched a movie.
def timesWatched(custID):

    # Get all the inputs
    title = input("What is the movie's name? ")
    # Check to make sure the movie exists in the database
    mycursor.execute("SELECT MovieID FROM Movie WHERE Title=\'" + title + "\';")
    titleResult = mycursor.fetchone()
    if titleResult is None:
        print("This movie is not found in our theater's history...\n")
        return
    movieID = str(titleResult[0])
    times = input("How many times have you watched this movie? ")

    # Get the id of the last customer in the table
    # Perform the query and print
    mycursor.execute("SELECT MovieID FROM Watched NATURAL JOIN Movie WHERE Title=\'" + title + "\';")
    movieIDResult = mycursor.fetchone()

    if movieIDResult is None:   # Not in the watched table (INSERT)

        mycursor.execute("INSERT INTO Watched (CustID, MovieID, NumOfTimes)"
            + "VALUES (%s,%s,%s)", (custID, movieID, times))
        theaterDB.commit() # This function makes sure the insert in the database happens.
        print("We have created your number of watched times for this movie.\n")
    else:   # In the watched table (UPDATE)
        movieID = str(movieIDResult[0])
        mycursor.execute("UPDATE Watched SET NumOfTimes = \'" + times + "\' WHERE " +
            "CustID = \'" + custID + "\' AND MovieID = \'" + movieID + "\';")
        theaterDB.commit() # This function makes sure the update in the database happens.
        print("We have updated your number of watched times for this movie.\n")
```

The last query above allows a customer to enter the amount of times they have watched a movie. The movie must exist in the Movie table with a title query check. An insert into the Watched table will occur if an entry didn't exist and an update to the NumOfTimes attribute will happen if it does.

<u>One-Time Queries</u>

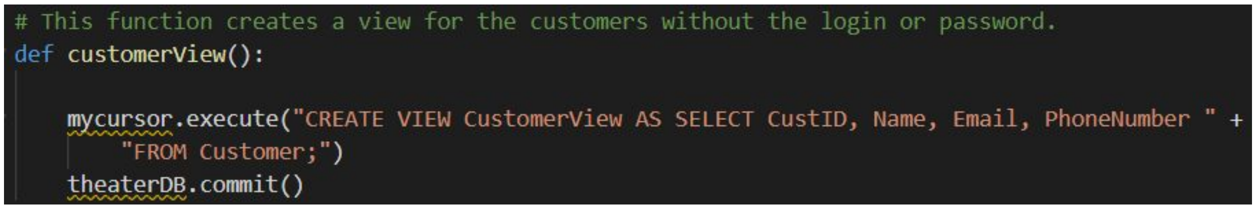
We also created queries for creating a table and views. However, we noticed that they only had to run once on any database assessor's localhost as they're creation queries.

Below is the function for creating the Watched table because it is defined as a many-to-many relationship. There are foreign keys for referencing the primary keys of the Customer and Employee tables.

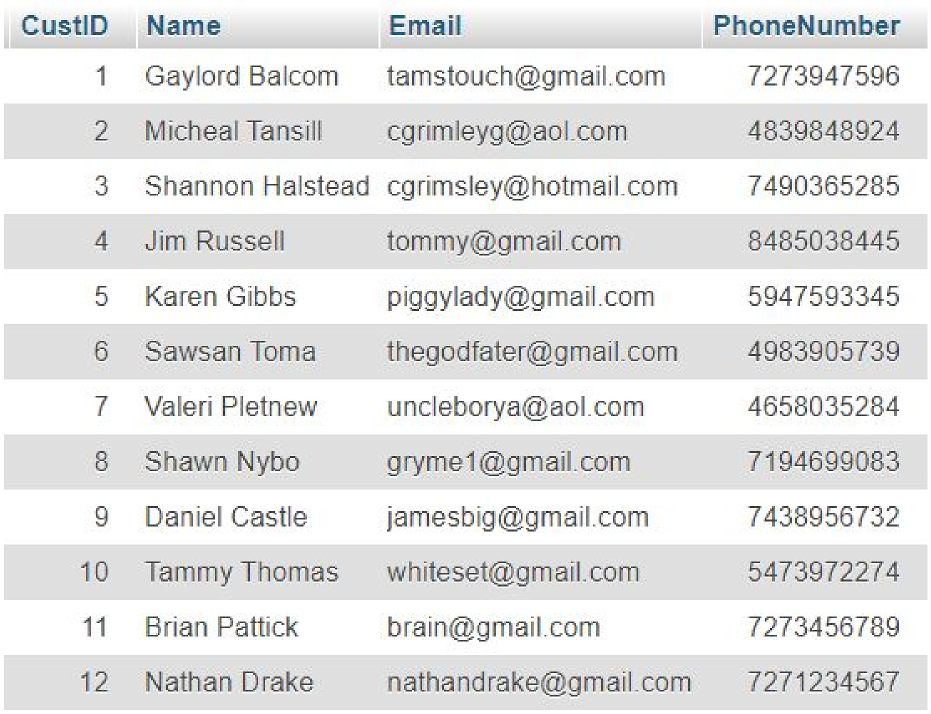


```python
# This function creates a Table for the Watched(Customer, Movie) relationship.
def createWatched():

    mycursor.execute("CREATE TABLE Watched(" +
        "CustID int, EmpID int, " +
        "FOREIGN KEY (CustID) REFERENCES Customer(CustID)," +
        "FOREIGN KEY (EmpID) REFERENCES Employee(EmpID)," +
        "NumOfTimes int) ENGINE=INNODB;")
    theaterDB.commit()
```

The next function (customerView()) below creates a view for customers by selecting the necessary attributes for this view. Similarly, the employeeView() function is an almost identical function.



```python
# This function creates a view for the customers without the login or password.
def customerView():

    mycursor.execute("CREATE VIEW CustomerView AS SELECT CustID, Name, Email, PhoneNumber " +
        "FROM Customer;")
    theaterDB.commit()
```



CustID	Name	Email	PhoneNumber
1	Gaylord Balcom	tamstouch@gmail.com	7273947596
2	Micheal Tansill	cgrimleyg@aol.com	4839848924
3	Shannon Halstead	cgrimsley@hotmail.com	7490365285
4	Jim Russell	tommy@gmail.com	8485038445
5	Karen Gibbs	piggylady@gmail.com	5947593345
6	Sawsan Toma	thegodfater@gmail.com	4983905739
7	Valeri Pletnew	uncleborya@aol.com	4658035284
8	Shawn Nybo	gryme1@gmail.com	7194699083
9	Daniel Castle	jamesbig@gmail.com	7438956732
10	Tammy Thomas	whiteset@gmail.com	5473972274
11	Brian Pattick	brain@gmail.com	7273456789
12	Nathan Drake	nathandrake@gmail.com	7271234567