Aleksas Murauskas 260718389
Pierre Venereau 260669485
Cyril Saidane 260706157
Paul Seimandi 260707808

COMP 421: Project Deliverable 3 Report

Part 1.

For this part, we decided to implement a stored procedure that could be used regularly and that would have a beneficial impact on the system. We chose to add a feature that allows us to clear all of the irrelevant screenings from the screenings list. If a screening has a date that has already passed (comparison is made with current date), then it is removed from the list. This has been done using a cursor to check for this constraint through our table.

The procedure:

```
CREATE OR REPLACE FUNCTION UPDATE_SCREENINGS (cur_date DATE)
RETURNS VOID AS $$
DECLARE c_screeningnum INT;
DECLARE c_date DATE;
DECLARE C1 CURSOR FOR
SELECT screeningnum, date FROM screening;
BEGIN
OPEN C1;
FETCH C1 INTO c_screeningnum, c_date;
WHILE FOUND LOOP
IF(c_date < cur_date)
THEN DELETE FROM screening WHERE screeningnum = c_screeningnum;
END IF;
FETCH C1 INTO c_screeningnum, c_date;
END LOOP;
CLOSE C1;
RETURN;
END
$$
LANGUAGE PLPGSQL;
```

```
SELECT update_screenings('2020-04-12');
```

Figure 1: procedure call

	movieid integer	cid integer	nbr integer	screenir integer	time character v	date date
1	323	1357	6	24	11:00 pm	1919-06-26
2	332	1370	7	27	10:00 pm	
3	341	1383	7	30	11 am	1919-09-24
4	350	1396	1	33	9:30 pm	2121-02-01
5	359	1409	10	36	7:30 pm	2020-01-07
6	368	1422	5	39	5:00 pm	1919-12-06
7	377	1435	6	42	9:45 pm	1919-03-30
8	386	1448	5	45	11 am	1919-12-07
9	395	1461	5	48	1:00 pm	2020-07-27
10	404	1474	10	51	4:30 pm	2020-09-19
11	413	1487	1	54	11:00 pm	1919-05-17
12	422	1500	2	57	10:30 am	2020-02-11
13	431	1513	10	60	7:00 pm	2020-03-22
14	440	1526	4	63	2:00 pm	1919-12-24
15	449	1539	6	66	1:00 pm	1919-11-02
16	458	1552	10	69	7:30 pm	2020-09-11
17	467	1565	7	72	8:30 pm	1919-12-18
18	476	1578	1	75	5:00 pm	1919-09-09
19	485	1591	7	78	5:00 pm	2020-03-01
20	494	1604	10	81	10:30 am	2121-02-06
21	503	1617	7	84	1:00 pm	1919-10-12
22	512	1630	1	87	4:00 pm	1919-12-05
23	521	1643	6	90	10:30 pm	2121-01-15
24	530	1656	9	93	12:30 pm	1919-10-26
25	539	1669	4	96	9:00 pm	1919-10-25

Figure 2: screening table before procedure

	movie integer	cid integer	nbr integer	screer [PK] ir	time character	date date
1	350	1396	1	33	9:30 pm	2121-02-01
2	395	1461	5	48	1:00 pm	2020-07-27
3	404	1474	10	51	4:30 pm	2020-09-19
4	458	1552	10	69	7:30 pm	2020-09-11
5	494	1604	10	81	10:30 a...	2121-02-06
6	521	1643	6	90	10:30 ...	2121-01-15
7	584	1734	3	111	3:45 pm	2020-12-30
8	593	1747	9	114	10:30 ...	2020-09-27
9	602	1760	9	117	2:00 pm	2020-12-19
10	683	1877	1	144	1:00 pm	2020-08-07
11	701	1903	2	150	4:30 pm	2020-08-23
12	710	1916	1	153	8:00 pm	2121-02-01
13	728	1942	9	159	11:30 ...	2020-04-13
14	737	1955	8	162	1:00 pm	2020-08-05
15	746	1968	6	165	8:00 pm	2020-09-17

Figure 3: screening table after procedure

Part 2.

For part two we created a Python script that prompts the user to select from a menu of options, once prompted, the user can choose from 5 queries that modify or return data from the database. Once the respective number 1-5, is entered the program moves to a method that executes that specific query. To exit, 10 can be entered at the main menu prompt and after a query option terminates, the user is prompted if they would like to continue. The query will be printed alongside the results retrieved from the database.

```
def display_menu():
    print("------------------Program Menu-----------------")
    print("Enter the associted number to begin a process:")
    print("0: Redisplay Menu")
    print("1: Create a new Employee")
    print("2: What movie screenings are on this date")
    print("3: How many cinemas are have screened a certain movie")
    print("4: How many Items has a customer purchased ")
    print("5: Show which cinemas have a room larger than x seats")
    print("10: Exit Program")
```

Figure 3: Code to display menu Menu Display

For all queries there are commented lines that allow user inputs, in order to input user fields for queries. For simplicity of testing database connection, there are also hard coded values. In order to test the program with user inputs.

Query 1: Create a new Employee

This query will insert a new tuple into the employee table. It will first prompt the user for all the inputs within the employee tuple, and then query the system. If the query is successful it will return the eid of the inserted employee.

```
def processOne(connection): #Make a New Employee
    print("What is their name?")
    #new_name= input()
    new_name= "Aleks"
    print("What is their address?")
    #new_add= input()
    new_add= "3434 St. Famille"
    print("What is their new email?")
    #new_email= input()
    new_email= "avm@avm.ca"
    new_add= "3434 St. Famille"
    print("Where do they work? please enter a cid")
    #new_cid= input()
    new_cid= "1461"
    print("What is new employeeID?")
    #new_eid= input()
    new_eid= "11111"
    print("What is their starting salary?")
    #new_sal= input()
    new_sal= "500000"
    Query ="INSERT INTO employee(eid, cid, name, email, salary, address) VALUES('"
    Query_with_val = Query+new_eid+"', '"+new_cid+"', '"+new_name+"', '"+new_email+"','"+new_sal+"', '"+new_add+"') RETURNING eid;"
    #dat1 = pd.read_sql_query(Query_fin, connection)
    dat1 = pd.read_sql_query(Query_with_val, connection)
    dat1.set_index(['eid'])
    print(dat1)
    print(Query_with_val)
```

Figure 4:Method code that executes query 1

```
------------------Program Menu-----------------
Enter the associted number to begin a process:
0: Redisplay Menu
1: Create a new Employee
2: What movie screenings are on this date
3: How many cinemas are have screened a certain movie
4: How many Items has a customer purchased
5: Show which cinemas have a room larger than x seats
10: Exit Program
Input: 1
What is their name?
Aleks
What is their address?
3434 rue University
What is their new email?
acm@avm.ca
Where do they work? please enter a cid
1461
What is new employeeID?
111111
What is their starting salary?
50000
      eid
0  111111
INSERT INTO employee(eid, cid, name, email, salary, address) VALUES('111111', '1461', 'Aleks', 'acm@avm.ca','50000', '3434 rue University') RETURNING eid;
Do you wish to continue? (Y/N)
```

Figure 5: Console view of Query 1 Execution

Query 2: What Movie screenings on date

This query will return the name of movies screened on a certain date. It will first prompt the user for the date. The query joins screening and movie on screening id and then selects tuples that have the correct date. The query will then execute and return a list of movie titles that fit the criteria.

```
def processTwo(connection): #What movie titles are screening on this date
    print("What is the date you would like to select")
    #in_date =str(input())
    in_date ="2121-02-01"
    query = "SELECT name, date from screening s, movie m where s.movieid = m.movieid and date ='2121-02-01'"
    q_input = "SELECT name from screening s, movie m where s.movieid = m.movieid and date ="
    q_fin = q_input+"'"+in_date+"';"

    dat1 = pd.read_sql_query(q_fin, connection)
    dat1.set_index(['name'])
    print("Movie Titles:\n")
    print(dat1)
    print(q_fin)
```

Figure 6:Method code that executes query 2

```
-----------------Program Menu-----------------
Enter the associted number to begin a process:
0: Redisplay Menu
1: Create a new Employee
2: What movie screenings are on this date
3: How many cinemas are have screened a certain movie
4: How many Items has a customer purchased
5: Show which cinemas have a room larger than x seats
10: Exit Program
Input: 2
What is the date you would like to select
2121-02-01
Movie Titles:

                                    name
0  ultrices. Vivamus rhoncus. Donec est.
1    id enim. Curabitur massa. Vestibulum
SELECT name from screening s, movie m where s.movieid = m.movieid and date ='2121-02-01';
Do you wish to continue? (Y/N)
```

Figure 7: Console view of Query 2 Execution

Query 3: How many cinemas have screened a specific movie

The method will prompt the user for a movie title that it would like to see the number of screenings of. The query joins screening and movie on movieid, and then selects from the tuple where the name is like the inputted movie name, then the number of matching tuples are counted. The query will execute and return the Number of cinemas that have screened the movie.

```
def processThree(connection): #How many cinemas are have screened a movie
    print("What movie are you looking for?")
    #in_date =str(input())
    movie ="quis urna. Nunc quis arcu"
    query = "SELECT count(cid) from screening s, movie m  where m.name like '%"+movie+"%' and m.movieid=s.movieid"

    dat1 = pd.read_sql_query(query, connection)
    print(dat1)
    print("Number of cinemas that screened the movie:\n")
    print(dat1['count'][0])
    print(query)
```

Figure 8:Method code that executes query 3

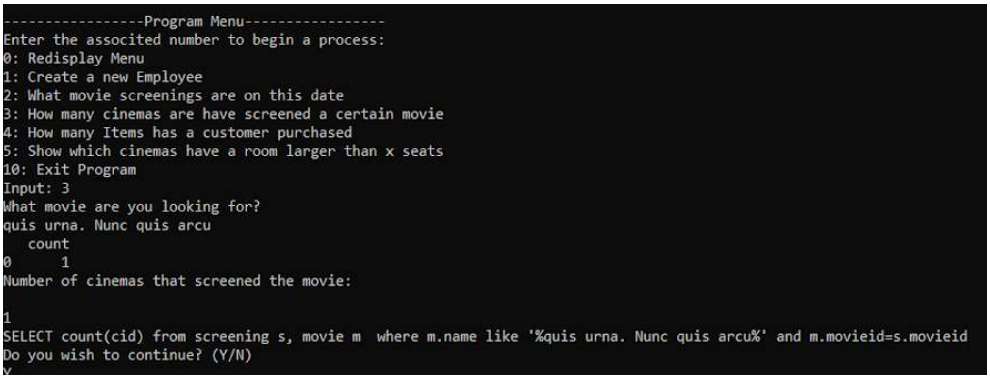


Figure 9: Console view of Query 3 Execution

Query 4: how many items a customer has purchased.

First the method requests the user input the customer's email. The Query will join orders and customers on custid, and select where custemail equals the email inputted by the user. The quantity column of the tuple is then summed and returned. Then output of the query is printed to the console.

```
def processFour(connection): # How many Items has a customer purchased the
    print("Email of the customer?")
    #email =str(input())
    email='Quisque@sodales.co.uk'
    query="SELECT SUM(quantity) from orders o, customer c where o.custid=c.custid and c.custemail='"+email+"'"
    dat1 = pd.read_sql_query(query, connection)
    print('Total items bought')
    print(dat1['sum'][0])
    print(query)
```

Figure 10:Method code that executes query 4

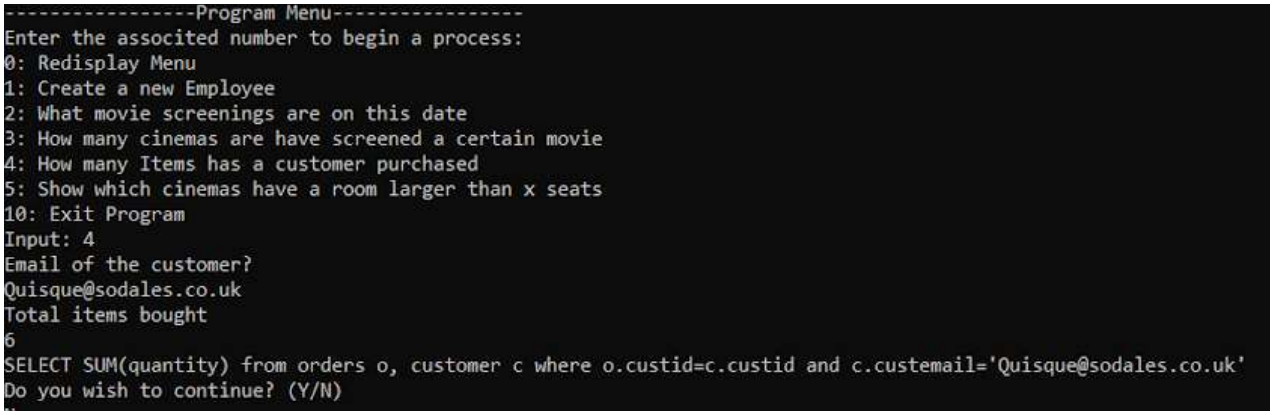


Figure 11: Console view of Query 4 Execution

Query 5: What Cinemas have rooms with more than a inputted number of seats

First the method prompts the user to enter the desired number of seats. The query joins cinema and room on cid, and selects from the tuple where the room's capacity is larger than the value the user inputted. Address, room_nb and capacity are projected from the join and printed to the console.

```
def processFive(connection): #Show which cinemas have a room larger than 260 seats

    print("What size of room are you looking for?")
    cap=input()
    #cap=280
    query= """select address, nbr as room_nb, capacity from cinema c, room r
            where c.cid=r.cid
            and r.capacity > """+str(cap)

    dat1 = pd.read_sql_query(query, connection)
    print(dat1.keys())
    dat1.set_index(['address', 'room_nb', 'capacity'])
    print("Cinemas:\n")
    print(dat1)
    print(query)
```

Figure 7:Method code that executes query 5

```
----------------Program Menu----------------
Enter the associted number to begin a process:
0: Redisplay Menu
1: Create a new Employee
2: What movie screenings are on this date
3: How many cinemas are have screened a certain movie
4: How many Items has a customer purchased
5: Show which cinemas have a room larger than x seats
10: Exit Program
Input: 5
What size of room are you looking for?
280
Index(['address', 'room_nb', 'capacity'], dtype='object')
Cinemas:

                          address  room_nb  capacity
0       Ap #717-1829 Sociis Street        5       288
1                298-1479 Proin Rd.        1       296
2   P.O. Box 332, 7721 Ante, Avenue        2       294
3      P.O. Box 690, 6579 Vitae Rd.        8       292
4            561-4584 Feugiat. Road        3       292
5            172-7706 Sagittis Road        3       289
6     P.O. Box 265, 8907 Morbi Road        4       282
7                  6411 Proin St.         8       294
8             780-641 Fermentum Rd.        2       287
9                7937 Maecenas Road        6       299
10          Ap #953-9720 Proin Ave        4       284
11                 6703 Ut Street        4       298
select address, nbr as room_nb, capacity from cinema c, room r
                    where c.cid=r.cid
                    and r.capacity > 280
Do you wish to continue? (Y/N)
```

Figure 5: Console view of Query 5 Execution

Part 3.

1) CREATE INDEX ind_screening_date ON Screening(date)

This index will be useful when running our second process from question 2 for example. We will be able to increase the efficiency of the query because the index allows us to sort the screenings by date in increasing or decreasing order. Once this is done, looking for movies with a specific date will be much simpler since we won't have to go through an entire unordered list.

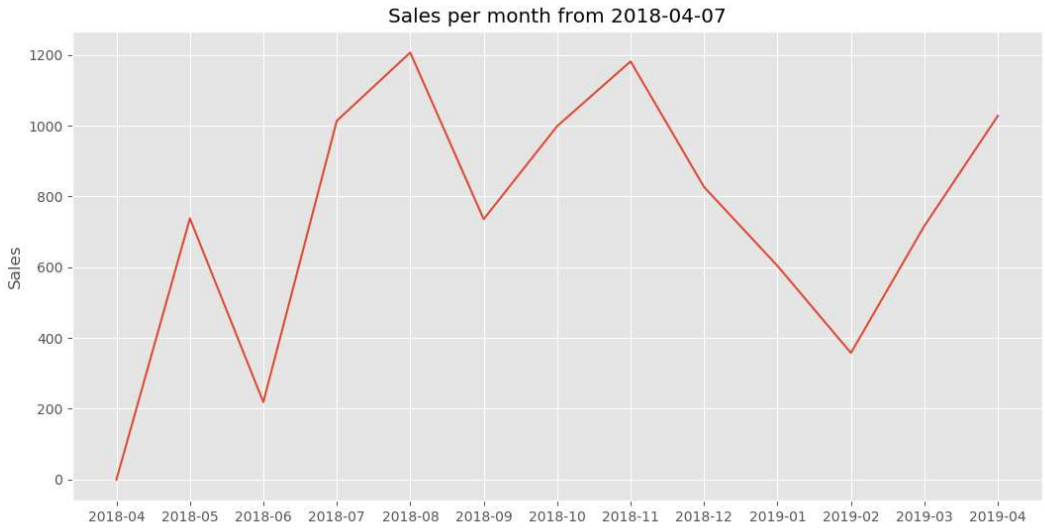
2) CREATE INDEX ind_customer_id ON Orders(custid)

With this index, we will be able to sort the orders based on customer IDs. The use of this index will therefore shorten some queries that deal with that data. It is notably the case of our fourth process from question 2 where we want to calculate the total number of items bought by a particular customer. By sorting the orders with the index, we have a much simpler search since all orders from the same customer will be in sequence in the linked list.

Part 4.

1) Sales per months from a given date

The first visualization we wanted to implement was the evolution of sales within the cinema from a given date.



The code used for this chart is found below. It incorporates a simple UI prompting the user for the date he wants to start the visualization from.

Sql used to generate the data:

"""select price, date, quantity from product p, orders o where p.productid=o.productid and date>""" + date + """ order by date"""

Where date is the python variable referencing to what the user inputted.

```
def main():
    passw = "Cinema2078"
    # print("please enter the password for cs421g78:")
    # pw= str(input())
    connection = psycopg2.connect(user="cs421g78",
                                  password=passw,
                                  host="comp421.cs.mcgill.ca",
                                  port="5432",
                                  database="cs421")
    profitsPerMonth(connection)

def profitsPerMonth(connection):
    print('Enter date to look from')
    date = input()
    query = """select price, date, quantity from product p, orders o
     where p.productid=o.productid and date>'""" + date + """' order by date"""
    dat1 = pd.read_sql_query(query, connection)

    connection.close()
    print(dat1)
    plotarray1 = []
    for index, elem in dat1.iterrows():
        print(elem)
        plotarray1.append([str(elem['date']), int(elem['price']) * int(elem['quantity'])])
    print(plotarray1)
    curmonth = str(date[:-3])
    month = []
    profit = []
    profitc = 0
    for elem in plotarray1:
        print(str(elem[0][:-3]))
        print(str(curmonth))
        print(elem[0][-5:-2] == curmonth)
        if (elem[0][-5:-2] == curmonth):
            print(elem)
            profitc += elem[1]
            print(profitc)
        else:
            print('flag')
            curmonth = elem[0][-5:-2]
            profit.append(profitc)
            profitc = elem[1]
            month.append(elem[0][:-3])
    print(month)
    print(profit)
    plt.style.use('ggplot')
    plt.plot(month, profit)
    plt.title("Sales per month from " + date)
    plt.ylabel("Sales")
    plt.show()
if __name__ == "__main__":
    main()
```

2) Secondly we decided to find the number of movies by rating (rounding ratings up to 1 decimal) to display roughly the distribution of movies per rating:

Sql used to generate the data:

CREATE TABLE movie_rated AS SELECT ROUND(CAST(rating AS numeric), 1), count(name) from movie group by ROUND(CAST(rating AS numeric), 1) order by ROUND

Chart: