Database Final Project Description

1. Operation Steps:

* Start your MySQL Database Server and Apache Web Server. Make sure there is database called “airsystem” inside your server.
* Open terminal, change your current directory to Database\_Final.
* Type “python3 init1.py”.
* Go to your browser and type “127.0.0.1” in the address bar.

1. Use Cases Description:
2. Public Use Cases:

* View public Info (a):

Using a list to record all the information that the user has typed in the web page:

1. control\_list=[]
2. **if** dept\_airport!='':
3. control\_list.append("flight.departure\_airport = '%s'"%(dept\_airport))
4. **if** arr\_airport!='':
5. control\_list.append("flight.arrival\_airport = '%s'"%(arr\_airport))
6. **if** date!='':
7. control\_list.append("DATE(flight.departure\_time) = '%s'"%(date))
8. **if** dept\_city!='':
9. control\_list.append("dept.airport\_city = '%s'"%(dept\_city))
10. **if** arr\_city!='':
11. control\_list.append("arr.airport\_city = '%s'"%(arr\_city))

Default Condition:

1. **if** len(control\_list)==0:
2. query = "SELECT flight.arrival\_time as arrival\_time, flight.status as status, flight.flight\_num as flight\_number, flight.departure\_time as departure\_time, dept.airport\_name as departure\_airport, dept.airport\_city as departure\_city, arr.airport\_name as arrival\_airport, arr.airport\_city as arrival\_city, flight.seats\_left as seats, flight.airline\_name as airline, flight.price as price\
3. FROM (flight, airport as dept, airport as arr)\
4. WHERE flight.departure\_airport=dept.airport\_name and flight.arrival\_airport=arr.airport\_name AND flight.status = 'upcoming'"

If information provided:

1. **else**:
2. query = "SELECT flight.arrival\_time as arrival\_time, flight.status as status, flight.flight\_num as flight\_number, flight.departure\_time as departure\_time, dept.airport\_name as departure\_airport, dept.airport\_city as departure\_city, arr.airport\_name as arrival\_airport, arr.airport\_city as arrival\_city, flight.seats\_left as seats, flight.airline\_name as airline, flight.price as price\
3. FROM (flight, airport as dept, airport as arr) \
4. WHERE flight.departure\_airport=dept.airport\_name and flight.arrival\_airport=arr.airport\_name AND flight.status = 'upcoming' AND " + " AND ".join(control\_list);
5. cursor.execute(query)

* View Public Info (b):

Using a list to record all the information that the user has typed in the web page.

1. control\_list=[]
2. **if** flight\_num!='':
3. control\_list.append("flight\_num='%d'"%(**int**(flight\_num)))
4. **if** departure\_date != '':
5. control\_list.append("DATE(departure\_time) = '%s'"%(departure\_date))
6. **if** arrival\_date != '':
7. control\_list.append("DATE(arrival\_time) = '%s'"%(arrival\_date))
8. **if** len(control\_list) == 0:
9. query = "SELECT \* FROM flight"
10. **else**:
11. query = "SELECT \* FROM flight WHERE " + " AND ".join(control\_list)
12. cursor=conn.cursor()
13. cursor.execute(query)

Customer need to select a user type during registration. Then for different user types, different information are required.

* Registration (Customer):

Check whether the username has been registered:

1. cursor = conn.cursor()
2. query = 'SELECT \* FROM customer WHERE email = %s'
3. cursor.execute(query, (username))
4. data = cursor.fetchone()
5. **if**(data):
6. error = "This user already exists"
7. **return** render\_template('register.html', error = error)

Registration:

1. **else**:
2. ins = 'INSERT INTO customer VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)'
3. cursor.execute(ins,(username,name,password,building\_number,street,city,state,phone,passport\_number,passport\_expiration,passport\_country,date\_of\_birth))
4. conn.commit()
5. cursor.close()
6. **return** render\_template('index.html')

* Registration (Booking\_agent):

Generating unique booking\_agent\_id:

1. query\_1 = 'SELECT MAX(booking\_agent\_id) FROM booking\_agent'
2. cursor.execute(query\_1)
3. data\_1 = cursor.fetchone()
4. **if** data\_1==None:
5. max\_id=1
6. **else**:
7. max\_id = data\_1['MAX(booking\_agent\_id)']

Registration:

1. query = 'SELECT \* FROM booking\_agent WHERE email = %s'
2. cursor.execute(query, (username))
3. data = cursor.fetchone()
4. **if** (data):
5. error = "This agent already exists"
6. **return** render\_template('register.html', error = error)
7. **else**:
8. ins = 'INSERT INTO booking\_agent VALUES(%s,%s,%s)'
9. cursor.execute(ins,(username,password,max\_id+1))

* Registration (Airline\_staff):

Check whether the airline provided by the user is valid first and then register.

1. cursor = conn.cursor()
2. query = 'SELECT \* FROM airline\_staff WHERE username = %s'
3. query2 = 'SELECT \* FROM airline WHERE airline\_name = %s'
4. cursor.execute(query, (username))
5. data = cursor.fetchone()
6. cursor.execute(query2,(airline\_name))
7. data1 = cursor.fetchone()
8. **if** (data1 == None):
9. error = "This airline doesn't exist"
10. **return** render\_template('register.html', error = error)
11. **if** (data):
12. error = "This staff already exists"
13. **return** render\_template('register.html', error = error)
14. **else**:
15. ins = 'INSERT INTO airline\_staff VALUES(%s,%s,%s,%s,%s,%s)'
16. cursor.execute(ins,(username,password,first\_name,last\_name,date\_of\_birth,airline\_name))

* Login (Customer):

Check username and password:

1. cursor = conn.cursor()
2. query = 'SELECT \* FROM customer WHERE email = %s and password = %s'
3. cursor.execute(query, (username, password))
4. data = cursor.fetchone()

Create session and log in:

1. **if** (data):
2. session['username'] = username
3. session['type'] = 'customer'
4. **return** redirect(url\_for('hello'))
5. **else**:
6. error = "Invalid username or incorrect password"
7. **return** render\_template('login.html', error=error)

* Login (Booking\_agent):

Check username and password:

1. cursor = conn.cursor()
2. query = 'SELECT \* FROM booking\_agent WHERE email = %s and password = %s'
3. cursor.execute(query, (username, password))
4. data = cursor.fetchone()

Create session and log in:

1. **if** (data):
2. session['username'] = username
3. session['type'] = 'booking\_agent'
4. **return** redirect(url\_for('hello'))
5. **else**:
6. error = "Invalid username or incorrect password"
7. **return** render\_template('login.html', error=error)

* Login (Airline\_staff):

Check username and password:

1. cursor = conn.cursor()
2. query = 'SELECT \* FROM airline\_staff WHERE username = %s and password = %s'
3. cursor.execute(query, (username, password))
4. data = cursor.fetchone()

Create session and log in:

1. **if** (data):
2. session['username'] = username
3. session['type'] = 'airline\_staff'
4. **return** redirect(url\_for('hello'))
5. **else**:
6. error = "Invalid username or incorrect password"
7. **return** render\_template('login.html', error=error)

* Log out:

1. @app.route('/logout')
2. def logout():
3. session.pop('username')
4. session.pop('type')
5. **return** redirect('/')
6. Customer use case:

* View My Flight:

Use a control list to record all the information typed on the web page:

1. control\_list=[]
2. **if** dept\_airport!='':
3. control\_list.append("departure\_airport = '%s'"%(dept\_airport))
4. **if** arr\_airport!='':
5. control\_list.append("arrival\_airport = '%s'"%(arr\_airport))
6. **if** start\_date!='':
7. control\_list.append("DATE(departure\_time) >= '%s'"%(start\_date))
8. **if** end\_date!='':
9. control\_list.append("DATE(departure\_time) <= '%s'"%(end\_date))

Provide flight information based on information provided:

1. **if** len(control\_list) == 0:
2. query = "SELECT \* FROM flight NATURAL JOIN ticket NATURAL JOIN purchases \
3. WHERE status = 'upcoming' AND customer\_email = '%s'"%(username)
4. **else**:
5. query = "SELECT \* FROM flight NATURAL JOIN ticket NATURAL JOIN purchases \
6. WHERE customer\_email = '%s' AND "%(username) + " AND ".join(control\_list)

* Purchase ticket:

User will click on the flight that he/she wants to buy, given the information, server will check whether there are remaining seats for that flight. If there are, generate a new tickedt\_id and create a new ticket.

1. cursor=conn.cursor()
2. message = None
3. query\_0 = 'SELECT seats\_left FROM flight WHERE airline\_name = %s AND flight\_num = %s'
4. cursor.execute(query\_0,(airline\_name,flight\_num))
5. data\_0 = cursor.fetchone()
6. seats\_left = **int**(data\_0['seats\_left'])
7. **if** (seats\_left > 0):
8. query\_1 = 'SELECT MAX(ticket\_id) FROM ticket'
9. cursor.execute(query\_1)
10. data\_1 = cursor.fetchone()
11. max = data\_1['MAX(ticket\_id)']
12. **if** (max == None):
13. new\_id = 1;
14. **else**:
15. new\_id = **int**(max) + 1

Insert a new tuple in purchases and update how many seats left for that flight:

1. query\_3 = 'INSERT INTO purchases VALUES(%s,%s,NULL,CURDATE())'
2. cursor.execute(query\_3,(str(new\_id),username))
3. query\_4 = 'UPDATE flight SET seats\_left = seats\_left - 1 WHERE flight.airline\_name=%s AND flight\_num = %s'
4. cursor.execute(query\_4,(airline\_name,flight\_num))

* Track Customer Spending:

Get total spending of this customer:

1. cursor = conn.cursor()
2. query = 'SELECT sum(price) as tot\
3. FROM purchases NATURAL JOIN ticket NATURAL JOIN flight\
4. WHERE customer\_email = %s and purchase\_date>=%s\
5. and purchase\_date<DATE\_ADD(%s, INTERVAL 1 MONTH)'
6. cursor.execute(query, (username, start\_date, end\_date))
7. total\_spending = cursor.fetchone()
8. total\_spending = total\_spending['tot']
9. **if** total\_spending==None:
10. total\_spending=0
11. **else**:
12. total\_spending=**int**(total\_spending)

Base on the start\_date and end\_date that the user provided (or default value), the server will fetch spending each month and record the information in a list called data. Also the server will record the corresponding date in a list called label.

1. cur\_y=**int**(start\_date[:4])
2. cur\_m=**int**(start\_date[5:7])
3. end\_y=**int**(end\_date[:4])
4. end\_m=**int**(end\_date[5:7])
5. end\_m+=1;
6. **if** end\_m>12:
7. end\_m=1;
8. end\_y+=1;
9. data=[]
10. label=[]
11. **while** not (cur\_y==end\_y and cur\_m==end\_m):
12. start\_date = date(cur\_y, cur\_m, 1).isoformat()
13. end\_date = date(cur\_y, cur\_m, 1).isoformat()
14. print(start\_date, end\_date)
15. query = 'SELECT sum(price) as tot\
16. FROM purchases NATURAL JOIN ticket NATURAL JOIN flight\
17. WHERE customer\_email = %s and purchase\_date>=%s\
18. and purchase\_date<DATE\_ADD(%s, INTERVAL 1 MONTH)'
19. print(query%(username, start\_date, end\_date))
20. cursor.execute(query, (username, start\_date, end\_date))
21. cur\_spending = cursor.fetchone()
22. cur\_spending = cur\_spending['tot']
23. **if** cur\_spending==None:
24. cur\_spending=0;
25. label.append(start\_date[:7])
26. data.append(**int**(cur\_spending))
27. cur\_m+=1
28. **if** cur\_m>12:
29. cur\_m=1
30. cur\_y+=1

‘chart.html’ will receive two lists: data and label and generate a bar chart.

1. **return** render\_template('chart.html', total\_spending=total\_spending, data=data, label=label, act='Spending')
2. Booking\_agent Use Case:

* View my flight:

Use a control list to record all the information typed on the web page:

1. control\_list=[]
2. **if** dept\_airport!='':
3. control\_list.append("departure\_airport = '%s'"%(dept\_airport))
4. **if** arr\_airport!='':
5. control\_list.append("arrival\_airport = '%s'"%(arr\_airport))
6. **if** start\_date!='':
7. control\_list.append("DATE(purchase\_date)>'%s'"%(start\_date))
8. **if** end\_date!='':
9. control\_list.append("DATE(purchase\_date)<'%s'"%(end\_date))

Get the booking\_agent\_id of this booking\_agent. Here the booking\_agent\_id is unique since it is generated by the server directly. Provide flight information based on information provided:

1. cursor = conn.cursor()
2. query\_0 = 'SELECT booking\_agent\_id FROM booking\_agent WHERE email = %s'
3. cursor.execute(query\_0,(username))
4. data\_0 = cursor.fetchone()
5. booking\_agent\_id = data\_0['booking\_agent\_id']
7. **if** len(control\_list) == 0:
8. query\_1 = "SELECT \* FROM flight NATURAL JOIN ticket NATURAL JOIN purchases \
9. WHERE status = 'upcoming' AND booking\_agent\_id = '%s'"%(booking\_agent\_id)
10. **else**:
11. query\_1 ="SELECT \* FROM flight NATURAL JOIN ticket NATURAL JOIN purchases \
12. WHERE booking\_agent\_id = '%s' AND "%(booking\_agent\_id) + " AND ".join(control\_list)
13. cursor.execute(query\_1)

* Purchase Ticket:

Booking\_agent will click on the ticket that he/she wants to buy for the customer, session will record the flight information and direct the user to ‘purchase\_agent.ht

ml’ to provide customer information:

1. session['airline\_name'] = airline\_name
2. session['flight\_num'] = flight\_num
3. **return** render\_template('purchase\_agent.html')

Given the user information on ‘purchase\_agent.html’ that provided by booking\_ag

ent, the server will check whether that user exists:

1. query\_0 = 'SELECT email FROM customer WHERE email = %s'
2. cursor = conn.cursor()
3. cursor.execute(query\_0,(username\_c))
4. data\_0 = cursor.fetchone()
5. **if** data\_0 == None:
6. error = 'Please provide valid information'
7. **return** render\_template('purchase\_agent.html', error = error)

Check whether there are remaining seats for the given flight. If seats are running out, inform the user. Otherwise create a new ticke\_id and a new ticket for that flight.

1. query\_1 = 'SELECT seats\_left FROM flight WHERE airline\_name = %s AND flight\_num = %s'
2. cursor.execute(query\_1,(airline\_name\_1,flight\_num\_1))
3. data\_1 = cursor.fetchone()
4. seats\_left = **int**(data\_1['seats\_left'])
5. **if** (seats\_left < 1):
6. message = 'Purchase Failure. No more seats for this flight'
7. **return** render\_template('purchase.html',message = message)
8. query\_2 = 'SELECT MAX(ticket\_id) FROM ticket'
9. cursor.execute(query\_2)
10. data\_2 = cursor.fetchone()
11. max = data\_2['MAX(ticket\_id)']
12. **if** (max == None):
13. new\_id = 1
14. **else**:
15. new\_id = **int**(max) + 1
16. query\_3 = 'INSERT INTO ticket VALUES(%s,%s,%s)'
17. cursor.execute(query\_3,(str(new\_id),airline\_name\_1,flight\_num\_1))

Get the booking\_agent\_id from the database and then insert purchase information into purchases relation. Finally update seats\_left attribute of the flight that has just been bought.

1. query\_4 = 'SELECT booking\_agent\_id FROM booking\_agent WHERE email = %s'
2. cursor.execute(query\_4,(username\_b))
3. data\_4 = cursor.fetchone()
4. b\_id = str(data\_4['booking\_agent\_id'])
5. query\_5 = 'INSERT INTO purchases VALUES(%s,%s,%s,CURDATE())'
6. cursor.execute(query\_5,(str(new\_id),username\_c,b\_id))
7. query\_6 = 'UPDATE flight SET seats\_left = seats\_left - 1 WHERE airline\_name =%s AND flight\_num=%s'
8. cursor.execute(query\_6,(airline\_name\_1,flight\_num\_1))

* View my commission:

Get booking\_agent\_id of the booking\_agent currently logged in.

1. query\_0 = 'SELECT booking\_agent\_id FROM booking\_agent WHERE email = %s'
2. cursor.execute(query\_0,(username))
3. data\_0 = cursor.fetchone()
4. b\_id = data\_0['booking\_agent\_id']

If the booking\_agent doesn’t specify a range of date, the server will assume the range is last 30 days. The server will first fetech total commission during the last 30 days from the database and then fetech how many tickets sold during the last 30 days. After that average commission will be calculated based on how many tickets sold:

1. start\_date = (date.today()-timedelta(days=30)).isoformat()
2. end\_date = date.today().isoformat()
3. print(start\_date)
4. print(end\_date)
5. query = 'select sum(price)\*0.1 as tot\_commission\
6. from purchases natural join ticket natural join flight\
7. where booking\_agent\_id=%s and purchase\_date>=%s and purchase\_date<=%s';
8. cursor.execute(query, (b\_id, start\_date, end\_date))
9. data = cursor.fetchone()
10. commission = data['tot\_commission']
11. **if** commission==None:
12. commission=0;
13. query = 'select count(ticket\_id) as tot\_ticket\
14. from purchases natural join ticket natural join flight\
15. where booking\_agent\_id=%s and purchase\_date>=%s and purchase\_date<=%s';
16. cursor.execute(query, (b\_id, start\_date, end\_date))
17. data = cursor.fetchone()
18. ticket = data['tot\_ticket']
19. **if** ticket!=0:
20. commission = commission / ticket;
21. ans = ['Your average commission per ticket of past 30 days is %.2f'%(commission),
22. 'Your total ticket sold of past 30 days is %d'%(ticket)];
23. **return** render\_template('commission.html', message=ans)

If the user specifies the range of dates, the server will first fetch total commission during this time period and then fetch how many tickets sold during this time period. Finally, the server will calculate average commission based on how many tickets sold.

1. start\_date=str(request.form['start\_date'])
2. end\_date=str(request.form['end\_date'])
3. **if** start\_date>end\_date:
4. **return** render\_template('commission.html', message=['wrong start or end date']);
5. query = 'select sum(price)\*0.1 as tot\_commission\
6. from purchases natural join ticket natural join flight\
7. where booking\_agent\_id=%s and purchase\_date>=%s and purchase\_date<=%s';
8. cursor.execute(query, (b\_id, start\_date, end\_date))
9. data = cursor.fetchone()
10. commission = data['tot\_commission']
11. **if** commission==None:
12. commission=0;
13. query = 'select count(ticket\_id) as tot\_ticket\
14. from purchases natural join ticket natural join flight\
15. where booking\_agent\_id=%s and purchase\_date>=%s and purchase\_date<=%s';
16. cursor.execute(query, (b\_id, start\_date, end\_date))
17. data = cursor.fetchone()
18. ticket = data['tot\_ticket']
19. ans = ['Your total commission of this time period is %d'%(commission),
20. 'Your total ticket sold of this time period is %d'%(ticket)];
21. **return** render\_template('commission.html', message=ans)

* View Top Customers

Get booking\_agent\_id of the booking\_agent currently logged in:

1. query\_0 = 'SELECT booking\_agent\_id FROM booking\_agent WHERE email = %s'
2. cursor.execute(query\_0,(username))
3. data\_0 = cursor.fetchone()
4. b\_id = data\_0['booking\_agent\_id']

Generate current date and the date six months ago:

1. end\_date = date.today().isoformat()[:-3]+"-01"
2. cur\_y = date.today().year
3. cur\_m = date.today().month
4. cur\_m = cur\_m - 5
5. **if** (cur\_m < 1):
6. cur\_y = cur\_y - 1
7. cur\_m = cur\_m + 12
8. start\_date = date(cur\_y,cur\_m,1).isoformat()

The server will first fetch the top 5 customers based on how many tickets they have bought during the last 6 months. Data will be recorded in the list data1 and the corresponding month will be recorded in label1.

1. query\_1 = 'SELECT customer\_email,COUNT(DISTINCT ticket\_id) AS count FROM ticket NATURAL JOIN purchases \
2. WHERE booking\_agent\_id = %s AND purchase\_date >%s AND purchase\_date < DATE\_ADD(%s, INTERVAL 1 MONTH) \
3. GROUP BY customer\_email ORDER BY count DESC'
4. cursor.execute(query\_1,(str(b\_id),start\_date,end\_date))
5. data\_1 = cursor.fetchall()[:5]
6. data1=[]
7. label1=[]

The server will then fetch top 5 customers based on how much commission the booking\_agent earned from them during the last month. Data will be recorded in the list data2 and the corresponding month will be recorded in label2.

1. query\_2 = 'SELECT customer\_email, SUM(price)\*0.1 AS commission FROM purchases NATURAL JOIN ticket NATURAL JOIN flight \
2. WHERE booking\_agent\_id = %s AND purchase\_date > %s AND purchase\_date < DATE\_ADD(%s, INTERVAL 1 MONTH) \
3. GROUP BY customer\_email ORDER BY commission DESC'
4. cursor.execute(query\_2,(str(b\_id),start\_date,end\_date))
5. data\_2 = cursor.fetchall()[:5]
6. data2=[]
7. label2=[]
8. Airline Staff Use Cases
9. View my flights:
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get flight info:

SELECT \*

From flight

WHERE airline\_name = '%s' and status='upcoming' and DATE(departure\_time) > CURDATE() and DATE(departure\_time) < DATE\_ADD(CURDATE(), INTERVAL 1 MONTH) and departure\_airport = '%s' and arrival\_airport = '%s' and DATE(departure\_time)>'%s' and DATE(arrival\_time)<'%s'

* 1. To view customer on particular flight:

SELECT DISTINCT customer\_email

FROM ticket NATURAL JOIN purchases

WHERE airline\_name =%s AND flight\_num=%s

1. Create new flights:
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get seats of the airplane used:

SELECT seats

FROM airplane

WHERE airplane\_id = %s AND airline\_name = %s

* 1. To check if exists same flight:

SELECT \*

FROM flight

WHERE airline\_name = %s AND flight\_num = %s

* 1. To create flight:

‘INSERT INTO flight VALUES(%s,%s,%s,%s,%s,%s,%s,"upcoming",%s,%s)’% (airline\_name,flight\_num,dept\_airport,dept\_time,arr\_airport,arr\_time,price,airplane\_id,seats\_left)

* 1. To show upcoming flight for next 30 days:

SELECT \*

FROM flight

WHERE airline\_name=%s AND DATE(departure\_time)>=CURDATE() AND DATE(departure\_time)<=DATE\_ADD(CURDATE(), INTERVAL 1 MONTH) AND status='upcoming'

1. Change status of flights
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To check if exist this flight:

SELECT \*

FROM flight

WHERE airline\_name='%s' and flight\_num=%s

* 1. To update status:

UPDATE flight

SET status=%s

WHERE airline\_name=%s and flight\_num=%s

1. Add airplane
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To check if exists airplane:

SELECT \*

FROM airplane

WHERE airline\_name='%s' and airplane\_id=%s

* 1. To insert airplane:

"INSERT INTO airplane

VALUES ('%s', %s, %s)"%(airline\_name,airplane\_id,seats)

* 1. To show airplanes owned by this airline:

SELECT \*

FROM airplane

WHERE airline\_name='%s'

1. Add airport
   1. To check if exists this airport:

SELECT \*

FROM airport

WHERE airport\_name='%s'

* 1. To insert airport:

"INSERT INTO airport

VALUES ('%s', '%s')"%(airport\_name,airport\_city)

1. View all booking agents
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get top 5 agent last month by ticket number:

SELECT booking\_agent\_id, COUNT(DISTINCT ticket\_id) AS count

FROM ticket NATURAL JOIN purchases

WHERE purchase\_date >=DATE\_ADD(CURDATE(), INTERVAL -1 MONTH) AND purchase\_date <=CURDATE() AND booking\_agent\_id is not NULL and airline\_name=%s

GROUP BY booking\_agent\_id ORDER BY count DESC

* 1. To get top 5 agent last year by ticket number:

SELECT booking\_agent\_id, COUNT(DISTINCT ticket\_id) AS count

FROM ticket NATURAL JOIN purchases

WHERE purchase\_date >=DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) AND purchase\_date <=CURDATE() AND booking\_agent\_id is not NULL and airline\_name=%s

GROUP BY booking\_agent\_id ORDER BY count DESC

* 1. To get top 5 agent last year by commission:

SELECT booking\_agent\_id, sum(price)\*0.1 AS count

FROM ticket NATURAL JOIN purchases NATURAL JOIN flight

WHERE purchase\_date >=DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) AND purchase\_date<=CURDATE() AND booking\_agent\_id is not NULL and airline\_name=%s

GROUP BY booking\_agent\_id ORDER BY count DESC

1. View frequent customers
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get most frequent customer:

SELECT customer\_email, COUNT(ticket\_id) AS count

FROM purchases NATURAL JOIN ticket

WHERE airline\_name = %s

GROUP BY customer\_email ORDER BY count DESC

* 1. To get all customer emails:

SELECT DISTINCT customer\_email

FROM purchases NATURAL JOIN ticket

WHERE airline\_name = %s

* 1. To check a particular customer’s flights

SELECT \*

FROM ticket NATURAL JOIN flight NATURAL JOIN purchases

WHERE airline\_name = %s AND customer\_email = %s

1. View reports
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get total ticket sold in the a time period:

SELECT count(DISTINCT ticket\_id) as tot\

FROM purchases NATURAL JOIN ticket\

WHERE purchase\_date>=%s and purchase\_date<DATE\_ADD(%s, INTERVAL 1 MONTH) and airline\_name=%s

1. Comparison of revenue
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get direct sale last month:

SELECT count(DISTINCT ticket\_id) as tot \

FROM purchases NATURAL JOIN ticket\

WHERE booking\_agent\_id is NULL AND purchase\_date>=DATE\_ADD(CURDATE(), INTERVAL -1 MONTH) AND purchase\_date<=CURDATE() and airline\_name=%s

* 1. To get in direct sale last month:

SELECT count(DISTINCT ticket\_id) as tot \

FROM purchases NATURAL JOIN ticket \

WHERE booking\_agent\_id is not NULL AND purchase\_date>=DATE\_ADD(CURDATE(), INTERVAL -1 MONTH) AND purchase\_date<=CURDATE() and airline\_name=%s

* 1. To get direct sale last year:

SELECT count(DISTINCT ticket\_id) as tot \

FROM purchases NATURAL JOIN ticket\

WHERE booking\_agent\_id is NULL AND purchase\_date>=DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) AND purchase\_date<=CURDATE() and airline\_name=%s

* 1. To get in direct sale last year:

SELECT count(DISTINCT ticket\_id) as tot \

FROM purchases NATURAL JOIN ticket \

WHERE booking\_agent\_id is not NULL AND purchase\_date>=DATE\_ADD(CURDATE(), INTERVAL -1 YEAR) AND purchase\_date<=CURDATE() and airline\_name=%s

1. View top destinations
   1. To get airline name:

SELECT airline\_name

FROM airline\_staff

WHERE username = %s

* 1. To get top 3 popular destination in the last 3 months:

SELECT arrival\_airport,COUNT(arrival\_airport) AS count

FROM flight

WHERE airline\_name = %s AND DATE(departure\_time) < CURDATE() AND DATE(departure\_time) > DATE\_ADD(CURDATE(),INTERVAL -3 MONTH)

GROUP BY arrival\_airport ORDER BY count DESC

* 1. To get top 3 popular destination in the last year:

SELECT arrival\_airport,COUNT(arrival\_airport) AS count

FROM flight

WHERE airline\_name = %s AND DATE(departure\_time) < CURDATE() AND DATE(departure\_time) > DATE\_ADD(CURDATE(),INTERVAL -12 MONTH) \

GROUP BY arrival\_airport ORDER BY count DESC