

Name : Kirti Gupta
Employment id :TAS063

SQL assignment:

(f and t mean false and true, respectively)

Please share query and output for each:

Questions:

What is the time period used?

Earliest date of this dataset is 2016-09-06 and the latest date of this dataset is 2016-09-06, So the time period range is 2016-09-06 to 2016-09-06

```
select min(date) as Start_Date ,max(date) as Last_Date from airbnb__calender;
```

The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
1 select * from airbnb__calender order by date;
2
3 select * from airbnb__calender order by date desc;
4 select min(date) from airbnb__calender;
5
6
7
8 select min(date) as Start_Date ,max(date) as Last_Date from airbnb__calender;
9
```

The results pane displays the output of the query. The first two queries return all rows from the `airbnb__calender` table, ordered by date and descending by date, respectively. The third query returns the minimum date from the `airbnb__calender` table. The fourth query returns the minimum date as `Start_Date` and the maximum date as `Last_Date` from the `airbnb__calender` table.

Start_Date	Last_Date
2016-09-06	2017-09-05

The results pane also shows the duration and fetch time for each query. The first query took 0.378 sec / 0.000020... to execute. The second query took 0.282 sec / 0.000008... to execute. The third query took 0.394 sec / 0.00002... to execute. The fourth query took 0.408 sec / 0.000018... to execute.

2. How many properties have duplicate entries? Remove duplicate rows (say a row appears 3 times, remove 2 and keep 1)

We use count(*) to return the number of rows in a specified table In this database we have two attribute which together have distinct value in each row, so by using count(*) we can know how many rows appear duplicate.

```
select listing_id,count(*) from airbnb__calender
group by listing_id,date having count(*)>1;
```

The screenshot shows a SQL IDE interface with a query editor, a schema browser, and a result grid. The query editor contains the following SQL code:

```
6 • select count(distinct(listing_id)) from airbnb__calender; -- 3585 return rows
7
8 • select listing_id, count(listing_id) from airbnb__calender-- each id contain 365 enteries
9   group by listing_id;
10
11 • select listing_id, count(listing_id) as number_of_entries from airbnb__calender
12   group by listing_id having count(listing_id) <= 365;
13
14 • select listing_id, count(*) from airbnb__calender
15   group by listing_id, date having count(*) > 1;
16
17
18
19 • select listing_id, count(listing_id) from airbnb__calender
20   group by listing_id
```

The result grid shows the following data:

listing_id	count(*)
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2
12898806	2

The action output shows the following results:

Time	Action	Response	Duration / Fetch Time
23:25:46	select listing_id, count(*) from airbnb__calender where count(*) > 1	Error Code: 1111. Invalid use of group function	0.00036 sec
23:27:13	select listing_id, count(*) from airbnb__calender group by listing_id, date having count(*) > 1	365 row(s) returned	3.341 sec / 0.000059...
23:28:39	select sum(count(*)) from airbnb__calender group by listing_id, date having count(*) > 1	Error Code: 1111. Invalid use of group function	0.00040 sec
23:39:03	select listing_id, count(*) from airbnb__calender group by listing_id, date having count(*) > 1	365 row(s) returned	3.318 sec / 0.00013 s...

Create new table having no duplicate values:

```
create table new_airbnb__calender as
select distinct * from airbnb__calender;
```

The screenshot shows a SQL IDE interface with a query editor, a schema browser, a result grid, and an action output pane.

Query Editor:

```

15 group by listing_id,date having count(*)>1;
16
17 -- create a table having no duplicate values
18
19 • create table new_airbnb__calender as
20   select distinct * from airbnb__calender;
21
22 • select * from new_airbnb__calender;
23
24
25
26
27
28
29

```

Schema Browser: The left pane shows a database schema with tables, views, stored procedures, and functions. The table `new_airbnb__calender` is selected.

Result Grid: The bottom pane shows the result of the query. It displays a table with columns `listing_id`, `date`, `available`, and `price`. The data is filtered to show only rows where `available` is true.

Action Output: The bottom pane shows the execution details of the query. It includes the time taken for each step, the action performed, the response, and the duration/fetch time.

Time	Action	Response	Duration / Fetch Time
42 23:39:03	select listing_id,count(*) from airbnb__calender group by listing_id,date having count(*)>1	365 row(s) returned	3.318 sec / 0.00013 s...
43 23:44:01	create table new_airbnb__calender as select distinct * from airbnb__calender	1308525 row(s) affected Records: 1308525 Duplicat...	5.440 sec
44 23:47:22	select * from new_airbnb__calender	Error Code: 1146. Table 'sql_assignment.new_airbnb_c...	0.00060 sec
45 23:49:33	select * from new_airbnb__calender	1308525 row(s) returned	0.00086 sec / 0.379...

1. For each property, find out the number of days the property was available and not available (create a table with listing_id, available days, unavailable days and available days as a fraction of total days)

Create a table with store availability or not availability of the property

```

create table available_table as
select listing_id,
case available
when 't' then 1
else 0
end is_available,
case available
when 'f' then 1
else 0
end is_not_available
from new_airbnb__calender;

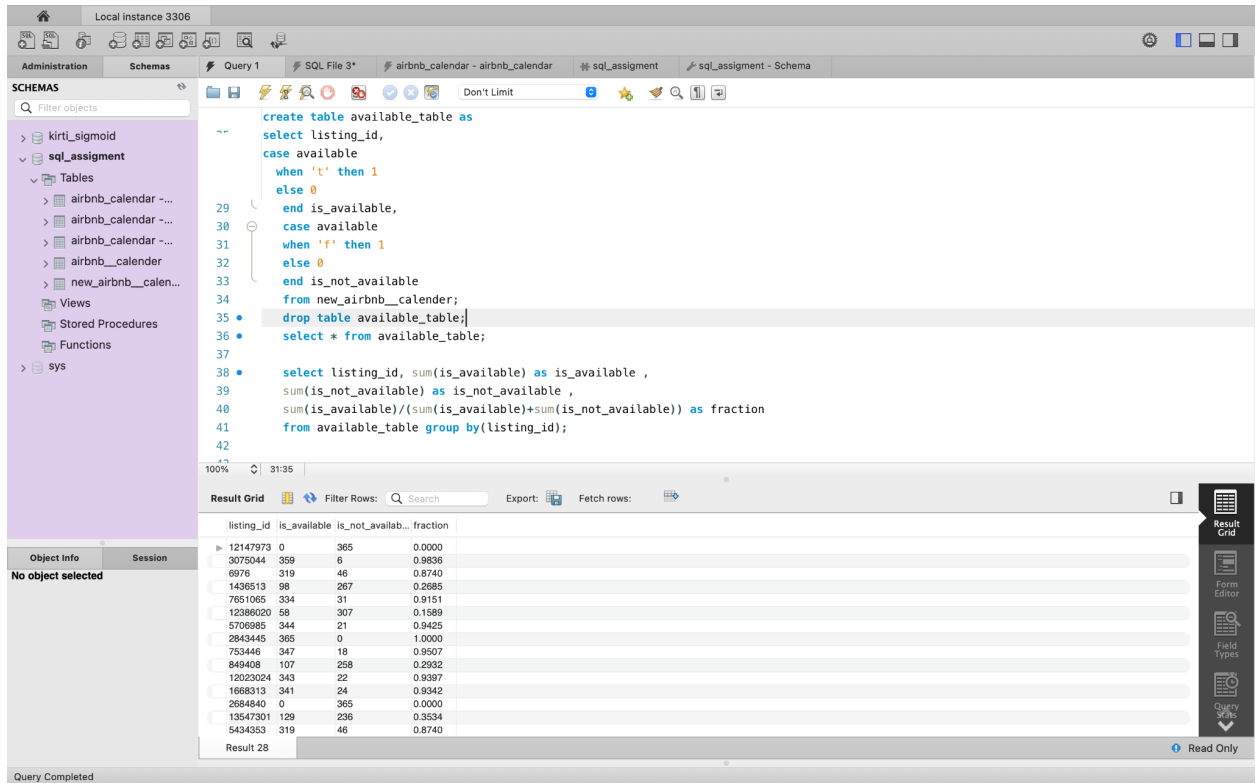
```

By querying we can get the number of days property was available or not

```

select listing_id, sum(is_available) as is_available ,
       sum(is_not_available) as is_not_available ,
       sum(is_available)/(sum(is_available)+sum(is_not_available)) as fraction
from available_table group by(listing_id);

```



The screenshot shows a SQL IDE interface with a query editor and a result grid. The query editor contains the following SQL code:

```

create table available_table as
select listing_id,
       case available
       when 't' then 1
       else 0
       end is_available,
       case available
       when 'f' then 1
       else 0
       end is_not_available
from new_airbnb_calendar;
drop table available_table;
select * from available_table;
select listing_id, sum(is_available) as is_available ,
       sum(is_not_available) as is_not_available ,
       sum(is_available)/(sum(is_available)+sum(is_not_available)) as fraction
from available_table group by(listing_id);

```

The result grid shows the output of the query, which is a table with 4 columns: listing_id, is_available, is_not_available, and fraction. The table contains 28 rows of data.

listing_id	is_available	is_not_available	fraction
12147973	0	365	0.0000
3075044	359	6	0.9836
6978	319	46	0.8740
1436513	98	267	0.2685
7651065	334	31	0.9151
12386020	58	307	0.1589
5706965	344	21	0.9425
2843445	365	0	1.0000
753446	347	18	0.9507
849408	107	258	0.2932
12023024	343	22	0.9397
1668313	341	24	0.9342
2664840	0	365	0.0000
13547301	129	236	0.3534
5434353	319	46	0.8740

4. How many properties were available on more than 50% of the days? How many properties were available on more than 75% of the days?

Create a table from the previous table in which property and their availability attributes are present, In new table, where we store property and their fraction of available days to total days.

```

create table available_fraction as
select listing_id,
       sum(is_available)/(sum(is_available)+sum(is_not_available)) as fraction
from available_table group by(listing_id);

```

Query 1 : properties are available more then 50% of the day

```
select count(listing_id) from available_fraction
where fraction > 0.5;
```

The screenshot displays the DBeaver SQL client interface for 'Local Instance 3306'. The left sidebar contains navigation panels for 'MANAGEMENT', 'INSTANCE', and 'PERFORMANCE'. The main editor shows a SQL script for 'Query 1' with the following content:

```
-- query 4
43 • create table available_fraction as
44 • select listing_id,
45 • sum(is_available)/(sum(is_available)+sum(is_not_available)) as fraction
46 • from available_table group by(listing_id);
47 -- properties are available more then 50% of the day
48 • select count(listing_id) from available_fraction
49 • where fraction > 0.5;
50
51
52
53
54
55
```

Below the editor, the 'Result Grid' shows the results of the query. The first column is 'count(listing_id)' and the second column is '1732'. The 'Action Output' panel at the bottom shows the execution log:

	Time	Action	Response	Duration / Fetch Time
1	12:27:36	select * from available_table	1308525 row(s) returned	0.0039 sec / 0.322 sec
2	12:30:17	create table available_fraction as select listing_id, sum(is_available) as is_available, sum(is_not_available) as is_no...	3585 row(s) affected, 1024 warning(s): 1265 Data tr	
3	12:31:02	drop table available_fraction	0 row(s) affected	
4	12:31:31	create table available_fraction as select listing_id, sum(is_available)/(sum(is_available)+sum(is_not_available)) as f...	3585 row(s) affected, 1024 warning(s): 1265 Data tr	
5	12:32:24	select * from available_fraction	3585 row(s) returned	
6	12:34:04	select count(listing_id) from available_fraction where fraction > 0.5	1 row(s) returned	

The status bar at the bottom indicates 'Query Completed'.

Query 2: properties are available more then 75% of the day

```
select count(listing_id) from available_fraction
where fraction> 0.75;
```

The screenshot shows a database management tool interface with a sidebar on the left containing sections for MANAGEMENT, INSTANCE, and PERFORMANCE. The main area displays a SQL query in a text editor. Below the editor, a 'Result Grid' shows the results of the query. At the bottom, an 'Action Output' table provides a detailed log of the database actions performed.

SQL Query:

```
-- query 4
44 • create table available_fraction as
45   select listing_id,
46     sum(is_available)/(sum(is_available)+sum(is_not_available)) as fraction
47   from available_table group by(listing_id);
48   -- properties are available more then 50% of the day
49 • select count(listing_id) from available_fraction
50   where fraction> 0.5;
51   -- properties are available more then 75% of the day
52 • select count(listing_id) from available_fraction
53   where fraction> 0.75;
```

Result Grid:

count(listing_...
1429

Action Output:

	Time	Action	Response	Duration / Fetch Time
1	12:27:36	select * from available_table	1308525 row(s) returned	0.0039 sec / 0.322 sec
2	12:30:17	create table available_fraction as select listing_id, sum(is_available) as is_available , sum(is_not_available) as is_no...	3585 row(s) affected, 1024 warning(s): 1265 Data tru...	0.737 sec
3	12:31:02	drop table available_fraction	0 row(s) affected	0.0044 sec
4	12:31:31	create table available_fraction as select listing_id, sum(is_available)/(sum(is_available)+sum(is_not_available)) as f...	3585 row(s) affected, 1024 warning(s): 1265 Data tru...	0.613 sec
5	12:32:24	select * from available_fraction	3585 row(s) returned	0.00083 sec / 0.0017...
6	12:34:04	select count(listing_id) from available_fraction where fraction> 0.5	1 row(s) returned	0.0051 sec / 0.00001...
7	12:36:20	select count(listing_id) from available_fraction where fraction> 0.75	1 row(s) returned	0.0043 sec / 0.00000...

Query Completed

5. Create a table with max, min and average price of each property

```
create table table_min_max_average as
select listing_id, min(price) as min,
max(price) as max,
avg(price) as average from new_airbnb__calender
where available='t'
group by (listing_id) ;
```

```
select * from table_min_max_average order by listing_id;
```

Local instance 3306

AdministrationSchemasQuery 1SQL File 3*airbnb_calendar ->airbnb_calendar

SCHEMAS

Filter objects

▼

kirti_sigmod

▼

Tables

Views

Stored Procedures

Functions

▼

sql_assignment

▼

Tables

airbnb_calendar ->

airbnb_calendar ->

airbnb_calendar ->

airbnb_calendar

available_fraction

available_table

new_airbnb_calen...

Views

Stored Procedures

Functions

▼

sys

Object Info

Session

Table: new_airbnb_calendar

Columns:

listing_id

int

date

text

available

text

price

text

52

•

select count(listing_id) from available_fraction

53

where fraction> 0.75;

54

55

-- find the min ,max and average of each price

56

•

create table table_min_max_average as

57

select listing_id, min(price) as min,

58

max(price) as max,

59

avg(price) as average from new_airbnb_calendar

60

where available='t'

61

group by (listing_id) ;

62

63

•

select * from table_min_max_average order by listing_id;

64

65

66

67

68

100%

48:55

Result Grid

Filter Rows:

Search

Export:

Fetch rows:

	listing_id	min	max	average
▶	3353	32	36	35.204819277108435
▶	5506	145	275	147.2674418604651
▶	6695	195	325	197.40740740740742
▶	6976	65	65	65
▶	8792	154	154	154
▶	9273	225	225	225
▶	9765	192	490	236.85635359116023
▶	9824	209	490	222.32198142414862
▶	9855	259	309	266.55494505494505
▶	9857	301	702	363.39481268011525
▶	9858	449	699	486.77472527472525
▶	9860	240	533	269.75920679886684

table_min_max_average 8

Read Only

Action Output

Time

Action

Response

Duration / Fetch Time

9

12:45:46

select * from table_min_max_average

3585 row(s) returned

0.00079 sec / 0.0025...

10

12:48:00

select * from table_min_max_average order by listing_id

3585 row(s) returned

0.0073 sec / 0.0024...

Query Completed

6. Extract properties with an average price of more than \$500

select listing_id, average from table_min_max_average
where average > 500;

The screenshot shows a database management interface with a SQL editor and a results grid. The SQL editor contains the following queries:

```
54  
55 -- find the min ,max and average of each price  
56 • create table table_min_max_average as  
57 select listing_id, min(price) as min,  
58 max(price) as max,  
59 avg(price) as average from new_airbnb_calender  
60 where available='t'  
61 group by (listing_id) ;  
62  
63 • select * from table_min_max_average order by listing_id;  
64 -- extract the listing id having average > 500  
65 • select listing_id, average from table_min_max_average  
66 where average > 500;  
67  
68  
69  
70
```

The results grid displays the following data:

listing_id	average
50032	725
115836	525
475259	506.8965517241379
743211	569.9850299401197
1214214	521.4285714285714
1810397	800
1966195	519
2214923	514.5813953488372
2277821	608.6802325581396
2649521	600
2881388	671.1987767584097
3351728	819.1111111111111

The interface also shows a sidebar with a tree view of database objects, including 'kirti_sigmoid', 'Tables', 'Views', 'Stored Procedures', 'Functions', and 'sql_assignment'. The 'table_min_max_av...' table is selected under 'sql_assignment'.

The Action Output section shows the following results:

Time	Action	Response	Duration / Fetch Time
12:58:39	select listing_id, average from table_min_max_average where average > 500	68 row(s) returned	0.0037 sec / 0.00001...