Experiment No.07

PART A

(PART A: TO BE REFERRED BY STUDENTS)

A.1 Aim: To implement group by, having clause, aggregate functions for solving queries.

A.2 Prerequisite:

DML commands of SQL

A.3 Outcome:

After successful completion of this experiment students will be able to

1. Apply knowledge of relational algebra and structured query language to retrieve and manage data in relational databases.

A.4 Theory:

Aggregate Functions:

Aggregate functions are statistical functions such as count, min, max etc. They are used to compute a single value from a set of attribute values of a column:

count Counting Rows

Example: How many tuples are stored in the relation EMP?

select count(*) from EMP;

Example: How many different job titles are stored in the relation EMP?

select count(distinct JOB) from EMP;

max Maximum value for a column

min Minimum value for a column

Example: List the minimum and maximum salary.

select min(SAL), max(SAL) from EMP;

Example: Compute the difference between the minimum and maximum salary.

select max(SAL) - min(SAL) from EMP;

sum Computes the sum of values (only applicable to the data type number)

Example: Sum of all salaries of employees working in the department 30.

select sum(SAL) from EMP where DEPTNO = 30;

avg Computes average value for a column (only applicable to the data type number)

Note: avg, min and max ignore tuples that have a null value for the specified attribute, but count considers null values.

Grouping:

We have seen how aggregate functions can be used to compute a single value for a column. Often applications require grouping rows that have certain properties and then applying an aggregate function on one column for each group separately. For this, SQL provides the clause group by <group column(s)>. This clause appears after the where clause and must refer to columns of tables listed in the from clause.

Syntax:

select <column(s)> from <table(s)> where <condition> group by <group column(s)> [having <group condition(s)>];

Those rows retrieved by the selected clause that have the same value(s) for <group column(s)> are grouped. Aggregations specified in the select clause are then applied to each group separately. It is important that only those columns that appear in the <group column(s)> clause can be listed without an aggregate function in the select clause!

Example: For each department, we want to retrieve the minimum and maximum salary.

select DEPTNO, min(SAL), max(SAL) from EMP group by DEPTNO;

Rows from the table EMP are grouped such that all rows in a group have the same department number. The aggregate functions are then applied to each such group.

We thus get the following query result:

```
DEPTNO   MIN(SAL)  MAX(SAL)
10       1300      5000
20       800       3000
30       950       2850
```

Rows to form a group can be restricted in the where clause. For example, if we add the condition where JOB = 'CLERK', only respective rows build a group. The query then would retrieve the minimum and maximum salary of all clerks for each department. Note that is not allowed to specify any other column than DEPTNO without an aggregate function in the select clause since this is the only column listed in the group by clause (is it also easy to see that other columns would not make any sense).

Once groups have been formed, certain groups can be eliminated based on their properties, e.g., if a group contains less than three rows. This type of condition is specified using the having clause. As for the select clause also in a having clause only <group column(s)> and aggregations can be used.

Example: Retrieve the minimum and maximum salary of clerks for each department having more than three clerks.

select DEPTNO, min(SAL), max(SAL) from EMP where JOB = 'CLERK' group by DEPTNO having count(job) > 3;

A query containing a group by clause is processed in the following way:

1. Select all rows that satisfy the condition specified in the where clause.
2. From these rows form groups according to the group by clause.
3. Discard all groups that do not satisfy the condition in the having clause.
4. Apply aggregate functions to each group.
5. Retrieve values for the columns and aggregations listed in the select clause.

A.5 Task: For given tables solve below queries:

Queries:

1. **Give the average of the total fare.**
2. **Give the total collection of fare.**
3. **Which bus runs for the minimum distance?**
4. **Give the total number of people who have traveled so far group by ticket number.**
5. **Find out total fare for routes with the same origin.**
6. **Find out total fare for routes with origin as madras.**
7. **Calculate average fare for each ticket.**
8. **Count how many male or female passengers have traveled till now.**
9. **Count total number of routes for each category except category 02.**
10. **Find out details of tickets from the ticket header having more than one passenger.**
11. **Count the total number of non-stop routes for each place id.**
12. **Display those ticket_no for which the total number of males traveling on the ticket are more than two.**

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Portal access available)

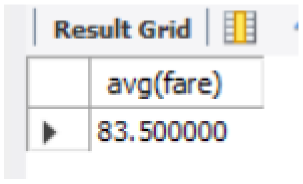
Roll No. I066	Name: Srihari Thyagarajan
Program: B Tech Artificial Intelligence	Division: I
Batch: B3	Date of Experiment: 16/09/2022
Date of Submission: 16/09/2022	Grade:

B.1 Commands and Output:

1)

# Query 1

SELECT avg(fare) FROM route_header;

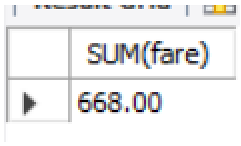


```
avg(fare)
83.500000
```

2)

# Query 2

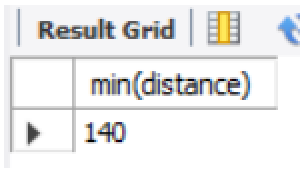
SELECT SUM(fare) FROM route_header;



```
SUM(fare)
668.00
```

3) # Query 3

SELECT min(distance) FROM route_header;


Result Grid

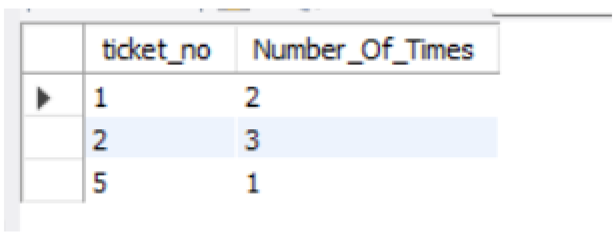
	min(distance)
▶	140

4)

# Query 4

SELECT ticket_no, COUNT(ticket_no) as Number_Of_Times FROM ticket_detail

GROUP BY ticket_no;



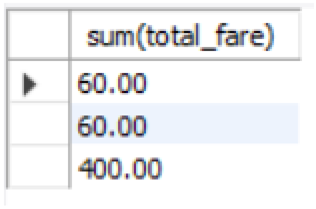
	ticket_no	Number_Of_Times
▶	1	2
	2	3
	5	1

5)

# Query 5

SELECT sum(total_fare) from ticket_header

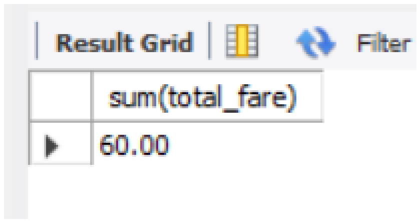
GROUP BY origin;



	sum(total_fare)
▶	60.00
	60.00
	400.00

6)

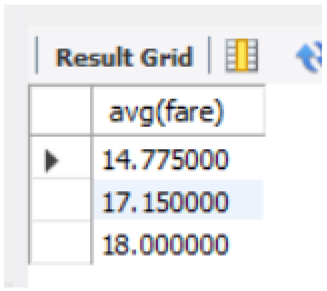
```
# Query 6
SELECT sum(total_fare) from ticket_header
WHERE origin = 'Madrasa';
```



```
sum(total_fare)
60.00
```

7)

```
# Query 7
SELECT avg(fare) from ticket_detail
GROUP by ticket_no;
```



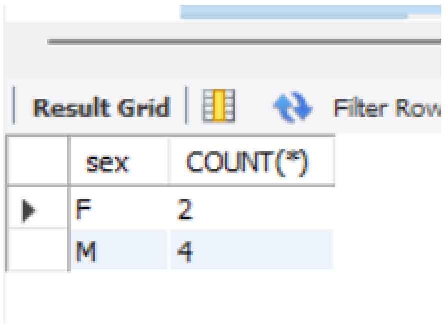
```
avg(fare)
14.775000
17.150000
18.000000
```

8)

# Query 8

SELECT sex, COUNT(*) from ticket_detail

GROUP by sex;



Result Grid | Filter Row

	sex	COUNT(*)
▶	F	2
	M	4

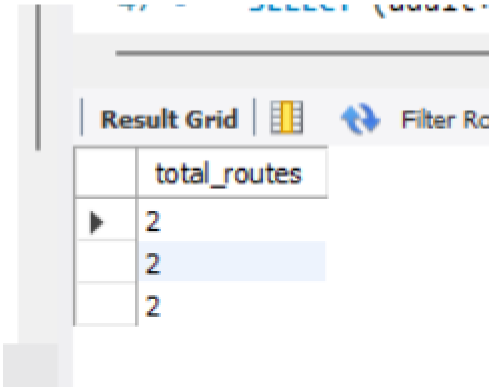
9)

# Query 9

SELECT COUNT(route_no) as total_routes from route_header

GROUP BY cat_code

HAVING not(cat_code = 2);



Result Grid | Filter Rc

	total_routes
▶	2
	2
	2

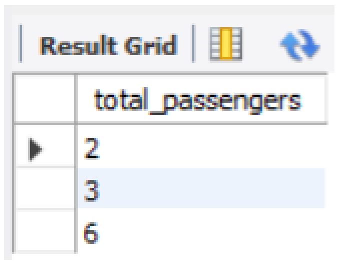
10)

# Query 10

SELECT (adult+children) as total_passengers FROM ticket_header

GROUP BY ticket_no

having total_passengers > 1;



total_passengers
2
3
6

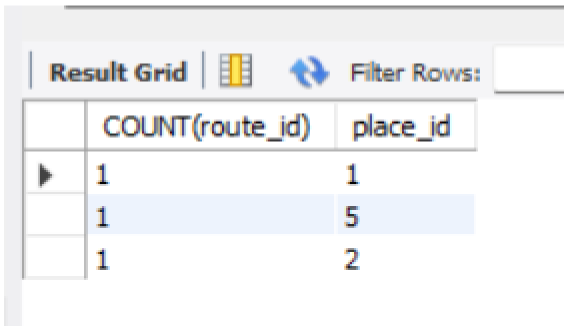
11)

# Query 11

SELECT COUNT(route_id), place_id FROM route_detail

Where nonstop = 'N'

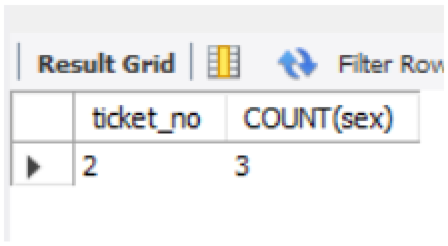
GROUP BY place_id;



COUNT(route_id)	place_id
1	1
1	5
1	2

12)

```
# Query 12
SELECT ticket_no, COUNT(sex) FROM ticket_detail
Where sex = 'M'
GROUP BY ticket_no
HAVING COUNT(SEX) > 2;
```



ticket_no	COUNT(sex)
2	3

B.2 Curiosity Questions:

1. Solve below queries for given tables:

Distributor (Dno, Dname, Daddress, Dphone)
Item (Itemno, Itemname, Colour, Weight)
Dist_Item(Dno, Itemno, Qty)

- **a. Find a distributor who has never supplied any item (using sub query).**
- **b. Count the total number of items of each color.**
- **c. Count the number of items supplied by each distributor.**

Solutions -

b.
```
SELECT Colour, COUNT(*) FROM Item
GROUP BY Colour;
```

c.
```
SELECT Dno, Itemno, COUNT(Itemno)

FROM Dist_Item

GROUP BY Dno;
```

B.3 Conclusion:

From the above experiment, I learnt the following:

- Implementation of knowledge of relational algebra and structured query language to retrieve and manage data in relational databases.
- Using GROUP and HAVING BY Clauses.