Experiment No.10

PART A

(PART A: TO BE REFFERED BY STUDENTS)

A.1 Aim: To implement group by, having clause, aggregate functions, sub queries, set operation and views for solving queries.

A.2 Prerequisite:

DML commands of SQL

A.3 Outcome:

After successful completion of this experiment students will be able to

8. Apply knowledge of relational algebra and structural 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.

Sub Queries:

Up to now we have only concentrated on simple comparison conditions in a where clause, i.e., we have compared a column with a constant or we have compared two columns. As we have already seen for the insert statement, queries can be used for assignments to columns. A query result can also be used in a condition of a where clause. In such a case the query is called a subquery and the complete select statement is called a nested query.

In a where clause conditions using subqueries can be combined arbitrarily by using the logical connectives and and or.

Example: List the name and salary of employees of the department 20 who are leading a project that started before December 31, 1990:

select ENAME, SAL from EMP where EMPNO in (select PMGR from PROJECT where PSTART < '31-DEC-90') and DEPTNO =20;

Explanation: The subquery retrieves the set of those employees who manage a project that started before December 31, 1990. If the employee working in department 20 is contained in this set (in operator), this tuple belongs to the query result set.

Example: List all employees who are working in a department located in BOSTON:

select * from EMP where DEPTNO in (select DEPTNO from DEPT where LOC = 'BOSTON');

The subquery retrieves only one value (the number of the department located in Boston). Thus it is possible to use "=" instead of in. As long as the result of a subquery is not known in advance, i.e., whether it is a single value or a set, it is advisable to use the in operator.

Conditions of the form <expression> <comparison operator> [any|all] <subquery> are used to compare a given <expression> with each value selected by <subquery>.

- For the clause any, the condition evaluates to true if there exists at least on row selected by the subquery for which the comparison holds. If the subquery yields an empty result set, the condition is not satisfied.
- For the clause all, in contrast, the condition evaluates to true if for all rows selected by the subquery the comparison holds. In this case the condition evaluates to true if the subquery does not yield any row or value.

Example: Retrieve all employees who are working in department 10 and who earn at least as much as any (i.e., at least one) employee working in department 30:

```
select * from EMP where SAL >= any (select SAL from EMP where DEPTNO = 30)
and DEPTNO = 10;
```

Note: Also in this subquery no aliases are necessary since the columns refer to the innermost from clause.

Example: List all employees who are not working in department 30 and who earn more than all employees working in department 30:

```
select * from EMP where SAL > all (select SAL from EMP where DEPTNO = 30) and DEPTNO
<> 30;
```

SET OPERATION

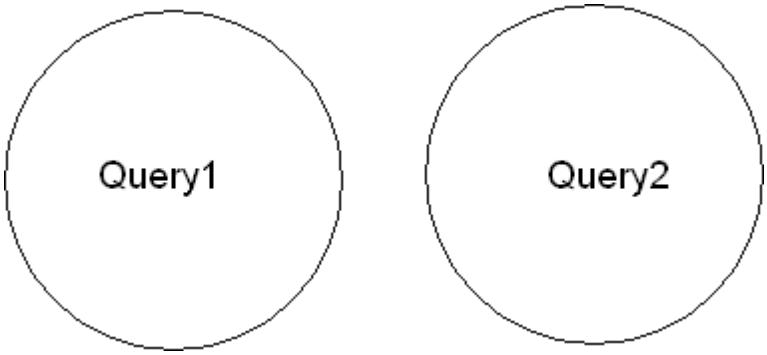
According to SQL Standard there are following Set operator types:

- Union [DISTINCT];
- Union All;
- Intersect [DISTINCT];
- Intersect All
- Minus

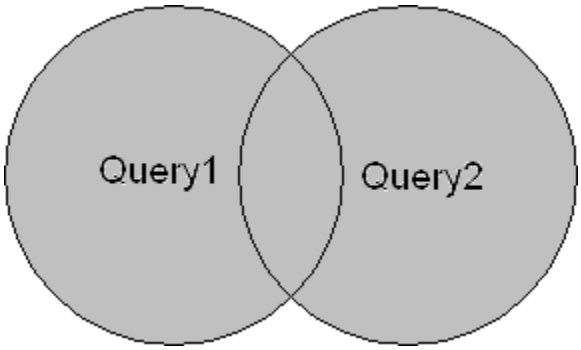
Some facts:

- UNION and INTERSECT operators are commutative, i.e. the order of queries is not important; it doesn't change the final result.
- Minus operator is NOT commutative, it IS important which query is first, which second using Minus operator.
- UNION, Minus and INTERSECT used without anything or with DISTINCT returns only unique values. This is especially interesting when one query returning many nonunique rows is UNIONED to another query returning zero rows. The final result contains fewer rows than first query.

These usually are most widely used set operators. Quite many times one cannot get all the result from one Select statement. Then one of the UNIONS can help. Graphically UNION can be visualised using Venn diagrams. Assume we have two row sets.



Then Query1 UNION Query2 would be as follows. Grey area shows resultant set.



Example of Union:

The **First** table,

ID	Name
1	John
2	adam

The **Second** table,

ID	Name
2	Adam
3	Chirag

Union SQL query will be:

ID	Name
1	John
2	Adam
3	Chirag

```
select * from First UNION select * from second
```

The result table will be:

Union All:

This operation is similar to Union. But it also shows the duplicate rows.

For above tables, Union all will be:

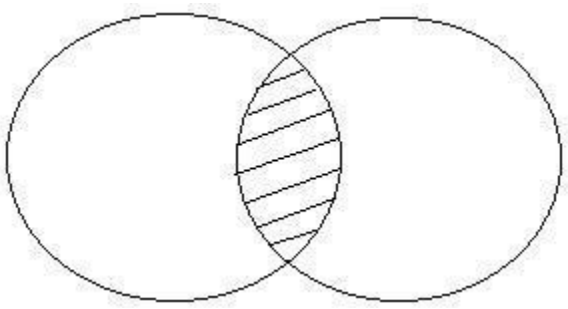
select * from First **UNION ALL** select * from second

The result table will be:

ID	Name
1	John
2	Adam
2	Adam
3	Chirag

Intersect:

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and data type must be same.



For the given above tables (In example of Union), intersect query will be:

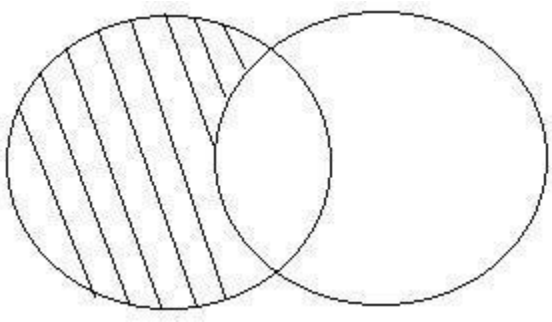
```
select * from First INTERSECT select * from second
```

The result table will be as below:

ID	Name
2	Adam

Minus:

Minus operation combines result of two Select statements and return only those results which belong to first set of result.



For the above tables (given in example of Union), Minus query will be:

```
select * from First MINUS select * from second
```

The result table will be:

ID	Name
1	John

DCL command:

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

- **System :** creating session, table etc are all types of system privilege.
- **Object :** any command or query to work on tables comes under object privilege.

DCL defines two commands,

- **Grant :** Gives user access privileges to database.
- **Revoke :** Take back permissions from user.

Example:

To give access of table faculty to user U2 command will be:

grant all on faculty to U2;

To revoke access of table faculty from user U2 command will be:

Revoke all on faculty from U2;

Views:

After a table is created, it may become necessary to prevent all users from accessing all columns of a table, for data security reasons. This would mean creating several tables having the appropriate number of columns and assigning specific users to each table as required. But this will increase redundancy. To reduce redundant data, oracle allows the creation of an object called a view. A view takes the output of a query and treats it as a table, therefore view can be thought of as a 'stored query' or a 'virtual table'. The tables upon which views are based are known as base tables.

The syntax for creating a view:

create [or replace][no][force]view <viewname>[column alias name] as <query>[with check option][read only][constraint];

1. Or replace – replaces an existing view with the same name, if it already exists
2. The force option- creates a view even though underlying table does not exist.
3. The column alias are used for the columns selected by the sub-query.
4. With read only – is to make sure that the data in the underlying table are not changed.

Example:

create view emp_vw as select * from emp;

Suppose we want to create view on employee table to see only employees of department 10, view can be written as:

create view emp_vw_dept as select & from emp where dept_no=10;

If we don' t want user to update data of table through views we can create read only views as below:

create or replace emp_vw as select * from emp with read only;

If we now fire query as below then it will give an error:

update emp_vw set sal=10000 where empno=7456;

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

Category_header

Cat_code	Cate_desc
01	super delux
02	delux
03	super fast
04	normal

route_Header

Route_id	Route_no	Cate_code	Origin	Destination	Fare	Distance	Capacity
101	33	01	Madurai	Madras	35	250	50
102	25	02	Trichy	Madurai	40	159	50
103	15	03	Thanjavur	Madurai	59	140	50
104	36	04	Madras	Banglore	79	375	50
105	40	01	Banglore	Madras	80	375	50
106	38	02	Madras	Madurai	39	250	50
107	39	03	Hydrabad	Madras	50	430	50
108	41	04	Madras	Cochin	47	576	50

Place Header:

Place_id	Place_name	Place_address	Bus_station
01	Madras	10, ptc road	Parrys
02	Madurai	21, canal bank road	Kknagar
03	Trichy	11, first cross road	Bheltown
04	Banglore	15, first main road	Cubbon park
05	Hydrabad	115,lake view road	Charminar
06	Thanjavur	12, temple road	Railway jn.

Fleet Header:

Fleet_id	Day	Route_id	Cat_code
01	10-apr-96	101	01
02	10-apr-96	101	01
03	10-apr-96	101	01
04	10-apr-96	102	02
05	10-apr-96	102	03
06	10-apr-96	103	04

Ticket Header:

Fleet_id	Ticket_no	Doi	Dot
01	01	10-apr-96	10-may-96
02	02	12-apr-96	5-may-96
03	03	21-apr-96	15-may-96

Time_travel	Board_place	Origin	Destinition
15:00:00	Parrys	Madrsa	Madurai
09:00:00	Kknagar	Madurai	Madras
21:00:00	Cubbon park	Banglore	Madras

Adults	Children	Total_fare	Route_id
1	1	60	101
2	1	60	102
4	2	400	101

Ticket Detail:

~~Adults~~ ticket_no ~~Fleet_id~~	Name	Sex	Age	Fare
01	Charu	F	24	14.00
01	Lathu	F	10	15.55
02	Anand	M	28	17.80
02	Guatham	M	28	16.00
02	Bala	M	09	17.65
05	Sandip	M	30	18.00

Route Detail:

Route_id	Place_id	Nonstop
105	01	N
012	02	S
106	01	S
108	05	N
106	02	N

Task:

1. **Create read only view route_vw on table route_header to display route_id, origin and destination.**
2. **Try to update any record of view route_vw and explain the error message.**
3. **Create view route_vw2 on table route_header with columns route_no, cat_code, origin and destination.**

4. **Display records of view route_vw2.**
5. **Try to insert record into view route_vw2 and state the output if possible else explain error message.**
6. **Create view route_category_vw to display those routes which belong to category ‘delux’.**

Practice Queries:

7. **Give average of the total fare.**
8. **Give the total collection of fare.**
9. **Which bus runs for minimum distance?**

10. **Give the total number of people who have traveled so far group by ticket number.**
11. **Find out total fare for routes with same origin.**
12. **Find out total fare for routes with origin as madras.**
13. **Calculate average fare for each ticket.**
14. **Find out details of fleets that run on route number 33 or 25.**
15. **Count how many male or female passengers have travelled till now.**
16. **Count total number of routes for each category except category 02.**
17. **Find out place names for which nonstop buses run.**
18. **Find out details of tickets from ticket header having more than one passenger.**
19. **Find out details of route having category code same as category code of a route having maximum distance.**
20. **Display distinct ticket number from both ticket_header and ticket_detail.**
21. **Display all ticket numbers from both ticket_header and ticket_detail.**
22. **Display only common fleet_id's that are present in fleet_header and ticket_header.**
23. **Select distinct route_id from route_header and not in route_detail using both the tables.**
24. **Display distinct route_id's from route header and route_detail.**
25. **Display all category_code from route_header and category_header.**
26. **Display only common place_id's that are present in place_header and route_detail.**
27. **Select common ticket_numbers from ticket_header and ticket_detail where the route_id's are smaller than the route_id which belong to place id as 01.**
28. **Select unique fleet_id's from ticket_header and fleet_header where the route_id is greater than route_id's in route_header which belongs to the category_code as 01.**

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: 30/09/2022
Date of Submission: 30/09/2022	Grade :

B.1 Commands and Output:

Task 1:

```
create view route_vw  as (select route_id, origin, destination from route_header) ;
```

Task 2:

```
update route_vw set origin = "velllore" where route_id=4;
```

Task 3:

```
create view route_vw2  as (select route_id,route_no,cat_code, origin, destination from
route_header) ;
```

Task 4:

```
select * from route_vw2;
```

Task 5:

```
insert into route_vw2(route_id,route_no,cat_code, origin, destination)
values(149,50,1,"Madrid","London");
```

route_id	route_no	cat_code	origin	destination
101	33	1	Madurai	Madras
102	25	2	Trichy	Madurai
103	15	3	Thanjavur	Madurai
104	36	4	Madras	Banglore
105	40	1	Banglore	Madras
106	38	2	Madras	Madurai
107	39	3	Hyderabad	Madras
108	41	4	Madras	Cochin

Task 6:

```
create view route_category_vw as (select route_id from category_header as c,route_header as rh
where c.cat_code=rh.cat_code and cat_desc=&#39;deluxe&#39;) ;
select * from route_category_vw;
```

route_id
102
106

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 distributor who has never supplied any item (using sub query).**
   - Select Dname from Distributor where Dno not in(Select Dno From Dist_Item);

b. **Count total number of items of each colour.**
   - Select Itemname,count(Itemname) From Item Group by colour;

c. **Count number of items supplied by each distributor.**
   - Select D.Dname,count(Di.Itemno) From Distributor as D, Dist_Item as Di where D.Dno=Di.Dno;

2. Justify the statement, "Views can be used for better manipulation of the given table".
- They can be used to store complex queries. Inserting into views and updating them help us in updating their underlying tables. Hence they serve for better modification and manipulation of the table.

3. Differentiate between 'Union and 'Union All' used for Databases.
- Union keeps all records that are input in the given query. It only keeps unique records.
- Union ALL extracts all the records and also retains duplicates.

4. Explain at least one real life Application of views to solve any real life problem.
- Views are virtual tables that serve in better manipulation of data in various relations.
- They help us in access specification for various users; help us in specifying relevant data to respective users.
- Hiding data and storing complex queries.

B.3 Conclusion:

After this experiment, I was able to implement the knowledge of relational algebra and structural query language to retrieve and manage data in relational databases.