DBMS
TASK-5
AIM: To implement Aggregate functions, Group By clause and Having clause

a) Aggregate Functions

Performing calculations on multiple rows

- Of a single column of a table
- And returning a single value.

The ISO standard defines five (5) aggregate functions namely;

1) COUNT
2) SUM
3) AVG
4) MIN
5) MAX

Why use aggregate functions.

From a business perspective, different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details.

>Aggregate functions allow us to easily produce summarized data from our database.

For instance, from our myflix database , management may require following reports

- Least rented movies.
- Most rented movies.
- Average number that each movie is rented out in a month.

We easily produce above reports using aggregate functions.

Let's look into aggregate functions in detail.

COUNT Function

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. **All aggregate functions by default exclude nulls values before working on the data.**

COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates.

The table shown below shows data in movierentals table

reference_ number	transaction_ date	return_date	membership_ number	movie_id	movie_ returned
11	20-06-2012	NULL	1	1	0
12	22-06-2012	25-06-2012	1	2	0
13	22-06-2012	25-06-2012	3	2	0
14	21-06-2012	24-06-2012	2	2	0
15	23-06-2012	NULL	3	3	0

Let's suppose that we want to get the number of times that the movie with id 2 has been rented out

SELECT COUNT(`movie_id`) FROM `movierentals` WHERE `movie_id` = 2;
Executing the above query in MySQL workbench against myflixdb gives us the following results.

COUNT('movie_id')
3

DISTINCT Keyword

The DISTINCT keyword that allows us to omit duplicates from our results. This is achieved by grouping similar values together .

To appreciate the concept of Distinct, lets execute a simple query

SELECT `movie_id` FROM `movierentals`;

movie_id
1
2
2
2
3

Now let's execute the same query with the distinct keyword -

```
SELECT DISTINCT `movie_id` FROM `movierentals`;
```

As shown below , distinct omits duplicate records from the results.

movie_id

1
2
3

MIN function

The MIN function **returns the smallest value in the specified table field**.

As an example, let's suppose we want to know the year in which the oldest movie in our library was released, we can use MySQL's MIN function to get the desired information.

The following query helps us achieve that

```
SELECT MIN(`year_released`) FROM `movies`;
```

Executing the above query in MySQL workbench against myflixdb gives us the following results.

MIN('year_released')
2005

MAX function :

Just as the name suggests, the MAX function is the opposite of the MIN function. It **returns the largest value from the specified table field**.

Let's assume we want to get the year that the latest movie in our database was released. We can easily use the MAX function to achieve that.

The following example returns the latest movie year released.

```
SELECT MAX(`year_released`)  FROM `movies`;
```

Executing the above query in MySQL workbench using myflixdb gives us the following results.

MAX('year_released')
2012

SUM function :

Suppose we want a report that gives total amount of payments made so far. We can use the MySQL **SUM** function which **returns the sum of all the values in the specified column**. **SUM works on numeric fields only**. **Null values are excluded from the result returned.**

The following table shows the data in payments table-

payment_ id	membership_ number	payment_ date	description	amount_ paid	external_ reference _number
1	1	23-07-2012	Movie rental payment	2500	11
2	1	25-07-2012	Movie rental payment	2000	12
3	3	30-07-2012	Movie rental payment	6000	NULL

The query shown below gets the all payments made and sums them up to return a single result.

```
SELECT SUM(`amount_paid`) FROM `payments`;
```

Executing the above query in MySQL workbench against the myflixdb gives the following results.

SUM('amount_paid')
10500

AVG function :

MySQL AVG function **returns the average of the values in a specified column**. Just like the SUM function, it **works only on numeric data types**.

Suppose we want to find the average amount paid. We can use the following query -

```
SELECT AVG(`amount_paid`) FROM `payments`;
```

Executing the above query in MySQL workbench, gives us the following results.

AVG('amount_paid')
3500

Summary

- MySQL supports all the five (5) ISO standard aggregate functions COUNT, SUM, AVG, MIN and MAX.
- SUM and AVG functions only work on numeric data.
- If you want to exclude duplicate values from the aggregate function results, use the DISTINCT keyword. The ALL keyword includes even duplicates. If nothing is specified the ALL is assumed as the default.
- Aggregate functions can be used in conjunction with other SQL clauses such as GROUP BY

b) ORDER BY in DBMS

- Sometimes the user may be interested in ***arranging the data in the table in some increasing or decreasing order of values***
- **Example**: If you want to display the details of all students based on descending order of their attendance or marks etc

ORDER BY clause

The `order by` clause is used to arrange the fetched data from the database table in ascending or descending order of data values based on one or more columns

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

ASC: Displays ***data based on increasing order of values in the column***
DESC: Displays ***data based on decreasing order of values in the column***

Consider the following sample *EMP* table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	10
7566	JONES	MANAGER	7839	02-APR-81	2975	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	20
7369	SMITH	CLERK	7902	17-DEC-80	800	20

Sorting according to a single column

Here we ***use only a single column for data arrangement***

Display the names, salaries of all employees based on decreasing order of their salaries

```
select ename,sal from emp
ORDER BY sal DESC;
```

O/P

```
7 ROWS SELECTED
```

ENAME	SAL
KING	5000
FORD	3000
SCOTT	3000
JONES	2975
BLAKE	2850
CLARK	2450
SMITH	800

The row which contains the highest salary is displayed first, the row with the second highest salary is displayed next and so on

Display the department number and employee name as per increasing order of department numbers

```
 select deptno,ename from emp
ORDER BY deptno;
```

O/P

```
7 rows selected
```

DEPTNO	ENAME
10	KING
10	CLARK
20	JONES
20	SMITH
20	SCOTT
20	FORD
30	BLAKE

As we have 3 departments in the employee table, first the rows of 10th department followed by 20 and 30 departments are displayed i.e increasing order of department numbers

Note:

ASC is optional and the Default value for order by, If you don't specify ASC or DSC by default data is arranged in ascending order, but for descending arrangement of data you must specify DESC in order by explicitly

Display all employees for working in a company based on seniority level

```
select ename,job,sal,hiredate
from emp
order by hiredate desc;
7 rows selected
```

ENAME	JOB	SAL	HIREDATE
SCOTT	ANALYST	3000	19-APR-87
FORD	ANALYST	3000	03-DEC-81
KING	PRESIDENT	5000	17-NOV-81
CLARK	MANAGER	2450	09-JUN-81
BLAKE	MANAGER	2850	01-MAY-81
JONES	MANAGER	2975	02-APR-81
SMITH	CLERK	800	17-DEC-80

Employees who are having an older hire date is displayed first followed by the second older hire date and so on

Sorting according to more than one column

You can ***specify more than one column in the order by clause this is done when you further need sorting internally based on a certain column***

Display the details of employees based on increasing order of Departments and in each department salary should be further arranged in highest to lowest order

```
select ename,sal,deptno from emp
ORDER BY deptno,sal DESC;
```

O/P

```
7 rows selected
```

ENAME	SAL	DEPTNO
KING	5000	10
CLARK	2450	10
FORD	3000	20
SCOTT	3000	20

JONES	2975	20
SMITH	800	20
BLAKE	2850	30

Before displaying the rows of30th department, rows or sorted based on the salaries of30 department in decreasing order i.e that is first highest salary in the 30th department and the second highest salary in 30th department followed by 20th department first highest salary followed by 20th department second highest salary and so on

Using where clause in ORDER BY

- You can also ***specify some conditions and filter the data by using where class and then sort the data***
- Here first ***where*** clause is to be followed by ***order by*** class

Display the names,sal,jobs of employees who are working as manager in highest to lowest order

```
select ename,job,sal from emp
where job='MANAGER'
order by sal desc ;
```

O/P

```
3 rows selected
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450

Here only records of managers are displayed as per decreasing order of their salaries

Specifying column numbers in order by

Instead of column names, ***you can also use the position number of columns are specified in the select statement***

```
select ename, sal, job
from emp
order by 2 DESC;
```

O/P

```
7 rows selected
```

ENAME	SAL	JOB

KING	5000	PRESIDENT
FORD	3000	ANALYST
SCOTT	3000	ANALYST
JONES	2975	MANAGER
BLAKE	2850	MANAGER
CLARK	2450	MANAGER
SMITH	800	CLERK

The second column in the select statement is 'sal' hence in above query sorting of rows is done as per increasing order of salaries

Sorting data based on expressions

User can specify basic arithmetic expressions and sort the rows based on this column generated by expression

Display the names, job, annual salary of all employees based on decreasing order of their annual salary

```
select ename ,sal*12 annsal ,job,hiredate
from emp
order by annsal desc;
```

O/P

```
7 ROWS SELECTED
```

ENAME	ANNSAL	JOB	HIREDATE
KING	60000	PRESIDENT	17-NOV-81
FORD	36000	ANALYST	03-DEC-81
SCOTT	36000	ANALYST	19-APR-87
JONES	35700	MANAGER	02-APR-81
BLAKE	34200	MANAGER	01-MAY-81
CLARK	29400	MANAGER	09-JUN-81
SMITH	9600	CLERK	17-DEC-80

Here we have performed operations on the existing ***sal column*** and the rows are sorted based on this newly formed display purpose column

c) Group by in DBMS

If you want to display the total salaries of each department in a company or else to display the highest paid employees of each branch of that company. this purpose is also called a by using a
`group by` clause
Group by clause in SQL used to arrange logically related data into groups with help of some functions i.e if a particular column has the same type of data in different rows then they can be organized this into a logical groups
General Syntax for GROUP BY clause

```
SELECT column_name(s),function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Simple example for GROUP BY

Consider a sample table *'emp'*

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	–	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	–	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	–	20

Display the number of employees present in each department

```
select deptno,count(*)
from emp
group by deptno;
```

O/P

```
3 rows selected.
```

DEPTNO	COUNT(*)
30	1
10	2
20	4

The above query display group wise count of each department

Display the highest played employees in each department

```
select deptno,max(sal)
from emp
group by deptno
```

O/P

```
3 rows selected.
```

DEPTNO	MAX(SAL)
30	2850
10	5000
20	3000

Department wise highest salary i.e logically categorizing category employees into department wise

Using WHERE clause in the GROUP BY

Using ***where clause rows can be pre excluded before dividing them into groups***, ***where*** clause must be specified before the ***group by*** clause when it is used in the query

Example

```
select deptno,max(sal)
from emp
where deptno!=30
group by deptno;
```

O/P

```
2 rows selected.
```

DEPTNO	MAX(SAL)
10	5000
20	3000

Highest salaries of all departments expect deptno 30 are displayed

Using ORDER BY with GROUP BY

We can also ***display the rows in sorted order after logical organizing into groups using order by clause*** along with group by clause

Example

```
select deptno,max(sal)
from emp
group by deptno
order by deptno;
```

O/P

```
3 ROWS SELECTED
```

DEPTNO	MAX(SAL)
10	5000
20	3000
30	2850

The above query displays the highest salaries in each department but first deptno 30 details followed by deptno 20 and deptno 30 i.e increasing order of department numbers

Points to note about GROUP BY clause

- ***GROUP BY clause is used only with the SELECT statement.***
- ***Where class is placed before group by class*** if it is used in the query.
- ***Order by class is placed after the group by class*** if it is used in the query group by class
- ***All the columns that are used in the select statement must be specified by using group by clause***
- ***If a group function is included in the select clause then we cannot use individual result columns***

d) HAVING clause

- Having Clause is ***used to place conditions and decide which group will be part of the final result***
- You cannot use aggregate functions like sum() count() etc with where clause
- Hence we need to use having clause if you want to specify conditions using this aggregate functions

Example

```
SELECT ename , SUM(sal) FROM emp
GROUP BY ename
HAVING SUM(sal)>2000;
```

O/P

```
6 rows selected.
```

ENAME	SUM(SAL)
JONES	2975
KING	5000
CLARK	2450
SCOTT	3000
BLAKE	2850
FORD	3000

Examples:

Table Name:orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.4	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.6	2012-04-25	3002	5001

Write a SQL statement to find the total purchase amount for all orders.

Sample table: orders

Sample Solution:

```
SELECT SUM (purch_amt)

FROM orders;
```

Output of the Query:

```
sum
```

```
17541.18
```

Write a SQL statement to find the number of salesman currently listing for all of their customers.
Sample Solution:

```
SELECT COUNT (DISTINCT salesman_id)
FROM orders;
```

Output of the Query:

```
count
6
```

Write a query in SQL to obtain the name of the physicians who are the head of the department.

Sample table: physician

```
employeeid |       name         |           position            |    ssn
-----------+--------------------+-------------------------------+-----------
         1 | John Dorian        | Staff Internist               | 111111111
         2 | Elliot Reid        | Attending Physician           | 222222222
         3 | Christopher Turk   | Surgical Attending Physician | 333333333
         4 | Percival Cox       | Senior Attending Physician    | 444444444
         5 | Bob Kelso          | Head Chief of Medicine        | 555555555
         6 | Todd Quinlan       | Surgical Attending Physician | 666666666
         7 | John Wen           | Surgical Attending Physician | 777777777
         8 | Keith Dudemeister | MD Resident                   | 888888888
         9 | Molly Clock        | Attending Psychiatrist        | 999999999
```

Sample table: department

```
departmentid |       name        | head
-------------+-------------------+------
           1 | General Medicine |    4
           2 | Surgery           |    7
           3 | Psychiatry        |    9
```

Sample Solution:

```
SELECT d.name AS "Department",

       p.name AS "Physician"

FROM department d,

     physician p

WHERE d.head=p.employeeid;
```

Sample Output:

```
    Department     |  Physician
-------------------+--------------
 General Medicine | Percival Cox
 Surgery          | John Wen
 Psychiatry       | Molly Clock
(3 rows)
```

Write a SQL statement to find the highest purchase amount with their ID and order date, for only those customers who have highest purchase amount in a day is more than 2000.

Sample table: orders

```
ord_no      purch_amt   ord_date    customer_id  salesman_id
----------  ----------  ----------  -----------  -----------
70001       150.5       2012-10-05  3005         5002
70009       270.65      2012-09-10  3001         5005
70002       65.26       2012-10-05  3002         5001
```

```
70004       110.5       2012-08-17  3009         5003
70007       948.5       2012-09-10  3005         5002
70005       2400.6      2012-07-27  3007         5001
70008       5760        2012-09-10  3002         5001
70010       1983.43     2012-10-10  3004         5006
70003       2480.4      2012-10-10  3009         5003
70012       250.45      2012-06-27  3008         5002
70011       75.29       2012-08-17  3003         5007
70013       3045.6      2012-04-25  3002         5001
```

Sample Solution:

```
SELECT customer_id,ord_date,MAX(purch_amt)

FROM orders

GROUP BY customer_id,ord_date

HAVING MAX(purch_amt)>2000.00;
```

Output of the Query:

```
customer_id ord_date    max
3007        2012-07-27  2400.60
3002        2012-09-10  5760.00
3009        2012-10-10  2480.40
3002        2012-04-25  3045.60
```

Write a SQL statement to find the highest purchase amount with their ID and order date, for those customers who have a higher purchase amount in a day is within the range 2000 and 6000.

Sample table: orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.4	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.6	2012-04-25	3002	5001

Sample Solution:

```
SELECT customer_id,ord_date,MAX(purch_amt)

FROM orders

GROUP BY customer_id,ord_date

HAVING MAX(purch_amt) BETWEEN 2000 AND 6000;
```

Copy
Output of the Query:

```
customer_id ord_date    max
3007        2012-07-27  2400.60
3002        2012-09-10  5760.00
```

```
3009          2012-10-10  2480.40
3002          2012-04-25  3045.60
:
```