

Name: Prashant Suresh Shirgave

Roll No: 3

Batch: T1

Class: TY(CSE-AIML)

Experiment No. 6

Title: Aggregate functions and Group by, having, between, Order by clauses

Objective: To understand the use of Aggregate functions and various clauses like group by, order by, having, between clause, etc.

Theory:

Aggregate functions An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

The most commonly used SQL aggregate functions are:

- MIN() - returns the smallest value within the selected column
- MAX() - returns the largest value within the selected column
- COUNT() - returns the number of rows in a set
- SUM() - returns the total sum of a numerical column
- AVG() - returns the average value of a numerical column

Syntax:

```
select groupfunction (column)
```

```
from tablename
```

```
[where condition(s)]
```

Group by clause: The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

Syntax:

```
select column , groupfunction (column)
```

```
from tablename
```

```
[where condition(s)] [group by column/exp]
```

Having clause: To apply condition on group, having clause is used.

Syntax:

```
select column , groupfunction (column)
```

```
from tablename [where condition(s)]
```

```
[group by column/exp]
```

```
[having groupcondition]
```

Between clause: The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

Syntax: SELECT column_name(s)

FROM table_name

WHERE column_name

BETWEEN value1 AND value2;

Order by clause: The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Syntax:

select columnlist

from tablename

[where condition(s)]

[order by column/exp[asc/desc]];

Consider the following schema.

account (account-number, branch-name, balance)

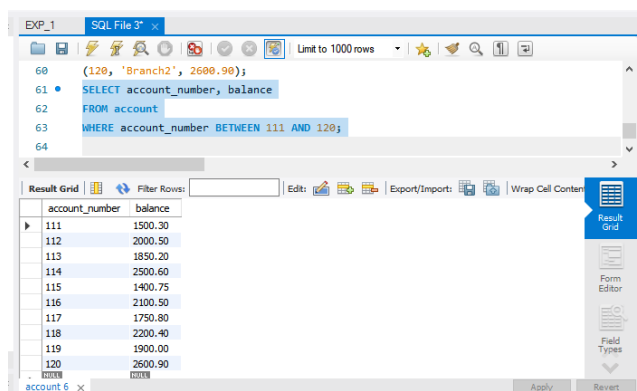
Execute the following queries:

1. Find the balance of account numbers in between '111' and '120'.

SELECT account_number, balance

FROM account

WHERE account_number BETWEEN 111 AND 120;



The screenshot shows a SQL IDE window titled 'EXP_1' with a tab 'SQL File 3'. The query editor contains the following SQL code:

```
(120, 'Branch2', 2600.90);
SELECT account_number, balance
FROM account
WHERE account_number BETWEEN 111 AND 120;
```

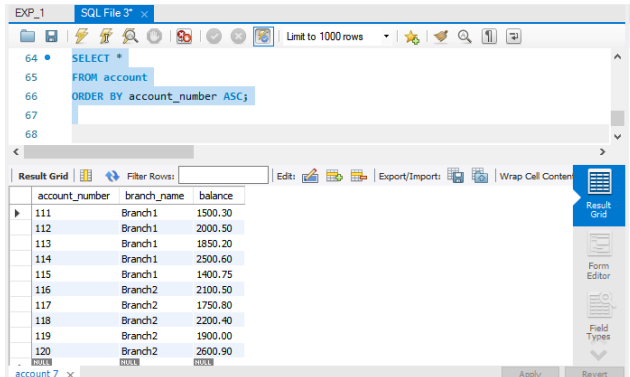
The 'Result Grid' shows the following data:

account_number	balance
111	1500.30
112	2000.50
113	1850.20
114	2500.60
115	1400.75
116	2100.50
117	1750.80
118	2200.40
119	1900.00
120	2600.90

The IDE interface includes a toolbar with icons for file operations, a 'Limit to 1000 rows' dropdown, and a sidebar with options like 'Result Grid', 'Form Editor', and 'Field Types'.

2. List entire account relation in ascending order by loan number.

```
SELECT * FROM account
ORDER BY account_number ASC;
```



The screenshot shows a SQL IDE window titled 'EXP_1' with a tab 'SQL File 3*'. The query editor contains the following SQL code:

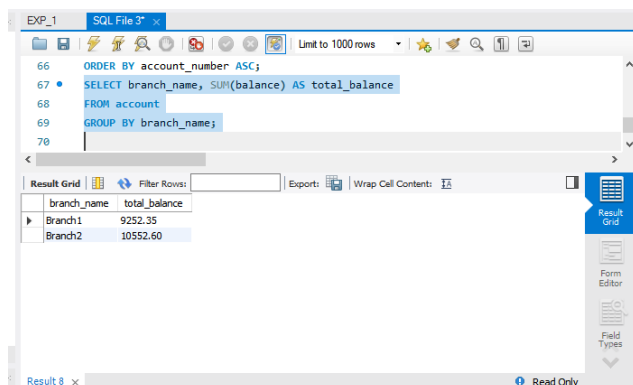
```
64 SELECT *
65 FROM account
66 ORDER BY account_number ASC;
67
68
```

The 'Result Grid' below the query shows the results of the query, which lists all accounts ordered by account number. The results are as follows:

account_number	branch_name	balance
111	Branch1	1500.30
112	Branch1	2000.50
113	Branch1	1850.20
114	Branch1	2500.60
115	Branch1	1400.75
116	Branch2	2100.50
117	Branch2	1750.80
118	Branch2	2200.40
119	Branch2	1900.00
120	Branch2	2600.90

3. Find the total account balance at each branch.

```
SELECT branch_name, SUM(balance) AS total_balance
FROM account GROUP BY branch_name;
```



The screenshot shows a SQL IDE window titled 'EXP_1' with a tab 'SQL File 3*'. The query editor contains the following SQL code:

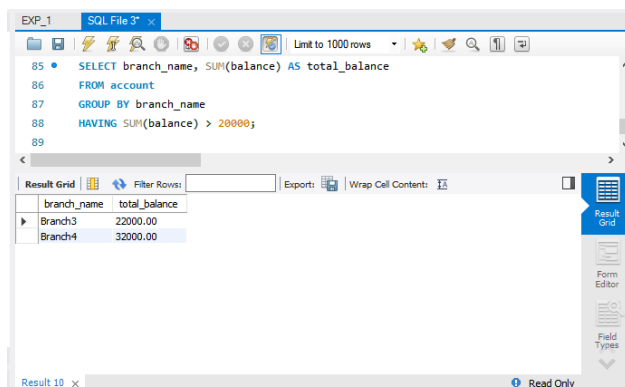
```
66 ORDER BY account_number ASC;
67 SELECT branch_name, SUM(balance) AS total_balance
68 FROM account
69 GROUP BY branch_name;
70
```

The 'Result Grid' below the query shows the results of the query, which lists the total account balance for each branch. The results are as follows:

branch_name	total_balance
Branch1	9252.35
Branch2	10552.60

4. Find those branches having total balance greater than 20,000

```
SELECT branch_name, SUM(balance) AS total_balance FROM account
GROUP BY branch_name HAVING SUM(balance) > 20000;
```



The screenshot shows a SQL IDE window titled 'EXP_1' with a tab 'SQL File 3*'. The query editor contains the following SQL code:

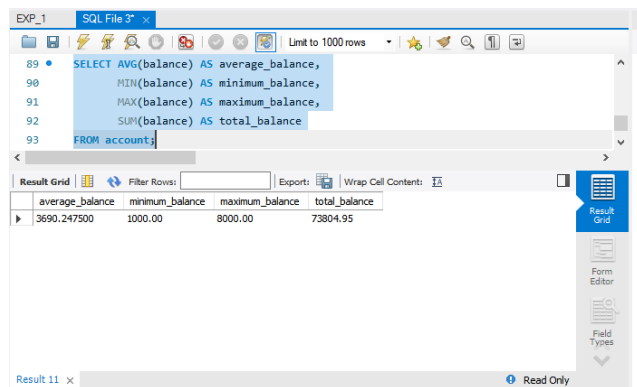
```
85 SELECT branch_name, SUM(balance) AS total_balance
86 FROM account
87 GROUP BY branch_name
88 HAVING SUM(balance) > 20000;
89
```

The 'Result Grid' below the query shows the results of the query, which lists the branches with a total balance greater than 20,000. The results are as follows:

branch_name	total_balance
Branch3	22000.00
Branch4	32000.00

5. Find the average, minimum, maximum, total account balance.

```
SELECT AVG(balance) AS average_balance,  
       MIN(balance) AS minimum_balance,  
       MAX(balance) AS maximum_balance,  
       SUM(balance) AS total_balance  
FROM account;
```



The screenshot shows the SQL Developer interface with a query window titled 'EXP_1' containing the following SQL code:

```
89 SELECT AVG(balance) AS average_balance,  
90        MIN(balance) AS minimum_balance,  
91        MAX(balance) AS maximum_balance,  
92        SUM(balance) AS total_balance  
93 FROM account;
```

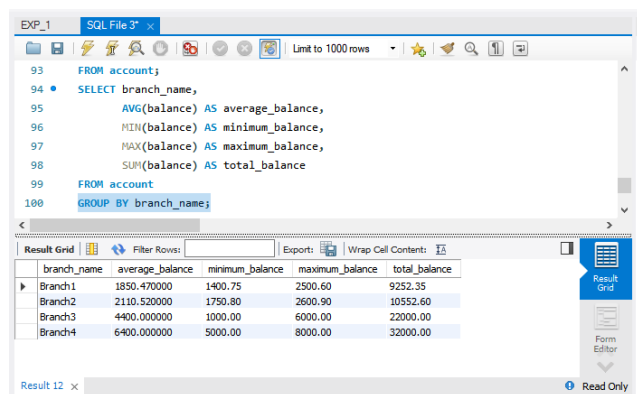
Below the query window, the 'Result Grid' displays the results of the query:

average_balance	minimum_balance	maximum_balance	total_balance
3690.247500	1000.00	8000.00	73804.95

The interface also shows a 'Filter Rows' section, an 'Export' button, and a 'Wrap Cell Contents' checkbox. The status bar at the bottom indicates 'Result 11' and 'Read Only'.

6. Find the average, minimum, maximum, total account balance at each branch.

```
SELECT branch_name,  
       AVG(balance) AS average_balance,  
       MIN(balance) AS minimum_balance,  
       MAX(balance) AS maximum_balance,  
       SUM(balance) AS total_balance  
FROM account  
GROUP BY branch_name;
```



The screenshot shows the SQL Developer interface with a query window titled 'EXP_1' containing the following SQL code:

```
93 FROM account;  
94 SELECT branch_name,  
95        AVG(balance) AS average_balance,  
96        MIN(balance) AS minimum_balance,  
97        MAX(balance) AS maximum_balance,  
98        SUM(balance) AS total_balance  
99 FROM account  
100 GROUP BY branch_name;
```

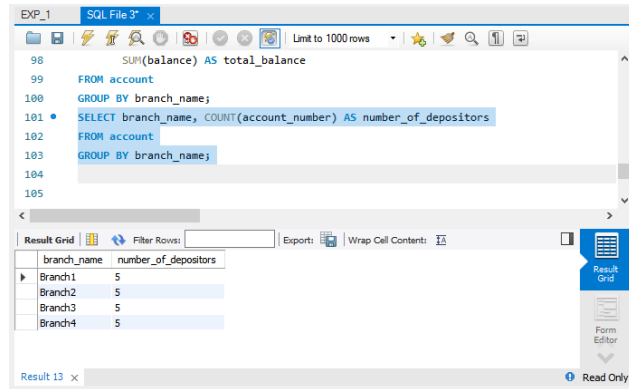
Below the query window, the 'Result Grid' displays the results of the query:

branch_name	average_balance	minimum_balance	maximum_balance	total_balance
Branch1	1850.470000	1400.75	2500.60	9252.35
Branch2	2110.520000	1750.80	2600.90	10552.60
Branch3	4400.000000	1000.00	6000.00	22000.00
Branch4	6400.000000	5000.00	8000.00	32000.00

The interface also shows a 'Filter Rows' section, an 'Export' button, and a 'Wrap Cell Contents' checkbox. The status bar at the bottom indicates 'Result 12' and 'Read Only'.

7. Find the number of depositors at each branch.

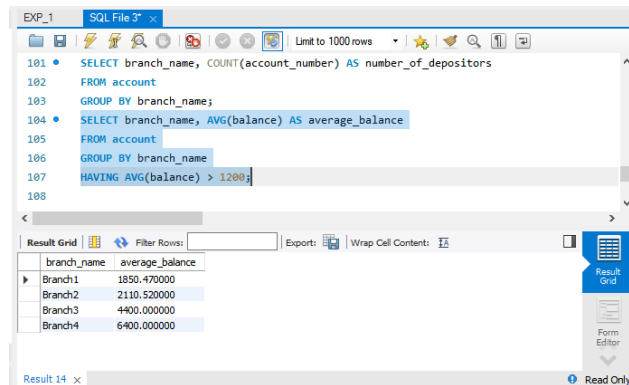
```
SELECT branch_name, COUNT(account_number) AS number_of_depositors  
FROM account  
GROUP BY branch_name;
```



branch_name	number_of_depositors
Branch1	5
Branch2	5
Branch3	5
Branch4	5

8. Find those branches where average account balance is more than 1200.

```
SELECT branch_name, AVG(balance) AS average_balance  
FROM account  
GROUP BY branch_name  
HAVING AVG(balance) > 1200;
```



branch_name	average_balance
Branch1	1850.470000
Branch2	2110.520000
Branch3	4400.000000
Branch4	6400.000000

Outcome: Students will be able to use aggregate functions and above clauses in their miniprojects & day-to-day problems solving.

