

# ANALYTIC FUNCTIONS

## 1. Partition by Clause

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

```
58 (006, 'Lead', '2016-06-11 00:00:00'),
59 (003, 'Lead', '2016-06-11 00:00:00');
60 • SELECT * FROM Worker;
61
62 • SELECT department, sum(salary) over (partition by department) as total_salary from Worker;
63
64 • SELECT department, sum(salary) over (partition by department order by salary) as total_salary from Worker;
65
66 • SELECT department, salary, row_number() over (partition by department order by salary DESC) as row_num from Worker;
```

The result grid displays the output of the query in line 62, showing the total salary for each department:

department	total_salary
Account	275000
Account	275000
Admin	1170000
Admin	1170000
Admin	1170000
Admin	1170000
HR	400000
HR	400000

## 2. Order by Clause

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

```
57 (007, 'Executive', '2016-06-11 00:00:00'),
58 (006, 'Lead', '2016-06-11 00:00:00'),
59 (003, 'Lead', '2016-06-11 00:00:00');
60 • SELECT * FROM Worker;
61
62 • SELECT department, sum(salary) over (partition by department) as total_salary from Worker;
63
64 • SELECT department, sum(salary) over (partition by department order by salary) as total_salary from Worker;
65
```

The result grid displays the output of the query in line 64, showing the total salary for each department, ordered by salary:

department	total_salary
Account	75000
Account	275000
Admin	80000
Admin	170000
Admin	1170000
Admin	1170000
HR	100000
HR	400000

### 3. Row number Function

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

```
58 (006, 'Lead', '2016-06-11 00:00:00'),
59 ( Execute the selected portion of the script or everything, if there is no selection
60 • SELECT * FROM Worker;
61
62 • SELECT department, sum(salary) over (partition by department) as total_salary from Worker;
63
64 • SELECT department, sum(salary) over (partition by department order by salary) as total_salary from Worker;
65
66 • SELECT department, salary, row_number() over (partition by department order by salary DESC) as row_num from Worker;
```

The result grid displays the output of the last query, showing the row number assigned to each worker within each department, ordered by salary in descending order.

department	salary	row_num
Account	200000	1
Account	75000	2
Admin	500000	1
Admin	500000	2
Admin	90000	3
Admin	80000	4
HR	300000	1
HR	100000	2

### 4. Rank function

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

```
61
62 • 5 Execute the selected portion of the script or everything, if there is no selection as total_salary from Worker;
63
64 • SELECT department, sum(salary) over (partition by department order by salary) as total_salary from Worker;
65
66 • SELECT department, salary, row_number() over (partition by department order by salary DESC) as row_num from Worker;
67
68 • SELECT department, salary, RANK() over (partition by department order by salary DESC) as rnk from Worker;
69
```

The result grid displays the output of the last query, showing the rank assigned to each worker within each department, ordered by salary in descending order.

department	salary	rnk
Account	200000	1
Account	75000	2
Admin	500000	1
Admin	500000	1
Admin	90000	3
Admin	80000	4
HR	300000	1
HR	100000	2

## 5. Dense\_rank Function

The screenshot shows a SQL IDE window with a script containing several queries. The query on line 70 is selected and highlighted:

```
70 • SELECT department, salary, DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS dense_rnk FROM Worker;
```

Below the script, the 'Result Grid' is displayed with the following data:

department	salary	dense_rnk
Account	200000	1
Account	75000	2
Admin	500000	1
Admin	500000	1
Admin	90000	2
Admin	80000	3
HR	300000	1
HR	100000	2

## 6. Lead Function

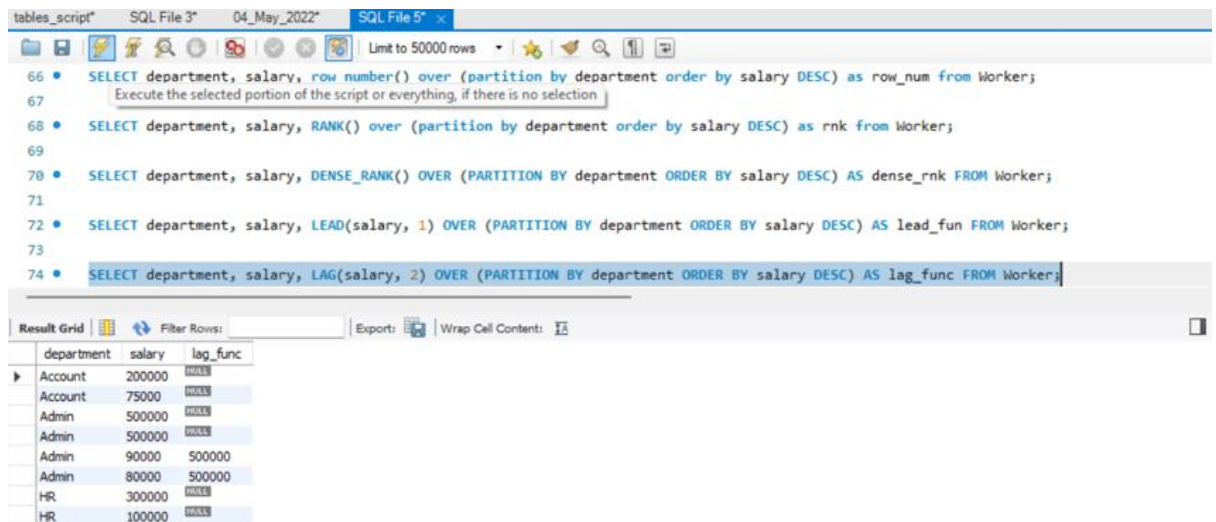
The screenshot shows a SQL IDE window with a script containing several queries. The query on line 72 is selected and highlighted:

```
72 • SELECT department, salary, LEAD(salary, 1) OVER (PARTITION BY department ORDER BY salary DESC) AS lead_fun FROM Worker;
```

Below the script, the 'Result Grid' is displayed with the following data:

department	salary	lead_fun
Account	200000	75000
Account	75000	NULL
Admin	500000	500000
Admin	500000	90000
Admin	90000	80000
Admin	80000	NULL
HR	300000	100000
HR	100000	NULL

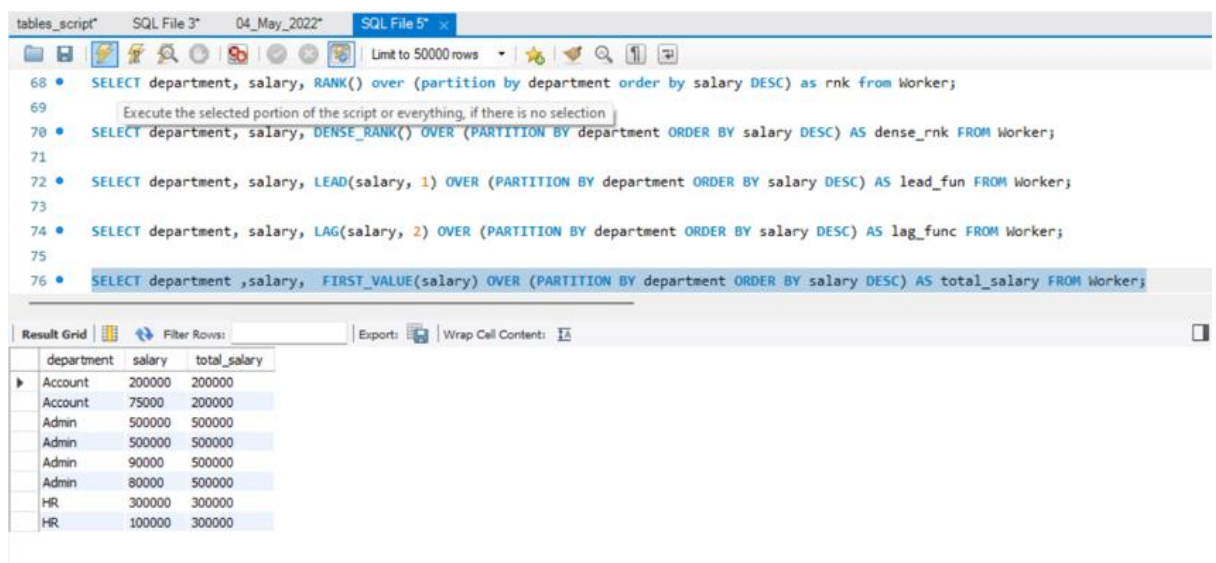
## 7. Lag Function



The screenshot shows a SQL script in a window titled 'SQL File 5'. The script contains several queries, with the last one highlighted: `SELECT department, salary, LAG(salary, 2) OVER (PARTITION BY department ORDER BY salary DESC) AS lag_func FROM Worker;`. Below the script, the 'Result Grid' displays the output of this query. The grid has three columns: 'department', 'salary', and 'lag\_func'. The data is sorted by department and then by salary in descending order. The 'lag\_func' column shows the salary of the employee two rows above in the same department.

department	salary	lag_func
Account	200000	NULL
Account	75000	NULL
Admin	500000	NULL
Admin	500000	NULL
Admin	90000	500000
Admin	80000	500000
HR	300000	NULL
HR	100000	NULL

## 8. First\_value Function



The screenshot shows a SQL script in a window titled 'SQL File 5'. The script contains several queries, with the last one highlighted: `SELECT department, salary, FIRST_VALUE(salary) OVER (PARTITION BY department ORDER BY salary DESC) AS total_salary FROM Worker;`. Below the script, the 'Result Grid' displays the output of this query. The grid has three columns: 'department', 'salary', and 'total\_salary'. The data is sorted by department and then by salary in descending order. The 'total\_salary' column shows the highest salary for each department.

department	salary	total_salary
Account	200000	200000
Account	75000	200000
Admin	500000	500000
Admin	500000	500000
Admin	90000	500000
Admin	80000	500000
HR	300000	300000
HR	100000	300000

## 9. Last\_Function

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

```
73
74 • SELECT department, salary, LAG(salary, 2) OVER (PARTITION BY department ORDER BY salary DESC) AS lag_func FROM Worker;
75
76 • SELECT department ,salary, FIRST_VALUE(salary) OVER (PARTITION BY department ORDER BY salary DESC) AS total_salary FROM Worker;
77
78 • SELECT department ,salary, LAST_VALUE(salary) OVER
79 (PARTITION BY department ORDER BY salary DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS total_salary FROM Worker;
80
81
```

The result grid displays the following data:

	department	salary	total_salary
▶	Account	200000	75000
	Account	75000	75000
	Admin	500000	80000
	Admin	500000	80000
	Admin	90000	80000
	Admin	80000	80000
	HR	300000	100000
	HR	100000	100000