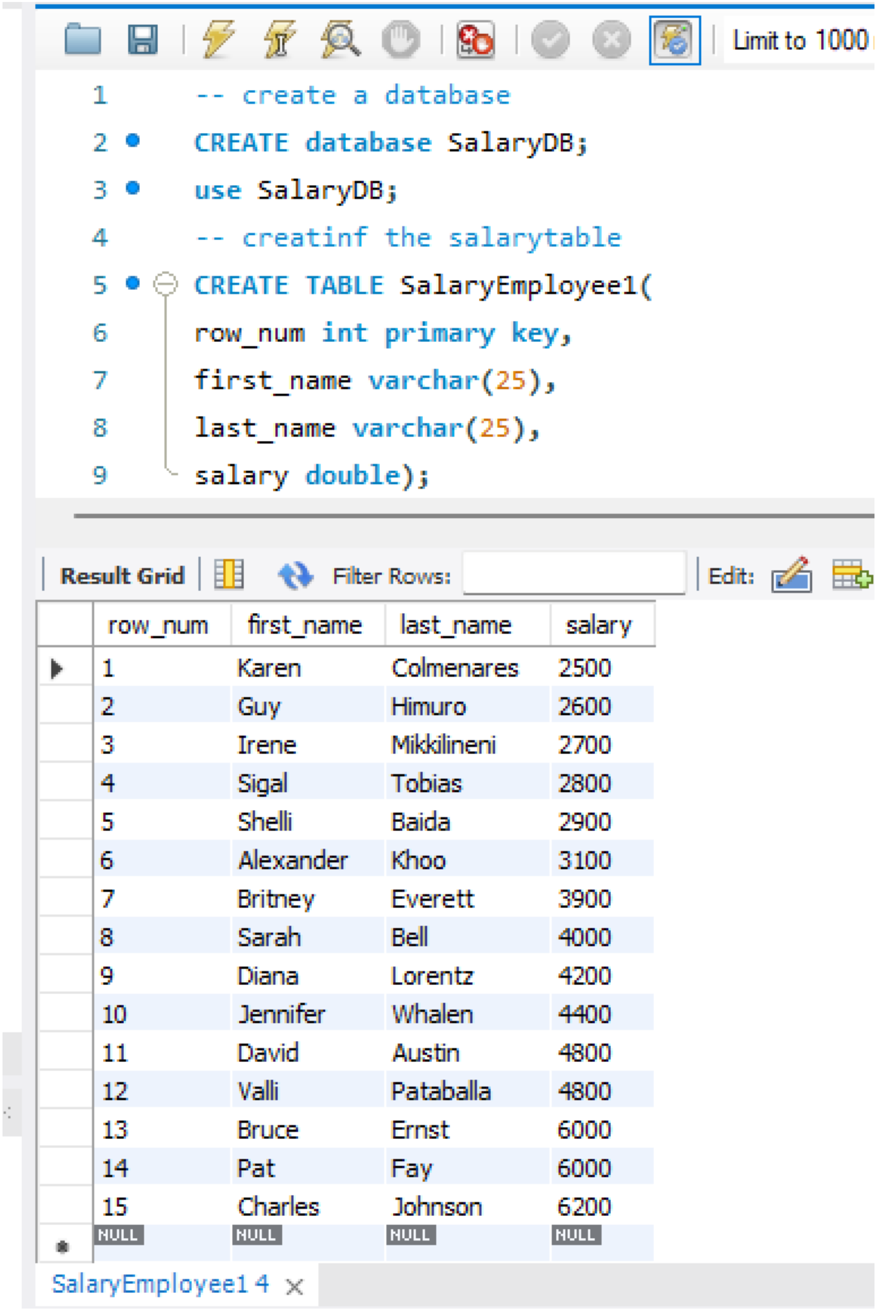
## Examination Guide:

## SQL Advanced window features: Queries, Output, and Formulas

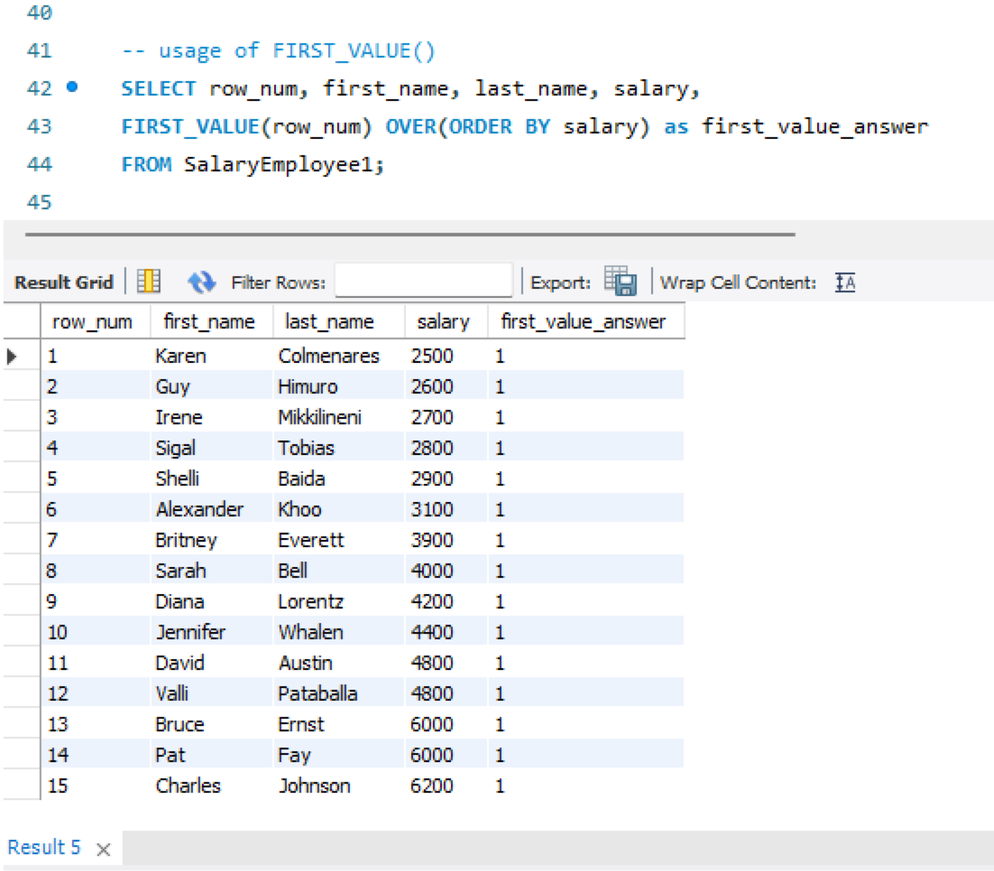
Using the dataset provided below, write the SQL queries, and expected output in table format for each of the window features supported by MySQL.



```sql
-- create a database
CREATE database SalaryDB;
use SalaryDB;
-- creatinf the salarytable
CREATE TABLE SalaryEmployee1(
row_num int primary key,
first_name varchar(25),
last_name varchar(25),
salary double);
```

| row_num | first_name | last_name | salary |
|---|---|---|---|
| 1 | Karen | Colmenares | 2500 |
| 2 | Guy | Himuro | 2600 |
| 3 | Irene | Mikkilineni | 2700 |
| 4 | Sigal | Tobias | 2800 |
| 5 | Shelli | Baida | 2900 |
| 6 | Alexander | Khoo | 3100 |
| 7 | Britney | Everett | 3900 |
| 8 | Sarah | Bell | 4000 |
| 9 | Diana | Lorentz | 4200 |
| 10 | Jennifer | Whalen | 4400 |
| 11 | David | Austin | 4800 |
| 12 | Valli | Pataballa | 4800 |
| 13 | Bruce | Ernst | 6000 |
| 14 | Pat | Fay | 6000 |
| 15 | Charles | Johnson | 6200 |
| NULL | NULL | NULL | NULL |

1. **Compute for the FIRST_VALUE() given the above data in the table.**
   FIRST_VALUE() – this window feature returns the first value in an ordered set of values. See screenshot for the SQL Command and the table output.



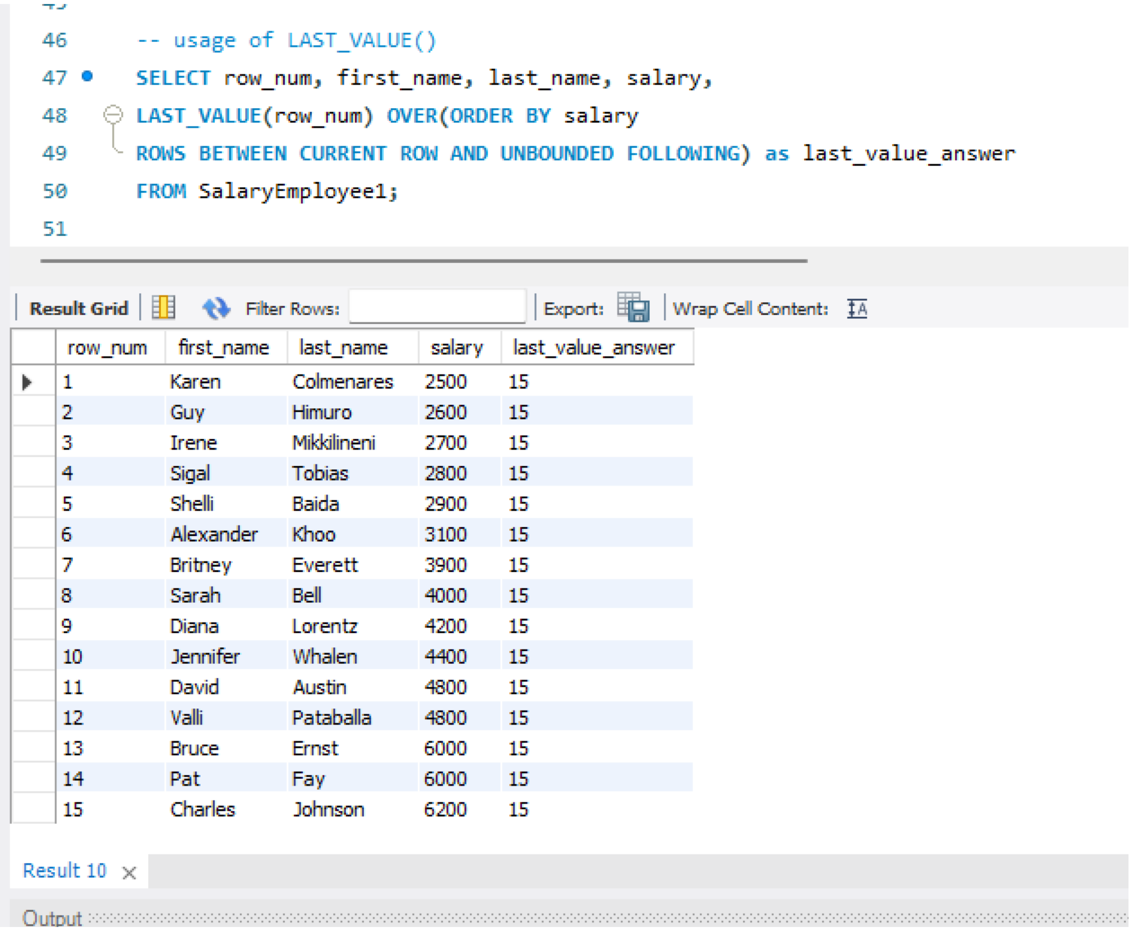
```sql

-- usage of FIRST_VALUE()
SELECT row_num, first_name, last_name, salary,
FIRST_VALUE(row_num) OVER(ORDER BY salary) as first_value_answer
FROM SalaryEmployee1;

```

| row_num | first_name | last_name | salary | first_value_answer |
|---|---|---|---|---|
| 1 | Karen | Colmenares | 2500 | 1 |
| 2 | Guy | Himuro | 2600 | 1 |
| 3 | Irene | Mikkilineni | 2700 | 1 |
| 4 | Sigal | Tobias | 2800 | 1 |
| 5 | Shelli | Baida | 2900 | 1 |
| 6 | Alexander | Khoo | 3100 | 1 |
| 7 | Britney | Everett | 3900 | 1 |
| 8 | Sarah | Bell | 4000 | 1 |
| 9 | Diana | Lorentz | 4200 | 1 |
| 10 | Jennifer | Whalen | 4400 | 1 |
| 11 | David | Austin | 4800 | 1 |
| 12 | Valli | Pataballa | 4800 | 1 |
| 13 | Bruce | Ernst | 6000 | 1 |
| 14 | Pat | Fay | 6000 | 1 |
| 15 | Charles | Johnson | 6200 | 1 |

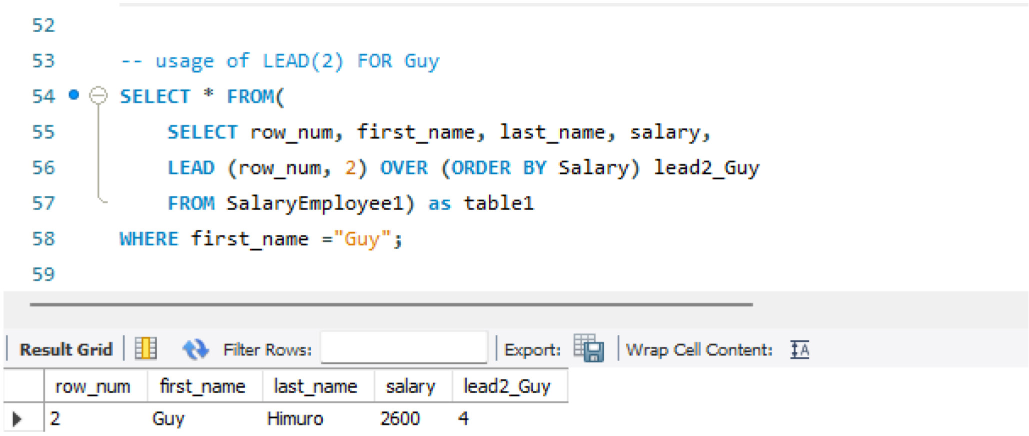
2. **Compute for the LAST_VALUE() given the above data in the table.**
LAST_VALUE() – this window feature returns the first value in an ordered set of values. See screenshot for the SQL Command and the table output.
**Note**: we use CURRENT ROW | UNBOUNDED PRECEDING | UNBOUNDED FOLLOWING to indicate where the window frame starts(preceding ) and ends (following).



```
-- usage of LAST_VALUE()
SELECT row_num, first_name, last_name, salary,
LAST_VALUE(row_num) OVER(ORDER BY salary
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) as last_value_answer
FROM SalaryEmployee1;
```

| row_num | first_name | last_name | salary | last_value_answer |
|---|---|---|---|---|
| 1 | Karen | Colmenares | 2500 | 15 |
| 2 | Guy | Himuro | 2600 | 15 |
| 3 | Irene | Mikkilineni | 2700 | 15 |
| 4 | Sigal | Tobias | 2800 | 15 |
| 5 | Shelli | Baida | 2900 | 15 |
| 6 | Alexander | Khoo | 3100 | 15 |
| 7 | Britney | Everett | 3900 | 15 |
| 8 | Sarah | Bell | 4000 | 15 |
| 9 | Diana | Lorentz | 4200 | 15 |
| 10 | Jennifer | Whalen | 4400 | 15 |
| 11 | David | Austin | 4800 | 15 |
| 12 | Valli | Pataballa | 4800 | 15 |
| 13 | Bruce | Ernst | 6000 | 15 |
| 14 | Pat | Fay | 6000 | 15 |
| 15 | Charles | Johnson | 6200 | 15 |

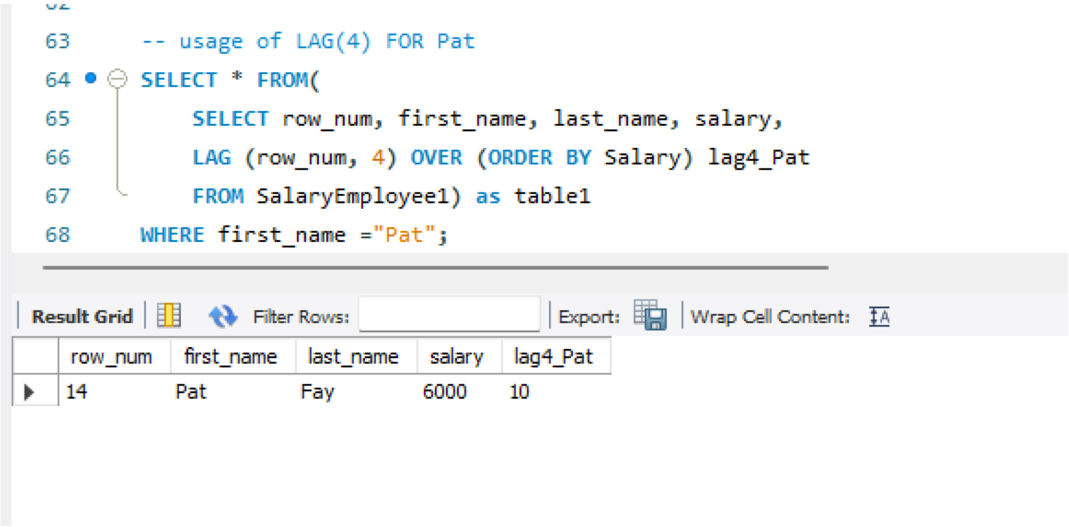
3. **Compute for the LEAD(2) for GUY.**
LEAD(n) – this window feature returns the nth NEXT rows from the current row. In this case current row = 2 for guy. Therefore, LEAD(2) for Guy.  Answer  row num= 4.
See screenshot for the SQL Command and the table output.



```
-- usage of LEAD(2) FOR Guy
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    LEAD (row_num, 2) OVER (ORDER BY Salary) lead2_Guy
    FROM SalaryEmployee1) as table1
WHERE first_name ="Guy";
```

| row_num | first_name | last_name | salary | lead2_Guy |
|---|---|---|---|---|
| 2 | Guy | Himuro | 2600 | 4 |

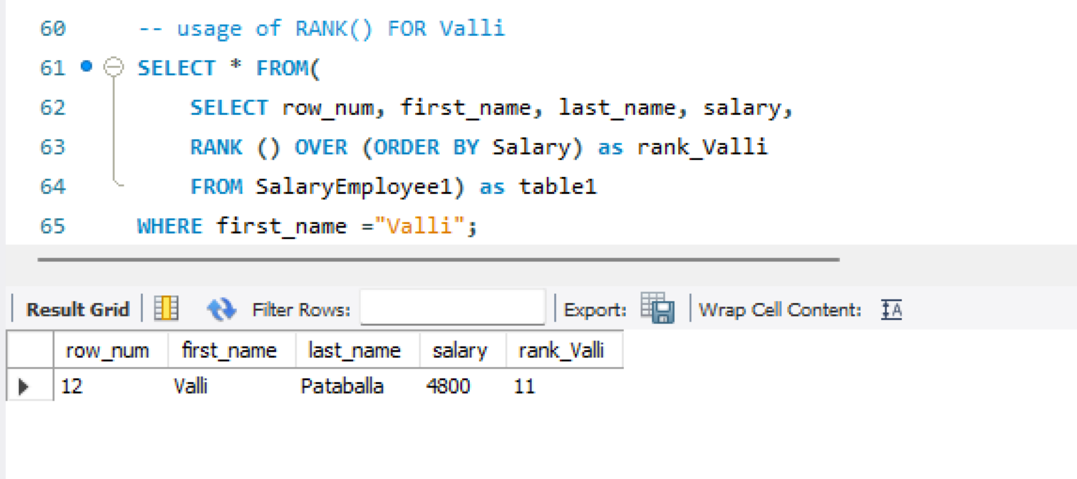
4. **Compute for the LAG(4) for Pat.**
LEAD(n) – this window feature returns the nth PREVIOUS row number from the current row. In this case the current row =14  for Pat. Therefore, LAG(4) for Guy.  Answer  row num= 10.
See screenshot for the SQL Command and the table output.



```
-- usage of LAG(4) FOR Pat
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    LAG (row_num, 4) OVER (ORDER BY Salary) lag4_Pat
    FROM SalaryEmployee1) as table1
WHERE first_name ="Pat";
```

| row_num | first_name | last_name | salary | lag4_Pat |
|---|---|---|---|---|
| 14 | Pat | Fay | 6000 | 10 |

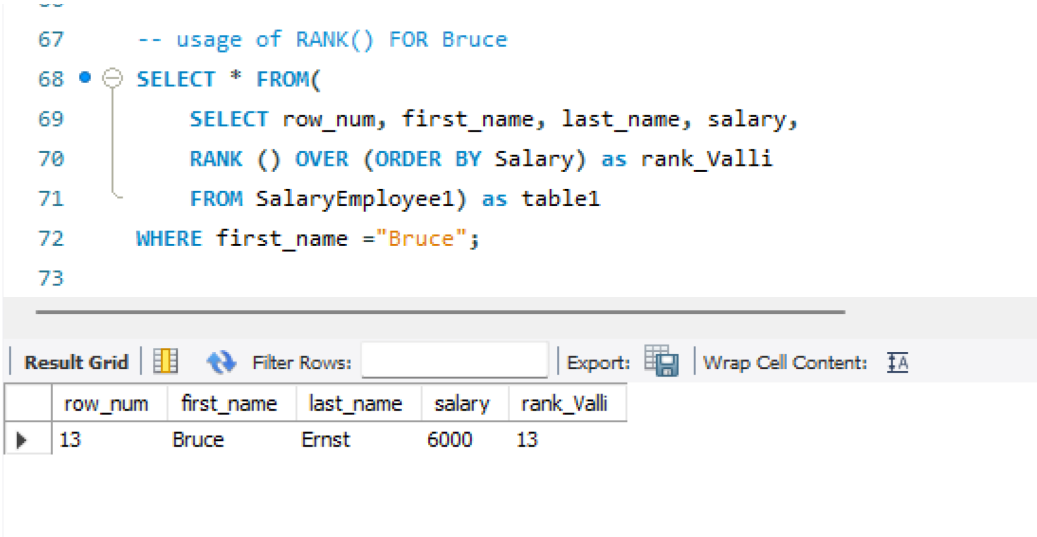
5. **Compute for the RANK() for Valli.**
RANK() – this window feature assigns a rank to each row within a partition of a result set. Note that RANK() – ranks values with gaps. (See DENSE_RANK() which ranks without gaps). Our dataset is ordered by Salary, and the rankings are based on salary.
See screenshot for the SQL Command and the table output.



```
-- usage of RANK() FOR Valli
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    RANK () OVER (ORDER BY Salary) as rank_Valli
    FROM SalaryEmployee1) as table1
WHERE first_name ="Valli";
```

| row_num | first_name | last_name | salary | rank_Valli |
|---|---|---|---|---|
| 12 | Valli | Pataballa | 4800 | 11 |

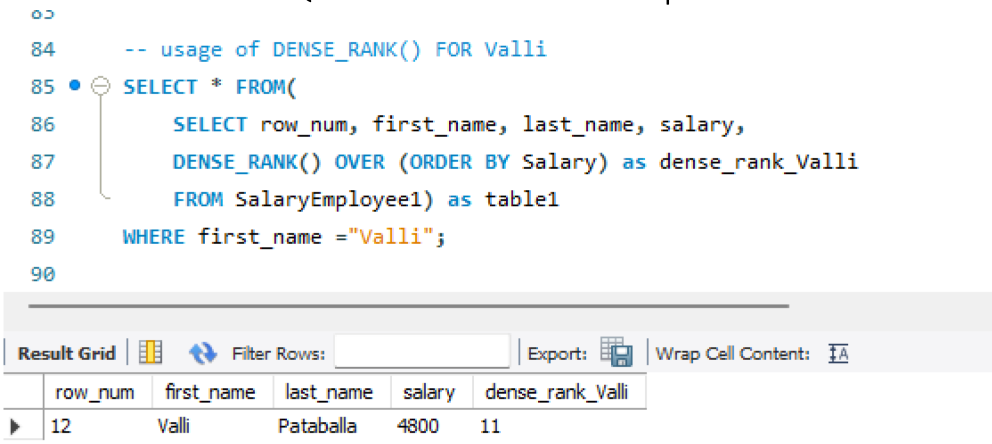
6. **Compute the RANK() for Bruce.**
RANK() – this window feature assigns a rank to each row within a partition of a result set. Note that RANK() – ranks results with gaps. (See DENSE_RANK() which ranks without gaps). Our dataset is ordered by Salary, and the rankings are based on salary.
See screenshot for the SQL Command and the table output.



```
-- usage of RANK() FOR Bruce
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    RANK () OVER (ORDER BY Salary) as rank_Valli
    FROM SalaryEmployee1) as table1
WHERE first_name ="Bruce";
```

| row_num | first_name | last_name | salary | rank_Valli |
|---|---|---|---|---|
| 13 | Bruce | Ernst | 6000 | 13 |

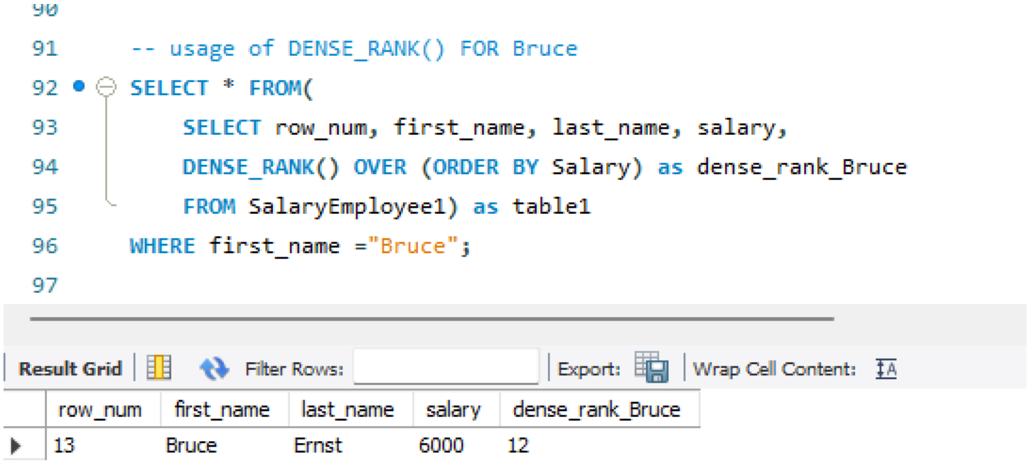
7. **Compute the DENSE_RANK() for Valli.**
DENSE_RANK() – this window feature assigns a rank to each row within a partition of a result set, with no gaps in ranking values. Our dataset is ordered by Salary, and the rankings are based on salary.
See screenshot for the SQL Command and the table output.



```
-- usage of DENSE_RANK() FOR Valli
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    DENSE_RANK() OVER (ORDER BY Salary) as dense_rank_Valli
    FROM SalaryEmployee1) as table1
WHERE first_name ="Valli";
```

| row_num | first_name | last_name | salary | dense_rank_Valli |
|---|---|---|---|---|
| 12 | Valli | Pataballa | 4800 | 11 |

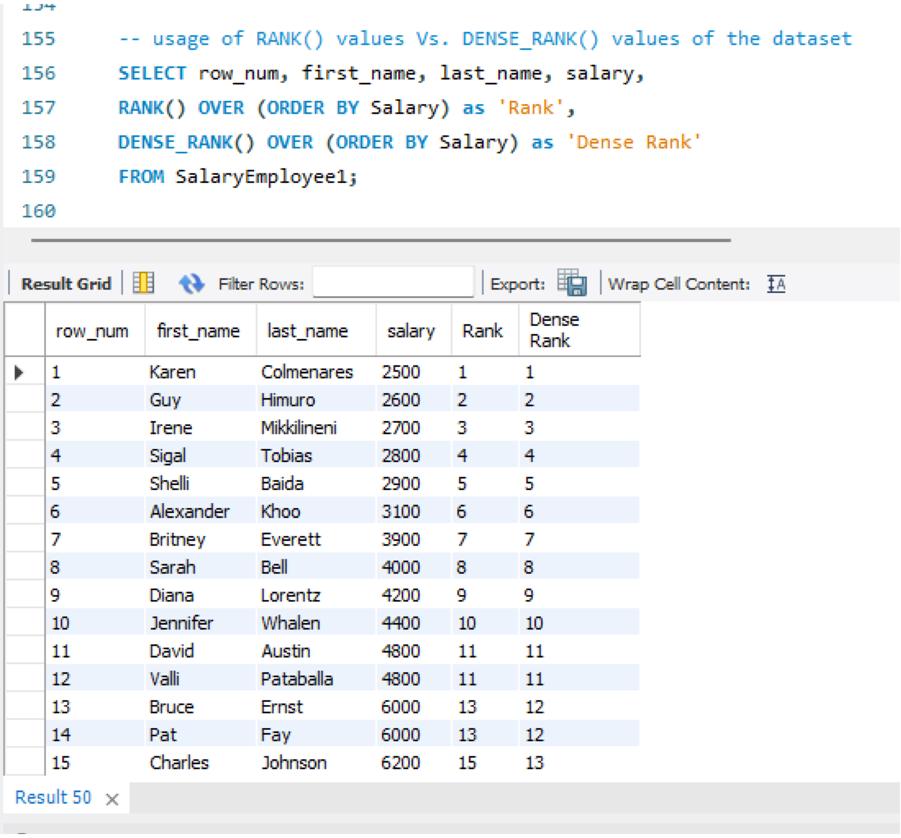
8. **Compute the DENSE_RANK() for Bruce.**
DENSE_RANK() – this window feature assigns a rank to each row within a partition of a result set, with no gaps in ranking values. Our dataset is ordered by Salary, and the rankings are based on salary.
See screenshot for the SQL Command and the table output.



```
-- usage of DENSE_RANK() FOR Bruce
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    DENSE_RANK() OVER (ORDER BY Salary) as dense_rank_Bruce
    FROM SalaryEmployee1) as table1
WHERE first_name ="Bruce";
```

| row_num | first_name | last_name | salary | dense_rank_Bruce |
|---|---|---|---|---|
| 13 | Bruce | Ernst | 6000 | 12 |

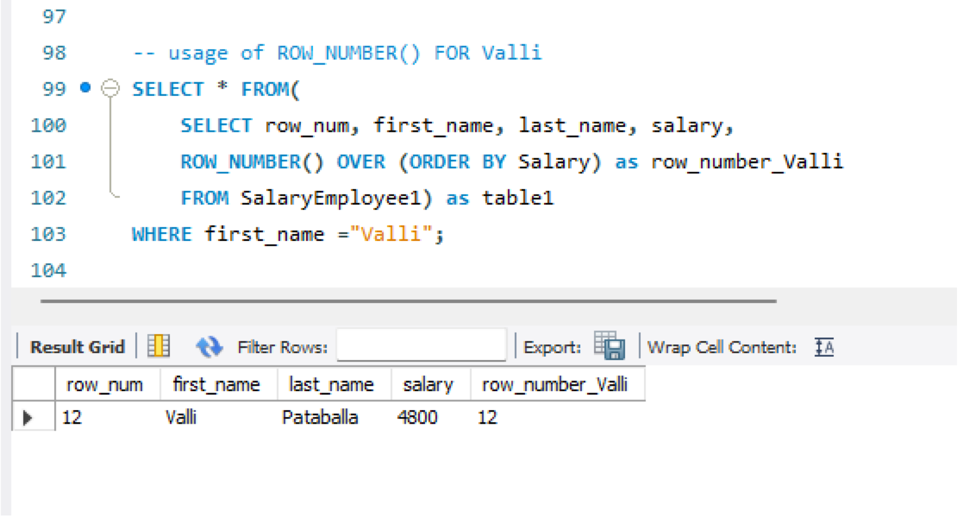
9. **Write a query computing for RANK() and DENSE_RANK() of the same dataset in the question.**
Notice the difference in the values -- RANK() Vs. DENSE-RANK()



```
-- usage of RANK() values Vs. DENSE_RANK() values of the dataset
SELECT row_num, first_name, last_name, salary,
RANK() OVER (ORDER BY Salary) as 'Rank',
DENSE_RANK() OVER (ORDER BY Salary) as 'Dense Rank'
FROM SalaryEmployee1;
```

| row_num | first_name | last_name | salary | Rank | Dense Rank |
|---|---|---|---|---|---|
| 1 | Karen | Colmenares | 2500 | 1 | 1 |
| 2 | Guy | Himuro | 2600 | 2 | 2 |
| 3 | Irene | Mikkilineni | 2700 | 3 | 3 |
| 4 | Sigal | Tobias | 2800 | 4 | 4 |
| 5 | Shelli | Baida | 2900 | 5 | 5 |
| 6 | Alexander | Khoo | 3100 | 6 | 6 |
| 7 | Britney | Everett | 3900 | 7 | 7 |
| 8 | Sarah | Bell | 4000 | 8 | 8 |
| 9 | Diana | Lorentz | 4200 | 9 | 9 |
| 10 | Jennifer | Whalen | 4400 | 10 | 10 |
| 11 | David | Austin | 4800 | 11 | 11 |
| 12 | Valli | Pataballa | 4800 | 11 | 11 |
| 13 | Bruce | Ernst | 6000 | 13 | 12 |
| 14 | Pat | Fay | 6000 | 13 | 12 |
| 15 | Charles | Johnson | 6200 | 15 | 13 |

10. **Compute the ROW_NUMBER() for Valli.**
ROW_NUMBER() – this window feature assigns a sequential integer to each row within the partition of a result set. The row number starts with 1 for the first row in each partition.
See screenshot for the SQL Command and the table output.

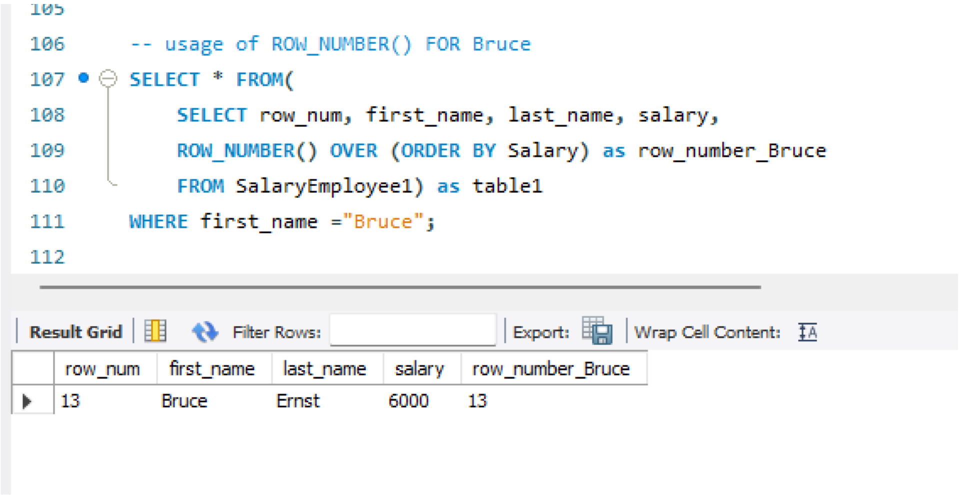


```
-- usage of ROW_NUMBER() FOR Valli
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    ROW_NUMBER() OVER (ORDER BY Salary) as row_number_Valli
    FROM SalaryEmployee1) as table1
WHERE first_name ="Valli";
```

| row_num | first_name | last_name | salary | row_number_Valli |
|---|---|---|---|---|
| 12 | Valli | Pataballa | 4800 | 12 |

**11. Compute the ROW_NUMBER() for Bruce.**

ROW_NUMBER() – this window feature assigns a sequential integer to each row within the partition of a result set. The row number starts with 1 for the first row in each partition. See screenshot for the SQL Command and the table output.



```
-- usage of ROW_NUMBER() FOR Bruce
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    ROW_NUMBER() OVER (ORDER BY Salary) as row_number_Bruce
    FROM SalaryEmployee1) as table1
WHERE first_name ="Bruce";
```

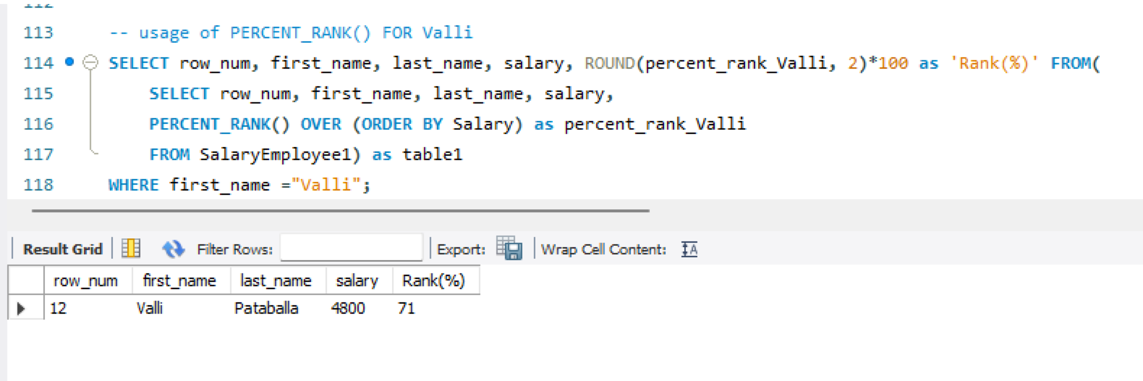
| row_num | first_name | last_name | salary | row_number_Bruce |
|---|---|---|---|---|
| 13 | Bruce | Ernst | 6000 | 13 |

12. **Compute the PERCENT_RANK() for Valli.**

PERCENT_RANK() – this window feature calculates the percentile ranking of a row within a partition or result set.

PERCENT_RANK() calculates the rank of that row minus one, divided by 1 less than number of rows in the evaluated partition or query result set: **Formula : ((rank - 1) / (total_rows - 1)) * 100**

See screenshot for the SQL Command and the table output.



```
-- usage of PERCENT_RANK() FOR Valli
SELECT row_num, first_name, last_name, salary, ROUND(percent_rank_Valli, 2)*100 as 'Rank(%)' FROM(
    SELECT row_num, first_name, last_name, salary,
    PERCENT_RANK() OVER (ORDER BY Salary) as percent_rank_Valli
    FROM SalaryEmployee1) as table1
WHERE first_name ="Valli";
```

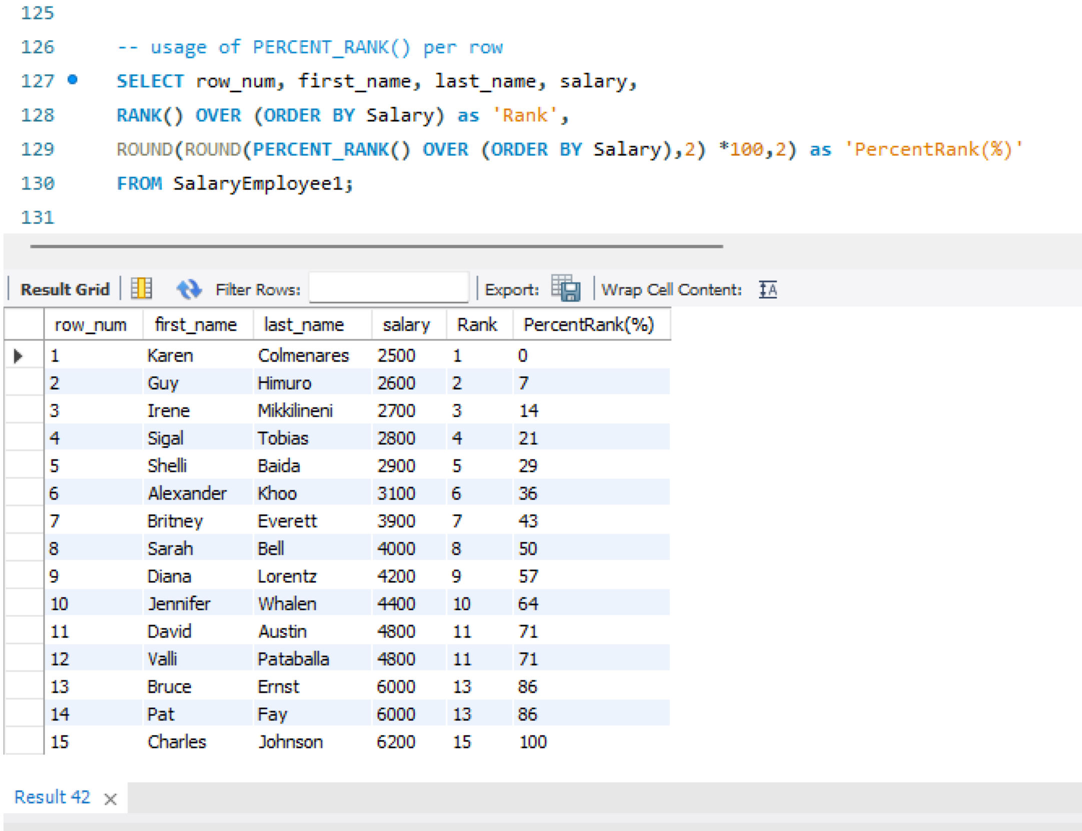
| row_num | first_name | last_name | salary | Rank(%) |
|---|---|---|---|---|
| 12 | Valli | Pataballa | 4800 | 71 |

13. **Compute the PERCENT_RANK() Per row/record.**

PERCENT_RANK() – this window feature calculates the percentile ranking of a row within a partition or result set.

PERCENT_RANK() calculates the rank of that row minus one, divided by 1 less than number of rows in the evaluated partition or query result set: **Formula : ((rank - 1) / (total_rows - 1)) * 100**

See screenshot for the SQL Command and the table output.



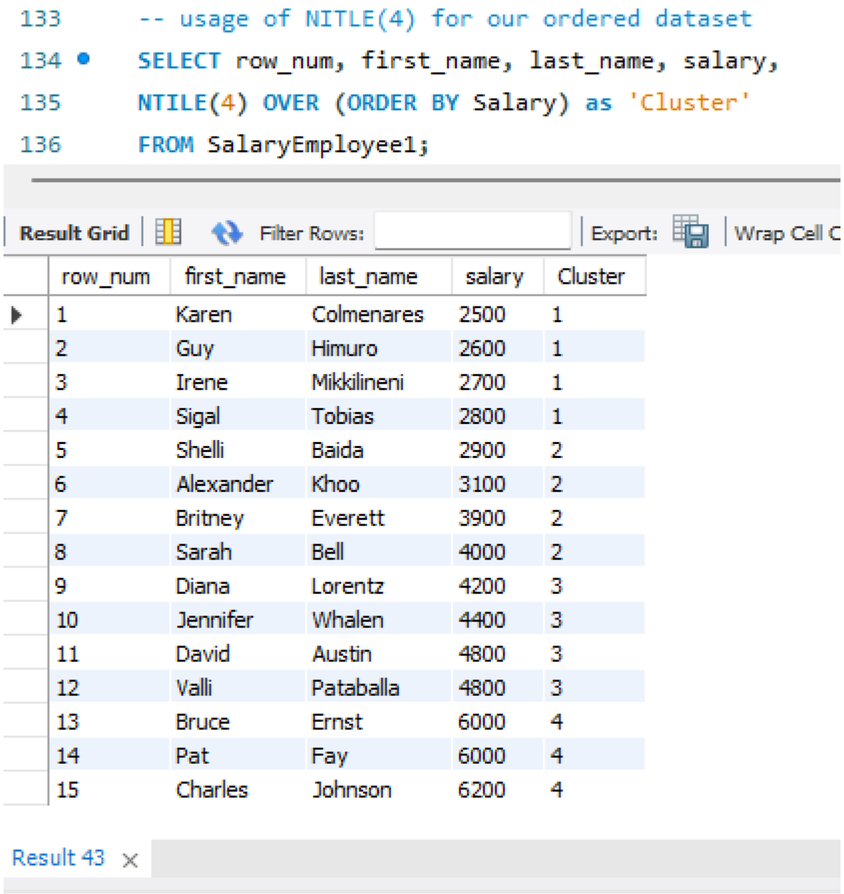
```
-- usage of PERCENT_RANK() per row
SELECT row_num, first_name, last_name, salary,
RANK() OVER (ORDER BY Salary) as 'Rank',
ROUND(ROUND(PERCENT_RANK() OVER (ORDER BY Salary),2) *100,2) as 'PercentRank(%)'
FROM SalaryEmployee1;
```

| row_num | first_name | last_name | salary | Rank | PercentRank(%) |
|---|---|---|---|---|---|
| 1 | Karen | Colmenares | 2500 | 1 | 0 |
| 2 | Guy | Himuro | 2600 | 2 | 7 |
| 3 | Irene | Mikkilineni | 2700 | 3 | 14 |
| 4 | Sigal | Tobias | 2800 | 4 | 21 |
| 5 | Shelli | Baida | 2900 | 5 | 29 |
| 6 | Alexander | Khoo | 3100 | 6 | 36 |
| 7 | Britney | Everett | 3900 | 7 | 43 |
| 8 | Sarah | Bell | 4000 | 8 | 50 |
| 9 | Diana | Lorentz | 4200 | 9 | 57 |
| 10 | Jennifer | Whalen | 4400 | 10 | 64 |
| 11 | David | Austin | 4800 | 11 | 71 |
| 12 | Valli | Pataballa | 4800 | 11 | 71 |
| 13 | Bruce | Ernst | 6000 | 13 | 86 |
| 14 | Pat | Fay | 6000 | 13 | 86 |
| 15 | Charles | Johnson | 6200 | 15 | 100 |

14. **Compute the NTILE(n) of the given dataset.**

NTILE (n) – this window feature distributes rows of an ordered partition into a specified number of approximately equal groups, or buckets. It assigns each group a bucket number starting from one. For example, NTILE(4) divides the datasets into 4 equal portions by assigning a number to the same bucket or group.

See screenshot for the SQL Command and the table output.



```
-- usage of NITLE(4) for our ordered dataset
SELECT row_num, first_name, last_name, salary,
NTILE(4) OVER (ORDER BY Salary) as 'Cluster'
FROM SalaryEmployee1;
```

| row_num | first_name | last_name | salary | Cluster |
|---|---|---|---|---|
| 1 | Karen | Colmenares | 2500 | 1 |
| 2 | Guy | Himuro | 2600 | 1 |
| 3 | Irene | Mikkilineni | 2700 | 1 |
| 4 | Sigal | Tobias | 2800 | 1 |
| 5 | Shelli | Baida | 2900 | 2 |
| 6 | Alexander | Khoo | 3100 | 2 |
| 7 | Britney | Everett | 3900 | 2 |
| 8 | Sarah | Bell | 4000 | 2 |
| 9 | Diana | Lorentz | 4200 | 3 |
| 10 | Jennifer | Whalen | 4400 | 3 |
| 11 | David | Austin | 4800 | 3 |
| 12 | Valli | Pataballa | 4800 | 3 |
| 13 | Bruce | Ernst | 6000 | 4 |
| 14 | Pat | Fay | 6000 | 4 |
| 15 | Charles | Johnson | 6200 | 4 |

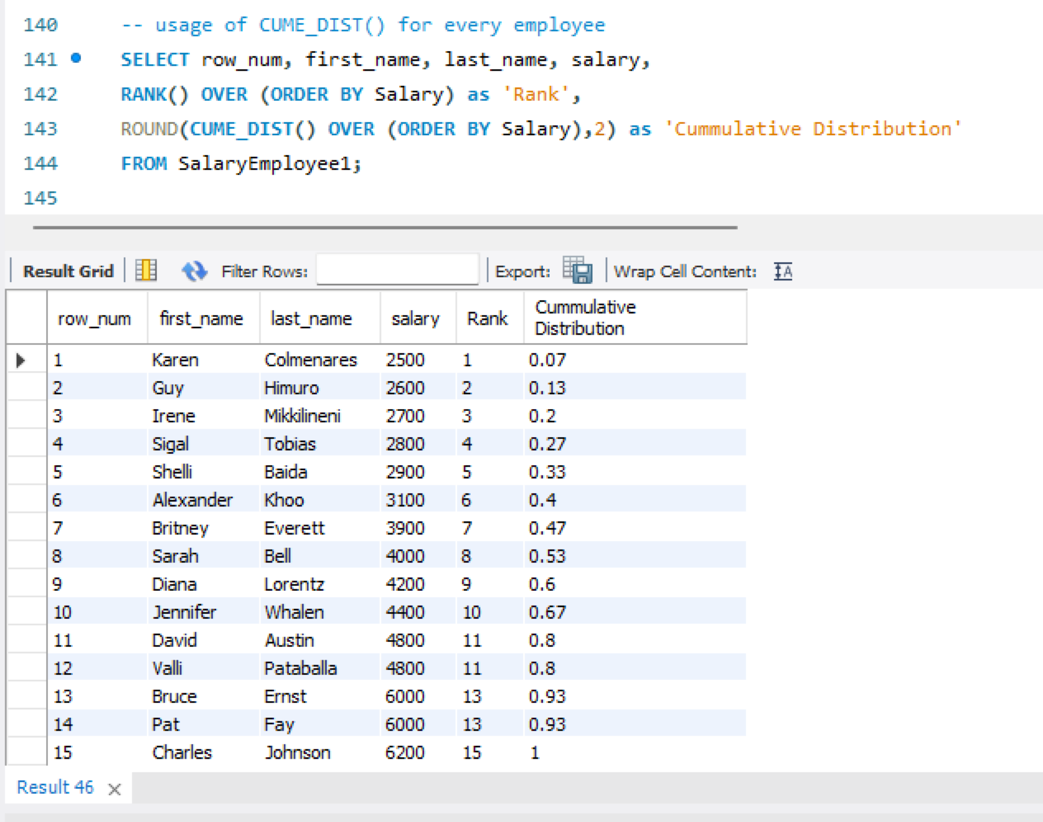
**15. Compute the CUME_DIST() for every record of the given dataset.**

CUME_DIST() – this window feature calculates the calculate a cumulative distribution of a value within a group of values. . It represents the number of rows with values less than or equal to that row's value divided by the total number of row(value ranges from 0 and 1)

**Formula : ROW_NUMBER / total_rows**

See screenshot for the SQL Command and the table output.

**NOTE**: Salary with the same rank MUST have the same CUME_DIST value



```
-- usage of CUME_DIST() for every employee
SELECT row_num, first_name, last_name, salary,
RANK() OVER (ORDER BY Salary) as 'Rank',
ROUND(CUME_DIST() OVER (ORDER BY Salary),2) as 'Cummulative Distribution'
FROM SalaryEmployee1;
```

| row_num | first_name | last_name | salary | Rank | Cummulative Distribution |
|---|---|---|---|---|---|
| 1 | Karen | Colmenares | 2500 | 1 | 0.07 |
| 2 | Guy | Himuro | 2600 | 2 | 0.13 |
| 3 | Irene | Mikkilineni | 2700 | 3 | 0.2 |
| 4 | Sigal | Tobias | 2800 | 4 | 0.27 |
| 5 | Shelli | Baida | 2900 | 5 | 0.33 |
| 6 | Alexander | Khoo | 3100 | 6 | 0.4 |
| 7 | Britney | Everett | 3900 | 7 | 0.47 |
| 8 | Sarah | Bell | 4000 | 8 | 0.53 |
| 9 | Diana | Lorentz | 4200 | 9 | 0.6 |
| 10 | Jennifer | Whalen | 4400 | 10 | 0.67 |
| 11 | David | Austin | 4800 | 11 | 0.8 |
| 12 | Valli | Pataballa | 4800 | 11 | 0.8 |
| 13 | Bruce | Ernst | 6000 | 13 | 0.93 |
| 14 | Pat | Fay | 6000 | 13 | 0.93 |
| 15 | Charles | Johnson | 6200 | 15 | 1 |

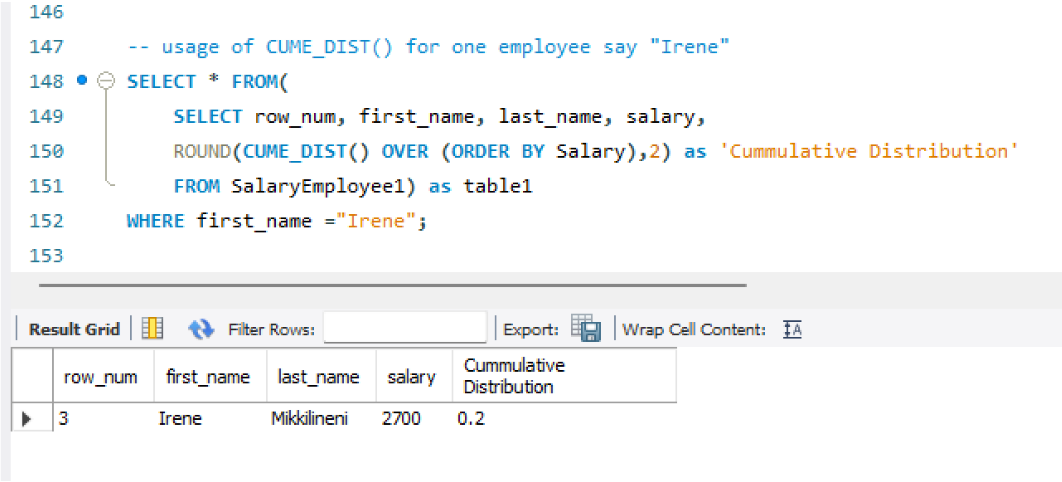
16. **Compute the CUME_DIST() for one employee "Irene".**

CUME_DIST() – this window feature calculates the calculate a cumulative distribution of a value within a group of values. . It represents the number of rows with values less than or equal to that row's value divided by the total number of row(value ranges from 0 and 1)

**Formula : ROW_NUMBER / total_rows**

See screenshot for the SQL Command and the table output.

**NOTE**: Salary with the same rank MUST have the same CUME_DIST value



```

-- usage of CUME_DIST() for one employee say "Irene"
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    ROUND(CUME_DIST() OVER (ORDER BY Salary),2) as 'Cummulative Distribution'
    FROM SalaryEmployee1) as table1
WHERE first_name ="Irene";

```

| row_num | first_name | last_name | salary | Cummulative Distribution |
|---|---|---|---|---|
| 3 | Irene | Mikkilineni | 2700 | 0.2 |

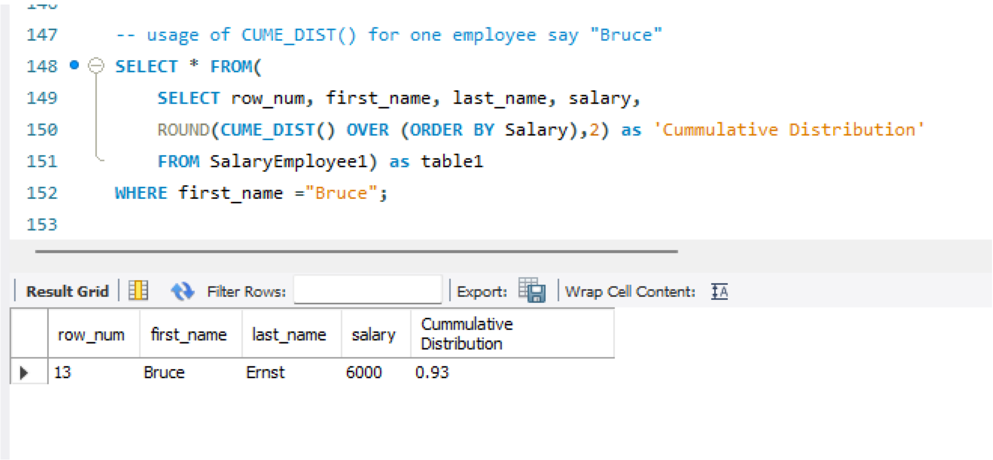
17. **Compute the CUME_DIST() for one employee "Bruce".**

CUME_DIST() – this window feature calculates the calculate a cumulative distribution of a value within a group of values. . It represents the number of rows with values less than or equal to that row's value divided by the total number of row(value ranges from 0 and 1)

**Formula : ROW_NUMBER / total_rows**

See screenshot for the SQL Command and the table output.

**NOTE**: Salary with the same rank MUST have the same CUME_DIST value



```
-- usage of CUME_DIST() for one employee say "Bruce"
SELECT * FROM(
    SELECT row_num, first_name, last_name, salary,
    ROUND(CUME_DIST() OVER (ORDER BY Salary),2) as 'Cummulative Distribution'
    FROM SalaryEmployee1) as table1
WHERE first_name ="Bruce";

```

| row_num | first_name | last_name | salary | Cummulative Distribution |
|---|---|---|---|---|
| 13 | Bruce | Ernst | 6000 | 0.93 |