1. SQL Analytic Functions VS Aggregate Functions

An Analytic function calculates an aggregate value over a group of rows and returns a single result for each row of the group. This is different from an aggregate function, which returns a single result for each group of rows.

Before getting into interview questions let us try to understand the differences between the Analytic Functions and Aggregate Functions.

Let us understand the differences with an example. Oracle HR Schema's Employee table is used in all the demonstrations in the article.

Employee table sample data:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100
110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	8200	(null)	108	100
111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100
112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100
113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCOUNT	6900	(null)	108	100

Average Salary of all employees

```
SELECT AVG(Salary) as AVG_SAL FROM EMPLOYEES

AVG_SAL
------------------
6461.83
```

Here the average is calculated for all the employees of all the departments present in the table.

Average Salary of all employees Department wise

```
SELECT Department_Id, AVG(Salary) as AVG_SAL FROM EMPLOYEES
GROUP BY Department_Id ORDER BY Department_Id
```

DEPARTMENT_ID	AVG_SAL
10	4400
20	9500
30	4150
40	6500
50	3475.56
60	5760
70	10000
80	8955.88
90	19333.33
100	8601.33
110	10154
(null)	7000

Here average salary of employees is calculated department wise. Here we can observe that though there may be any number of employee records per department, only one record per department is returned as a result.

Displaying average salary against each employee record

```
SELECT
   B.Employee_Id,
   B.First_Name,
   B.Department_Id,
   B.Salary,
   A.AVG_SAL
FROM
(SELECT Department_Id, AVG(Salary) as AVG_SAL FROM Employees GROUP BY Department_Id) A,
EMPLOYEES B
WHERE A.Department_Id = B.Department_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	AVG_SAL
100	Steven	90	24000	19333.33
101	Neena	90	17000	19333.33
102	Lex	90	17000	19333.33
103	Alexander	60	9000	5760
104	Bruce	60	6000	5760
105	David	60	4800	5760
106	Valli	60	4800	5760
107	Diana	60	4200	5760
108	Nancy	100	12008	8601.33
109	Daniel	100	9000	8601.33
110	John	100	8200	8601.33
111	Ismael	100	7700	8601.33
112	Jose Manuel	100	7800	8601.33
113	Luis	100	6900	8601.33

Here we are displaying all the employee records with average salary of employees of the department in a separate column AVG_SAL.

To achieve this we have performed a self-join on Employee table. We have taken the result of “Average salary per department in Employee table” and “entire Employee table” and joined them based on the Department_Id.

> *The same result can be achieved by a query using the **Analytic function** as below.*

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    AVG(Salary) OVER(PARTITION BY Department_Id) as AVG_SAL
FROM EMPLOYEES ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	AVG_SAL
100	Steven	90	24000	19333.33
101	Neena	90	17000	19333.33
102	Lex	90	17000	19333.33
103	Alexander	60	9000	5760
104	Bruce	60	6000	5760
105	David	60	4800	5760
106	Valli	60	4800	5760
107	Diana	60	4200	5760
108	Nancy	100	12008	8601.33
109	Daniel	100	9000	8601.33
110	John	100	8200	8601.33
111	Ismael	100	7700	8601.33
112	Jose Manuel	100	7800	8601.33
113	Luis	100	6900	8601.33

The results are same but the queries are different. Using the aggregate functions we can display only those columns which are in GROUP BY clause. But using an Analytic function you can display all the columns along with aggregated result.

I hope the difference between the two is clear now.

Related Article: **SQL Scenario based Interview Questions**

2. Analytic Functions Syntax

Before proceeding further let us understand the syntax of Analytic Functions. The syntax may vary for each function but below is the basic syntax.

Analytic_Function([arguments]) OVER ([partition_clause] [order_by_clause [windowing_clause]])

An **argument** is the column on which the aggregation is done.

Partition Clause defines the group of rows upon which the aggregation needs to be done. For example in the previous example Partition is specified on Department_Id.

If Partition Clause is not specified the aggregation is done on all the records of the table.

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary, AVG(Salary) OVER() as AVG_SAL
FROM EMPLOYEES
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	AVG_SAL
100	Steven	90	24000	6461.83
101	Neena	90	17000	6461.83
102	Lex	90	17000	6461.83
103	Alexander	60	9000	6461.83
104	Bruce	60	6000	6461.83
105	David	60	4800	6461.83
106	Valli	60	4800	6461.83
107	Diana	60	4200	6461.83
108	Nancy	100	12008	6461.83
109	Daniel	100	9000	6461.83
110	John	100	8200	6461.83
111	Ismael	100	7700	6461.83
112	Jose Manuel	100	7800	6461.83
113	Luis	100	6900	6461.83

Here the AVG_SAL is calculated for all the employees instead of department wise as partition clause is empty.

Order By Clause is used to order the rows with in the partition. If an analytic function behaves differently according to order of rows in a partition, the order by clause needs to be specified. We shall see more about this in below examples.

Windowing Clause just like partition clause gives a further degree of control over the window within which an analytic function can apply.

> *This is an extension of order by clause. So we can't use windowing clause without order by clause.*

3. Calculate total sum of salary department wise

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    SUM(Salary) OVER(PARTITION BY Department_Id) as SUM_SAL
FROM EMPLOYEES ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	SUM_SAL
100	Steven	90	24000	58000
101	Neena	90	17000	58000
102	Lex	90	17000	58000
103	Alexander	60	9000	28800
104	Bruce	60	6000	28800
105	David	60	4800	28800
106	Valli	60	4800	28800
107	Diana	60	4200	28800
108	Nancy	100	12008	51608
109	Daniel	100	9000	51608
110	John	100	8200	51608
111	Ismael	100	7700	51608
112	Jose Manuel	100	7800	51608
113	Luis	100	6900	51608

Here sum of salary per department is calculated and displayed against each employee record.

For Department_ID=90, the sum is calculated as SUM(24000,17000,17000)=58000

4. Calculate cumulative sum of salary department wise

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    SUM(Salary) OVER(PARTITION BY Department_Id ORDER BY Employee_Id) as CUML_SAL
FROM EMPLOYEES ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	CUML_SAL
100	Steven	90	24000	24000
101	Neena	90	17000	41000
102	Lex	90	17000	58000
103	Alexander	60	9000	9000
104	Bruce	60	6000	15000
105	David	60	4800	19800
106	Valli	60	4800	24600
107	Diana	60	4200	28800
108	Nancy	100	12008	12008
109	Daniel	100	9000	21008
110	John	100	8200	29208
111	Ismael	100	7700	36908
112	Jose Manuel	100	7800	44708
113	Luis	100	6900	51608

The order by clause addition changed the behavior of the sum function allowing it to calculate the cumulative sum.

For emp_id 100, the CUML_SAL is calculated as SUM(24000) =24000
For emp_id 101, the CUML_SAL is calculated as SUM (24000,17000) =41000
For emp_id 102, the CUML_SAL is calculated as SUM (24000,17000,17000) =58000

5. Calculate cumulative sum of the organization

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    SUM(Salary) OVER(ORDER BY Employee_Id) as CUML_SAL
FROM EMPLOYEES ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	CUML_SAL
100	Steven	90	24000	24000
101	Neena	90	17000	41000
102	Lex	90	17000	58000
103	Alexander	60	9000	67000
104	Bruce	60	6000	73000
105	David	60	4800	77800
106	Valli	60	4800	82600
107	Diana	60	4200	86800
108	Nancy	100	12008	98808
109	Daniel	100	9000	107808
110	John	100	8200	116008
111	Ismael	100	7700	123708
112	Jose Manuel	100	7800	131508
113	Luis	100	6900	138408

Since there is no partition clause is mentioned, the sum keeps increasing for each record and final record will give us the total sum of salary of all employees.

6. Calculate Cumulative average of the salary department wise

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    AVG(Salary) OVER(PARTITION BY Department_Id ORDER BY Employee_Id) as AVG_SAL
FROM EMPLOYEES ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	AVG_SAL
100	Steven	90	24000	24000
101	Neena	90	17000	20500
102	Lex	90	17000	19333.33
103	Alexander	60	9000	9000
104	Bruce	60	6000	7500
105	David	60	4800	6600
106	Valli	60	4800	6150
107	Diana	60	4200	5760
108	Nancy	100	12008	12008
109	Daniel	100	9000	10504
110	John	100	8200	9736
111	Ismael	100	7700	9227
112	Jose Manuel	100	7800	8941.6
113	Luis	100	6900	8601.33

For emp_id 100, the AVG_SAL is calculated as AVG(24000) =24000
For emp_id 101, the AVG_SAL is calculated as AVG(24000,17000) =20500
For emp_id 102, the AVG_SAL is calculated as AVG(24000,17000,17000) =19333.33

Hence the last record of the department gives the overall average of the salary for that department.

7. Calculate average of salary for current and previous record department wise

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    AVG(Salary) OVER(PARTITION BY Department_Id ORDER BY Employee_Id ROWS 1
PRECEDING) as AVG_SAL
FROM EMPLOYEES ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	AVG_SAL
100	Steven	90	24000	24000
101	Neena	90	17000	20500
102	Lex	90	17000	17000
103	Alexander	60	9000	9000
104	Bruce	60	6000	7500
105	David	60	4800	5400
106	Valli	60	4800	4800
107	Diana	60	4200	4500
108	Nancy	100	12008	12008
109	Daniel	100	9000	10504
110	John	100	8200	8600
111	Ismael	100	7700	7950
112	Jose Manuel	100	7800	7750
113	Luis	100	6900	7350

For emp_id 100, the AVG_SAL is calculated as AVG(24000) =24000
For emp_id 101, the AVG_SAL is calculated as AVG(24000,17000) =20500
For emp_id 102, the AVG_SAL is calculated as AVG(17000,17000) =17000

Here the AVG_SAL value is the average of the salary of current row and previous row in a department. This is achieved using the windowing function ROWS 1 PRECEDING after the Order by clause.

8. Find the oldest joinee department wise using LAG Analytic function

LAG Analytic function helps to fetch the row details of the previous record.

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Hire_date,
    LAG(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date) as PREV_HIREDATE
FROM EMPLOYEES
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	HIRE_DATE	PREV_HIREDATE
102	Lex	90	13-JAN-01	(null)
100	Steven	90	17-JUN-03	13-JAN-01
101	Neena	90	21-SEP-05	17-JUN-03

For employee 102, the PREV_HIREDATE is NULL as he is the oldest joinee in the department.
For employee 100, the PREV_HIREDATE is the hire date of the employee 102 and so on.

The below query fetches the oldest joinee details department wise

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Hire_date,
        LAG(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date) as
PREV_HIREDATE
    FROM EMPLOYEES)
WHERE PREV_HIREDATE IS NULL
```

9. Find the newest joinee department wise using LAG Analytic function

By adding DESC to the ORDER BY clause we can change the order in which LAG Analytic function calculates the previous value.

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Hire_date,
    LAG(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date desc) as
NEXT_HIREDATE
FROM EMPLOYEES
ORDER BY Department_Id, Hire_date
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	HIRE_DATE	NEXT_HIREDATE
102	Lex	90	13-JAN-01	17-JUN-03
100	Steven	90	17-JUN-03	21-SEP-05
101	Neena	90	21-SEP-05	(null)

Since we have mentioned DESC, the Lag starts from the highest Hire date and since there won't be any value before that, the lag value is calculated as NULL

The below query fetches the newst joinee details department wise

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Hire_date,
        LAG(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date desc) as
NEXT_HIREDATE
    FROM EMPLOYEES )
WHERE NEXT_HIREDATE IS NULL
```

10. Find the oldest joinee department wise using LEAD Analytic function

LEAD is exact opposite of the LAG Analytic function. LAG fetches the row details of previous record and LEAD fetches the row details of next record.

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Hire_date,
    LEAD(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date DESC) as
PREV_HIREDATE
FROM EMPLOYEES
ORDER BY Department_Id, Hire_date
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	HIRE_DATE	PREV_HIREDATE
102	Lex	90	13-JAN-01	(null)
100	Steven	90	17-JUN-03	13-JAN-01
101	Neena	90	21-SEP-05	17-JUN-03

Since LEAD fetches the next value we used the DESC in the ORDER BY clause so that the LEAD Analytic function starts working from latest Hire date and keep picking the hire date of the employee joined before him. So finally for the oldest joinee the value will be NULL.

The below query fetches the oldest joinee details department wise

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Hire_date,
        LEAD(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date DESC) as
PREV_HIREDATE
    FROM EMPLOYEES)
WHERE PREV_HIREDATE IS NULL
```

11. Find the newest joinee department wise using LEAD Analytic function

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Hire_date,
    LEAD(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date) as
```

```
    NEXT_HIREDATE
    FROM EMPLOYEES
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	HIRE_DATE	NEXT_HIREDATE
102	Lex	90	13-JAN-01	17-JUN-03
100	Steven	90	17-JUN-03	21-SEP-05
101	Neena	90	21-SEP-05	(null)

Since the LEAD Analytic function fetches the value of the next record, the newest joinee will have the value as NULL.

The below query fetches the newest joinee details department wise

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Hire_date,
        LEAD(Hire_date) OVER(PARTITION BY Department_Id ORDER BY Hire_date) as
NEXT_HIREDATE
    FROM EMPLOYEES)
WHERE NEXT_HIREDATE IS NULL
```

12. RANK and DENSE_RANK

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    RANK() OVER(PARTITION BY Department_Id ORDER BY Salary) as SAL_RANK
FROM EMPLOYEES
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	SAL_RANK
102	Lex	90	17000	1
101	Neena	90	17000	1
100	Steven	90	24000	3
113	Luis	100	6900	1
111	Ismael	100	7700	2
112	Jose Manuel	100	7800	3
110	John	100	8200	4
109	Daniel	100	9000	5
108	Nancy	100	12008	6

As the employees 101 and 102 have the same salary amount, the RANK function skipped the rank 2 in above example.

As we have given ORDER BY clause by Salary, the rank starts from the lowest salary to highest salary. The same can be observed for both departments 90 and 100. The ranking started from lowest to the highest salary value. This behavior can be changed by adding the DESC at the end of ORDER BY clause. Then the RANK starts ranking salary from highest to lowest.

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    DENSE_RANK() OVER(PARTITION BY Department_Id ORDER BY Salary) as SAL_RANK
FROM EMPLOYEES
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	SAL_RANK
102	Lex	90	17000	1
101	Neena	90	17000	1
100	Steven	90	24000	2
113	Luis	100	6900	1
111	Ismael	100	7700	2
112	Jose Manuel	100	7800	3
110	John	100	8200	4
109	Daniel	100	9000	5
108	Nancy	100	12008	6

> ***DENSE_RANK will not skip the rank value like in RANK function.***

In the above example employees 101 and 102 both have same salary value and are awarded rank 1. The next salary value is awarded rank 2 unlike RANK.

12.1 Find employee with MAX salary department wise

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Salary,
        RANK() OVER(PARTITION BY Department_Id ORDER BY Salary DESC) as SAL_RANK
    FROM EMPLOYEES
)
WHERE SAL_RANK =1
```

12.2 Find employee with MIN salary department wise

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Salary,
        RANK() OVER(PARTITION BY Department_Id ORDER BY Salary) as SAL_RANK
    FROM EMPLOYEES
)
WHERE SAL_RANK =1
```

If the question is to find the employee with 2nd max or min salary change the SAL_RANK accordingly in the query.

13. Find the difference between the salary of an employee and max salary of the employee in the department

```
SELECT Employee_Id,
    First_Name,
    Department_Id,
    Salary,
    MAX(Salary) OVER(PARTITION BY Department_Id ) as MAX_SAL,
    (MAX(Salary) OVER(PARTITION BY Department_Id )-Salary ) as SAL_DIFF
FROM EMPLOYEES
ORDER BY Employee_Id
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	MAX_SAL	SAL_DIFF
100	Steven	90	24000	24000	0
101	Neena	90	17000	24000	7000
102	Lex	90	17000	24000	7000
103	Alexander	60	9000	9000	0
104	Bruce	60	6000	9000	3000
105	David	60	4800	9000	4200
106	Valli	60	4800	9000	4200
107	Diana	60	4200	9000	4800
108	Nancy	100	12008	12008	0
109	Daniel	100	9000	12008	3008
110	John	100	8200	12008	3808
111	Ismael	100	7700	12008	4308
112	Jose Manuel	100	7800	12008	4208
113	Luis	100	6900	12008	5108

In the above query MAX_SAL represents the maximum salary of an employee in that department. SAL_DIFF gives the difference between the Max salary in the department and

the salary of the employee.

14. Find employee with MAX salary department wise without using RANK or DENSE_RANK

This can be achieved using the MAX function by extending the above query. The employee with MAX salary will have a SAL_DIFF as 0.

```
SELECT * FROM(
    SELECT Employee_Id,
        First_Name,
        Department_Id,
        Salary,
        MAX(Salary) OVER(PARTITION BY Department_Id ) as MAX_SAL,
        (MAX(Salary) OVER(PARTITION BY Department_Id )-Salary ) as SAL_DIFF
    FROM EMPLOYEES)
WHERE SAL_DIFF = 0
```

EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID	SALARY	MAX_SAL	SAL_DIFF
100	Steven	90	24000	24000	0
103	Alexander	60	9000	9000	0
108	Nancy	100	12008	12008	0
114	Den	30	11000	11000	0
121	Adam	50	8200	8200	0
145	John	80	14000	14000	0
178	Kimberely	(null)	7000	7000	0
200	Jennifer	10	4400	4400	0
201	Michael	20	13000	13000	0
203	Susan	40	6500	6500	0
204	Hermann	70	10000	10000	0
205	Shelley	110	12008	12008	0

Related Articles: