Implementation for SQL Ranking Functions

Step 1: Create Necessary Table

EmployeeSalary Table

```
CREATE TABLE EmployeeSalary (
    EmployeeID INT PRIMARY KEY,
    JobTitle VARCHAR(100),
    Salary DECIMAL(10, 2)
);
```

Step 2: Insert Data

```
INSERT INTO EmployeeSalary VALUES
(1, 'Software Engineer', 70000),
(2, 'Software Engineer', 80000),
(3, 'Software Engineer', 75000),
(4, 'Data Scientist', 85000),
(5, 'Data Scientist', 95000),
```

```
(6, 'Data Scientist', 85000),
(7, 'Project Manager', 100000),
(8, 'Project Manager', 95000),
(9, 'Project Manager', 105000);
```

Step 3: Perform Queries

---

1. `ROW_NUMBER()`

The `ROW_NUMBER()` function assigns a unique sequential number to each row in the result set.

Query:

```
SELECT *,
       ROW_NUMBER() OVER (ORDER BY Salary DESC) AS SalaryRank
FROM EmployeeSalary;
```

Output:

Employee ID	JobTitle	Salary	SalaryRank
9	Project Manager	105000.00	1
7	Project Manager	100000.00	2
5	Data Scientist	95000.00	3
8	Project Manager	95000.00	4
4	Data Scientist	85000.00	5
6	Data Scientist	85000.00	6
2	Software Engineer	80000.00	7
3	Software Engineer	75000.00	8
1	Software Engineer	70000.00	9

2. `RANK()`

The `RANK()` function provides the same rank for rows with duplicate values but skips ranks for duplicates.

Query Without `PARTITION BY`:

```
SELECT *,
       RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
FROM EmployeeSalary;
```

Output:

EmployeeID	JobTitle	Salary	SalaryRank
9	Project Manager	105000.00	1
7	Project Manager	100000.00	2
5	Data Scientist	95000.00	3

8	Project Manager	95000.00	3
4	Data Scientist	85000.00	5
6	Data Scientist	85000.00	5
2	Software Engineer	80000.00	7
3	Software Engineer	75000.00	8
1	Software Engineer	70000.00	9

Query With `PARTITION BY`:

```
SELECT *,
       RANK() OVER (PARTITION BY JobTitle ORDER BY
Salary DESC) AS SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank;
```

Output:

Employee ID	JobTitle	Salary	SalaryRank
5	Data Scientist	95000.00	1
4	Data Scientist	85000.00	2
6	Data Scientist	2	
7	Project Manager	105000.00	1
9	Project Manager	100000.00	2
8	Project Manager	95000.00	3
2	Software Engineer	80000.00	1
3	Software Engineer	75000.00	2
1	Software Engineer	70000.00	3

3. `DENSE_RANK()`

The `DENSE_RANK()` function assigns the same rank for duplicate rows but does not skip ranks for duplicates.

Query Without `PARTITION BY`:

```
SELECT *,
       DENSE_RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
FROM EmployeeSalary;
```

Output:

EmployeeID	JobTitle	Salary	SalaryRank
9	Project Manager	105000.00	1
7	Project Manager	100000.00	2
5	Data Scientist	95000.00	3

8	Project Manager	95000.00	3
4	Data Scientist	85000.00	4
6	Data Scientist	85000.00	4
2	Software Engineer	80000.00	5
3	Software Engineer	75000.00	6
1	Software Engineer	70000.00	7

Query With `PARTITION BY`:

```
SELECT *,
       DENSE_RANK() OVER (PARTITION BY JobTitle ORDER
BY Salary DESC) AS SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank;
```

Output:

EmployeeID	JobTitle	Salary	SalaryRank
5	Data Scientist	95000.00	1
4	Data Scientist	85000.00	2
6	Data Scientist	2	
7	Project Manager	105000.00	1
9	Project Manager	100000.00	2
8	Project Manager	95000.00	3
2	Software Engineer	80000.00	1
3	Software Engineer	75000.00	2
1	Software Engineer	70000.00	3

4. `NTILE()`

The `NTILE()` function divides rows into a specified number of groups.

Query Without `PARTITION BY`:

```
SELECT *,
       NTILE(3) OVER (ORDER BY Salary DESC) AS
SalaryRank
FROM EmployeeSalary;
```

Query With `PARTITION BY`:

```
SELECT *,
       NTILE(3) OVER (PARTITION BY JobTitle ORDER BY
Salary DESC) AS SalaryRank
FROM EmployeeSalary
ORDER BY JobTitle, SalaryRank;
```