The Learning House

427 S 4th Street #300

Louisville KY 40202

Ranking Functions

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Learn the SQL Functions that help us rank data within our results sets
- Explore the usage of the OVER() clause with ranking functions
- Learn the utility of RANK(), DENSE_RANK(), ROW_NUMBER(), and NTILE()

Why Ranking Functions?

Ranking functions were introduced in SQL Server 2005. They allow inserting numerical values into result sets such that each row receives a value.

The main difference between the functions is how they treat ties and how they handle gaps (vs sequential rows).

Creating Sample Data

```
CREATE TABLE Players (
    PlayerId int not null identity(1,1) primary key,
    Name varchar(20) not null,
    PointsPerGame decimal(5,2) not null default(0)
)

GO

INSERT INTO Players (Name, PointsPerGame)
VALUES ('Jane', 9.9),
('Joe', 9.7),
('Mary', 10.8),
('George', 10.8),
('Todd', 11.2),
('Heather', 12.3);
```

Ranking Functions and OVER()

Each of the ranking functions must be followed by the OVER() clause. The OVER() clause's purpose is to specify the ranking order with an ORDER BY Statement.

RANK() vs DENSE_RANK()

Try out the query to the right.

In the result set, both of the ranking functions rank the values in order, but notice what happens when there is a tie.

In RANK(), the two ties are both given rank 3 and the next value is given rank 5.

In DENSE_RANK() both ties are given a value of 3 and the next one continues counting at 4.

```
SELECT Name,
    RANK() OVER(ORDER BY PointsPerGame DESC) as [Rank],
    DENSE_RANK() OVER(ORDER BY PointsPerGame DESC) as [DenseRank]
FROM Players
```

	Name	Rank	DenseRank
1	Heather	1	1
2	Todd	2	2
3	Mary	3	3
4	George	3	3
5	Jane	5	4
6	Joe	6	5

Filtering Ranked Queries

In order to filter a ranked query, we have to wrap it with another select statement and alias it. This is a common technique when working with ranked queries, called *making a derived table.*

```
SELECT * FROM (                                          inner query
    SELECT Name,
        RANK() OVER(ORDER BY PointsPerGame DESC) as [Rank],
        DENSE_RANK() OVER(ORDER BY PointsPerGame DESC) as [DenseRank]
    FROM Players) AS RankedPlayers
WHERE [Rank] <= 3
```

derived table

```
                                                  outer query
SELECT * FROM (

                ) AS RankedPlayers
WHERE [Rank] <= 3
```

Lab Exercise (SWCCorp)

1. Use a ranking function to assign ranked values for each record in the employee table based on HireDate. The most recent hire date should be rank 1 and each distinct older date should add one without any gaps.
2. Join the Employee and PayRates table together and display the FirstName, LastName, YearlySalary, and the rank from highest to lowest of the salary. Limit this to the first two ranks.

ROW_NUMBER

ROW_NUMBER is a ranking function that simply counts the number of rows returned. This is often used for paging results on the client.

```
SELECT Name,
    RANK() OVER(ORDER BY PointsPerGame DESC) as [Rank],
    DENSE_RANK() OVER(ORDER BY PointsPerGame DESC) as [DenseRank],
    ROW_NUMBER() OVER(ORDER BY PointsPerGame DESC) as [RowNum]
FROM Players
```

	Name	Rank	DenseRank	RowNum
1	Heather	1	1	1
2	Todd	2	2	2
3	Mary	3	3	3
4	George	3	3	4
5	Jane	5	4	5
6	Joe	6	5	6

NTILE()

NTILE ranks query data into groups. You must provide the number of groups as a parameter.

Say we wanted to put our players into 3 groups, to report on who is in the top 33 percentile:

```
SELECT Name,
    NTILE(3) OVER(ORDER BY PointsPerGame DESC) as [PercentileGroup]
FROM Players
```

What About Odd Matches?

In the case where groups cannot be split evenly, NTILE() will put the extra record into the highest group.

It also ignores ties, so it’s possible for two records with the same value to be in different groups.

Thought Exercise

Based on what we have learned, can you write a query against the employee table to display every third hire? Output should be like this:

	LastName	FirstName	HireDate	HireNumber
1	Kennson	David	1996-03-16 00:00:00.000	3
2	Adams	Alex	2001-01-01 00:00:00.000	6
3	Brown	Barry	2002-08-12 00:00:00.000	9
4	Bender	Eric	2007-05-17 00:00:00.000	12

Hint: T-SQL does support modulus (%)