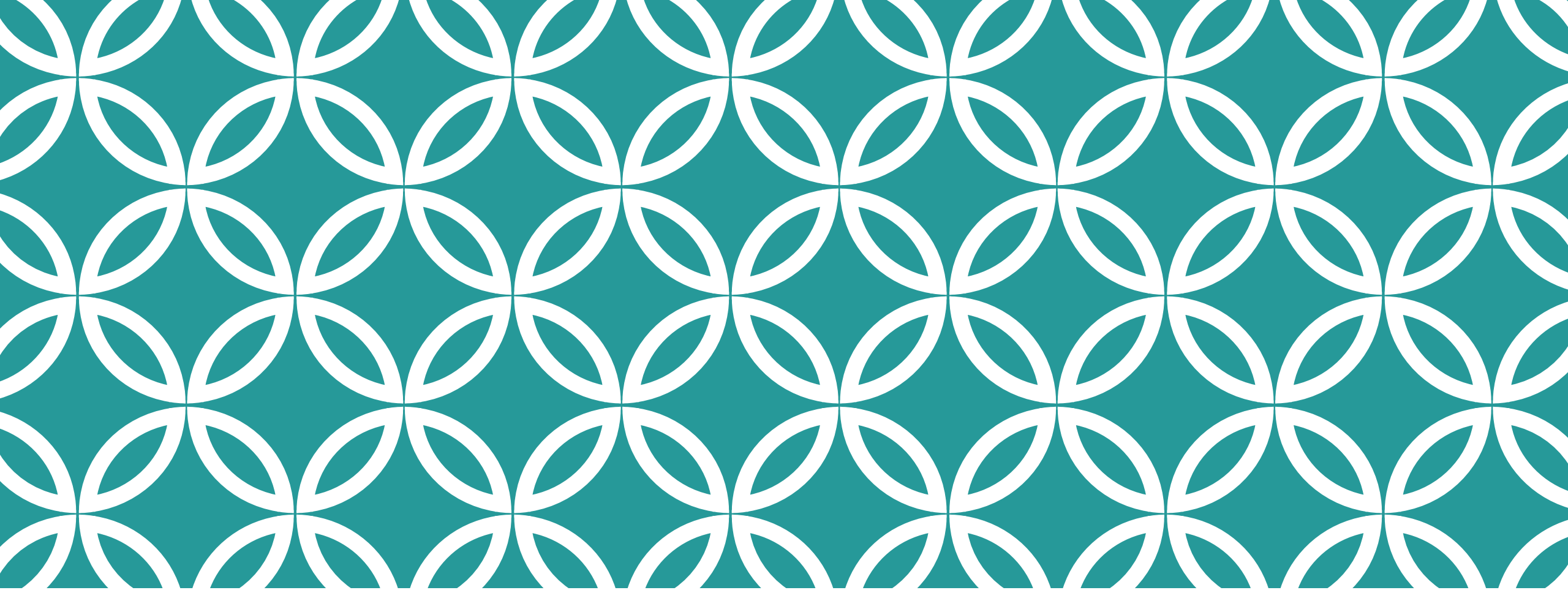


CHAPTER 18: TEMP TABLES

2023

CHAPTER 18: TEMP TABLES

Contents:

1. Introduction to temp tables
2. Local temp tables
3. global temp tables
4. Rank functions
5. Row functions
6. Positional functions
7. pivoting data (read)

TEMP TABLES

Temporary tables are used to store data temporarily and they can perform CRUD (Create, Read, Update, and Delete), join, and some other operations like the persistent database tables.

Temporary tables are **dropped when the session that creates the table has closed** or can also be explicitly dropped by users. At the same time, temporary tables can act like physical tables in many ways, which gives us more flexibility. Such as, we can create constraints, indexes, or statistics in these tables.

SQL Server provides two types of temporary tables according to their scope:

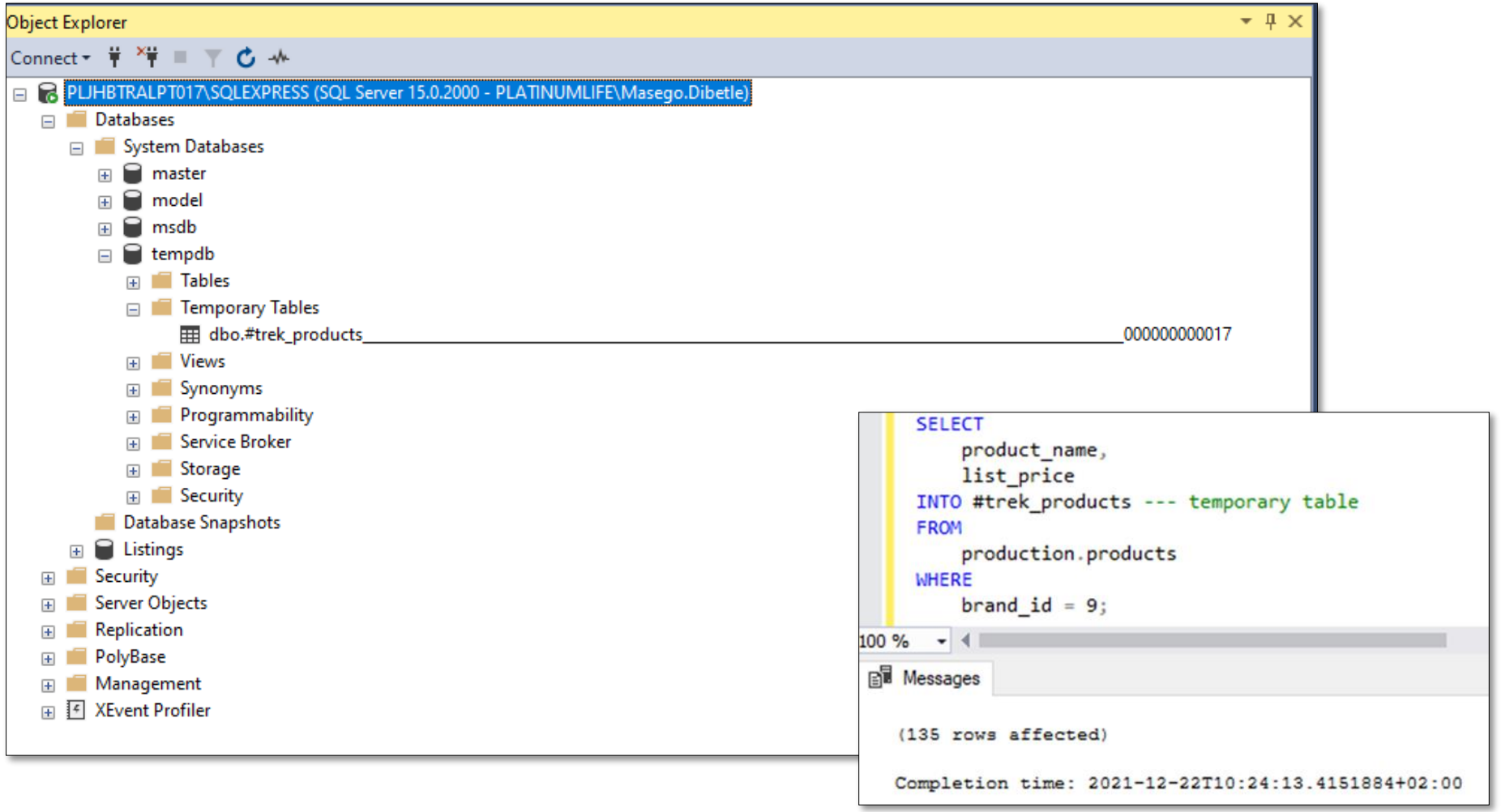
1. **Local Temporary Table** - tables are visible for the duration of the connection and when the session is closed the local temporary table are dropped automatically

2. **Global Temp Table** - These tables can be accessed by all other sessions, unlike local ones.

TEMP TABLES

Local Temp Tables	Global Temp Tables
Created using the "**CREATE TABLE**" command.	Created using the "**CREATE TABLE**" command.
Created using a single hashtag (#).	Created using a double hashtags (##).
The table will be visible for the duration of the connection only.	The table will be visible in all sessions.
Example: CREATE TABLE #TempPersonTable (PersonID int PRIMARY KEY IDENTITY(1,1), LastName varchar(255), FirstName varchar(255), City varchar(255))	Example: CREATE TABLE ##Customers (CustomerId INT IDENTITY(1,1) PRIMARY KEY, CustomerFullName VARCHAR(50), EMail VARCHAR(50), CustomerAddress VARCHAR(50), Country VARCHAR(50))

TEMP TABLES


Object Explorer
Connect
PLJHBTRALPT017\SQLEXPRESS (SQL Server 15.0.2000 - PLATINUMLIFE\Masego.Dibetle)
Databases
System Databases
master
model
msdb
tempdb
Tables
Temporary Tables
dbo.#trek_products_____000000000017
Views
Synonyms
Programmability
Service Broker
Storage
Security
Database Snapshots
Listings
Security
Server Objects
Replication
PolyBase
Management
XEvent Profiler
SELECT
product_name,
list_price
INTO #trek_products --- temporary table
FROM
production.products
WHERE
brand_id = 9;
100 %
Messages
(135 rows affected)
Completion time: 2021-12-22T10:24:13.4151884+02:00

TEMP TABLES

With this one you use the same syntax as creating a regular table. However, the name of the temporary table starts with a hash symbol (#) Like below image.

```
CREATE TABLE #haro_products
(
product_name VARCHAR(MAX),
list_price DEC(10,2)
);
```

After creating the temporary table, you can insert data into this table as a regular table

```
INSERT INTO #haro_products
SELECT
    product_name,
    list_price
FROM
    production.products
WHERE
    brand_id = 2;
```

```
.00 %
Messages

(10 rows affected)

Completion time: 2021-12-22T12:09:17.1226760+02:00
```

TEMP TABLES

To confirm you can select from the temporary table created. However if you open another connection and try the query below, you will get an error: Invalid object name '#haro_products'.

```
SELECT
    *
FROM
    #haro_products;
```

100 %

Results | Messages

	product_name	list_price
1	Haro Flightline One ST - 2017	379.99
2	Haro Flightline Two 26 Plus - 2017	549.99
3	Haro Shift R3 - 2017	1469.99
4	Haro SR 1.1 - 2017	539.99
5	Haro SR 1.2 - 2017	869.99
6	Haro SR 1.3 - 2017	1409.99
7	Haro Downtown 16 - 2017	329.99
8	Haro Shredder 20 - 2017	209.99
9	Haro Shredder 20 Girls - 2017	209.99
10	Haro Shredder Pro 20 - 2017	249.99

GLOBAL TEMP TABLES

Sometimes, you may want to create a temporary table that is accessible across connections. In this case, you can use global temporary tables.

Unlike a temporary table, the name of a global temporary table starts with a double hash symbol (**##**).

SQL Server drops a temporary table automatically when you close the connection that created it.

SQL Server drops a global temporary table once the connection that created it closed and the queries against this table from other connections completes

you can manually remove the temporary table by using the

"DROP TABLE ##table_name" statement

```sql
CREATE TABLE ##heller_products (
    product_name VARCHAR(MAX),
    list_price DEC(10,2)
);

INSERT INTO ##heller_products
SELECT
    product_name,
    list_price
FROM
    production.products
WHERE
    brand_id = 3;
select * from ##heller_products
```

100 %

Results | Messages

	product_name	list_price
1	Heller Shagamaw Frame - 2016	1320.99
2	Heller Bloodhound Trail - 2018	2599.00
3	Heller Shagamaw GX1 - 2018	2599.00

DIFFERENCE BETWEEN TEMP TABLES AND MATERIALIZED VIEWS

Temporary tables are:

- tables that are created for temporary use within a specific session or transaction.
- typically used to store intermediate results of complex queries or to stage data for processing.
- stored in the tempdb database, which is a system database that is used to store temporary objects.

Materialized views:

- database objects that contain precomputed results of a query.
- stored in the same way as regular tables and can be indexed and optimized for query performance.
- typically used for frequently executed queries that involve complex joins or aggregations.

Practice TEMP TABLES

1. Create a database called ‘Disney’.
2. Create a local temp table called characters, with the following columns and data: name, city and scene.

name	city	scene
Goofy	France	301
Minnie Mouse	Italy	304
Simba	Texas	400

3. Create a global temp table called CharNames from the characters table using on Names column.
4. Disconnect the session from your server and reconnect which tables were dropped, Why?
5. Delete the CharNames table from the DB.

RANK FUNCTIONS

The RANK () function is a window function that assigns a rank to each row in the result set partition.

Rows in partitions with the same value are given the same rank.

The rank of the first row in the partition is 1.

The RANK () function adds the number of rows in a tie to the rank of the tie and calculates the rank of the next row so the ranks may not be contiguous.

```sql
SELECT
    product_id,
    product_name,
    list_price,
    RANK () OVER (
        ORDER BY list_price DESC) price_rank
FROM production.products;
```

100 %

Results | Messages

	product_id	product_name	list_price	price_rank
1	155	Trek Domane SLR 9 Disc - 2018	11999.99	1
2	149	Trek Domane SLR 8 Disc - 2018	7499.99	2
3	156	Trek Domane SL Frameset - 2018	6499.99	3
4	157	Trek Domane SL Frameset Women's - 2018	6499.99	3
5	169	Trek Emonda SLR 8 - 2018	6499.99	3
6	51	Trek Silque SLR 8 Women's - 2017	6499.99	3
7	50	Trek Silque SLR 7 Women's - 2017	5999.99	7
8	56	Trek Domane SLR 6 Disc - 2017	5499.99	8
9	177	Trek Domane SLR 6 Disc - 2018	5499.99	8
10	154	Trek Domane SLR 6 Disc Women's - 2018	5499.99	8
11	148	Trek Domane SL 8 Disc - 2018	5499.99	8
12	47	Trek Remedy 9.8 - 2017	5299.99	12
13	43	Trek Fuel EX 9.8 27.5 Plus - 2017	5299.99	12
14	40	Trek Fuel EX 9.8 29 - 2017	4999.99	14
15	61	Trek Powerfly 8 FS Plus - 2017	4999.99	14
16	58	Trek Madone 9.2 - 2017	4999.99	14
17	140	Trek Remedy 9.8 27.5 - 2018	4999.99	14
18	146	Trek Domane SLR 6 - 2018	4999.99	14
19	153	Trek Domane SL 7 Women's - 2018	4999.99	14

Query executed successfully. | PLJHBTRALPT017\SQLEXPRESS (... | PLATINUMLIFE\Masego.Di... | master | 00:00:00 | 321 rows

MORE RANK EXAMPLES

Practice RANK FUNCTION

1. Use the RANK() function to write a query that assign a rank to each product by list price in each brand and returns products with rank less than or equal to three
2. First the PARTITION BY clause divides the products into partitions by brand Id.
3. Second the ORDER BY clause sorts products in each partition by list prices.
4. Third the outer query returns the products whose rank values are less than or equal to three.

ROW NUMBER FUNCTION

ROW_NUMBER function is similar to the Rank function except that it numbers all rows sequentially (for example 1, 2, 3, 4, 5). RANK provides the same numeric value for ties (for example 1, 2, 2, 4, 5).

SYNTAX:

```
ROW_NUMBER ( ) OVER ( [ PARTITION BY value_expression , ... [ n ] ]
order_by_clause )
```

ROW NUMBER FUNCTION

A table query before the row function

```
SELECT
  name, recovery_model_desc
FROM sys.databases
WHERE database_id < 5
ORDER BY name ASC;
```

name	recovery_model_desc
master	SIMPLE
model	FULL
msdb	SIMPLE
tempdb	SIMPLE

A table query applying the row function

```
SELECT
  ROW_NUMBER() OVER(ORDER BY name ASC) AS Row#,
  name, recovery_model_desc
FROM sys.databases
WHERE database_id < 5;
```

Row#	name	recovery_model_desc
1	master	SIMPLE
2	model	FULL
3	msdb	SIMPLE
4	tempdb	SIMPLE

POSITIONAL FUNCTIONS

These are window functions and are very useful in creating reports, because they can refer to data from rows above or below the current row. The LAG() and LEAD() are positional functions which we will cover.

The LAG() function

The LAG() function allows access to a value stored in a different row above the current row. The row above may be adjacent, or some number of rows above, as sorted by a specified column or set of columns.

The LEAD() function

LEAD() is similar to LAG(). Whereas LAG() accesses a value stored in a row above, LEAD() accesses a value stored in a row below.

THE LAG() FUNCTION

SYNTAX:

LAG(expression [,offset[,default_value]]) OVER(ORDER BY columns)

This function takes three arguments: the name of the column or an expression from which the value is obtained, the number of rows to skip (offset) above, and the default value to be returned if the stored value obtained from the row above is empty. Only the first argument is required. The third argument (default value) is allowed only if you specify the second argument, the offset.

THE LAG() FUNCTION

Lets consider the table below

id	seller_name	sale_value
3	Stef	7000
1	Alice	12000
2	Mili	25000

```
SELECT seller_name, sale_value,
  LAG(sale_value) OVER(ORDER BY sale_value) as previous_sale_value
FROM sale;
```

The output/Result:

seller_name	sale_value	previous_sale_value
Stef	7000	NULL
Alice	12000	7000
Mili	25000	12000

THE LEAD() FUNCTION

SYNTAX:
LEAD(expression [,offset[,default_value]]) OVER(ORDER BY columns)

LEAD() is like LAG(). Whereas LAG() accesses a value stored in a row above, LEAD() accesses a value stored in a row below. the LEAD() function takes three arguments: the name of a column or an expression, the offset to be skipped below, and the default value to be returned if the stored value obtained from the row below is empty. Only the first argument is required. The third argument, the default value, can be specified only if you specify the second argument, the offset.

THE LEAD() FUNCTION

Lets consider the table below

id	seller_name	sale_value
3	Stef	7000
1	Alice	12000
2	Mili	25000

The output/Result:

seller_name	sale_value	next_sale_value
Stef	7000	12000
Alice	12000	25000
Mili	25000	NULL

```
SELECT seller_name, sale_value,
  LEAD(sale_value) OVER(ORDER BY sale_value) as next_sale_value
FROM sale;
```

COMPARING DATA USING POSITIONAL FUNCTIONS

Lets consider the sales table

year	total_sale
2015	23000
2016	25000
2017	34000
2018	32000
2019	33000

As you can see, this table contains the total sale amount by year. Using LAG() and LEAD(), we can compare annual sale amounts across years.

```
SELECT
   year,
   total_sale AS current_total_sale,
   LAG(total_sale) OVER(ORDER BY year) AS previous_total_sale,
   total_sale - LAG(total_sale) OVER(ORDER BY year) AS difference
FROM annual_sale;
```


year	current_total_sale	previous_total_sale	difference
2015	23000	NULL	NULL
2016	25000	23000	2000
2017	34000	25000	9000
2018	32000	34000	-2000
2019	33000	32000	1000

Practice POSITIONAL FUNCTIONS

	A	B	C	D
1	Dept	TSR	Date	Salary
2	A	Zee	2013/06/01	R 55 000,00
3	B	Lindi	2014/06/01	R 60 000,00
4	C	Nomsa	2015/06/01	R 65 000,00
5	D	David	2016/06/01	R 70 000,00

1. Create the following table and add the data as above
2. Each year the TSR is moved to the next department Use the suitable functions on the date column to estimate a calculated departure date of the TSR and their new salaries, ensure your query returns the name, original date of the employment and your new columns, give them suitable names.
3. Use the suitable function on the department field to know which department the TSR is from, ensure your query returns the name of the TSR, their current department and a new department column, give it a suitable name.
4. Compose all the data from the above queries into one output: name of TSR, their current department, date of hire and your new two columns from above.

PIVOTING DATA

Data pivoting enables you to rearrange the columns and rows in a report so you can view data from different perspectives.

With data pivoting, you can do the following:

1. Move an object (a business attribute or a metric calculation) and its related data from a row to a column.
2. Move an object (a business attribute or a metric calculation) and its related data from a column to a row.
3. Change the order of objects in the rows.
4. Change the order of objects in the columns.

PIVOTING DATA

For example, in the image below, the Inventory Received from Suppliers by Quarter report shows a set of data spread across the screen in a large grid display. (The image below shows only a small section of the full report.) It is not always easy to compare numbers in reports of this size.

Subcategory	Supplier	Quarter Metrics	2005 Q1 Units Received	2005 Q2 Units Received	2005 Q3 Units Received	2005 Q4 Units Received	2006 Q1 Units Received	2006 Q2 Units Received
Art & Architecture	Bantam Books			20	40	30	10	20
	John Wiley & Sons			30	30	50	30	40
	Scribner				20		10	10
	Simon & Schuster				10	10	10	10
	Warner Books					15	10	
	Total			**50**	**100**	**105**	**70**	**80**
Business	Bantam Books			15	15	15	15	15
	John Wiley & Sons			45		45		45
	Prentice Hall			15		15		15
	Simon & Schuster			45	30	35	45	45
	Warner Books			45	45	10	45	45
	Total			**165**	**90**	**120**	**105**	**165**
	Bantam Books			30	30	15	30	30
	Perigee			45	45	45	75	30

PIVOTING DATA

If you pivot the objects on the report, so that the objects that were in the columns are now in the rows, and the objects that were in the rows are now in the columns, much of the data is easier to read and compare, as shown in the image below.

For example, in this pivoted report it is simpler to analyze total units received each quarter within a subcategory of books, because the totals are listed in a single column, making them easy to compare. Any anomalies in the numbers quickly become apparent. To perform the same comparison analysis with the first report above, you must visually skip over groups of data and try to focus only on totals.

Quarter	Subcategory / Supplier / Metrics	Art & Architecture						Business			
		Bantam Books Units Received	John Wiley & Sons Units Received	Scribner Units Received	Simon & Schuster Units Received	Warner Books Units Received	Total Units Received	Bantam Books Units Received	John Wiley & Sons Units Received	Prentice Hall Units Received	Simon & Schuster Units Received
2005 Q1											
2005 Q2		20	30				50	15	45	15	45
2005 Q3		40	30	20	10		100	15			30
2005 Q4		30	50		10	15	105	15	45	15	35
2006 Q1		10	30	10	10	10	70	15			45
2006 Q2		20	40	10	10		80	15	45	15	45
2006 Q3		10	10	20		10	50	15	15	15	75
2006 Q4		30	15		10		55	10	30		30
Total		**160**	**205**	**60**	**50**	**35**	**510**	**100**	**180**	**60**	**305**

PIVOTING DATA

This is the setup for a sample for pivoting data

```
SELECT <non-pivoted column>,
    [first pivoted column] AS <column name>,
    [second pivoted column] AS <column name>,
    ...
    [last pivoted column] AS <column name>
FROM
    (<SELECT query that produces the data>)
    AS <alias for the source query>
PIVOT
(
    <aggregation function>(<column being aggregated>)
FOR
[<column that contains the values that will become column headers>]
    IN ( [first pivoted column], [second pivoted column],
    ... [last pivoted column])
) AS <alias for the pivot table>
<optional ORDER BY clause>;
```

PIVOTING DATA

PIVOT relational operators convert data from row level to column level.

PIVOT rotates an expression in a table value by converting a unique value from one column of the expression to multiple columns of output.

You can use the PIVOT operator to perform aggregate operations wherever you need them.

```sql
-- Pivot table with one row and five columns
SELECT 'AverageCost' AS Cost_Sorted_By_Production_Days,
[0], [1], [2], [3], [4]
FROM
(SELECT DaysToManufacture, StandardCost
    FROM Production.Product) AS SourceTable
PIVOT
(
AVG(StandardCost)
FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
) AS PivotTable;
```

```
Cost_Sorted_By_Production_Days 0           1           2           3           4
------------------------------ ----------- ----------- ----------- ----------- -----------
AverageCost                    5.0885      223.88      359.1082    NULL        949.4105
```

PIVOT CONTINUED..

```sql
SELECT * FROM
(
    SELECT
        category_name,
        product_id
    FROM
        production.products p
        INNER JOIN production.categories c
            ON c.category_id = p.category_id
) t
PIVOT(
    COUNT(product_id)
    FOR category_name IN (
        [Children Bicycles],
        [Comfort Bicycles],
        [Cruisers Bicycles],
        [Cyclocross Bicycles],
        [Electric Bikes],
        [Mountain Bikes],
        [Road Bikes])
) AS pivot_table;
```

100 %

Results | Messages

	Children Bicycles	Comfort Bicycles	Cruisers Bicycles	Cyclocross Bicycles	Electric Bikes	Mountain Bikes	Road Bikes
1	59	30	78	10	24	60	60

PIVOTING DATA - COMPLEX

```
SELECT VendorID, [250] AS Emp1, [251] AS Emp2, [256] AS Emp3, [257] AS Emp4, [260] AS Emp
FROM
(SELECT PurchaseOrderID, EmployeeID, VendorID
FROM Purchasing.PurchaseOrderHeader) p
PIVOT
(
COUNT (PurchaseOrderID)
FOR EmployeeID IN
( [250], [251], [256], [257], [260] )
) AS pvt
ORDER BY pvt.VendorID;
```

VendorID	Emp1	Emp2	Emp3	Emp4	Emp5
1492	2	5	4	4	4
1494	2	5	4	5	4
1496	2	4	4	5	5
1498	2	5	4	4	4
1500	3	4	4	5	4

PIVOTING DATA - COMPLEX CONTINUED..

```sql
SELECT * FROM
(
    SELECT
        category_name,
        product_id,
        model_year
    FROM
        production.products p
        INNER JOIN production.categories c
            ON c.category_id = p.category_id
) t
PIVOT(
    COUNT(product_id)
    FOR category_name IN (
        [Children Bicycles],
        [Comfort Bicycles],
        [Cruisers Bicycles],
        [Cyclocross Bicycles],
        [Electric Bikes],
        [Mountain Bikes],
        [Road Bikes])
) AS pivot_table;
```

100 %

Results | Messages

	model_year	Children Bicycles	Comfort Bicycles	Cruisers Bicycles	Cyclocross Bicycles	Electric Bikes	Mountain Bikes	Road Bikes
1	2016	3	3	9	2	1	8	0
2	2017	19	10	19	2	2	21	12
3	2018	37	17	50	6	21	31	42
4	2019	0	0	0	0	0	0	6