MODULE 11
DESIGNING AND IMPLEMENTING USER-DEFINED FUNCTIONS

DATABASE DESIGN & BUSINESS APPLICATION DEVELOPMENT

Module Overview

- Overview of Functions
- Designing and Implementing Scalar Functions
- Designing and Implementing Table-valued Functions
- Considerations for Implementing Functions
- Alternatives to Functions

Lesson 1: Overview of Functions

Types of Functions

System Functions

Types of Functions

- Scalar Functions
- Table-valued Functions
  - Inline vs Multi-Statement Functions
- System Functions

Functions cannot modify the data in the database

System Functions

SQL Server includes a large number of built-in functions

Most system functions are scalar functions

Configuration	Cryptographic	Cursor
Data Type	Date and Time	Mathematical
Metadata	ODBC	Replication
Security	String	System
Statistical	Text and Image	Trigger

Other types of functions are also provided

Rowset	Aggregate	Ranking

Lesson 2: Designing and Implementing Scalar Functions

- What is a Scalar Function?
- Creating Scalar Functions
- Deterministic and Non-deterministic Functions
- Demonstration 2A: Working with Scalar Functions

What Is a Scalar Function?

Scalar Functions:

Return a single data value

Can return any data type except rowversion, cursor and table when implemented in T-SQL

Can return any data type except for rowversion, cursor, table, text, ntext and image when implemented in managed code

Creating Scalar Functions

- Scalar User-defined Functions (UDFs) return a single data-type
- Scalar UDFs usually include parameters
- CREATE FUNCTION must be the only statement in a batch

```
CREATE FUNCTION dbo.RectangleArea
(@X1 float, @Y1 float, @X2 float, @Y2 float)
RETURNS float
AS BEGIN
  RETURN ABS(@X1 - @X2) * ABS(@Y1 - @Y2);
END;
```

Deterministic and Non-deterministic Functions

Type	Description
Deterministic Functions	Always return the same result given a specific input for a given database state
Non-deterministic functions	May return different results given a specific input
Built-in functions	Are deterministic or non-deterministic depending on how they are applied

```sql
CREATE FUNCTION dbo.TodayAsString(@Format int = 112)
RETURNS VARCHAR(20)
AS BEGIN
  RETURN CONVERT(VARCHAR(20),
                 CAST(SYSDATETIME() AS date) ,
                 @Format);
END;
GO
SELECT OBJECTPROPERTY(OBJECT_ID('dbo.TodayAsString'),
                              'IsDeterministic');
```

Demonstration 2A: Working with Scalar Functions

In this demonstration, you will see:

- How to create scalar user-defined functions
- How to query scalar user-defined functions
- How to determine if a scalar user-defined function is deterministic
- How to drop scalar user-defined functions

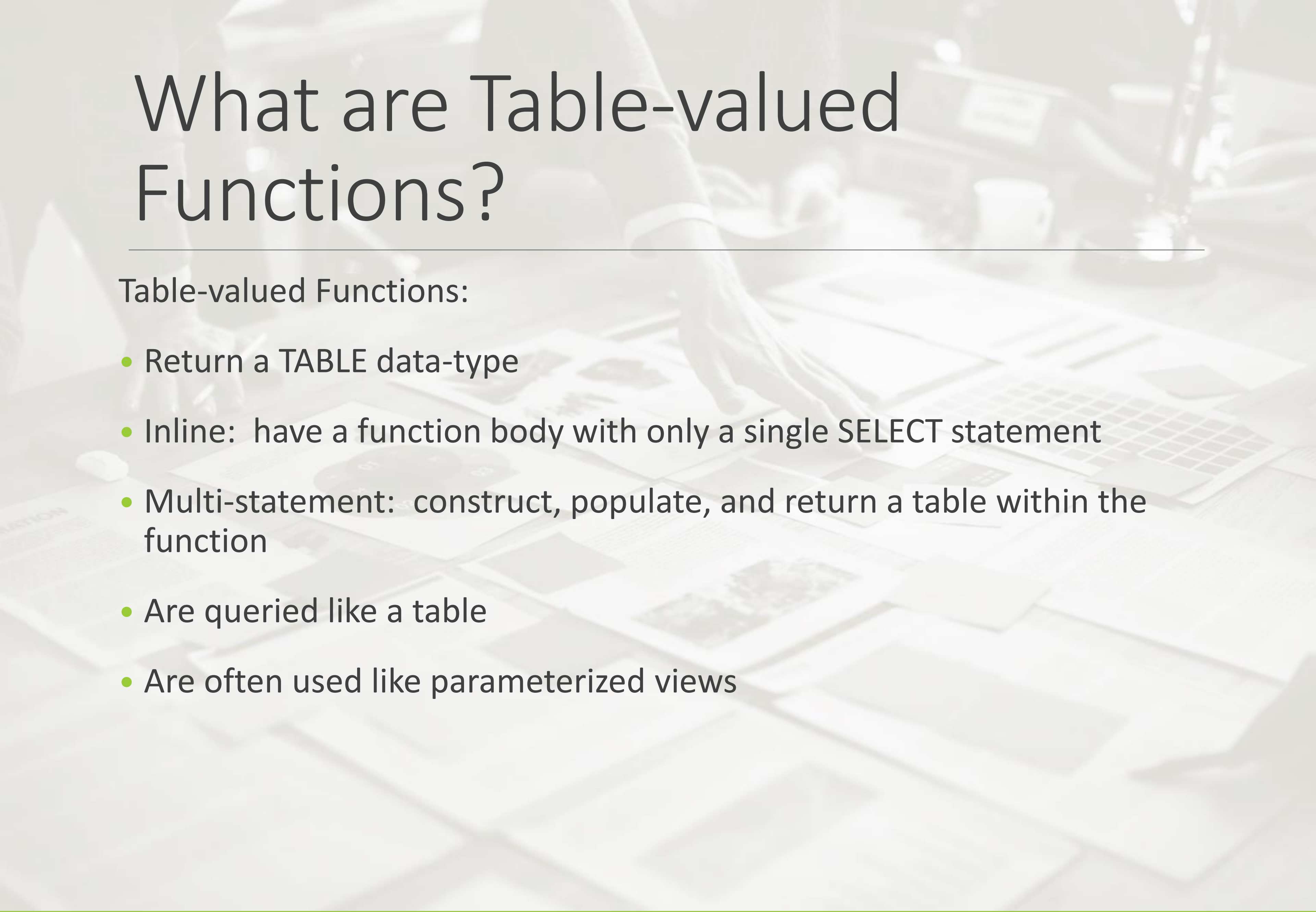
Lesson 3: Designing and Implementing Table-valued Functions

What are Table-valued Functions?

Inline Table-valued Functions

Multi-statement Table-valued Functions

Demonstration 3A: Implementing Table-valued Functions



What are Table-valued Functions?

Table-valued Functions:

- Return a TABLE data-type
- Inline: have a function body with only a single SELECT statement
- Multi-statement: construct, populate, and return a table within the function
- Are queried like a table
- Are often used like parameterized views

Inline Table-valued Functions

- Are comprised of a single result set and have no function body
- Have a returned table definition taken from a SELECT statement
- Can be seen as a parameterized view

```
CREATE FUNCTION Sales.GetLastOrdersForCustomer
(@CustomerID int, @NumberOfOrders int)
RETURNS TABLE
AS
RETURN (SELECT TOP(@NumberOfOrders)
               soh.SalesOrderID,
               soh.OrderDate,
               soh.PurchaseOrderNumber
        FROM Sales.SalesOrderHeader AS soh
        WHERE soh.CustomerID = @CustomerID
        ORDER BY soh.OrderDate DESC, soh.SalesOrderID DESC
               );
GO
SELECT * FROM Sales.GetLastOrdersForCustomer(17288,2);
```

Multi-statement Table-valued Functions

- Function body enclosed by BEGIN and END
- Definition of returned table must be supplied
- Table variable populated within function body and then returned

```
CREATE FUNCTION dbo.GetDateRange
(@StartDate date, @NumberOfDays int)
RETURNS @DateList TABLE (Position int, DateValue date)
AS BEGIN
    DECLARE @Counter int = 0;
    WHILE (@Counter < @NumberOfDays) BEGIN
        INSERT INTO @DateList
            VALUES(@Counter + 1,
                   DATEADD(day,@Counter,@StartDate));
        SET @Counter += 1;
    END;
    RETURN;
END;
GO
SELECT * FROM dbo.GetDateRange('2009-12-31',14);
```

Demonstration 3A: Implementing Table-valued Functions

In this demonstration, you will see:

How to create a table-valued function

How to query a table-valued function

How to drop a table-valued function

Lesson 4: Considerations for Implementing Functions

Performance Impact of Scalar Functions

Performance Impact of Table-valued Functions

Controlling Execution Context

Using the EXECUTE AS Clause

Guidelines for Creating Functions

Demonstration 4A: Controlling Execution Context

Performance Impact of Scalar Functions

Code for scalar functions is not incorporated into the surrounding query
- Different to views in this respect

Scalar functions used in SELECT lists or WHERE clause predicates can suffer significant performance-related issues

```
SELECT soh.CustomerID,
       soh.SalesOrderID
FROM Sales.SalesOrderHeader AS soh
WHERE soh.SalesOrderID =
         dbo.GetLatestBulkOrder(soh.CustomerID)
ORDER BY soh.CustomerID, soh.SalesOrderID;
```

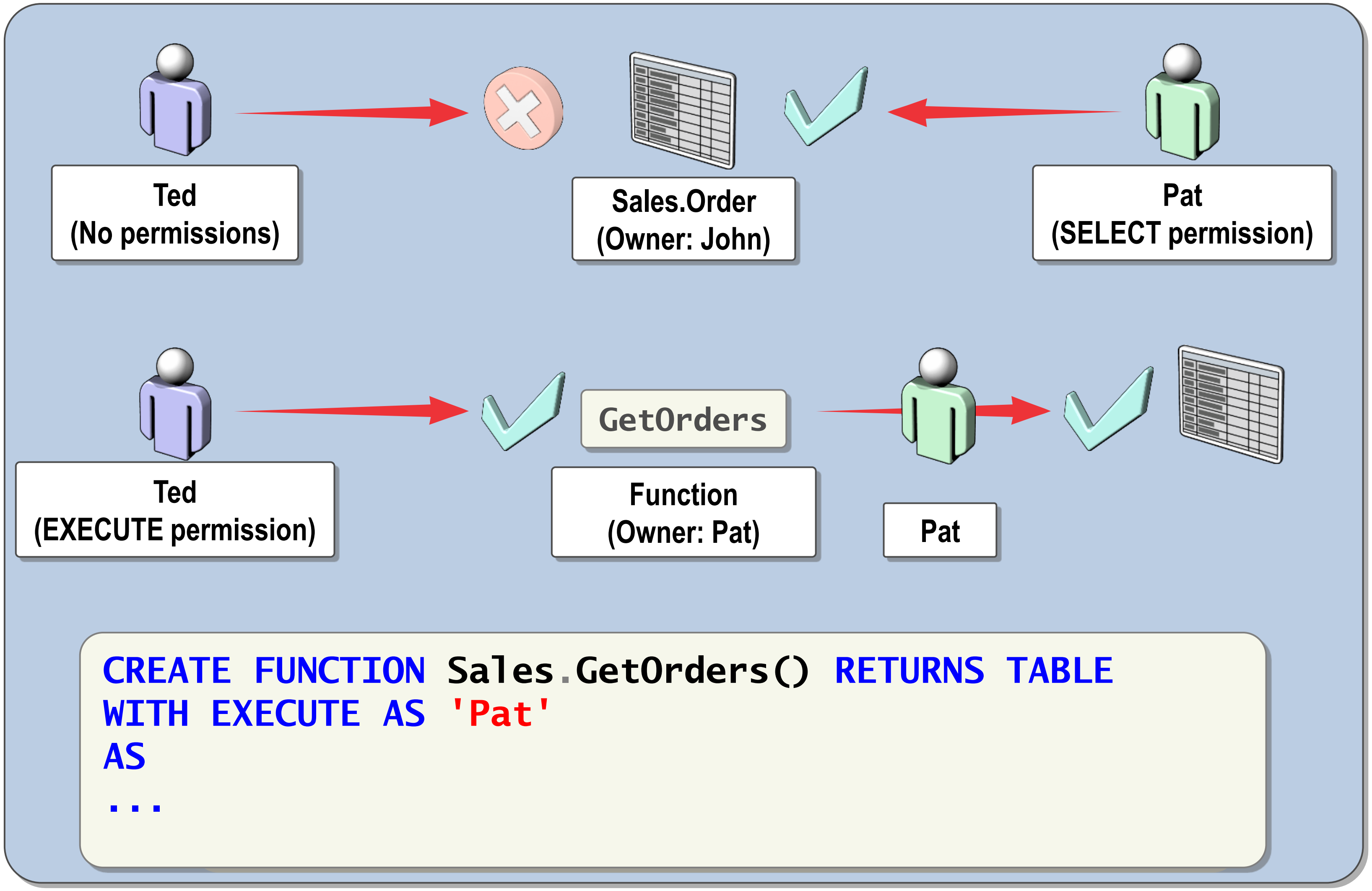
Performance Impact of Table-valued Functions

Code for inline TVFs is incorporated into the surrounding query

Code for table-valued functions is not incorporated into the surrounding query

- Performance can be poor except where executed only one time in a query
- Very common cause of performance problems
- CROSS APPLY can cause table-valued functions to be repeatedly executed

```
SELECT c.CustomerID,
       c.AccountNumber,
       glofc.SalesOrderID,
       glofc.OrderDate
FROM Sales.Customer AS c
CROSS APPLY
Sales.GetLastOrdersForCustomer(c.CustomerID,3) AS glofc
ORDER BY c.CustomerID,glofc.SalesOrderID;
```



```
CREATE FUNCTION Sales.GetOrders() RETURNS TABLE
WITH EXECUTE AS 'Pat'
AS
...
```

The EXECUTE AS Clause

The Execute AS Clause:

- Enables Impersonation
- Provides access to modules via impersonation
- Can be used to impersonate server-level principals or logins via the EXECUTE AS LOGIN statement
- Can be used to impersonate database level principals or users via the EXECUTE AS USER statement

```
CREATE FUNCTION Sales.GetOrders RETURNS TABLE
WITH EXECUTE AS {CALLER | SELF | OWNER | ‘user_name’ }
AS
...
```

Guidelines for Creating Functions

- Determine function type
- Create one function for one task
- Qualify object names inside function
- Consider the performance impacts of how you intend to use the function
  - Particular issues exist with the inability to index function results
- Functions cannot contain structured exception handling

Demonstration 4A: Controlling Execution Context

In this demonstration, you will see how to alter the execution context of a function

Lesson 5: Alternatives to Functions

- Comparing Table-valued Functions and Stored Procedures
- Comparing Table-valued Functions and Views

Both can often achieve similar outcomes

Some source applications can only call one or the other

Functions

- are easier to consume the output of in code
- can return table output in a variable
- cannot have data-related side effects
- often cause significant performance issues when multi-statement

Stored procedures

- can alter the data
- can execute dynamic SQL statements
- can include detailed exception handling
- can return multiple result sets

Comparing Table-valued Functions and Stored Procedures

Comparing Table-valued Functions and Views

Can often achieve similar outcomes

Views

- can be consumed by almost all applications
- very similar to tables
- can be updatable
- can have INSTEAD OF triggers associated with them

Table-valued Functions

- are like parameterized views
- can often lead to significant performance problems
- can be updatable when inline

Avoid multi-statement TVFs if there is any option to inline the same logic

Module References

- Create Function (Transact-SQL) - https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?f1url=%3FappId%3DDev15IDEF1%26l%3DEN-US%26k%3Dk(create_function_TSQL);k(sql13.swb.tsqlresults.f1);k(sql13.swb.tsqlquery.f1);k(MiscellaneousFilesProject);k(DevLang-TSQL)%26rd%3Dtrue&view=sql-server-ver15
- Create Function Oracle Database - https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_5009.htm
- Create Function MySQL Database - https://dev.mysql.com/doc/refman/8.0/en/create-function.html