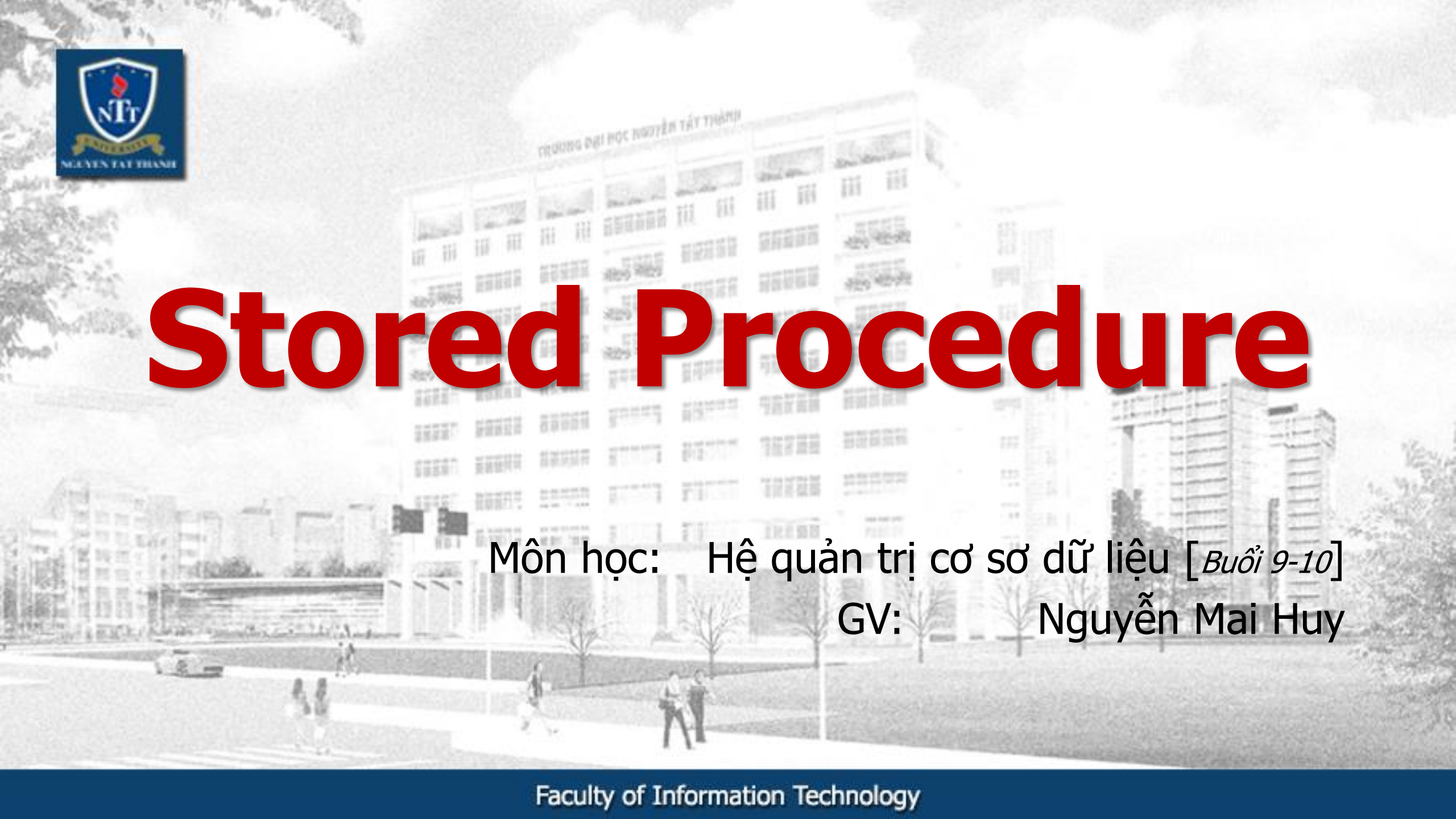
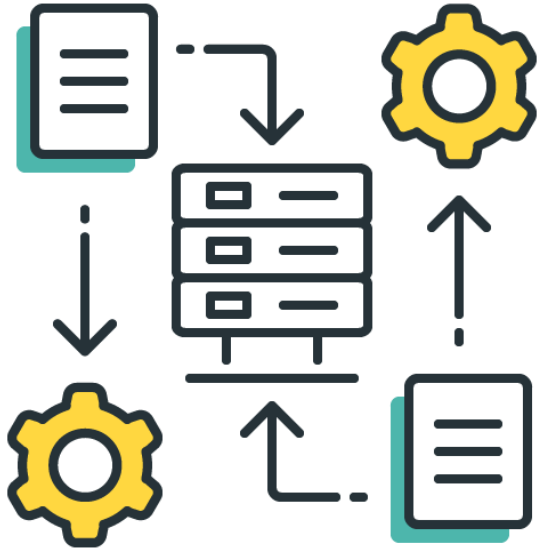


Stored Procedure

Môn học: Hệ quản trị cơ sơ dữ liệu [*Buổi 9-10*]

GV: Nguyễn Mai Huy

Stored Procedure



Stored Procedure :: SP

Stored Procedure là một ***Database object***, được tạo ra trong cơ sở dữ liệu để phục vụ cho mục tiêu thực thi một số các hành động nào đó theo thuật toán đã được định sẵn, bởi Database developer (*hoặc do RDBMS cung cấp sẵn*).

Nói cách khác, Stored Procedure trong SQL Server bao gồm một tập các lệnh T-SQL được lưu trữ trong database dưới dạng "*một đơn vị xử lý logic*" để phục vụ cho một nhiệm vụ nào đó trong quá trình hoạt động của cơ sở dữ liệu.

Stored Procedure :: SP

SP cho phép *nhận dữ liệu đầu vào* (**input**) cho quá trình xử lý, sau đó dựa trên những dữ liệu này để *thi hành các lệnh T-SQL đã được "lập trình"* theo một thuật toán đã định (**process**), và *kết quả của quá trình xử lý sẽ trả về* (**output**) cho nơi gọi SP thi hành.

Khi một SP được gọi thi hành lần đầu tiên, SQL Server sẽ "***biên dịch***" và **lưu trữ** mã thực thi của SP vào **trong Cache** của Database server. Đối với các lần thực thi tiếp theo của SP, SQL Server sẽ tiến hành gọi lại các mã đã được lưu trữ trước đó. Do vậy, SP có ***tốc độ thực thi rất nhanh với hiệu suất đáng tin cậy***.

Benefits of using Stored Procedure

- ***Improved Performance***: *The SQL Server stored procedure when executed for the first time creates a plan and stores it in the buffer pool so that the plan can be reused when it executes next time.*
- ***Reuse of code***: *Stored procedures can be executed by multiple users or multiple client applications without the need of writing the code again.*
- ***Reduced network traffic***: *When we use stored procedures instead of writing T-SQL queries at the application level, only the procedure name is passed over the network instead of the whole T-SQL code.*
- ***Stronger Security***: *Stored procedures reduce the threat by eliminating direct access to the tables. we can also encrypt the stored procedures while creating them so that source code inside the stored procedure is not visible.*

Types of Stored Procedure

- **System Procedure**: System procedures are included with SQL Server. They are physically stored in the internal, hidden **Resource** database and logically appear in the **sys** schema of every system- and user-defined database. In addition, the **msdb** database also contains system stored procedures in the **dbo** schema that are used for scheduling alerts and jobs. System procedures start with the prefix "**sp_**" (*take care when name your SP*)
- **User-defined**: A user-defined procedure can be created in a user-defined database or in all system databases except the **Resource** database. The procedure can be developed in either Transact-SQL or as a reference to a Microsoft .NET Framework common runtime language (CLR) method
- **Temporary**: Temporary procedures are a form of user-defined procedures. The temporary procedures are like a permanent procedure, except temporary procedures are stored in **tempdb**. There are two types of temporary procedures: local and global. They differ from each other in their names, their visibility, and their availability
- **Extended User-Defined**: Extended procedures enable creating external routines in a programming language such as C. These procedures are DLLs that an instance of SQL Server can dynamically load and run.

Create or Alter Stored Procedure

```
CREATE [ OR ALTER ] PROCEDURE <procedure_name>
    @param01    DataType    [ = defaultValue ] [OUTPUT],
    @param02    DataType    [ = defaultValue ] [OUTPUT],
    . . .
WITH ENCRYPTION
AS
BEGIN
    -- Your T-SQL statements go here
    -- .....
END
```

For **security** -> To Prevent **decompilation** of procedural code

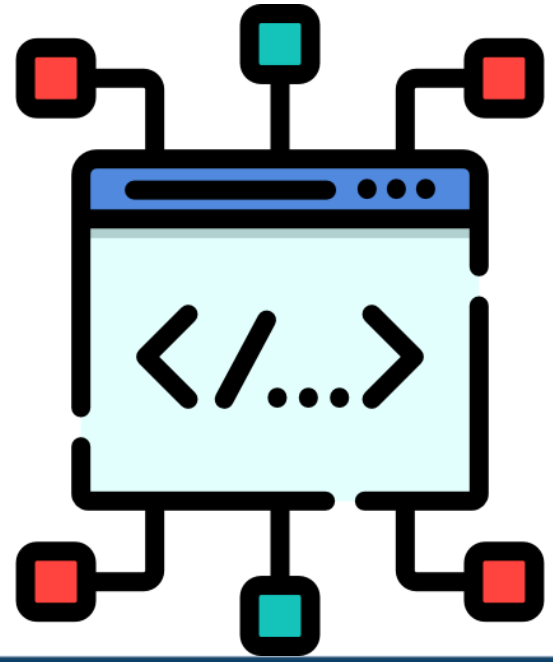
Drop, Rename or Execute SP

Syntax

- **sp_rename** <*Old_SP_Name*>, <*New_SP_Name*>;
- **EXECUTE** <*procedure_name*>;
- **DROP PROCEDURE** <*procedure_name*>;

Example

- **Exec** *bhol_proc01*;
- **Sp_rename** *bhol_proc01*, *bhol_TestSP*;
- **Drop proc** *bhol_TestSP*;



Programming with
Stored Procedure

Simple cases :: No parameters

Create – Execute

SQLQuery1.sql - BO...angOnline (sa (53))*

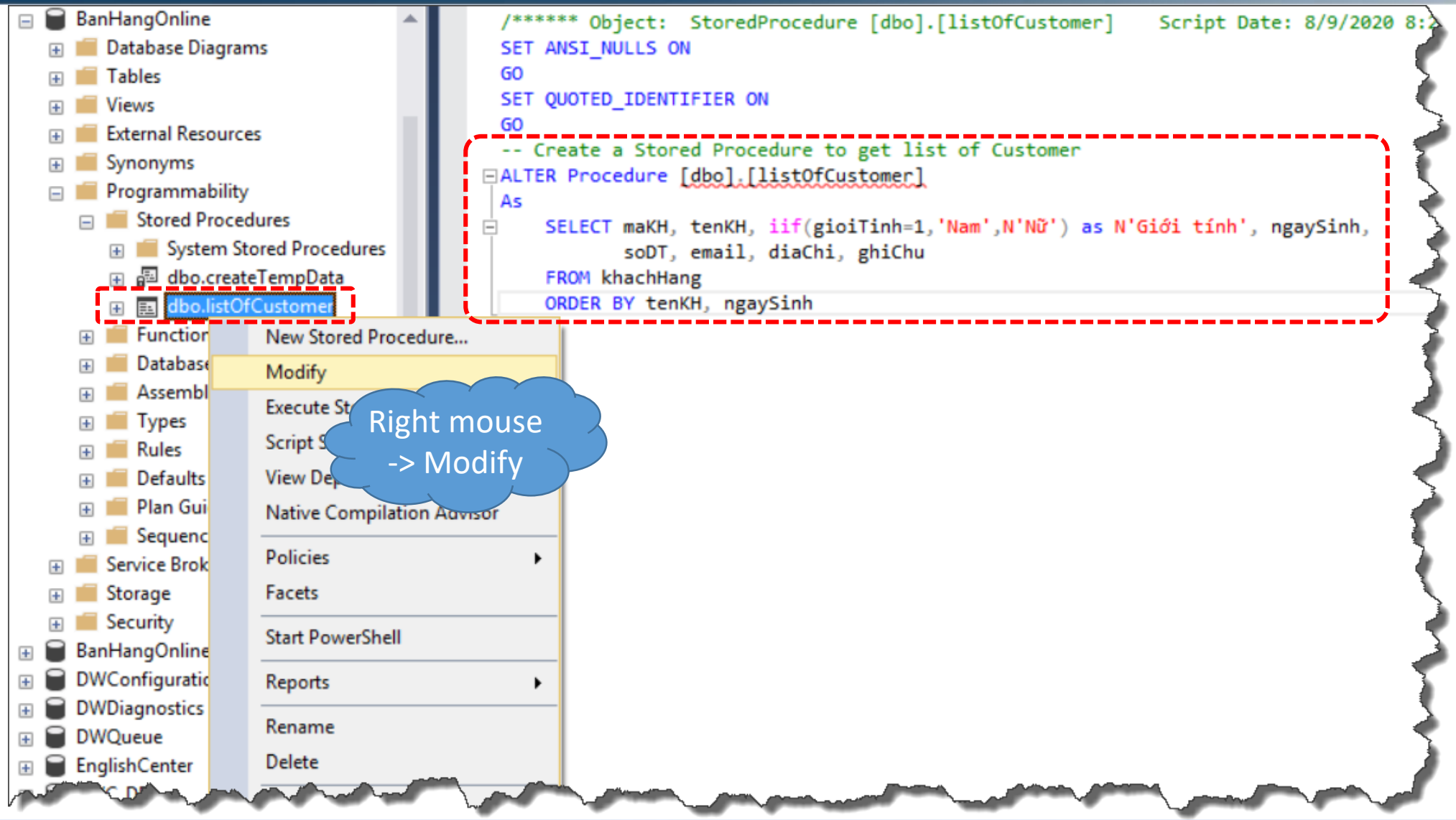
```
use BanHangOnline
go
-- Create a Stored Procedure to get list of Customer
Create Procedure listOfCustomer
As
    SELECT maKH, tenKH, iif(gioiTinh=1,'Nam',N'Nữ') as N'Giới tính', ngaySinh,
           soDT, email, diaChi, ghiChu
    FROM khachHang
    ORDER BY tenKH, ngaySinh

-- Run listIfCustomer by execute command
execute listOfCustomer
go
```

100 %

Results | Messages

	maKH	tenKH	Giới tính	ngaySinh	soDT	email	diaChi	ghiChu
1	KH002	Nguyễn Quang Hưng	Nam	2000-10-28 00:00:00.000	0705101028	nqhung@bodua.com	113 Tên Lửa, P.An Lạc, Q.Bình Tân, TP.Hồ Chí Minh	NULL
2	KH001	Trần Minh	Nam	1998-12-23 00:00:00.000	Chi?n	tmchien@hotmail.com	123 Trần Nguyên	NULL

Alter :: Stored Procdure



Statement – Block

- Statement end of with "**;**" symbol
- Block is limited by **BEGIN** … **END**

```
-- Cretae a Stored Procedure to describe how to Declare or set a value
Create Proc exampleForVariable
    @prdId varchar(10)
As
Begin
    Declare @Product nvarchar(500);
    Declare @Price bigint;
    select @product = tenSP, @Price = giaBan from sanPham where maSP = @prdId;
    print @product;
    print @price
End
go

exec exampleForVariable '01';
go
```

100 %

Messages

```
(1 row affected)
Túi xách nam - da bò
1650000
```

Declaring & Assigning a variable

- ✓ **Declare @Val_name DataType;**
- ✓ **Set @Val_name = *Value*;**

```
Declare @Dem int;
Set @Dem = 1001;
-- Print on the Messages area
print @Dem
```

100 %

Messages

1001

```
-- Cretae a Stored Procedure to describe how to Declare or set a value
Create Proc exampleForVariable
    @prdId varchar(10)
As
Begin
    Declare @Product nvarchar(500);
    Declare @Price bigint;
    Set @product = (select tenSP from sanPham where maSP = @prdId);
    Set @Price = (select giaBan from sanPham where masp = @prdId);
    print @product;
    print @price
End
go
```

Default Parameters

```
-- Create Stored procdure to list of products with price between Min And Max
Create Proc listOfProductInRange
    @Min int = 0,
    @Max int = 5000000
With Encryption
As
    Select maSP, tenSP, ngayDang, giaBan, giamGia*giaBan as N'Giảm'
    From sanPham
    Where giaBan between @Min and @Max
    Order by giaBan DESC;

go
-- List of product with price is no filter
exec listOfProductInRange
go
```

100 %

Results | Messages

	maSP	tenSP	ngayDang	giaBan	Giảm
1	04	Đồng hồ nam - cơ thụy sỹ	2020-07-28 17:22:10.863	4500000	157500000
2	05	Giày nam trẻ trung	2020-07-28 17:22:10.863	3500000	80500000
3	03	Túi xách nữ - da trăn	2020-07-28 17:22:10.863	2300000	46000000
4	01	Túi xách nam - da bò	2020-07-28 17:22:10.863	1650000	33000000
5	08	Bóp da nam - cá sấu	2020-07-28 17:22:10.880	1300000	35100000
6	02	Túi xách thời trang trẻ trung - vải bố	2020-07-28 17:22:10.863	450000	6750000
7	07	Dây nịt nam - trung niên	2020-07-28 17:22:10.863	300000	3600000

SP :: Output Parameters

```
-- Example for Stored Proceduce using OUTPUT parameter
Create Proc detailsByOrderID
    @donHang varchar(10),
    @num int output
With Encryption
As
Begin
    Select * From ctDonHang Where soDH = @donHang;
    Select @num = @@ROWCOUNT;
End

go

-- Exampe to use OUtPUT parameter
Declare @numberOfProduct int;
Set @numberOfProduct = 0;
Exec detailsByOrderID 'DH010', @numberOfProduct output;
-- Display @numberOfProduct on the Messages area
print N'Số sản phẩm đã mua: '+ cast(@numberOfProduct As varchar);
```

100 %

Results | Messages

```
(4 rows affected)
Số sản phẩm đã mua: 4
```

@@ROWCOUNT & SET NOCOUNT

- **@@ROWCOUNT**

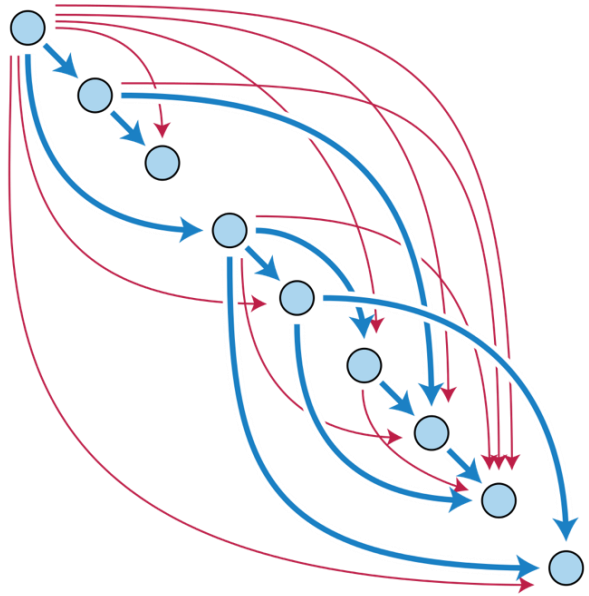
  Returns the number of rows affected by the last statement.

- **SET NOCOUNT** [ON | OFF]

  Stops the message that shows the count of the number of rows affected by a Transact-SQL statement or stored procedure from being returned as part of the result set.

  Remark: For stored procedures that contain several statements that do not return much actual data, or for procedures that contain Transact-SQL loops, setting SET NOCOUNT to ON can provide a *significant performance boost*, *because network traffic is greatly reduced*.

Control of flow



IF ... ELSE ...

syntaxsql

```
IF Boolean_expression
     { sql_statement | statement_block }
[ ELSE
     { sql_statement | statement_block } ]
```

SQL Copy

```
IF DATENAME(weekday, GETDATE()) IN (N'Saturday', N'Sunday')
       SELECT 'Weekend';
ELSE
       SELECT 'Weekday';
```

Imposes conditions on the execution of a Transact-SQL statement.

The Transact-SQL statement that follows an **IF** keyword and its condition is executed if the condition is satisfied: the Boolean expression returns **TRUE**.

The optional **ELSE** keyword introduces another Transact-SQL statement that is executed when the **IF** condition is not satisfied: the Boolean expression returns **FALSE**.

WHILE ...

syntaxsql

Copy

```
-- Syntax for SQL Server and Azure SQL Database

WHILE Boolean_expression
     { sql_statement | statement_block | BREAK | CONTINUE }
```

SQLQuery1.sql - BO...UA.master (sa (54))*

```
DECLARE @counter INT = 1;

WHILE @counter <= 5
BEGIN
     PRINT @counter;
     SET @counter = @counter + 1;
END
```

100 %

Messages

```
1
2
3
4
5
```

Sets a condition for ***the repeated execution*** of an SQL statement or statement block.

The statements are executed repeatedly as long as the specified condition is true.

The execution of statements in the **WHILE** loop can be controlled from inside the loop with the **BREAK** and **CONTINUE** keywords.

BREAK - CONTINUE

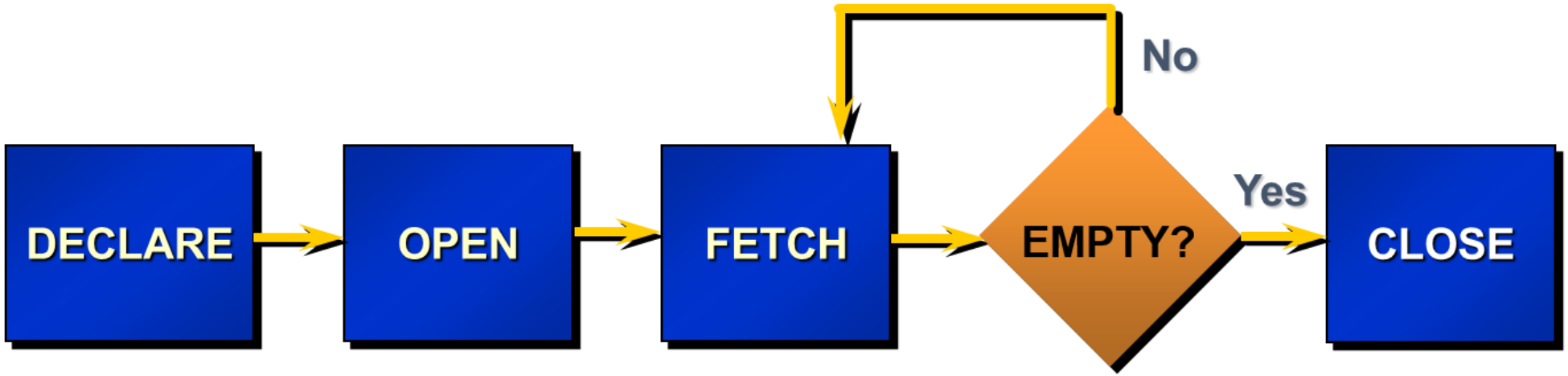
- To exit the current iteration of a loop, you use the **BREAK** statement.

```
WHILE Boolean_expression
BEGIN
    -- statements
   IF condition
        BREAK;
    -- other statements
END
```

- The **CONTINUE** statement stops the current iteration of the loop and starts the new one. The following illustrates the syntax of the **CONTINUE** statement:

```
WHILE Boolean_expression
BEGIN
    -- code to be executed
    IF condition
        CONTINUE;
    -- code will be skipped if the condition is met
END
```

CURSORS
in SQL Server

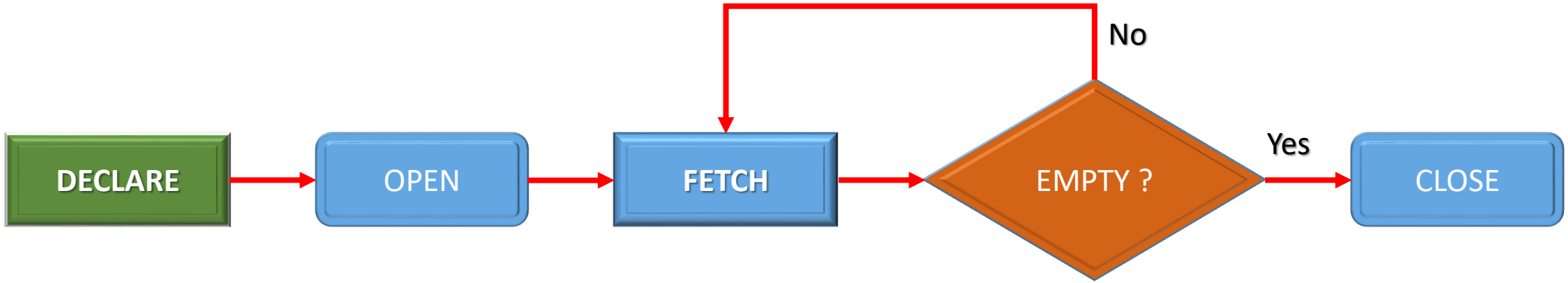


Cursors in RecordSet

Microsoft SQL Server statements produce a complete result set, but there are times when the results are best processed one row at a time. Opening a cursor on a result set *allows processing the result set one row at a time*. You can assign a cursor to a variable or parameter with a **cursor** data type.

	maSP	tenSP	hinhDD	ndTomTat	nhaSanXuat	ngayDang	maLoai	noiDung	taiKhoan	daDuyet	giaBan	giamGia
1	01	Túi xách nam - da bò	/images/product06.jpg	Giới thiệu túi xách ...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	1650000	20
2	02	Túi xách thời trang trẻ t...	/images/product01.jpg	Giới thiệu túi xách ...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	450000	15
3	03	Túi xách nữ - da trăn	/images/product07.jpg	Giới thiệu túi xách ...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	2300000	20
4	04	Đồng hồ nam - cơ thuy...	/images/product02.jpg	Giới thiệu đồng hồ...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	4500000	35
5	05	Giày nam trẻ trung	/images/product04.jpg	Giới thiệu giày nam...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	3500000	23
6	06	Giày nữ thời trang	/images/product05.jpg	Giới thiệu Giày nữ t...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	31500000	42
7	07	Dây nit nam - trung niên	/images/product08.jpg	Giới thiệu Dây nit n...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	300000	12
8	08	Bóp da nam - cá sấu	/images/product03.jpg	Giới thiệu Bóp da ...		2020-08-12 ...	6	Nội dung Giới thi...	minh	0	1300000	27

Step to use Cursor



Declare a pointer variable to *traverse* the rows on RecordSet

Open cursors and *identity the set*

Load one row from RecordSet into cursor

Repeat or stop, depending on whether the data is still available

Release *the active set*

Declare, Open, Close & Deallocate cursor

- Syntax:

  DECLARE <cursor_name> **CURSOR FOR** <select_statement> ;

  Open <cursor_name>;

  Close <cursor_name>;

  Deallocate <cursor_name>;

- Example

```
DECLARE product_cursor CURSOR FOR
                    Select * from sanPham Where giamGia>0 ;
Open product_cursor;
 …
Close product_cursor;
Deallocate product_cursor;
```

FETCH Data to Variable

- Syntax:

```
FETCH  [NEXT | PRIOR | FIRST | LAST ] FROM
                                <Cursor_name> [INTO Variable];
```

- Example:

```
Declare @ten nvarchar(500), @gia int;
DECLARE product_cursor CURSOR FOR
                    Select tenSP, giaBan from sanPham Where giamGia>0 ;
-- Fetch data to variable
FETCH NEXT FROM product_cursor INTO @ten, @gia;
```

@@FETCH_STATUS :: Function

@@FETCH_STATUS :: This function returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.

Return value	Description
0	The FETCH statement was successful.
-1	The FETCH statement failed or the row was beyond the result set.
-2	The row fetched is missing.
-9	The cursor is not performing a fetch operation.

@@CURSOR_ROWS :: Function

@@CURSOR_ROWS ::This returns the number of qualifying rows currently in the last cursor opened on the connection. To improve performance, SQL Server can populate large keyset and static cursors asynchronously. @@CURSOR_ROWS can be called to determine that the number of the rows that qualify for a cursor are retrieved at the time of the @@CURSOR_ROWS call.

Return value	Description
-m	The cursor populates asynchronously. The value returned (*-m*) is the number of rows currently in the keyset.
-1	The cursor is dynamic. Because dynamic cursors reflect all changes, the number of rows that qualify for the cursor constantly changes. The cursor does not necessarily retrieve all qualified rows.
0	No cursors have been opened, no rows qualified for the last opened cursor, or the last-opened cursor is closed or deallocated.
n	The cursor is fully populated. The value returned (*n*) is the total number of rows in the cursor.

Example to work with cursor

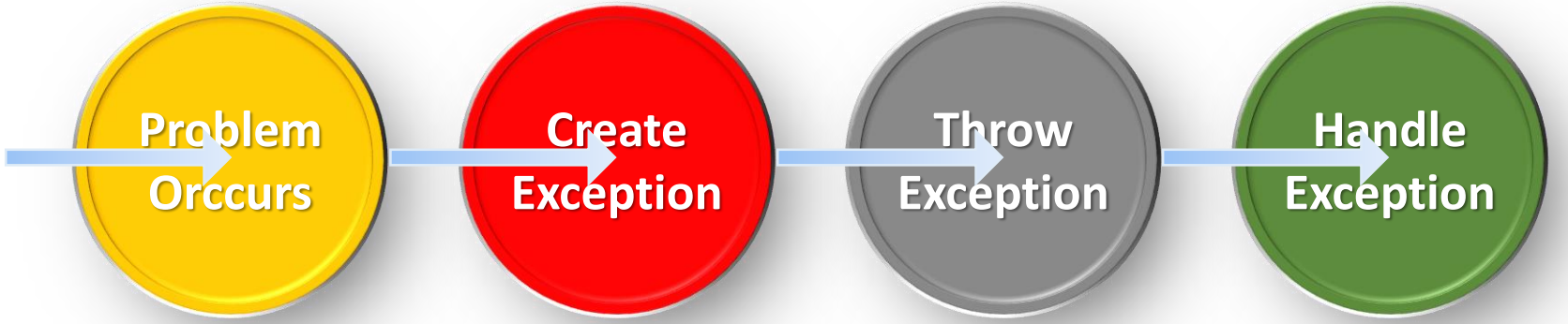
```
use BanHangOnline;
go
Declare @tenSanPham nvarchar(500), @gia int, @n int;
-- Declare a cursor variable for sanPham Record set
Declare product_cursor Cursor for
                    Select tenSP, giaBan from sanPham where giamGia>0;
-- Open cursor
Open product_cursor;
-- Fetch data to variable
Fetch next from product_cursor into @tenSanPham, @gia;
-- Loop if data is available
set @n=1;
while @@FETCH_STATUS=0
begin
    set @n = @n+1;
    print N'Tên sản phẩm:'+@tenSanPham ;
    print N'Giá bán: '+cast(@gia as varchar(20));
    Fetch next from product_cursor into @tenSanPham, @gia;
end
-- Close cursor and deallocate
Close product_cursor;
Deallocate product_cursor;
```

Example to work with cursor

Messages

```
Tên sản phẩm:Túi xách nam - da bò
Giá bán: 1650000
Tên sản phẩm:Túi xách thời trang trẻ trung - vải bố
Giá bán: 450000
Tên sản phẩm:Túi xách nữ - da trăn
Giá bán: 2300000
Tên sản phẩm:Đồng hồ nam - cơ thụy sỹ
Giá bán: 4500000
Tên sản phẩm:Giày nam trẻ trung
Giá bán: 3500000
Tên sản phẩm:Giày nữ thời trang
Giá bán: 31500000
Tên sản phẩm:Dây nịt nam - trung niên
Giá bán: 300000
Tên sản phẩm:Bóp da nam - cá sấu
Giá bán: 1300000
```

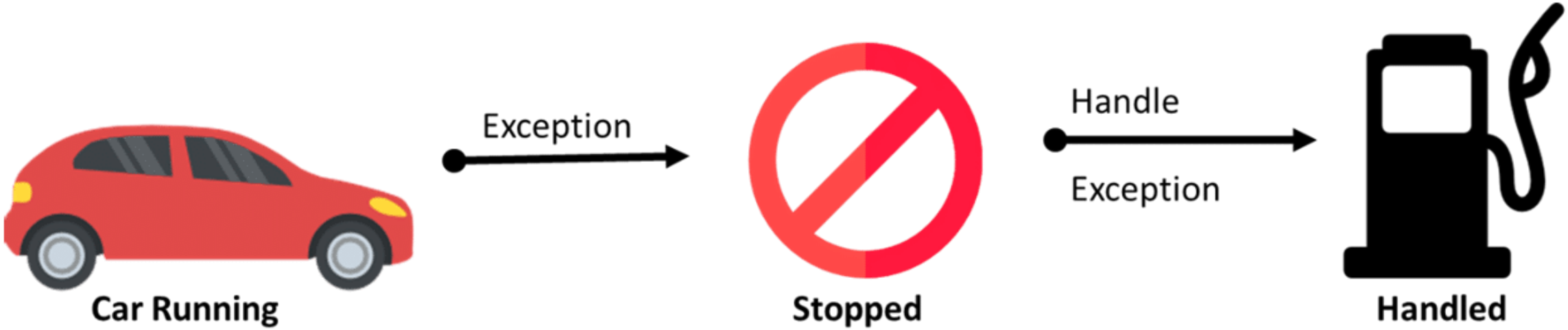
Exceptions Handling



TRY – CATCH block

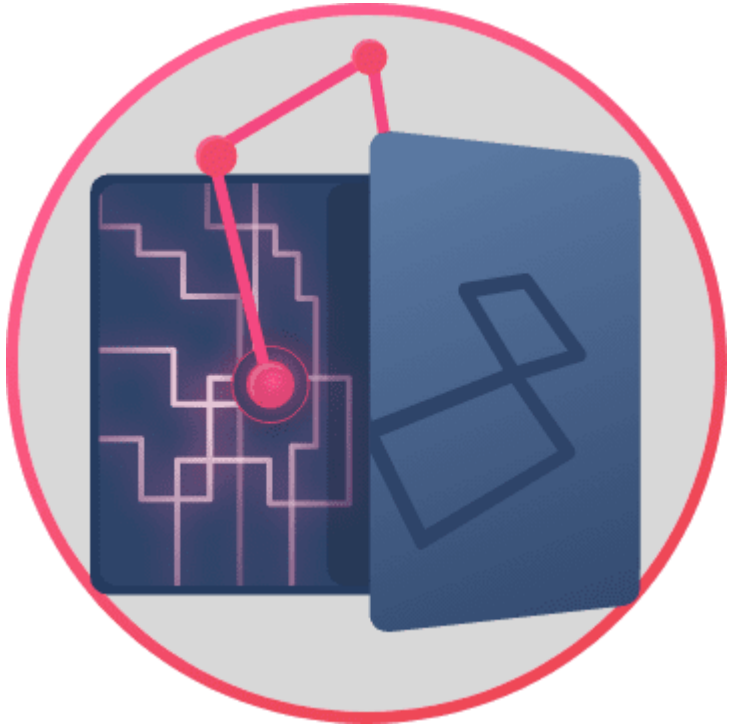
Implements **error handling** for Transact-SQL that is similar to the exception handling in the Microsoft Visual C# or Java languages.

A group of Transact-SQL statements can be enclosed in a **TRY** block. *If an error occurs in the **TRY** block, control is passed to* another group of statements that is enclosed in a ***CATCH** block*.



TRY – CATCH block

```
BEGIN TRY
    -- Statements that may cause exceptions
    . . .
END TRY
BEGIN CATCH
    -- Statements that handle exception
    . . .
END CATCH
```



Catch function

- **ERROR_MESSAGE():** This function returns the message text of the error that caused the CATCH block of a TRY...CATCH construct to execute.
- **ERROR_LINE():** This function returns the line number of occurrence of an error that caused the CATCH block of a TRY ... CATCH construct to execute.
- **ERROR_STATE():** Returns the state number of the error that caused the CATCH block of a TRY ... CATCH construct to be run.

Nhớ gì ?!!!

- Stored Procedure & những ưu điểm khi sử dụng cho Database
- DDL & Stored Procedure: Syntax
- Khai báo, thiết lập giá trị cho biến
- Phạm vi của tập lệnh
- Flow of control trong lập trình với Stored Procedure
- Khái niệm **Cursors** ::

  DECLARED – OPEN – FETCH – CLOSE – DEALLOCATE

Tài liệu tham khảo

- Itzik Ben-Gan, "**Microsoft® SQL Server ® 2012 T-SQL Fundamentals**", O'Reilly Media Inc, 2012
- Itzik Ben-Gan, Dejan Sarka, Ed Katibah, Greg Low, Roger Wolter, and Isaac Kunen, "**Inside Microsoft SQL Server 2008: T-SQL Programming**", Microsoft Press, 2010
- Microsoft SQL Docs, "**Tutorials for SQL Server**", https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver15, 10:54 PM, 18/06/2020