TÓM TẮT

Ngày học	@December 19, 2023

A. CƠ SỞ:

I. Insert data:

📝 **insert into** <tên bảng> (ds các cột cần chèn data) *(các cột cách nhau dấu ,)*

values (ds các giá trị tương ứng vs các cột)

Thêm data từ data của các bảng đã có sẵn:

📝 **Insert into** <tên bảng> (ds các cột cần chèn data)

Câu lệnh select (các giá trị tương ứng các cột)

II. Update date

Cập nhật data liên quan đến 1 bảng:

📝 **Update** <tên bảng>

Set tên cột 1 = <biểu thức 1>

[**,** tên cột 2 = <biểu thức 2>]

...

[**,** tên cột n = <biểu thức n>]

[**where** <điều kiện update data>]

Cập nhật data liên quan đến nhiều bảng

📝 **update** <tên bảng cần update data>

set tên cột 1 = <biểu thức 1>

[**,** tên cột 2 = <biểu thức 2>]

...

[**,** tên cột n = <biểu thức n>]

[**from** <tên bảng **liên quan**>]

[**where** <điều kiện nối, update data>]

III. Delete data

Xóa data liên quan đến 1 bảng

📝 delete from <tên bảng cần xóa>
[where <điều kiện xóa data>]

Xóa data liên quan đến nhiều bảng:

📝 **delete from** <tên bảng cần xóa>

[**from** <các bảng **liên quan**>]

[**where** <điều kiện xóa data>]

💡 Nếu cần reset lại Indentity:
Execute lệnh:
DBCC CHECKIDENT('Tên_bảng', RESEED, 0)

IV. BACK UP DATABASE

💡 BACKUP DATABASE Tên_database
TO DISK = 'Đường dẫn\Tên_muốn đặt.bak'

B. T - SQL

I. Khai báo biến

💡 **DECLARE @Tên_biến Kiểu_dữ_liệu [= Giá_trị_khởi tạo]**

> - Nếu ngắt lệnh bởi GO, thì muốn dùng lại biến đó, thì phải khai báo lại

II. Phép gán

Trong T-SQL, có 2 cách để viết phép gán:

Cách 1: Dùng lệnh SET theo cú pháp

```
SET @Tên_biến = Biểu thức
```

> **Lưu ý: Trong SQL, 1 câu lệnh SELECT trả về 1 dòng và tối đa 1 cột được xem là 1 biểu thức.**
>
> Mỗi lệnh **SET** chỉ thực hiện phép gán cho 1 biến (không viết gộp).

Cách 2: Sử dụng lệnh SELECT

```
SELECT @Tên_biến_1 = Biểu_thức 1,
        @Tên_biến_2 = Biểu_thức_2,
        …….
        @Tên_biến_n = Biểu_thức_N
FROM …..
WHERE …..
```

Phép gán này thường dùng để truy vấn dữ liệu và lưu kết quả vào biến.

III. Cấu trúc điều khiển

a. Rẽ nhánh

```
IF điều_kiện
    Khối_lệnh_của_IF
ELSE
    Khối_lệnh_của_ELSE
```

b. Lặp

💡 **WHILE điều_kiện**
Khối_lệnh_của_WHILE

> Chú ý: Có thể sử dụng lệnh **BREAK** và **CONTINUE** trong vòng lặp WHILE.
>
> **NHỚ: viết điều kiện dừng, lặp**

IV. Biến kiểu bảng

- Biến kiểu bảng trong T-SQL có thể dùng để biểu diễn dữ liệu “phức tạp”.

💡 **DECLARE @tên_biến TABLE**
(
Khai_báo_các_cột_của_bảng_(như_lệnh_CREATE_TABLE)
Tên_cột kiểu_data
)

> Một biến kiểu bảng sử dụng để lưu trữ dữ liệu dưới dạng tập các dòng và các cột.
> Không thể sử dụng các lệnh GÁN thông thường cho các biến kiểu bảng.
> Trên biến kiểu bảng, chỉ dùng các lệnh SELECT, INSERT, UPDATE và DELETE.
> Biến bảng được khởi tạo với DECLARE độc lập.

V. Bảng tạm (Temporary table)

Bảng tạm (thường lưu trữ trong bộ nhớ cache) thường được sử dụng để lưu trữ tạm thời dữ liệu dạng bảng, tuy nhiên phạm vi tồn tại của bảng tạm thì “lâu” hơn so với biến bảng.

- **Biến bảng**: phạm vi tồn tại trong khối lệnh mà nó được khai báo.
- **Bảng tạm**: phạm vi tồn tại là trong phiên làm viên (session).

Khác với biến bảng, bảng tạm được tạo ra bằng cách tương tự như khi tạo ra bảng vật lý:

- **Dùng lệnh CREATE TABLE**
- **Dùng lệnh SELECT … INTO … (thường dùng)**

Bảng tạm khi tạo ra được quản lý bởi CSDL tempdb (Databases → System Databases → tempdb)

> ### Tên bảng tạm phải bắt đầu bởi dấu #

- Tên phải bắt đầu bởi dấu #
- Sử dụng được trong phiên làm việc mà tạo ra nó
- Giải phóng (xóa) khi kết thúc phiên làm việc hoặc do chúng ta chủ động xóa bằng lệnh
DROP TABLE

- Sử dụng bảng tạm tương tự như bảng vật lý: SELECT, INSERT, UPDATE, DELETE

Cần truy vấn dữ liệu từ database và lưu vào 1 cấu trúc dạng bảng để xử lý

- Dùng biến bảng:
  - Khai báo 1 biến bảng có cấu trúc phù hợp với kết quả của truy vấn.
  - Đưa dữ liệu vào biến bảng bằng lệnh: **INSERT ….**
    SELECT ….
- Dùng bảng tạm: Dùng lệnh **INSERT … INTO**

VI. Common Table Expression (CTE)

- Truy vấn con (sub-query): sử dụng khi cần thực thi 1 câu lệnh SELECT và sử dụng kết quả câu lệnh đó bên trong 1 câu lệnh khác (SELECT, INSERT, UPDATE,

DELETE)

- CTE cũng được sử dụng để thực thi 1 câu lệnh SELECT, lưu tạm thời để sử dụng cho 1 câu lệnh khác.

```
SELECT p.*, tk.SumOfQuatity
FROM Products as p
    LEFT JOIN (SELECT ProductId, SUM(Quantity) as SumOfQuatity
                FROM OrderDetails
                GROUP BY ProductId
            ) AS tk ON p.ProductId = tk.ProductId

-- Viết lại bằng cách dùng CTE:
WITH cte_Products AS
(
    SELECT ProductId, SUM(Quantity) AS SumOfQuatity
    FROM OrderDetails
    GROUP BY ProductId
)

SELECT p.*, tk.SumOfQuatity
FROM Products AS p
    LEFT JOIN cte_Products AS tk ON p.ProductId = tk.ProductId
```

Cú pháp để khai báo CTE

💡
```
; WITH cte_name1 AS
(
     Lệnh_SELECT
)
, cte_name2 AS
(
     Lệnh_SELECT
)
….
,  cte_nameN AS
(
     Lệnh_SELECT
)

Lệnh_sử_dụng_các_CTE
```

> **LƯU Ý: Lệnh SELECT để lấy dữ liệu cho các CTE dưới có thể truy vấn dữ liệu từ các CTE đã định nghĩa trước đó**
> **Trước câu lệnh của CTE, phải có dấu ;**

```
;WITH cte_Category AS
(
    SELECT  CategoryId, CategoryName
    FROM Categories
)
, cte_Products AS
(
    SELECT ProductName, CategoryId, Price
    FROM Products
    WHERE Price < 5
)

SELECT *
```

```
FROM cte_Category as t1
    JOIN cte_Products as t2
    ON t1.CategoryId = t2.CategoryId
```

C. THỦ TỤC

IF EXISTS (SELECT * FROM sys.objects WHERE name = 'Tên_thủ_tục')
DROP PROCEDURE Tên_thủ_tục
GO

CREATE PROCEDURE Tên_thủ_tục
    Danh_sách_tham_số_(nếu có)
AS
BEGIN

SET NOCOUNT ON;
    Phần_thân_của_thủ_tục
END
GO

EXECUTE Tên_thủ_tục
        Danh_sách_đối_số_truyền_cho_thủ_tục
hoặc
EXEC Tên_thủ_tục Danh_sách_đối_số

VD: EXECUTE proc_Category_InsertOrUpdate
        @description = N'Các loại hàng thực phẩm',
        @categoryName = N'Thực phẩm';

1. Tham số của thủ tục

- Nếu một tham số mà không có giá trị mặc định thì bắt buộc phải được truyền giá trị khi sử dụng.

- Trong trường hợp tham số có giá trị mặc định thì có thể bỏ qua không truyền giá trị khi sử dụng.
- Tham số đầu ra (output): là tham số mà sự thay đổi về giá trị của nó trong thủ tục sẽ được giữ nguyên sau khi thoát khỏi thủ tục (tương tự như tham biến trong ngôn ngữ lập trình).
- Tham số đầu ra khi khai báo (khi tạo proc) có thêm từ khóa **output**

@Tên_tham_số Kiểu_dữ_liệu output

- Khi sử dụng, nếu muốn 1 tham số là được dùng dưới dạng output thì:
  - Phải truyền bằng biến
  - Phải kèm theo từ khóa **output** khi truyền đối số (từ output trong SQL như & trong code) (truyền ở tạo thủ tục và truyền đối số đó khi thực thi)

2. Tham số đầu vào kiểu bảng

Để sử dụng tham số đầu vào của thủ tục dưới dạng bảng, cần:

- Định nghĩa kiểu dữ liệu dạng bảng, sử dụng cú pháp:

```
CREATE TYPE Tên_kiểu_dữ_liệu_bảng AS TABLE
(
        Cấu_trúc_bảng
)
```

- Khai báo tham số đầu vào kiểu bảng cho thủ tục theo cú pháp:

@Tên_tham_số Kiểu_dữ_liệu_bảng READONLY
(READONLY là bắt buộc)

Tham số kiểu bảng khi truyền phải truyền bằng biến và nó có tính chất “chỉ đọc” khi sử dụng trong thủ tục (tức là chỉ được phép dùng lệnh SELECT đối với tham số này)

D. HÀM

I. Scalar function

- Hàm dạng này trả về 1 giá trị

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'Tên_hàm')
        DROP FUNCTION Tên_hàm;
GO
CREATE FUNCTION Tên_hàm (Danh_sách_tham_số)
RETURNS Kiểu_dữ_liệu_trả_về_của_hàm
AS
BEGIN
        DECLARE @biến_trả_về kiểu_data;
        SELECT @biến_trả_về =  (thường sẽ là hàm gộp)
                                        FROM …..

 Phần thân của hàm
        RETURN @biến_trả_về;
END
GO
```

- Với hàm, không thể sử dụng tham số kiểu bảng như thủ tục.

Không được sử dụng các lệnh có trả dữ liệu về cho Client (VD: SELECT) bên trong phần thân của hàm.

Gọi hàm:

SELECT **dbo.Tên_hàm(biến_truyền)**

II. Hàm với giá trị trả về kiểu bảng (table-valued function)

- Hàm dạng này có thể sử dụng ở những vị trí mà 1 bảng là được cho phép (**sau FROM**).
- Hàm dạng này chia thành 2 loại:

> Inline function: là loại hàm mà kết quả trả về của hàm chỉ cần giải quyết bởi duy nhất 1 câu lệnh SELECT (**Trong phần thân của hàm chỉ là một câu lệnh SELECT**).

> Multi-statement function: là loại hàm mà bên trong phần thân của hàm có thể sử dụng kết hợp nhiều lệnh để lập trình.

1. Inline function

💡
```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'Tên_hàm')
        DROP FUNCTION Tên_hàm;
GO
CREATE FUNCTION Tên_hàm (Danh_sách_tham_số)
RETURNS TABLE
AS
RETURN
(
        Câu_lệnh_SELECT_trả_dữ_liệu_về_cho_hàm
)
GO
```

2. Multi-statement function

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'Tên_hàm')
	DROP FUNCTION Tên_hàm;
GO
CREATE FUNCTION Tên_hàm (Danh_sách_tham_số)
RETURNS @Tên_biến TABLE
(
	Cấu_trúc_của_bảng_chứa_dữ_liệu_trả_về
)
AS
BEGIN

	RETURN;
END
GO
```

- Phải định nghĩa cấu trúc của bảng chứa dữ liệu trả về
- Cuối cùng của phần thân hàm phải là lệnh RETURN và chỉ có **duy nhất 1 lệnh RETURN** trong thân hàm. (Dữ liệu được return chính là @Tên_biến).
- Trong quá trình lập trình trong hàm, phải đưa được dữ liệu vào @Tên_biến. (INSERT INTO @Tên_biến (…….)…)
- Trong thân hàm không được sử dụng câu lệnh có kiểu trả về cho Client.

E. TRIGGER

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'Tên_Trigger')
DROP TRIGGER Tên_Trigger;
GO

CREATE TRIGGER trg_xxxx
ON tên_bảng /Khung nhìn / ALL SERVER
FOR / ALTER {[INSERT][,][UPDATE][,][DELETE]}
AS
BEGIN
        SET NOCOUNT ON;
        [
IF UPDATE(tên_cột)
        [AND UPDATE(tên_cột)|OR UPDATE(tên_cột)]
        ...]
        các_câu_lệnh_của_trigger
END
GO
```

> **Sau mệnh đề ON: bảng mà trigger cần tạo sẽ tác động đến**.
> • Khi câu lệnh
> **DELETE** được thực thi trên bảng, các **dòng dữ liệu bị xoá sẽ được sao chép vào trong bảng DELETED.**
> • Khi câu lệnh
> **INSERT** được thực thi trên bảng, **dòng dữ liệu được bổ sung vào bảng sẽ được sao chép vào trong bảng INSERTED,**
> • Khi câu lệnh
> **UPDATE** được thực thi trên bảng, **các dòng dữ liệu cũ chịu sự tác động của câu lệnh sẽ được sao chép vào bảng DELETED**, còn **trong bảng INSERTED sẽ là các dòng sau khi đã được cập nhật**.

VD:

```
CREATE TRIGGER trg_nhatkybanhang_insert
ON nhatkybanhang
FOR INSERT
AS
        UPDATE mathang
        SET mathang.soluong = mathang.soluong - inserted.soluong
        FROM mathang INNER JOIN inserted
        ON mathang.mahang = inserted.mahang
```

Sử dụng mệnh đề IF UPDATE trong trigger

F. CẤP QUYỀN

💡
```
-- Tạo tài khoản
use master
go
create login Tên_user with password = 'password'
go

-- Tạo người dùng CSDL
use Tên_CSDL;
go
create user Tên_user for login Tên_user;
go

-- Cấp quyền
grant select, update on Tên_bảng to Tên_user;

grant execute on Tên_thủ_tục_muốn_cấp_quyền_1 to Tên_user;
grant execute on Tên_thủ_tục_muốn_cấp_quyền_2 to Tên_user;
…
grant execute on Tên_thủ_tục_muốn_cấp_quyền_n to Tên_user;

grant select on Tên_hàm_muốn_cấp_quyền_1 to Tên_user;
…
grant select on Tên_hàm_muốn_cấp_quyền_n to Tên_user
```

Các dạng

I. Thống kê full

Nếu không quen sài LEFT JOIN:

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'proc_ThongKeD
DROP PROCEDURE proc_ThongKeDangKyHoc;
```

```
GO

CREATE PROCEDURE proc_ThongKeDangKyHoc
(
    @MaLopHocPhan nvarchar(50),
    @TuNgay date,
    @DenNgay date
)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @tbl TABLE
    (
    Ngay date,
    SoLuongDangKy int
    )

    INSERT INTO @tbl (Ngay, SoLuongDangKy)
    SELECT NgayDangKy, COUNT(MaSinhVien)
    FROM LopHocPhan_SinhVien
    WHERE MaLopHocPhan = @MaLopHocPhan
    AND NgayDangKy BETWEEN @TuNgay AND @DenNgay
    GROUP BY NgayDangKy

    DECLARE @tmpngay date = @TuNgay
    WHILE (@tmpngay <= @DenNgay)
    BEGIN
        IF NOT EXISTS (SELECT * FROM LopHocPhan_SinhVien WHERE M
        INSERT INTO @tbl (Ngay, SoLuongDangKy) values (@tmpngay,
        SET @tmpngay = DATEADD(dd, 1, @tmpngay)
    END

    SELECT * FROM @tbl
    ORDER BY Ngay
END
GO
```

```
EXEC proc_ThongKeDangKyHoc
                        @MaLopHocPhan = 'L0001',
                        @TuNgay = '2023-12-01',
                        @DenNgay = '2023-12-31'
GO
```

💡 B1: Khai báo biến bảng: gồm Ngày, và giá trị cần thống kê
B2: INSERT vào bảng như bình thường
B3: Khai báo biến tmp để chạy vòng lặp, giá trị ban đầu là giá trị from
B4: Chạy vòng lặp WHILE (biến tmp < to)
Nếu không tồn tại giá trị biến tmp trong data, chèn vào bảng giá trị (biến tmp, 0)
Nhớ set biến chạy
B5: Nhớ SELECT hiển thị, và ORDER BY theo Ngay

Hàm

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'fn_GetRevenue
    DROP FUNCTION fn_GetRevenueByFullDates;
GO
CREATE FUNCTION fn_GetRevenueByFullDates
(
    @fromDate date,
    @toDate date
)
RETURNS @tbl TABLE
(
    SummaryDate date primary key,
    Revenue money
)
AS
BEGIN
    INSERT INTO @tbl(SummaryDate, Revenue)
```

```
            SELECT o.OrderDate, SUM(od.Quantity * od.SalePrice)
            FROM Orders as o JOIN OrderDetails as od
            on o.OrderId = od.OrderId
            WHERE o.OrderDate between @fromDate and @toDate
            GROUP BY o.OrderDate

    DECLARE @d date = @fromDate;
    WHILE @d <= @toDate
        BEGIN
        -- Lệnh Exists chỉ để kiểm tra có hay không có data (số
            IF NOT EXISTS (SELECT * FROM @tbl WHERE SummaryDate
                INSERT INTO @tbl(SummaryDate, Revenue) VALUES(@
            SET @d = DATEADD(DAY, 1, @d);
        END
    RETURN;
END
GO
--Test
SELECT *
FROM dbo.fn_GetRevenueByFullDates('2017/12/1', '2017/12/31')
```

Thủ tục

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'proc_ThongKeD
DROP PROCEDURE proc_ThongKeDangKyHoc;
GO

CREATE PROCEDURE proc_ThongKeDangKyHoc
(
    @MaLopHocPhan nvarchar(50),
    @TuNgay date,
    @DenNgay date
)
AS
BEGIN
```

```
    SET NOCOUNT ON;
    DECLARE @tblNgay Table
    (
         Ngay date
    )
    DECLARE @tmp date = @TuNgay;
    WHILE (@tmp <= @DenNgay)
    BEGIN
        insert into @tblNgay values (@tmp);
        set @tmp = DATEADD(day, 1, @tmp);
    END
    SELECT t1.ngay, ISNULL(t2.SoLuongDangKy, 0) as Revenue
    FROM @tblNgay AS t1
    LEFT JOIN
    (
        SELECT  NgayDangKy, COUNT(MaSinhVien) as SoLuongDangKy
        FROM LopHocPhan_SinhVien
        WHERE MaLopHocPhan = @MaLopHocPhan
        AND NgayDangKy BETWEEN @TuNgay AND @DenNgay
        GROUP BY NgayDangKy
    ) AS t2
    ON t1.ngay = t2.NgayDangKy
END
GO
```

Phân trang

Thủ tục proc_Invoice_Select
@Page int = 1,
@PageSize int = 0,
@CustomerId int = 0,
@FromTime datetime = null,
@ToTime datetime = null,
@RowCount OUTPUT,
@PageCount OUTPUT

Có chức năng tìm kiếm và hiển thị danh sách các hóa đơn hàng bán thẻ dưới dạng phân trang.
Thông tin cần hiển thị bao gồm: mã đơn hàng, mã khách hàng, tên khách hàng, thời điểm tạo hóa đơn, trạng thái hóa đơn, thời điểm kết thúc và tổng trị giá (tiền) của đơn hàng. Trong đó các tham số của thủ tục được mô tả như sau:
@Page ~ @Trang: Trang cần hiển thị dữ liệu
@PageSize ~ @SoDongMoiTrang : Số dòng trên mỗi trang (nếu @PageSize = 0 thì hiển thị toàn bộ dữ liệu tìm được)

@CustomerId : Mã của khách hàng cần tìm kiếm đơn hàng. Nếu @CustomerId = 0 thì tìm kiếm theo tất cả các khách hàng.

@FromTime : Thời điểm bắt đầu của khoảng thời gian cần tìm kiếm (tính theo thời điểm tạo đơn hàng). Trường hợp @FromTime là NULL thì không hạn chế thời điểm bắt đầu khi tìm kiếm.

@ToTime : Thời điểm kết thúc của khoảng thời gian cần tìm kiếm (tính theo thời điểm tạo đơn hàng). Trường hợp @ToTime là NULL thì không hạn chế thời điểm kết thúc khi tìm kiếm.

@RowCount ~ @SoLuong: Tham số đầu ra cho biết tổng số dòng dữ liệu tìm được
@PageCount : Tham số đầu ra cho biết tổng số trang.

“ROW_NUMBER() OVER(ORDER BY OrderDate)” nghĩa là gán một con số tăng tuần tự cho mỗi bản ghi theo thứ tự của OrderDate, bắt đầu từ 1

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'proc_Customer
DROP PROCEDURE proc_Customer_Select
GO

CREATE PROCEDURE proc_Customer_Select
(
    @Page int = 0,
    @PageSize int = 0,
    @SearchValue nvarchar(255) = N'',
    @RowCount int OUTPUT,
    @PageCount int OUTPUT
```

```
)
AS
BEGIN
    SET NOCOUNT ON;

    SELECT *, ROW_NUMBER() over(order by CustomerName) as RowNu
    INTO #TempCustomer
    FROM Customer

    SELECT @RowCount = COUNT(*)
    FROM Customer
    where (@SearchValue = N'') or (CustomerName like @SearchVal

    -- Set số trang
    IF(@PageSize = 0)
        SET @PageCount = 1
    ELSE
        BEGIN
            SET @PageCount = @RowCount / @PageSize;
            IF (@RowCount % @PageSize > 0)
                SET @PageCount += 1;
        END;


    ;WITH cte as
    (
        SELECT * from #TempCustomer
        where (@SearchValue = N'') or (CustomerName like @Searc
    )
    SELECT * FROM cte
    where (@PageSize = 0) or
            RowNumber between (@Page - 1) * @PageSize + 1 AND @
    ORDER BY RowNumber
END
GO
```

```
--TEST
DECLARE @RowCount int,
        @PageCount int
EXECUTE proc_Customer_Select
    @Page = 1,
    @PageSize = 10,
    @SearchValue = N'',
    @RowCount = @RowCount OUTPUT,
    @PageCount = @PageCount OUTPUT
GO
```