MỤC LỤC

GIÁO TRÌNH ĐÀO TẠO SQL

I. GIỚI THIỆU CHUNG

1. Đối tượng áp dụng:

- Những nhân viên kỹ thuật mới hoặc thực tập sinh.
- Đã được đào tạo về Cơ Sở Dữ Liệu SQL Server.

2. Mục đích:

- Giúp nhân viên nắm vững hơn về kiến thức cơ bản.
- Tiếp cận các bài toán và thuật toán thực tế.

3. Thống nhất cách thức làm việc:

- Khi làm bài xong gửi đính kèm theo file và đặt tên theo quy tắc: Họ và tên + Tên nhóm + Buổi học.
- Phải hoàn thành đúng theo thời gian được giao, tiếp cận kỹ năng đặt kế hoạch và hoàn thành kế hoạch.
- Trong quá trình trả bài: Phải show bài của mình trên máy chiếu và trình bày cách thức giải quyết các bài toán được giao để học kỹ năng thuyết trình.

4. Giới thiệu về phiên bản SQL Server:

- Enterprise
- Standard
- Developer
- Express

5. Lưu ý khi cài đặt SQL Server:

- Tìm hiểu các yêu cầu của các phiên bản SQL dự định cài đặt
- Đặt tên Named Instance:
- Mục đích cài đặt: Nếu cài đặt với mô hình lớn cần phải tìm hiểu thêm (AlwaysOn, Link server,….).

6. Giới thiệu về giao diện:

- **Các Database Hệ thống:**
  - **Master:** chứa thông tin về hệ thống SQL Server: Các tài khoản đăng nhập, cấu hình hệ thống, thông tin về các cơ sở dữ liệu đã tạo, các thủ tục hệ thống, các thủ tục do người dùng định nghĩa…Ta phải quản trị Database này cẩn thận, khi sửa đổi hệ thống thì cần Backup database Master.
  - **Tempdb** database chứa các đối tượng tạm thời như Global và Local Temporary Table. Tempdb tự động khởi tạo lại khi SQL Server được khởi động lại.
  - **Msdb:** Chứa thông tin về những bản backup của Database, thông tin về SQL Agent, SQL Server Jobs, các cảnh báo lỗi và một vài thông tin về replication như Log Shipping
  - **Model database**: đây là một Database template khi ta tạo các Database mới. Khi ta tạo mới một cơ sở dữ liệu thì SQL Server lấy tất cả các mẫu (bao gồm Tables, Views,…) từ model database.
- **Security:** Thông tin về user đăng nhập SQL, Các Roles, Chứng chỉ.
- **Server Objects:**

- **Replication**: Tính năng nâng cao của SQL để đồng bộ dữ liệu (Tìm hiểu ngoài)
- **Always On High Avilability:** Tính năng nâng cao của SQL hỗ trợ chạy SQL 24/7 giảm thiểu rủi ro (Tìm hiểu ngoài)
- **Management**

  Ý nghĩa các Object trong 1 Database:

  - Tables
  - Views
  - Procedured
  - Functions
  - Trigger
  - ...

7. Cách thức viết câu lệnh:

- Nguyên tắc người viết code: "Tinh tế trong từng câu lệnh", luôn luôn phải hiểu "Viết ra đoạn code để cho người khác đọc và hiểu được" do đó:
  - Trình bày trong khuôn màn hình ngang máy tính
  - Sử dụng tab khi lùi dòng vào phía trong
  - Trình bày phải có dấu cách sau dấu ","
  - Trình bày theo khối lệnh, gọn gàng, cách dòng trong khối lệnh cho dễ quan sát
  - Các từ khóa viết hoa: SELECT …FROM
- Lưu ý đến Performance của đoạn lệnh:
  - Kiểm tra chỉ mục trên tất cả các field trên mệnh đề WHERE và JOIN trong câu lệnh SQL
  - Giới hạn kích thước của bảng dữ liệu làm việc: Kiểm tra các bảng được sử dụng trong câu SELECT để có thể áp dụng lọc qua WHERE
  - Chỉ chọn các field mà trong báo cáo cần tránh thừa thông tin.
  - Loại bỏ các phần tính toán trong mệnh đề WHERE và JOIN
  - Đặt Index với các Field hay được truy vấn.
  - Đặt kích thước của các field theo nguyên byte: 2^n
- Đầu mã thủ tục, hàm phải có thông tin chung:
  - Người tạo
  - Ngày khởi tạo
  - Mô tả chung
- Mỗi lần đoạn mã được sửa đổi phải được ghi chú (Comment) rõ ràng:
  - Người sửa đổi
  - Ngày sửa đổi
  - Nội dung chỉnh sửa
- Các đoạn lệnh có thuật toán phức tạp cần có ghi chú rõ ràng:
- Các câu ghi chú cần rõ ràng, ngắn gọn, đủ nghĩa, không dài dòng.

II. KIỂU DỮ LIỆU VÀ CÁC LỆNH ĐỊNH NGHĨA DỮ LIỆU

1. Kiểu dữ liệu trong SQL

Các Kiểu dữ liệu số

Kiểu dữ liệu	Từ	Đến
bigint	-9,223,372,036,854,775,808	9.223.372.036.854.775,807
int	-2.147.483.648	2.147.483.647
smallint	-32.768	32.767
tinyint	0	255
bit	0	1
decimal	-10 ^ 38 +1	10 ^ 38 -1
numeric	-10 ^ 38 +1	10 ^ 38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214.748.3648	+214.748.3647

Các Kiểu dữ liệu Date and Time

Kiểu dữ liệu	Từ	Đến
datetime	Ngày 1 tháng 1 năm 1753	Ngày 31 tháng 12 năm 9999
smalldatetime	1 tháng 1 năm 1900	6 tháng 6 năm 2079
date	Lưu trữ một ngày như ngày 30 tháng 6 năm 1991	
time	Lưu trữ thời gian trong ngày như 12:30 PM	

Các Kiểu dữ liệu chuỗi ký tự

No.	Kiểu dữ liệu & Mô tả
1	**Char** Chiều dài tối đa 8.000 ký tự (cố định chiều dài ký tự không phải là ký tự Unicode)

2	**Varchar** Tối đa 8.000 ký tự. (Dữ liệu không phải dạng Unicode có độ dài biến đổi).
3	**Varchar (max)**
4	**Text** Dữ liệu không phải dạng Unicode có chiều dài không thay đổi với độ dài tối đa là 2,147,483,647 ký tự

Các Kiểu dữ liệu chuỗi ký tự Unicode

No.	Kiểu dữ liệu & Mô tả
1	**Nchar** Chiều dài tối đa 4.000 ký tự. (Chiều dài cố định Unicode)
2	**Nvarchar** Chiều dài tối đa 4.000 ký tự. (Chiều dài biến Unicode)
3	**Nvarchar (MAX)**
4	**Ntxt** Chiều dài tối đa là 1.073.741.823 ký tự. (Chiều dài biến Unicode)

Các Kiểu dữ liệu nhị phân

No.	Kiểu dữ liệu & Mô tả
1	**binary** Chiều dài tối đa 8.000 byte (Dữ liệu nhị phân chiều dài cố định)
2	**varbinary** Chiều dài tối đa 8.000 byte (Dữ liệu nhị phân chiều dài biến đổi)
3	**varbinary(max)** Chiều dài tối đa 231 byte (chỉ SQL Server 2005). (Dữ liệu nhị phân chiều dài biến)
4	**image** Chiều dài tối đa là 2,147,483,647 byte. (Dữ liệu nhị phân chiều dài biến)

Các Kiểu dữ liệu khác:

No.	Kiểu dữ liệu & Mô tả
1	**sql_variant** Lưu trữ các giá trị của các Kiểu dữ liệu khác nhau được hỗ trợ bởi SQL Server, ngoại trừ văn bản, ntext và dấu thời gian.
2	**timestamp** Lưu trữ một số duy nhất trong cơ sở dữ liệu được cập nhật mỗi khi một hàng được cập nhật
3	**uniqueidentifier** Lưu trữ một định danh duy nhất trên toàn cầu(GUID)
4	**xml** Lưu trữ dữ liệu XML. Có thể lưu trữ các thể hiện xml trong một cột hoặc một biến.
5	**cursor - con trỏ** Tham chiếu Từ một đối tượng con trỏ
6	**table** Lưu trữ một tập kết quả để xử lý sau

2. Các câu lệnh định nghĩa dữ liệu

2.1. Lệnh CREATE

- *Ý nghĩa:* Lệnh CREATE dùng để tạo các đối tượng cơ sở dữ liệu như các bảng, các view, các thủ tục, hàm .v.v…
- *Cú pháp:*
  + CREATE TABLE <Tên bảng> (<Danh sách: Tên cột, kiểu cột>, <Điều kiện kiểm soát dữ liệu>)
  + CREATE VIEW <Tên View> (<Danh sách: Tên cột, kiểu cột>, <Điều kiện kiểm soát dữ liệu>) AS Q; với Q là một khối câu lệnh SELECT định nghĩa khung nhìn (view)
  + CREATE [UNIQUE] INDEX ON <Tên chỉ số> ON <Tên bảng> (Tên cột [ASC|DESC])

2.2. Lệnh thay thế sửa đổi ALTER

- *Ý nghĩa*: Dùng để thay đổi cấu trúc bảng của các đối tượng CSDL
- *Cú pháp:* ALTER TABLE <Tên bảng> <Thực hiện các lệnh trên cột>

  Các lệnh trên cột có thể là:

  - Xóa một cột: Drop Column
  - Thêm một cột: Add
  - Xóa khóa chính: DROP CONSTRAINT <Ten khóa chính>
  - Thiết lập khóa chính: ADD CONSTRAINT <Tên khóa chính> PRIMARY KEY (Tên trường,…)

2.3. Xoá cấu trúc DROP

- *Ý nghĩa:* Dùng để xóa các đối tượng cơ sở dữ liệu như Table, View, Index, .v.v…
- *Cú pháp:*

  DROP TABLE <Tên bảng>

  DROP VIEW <Tên View>

  DROP INDEX <Tên Index>

3. Các câu lệnh cập nhật dữ liệu

3.1. Lệnh Insert

- *Ý nghĩa: Dùng để chèn một hàng hoặc một số hàng cho bảng.*
- Cú pháp:
  + INSERT INTO <Tên bảng> (Danh sách các cột) VALUES (Danh sách các giá trị) hoặc
  + INSERT INTO <Tên bảng> (Danh sách các cột) (Các truy vấn con)

3.2. Lệnh Update

- *Ý nghĩa:* Dùng để sửa đổi dữ liệu.
- *Cú pháp:* UPDATE <Tên bảng> SET <Tên_cột_1=Biểu_thức_1, Tên_cột_2=Biểu_thức_2,…> [WHERE <Điều kiện>]

3.3. Lệnh Delete

- *Ý nghĩa:* Xoá một số hàng trong bảng.
- *Cú pháp:* DELETE FROM <Tên bảng> WHERE <Điều kiện>

4. Bài tập:

Thiết kế CSDL Tech với cấu trúc

Lưu ý tên trường và tên bảng đặt đúng để dễ chấm bài
Tạo một database có tên: Tech bằng cách viết lệnh SQL
Tạo các bảng sau bằng cách viết lệnh SQL.

Item	
ItemCode	NVARCHAR (16)
ItemName	NVARCHAR (96)
Unit	NVARCHAR (10)
ItemType	INT -> 0: dịch vụ; 1: vật tư; 2- sản phẩm
IsActive	INT -> 0: Không sử dụng; 1: Sử dụng

Customer	
CustomerCode	NVARCHAR (16)
CustomerName	NVARCHAR (96)
CustomerType	INT -> 1: Cá nhân; 2: Đơn vị; 3: Nội bộ
IsActive	INT -> 0: Không sử dụng; 1: Sử dụng

AccDoc	
DocCode	CHAR (2)
DocNo	NVARCHAR (10) -> Duy nhất (khóa)
DocDate	DATETIME
CustomerCode	NVARCHAR (16)
DocGroup	INT -> 1: Nhập; 2: Xuất
Description	NVARCHAR (256)
IsActive	INT -> 0: Không sử dụng; 1: Sử dụng

AccDocDetail	
DocCode	CHAR (2) -> Link DocCode của AccDoc
DocNo	NVARCHAR (10) -> Link DocNo của AccDoc
ItemCode	NVARCHAR (16)
Quantity	NUMERIC (15,3)
UnitCost	NUMERIC (15,5)
Amount1	NUMERIC (18,2)
UnitPrice	NUMERIC (15,5)
Amount2	NUMERIC (18,2)

OpenInventory	
ItemCode	NVARCHAR (16)
Quantity	NUMERIC (15,3)
Amount	NUMERIC (18,2)

Đáp án: Kiểm tra thực hiện theo các quy tắc viết code như đã giảng phần trên hay chưa. Kiểm tra xem thực hiện đúng các lưu ý đã dạy.

Dùng câu lệnh Insert đưa dữ liệu vào các bảng như sau:

- **Bảng Item:**

ItemCode	ItemName	CustomerType	ItemType	IsActive
NVLC01	Máy nén	Cái	1	1
NVLC02	Tôn dày 0.5 mm	Kg	1	1
NVLC03	Ống đồng	M	1	1
TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	2	1
TP02	Tủ đông kích thước 1.5*0.8*0.8	Cái	2	1
TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	2	1

- **Bảng Customer**

CustomerCode	CustomerName	CustomerType	IsActive
NCC01	Công ty TNHH Vạn Xuân	2	1
NCC02	Công ty Cổ phần Đại Phát	2	1
NCC03	Công ty Cổ phần tôn Hòa Phát	2	1
KH01	Đại lý Cô Tám	1	1
KH02	Công ty cổ phần đầu tư xây dựng Dacinco	2	1
KH03	Công ty TNHH cà phê Thắng Lợi	2	1
NB01	Phi Công Anh	1	1
NB02	Đàm Văn Đức	1	1
NB03	Phân xưởng sản xuất	3	1

- **Bảng AccDoc**

DocCode	DocNo	DocDate	CustomerCode	DocGroup	Description	IsActive
NM	NM001	01/01/2022	NCC01	1	Nhập mua NPL	1
NM	NM002	10/01/2022	NCC02	1	Nhập mua NPL	1
NM	NM003	11/01/2022	NCC01	1	Nhập mua NPL	1
NM	NM004	15/01/2022	NCC03	1	Nhập mua NPL	1
PX	PX001	02/01/2022	NB03	2	Xuất sản xuất	1
PX	PX002	13/01/2022	NB03	2	Xuất sản xuất	1
PX	PX003	22/01/2022	NB03	2	Xuất sản xuất	1
PX	PX004	28/01/2022	NB03	2	Xuất sản xuất	1
TP	NM001	15/01/2022	NB03	1	Nhập thành phẩm	1
TP	NM002	31/01/2022	NB03	1	Nhập thành phẩm	1
HD	HD001	05/01/2022	KH01	2	Xuất bán hàng	1
HD	HD002	06/01/2022	KH03	2	Xuất bán hàng	1
HD	HD003	10/01/2022	KH02	2	Xuất bán hàng	1
HD	HD004	12/01/2022	KH01	2	Xuất bán hàng	1
HD	HD005	16/01/2022	KH02	2	Xuất bán hàng	1
HD	HD006	18/01/2022	KH03	2	Xuất bán hàng	1
HD	HD007	23/01/2022	KH01	2	Xuất bán hàng	1
HD	HD008	31/01/2022	KH03	2	Xuất bán hàng	1

- **Bảng AccDocDetail**

DocCode	DocNo	ItemCode	Quantiy	UnitCost	Amount1	UnitPrice	Amount2
NM	NM001	NVLC01	100.00	3,099,000.00	309,900,000	0	0
NM	NM001	NVLC02	999.95	29,956.35	29,954,852	0	0
NM	NM002	NVLC01	50.00	3,050,000.00	152,500,000	0	0

NM	NM002	NVLC02	200.33	29,956.55	6,001,196	0	0
NM	NM002	NVLC03	2,000.00	105,987.92	211,975,840	0	0
NM	NM003	NVLC01	60.00	3,050,000.00	183,000,000	0	0
NM	NM003	NVLC02	100.00	29,956.55	2,995,655	0	0
NM	NM003	NVLC03	150.00	105,987.92	15,898,188	0	0
NM	NM004	NVLC01	90.00	3,050,000.00	274,500,000	0	0
NM	NM004	NVLC02	300.00	29,956.55	8,986,965	0	0
NM	NM004	NVLC03	200.00	105,987.92	21,197,584	0	0
PX	PX001	NVLC01	90.00	0	0	0	0
PX	PX001	NVLC02	500.00	0	0	0	0
PX	PX001	NVLC03	400.00	0	0	0	0
PX	PX002	NVLC01	50.00	0	0	0	0
PX	PX002	NVLC02	200.33	0	0	0	0
PX	PX002	NVLC03	1,000.00	0	0	0	0
PX	PX003	NVLC01	100.00	0	0	0	0
PX	PX003	NVLC02	150.00	0	0	0	0
PX	PX003	NVLC03	200.00	0	0	0	0
PX	PX004	NVLC01	90.00	0	0	0	0
PX	PX004	NVLC02	320.00	0	0	0	0
PX	PX004	NVLC03	170.00	0	0	0	0
TP	TP001	TP01	400.00	0	0	0	0
TP	TP001	TP02	500.00	0	0	0	0
TP	TP001	TP03	700.00	0	0	0	0
TP	TP002	TP01	300.00	0	0	0	0
TP	TP002	TP02	200.00	0	0	0	0
TP	TP002	TP03	200.00	0	0	0	0
HD	HD001	TP02	100.00	0	0	11,000,000	1,100,000,000
HD	HD001	TP03	300.00	0	0	12,000,000	3,600,000,000
HD	HD002	TP01	50.00	0	0	10,000,000	500,000,000
HD	HD002	TP03	150.00	0	0	12,000,000	1,800,000,000

HD	HD003	TP01	50.00	0	0	10,000,000	500,000,000
HD	HD003	TP02	100.00	0	0	11,000,000	1,100,000,000
HD	HD004	TP01	90.00	0	0	10,000,000	900,000,000
HD	HD004	TP02	270.00	0	0	11,000,000	2,970,000,000
HD	HD004	TP03	140.00	0	0	12,000,000	1,680,000,000
HD	HD005	TP02	100.00	0	0	11,000,000	1,100,000,000
HD	HD005	TP03	300.00	0	0	12,000,000	3,600,000,000
HD	HD006	TP01	50.00	0	0	10,000,000	500,000,000
HD	HD006	TP03	150.00	0	0	12,000,000	1,800,000,000
HD	HD007	TP01	50.00	0	0	10,000,000	500,000,000
HD	HD008	TP02	100.00	0	0	11,000,000	1,100,000,000
HD	HD008	TP01	90.00	0	0	10,000,000	900,000,000
HD	HD008	TP02	270.00	0	0	11,000,000	2,970,000,000
HD	HD008	TP03	140.00	0	0	12,000,000	1,680,000,000

- **Bảng OpenInventory**

ItemCode	Quantity	Amount
NVLC01	1,000	3,000,000,000
NVLC02	1,500	46,500,000
NVLC03	2,000	210,000,000
TP01	500	2,000,000,000
TP02	400	2,000,000,000
TP03	600	2,520,000,000

III. CÁC HÀM VÀ TOÁN TỬ TRONG SQL

1. Các hàm thường sử dụng trong SQL

1.1. Hàm Toán học:

Hàm	Diễn Giải
POWER(X,Y)	Giá trị của X lũy thừa Y
ROUND(X,n)	Số X làm tròn, n chữ số tính từ dấu thập phân

SQUARE(X)	Bình phương giá trị X
SQRT(X)	Căn bậc 2 giá trị X
ABS (X)	Lấy giá trị tuyệt đối X
FLOOR (X)	Làm tròn lên phần nguyên giá trị X (thập phân)
CEILING (X)	Làm tròn xuống phần nguyên giá trị X (thập phân)
A%B	Lấy số dư phép chia A/B
SIGN(X)	Trả về kiểu số X (số thập phân) và giá trị: 1- Số dương, 0- Số 0; -1 Số âm
RAND(@_X)	Trả về số ngẫu nhiên cho biến @_X

1.2. Hàm chuỗi:

Hàm	**Diễn Giải**
"Chuỗi 1" + "Chuỗi 2"	Nối chuỗi bằng toán tử cộng
SPACE(n)	Trả về chuỗi có n khoảng trắng
LOWER(Chuỗi kỹ tự)	Chuyển chuỗi thành chữ thường
UPPER(Chuỗi ký tự)	Chuyển thành chữ hoa
LTRIM(Chuỗi ký tự)	Trả về dữ liệu không có khoảng trắng bên trái
RTRIM(Chuỗi ký tự)	Trả về dữ liệu không có khoảng trắng bên phải
LEFT(Chuỗi ký tự, n)	Trả về một chuỗi n ký tự tính từ bên trái
RIGHT(Chuỗi ký tự, n)	Trả về một chuỗi n ký tự tính từ bên phải
SUBSTRING(Chuỗi ký tự, n1, n2)	Trả về một chuỗi bắt đầu từ vị trí n1 lấy n2 ký tự
REPLACE(Chuỗi 1, Chuỗi 2, Chuỗi 3)	Thay thế chuỗi 2 bằng chuỗi 3 ở chuỗi 1
CHARINDEX (Chuỗi 1, Chuỗi 2 [,Từ vị trí]	Trả về vị trí tìm thấy Chuỗi 1 trong Chuỗi 2 [Tính từ vị trí]
STUFF(Chuỗi 1, vị trí, chiều dài, Chuỗi 2)	Chèn Chuỗi 2 vào Chuỗi 1 từ "vị trí" và xóa bỏ số ký tự bằng "chiều dài" Chuỗi 1
LEN("Chuỗi 1")	Trả về giá trị độ dài chuỗi 1

REVERSE()	Đảo ngược chuỗi

2. Toán tử trong SQL

Toán tử là một từ dành riêng hoặc một ký tự được sử dụng chủ yếu trong mệnh đề WHERE của câu lệnh SQL để thực hiện các thao tác, chẳng hạn như so sánh và các phép toán số học. Các toán tử này được sử dụng để chỉ định các điều kiện trong câu lệnh SQL và dùng làm ngữ pháp cho nhiều điều kiện trong một câu lệnh.

- Toán tử số học
- Toán tử so sánh
- Toán tử logic

2.1. Các toán tử số học trong SQL

Giả sử **'biến a'** là 10 và **'biến b'** là 20, khi đó

Toán tử	Mô tả	Ví dụ
+ (cộng)	Thêm các giá trị ở hai bên của toán tử.	a + b sẽ cho 30
- (trừ)	Lấy toán hạng bên phải trừ toán hạng bên trái.	a - b sẽ cho -10
* (nhân)	Nhân giá trị ở hai bên của toán tử.	A * b sẽ cho 200
/ (chia)	Chia lấy phần nguyên.	b / a sẽ cho 2
% (Modulus)	Chia lấy phần dư.	b % a sẽ cho 0

2.2. Các toán tử so sánh trong SQL

Giả sử **'biến a'** là 10 và **'biến b'** là 20, khi đó

Toán tử	Mô tả	Ví dụ
=	Kiểm tra nếu các giá trị của hai toán hạng bằng nhau hay không, nếu có thì điều kiện trở thành true.	(a = b) là false.
!=	Kiểm tra nếu các giá trị của hai toán hạng bằng hoặc không, nếu các giá trị không bằng nhau thì điều kiện trở thành true.	(a != b) là true.
<>	Kiểm tra nếu các giá trị của hai toán hạng bằng hoặc không, nếu các giá trị không bằng nhau thì điều kiện trở thành true.	(a <> b) là true.
>	Kiểm tra nếu giá trị của toán hạng trái lớn hơn giá trị của toán hạng phải, nếu có thì điều kiện trở thành true.	(a > b) là là false.

<	Kiểm tra nếu giá trị của toán hạng trái nhỏ hơn giá trị của toán hạng phải, nếu có thì điều kiện trở thành true.	(a < b) là true.
>=	Kiểm tra nếu giá trị của toán hạng trái lớn hơn hoặc bằng giá trị của toán hạng phải, nếu có thì điều kiện trở thành true.	(a >= b) là là false.
<=	Kiểm tra nếu giá trị của toán hạng trái nhỏ hơn hoặc bằng giá trị của toán hạng phải, nếu có thì điều kiện trở thành true.	(a <= b) là true.
!<	Kiểm tra nếu giá trị của toán hạng trái không nhỏ hơn giá trị của toán hạng phải, nếu có thì điều kiện trở thành true.	(a !< b) là sai.
!>	Kiểm tra nếu giá trị của toán hạng trái không lớn hơn giá trị của toán hạng phải, nếu có thì điều kiện trở thành true.	(a! b) là true.

2.3. Các toán tử logical trong SQL

Dưới đây là danh sách tất cả các toán tử logic có sẵn trong SQL

Stt	Toán tử & mô tả
1	**ALL** Toán tử ALL được sử dụng để so sánh một giá trị với tất cả các giá trị trong một tập hợp giá trị khác.
2	**AND** Toán tử AND cho phép tồn tại nhiều điều kiện trong mệnh đề WHERE của câu lệnh SQL.
3	**ANY** Toán tử ANY được sử dụng để so sánh một giá trị với bất kỳ giá trị thích hợp nào trong danh sách theo điều kiện.
4	**BETWEEN** Toán tử BETWEEN được sử dụng để tìm kiếm các giá trị trong một tập các giá trị, với giá trị nhỏ nhất và giá trị lớn nhất.
5	**EXISTS** Toán tử EXISTS được sử dụng để tìm kiếm sự tồn tại của một hàng trong một bảng được chỉ định đáp ứng một tiêu chí nhất định.
6	**IN** Toán tử IN được sử dụng để so sánh một giá trị với một danh sách các giá trị văn bản đã được chỉ định.

7	**LIKE** Toán tử LIKE được sử dụng để so sánh một giá trị với các giá trị tương tự sử dụng toán tử ký tự đại diện.
8	**NOT** Toán tử NOT đảo ngược ý nghĩa của toán tử logic mà nó được sử dụng. Ví dụ: NOT EXISTS, NOT BETWEEN, NOT IN, vv **Đây là một toán tử phủ định.**
9	**OR** Toán tử OR được sử dụng để kết hợp nhiều điều kiện trong mệnh đề WHERE của câu lệnh SQL.
10	**IS NULL** Toán tử NULL được sử dụng để so sánh một giá trị với một giá trị NULL.
11	**UNIQUE** Toán tử UNIQUE tìm kiếm tất cả các hàng của một bảng quy định cho tính duy nhất (không có bản sao)

3. Bài tập:

BAITAP1: Hiện thị thông tin: Bảng chi tiết chứng từ với những vật tư có Quantity không phải là số nguyên.

BAITAP2: Cho một chuỗi họ và tên: Lấy ra tên đệm và tên.

BAITAP3: Cho chuỗi @_A cắt ký tự chuỗi @_A để hiện thị dữ liệu theo từng dòng

DECLARE @_A NVARCHAR (32); SELECT @_A = N'Bố, Mẹ, Anh, Chị' -- cắt ký tự xuống dòng (Không dùng vòng lặp)

BAITAP4: Hiển thị danh mục vật tư mà có tên vật tư trong chuỗi danh sách @_List_Ten_VT ='Máy,Đồng,Tôn'

BAITAP5: Hiển thị ký tự đầu viết hoa và ký tự khác viết thường trong chuỗi tên có 3 từ 'ninH ngQc hiếU'

(*) BAITAP6: Hiển thị ký tự đầu tiên viết hoa và ký tự khác viết thường trong chuỗi tên ‘Hải hòa vinH lê nguyên quang’

IV. ĐỊNH NGHĨA BIẾN TRONG SQL VÀ CÂU LỆNH EXECUTE (‘Chuỗi’)

1. Biến trong SQL

- Để khai báo biến trong SQL Server, ta sử dụng câu lệnh DECLARE, cú pháp như sau:

  DECLARE @<Tên biến> <Kiểu dữ liệu của biến> [= Giá trị gán cho biến]

  + <Tên biến>: Tên của biến. Đối với Bravo thì yêu cầu có dấu gạch ‘_’ trước tên của biến.
  + <Kiểu dữ liệu>: Kiểu dữ liệu định nghĩa cho biến.
  + [= Giá trị gán cho biến]: Gán giá trị cho biến (không bắt buộc)

- Ví dụ: Để khai báo biến ItemCode thì khai báo như sau:

```
DECLARE @_ItemCode NVARCHAR(16)
```

2. Câu lệnh EXECUTE('Chuỗi')

- Cấu trúc: `EXECUTE ('SELECT * FROM Item')`
- Nguyên tắc biến câu lệnh Query SQL thành chuỗi trong câu lệnh EXECUTE():
  - Thêm 2 dấu nháy đơn vào đằng trước và đằng sau câu lệnh.
  - Với mỗi dấu nháy đơn ở câu lệnh Query SQL thông thường khi vào chuỗi để sử dụng trong Câu lệnh EXECUTE thì sẽ thêm 1 dấu nháy nữa.

    Ví dụ:

    ```
    SELECT * FROM Item WHERE ItemCode = 'TP01'
    ```

    Nếu đặt trong EXECUTE() thì viết thành:

    ```
    EXEC('SELECT * FROM Item WHERE ItemCode = ''TP01''')
    ```

- Trường hợp muốn cộng ký tự đặc biệt vào chuỗi thì dung hàm CHAR(Mã KeyAscii).

  Ví dụ:

  CHAR(13) → Ký tự Enter

  CHAR(39) → ký tự nháy đơn

V. HÀM NGÀY THÁNG

1. Hàm ngày tháng

- GETDATE(): Trả về ngày giờ hiện hành
- GETUTCDATE(): trả về ngày và giờ UTC hiện tại
- CURRENT_TIMESTAMP: Trả về ngày giờ hiện hành
- DATEADD(*interval*, *number*, *date*): trả về một ngày mà sau đó một khoảng thời gian/ngày nhất định đã được thêm vào.

- interval**:** Nó có thể là một trong những giá trị sau:

Giá trị	Giải thích
year, yyyy, yy	Khoảng thời gian năm
quarter, qq, q	Khoảng thời gian quý
month, mm, m	Khoảng thời gian tháng
dayofyear	Ngày trong năm
day, dy, y	Khoảng thời gian trong ngày
week, ww, wk	Khoảng thời gian trong tuần
weekday, dw, w	Khoảng thời gian các ngày trong tuần
hour, hh	Khoảng thời gian giờ

minute, mi, n	Khoảng thời gian phút
second, ss, s	Khoảng thời gian giây
millisecond, ms	Khoảng thời gian micro giây

- number**:** Số lượng khoảng thời gian muốn thêm
- date: Ngày mà khoảng thời gian được thêm vào
- Ví dụ:

```
SELECT DATEADD(year, 3, '2022/04/06')
--Ket qua: '2025-06-04 00:00:00.000'

SELECT DATEADD(yyyy, 4, '2022/04/06')
--Ket qua: '2026-06-04 00:00:00.000

SELECT DATEADD(yy, 9, '2022/04/06');
--Ket qua: '2031-06-04 00:00:00.000'

SELECT DATEADD(year, -2, '2022/04/06');
--Ket qua: '2020-06-04 00:00:00.000'

SELECT DATEADD(month, 8, '2022/04/06');
--Ket qua: '2023-02-04 00:00:00.000'

SELECT DATEADD(month, -6, '2022/04/06');
--Ket qua: '2021-12-04 00:00:00.000'

SELECT DATEADD(day, 4, '2022/04/06');
--Ket qua: '2022-06-08 00:00:00.000'

SELECT DATEADD(day, -5, '2022/04/06');
--Ket qua: '2022-05-30 00:00:00.000'
```

❖ DATEDIFF(interval, date1, date2): trả về chênh lệch giữa hai giá trị ngày, dựa trên khoảng thời gian được chỉ định.

- interval: Khoảng thời gian sử dụng để tính chênh lệch giữa *date1* và *date2*. Nó có thể là một trong những giá trị sau.

Giá trị	**Giải thích**
year, yyyy, yy	Khoảng thời gian năm
quarter, qq, q	Khoảng thời gian quý
month, mm, m	Khoảng thời gian tháng
dayofyear	Ngày trong năm
day, dy, y	Khoảng thời gian ngày
week, ww, wk	Khoảng thời gian tuần

weekday, dw, w	Khoảng thời gian ngày trong tuần
hour, hh	Khoảng thời gian giờ
minute, mi, n	Khoảng thời gian phút
second, ss, s	Khoảng thời gian giây
millisecond, ms	Khoảng thời gian micro giây

- date1, date2: Hai ngày để tính chênh lệch
- *Ví dụ:*

```
SELECT DATEDIFF(year, '2022/04/06', '2018/09/25')
--Ket qua: -4

SELECT DATEDIFF(yyyy, '2022/04/06', '2025/05/20')
--Ket qua: 3

SELECT DATEDIFF(yy, '2022/04/06', '2033/03/24')
--Ket qua: 11

SELECT DATEDIFF(month, '2022/04/06', '2022/02/20')
--Ket qua: -2

SELECT DATEDIFF(day, '2022/04/06', '2022/04/18')
--Ket qua: 12

SELECT DATEDIFF(hour, '2022/04/06 05:00', '2022/09/29 12:40')
--Ket qua: 4231

SELECT DATEDIFF(minute, '2022/04/06 05:00', '2022/01/25 15:45')
--Ket qua: -101595
```

❖ DATEPART(interval, date): trả về một phần của một ngày nhất định, dưới dạng một giá trị nguyên.

- interval: Khoảng thời gian/ngày muốn lấy từ date. Nó có thể là một trong những giá trị sau

Giá trị	**Giải thích**
year, yyyy, yy	Khoảng thời gian năm
quarter, qq, q	Khoảng thời gian quý
month, mm, m	Khoảng thời gian tháng
dayofyear	Ngày trong năm
day, dy, y	Khoảng thời gian ngày
week, ww, wk	Khoảng thời gian tuần

weekday, dw, w	Khoảng thời gian ngày trong tuần
hour, hh	Khoảng thời gian giờ
minute, mi, n	Khoảng thời gian phút
second, ss, s	Khoảng thời gian giây
millisecond, ms	Khoảng thời gian micro giây

- date: Ngày sử dụng để lấy giá trị interval.

- Lưu ý: Hàm DATEPART trả về kết quả dưới dạng giá trị nguyên

- Ví dụ:

```
SELECT DATEPART(year, '2022/04/06')
--Ket qua:  2022

SELECT DATEPART(yyyy, '2022/04/06')
--Ket qua:  2022

SELECT DATEPART(yy, '2022/04/06')
--Ket qua: 2022

SELECT DATEPART(month, '2022/04/06')
--Ket qua: 4

SELECT DATEPART(day, '2022/04/06')
--Ket qua: 6

SELECT DATEPART(quarter, '2022/04/06')
--Ket qua: 2

SELECT DATEPART(hour, '2022/04/06 05:40')
--Ket qua: 5

SELECT DATEPART(minute, '2022/04/06 09:43')
--Ket qua: 43

SELECT DATEPART(second, '2022/04/06 05:45:12')
--Ket qua: 12

SELECT DATEPART(millisecond, '2022/04/06 08:45:12.726')
--Ket qua: 726
```

❖ DATENAME(interval, date): trả về một phần của một ngày nhất định, dưới dạng giá trị chuỗi

- interval: Khoảng thời gian/ngày mà bạn muốn lấy từ *date*. Nó có thể là một trong những giá trị sau:

Giá trị	**Giải thích**
year, yyyy, yy	Khoảng thời gian năm

quarter, qq, q	Khoảng thời gian quý
month, mm, m	Khoảng thời gian tháng
dayofyear	Ngày trong năm
day, dy, y	Khoảng thời gian ngày
week, ww, wk	Khoảng thời gian tuần
weekday, dw, w	Khoảng thời gian ngày trong tuần
hour, hh	Khoảng thời gian giờ
minute, mi, n	Khoảng thời gian phút
second, ss, s	Khoảng thời gian giây
millisecond, ms	Khoảng thời gian micro giây

- date: Ngày sử dụng để lấy giá trị *interval*.

- Lưu ý: Hàm DATENAME trả về kết quả dưới dạng giá trị chuỗi

- Ví dụ:

```
SELECT DATENAME(year, '2022/04/06')
--Ket qua: '2022'

SELECT DATENAME(yyyy, '2022/04/06')
--Ket qua: '2022'

SELECT DATENAME(yy, '2022/04/06')
--Ket qua: '2022'

SELECT DATENAME(month, '2022/04/06')
--Ket qua: 'April'

SELECT DATENAME(day, '2022/04/06')
--Ket qua: '6'

SELECT DATENAME(quarter, '2022/04/06')
--Ket qua: '2'

SELECT DATENAME(hour, '2022/04/06 05:40')
--Ket qua: '5'

SELECT DATENAME(minute, '2022/04/06 09:43')
--Ket qua: '43'

SELECT DATENAME(second, '2022/04/06 05:45:12')
--Ket qua: '12'

SELECT DATENAME(millisecond, '2022/04/06 08:45:12.726')
```

```
--Ket qua: '726'
```

❖ DAY(date_value): trả về ngày trong tháng (một số từ 1 đến 31) cho một giá trị ngày

- date_value: Ngày để trích xuất ngày trong tháng
- Ví dụ:

```
SELECT DAY('2022/04/06')
--Ket qua: 6

SELECT DAY('2022/04/15 15:45')
--Ket qua: 15

SELECT DAY('2022/04/20 12:01:18.665')
--Ket qua: 20
```

❖ MONTH(date_value): trả về tháng (một số từ 1 đến 12) cho một giá trị ngày

- date_value: Ngày để trích xuất tháng
- Ví dụ:

```
SELECT MONTH('2022/04/06')
-- Ket qua: 4

SELECT MONTH('2022/03/06 15:05')
-- Ket qua: 3

SELECT MONTH('2022/12/06 15:05:15.61')
-- Ket qua: 12
```

❖ YEAR(date_value): trả về một năm có bốn chữ số (dưới dạng số) với giá trị ngày

- date_value: Ngày để trích xuất năm có bốn chữ số
- Ví dụ:

```
SELECT YEAR('2022/04/06')
-- Ket qua: 2022

SELECT YEAR('2019/03/06 15:05')
-- Ket qua: 2019

SELECT YEAR('2018/12/06 15:05:15.61')
-- Ket qua: 2018
```

2. Bài tập

BAITAP1: Hiện thị các chứng từ phiếu xuất có phát sinh từ ngày hiện tại đến 180 ngày trở về trước gồm: DocNo, DocDate, DocGroup, Description.

BAITAP2: Hiện thị các phiếu được bán vào thứ 4 ngày 5.

BAITAP3: Sử dụng các hàm ngày tháng để hiển thị thông tin của bảng chứng gồm: DocCode, DocNo, DocDate, Thang, Quy, Nam, DocGroup, Description

BAITAP4: Cho một ngày bất kỳ.

- Lấy ra ngày đầu tháng và ngày cuối tháng
- Lấy ngày đầu tháng cùng kỳ năm trước và ngày cuối tháng cùng kỳ năm trước

VI. RÀNG BUỘC, CHỈ MỤC, HÀM VÀ CÁC LỆNH HỖ TRỢ TRUY VẤN

1. Ràng buộc

1.1. Tạo Primary Key

Khóa chính là tập hợp một hoặc nhiều column giúp phân biệt các record trong một table, đây là thông tin rất quan trọng bởi nếu thiếu nó thì lược đồ CSDL sẽ không hình thành.

Ví dụ bảng Item thì thường ta sẽ có column **ItemCode** dùng để nhận diện các **Code, ItemName** có thể bị trùng nhưng mã **ItemCode** thì không thể trùng.

Đặc điểm của khóa chính như sau:

- Có thể thiết lập khóa chính bằng một hoặc nhiều column, trong thực tế thì nên 1 column
- Khóa chính không được NULL, và là duy nhất (*unique*)
- Khóa chính nếu là kiểu số nguyên thì nên thiết lập tăng tự động sẽ giúp tối ưu database

Chúng ta có hai cách tạo primary key, thứ nhất là tạo trực tiếp ở lệnh CREATE TABLE và thứ hai là sử dụng lệnh ALTER TABLE

❖ **Tạo trực tiếp ở lệnh CREATE TABLE**

Cách này có hai cú pháp như sau

Cú pháp 1:

```
CREATE TABLE table_name (column_1 data_type  PRIMARY KEY, ...)
```

Cú pháp 2:

```
CREATE TABLE table_name (column_1 data_type, column_2 data type, ...
        PRIMARY KEY (column_1, column_2))
```

Ví dụ: Bảng Item 1 có khóa chính là ItemCode thì khi tạo bảng như sau:

```
CREATE TABLE Item (
        ItemCode NVARCHAR(16) NOT NULL DEFAULT('') PRIMARY KEY,
        ItemName NVARCHAR(96) NOT NULL DEFAULT(''),
        Unit NVARCHAR(10) NOT NULL DEFAULT(''),
        ItemType INT  NOT NULL DEFAULT((0)),
        IsActive INT NOT NULL DEFAULT((1)))
```

❖ **Tạo bằng lệnh ALTER TABLE**

Cú pháp: ALTER TABLE Table_Name ADD PRIMARY KEY(Clumn_1, Clumn_2,...)

Ví dụ: ALTER TABLE Item ADD PRIMARY KEY(ItemCode)

1.2. Khóa ngoại Foreign Key

- Khóa ngoại hay còn gọi là **Foreign Key**, đây là mối liên kết giữa hai bảng với nhau tạo thành một lược đồ cơ sở dữ liệu quan hệ.
- Giả sử ta có hai bảng: **AccDoc** và **AccDocDetial**, bảng **AccDoc** lưu trữ danh sách các chứng từ, bảng **AccDocDetail** lưu danh sách chi tiết của chứng từ. Dẫn đến có mối liên hệ chi tiết chứng từ trong **AccDocDetail** thuộc một chứng từ nào **AccDoc**.
- Để thể hiện mối liên hệ này thì trong bảng **AccDocDetail** sẽ có một column **DocNo** trỏ đến khóa chính **DocNo** của bảng **AccDoc**, ta gọi đây là **Foreign Key**
- Một số lưu ý của khóa ngoại:
  - Bảng A có khóa ngoại trỏ đến bảng B thì ta gọi A là bảng cha, còn B là bảng con (Trong ví dụ của bài tập thì **AccDoc** là bảng cha, **AccDocDetail** là bảng con)
  - Giá trị của khóa ngoại của bảng con phải tồn tại trong các giá trị khóa chính của bảng cha, đây ta gọi là ràng buộc toàn vẹn.

Tương tự như tạo khóa chính, có hai cách để tạo khóa ngoại

❖ **Tạo trực tiếp ở lệnh CREATE TABLE**

Sử dụng từ khóa CONSTRAINT ngay ở phía cuối danh sách column. Cú pháp:

```
CREATE TABLE Table_Name (Column_1 data_type, Column_2 data_type, ...,
    CONSTRAINT Foreign_Key_Name
    FOREIGN KEY (Column_1, Column2,...)
    REFERENCES Parent_Table_Name(Column1,Column2,..))
```

Ví dụ:

```
CREATE TABLE AccDocDetail (
            DocCode CHAR(2) NOT NULL DEFAULT(''),
            DocNo NVARCHAR(10) NOT NULL DEFAULT(''),
            ItemCode NVARCHAR(16) NOT NULL DEFAULT(''),
            Quantity NUMERIC (15,3) NOT NULL DEFAULT((0)),
            UnitCost NUMERIC (15,5) NOT NULL DEFAULT((0)),
            Amount1 NUMERIC (18,2) NOT NULL DEFAULT((0)),
            UnitPrice NUMERIC (15,5) NOT NULL DEFAULT((0)),
            Amount2 NUMERIC (18,2) NOT NULL DEFAULT((0)),
    CONSTRAINT PK_AccDocDetail_DocNo
    FOREIGN KEY (DocNo) REFERENCES AccDoc(DocNo))
```

❖ **Tạo bằng lệnh ALTER TABLE**

Cú pháp:

```
ALTER TABLE Table_Name ADD CONSTRAINT Foreign_Key_Name
    FOREIGN KEY (Column_Name) REFERENCES Parent_Table_Name(ColumnName))
```

Ví dụ:

```
ALTER TABLE AccDocDetail ADD CONSTRAINT PK_AccDocDetail_DocNo
    FOREIGN KEY (DocNo) REFERENCES AccDoc(DocNo)
```

❖ **Xóa khóa ngoại FOREIGN KEY**

Cú pháp:

```
ALTER TABLE Table_Name DROP CONSTRAINT Foreign_Key_Name
```

Ví dụ:

```
ALTER TABLE AccDocDetail DROP CONSTRAINT PK_AccDocDetail_DocNo
```

1.3. Ràng buộc UNIQUE

UNIQUE là ràng buộc trên giá trị duy nhất trên column, có nghĩa là các dòng dữ liệu không được có giá trị trùng nhau ở column đó. Ràng buộc này giống như khóa chính vậy, vì bản chất nó cũng là một khóa.

Ví dụ trong bảng Customer sẽ có CustomerCode là khóa chính, nếu thêm cột số điện thoại, email sẽ là UNIQUE vì số điện thoại, email sẽ không trùng nhau. Thực ra cũng có thể lấy số điện thoại là khóa chính nhưng như vậy sẽ không hay lắm.

Tương tự PRIMARY KEY, UNIQUE cũng có thể tạo lúc tạo bảng hoặc có thể bổ sung bằng ALTER TABLE. Cú pháp của các hai trường hợp như sau:

❖ **Tạo từ CREATE TABLE**

```
CREATE TABLE Table_Name (Column_1 data_type PRIMARY KEY,
Column_2 data_type, Column_3 data_type,…,
CONSTRAINT Unique_Name UNIQUE(Column_2, Column3)
```

Ví dụ: Bảng Customer

```
CREATE TABLE Customer (CustomerCode NVARCHAR(16) NOT NULL DEFAULT('') PRIMARY
KEY,
    CustomerName NVARCHAR(96) NOT NULL DEFAULT(''),
    CustomerType INT NOT NULL DEFAULT((0)),
```

```
    Tel NVARCHAR(16) NOT NULL DEFAULT(''),
    Email NVARCHAR(56) NOT NULL DEFAULT(''),
    IsActive INT  NOT NULL DEFAULT((1)),
CONSTRAINT Unique_Tel_Email UNIQUE(Tel, Email))
```

❖ **Tạo từ ALTER TABLE**

```
ALTER TABLE Table_Name ADD CONSTRAINT Unique_Name UNIQUE(Column_1, Column_2…)
```

Ví dụ:

```
ALTER TABLE Customer ADD CONSTRAINT Unique_Tel_Email UNIQUE(Tel, Email)
```

2. Chỉ mục INDEX

Các INDEX được sử dụng nhằm hỗ trợ việc truy cập đến các dòng dữ liệu được nhanh chóng dựa trên các giá trị của một hay nhiều cột. INDEX được chia thành hai loại: INDEX tụ nhóm (CLUSTERED) và INDEX không tụ nhóm (NONCLUSTERED)

- Một INDEX tụ nhóm là chỉ mục mà trong đó thứ tự logic của các khóa tương tự như các thứ tự vật lý của các dòng tương ứng tồn tại trong bảng. Một bảng chỉ có thể tối đa một chỉ mục tụ nhóm.
- Một INDEX không tụ nhóm là chỉ mục mà trong đó thứ tự logic của các khóa không như thứ tự vật lý của các dòng tương ứng tồn tại trong bảng.

⇨ Các chỉ mục tụ nhóm hỗ trợ việc truy cập đến các dòng dữ liệu nhanh hơn nhiều so với các chỉ mục không tụ nhóm.

⇨ Khi ta khai báo một khóa chính hay khóa UNIQUE trên một hay nhiều cột nào đó của bảng thì SQL sẽ tự động tạo INDEX trên các cột đó. Ngoài ra có thể tạo thêm INDEX khác bằng cách sử dụng câu lệnh có cú pháp sau:

```
CREATE [CLUSTERED| NONCLUSTERED] INDEX Index_Name ON Table_Name (Column_1
ASC|DESC, [Column_2 ASC|DESC, …])
```

Ví dụ: muốn tạo INDEX cho trường CustomerCode trong bảng AccDoc

```
CREATE NONCLUSTERED INDEX IX_AccDoc  ON AccDoc (CustomerCode)
```

- INDEX giúp tăng tốc các truy vấn SELECT chứa các mệnh đề WHERE hoặc ORDER, nhưng nó làm chậm việc dữ liệu nhập vào với các lệnh UPDATE và INSERT

Đối với các đơn vị có dữ liệu lớn thì việc quyết định sử dụng INDEX như thế nào sẽ có một bài học chuyên sâu hơn trong tài liệu hướng dẫn viết mã nguồn SQL của BRAVO. Sẽ được đề cập từ tháng thứ ba đối với nhân viên học việc.

3. Các hàm cơ bản thường dùng trong câu lệnh truy vấn

❖ **Hàm COUNT:** trả về số lượng của một biểu thức

Cú pháp:

```
SELECT COUNT(aggregate_expression)
    FROM Table_Nam
    WHERE Conditions
```

Hoặc:

```
SELECT expression1, expression2, ... expression_n, COUNT(aggregate_expression)
    FROM Table_Name
    WHERE Conditions
    GROUP BY expression1, expression2, ... expression_n
```

❖ **Hàm SUM:** trả về giá trị tổng của một biểu thức

Cú pháp:

```
SELECT SUM(Aggregate_Expression)
        FROM Table_Name
        WHERE Conditions
```

Hoặc:

```
SELECT expression1, expression2, ... expression_n, SUM(aggregate_expression)
        FROM Table_Name
        WHERE Conditions
        GROUP BY expression1, expression2, ... expression_n
```

❖ **Hàm MAX:** trả về giá trị lớn nhất của một biểu thức

Cú pháp:

```
SELECT MAX(Aggregate_Expression)
        FROM Table_Name
        WHERE Conditions
```

Hoặc:

```
SELECT expression1, expression2, ... expression_n, MAX(aggregate_expression)
        FROM Table_Name
        WHERE Conditions
```

❖ **Hàm MIN:** trả về giá trị lớn nhất của một biểu thức

Cú pháp:

```
SELECT MIN(Aggregate_Expression)
        FROM Table_Name
        WHERE Conditions
```

Hoặc:

```
SELECT expression1, expression2, ... expression_n, MIN(aggregate_expression)
        FROM Table_Name
        WHERE Conditions
```

❖ **Hàm AVG:** trả về giá trị trung bình của một biểu thức

Cú pháp:

```
SELECT AVG(Aggregate_Expression)
        FROM Table_Name
        WHERE Conditions
```

Hoặc:

```
SELECT expression1, expression2, ... expression_n, AVG(aggregate_expression)
        FROM Table_Name
        WHERE Conditions
```

4. Các chức năng chuyển đổi

❖ **Hàm CAST:** chuyển đổi một biểu thức từ một kiểu dữ liệu này sang kiểu dữ liệu khác. Nếu chuyển đổi không thành công, chức năng sẽ trả về một lỗi. Nếu không, nó sẽ trả về giá trị chuyển đổi.

```
CAST(Expression AS type [(Length)])
```

Trong đó:

- expression: Biểu thức để chuyển đổi sang kiểu dữ liệu khác.
- type: Kiểu dữ liệu mà bạn muốn chuyển đổi biểu thức thành
- length: Không bắt buộc. Độ dài của kiểu dữ liệu

Ví dụ:

```
SELECT CAST(14.85 AS int);
--Ket qua: 14
```

```
SELECT CAST(14.85 AS float);
--Ket qua: 14.85

SELECT CAST(15.6 AS varchar);
--Ket qua: '15.6'

SELECT CAST(15.6 AS varchar(4));
--Ket qua: '15.6'

SELECT CAST('15.6' AS float);
--Ket qua: 15.6

SELECT CAST('2019-04-06' AS datetime);

--Ket qua: '06.04.2019 00:00:00'
```

- **Hàm CONVERT:** Chuyển đổi một biểu thức từ một kiểu dữ liệu này sang kiểu dữ liệu khác. Nếu chuyển đổi không thành công, chức năng sẽ trả về một lỗi. Nếu không, nó sẽ trả về giá trị chuyển đổi.

  CONVERT(Type [(length)], Expression [, style])

  Ví dụ:

```
SELECT CONVERT(int, 14.85);
--Ket qua: 14

SELECT CONVERT(float, 14.85);
--Ket qua: 14.85

SELECT CONVERT(varchar, 15.6);
--Ket qua: '15.6'

SELECT CONVERT(varchar(4), 15.6);
--Ket qua: '15.6'

SELECT CONVERT(float, '15.6');
--Ket qua: 15.6

SELECT CONVERT(datetime, '2019-04-06');

--Ket qua: '06.04.2019 00:00:00
```

5. Câu lệnh CASE

Câu lệnh CASE có chức năng của câu lệnh IF-THEN-ELSE

```
CASE expression
  WHEN value_1 THEN result_1
  WHEN value_2 THEN result_2
  ...
  WHEN value_n THEN result_n
    ELSE result
END
```

Hoặc:

```
CASE
    WHEN condition_1 THEN result_1
    WHEN condition_2 THEN result_2
    ...
    WHEN condition_n THEN result_n
ELSE result
END
```

Trong đó:

- expression: Biểu thức sẽ được so sánh với từng giá trị được cung cấp. (ví dụ: value_1, value_2, ... value_n).
- value_1, value_2, ... value_n: Các giá trị sẽ được sử dụng trong đánh giá. Các giá trị được đánh giá theo thứ tự được liệt kê. Khi một giá trị khớp với biểu thức, câu lệnh CASE sẽ thực thi các câu lệnh tương ứng và không đánh giá thêm nữa.
- condition_1, condition_2, ... condition_n: Các điều kiện sẽ được đánh giá. Các điều kiện được đánh giá theo thứ tự được liệt kê. Khi một điều kiện được xác định là đúng, câu lệnh CASE sẽ trả về kết quả và không đánh giá các điều kiện nữa. Tất cả các điều kiện phải là cùng một kiểu dữ liệu.
- result_1, result_2, ... result_n: Giá trị được trả về sau khi một điều kiện được tìm thấy là đúng. Tất cả các giá trị phải là cùng một kiểu dữ liệu.

Lưu ý:

- Nếu không tìm thấy giá trị / điều kiện nào là TRUE, thì câu lệnh CASE sẽ trả về giá trị trong mệnh đề ELSE.
- Nếu mệnh đề ELSE bị bỏ qua và không có điều kiện nào được tìm thấy là đúng, thì câu lệnh CASE sẽ trả về NULL.
- Điều kiện được đánh giá theo thứ tự được liệt kê. Khi một điều kiện được xác định là đúng, câu lệnh CASE sẽ trả về kết quả và không đánh giá các điều kiện nữa

Ví dụ: Cần lấy ra kết quả nhập mua, nhập thành phẩm, xuất sản xuất, xuất bán của các vật tư.

```
SELECT DocCode, ItemCode,
    CASE WHEN DocCode = 'NM' THEN SUM(Quantity)
            ELSE CAST(0 AS NUMERIC(15,3)) END AS Nhap_Mua,
    CASE WHEN DocCode = 'PX' THEN SUM(Quantity)
            ELSE CAST(0 AS NUMERIC(15,3)) END AS Xuat_Sx,
    CASE WHEN DocCode = 'TP' THEN SUM(Quantity)
            ELSE CAST(0 AS NUMERIC(15,3)) END AS Nhap_Tp,
    CASE WHEN DocCode = 'HD' THEN SUM(Quantity)
            ELSE CAST(0 AS NUMERIC(15,3)) END AS Xuat_Ban
FROM AccDocDetail
GROUP BY DocCode, ItemCode
ORDER BY DocCode, ItemCode
```

Kết quả:

	DocCode	ItemCode	Nhap_Mua	Xuat_Sx	Nhap_Tp	Xuat_Ban
1	HD	TP01	0.000	0.000	0.000	380.000
2	HD	TP02	0.000	0.000	0.000	940.000
3	HD	TP03	0.000	0.000	0.000	1180.000
4	NM	NVLC01	300.000	0.000	0.000	0.000
5	NM	NVLC02	1600.280	0.000	0.000	0.000
6	NM	NVLC03	2350.000	0.000	0.000	0.000
7	PX	NVLC01	0.000	330.000	0.000	0.000
8	PX	NVLC02	0.000	1170.330	0.000	0.000
9	PX	NVLC03	0.000	1770.000	0.000	0.000
10	TP	TP01	0.000	0.000	700.000	0.000
11	TP	TP02	0.000	0.000	700.000	0.000
12	TP	TP03	0.000	0.000	900.000	0.000

6. Lệnh BEGIN – END

Lệnh BEGIN…END dùng để khai báo một khối lệnh SQL. Một khối lệnh là tập hợp những câu SQL sẽ được thực thi chung với nhau.

```
BEGIN
    { sql_statement | statement_block}
```

```
END
```

Bên trong khối lệnh BEGIN…END sẽ là tập hợp một hoặc nhiều câu lệnh khác, điều này sẽ giúp tạo ra một chương trình có tính thẩm mỹ và rõ ràng.

```
BEGIN
    SELECT
        ItemCode, UnitPrice
    FROM
        AccDocDetail
      WHERE
        Unitprice > 20000000;

  IF @@ROWCOUNT=  0
    PRINT N'Không có sản phẩm nào tồn tại';
END
```

Có thể sử dụng BEGIN…END lồng nhau, nghĩa là khai báo cặp BEGIN…END bên trong một cặp khác.

```
BEGIN
  DECLARE @_ItemCode NVARCHAR(16);

  SELECT TOP 1 @_ItemCode = ItemCode
            FROM AccDocDetail
            ORDER BY UnitPrice DESC

  IF @@ROWCOUNT <> 0
  BEGIN
    PRINT N'Sản phẩm có giá bán cao nhất là ' + @_ItemCode
  END
  ELSE
  BEGIN
    PRINT N'Không tìm thấy sản phẩm';
  END;

END
```

7. Lệnh rẽ nhánh IF ELSE

Lệnh IF sẽ kiểm tra một biểu thức có đúng hay không, nếu đúng thì thực thi nội dung bên trong của IF, nếu sai thì bỏ qua.

```
IF Boolean_Expression
BEGIN
  { Statement_Block }
END
```

Trong đó:

- Nội dung bên trong BEGIN ... END chính là phần thân của lệnh IF.
- Boolean_Expression là biểu thức điều kiện, nếu giá trị biểu thức là TRUE thì phần thân sẽ được chạy, ngược lại thì bỏ qua.

Ví dụ:

```
BEGIN
  DECLARE @_Amount NUMERIC(18,2);

  SELECT @_Amount = SUM(Amount2)
            FROM AccDocDetail
    WHERE DocCode = 'HD'

  IF @_Amount > 20000000000
```

```
    BEGIN
        PRINT N'Doanh số > 20000000000. Hoàn thành vượt mức kế hoạch';
    END
END
```

Trường hợp điều kiện ở IF là false thì chương trình sẽ chạy ở phần ELSE

```
IF Boolean_Expression
BEGIN
    -- Statement block executes when the Boolean expression is TRUE
END
ELSE
BEGIN
    -- Statement block executes when the Boolean expression is FALSE
END
```

Mỗi lệnh IF đều có biểu thức điều kiện, tuy nhiên phần ELSE thì không cần vì nó là phần sẽ chạy nếu như phần IF không thỏa

Ví dụ:

```
BEGIN
    DECLARE @_Amount NUMERIC(18,2);

    SELECT @_Amount = SUM(Amount2)
                FROM AccDocDetail
    WHERE DocCode = 'HD'

    IF @_Amount > 20000000000
    BEGIN
        PRINT N'Doanh số > 20000000000. Hoàn thành vượt mức kế hoạch'
    END
    ELSE
    BEGIN
        PRINT N'Không hoàn thành kế hoạch';
    END
END
```

Lệnh IF ELSE lồng nhau: IF ELSE lồng nhau tức là lệnh IF này nằm bên trong một lệnh IF khác, các sử dụng này sẽ giúp chương trình rẻ rất nhiều nhánh khác nhau

```
BEGIN
    DECLARE @_a INT = 10, @_b INT = 20;

    IF (@_a > 0)
    BEGIN
        IF (@_a < @_b)
            PRINT 'a > 0 and a < b'
        ELSE
            PRINT 'a > 0 and a >= b';
    END

END
```

8. Kiểm tra sự tồn tại EXISTS

Điều kiện EXISTS trong SQL được sử dụng để kiểm tra sự tồn tại của bất kỳ bản ghi nào trong truy vấn phụ. Kết quả trả về của EXISTS là một giá trị boolean TRUE hoặc FALSE. Nó có thể được sử dụng trong câu lệnh IF, SELECT, UPDATE, INSERT hoặc DELETE…

Ví dụ 1:

```
-- Kiểm tra sản phẩm xem có được bán hàng không
IF EXISTS(SELECT TOP 1 ItemCode
      FROM AccDocDetail
      WHERE ItemCode = 'TP01' AND DocCode = 'HD')
```

```
    PRINT N'Sản phẩm có bán'
```

Lưu ý: Để tối ưu hóa hiệu năng xử lý thì lưu ý khi sử dụng EXISTS với IF:

- Nên sử dụng IF EXISTS(SELECT TOP 1 FROM Table WHERE….)
- Tránh sử dụng IF EXISTS(SELECT COUNT(*) FROM Table WHERE…) > 0
- Tránh sử dụng IF EXISTS(SELECT * FROM Table WHERE…)

Ví dụ 2:

```
-- Lấy ra danh sách khách hàng có mua ít nhất một đơn hàng
SELECT CustomerCode, CustomerName
    FROM Customer
    WHERE EXISTS (SELECT TOP 1 CustomerCode
                  FROM AccDoc
                  WHERE Customer.CustomerCode = AccDoc.CustomerCode
                        AND AccDoc.DocCode = 'HD')
```

9. Bài tập

BÀI TẬP 1: Thay thế 'NM' thành 'TP' của DocNo trong bảng AccDoc với DocCode = 'TP'

BÀI TẬP 2: Tạo khóa chính cho bảng Item, Customer, AccDoc

BÀI TẬP 3: Tạo khóa ngoại cho các bảng AccDocDetail trường DocNo, OpenInventory trường ItemCode

BÀI TẬP 4: Thêm trường TaxCode vào bảng Customer và tạo UNIQUE cho TaxCode

BÀI Tập 5: Tạo thêm INDEX cho bảng AccDoc trường CustomerCode, bảng AccDocDetail trường ItemCode

BÀI TẬP 6: Tìm mặt hàng có tổng số lượng bán lớn nhất

BÀI TẬP 7: Tìm mặt hàng có tổng số lần bán lớn nhất

BÀI TẬP 8: Tìm mặt hàng có tổng doanh số bán lớn nhất.

BÀI TẬP 9: Thể hiện tổng số tiền mua hàng và số tiền bán hàng của từng ngày gồm các cột: DocCode, DocDate, Tien_Mua, Tien_Ban.

BÀI TẬP 10: Cho bảng kế hoạch sản xuất (ProductionPlan) như sau:

ItemCode	Plan
TP01	1000
TP02	800
TP03	700

- Lấy ra danh sách sản phẩm và sản lượng hoàn thành kế hoạch sản xuất
- Lấy ra danh sách sản phẩm và sản lượng không hoàn thành kế hoạch sản xuất

VII. TRUY VẤN VÀ KẾT NỐI DỮ LIỆU

1. Truy vấn dữ liệu

Để truy xuất dữ liệu từ các dòng và các cột của một hay nhiều bảng, khung nhìn, ta sử dụng câu lệnh SELECT. Câu lệnh này có thể dùng để thực hiện phép chọn (tức là truy xuất một tập con các dòng trong một hay nhiều bảng), phép chiếu (tức là truy xuất một tập con các cột trong một hay nhiều bảng) và phép nối (tức là liên kết các dòng trong hai hay nhiều bảng để truy xuất dữ liệu).

Cú pháp chung của câu lệnh SELECT có dạng như sau:

```
SELECT  ALL | DISTINCT  Select_List
        INTO Newtable_Name
```

```
FROM Table_Name | View_Name
        .....
        [Table_name | view_name]
WHERE Clause
GROUP BY Clause
HAVING BY Clause
ORDER BY Clause
COMPUTE Clause
```

Chú ý: Các thành phần trong một câu lệnh SELECT phải được sử dụng theo thứ tự được nêu trên

1.1. Xác định bảng bằng mệnh đề FROM

Mệnh đề FROM trong câu lệnh SELECT được sử dụng nhằm chỉ định các bảng và khung nhìn cần truy xuất dữ liệu. Sau mệnh đề FROM là danh sách tên các bảng và khung nhìn tham gia vào truy vấn (tên của các bảng và khung nhìn được phân cách nhau bởi dấu phẩy)

```
SELECT Select_List
    FROM Table_name | View_Name List
```

Để đơn giản hoá câu hỏi, ta có thể sử dụng các bí danh (alias) cho các bảng hay khung nhìn. Bí danh được gán trong mệnh đề FROM bằng cách chỉ định bí danh sau tên bảng. Ví dụ câu lệnh sau gán bí danh Vt cho bảng Item

```
SELECT ItemCode, ItemName, Unit FROM Item Vt
```

1.2. Mệnh đề WHERE

Mệnh đề WHERE trong câu lệnh SELECT xác định các điều kiện đối với việc truy xuất dữ liệu. Sau mệnh đề WHERE là một biểu thức logic và chỉ những dòng dữ liệu nào thoả mãn biểu thức sau WHERE mới được hiển thị trong kết quả truy vấn. Trong mệnh đề WHERE thường sử dụng:

- Các toán tử so sánh
- Giới hạn (BETWEEN và NOT BETWEEN).
- Danh sách (IN, NOT IN)
- Khuôn dạng (LIKE và NOT LIKE).
- Các giá trị chưa biết (IS NULL và IS NOT NULL).
- Kết hợp các điều kiện (AND, OR)

Ví dụ: Truy vấn sau đây cho biết mã, tên và đơn vị tính của những vật tư là thành phẩm

```
SELECT ItemCode, ItemName, Unit
    FROM Item
    WHERE ItemType = 2
```

Giới hạn (BETWEEN và NOT BETWEEN)

Từ khoá BETWEEN và NOT BETWEEN được sử dụng nhằm chỉ định khoảng giá trị tìm kiếm đối với câu lệnh SELECT. Câu lệnh dưới đây cho biết các hóa đơn được bán ra trong khoảng thời gian từ ngày 01/01/2022 đến ngày 10/01/2022

```
SET DATEFORMAT DMY
    SELECT DocCode, DocDate, Description
    FROM AccDoc
    WHERE DocCode = 'HD' AND DocDate BETWEEN '01/01/2022' AND '10/01/2022'
```

Danh sách (IN và NOT IN)

Từ khoá IN được sử dụng khi ta cần chỉ định điều kiện tìm kiếm dữ liệu cho câu lệnh SELECT là một danh sách các giá trị. Sau IN (hoặc NOT IN) có thể là một danh sách các giá trị hoặc là một câu lệnh SELECT khác.

Ví dụ: Để hiển thị thông tin về các vật tư là nguyên vật liệu hoặc thành phẩm, thay vì sử dụng câu lệnh:

```
SELECT ItemCode, ItemName, Unit
    FROM Item
    WHERE ItemType = 1 OR ItemType = 2
```

Ta có thể sử dụng câu lệnh sau:

```
SELECT ItemCode, ItemName, Unit
    FROM Item
    WHERE ItemType IN (1,2)
```

Các ký tự đại diện và mệnh đề LIKE

Từ khoá LIKE (NOT LIKE) sử dụng trong câu lệnh SELECT nhằm mô tả khuôn dạng của dữ liệu cần tìm kiếm. Chúng thường được kết hợp với các ký tự đại diện sau đây:

Ví dụ: Câu lệnh dưới đây hiển thị thông tin về các vật tư có tên là tủ đông.

```
SELECT ItemCode, ItemName, Unit
    FROM Item
    WHERE ItemName LIKE N'%Tủ đông%'
```

IS NULL và NOT IS NULL

Giá trị NULL có thể được nhập vào một cột cho phép chấp nhận giá trị NULL theo một trong ba cách sau:

- Nếu không có dữ liệu được đưa vào và không có mặc định cho cột hay kiểu dữ liệu trên cột đó.
- Người sử dụng trực tiếp đưa giá trị NULL vào cho cột đó.
- Một cột có kiểu dữ liệu là kiểu số sẽ chứa giá trị NULL nếu giá trị được chỉ định gây tràn số.

Trong mệnh đề WHERE, ta sử dụng IS NULL hoặc IS NOT NULL như sau:

```
WHERE Column_Name IS [NOT] NULL
```

Các toán tử logic

Các toán tử logic sử dụng trong mệnh đề WHERE bao gồm AND, OR, NOT. AND và OR được sử dụng để kết hợp nhiều điều kiện trong WHERE.

1.3. Danh sách chọn trong câu lệnh SELECT

Chọn tất cả các cột trong bảng

Khi muốn truy xuất tất cả các cột trong bảng, ta sử dụng câu lệnh SELECT có cú pháp sau:

```
SELECT * FROM Table_Name
```

Khi sử dụng câu lệnh này, các cột trong kết quả sẽ được hiển thị theo thứ tự mà chúng đã được tạo ra trong câu lệnh CREATE TABLE.

*Lưu ý: Tại BRAVO không cho phép viết lệnh kiểu này mà cần luôn liệt kê danh sách trường cần lấy trong câu lệnh SELECT. Kể cả liệt kê tất cả các cột trong bảng cũng đều phải list danh sách trường. Vì code là viết thì người khác còn phải đọc để bảo hành phần mềm, nếu chúng ta để * thì người đọc Code sẽ rất khó hình dung ý đồ trong câu lệnh.*

Chọn các cột được chỉ định

Để chọn ra một số cột nào đó trong bảng, ta sử dụng câu lệnh SELECT có cú pháp sau:

```
SELECT Column_1 [,...,Clumn_n]
    FROM Table_Name | View_Name
```

Các tên cột trong câu lệnh phải được phân cách nhau bằng dấu phẩy.

Chú ý: Trong câu lệnh SELECT, thứ tự của các cột được nêu ra trong câu lệnh sẽ xác định thứ tự của các cột được hiển thị ra trong kết quả.

Đổi tên các cột trong các kết quả

Khi kết quả được hiển thị, tiêu đề của các cột mặc định sẽ là tên của các cột khi nó được tạo ra trong câu lệnh CREATE TABLE. Tuy nhiên, để các tiêu đề trở nên thân thiện hơn, ta có thể đổi tên các tiêu đề của các cột. Để làm được việc này, ta có thể sử dụng một trong hai cách viết sau:

```
Tiêu_đề_cột = Tên_cột hoặc Tên_cột Tiêu_đề_cột
```

Ví dụ: Hai câu lệnh sau sẽ đặt tiêu đề Mã vật tư cho ItemCode và Tên vật tư cho ItemName khi kết quả được hiển thị cho người sử dụng:

```
SELECT N'Mã vật tư' = ItemCode, N'Tên vật tư'= ItemName
```

```
    FROM Item
```

Hoặc:

```
SELECT ItemCode N'Mã vật tư', ItemName N'Tên vật tư'
    FROM Item
```

Sử dụng cấu trúc CASE để thay đổi dữ liệu trong kết quả

Trong câu lệnh SELECT, ta có thể sử dụng cấu trúc CASE để thay đổi cách hiển thị kết quả ra màn hình.

Ví dụ: Câu lệnh sau cho biết mã vật tư, tên vật tư, loại vật tư:

```
SELECT N'Mã vật tư' = ItemCode, N'Tên vật tư' = ItemName,
            N'Loại vật tư' =
        CASE WHEN ItemType = 0 THEN N'Dịch vụ'
            WHEN ItemType = 1 THEN N'Vật tư hàng hóa'
            WHEN ItemType = 2 THEN N'Thành phẩm'
        ELSE N'Loại khác'
        END
    FROM Item
```

1.4. Tính toán giá trị trong câu lệnh SELECT

Danh sách chọn trong câu lệnh SELECT có thể có các biểu thức số học. Khi đó kết quả của biểu thức sẽ là một cột trong kết quả truy vấn
Ví dụ: Câu lệnh sau cho biết tiền thuế VAT 10% của các mặt hàng bán ra:

```
SELECT N'Mã chứng từ' = DocCode,
        N'Số chứng từ' = DocNo,
        N'Mã vật tư' = ItemCode,
        N'Tiền bán hàng' = Amount2,
        N'Tiền thuế VAT 10%' = CAST(ROUND(Amount2*10/100,0) AS NUMERIC(18,2))
    FROM AccDocDetail
    WHERE DocCode = 'HD'
```

1.5. Từ khóa DISTINCT

Từ khoá DISTINCT được sử dụng trong câu lệnh SELECT nhằm loại bỏ ra khỏi kết quả truy vấn những dòng dữ liệu có giá trị giống nhau.
Ví dụ: Nếu muốn lấy xem những mặt hàng nào được bán trong tháng 1 năm 2022, ta làm như sau:

```
SELECT DISTINCT ItemCode
    FROM AccDocDetail
    WHERE DocCode = 'HD'
```

1.6. Tạo bảng mới bằng câu lệnh SELECT ... INTO

Trong quá trình thực hiện truy vấn ta có thể đổ dữ liệu truy vấn vào bảng tạm để có thể tiếp tục dùng kết quả xử lý cho những yêu cầu khác.

❖ **Bảng tạm**

- Là bảng có tên có ký tự # ở đầu, ví dụ: #Item, #Customer
- Bảng tạm có 2 loại: Bảng tạm 1 # và bảng tạm 2 # (##)
  - Bảng tạm có 1 ký tự # ở đầu là bảng được tạo ra khi chạy câu lệnh CREATE TABLE và mất đi khi SESSION mất đi
  - Bảng tạm có 2 ký tự # là bảng tạm tạo ra khi chạy câu lệnh CREATE TABLE và mất đi khi bị DROP TABLE
  - Cách viết câu lệnh của bảng tạm cũng giống như với bảng thường tuy nhiên trước khi tạo bảng cần viết câu lệnh kiểm tra tồn tại của bảng để nếu có thì xóa đi:

    ```
    IF OBJECT_ID('tempdb..#TenBangTam') IS NOT NULL DROP TABLE #TenBangTam
    ```

- Ứng dụng của bảng tạm dùng rất nhiều trong thực tế giúp việc xử lý trung gian dữ liệu được thuận lợi hơn

❖ **Bảng tạm trong câu lệnh SELECT… INTO**

Câu lệnh SELECT ... INTO có tác dụng tạo một bảng mới có cấu trúc và dữ liệu được xác định từ kết quả của truy vấn. Bảng mới được tạo ra sẽ có số cột bằng số cột được chỉ định trong danh sách chọn và số dòng sẽ à số dòng kết quả của truy vấn.

Ví dụ: Câu lệnh dưới đây tạo mới một bảng có tên là ThanhPham bao gồm các vật tư là thành phẩm:

```
IF OBJECT_ID('tempdb..#ThanhPham') IS NOT NULL DROP TABLE #ThanhPham
SELECT ItemCode, ItemName, Unit, ItemType
    INTO #ThanhPham
    FROM Item
    WHERE ItemType = 2


SELECT * FROM #ThanhPham
```

1.7. Thống kê dữ liệu với GROUP BY và HAVING

Ta có thể sử dụng các mệnh đề GROUP BY và HAVING để thống kê dữ liệu. GROUP BY tổ chức dữ liệu vào các nhóm, HAVING thiết lập các điều kiện lên các nhóm trong kết quả truy vấn. Những mệnh đề này thường được sử dụng kết hợp với nhau (HAVING được sử dụng không kèm với GROUP BY có thể tạo ra những kết quả nhầm lẫn).

Các hàm gộp trả về các giá trị tóm lược cho cả bảng hoặc cho các nhóm trong bảng. Do đó, chúng thường được sử dụng với GROUP BY. Các hàm gộp có thể xuất hiện trong một dang sách chọn hay trong mệnh đề HAVING, nhưng không được sử dụng trong mệnh đề WHERE.

Ta có thể sử dụng các hàm gộp dưới đây với GROUP BY (Trong đó expression là một tên cột).

- SUM([ALL | DISTINCT] expression): Tính tổng các giá trị.
- AVG([ALL | DISTINCT] expression): Tính trung bình của các giá trị
- COUNT([ALL | DISTINCT] expression): Số các giá trị trong biểu thức.
- COUNT(*): Số các dòng được chọn.
- MAX(expression): Tính giá trị lớn nhất
- MIN(expression): Tính giá trị nhỏ nhất

Trong đó, SUM và AVG chỉ làm việc với những giá trị kiểu số. SUM, AVG, COUNT, MAX và MIN bỏ qua các giá trị null còn COUNT(*) thì không.

Ví dụ: Câu lệnh dưới đây cho biết tổng sản lượng và tiền bán ra của các mã hàng:

```
SELECT ItemCode, SUM(Quantity) AS Quantity, SUM(Amount2) AS Amount2
FROM AccDocDetail
WHERE DocCode = 'HD'
GROUP BY ItemCode
```

Chú ý: Danh sách các tên cột trong danh sách chọn của câu lệnh SELECT và danh sách các tên cột sau GROUP BY phải như nhau, nếu không câu lệnh sẽ không hợp lệ.

Mệnh đề HAVING thiết lập các điều kiện đối với mệnh đề GROUP BY tương tự như cách thức mệnh đề WHERE thiết lập các điều kiện cho câu lệnh SELECT. Mệnh đề HAVING sẽ không có nghĩa nếu như không sử dụng kết hợp với mệnh đề WHERE. Có một điểm khác biệt giữa HAVING và WHERE là trong điều kiện tìm kiếm của WHERE không được có các hàm gộp trong khi HAVING lại cho phép sử dụng các hàm gộp trong điều kiện tìm kiếm của mình. Mệnh đề HAVING có thể tham chiếu đến bất kỳ mục nào trong danh sách chọn và mệnh đề HAVING có thể chứa tối đa 128 điều kiện tìm kiếm.

Ví dụ: Câu lệnh dưới đây cho biết tổng sản lượng và tiền bán ra của các mã hàng mà có tổng sản lượng lớn hơn 500

```
SELECT ItemCode, SUM(Quantity) AS Quantity, SUM(Amount2) AS Amount2
    FROM AccDocDetail
    WHERE DocCode = 'HD'
    GROUP BY ItemCode
    HAVING SUM(Quantity) > 500
```

1.8. Sắp xếp kết quả truy vấn bằng ORDER BY

Mệnh đề ORDER BY được sử dụng nhằm sắp xếp kết quả truy vấn theo một hay nhiều cột (tối đa là 16 cột). Việc sắp xếp có thể theo thứ tự tăng tăng (ASC) hoặc giảm (DESC). Nếu không chỉ định rõ thì mặc định là tăng.

Ví dụ: Câu lệnh sau sẽ sắp xếp các vật tư theo thứ tự tăng dần của mã vật tư và nếu mã vật tư giống nhau thì sắp xếp theo thứ tự giảm dần của số lượng:

```
SELECT DocCode, DocNo, ItemCode, Quantity
    FROM AccDocDetail
    WHERE DocCode = 'HD'
    ORDER BY ItemCode ASC, Quantity DESC
```

2. Các lệnh kết nối dữ liệu

2.1. Lệnh INNER JOIN

Không chỉ SQL Server mà ở hầu hết các hệ quản trị CSDL quan hệ hiện nay thì INNER JOIN được sử dụng nhiều nhất trong các loại Join, nó giúp ta liên kết nhiều table với nhau thông qua ràng buộc khóa ngoại.

Giả sử ta có hai table AccDoc và AccDocDetail. Bảng AccDocDetail có một khóa ngoại liên kết đến bảng AccDoc qua trường DocNo

Câu hỏi đặt ra là làm sao lấy được thông tin của mỗi chứng từ và chi tiết các dòng dữ liệu thuộc về chứng từ đó. Nếu như truy vấn hai bảng thì đó đương nhiên kết quả sẽ không như mong đợi:

```
SELECT *
    FROM AccDoc, AccDocDetail
```

Nó sẽ trả ra kết quả là cấp số nhân của tổng record của hai bảng. Đây là dữ liệu không đúng, vì vậy ta sẽ phải sử dụng một phép tích khác, đó là phép INNER JOIN.

```
SELECT Select_List
    FROM Table_Name1 [Alias_Name1]
        INNER JOIN Table_Name2 [Alias_Name2] ON Join_Predicate
```

- Select_List: là danh sách các field muốn chọn ở hai bảng.
- Table_Name1 [Alias_Name1] và Table_Name2 [Alias_Name2] là hai table cần JOIN với nhau. Có thể dùng Alias hoặc không
- Join_Predicate là điều kiện JOIN.

Ví dụ:

```
SELECT Tb1.DocCode, Tb1.DocNo, Tb1.DocDate, Tb1.CustomerCode, Tb1.Description,
        Tb2.ItemCode, Tb2.Quantity, Tb2.UnitPrice, Tb2.Amount2
    FROM AccDocDetail Tb2
        INNER JOIN AccDoc Tb1 ON Tb1.DocNo = Tb2.DocNo
    WHERE Tb1.DocCode = 'HD'
    ORDER BY Tb1.DocNo
```

Lưu ý: INNER JOIN là từ khóa đầy đủ, có thể rút gọn bằng cách viết JOIN mà thôi.

```
SELECT Tb1.DocCode, Tb1.DocNo, Tb1.DocDate, Tb1.CustomerCode, Tb1.Description,
        Tb2.ItemCode, Tb2.Quantity, Tb2.UnitPrice, Tb2.Amount2
    FROM AccDocDetail Tb2
        JOIN AccDoc Tb1 ON Tb1.DocNo = Tb2.DocNo
    WHERE Tb1.DocCode = 'HD'
ORDER BY Tb1.DocNo
```

2.2. Lệnh LEFT OUTER JOIN

Nếu INNER JOIN là tích hai bảng và chỉ lấy các dòng trùng khóa ngoại với nhau thì LEFT OUTER JOIN sẽ lấy thêm cả các dòng mà bảng bên trái có nhưng bên phải không có.

LEFT OUTER JOIN sẽ hợp hai bảng lại và lấy những cặp record thỏa mãn ở điều kiện ON + Những dòng dữ liệu ở bảng bên trái có nhưng bảng bên phải không có.

```
SELECT Select_List
    FROM Table_Name1 [Alias_Name1]
    LEFT OUTER JOIN Table_Name2 [Alias_Name2] ON Join_Predicate
```

- Select_List: là danh sách các field muốn chọn ở hai bảng.
- Table_Name1 [Alias_Name1] và Table_Name2 [Alias_Name2] là hai table cần JOIN với nhau. Có thể dùng Alias hoặc không
- Join_Predicate là điều kiện JOIN

Ví dụ: Hóa đơn số HD008 có bán 4 mặt hàng, trong đó mã vật tư TP04 không tồn tại trong bảng Item. Xem hai cách viết sau để thấy sự khác biệt giữa INNER JOIN và LEFT OUTER JOIN

```
SELECT Tb1.DocNo, Tb1.ItemCode, Tb2.ItemName, Tb2.Unit,
                Tb1.Quantity, Tb1.UnitPrice, Tb1.Amount2
  FROM AccDocDetail Tb1
        INNER JOIN Item Tb2 ON Tb1.ItemCode = Tb2.ItemCode
  WHERE Tb1.DocNo = 'HD008'
  ORDER BY Tb1.DocNo, Tb1.ItemCode
--      Kết quả là mã TP01 không xuất hiện trong kết quả truy vấn vì nó không tồn tại trong
bảng Item

SELECT Tb1.DocNo, Tb1.ItemCode, Tb2.ItemName, Tb2.Unit,
                Tb1.Quantity, Tb1.UnitPrice, Tb1.Amount2
  FROM AccDocDetail Tb1
        LEFT JOIN Item Tb2 ON Tb1.ItemCode = Tb2.ItemCode
  WHERE Tb1.DocNo = 'HD008'
  ORDER BY Tb1.DocNo, Tb1.ItemCode
--  Kết quả là mã TP04 có xuất hiện trong kết quả truy vấn mặc dù nó không tồn tại trong bảng
    Item
```

Trên thực tế nếu chúng ta cần JOIN những bảng có yêu cầu ràng buột kiểu cha con thì sẽ dùng INNER JOIN, còn nếu có quan hệ 1 nhiều không theo kiểu cha con thì dùng LEFT OUTER JOIN. Cũng có thể kết hợp cả hai trong cùng 1 câu lệnh SELECT.

Ví dụ: Nếu muốn lấy bảng kê đầy đủ của những khách hàng và mặt hàng bán ra trong tháng thể hiện đầy đủ ngày bán hàng, tên khách hàng, tên hàng hóa thì sẽ kết hợp các hình thức JOIN dữ liệu từ 4 bảng.

```
SELECT Tb1.DocCode, Tb1.DocNo, Tb1.DocDate, Tb1.CustomerCode,
        Cus.CustomerName, b2.ItemCode, Item.ItemName, Item.Unit,
        Tb2.Quantity, Tb2.UnitPrice, Tb2.Amount2
  FROM AccDocDetail Tb2
        INNER JOIN AccDoc Tb1 ON Tb1.DocNo = Tb2.DocNo
        LEFT OUTER JOIN Customer Cus ON Tb1.CustomerCode = Cus.CustomerCode
        LEFT OUTER JOIN Item Item ON Tb2.ItemCode = Item.ItemCode
  WHERE Tb1.DocCode = 'HD'
  ORDER BY Tb1.DocNo, Tb2.ItemCode
```

3. Bài tập

BAITAP1: Hiển thị thông tin số chứng từ, ngày chứng từ, mã vật tư, tên vật tư, Số lượng của các chứng từ nhập kho.

BAITAP2: Hiển thị thông tin số chứng từ, ngày chứng từ, mã vật tư, tên vật tư, Số lượng của các chứng từ xuất kho sản xuất.

BAITAP3: Hiển thị thông tin ngày xuất gần nhất của vật tư nhập mua: Mã vật tư, Tên vật tư, Đơn vị tính, Ngày xuất gần nhất.

BAITAP4: Hiển thị số liệu nhập xuất theo ngày của vật tư: Ngày, Mã vật tư, tên vật tư, đơn vị tính, số lượng nhập, số lượng xuất.

BAITAP5: Hiển thị tổng số lượng vật tư được sản xuất và bán ra từ ngày 01/01/2022 đến ngày 15/01/2022: Mã vật tư, tên vật tư, đơn vị tính, số lượng sản xuất, số lượng bán ra.

BAITAP6: Hiển thị hai khách hàng có doanh số lớn nhất: Mã khách hàng, tên khách hàng, doanh số.

BAITAP7: Hiển thị tổng sản lượng theo từng vật tư theo cấu trúc: Mã vật tư, tên vật tư, đơn vị tính, Số lượng mua, số lượng xuất vào sản xuất, số lượng nhập thành phẩm, số lượng bán.

VIII. TOÁN TỬ UNION VÀ SUBQUERY

1. Toán tử UNION

Toán tử UNION cho phép ta hợp các kết quả của hai hay nhiều truy vấn thành một tập kết quả duy nhất. Cú pháp của phép hợp như sau:

```
Query_1
UNION [ALL] Query_2
...
UNION [ALL] Query_N
ORDER BY Clause]
```

Trong đó:

Query_1 có dạng:

```
SELECT Select_List
    INTO Clause
    FROM Clause
    WHERE clause
    GROUP BY Clause
    HAVING Clause
```

và Query_i (i=2,..,n) có dạng:

```
SELECT Select_List
    FROM Clause
    WHERE Clause
    GROUP BY Clause
    HAVING Clause
```

Lưu ý: Theo mặc định, phép toán UNION sẽ loại bỏ những dòng giống nhau trong kết quả. Nếu ta sử dụng tùy chọn ALL thì các dòng giống nhau sẽ không bị loại bỏ

Các nguyên tắc khi xây dựng câu lệnh UNION

Khi xây dựng các câu lệnh UNION, ta cần chú ý các nguyên tắc sau:

- Tất cả các danh sách chọn trong câu lệnh UNION phải có cùng số biểu thức (các tên cột, các biểu thức số học, các hàm gộp,...)
- Các cột tương ứng trong tất cả các bảng, hoặc tập con bất kỳ các cột được sử dụng trong bản thân mỗi truy vấn phải cùng kiểu dữ liệu.
- Các cột tương ứng trong bản thân từng truy vấn của một câu lệnh UNION phải xuất hiện theo thứ tự như nhau. Nguyên nhân là do phép hợp so sánh các cột từng cột một theo thứ tự được cho trong mỗi truy vấn.
- Khi các kiểu dữ liệu khác nhau được kết hợp với nhau trong câu lệnh UNION, chúng sẽ được chuyển sang kiểu dữ liệu cao hơn (nếu có thể được).
- Tiêu đề cột trong kết quả của phép hợp sẽ là tiêu đề cột được chỉ định trong truy vấn đầu tiên

Sử dụng UNION với các giao tác SQL khác

Các nguyên tắc sau phải được tuân theo khi sử dụng phép hợp với các câu lệnh giao tác SQL khác:

- Truy vấn đầu tiên trong câu lệnh UNION có thể có INTO để tạo một bảng từ kết quả cuối cùng.
- Mệnh đề ORDER BY và COMPUTE dùng để xác định thứ tự kết quả cuối cùng hoặc tính toán các giá trị tóm tắt chỉ được cho phép sử dụng ở cuối của câu lệnh UNION. Chúng không được phép sử dụng trong bất kỳ bản thân truy vấn nào trong phép hợp.
- Mệnh đề GROUP BY và HAVING chỉ có thể được sử dụng trong bản thân từng truy vấn. Chúng không thể được sử dụng để tác động lên kết quả cuối cùng.
- Phép toán UNION cũng có thể được sử dụng bên trong một câu lệnh INSERT.
- Phép toán UNION không thể sử dụng trong câu lệnh CREATE VIEW

2. Truy vấn con (Subquery)

Một truy vấn con là một câu lệnh SELECT được lồng vào bên trong một cậu lệnh SELECT, INSERT, UPDATE hay DELETE hoặc bên trong một truy vấn con khác. Câu lệnh truy vấn con có thể tham chiếu đến cùng 1 bảng với truy vấn ngoài hoặc một bảng khác.

Câu lệnh SELECT của truy vấn con luôn nằm trong dấu ngoặc. Nó không được chứa mệnh đề ORDER BY. Một truy vấn con có thể được lồng vào bên trong mệnh đề WHERE hay HAVING của một câu lệnh SELECT, INSERT hay DELETE, hoặc bên trong truy vấn con khác.

Các lệnh truy vấn con thường có một số các dạng sau:

(1) WHERE Expression [NOT] IN (Subquery)
(2) WHERE Expression Comparison_Operator [ANY|ALL] (Subquery)
(3) WHERE [NOT] EXISTS (Subquery)

Ví dụ: Lấy ra các chừng từ phát sinh trên các đối tượng không phải là cá nhân

```
SELECT DocCode, DocNo, DocDate, CustomerCode, Description
    FROM AccDoc
    WHERE CustomerCode IN (
        SELECT CustomerCode FROM Customer WHERE CustomerType <> 1)
```

3. Bài tập

BAITAP1:

- Tạo thêm bảng danh mục nhân viên: Employee với cấu trúc

Employee	
EmployeeCode	NVARCHAR (16)
EmloyeeName	NVARCHAR (96)
Gender	INT -> 1: Nam; 2: Nữ; 3: Khác
DeptCode	NVARCHAR (16)
Salary	NUMERIC(18,2)
IsActive	INT -> 0: Không sử dụng; 1: Sử dụng

- Thực hiện đưa dữ liệu sau vào bảng **Employee**

EmployeeCode	EmployeeName	Gender	DeptCode	Salary	IsActive
NV01	Phi Công Anh	1	TK	12.000.000	1
NV02	Đàm Văn Đức	1	TK	11.000.000	1
NV03	Ninh Ngọc Hiếu	1	TK	15.000.000	1
NV04	Nguyễn Thu Huyền	2	BH	10.000.000	1
NV05	Đỗ Xuân Thiết	1	BH	13.000.000	1

NV06	Nguyễn Xuân Dũng	1	CN	15.000.000	1
NV07	Nguyễn Sĩ Quyền	1	CN	14.000.000	1

- Lấy ra các đối tượng thuộc đơn vị tổ chức và nhân viên là nam với cấu trúc: Mã đối tượng, tên đối tượng.

BAITAP2: Hiển thị thông tin bảng kê những chứng từ bán hàng, sắp xếp theo thứ tự ngày tăng dần, số tăng dần. Phần bôi đậm sẽ được sắp xếp theo thứ tự giảm dần. Ra được kết quả như sau là đúng

Số chứng từ	Ngày chứng từ	Mã vật tư	Diễn giải	Đvt	Số lượng	Đơn giá	Tiền bán hàng
			KH03 - Công ty TNHH cà phê Thắng Lợi		**0**	**0**	**11250000000**
HD002	06/01/2022	TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	50	10000000	500000000
HD002	06/01/2022	TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	150	12000000	1800000000
HD006	18/01/2022	TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	50	10000000	500000000
HD006	18/01/2022	TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	150	12000000	1800000000
HD008	31/01/2022	TP02	Tủ đông kích thước 1.5*0.8*0.8	Cái	100	11000000	1100000000
HD008	31/01/2022	TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	90	10000000	900000000
HD008	31/01/2022	TP04	NULL	NULL	270	11000000	2970000000
HD008	31/01/2022	TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	140	12000000	1680000000
			KH02 - Công ty cổ phần đầu tư xây dựng Dacinco		**0**	**0**	**6300000000**
HD003	10/01/2022	TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	50	10000000	500000000
HD003	10/01/2022	TP02	Tủ đông kích thước 1.5*0.8*0.8	Cái	100	11000000	1100000000
HD005	16/01/2022	TP02	Tủ đông kích thước 1.5*0.8*0.8	Cái	100	11000000	1100000000
HD005	16/01/2022	TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	300	12000000	3600000000
			KH01 - Đại lý Cô Tám		**0**	**0**	**10750000000**
HD001	05/01/2022	TP02	Tủ đông kích thước 1.5*0.8*0.8	Cái	100	11000000	1100000000
HD001	05/01/2022	TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	300	12000000	3600000000
HD004	12/01/2022	TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	90	10000000	900000000
HD004	12/01/2022	TP02	Tủ đông kích thước 1.5*0.8*0.8	Cái	270	11000000	2970000000

HD004	12/01/2022	TP03	Tủ mát kích thước 0.8*0.8*1.9	Cái	140	12000000	1680000000
HD007	23/01/2022	TP01	Tủ đông kích thước 1.5*0.7*0.6	Cái	50	10000000	500000000

BAITAP3: Hiển thị báo cáo nhập xuất tồn theo cấu trúc sau:

Mã vật tư	Tên vật tư	Đvt	Tồn đầu	SL nhập	SL xuất	Tồn cuối
	Tổng cộng					

IX. MỆNH ĐỀ PIVOT, LỆNH VÒNG LẶP VÀ PROCEDURE

1. Mệnh đề PIVOT

Trong SQL Server, mệnh đề PIVOT cho phép phân tích bảng chéo chuyển dữ liệu từ bảng này sang bảng khác, tức là lấy kết quả tổng hợp rồi chuyển từ dòng thành cột.

Cú pháp mệnh đề PIVOT

```
SELECT Cot_Dautien AS <Bidanh_Cot_dautien>,
        [Giatri_Chuyen1], [Giatri_Chuyen2], … [Giatri_chuyen_n]
    FROM
        (<Bang_Nguon>) AS <Bidanh_Bang_Nguon>
    PIVOT
        (Ham_Tong (<Cot_Tong>)
    FOR <Cot_Chuyen>
    IN ([Giatri_Chuyen1], [Giatri_Chuyen2], … [Giatri_Chuyen_n])
        ) AS <Bidanh_Bang_Chuyen>;
```

Tên biến hoặc giá trị biến:

- Cot_Dautien: Cột hoặc biểu thức sẽ thành cột đầu tiên trong bảng mới sau khi chuyển.
- Bidanh_Cot_Dautien: Tên của cột đầu tiên trong bảng mới sau khi chuyển.
- Giatri_Chuyen1, Giatri_Chuyen2, … Giatri_Chuyen_n: Danh sách các giá trị cần chuyển.
- Bang_Nguon: Lệnh SELECT đưa dữ liệu nguồn (dữ liệu ban đầu) vào bảng mới.
- Bidanh_Bang_Nguon: Bí danh của bảng nguồn
- Ham_Tong: Hàm tính tổng như SUM COUNT, MIN, MAX hay AVG.
- Cot_Tong: Cột hoặc biểu thức được dùng với hàm tổng.
- Cot_Chuyen: Cột chứa giá trị cần chuyển.
- Bidanh_Bang_Chuyen: Bí danh của bảng sau khi chuyển.

Ví dụ:

Cần lấy tổng số tiền lương theo bộ phận của danh mục nhân viên và mã bộ phận là thể hiện trên cột.

```
SELECT N'Tổng lương' AS Tong_Luong, BH, CN, TK
    FROM (SELECT DeptCode, Salary FROM EmPloyee) AS BangNguon
    PIVOT
            (
            SUM(Salary)
            FOR DeptCode IN (BH, CN, TK)
```

```
        ) AS BangChuyen
```

Hoặc cần lấy bảng kê lương nhân viên theo bộ phận và tổng lương ở dòng cuối

```
SELECT EmployeeCode, EmployeeName, BH, CN, TK
    FROM Employee AS Tb1
    PIVOT
            (
            SUM (Salary) FOR DeptCode IN (BH, CN, TK)
            ) AS Tb2
    UNION ALL
    SELECT CAST ('' AS NVARCHAR(16)) AS EmployeeCode,
            N'Tổng lương' AS EmployeeName, BH, CN, TK
            FROM (SELECT DeptCode, Salary FROM EmPloyee) AS Tb3
                    PIVOT
                    (
                    SUM(Salary)
                    FOR DeptCode IN (BH, CN, TK)

                    ) AS Tb4
```

2. Lệnh vòng lặp WHILE

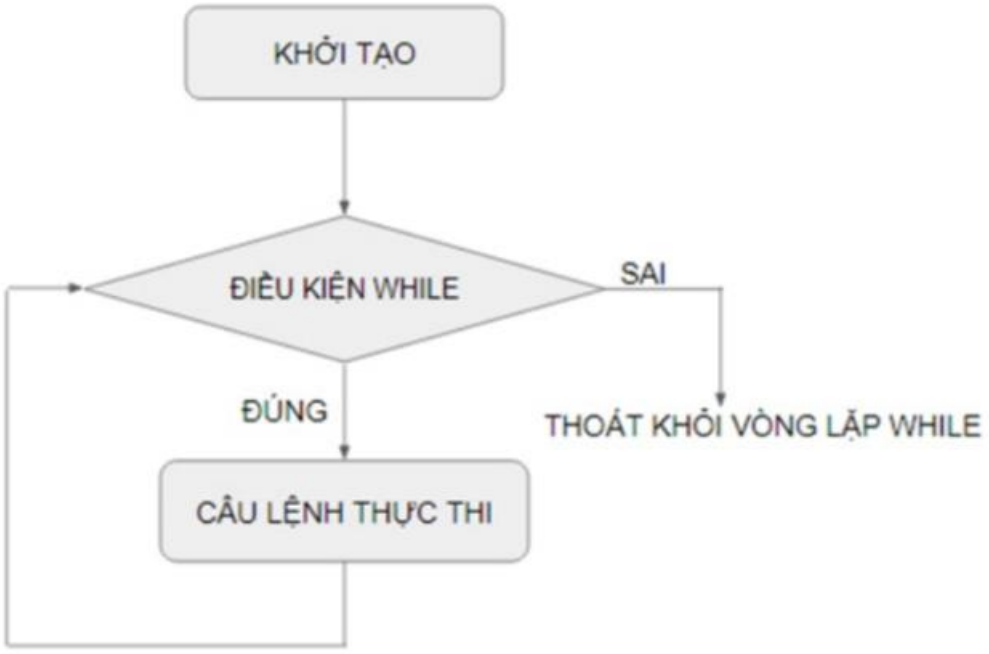
Vòng lặp WHILE được sử dụng nếu muốn chạy lặp đi lặp lại một đoạn mã khi điều kiện cho trước trả về giá trị là TRUE. Cú pháp:

```
WHILE Dieukien / * Các lệnh cần lặp * /
    BEGIN
        {.....Câu lệnh thực thi khi điều kiện là TRUE.....}
    END
```

Ghi chú:

- Sử dụng câu lệnh vòng lặp WHILE khi không chắc chắn về số lần muốn thực thi.
- Vì điều kiện WHILE được đánh giá trước khi vào vòng lặp nên vòng lặp có thể không thực hiện được lần nào (khi dieukien là FALSE thì vòng lặp sẽ kết thúc ngay lập tức).
- Câu lệnh BREAK sẽ thoát khỏi vòng lặp WHILE sớm.
- Câu lệnh CONTINUE sẽ khởi động lại vòng lặp WHILE từ đầu

Biểu đồ vòng lặp WHILE



Ví dụ:

```
DECLARE @_Number INT = 1, @_Total INT = 0

WHILE @_Number <= 10
    BEGIN
            SET @_Total = @_Total + @_Number;
            SET @_Number = @_Number + 1 ;
    END

PRINT @_Total
```

Trong ví dụ này, vòng lặp sẽ không thực hiện lần nào nếu ngay từ đầu @Number > 10, nó chỉ thực hiện và duy trì khi biến < = 10. Đến khi vượt quá điều kiện (> 10), vòng lặp sẽ kết thúc và tiếp tục thực thi các câu lệnh tiếp theo.

Nếu muốn kết thúc vòng lặp khi giá trị = 5

```
DECLARE @_Number INT = 1, @_Total INT = 0

WHILE @_Number <= 10
    BEGIN
            IF @_Number = 5
                BREAK;
            ELSE
            SET @_Total = @_Total + @_Number;
            SET @_Number = @_Number + 1 ;
    END

PRINT @_Total
```

3. Hàm RANKING

Các hàm Ranking cho phép ta có thể đánh số liên tục (xếp loại) cho các tập hợp kết quả. Các hàm này có thể được sử dụng để cung cấp số thứ tự trong hệ thống đánh số tuần tự khác nhau. Có thể hiểu đơn giản như sau: ta có từng con số nằm trên từng dòng liên tục, tại dòng thứ nhất xếp loại số 1, dòng thứ 2 xếp loại số là 2… Ta có thể sử dụng hàm ranking theo các nhóm số tuần tự, mỗi một nhóm sẽ được đánh số theo lược đồ 1,2,3 và nhóm tiếp theo lại bắt đầu bằng 1,2,3…

❖ **Hàm ROW_NUMBER**

Hàm ROW_NUMBER: Hàm này trả lại một dãy số tuần tự bắt đầu từ 1 cho mỗi dòng hay nhóm trong tập hợp kết quả. Hàm ROW_NUMBER sẽ có cú pháp sau:

```
ROW_NUMBER ( ) OVER ( [ ] )
```

Ví dụ: Lấy ra số thứ tự trong bảng Employee

```
SELECT ROW_NUMBER() OVER (ORDER BY EmployeeCode) AS [Order],
        EmployeeCode, EmployeeName, Gender, DeptCode, Salary
FROM Employee
```

Hàm ROW_NUMBER không chỉ cho phép sắp xếp toàn bộ tập hợp dòng mà còn có thể sử dụng mệnh đề PARTITION để lọc ra nhóm dòng cần đánh số. Các dòng sẽ được đánh số tuần tự trong từng giá trị PARTITION độc nhất. Các dãy số được đánh sẽ luôn bắt đầu từ 1 cho từng giá trị PARTITION mới trong tập hợp bản ghi.

Ví dụ: Lấy danh sách cán bộ nhân viên theo bộ phận và đánh số thứ tự theo danh sách nhân viên thuộc bộ phận sắp xếp theo lương giảm dần.

```
SELECT ROW_NUMBER() OVER (
    PARTITION BY DeptCode ORDER BY Salary DESC) AS [Order],
        DeptCode, EmployeeCode, EmployeeName, Salary
    FROM Employee
```

❖ **Hàm RANK**

Đôi khi ta muốn một dòng có cùng sắp xếp giá trị cột như các dòng khác có cùng một xếp loại. Nếu thế thì hàm RANK () có thể hỗ trợ. Hàm RANK có cú pháp như sau:

```
RANK ( ) OVER ( [] )
```

Hàm RANK sẽ đánh số liên tục một tập hợp bản ghi nhưng khi có 2 dòng có cùng giá trị sắp xếp thì hàm sẽ đánh giá là cùng bậc giá trị. Giá trị xếp loại vẫn sẽ tăng kể cả khi có 2 dòng cùng giá trị, vì vậy khi đánh giá một giá trị sắp xếp tiếp theo thì số thứ tự vẫn tiếp tục được đánh nhưng sẽ tăng thêm 1 giá trị vào các dòng tiếp theo trong tập hợp.

Ví dụ:

```
SELECT RANK() OVER (ORDER BY DeptCode) AS [Order],
        DeptCode, EmployeeCode, EmployeeName, Salary
FROM Employee
```

Nếu muốn có một nhiều xếp loại trong tập hợp bản ghi của mình thì với từng xếp loại cần đặt một nhóm cụ thể bằng cách sử dụng mệnh đề PARTITION BY trong hàm RANK.

Ví dụ:

```
SELECT RANK() OVER (PARTITION BY Gender ORDER BY DeptCode) AS [Order],
    EmployeeCode, EmployeeName, Gender, DeptCode, Salary
FROM Employee
```

❖ **Hàm DENSE_RANK**

Hàm DENSE_RANK cũng giống như hàm RANK, tuy vậy, hàm này không cung cấp khoảng cách giữa các số xếp loại. Thay vào đó, hàm này sẽ xếp loại liên tục cho từng giá trị ORDER BY cụ thể. Với hàm DENSE_RANK, kể cả khi có hai dòng có cùng giá trị xếp loại thì dòng tiếp theo vẫn chỉ tăng thêm một giá trị so với dòng trên. Hàm DENSE_RANK có cú pháp như hàm RANK.

Ví dụ:

```
SELECT DENSE_RANK() OVER (ORDER BY DeptCode) AS [Order],
        DeptCode, EmployeeCode, EmployeeName, Salary
FROM Employee

SELECT DENSE_RANK() OVER (PARTITION BY Gender ORDER BY DeptCode) AS [Order],
        EmployeeCode, EmployeeName, Gender, DeptCode, Salary
FROM Employee
```

4. PROCEDURE (Thủ tục) trong SQL Server)

Procedure là một chương trình trong cơ sở dữ liệu gồm nhiều câu lệnh lưu lại cho những lần sử dụng sau. Có thể truyền các tham số vào procedure, tuy nó không trả về một giá trị cụ thể như function (hàm) nhưng cho biết việc thực thi thành công hay thất bại.

Để tạo một procedure trong SQL Server, ta sử dụng cú pháp như dưới đây:

```
CREATE { PROCEDURE | PROC } [Schema_Name.]Procedure_Name
        [ @_Parameter [Type_Schema_Name.] DataType
        [ VARYING ] [ = Default ] [ OUT | OUTPUT | READONLY ]
                , @_Parameter [Type_Schema_Name.] DataType
        [ VARYING ] [ = Default ] [ OUT | OUTPUT | READONLY ] ]

        [ WITH { ENCRYPTION | RECOMPILE | EXECUTE AS Clause } ]
        [ FOR REPLICATION ]

    AS

    BEGIN
            [Declaration_Section]

            Executable_Section

    END
```

Tham số:

- *Schema_Name:* Tên Schema (lược đồ) sở hữu procedure.
- *Procedure_Name:* Tên gán cho procedure

- *@_Parameter:* Một hay nhiều tham số được truyền vào.
- *Type_Schema_Name:* Kiểu dữ liệu của Schema (nếu có).
- *DataType:* Kiểu dữ liệu cho @_Parameter.
- *Default:* Giá trị mặc định gán cho @_Parameter.
- *OUT/OUTPUT:* @_Parameter là một tham số đầu ra
- *READONLY:* @_Parameter không thể bị procedure ghi đè lên.
- *ENCRYPTION:* Mã nguồn (source) của procedure sẽ không được lưu trữ dưới dạng text trong hệ thống.
- *RECOMPILE:* Truy vấn sẽ không được lưu ở bộ nhớ đệm (cache) cho thủ tục này.
- *EXECUTE AS Clause:* Xác định ngữ cảnh bảo mật để thực thi thủ tục.
- *FOR REPLICATION:* Procedure đã lưu sẽ chỉ được thực thi trong quá trình replication (nhân bản)

5. Bài tập

BAITAP1: Trong câu lệnh PIVOT này

```
SELECT EmployeeCode, EmployeeName, BH, CN, TK
    FROM Employee AS Tb1
    PIVOT
            (
            SUM (Salary) FOR DeptCode IN (BH, CN, TK)
            ) AS Tb2
    UNION ALL
    SELECT CAST ('' AS NVARCHAR(16)) AS EmployeeCode,
            N'Tổng lương' AS EmployeeName, BH, CN, TK
            FROM (SELECT DeptCode, Salary FROM EmPloyee) AS Tb3
                    PIVOT
                    (
                    SUM(Salary)
                    FOR DeptCode IN (BH, CN, TK)

                    ) AS Tb4
```

Kết quả sẽ nhận được như hình:

EmployeeCode	EmployeeName	BH	CN	TK
NV01	Phi Công Anh	NULL	NULL	12000000.00
NV02	Đàm Văn Đức	NULL	NULL	11000000.00
NV03	Ninh Ngọc Hiếu	NULL	NULL	15000000.00
NV04	Nguyễn Thu Huyền	10000000.00	NULL	NULL
NV05	Đỗ Xuân Thiết	13000000.00	NULL	NULL
NV06	Nguyễn Xuân Dũng	NULL	15000000.00	NULL
NV07	Nguyễn Sĩ Quyền	NULL	14000000.00	NULL
	Tổng lương	23000000.00	29000000.00	38000000.00

Xử lý lại câu lệnh sao cho:

- ✓ Hiển thị kết quả không còn NULL.
- ✓ Có thể động được trong trường hợp đổi mã bộ phận (DeptCode) hoặc thêm mới bộ phận thì vẫn chạy được mà không sửa code

BAITAP2: Viết báo cáo tổng hợp số lượng bán hàng theo từng khách hàng và từng mặt hàng với cấu trúc như sau:

Stt	Mã vật tư	Tên vật tư	Đvt	Tên KH 1…	Tên KH 2…	…	Tổng số lượng
1							

2							
…							
	Tổng cộng						

BAITAP3: Viết báo cáo tổng hợp doanh số bán hàng theo từng khách hàng và từng mặt hàng với cấu trúc như sau

Stt	Mã khách hàng	Tên khách hàng	(Tên vật tư 1…)	(Tên vật tư 2…)	(…)	Tổng tiền
1						
2						
…						
	Tổng cộng					

BAITAP4: Viết báo cáo thể hiện bảng kê có nội dung và cấu trúc tương tự như bảng sau:

STT	Số chứng từ	Ngày chứng từ	Ma_Vt	Diễn giải(Tên vật tư)	Tiền
			A	**Vật tư A**	**300,000**
1	0001	01/02/18	A	Mua chứng từ	100,000
2	0011	10/02/18	A	Nhập kho	200,000
			B	**Vật tư B**	**550,000**
1	0002	01/02/18	B	Nhập thành phẩm	500,000
2	0012	10/02/18	B	Nhập kho	40,000
3	0016	12/02/18	B	Nhập kho	10,000
				Tổng cộng	**850,000**

BAITAP5: Viết báo cáo thể hiện chi tiết các lần nhập xuất vật tư có nội dung và cấu trúc tương tự bảng sau:

Mã hàng	Ngày	Nội dung	SL nhập	Tiền nhập	SL xuất	Tiền xuất	Tồn kho	Số dư
HANG1		Tồn đầu kỳ					3	15,000
HANG1	01/01/2013	Nhập hàng	3	15,000			6	30,000
HANG1	02/01/2013	Nhập hàng	10	50,000			16	80,000

HANG1	03/01/2013	Nhập hàng	25	125,000			41	205,000
HANG1	12/01/2013	Xuất hàng			10	50,000	31	155,000
HANG1	13/01/2013	Xuất hàng			25	125,000	6	30,000
HANG1		Tổng nhập/xuất	38	190,000	35	175,000		
HANG1		Tồn cuối kỳ					6	30,000
HANG2		Tồn đầu kỳ					6	118,200
HANG2	03/01/2013	Nhập hàng	6	118,200			12	236,400
HANG2	05/01/2013	Nhập hàng	30	582,000			42	818,400
HANG2	13/01/2013	Xuất hàng			25	486,250	17	332,150
HANG2	15/01/2013	Xuất hàng			5	97,250	12	234,900
HANG2		Tổng nhập/xuất	36	700,200	30	583,500		
HANG2		Tản cuèi kú					12	234,900
HANG3		Tồn đầu kỳ					80	960,000
HANG3	05/01/2013	Nhập hàng	123	1,476,000			203	2,436,000
HANG3	10/01/2013	Nhập hàng	80	960,000			283	3,396,000
HANG3	15/01/2013	Xuất hàng			125	1,500,000	158	1,896,000
HANG3	16/01/2013	Xuất hàng			25	300,000	133	1,596,000
HANG3		Tổng nhập/xuất	203	2,436,000	150	1,800,000		
HANG3		Tồn cuối kỳ					133	1,596,000

…End…