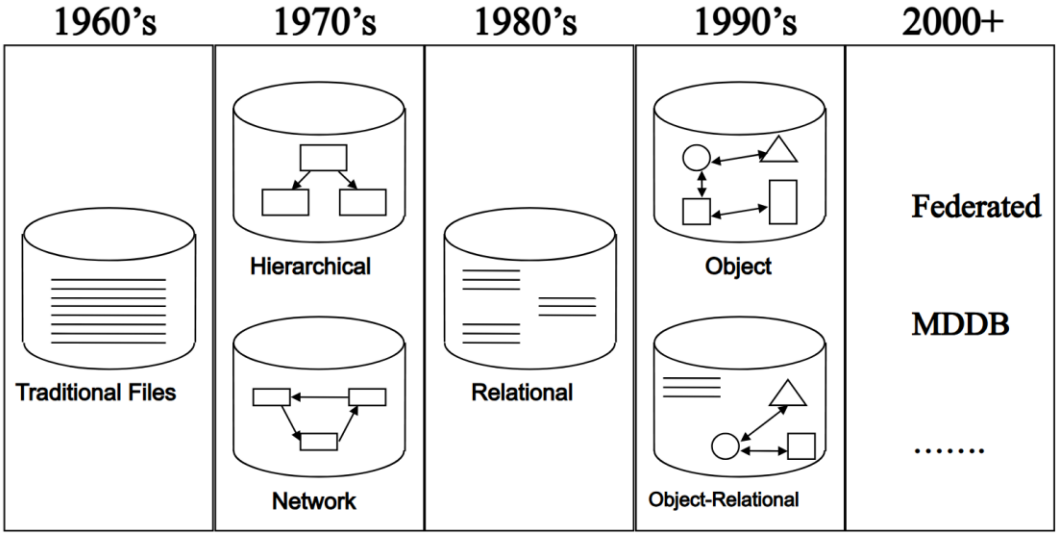
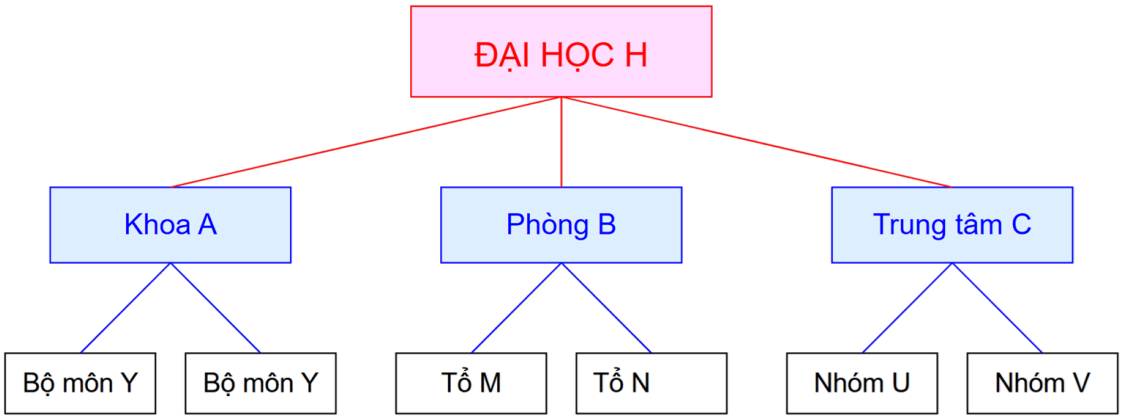
Chương I. TỔNG QUAN VỀ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

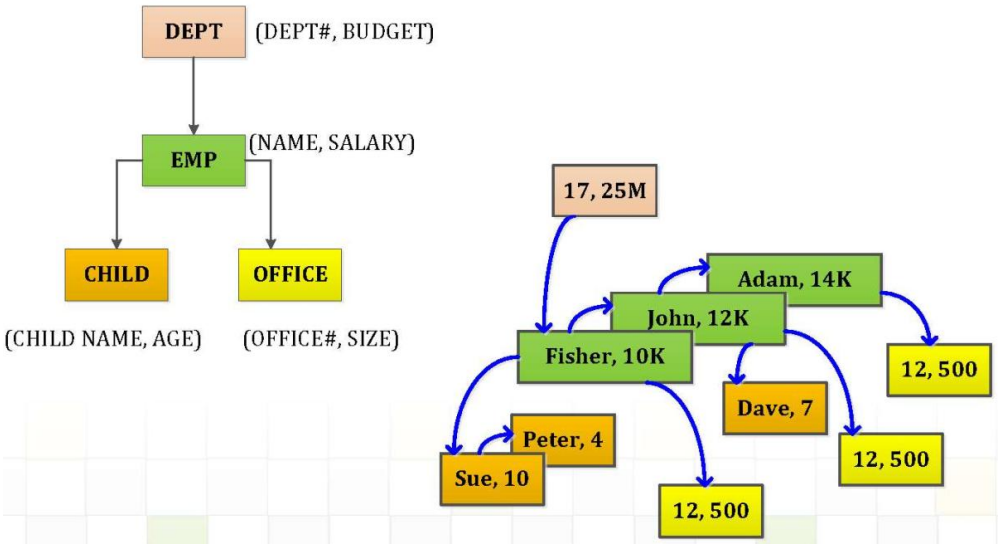
1.1 Đặt vấn đề & Nhắc lại

- Ứng dụng có sử dụng CSDL rất rất phổ biến hiện nay, bao phủ hầu hết trong các hoạt động kinh tế, xã hội, giáo dục, y tế ⇒ **Tầm quan trọng của một công cụ quản trị CSDL**
- **Lịch sử phát triển của mô hình CSDL** cũng qua nhiều giai đoạn:



- **Các loại mô hình dữ liệu:**
  - ***Mô hình dữ liệu phân cấp:***
    - Các dữ liệu được chứa trong các bản ghi (records) và được chia làm một số cấp. Mỗi đối tượng, còn gọi là nút (node), ở cấp trên *có thể có liên kết với một số đối tượng ở cấp thấp hơn.*

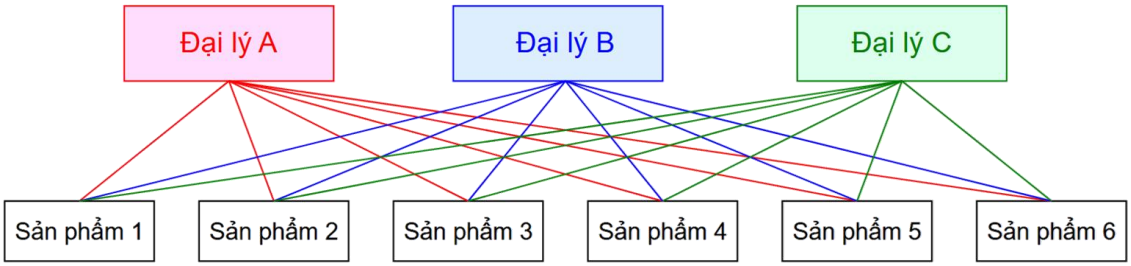


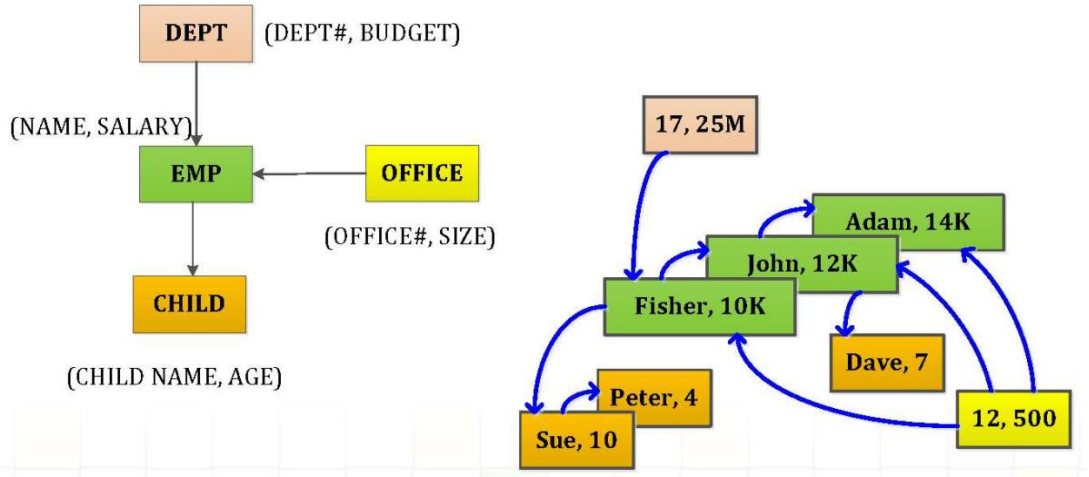


    - Mô hình này có ưu thế là có *cấu trúc đơn giản, dễ thiết kế và triển khai*. Tuy nhiên cấu trúc này lại không mềm dẻo, khi dữ liệu có sự thay đổi nào đó thì việc điều chỉnh lại HQTCSDL tương đối phức tạp và khó khăn. Mặt khác sự dư thừa

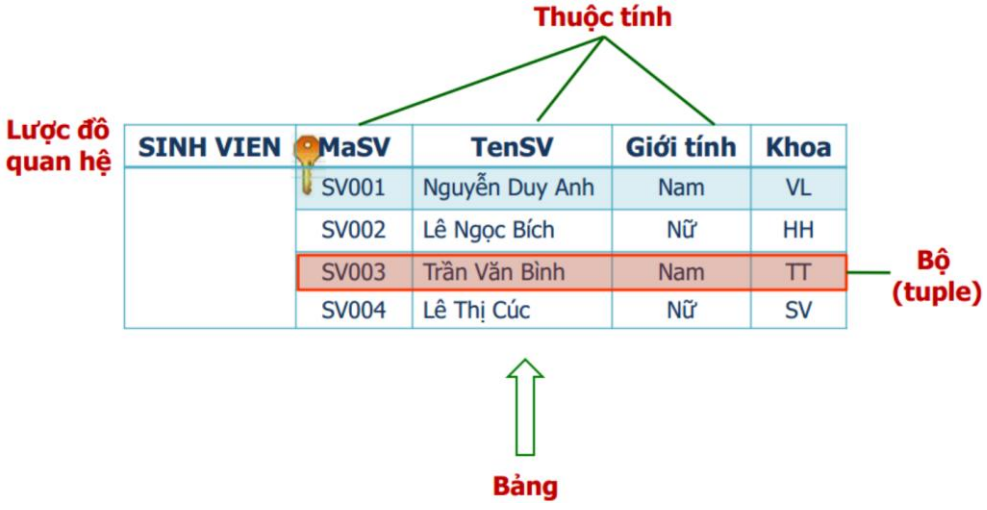
dữ liệu xảy ra do trùng lặp vẫn xảy ra nên vẫn đặt ra những vấn đề và phiền toái như khó cập nhật, không nhất quán. Tương đối khó dùng khi dữ liệu có cấu trúc phức tạp.

- ***Mô hình dữ liệu mạng:***
  - Cũng như mô hình phân cấp, dữ liệu trong mô hình cơ sở dữ liệu mạng (gọi tắt là mô hình mạng) cũng được chứa trong các tập tin.
  - Tuy nhiên cấu trúc trong mô hình mạng mềm dẻo hơn, các đối tượng có khả năng liên kết phong phú hơn như liên kết vượt cấp, ngang cấp, giữa cấp trên và cấp dưới không còn ràng buộc chặt chẽ.
  - **Ưu điểm:** tổ chức dữ liệu linh hoạt hơn, có thể gi ải quyết được một số khuyết điểm của mô hình phân cấp như sự trùng chập dữ liệu, sự không nhất quán.
  - Tuy nhiên do số liên kết tăng lên nên cấu trúc phức tạp hơn. Việc thiết kế, lập trình cho các chương trình ứng dụng và triển khai vào thực tế khó khăn hơn.

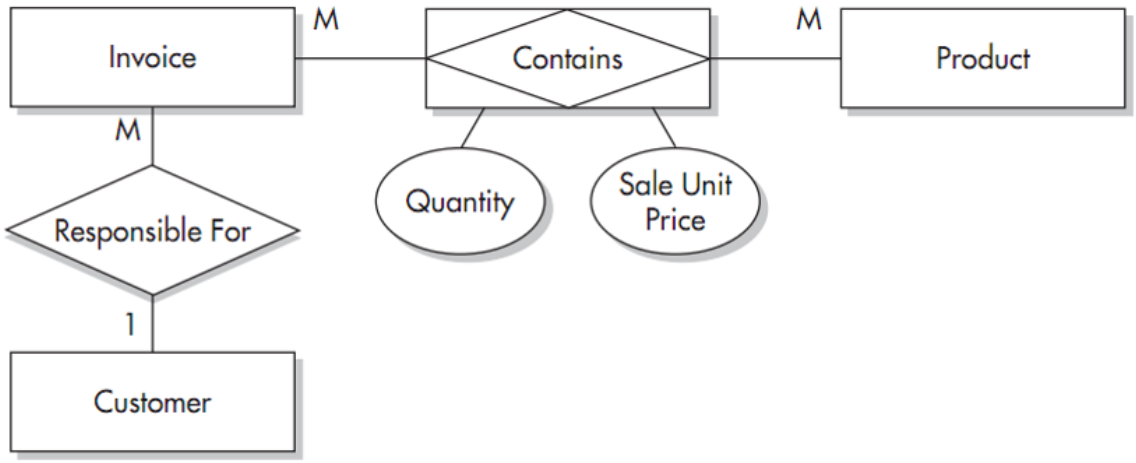




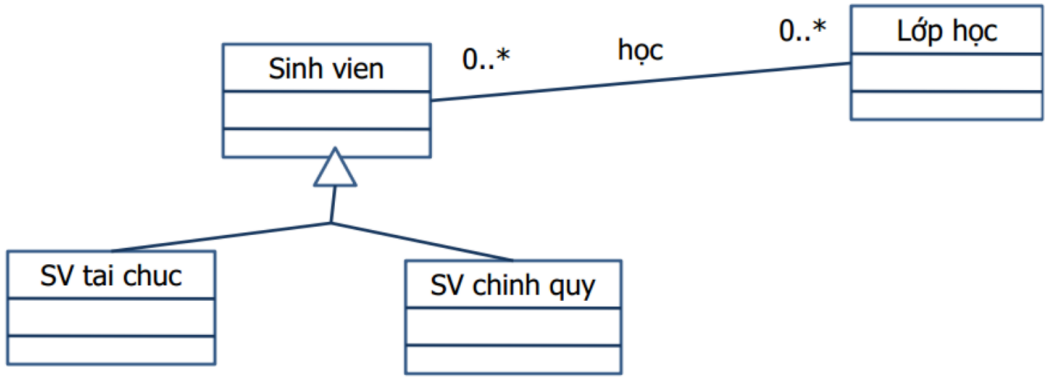
- ***Mô hình dữ liệu quan hệ:*** Thuộc tính, Lược đồ quan hệ, Bộ, Quan hệ, Khóa.



- ***Mô hình thực thể kết hợp:*** Thực thể, Thuộc tính, Mối quan hệ



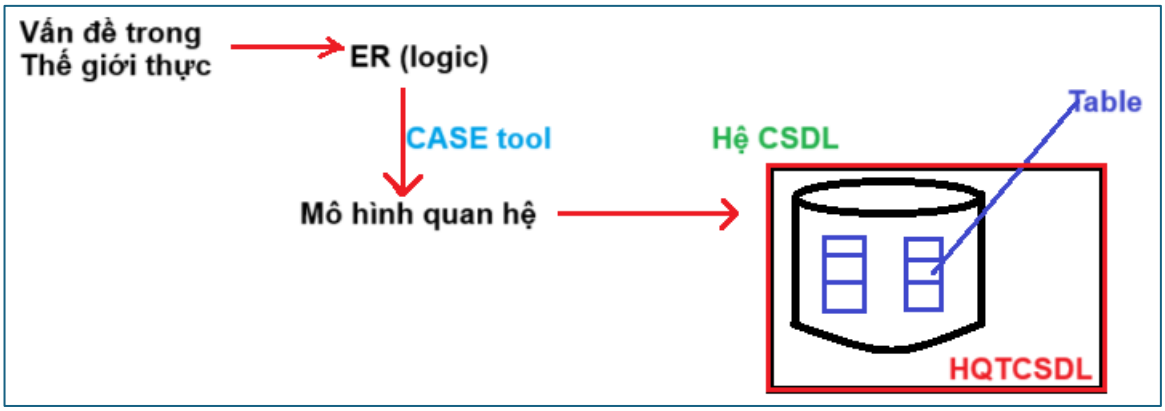
- ***Mô hình dữ liệu hướng đối tượng:*** dựa trên cách tiếp cận hướng đối tượng



- **Hệ thống cơ sở dữ liệu** là dữ liệu được lưu trữ có tổ chức, cho phép việc quản lý dữ liệu hiệu quả và nhanh chóng
  - **Lợi ích của hệ thống cơ sở dữ liệu**
    - Tránh dư thừa, trùng lặp dữ liệu.
    - Đảm bảo sự nhất quán trong cơ sở dữ liệu.
    - Các dữ liệu có thể được chia sẻ.
    - Có thể thiết lập các chuẩn trên cơ sở dữ liệu.
    - Đảm bảo tính toàn vẹn dữ liệu.
    - Khả năng bảo mật dữ liệu.

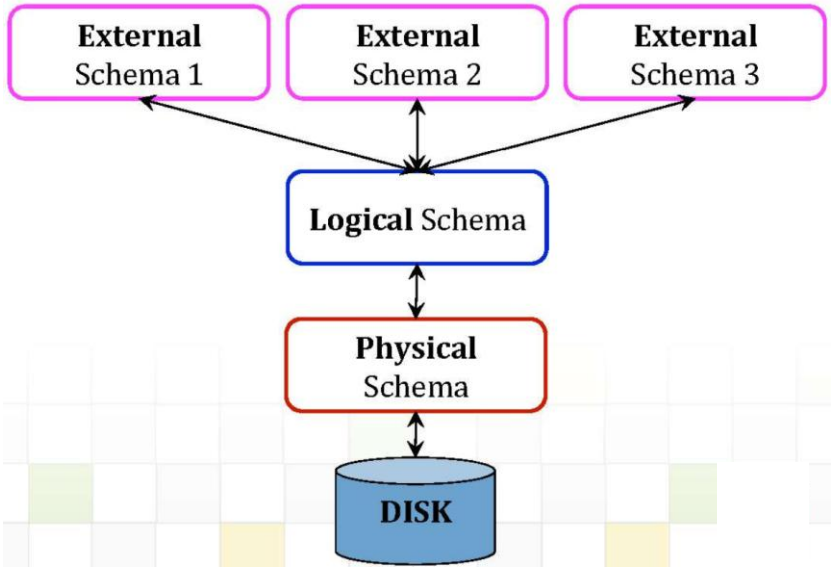
- **Hệ thống quản lý dựa trên tập tin**
  - ***Ưu điểm***
    - Lưu ngay khi cần nên tốc độ triển khai nhanh
    - Rõ ràng, trực quan
  - ***Nhược điểm***
    - Dư thừa và không nhất quán dữ liệu
    - Trùng lặp dữ liệu
    - Tính chia sẻ dữ liệu không cao
    - Vấn đề bảo mật

- **Cách vấn đề trong thực tế được hình thành nên cơ sở dữ liệu:**



1.2 Yêu cầu về dữ liệu trong CSDL

- Dữ liệu trong CSDL phải được thể hiện ở các mức độ trừu tượng khác nhau:
  - ***Mức ngoài (External level):*** Mô tả một phần của CSDL mà một đối tượng / một nhóm người dùng được quyền tiếp cận
  - ***Mức luận lý (Logic level):*** Mô tả những thông tin gì được lưu trữ trong CSDL và những mối quan hệ giữa những thông tin đó
  - ***Mức vật lý (Physical level):*** Dữ liệu được lưu trữ như thế nào trên thiết bị lưu trữ. *Bao gồm chi tiết về cấu trúc tệp, chỉ mục, và chiến lược lưu trữ.*
    - ⇒ Làm tăng **tính độc lập (data independence)** của cách thức lưu trữ dữ liệu, thiết kế dữ liệu và chương trình sử dụng dữ liệu.



- **Ví dụ:** Một hệ thống quản lý bệnh viện có thể được tổ chức như sau:
  - ❖ **Mức ngoài:**
    - Bác sĩ chỉ xem được thông tin bệnh án của bệnh nhân.
    - Nhân viên thu ngân chỉ xem thông tin thanh toán.
  - ❖ **Mức luận lý:**
    - Các bảng dữ liệu gồm:
      - BenhNhan (MaBN, HoTen, NgaySinh)
      - BenhAn (MaBA, MaBN, ChanDoan, NgayLap)
    - Ràng buộc: MaBN trong bảng BenhNhan là khóa ngoại trong bảng BenhAn.
  - ❖ **Mức vật lý:**
    - Hệ thống lưu trữ dữ liệu trên máy chủ SQL Server, sử dụng chỉ mục để tăng tốc truy vấn dựa trên MaBN.

- **Dữ liệu** trong CSDL **cần có các đặc trưng**:

- Ít hoặc không trùng lắp dữ liệu
- Chia sẻ cho nhiều người dùng mà không gây ra xung đột
- An ninh, bảo mật
- Khôi phục khi có sự cố
- ***Độc lập dữ liệu***
  - **Độc lập luận lý:** Khả năng thay đổi lược đồ mức luận lý mà không làm ảnh hưởng đến lược đồ ngoài cũng như chương trình ứng dụng.
    - Cho phép *thêm bớt thuộc tính, bảng, các mối quan hệ* mà không cần phải viết lại chương trình, ...
  - **Độc lập vật lý:** Khả năng thay đổi tổ chức vật lý của CSDL mà không làm ảnh hưởng đến lược đồ luận lý.
    - Cho phép thay đổi thiết bị lưu trữ, cách thức lưu trữ, các cấu trúc dữ liệu, các tổ chức tập tin khác nhau, các kiểu tổ chức chỉ mục khác nhau, ...
- Vì vậy cần có một hệ thống quản lý hiệu quả dữ liệu trong CSDL.

1.3 Khái niệm HQT CSDL

- Là một **hệ thống phần mềm** cung cấp các công cụ để xây dựng, khai thác và quản lý cơ sở dữ liệu.
  - **Xây dựng (Sử dụng ngôn ngữ DDL):** Định nghĩa cấu trúc CSDL, lưu trữ dữ liệu
  - **Khai thác (Sử dụng ngôn ngữ DML):** Truy vấn dữ liệu, Cập nhật dữ liệu
  - **Quản lý:** Quản lý an toàn và bảo mật, Điều khiển truy xuất đồng thời, Khôi phục khi có sự cố…
- *Một số HQTCSDL:* MS SQL Server, Oracle, DB2, ...

- **Chức năng chính:**
  - **Cung cấp môi trường tạo lập CSDL** là cung cấp cho người dùng ngôn ngữ định nghĩa kiểu dữ liệu để người dùng khai báo kiểu và các cấu trúc của dữ liệu đồng thời *tạo lập CSDL thông qua các giao diện đồ họa.*
  - **Cung cấp môi trường cập nhật và khai thác dữ liệu** là cung cấp cho người dùng ngôn ngữ thao tác dữ liệu để yêu cầu cập nhật hay khai thác thông tin.
    - **Thao tác dữ liệu gồm:**
      - **Cập nhật** (nhập, xóa, sửa dữ liệu)
      - **Khai thác** (Sắp xếp, tìm kiếm, kết xuất, báo cáo, ...)
  - **Cung cấp công cụ kiểm soát, điều khiển và truy cập vào CSDL:** Hệ HQTCSDL phải có các bộ chương trình thực hiện những nhiệm vụ sau:
    - Phát hiện, ngăn chặn sự truy cập trái phép
    - Duy trì tính nhất quán dữ liệu
    - Tổ chức điều khiển truy cập đồng thời
    - Khôi phục CSDL khi có sự cố ở phần cứng hay phần mềm
    - Quản lý các mô tả dữ liệu

1.4 Kiến trúc của một HQT CSDL

- **Thành phần Giao diện lập trình**
  - HQTCSDL cung cấp giao diện lập trình dễ sử dụng với một ngôn ngữ lập trình CSDL:
    - **Giao diện:** tương tác dòng lệnh (command line), đồ họa (GUI)
    - **Ngôn ngữ:** SQL, T-SQL

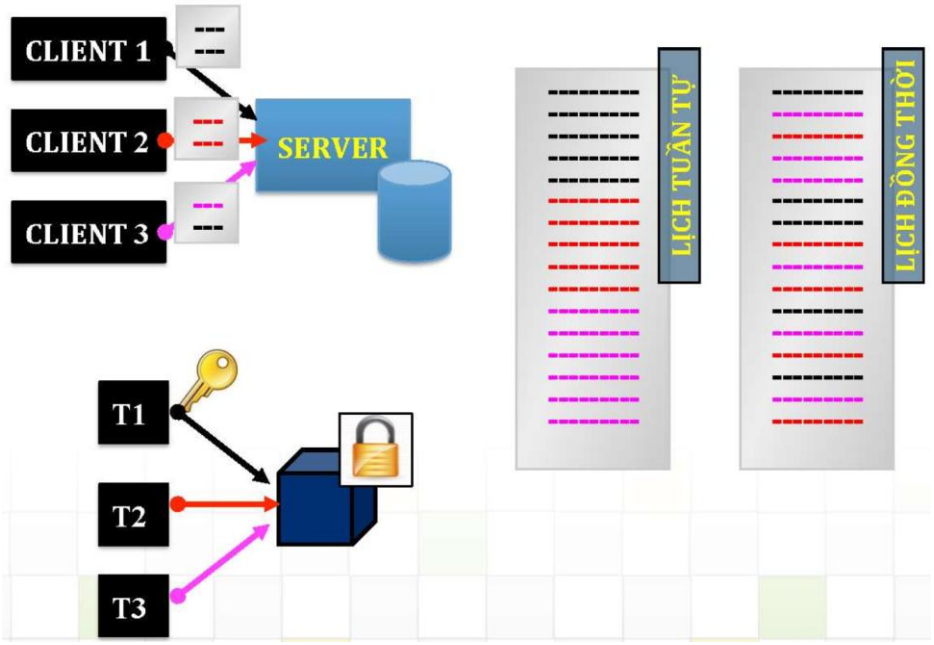
    VD: MS SQL Server cung cấp ngôn ngữ Transacion SQL (T-SQL)
  - **Các loại ngôn ngữ sử dụng trong HQTCSDL:**
    - *Ngôn ngữ định nghĩa dữ liệu (DDL - Data Definition Language):* Giúp người dùng ra lệnh cho HQTCSDL tạo ra các cấu trúc dữ liệu của CSDL (Cách tổ chức dữ liệu và mối liên hệ giữa các đối tượng dữ liệu).

- *Ngôn ngữ thao tác CSDL (DML - Data Manupulation Language):* Giúp người dùng tích luỹ, hiệu chỉnh và khai thác dữ liệu

- **Thành phần An ninh và bảo mật**
  - **Bảo mật dữ liệu:** HQTCSDL hỗ trợ các tính năng về chứng thực, phân quyền giúp kiểm soát tốt những người dùng hợp pháp của hệ thống …
  - **An ninh dữ liệu:** HQTCSDL hỗ trợ các phương pháp mã hóa dữ liệu để ngăn chặn các tấn công của những đối tượng tin tặc (đánh cắp thông tin trên đường truyền, đánh cắp nội dung CSDL).

- **Thành phần Khôi phục sau sự cố**
  - **Mục tiêu:** Đảm bảo sự tổn thất, sai sót về mặt dữ liệu là **ít nhất** có thể.
  - **Cách tiếp cận:** Để đảm bảo tính bền vững của CSDL, **mọi thay đổi lên CSDL phải được ghi nhận lại trong nhật ký (Log)**
  - **Các thành phần hỗ trợ quá trình khôi phuc sau sự cố:**
    - ***Bộ phận quản lý nhật ký (Log manager):*** đảm bảo ghi nhận đầy đủ và chính xác **mọi thay đổi** trên CSDL vào nhật ký.
    - ***Bộ phận quản lý khôi phục sự cố (Recovery Manager):*** dựa vào nhật ký để phục hồi lại CSDL về trạng thái nhất quán trước đó (Trạng thái **thoả tất cả RBTV** của CSDL).

- **Xử lý truy xuất đồng thời**
  - **Mục tiêu:** Đảm bảo các xử lý có thể được thực hiện đồng thời mà làm **không** làm cho **dữ liệu bị mất tính nhất quán** *(vi phạm các ràng buộc toàn vẹn)*
  - **Các thành phần con:** Bộ phận quản lý giao tác (Transaction Manager & Locking Manager)
  - **Phương pháp:**
    - Sử dụng khái niệm giao tác (**transaction**) để biểu diễn một đơn vị xử lý, một giao tác bao gồm các hành động mà được **thực hiện toàn bộ hoặc không có hành động nào được thực hiện.**
    - Bộ lập lịch (**scheduler**) có nhiệm vụ lập 1 lịch thực hiện từ n giao tác không tách biệt về thời gian sao cho kết quả không vi phạm tính nhất quán của CSDL.
    - Sử dụng cơ chế khóa (**lock**) để khóa các đơn vị dữ liệu nào đó khi cần → ngăn 2 **giao tác** cùng thao tác lên 1 đơn vị dữ liệu ấy tại cùng 1 điểm → Hỗ trợ để lập lịch.

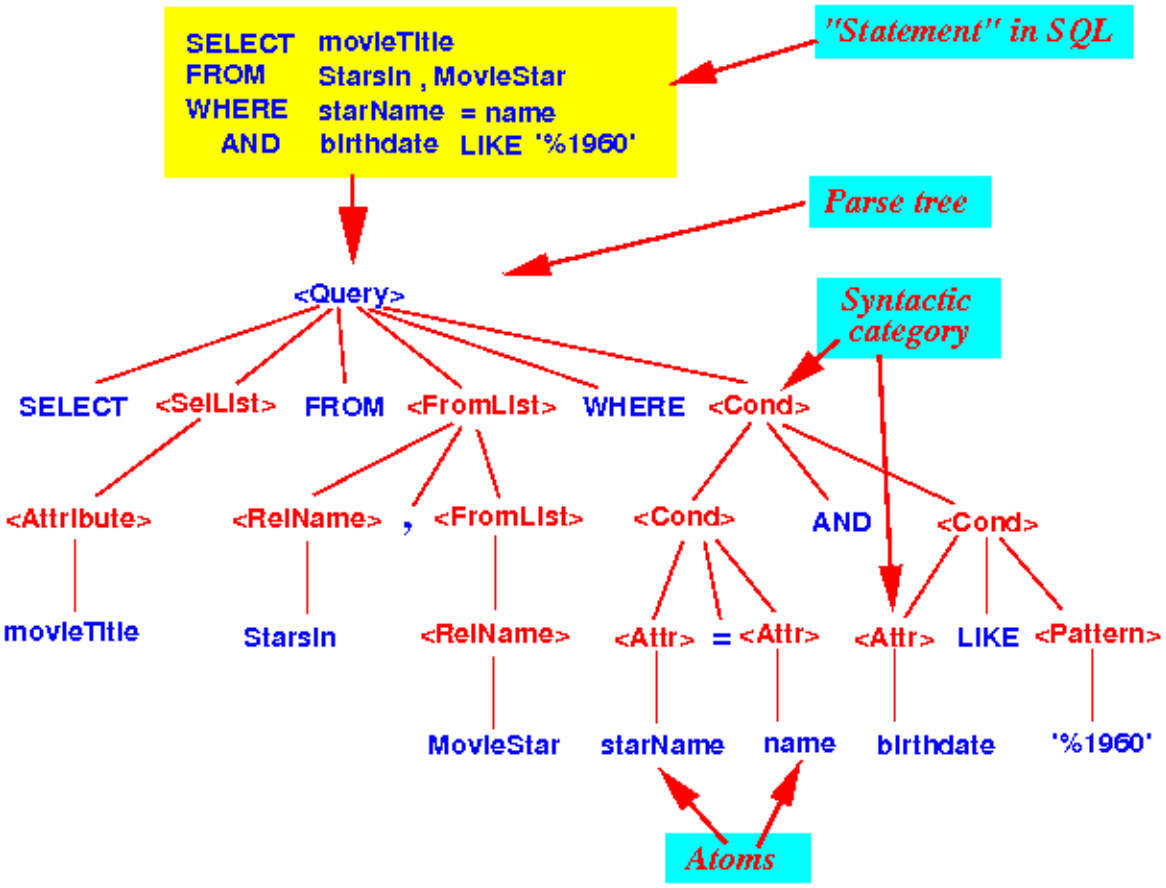


- **Vấn đề deadlock**
  - Do sử dụng cơ chế khóa nên các giao tác sẽ phải **chờ khi cần truy xuất 1 đơn vị dữ liệu đang bị khóa.**
  - Tình huống chờ vĩnh viễn mà vẫn không được truy xuất đơn vị dữ liệu bị khóa gọi là **Deadlock (khoá chết)**
    - Các giao tác chờ đợi lẫn nhau để được cấp phát tài nguyên và không giao tác nào có thể hoàn tất.
  - Thành phần quản lý giao tác sẽ phải can thiệp vào:
    - Hoặc *hủy bỏ một trong các giao tác gây deadlock*
    - Hoặc *ngăn chặn từ trước để không bao giờ xảy ra deadlock*

• Xử lý truy vấn

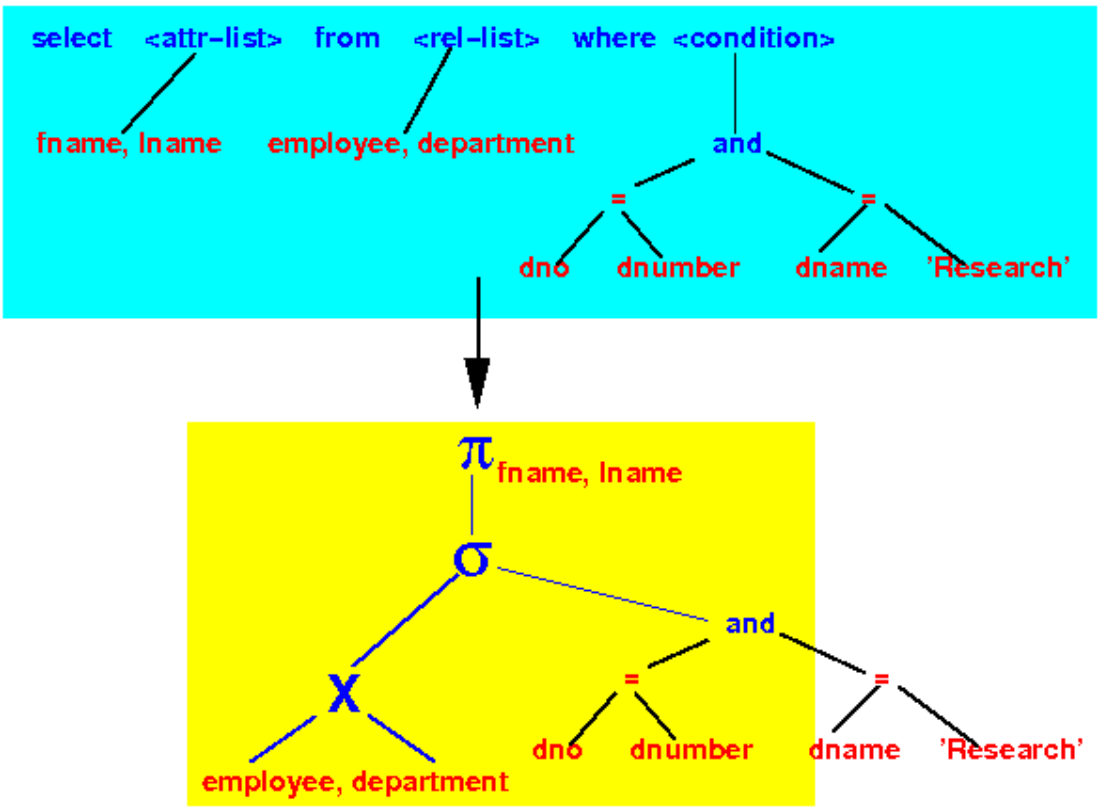
- Biểu diễn câu truy vấn ở dạng ngôn ngữ cấp cao (SQL) và thực hiện câu truy vấn có hiệu quả.
- **Query compiler** - biên dịch truy vấn, bao gồm các bước nhỏ hơn:
  - **Query parser:** Xây dựng cấu trúc phân tích câu truy vấn dưới dạng cây

    Ví dụ:



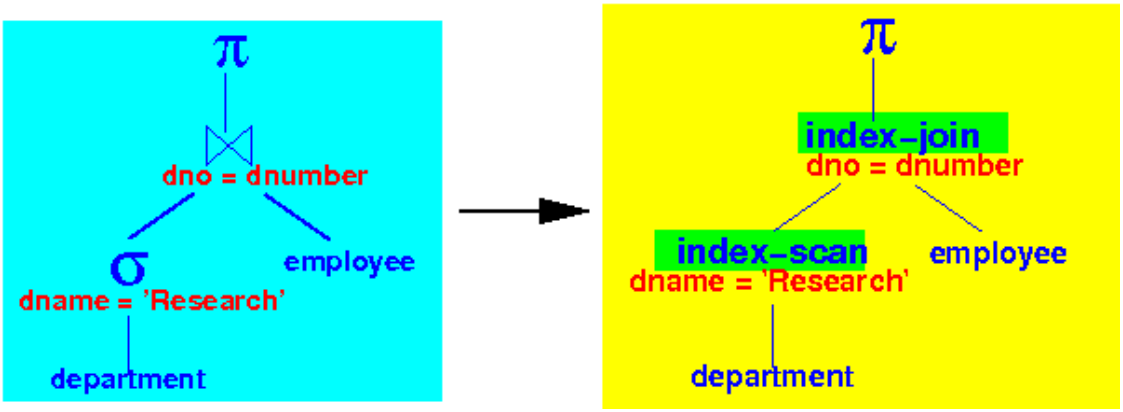
  - **Query preprocessor:**
    - Kiểm tra ngữ nghĩa của câu truy vấn
    - Chuyển đổi cấu trúc cây sang ngôn ngữ đại số quan hệ

```
select fname, lname
from employee, department
where   dno = dnumber
  and   dname = 'Research'
```



- **Query optimizer:** Sắp xếp các phép toán & lựa chọn thuật toán nhằm mục đích tối ưu hóa câu truy vấn.

  Ví dụ 1:



  Ví dụ 2:

  SELECT Ten
  FROM SinhVien
  WHERE DiemTrungBinh > 8.0 AND Lop = 'CNTT';

  → *Tạo parse tree.*

  → *Tạo ĐSQH:* π_Ten(σ_DiemTrungBinh > 8.0 ∧ Lop = 'CNTT' (SinhVien))

  → *Tối ưu:* π_Ten(σ_Lop = 'CNTT'(σ_DiemTrungBinh > 8.0 (SinhVien)))

  → *Áp dụng thuật toán tương ứng.*

- **Quản lý lưu trữ**
  - Thành phần có nhiệm vụ điều khiển việc **đọc/ghi dữ liệu qua lại** giữa **bộ nhớ** và **thiết bị lưu trữ**
  - Làm việc với các khái niệm:
    - ***Tập tin dữ liệu:*** *nơi lưu trữ thông tin thực tế (bản ghi, bảng,...) trong cơ sở dữ liệu.*
      Ví dụ: *Dữ liệu của bảng SinhVien sẽ được lưu trữ trong một tập tin vật lý như sinhvien.dat*

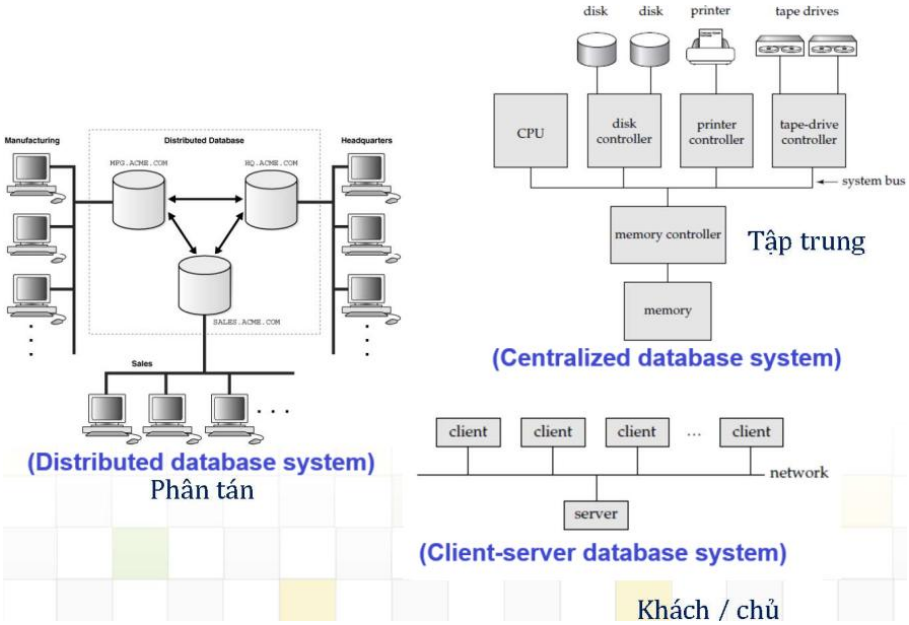
- ***Từ điển dữ liệu:*** Lưu trữ các metadata (Siêu dữ liệu) về cấu trúc của CSDL *(Các bảng, cột, kiểu dữ liệu; các ràng buộc (constraints); Quan hệ giữa các bảng)*, đặc biệt là lược đồ của CSDL

  Ví dụ: Từ điển dữ liệu lưu trữ rằng bảng SinhVien có các cột MaSV (int), Ten (varchar), DiemTrungBinh (float).

  - ***Chỉ mục:*** Giúp cho việc tìm kiếm Dữ liệu được nhanh chóng.

1.5 Phân loại HQT CSDL

- **Theo mô hình dữ liệu:** Phân cấp; Mạng; Quan hệ; Đối tượng
- **Theo kiến trúc tính toán:**



- **Theo đặc tính:**
  - ***HQTCSDL thời gian thực (real-time database system):***
    - Được thiết kế để xử lý và phản hồi các giao dịch trong **thời gian thực** hoặc gần như thời gian thực.
    - Có khả năng cập nhật và trả kết quả ngay lập tức hoặc trong khoảng thời gian giới hạn rất nhỏ.
    - Hỗ trợ các ứng dụng đòi hỏi tính tức thời và chính xác, **ví dụ:** hệ thống kiểm soát giao thông, hệ thống tài chính chứng khoán, Hệ thống theo dõi chuyến bay…
  - ***HQTCSDL chịu lỗi cao (high fault tolerance database system):***
    - Được thiết kế để duy trì hoạt động ổn định ngay cả khi xảy ra sự cố về phần cứng, phần mềm, hoặc các lỗi hệ thống khác.
    - Sử dụng các cơ chế như **replication (sao chép dữ liệu)**, **clustering**, và **backup định kỳ** để đảm bảo tính sẵn sàng và bảo vệ dữ liệu.
    - Đặc biệt quan trọng đối với các ứng dụng yêu cầu **tính liên tục cao**, không được phép dừng hoạt động. **Ví dụ:** Ngân hàng, hệ thống thanh toán trực tuyến; Hệ thống thương mại điện tử như Amazon, eBay; Dịch vụ lưu trữ đám mây (AWS, Google Cloud)…
  - ***HQTCSDL đa phương tiện (multi-media database system):***
    - Được thiết kế để lưu trữ, quản lý và truy vấn các loại dữ liệu đa phương tiện như hình ảnh, video, âm thanh, và văn bản.
    - Cung cấp khả năng xử lý các định dạng dữ liệu không phải dạng văn bản thông thường (structured data) mà là **unstructured hoặc semi-structured data**.
    - Thường tích hợp các công cụ tìm kiếm, phân tích, và xử lý dữ liệu đa phương tiện. **Ví dụ:** Hệ thống thư viện video, âm nhạc trực tuyến (YouTube, Spotify);

Lưu trữ và quản lý hình ảnh y tế (MRI, X-ray); Các nền tảng mạng xã hội (Instagram, Facebook)…

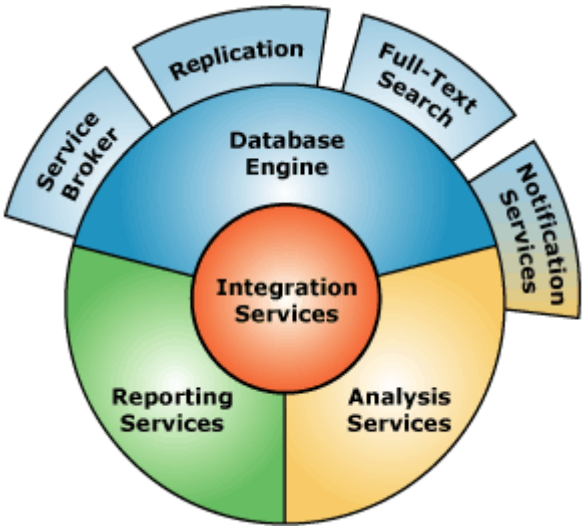
- **So sánh các loại HQTCSDL này:**

Tiêu chí	HQTCSDL Thời gian thực	HQTCSDL Chịu lỗi cao	HQTCSDL Đa phương tiện
Tốc độ phản hồi	Cao, tức thời	Phụ thuộc vào cơ chế sao lưu, chịu lỗi	Trung bình
Khả năng lưu trữ	Tập trung vào dữ liệu thời gian thực	Lưu trữ lớn, sao chép nhiều dữ liệu	Dữ liệu lớn, không cấu trúc
Mục đích sử dụng	Ứng dụng yêu cầu thời gian thực	Ứng dụng yêu cầu tính liên tục	Ứng dụng quản lý dữ liệu đa phương tiện
Công nghệ liên quan	Sensor, IoT, hệ thống phân tán	Replication, clustering, failover	AI/ML để phân tích dữ liệu

1.6 Giới thiệu về Microsoft SQL Server

- SQL Server là **một hệ quản trị cơ sở dữ liệu quan hệ (Relational Database Management System (RDBMS))** sử dụng câu lệnh SQL (Transact-SQL) để *trao đổi dữ liệu giữa máy Client và máy cài SQL Server*

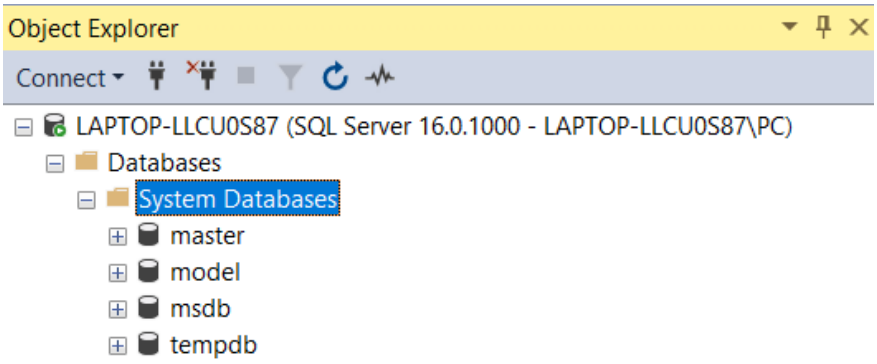
- **Các đặc điểm của SQL Server**
  - Cho phép quản trị một hệ CSDL lớn (lên đến vài tega byte), có tốc độ xử lý dữ liệu đáp ứng yêu cầu về thời gian.
  - Cho phép nhiều người cùng khai thác **trong một thời điểm** đối với một CSDL và toàn bộ quản trị CSDL (lên đến vài chục ngàn user).
  - Có hệ thống phân quyền bảo mật tương thích với hệ thống bảo mật của công nghệ NT (Network Technology), tích hợp với hệ thống bảo mật của Windows NT hoặc sử dụng hệ thống bảo vệ độc lập của SQL Server.
  - Hỗ trợ trong việc triển khai CSDL phân tán và phát triển ứng dụng trên Internet
  - Cho phép lập trình kết nối với nhiều ngôn ngữ lập trình khác dùng xây dựng các ứng dụng đặc thù (Visual Basic, C, C++, ASP, ASP.NET, XML, ...).
  - Sử dụng câu lệnh truy vấn dữ liệu Transaction-SQL (Access là SQL, Oracle là PL/SQL).
- **Các thành phần cơ bản:**



  - **DataBase Engine (Cái lõi của SQL Server):** Đây là một engine có khả năng chứa data ở những quy mô khác nhau dưới dạng table và support tất cả các kiểu kết nối thông dụng của Microsoft có thể kể đến như: ActiveX Data Objects (ADO), OLE DB và Open Database Connectivity (ODBC). Ngoài ra, Database Engine còn có khả năng tự điều chỉnh một cách hợp lý nhất
  - **Replication (Cơ chế bản sao):** Là công cụ dùng nhân bản dữ liệu, có thể sử dụng để tạo một Server khác với bộ dữ liệu giống bộ dữ liệu trên Server chính. Công cụ tạo cơ chế tự đồng bộ dữ liệu giữa Server chính và Server nhân bản.
  - **Integration Services (DTS):** Một tập hợp các công cụ đồ họa và các đối tượng lập trình cho việc *di chuyển, sao chép và chuyển đổi dữ liệu.*
  - **Analysis Services:** Một dịch vụ phân tích dữ liệu rất hay của Microsoft (phân tích kết quả, dự đoán, quy luật…)

- **Notification Services:** Dịch vụ thông báo Notification Services là nền tảng cho sự phát triển và triển khai các ứng dụng tạo và gửi thông báo. Notification Services có thể gửi thông báo theo đích thời đến hàng ngàn người đăng ký sử dụng nhiều loại thiết bị khác nhau.
- **Reporting Services:** Reporting Services bao gồm các thành phần server và client cho việc tạo, quản lý và triển khai các báo cáo. Reporting Services cũng là nền tảng cho việc phát triển và xây dựng các ứng dụng báo cáo.
- **Full Text Search Service:** Dịch vụ SQL Server Full Text Search là một dịch vụ đặc biệt cho đánh chỉ mục và truy vấn cho dữ liệu văn bản không cấu trúc được lưu trữ trong các CSDL SQL Server.
- **Service Broker:** Được sử dụng bên trong mỗi Instance, là môi trường lập trình cho việc các ứng dụng nhảy qua các Instance. Service Broker giao tiếp qua giao thức TCP/IP và *cho phép các component khác nhau có thể được đồng bộ cùng nhau theo hướng trao đổi các message*. Service Broker chạy như một phần của bộ máy cơ sở dữ **liệu, cung cấp một nền tảng truyền message tin cậy và theo hàng đợi cho các ứng dụng** SQL Server.

- **Các CSDL hệ thống của SQL Server**



- **Master:** Ghi nhận thông tin cấp hệ thống, thông tin **khởi tạo** SQL Server và các thiết lập **cấu hình** SQL Server, đồng thời cũng ghi nhận **tất cả tài khoản đăng nhập**, sự tồn tại của các **CSDL khác**, **vị trí của tập tin chính** cho tất cả các CSDL người dùng.
- **Tempdb:** Lưu trữ tạm thời tất cả các bảng và thủ tục do người dùng tạo ra. CSDL này cũng được dùng cho những ***nhu cầu lưu trữ tạm*** *khác của SQL Server* ***như sắp xếp dữ liệu*** → *Không lưu cứng trên bộ nhớ & cũng không ảnh hưởng đến hệ thống.* CSDL tempdb được tạo lại với **kích thước mặc định** của nó mỗi khi SQL được khởi động, sau đó sẽ **tự động gia tăng kích thước** khi cần.
- **Model:** Là khuôn mẫu cho tất cả các CSDL khác được tạo trên hệ thống, kể cả tempdb. CSDL model được dùng để tạo lại tempdb mỗi khi SQL server khởi động.
- **Msdb:** Lưu giữ các bảng mà SQL Server Agent dùng để tạo lập thời gian biểu thực thi các công việc, các cảnh báo và các operator - là những người chịu trách nhiệm cho cảnh báo và công việc.

- Các CSDL ta tạo ra nằm **cùng cấp với thư mục "Database"** chứ không nằm trong 4 CSDL hệ thống này.

- **Cấu trúc lưu trữ vật lý của CSDL SQL Server:** được tạo nên từ các loại file nhị phân như sau:
  - **Primary data file** dùng để lưu trữ các meta data của CSDL như định nghĩa các objects, thông số cấu hình CSDL và cả thông tin của các data file khác cũng được lưu ở đây. Ngoài ra, nó cũng được dùng để lưu user data. Mỗi database có **duy nhất một primary data file.** Để gợi nhớ thì primary file thường có đuôi là **.mdf.**
  - **Secondary data file** bao gồm tất cả các file dữ liệu khác với primary file. Một CSDL có thể không có hoặc có nhiều secondary data file. Các file này có thể nằm trên *nhiều ổ đĩa khác nhau* để tăng cường hiệu năng truy xuất dữ liệu. Để gợi nhớ loại file này có đuôi là **.ndf.**

- **Log file:** mỗi database phải có ít nhất một log file để chứa thông tin cần thiết dùng cho việc khôi phục lại những transaction đã thực hiện trong database, log file này thường có đuôi là **.ldf.**
- ***Lưu ý:*** Từ SQL Server 2008 trở đi SQL Server còn có **filestream data files** và **full-text data files.** Hai loại này phải chỉ định cụ thể và dùng cho những tình huống nhất định, nên mặc định *khi tạo mới một CSDL sẽ không có loại file này.*

1.7 SQL và T-SQL

1.7.1 SQL

❖ **SQL**, viết tắt của **Structured Query Language (ngôn ngữ truy vấn có cấu trúc),** là công cụ sử dụng để tổ chức, quản lý và truy xuất dữ liệu *được lưu trữ trong các hệ quản trị cơ sở dữ liệu quan hệ* (Relational Database Management System - RDBMS).

❖ **SQL không phải là hệ quản trị CSDL, chỉ là một phần của hệ quản trị**

❖ **Vai trò của SQL trong các HQTCSDL:**

- ***Là ngôn ngữ truy vấn có tính tương tác:*** người dùng có thể gửi các yêu cầu dưới dạng các câu lệnh SQL đến CSDL và nhận kết quả trả về từ CSDL
- Mặc dù SQL không phải là một ngôn ngữ lập trình như C, C++, Java, ... song các câu lệnh mà SQL cung cấp có thể được ***nhúng vào trong các ngôn ngữ lập trình*** nhằm xây dựng các ứng dụng tương tác với cơ sở dữ liệu.
- ***Là ngôn ngữ quản trị cơ sở dữ liệu:*** Thông qua SQL, người quản trị cơ sở dữ liệu có thể quản lý được cơ sở dữ liệu, định nghĩa các cấu trúc lưu trữ dữ liệu, điều khiển truy cập cơ sở dữ liệu, ...
- ***Là ngôn ngữ cho các hệ thống khách/chủ (client/server):*** Trong các hệ thống cơ sở dữ liệu khách/chủ, SQL được sử dụng như là công cụ để giao tiếp giữa các trình ứng dụng phía máy khách với máy chủ cơ sở dữ liệu.
- ***Là ngôn ngữ truy cập dữ liệu trên Internet:*** Cho đến nay, hầu hết các máy chủ Web cũng như các máy chủ trên Internet sử dụng SQL với vai trò là ngôn ngữ để tương tác với dữ liệu trong các cơ sở dữ liệu.
- ***Là ngôn ngữ cơ sở dữ liệu phân tán:*** Đối với các hệ quản trị cơ sở dữ liệu phân tán, mỗi một hệ thống sử dụng SQL để giao tiếp với các hệ thống khác trên mạng, gởi và nhận các yêu cầu truy xuất dữ liệu với nhau.
- ***Là ngôn ngữ sử dụng cho các cổng giao tiếp cơ sở dữ liệu:*** Trong một hệ thống mạng máy tính với nhiều hệ quản trị cơ sở dữ liệu khác nhau, SQL thường được sử dụng như là một **chuẩn ngôn ngữ** để giao tiếp giữa các HQTCSDL

- SQL tập trung vào truy vấn dữ liệu, thêm, sửa đổi và xóa dữ liệu từ các bảng.
- SQL không có khả năng lập trình hoặc điều khiển dòng, và thường được sử dụng trong các truy vấn đơn giản hoặc cơ bản.

1.7.2 T-SQL

❖ **Transact-SQL** là ngôn ngữ **SQL mở rộng** dựa trên SQL chuẩn của ISO (International Organization for Standardization) và ANSI (American National Standards Institute) **được sử dụng trong SQL Server**, khác với P-SQL (Procedural-SQL) dùng trong Oracle.

❖ It is **Microsoft's and Sybase's proprietary** (độc quyền) **extension to the SQL** used to interact with relational databases. T-SQL expands on the SQL standard to **include procedural programming, local variables, various support functions** for string processing, date processing, mathematics, etc. and changes to the DELETE and UPDATE statements.
T-SQL cung cấp:

- **Cấu trúc điều khiển (Control-of-flow):**
  - Cho phép sử dụng các câu lệnh như IF...ELSE, WHILE, và TRY...CATCH, giúp dễ dàng viết các thủ tục xử lý phức tạp.
  - Điều này vượt trội hơn SQL thông thường, vốn chỉ tập trung vào các truy vấn dữ liệu (DDL, DML).
- **Hỗ trợ Procedural Programming:**
  - T-SQL hỗ trợ các **hàm, thủ tục lưu trữ (stored procedures)** và **trigger**, giúp xây dựng các quy trình xử lý dữ liệu trực tiếp trên máy chủ cơ sở dữ liệu.
- **Tích hợp mạnh mẽ với SQL Server:**
  - T-SQL tối ưu hóa hiệu suất khi làm việc với SQL Server, đặc biệt trong các tác vụ nặng như xử lý hàng loạt hoặc thao tác dữ liệu phức tạp.

❖ Transact-SQL is **central to using Microsoft SQL Server.** Tất cả các ứng dụng giao tiếp với phiên bản SQL Server đều thực hiện việc này bằng cách gửi các câu lệnh Transact-SQL đến máy chủ, bất kể giao diện người dùng của ứng dụng.

❖ T-SQL được chia làm 3 nhóm:

- ***Ngôn ngữ định nghĩa dữ liệu (Data Definition Language - DDL)***
  - *Một trong những thành phần chính của ngôn ngữ truy vấn. Các lệnh trong DDL dùng để xây dựng, sửa đổi và xóa các đối tượng dữ liệu. Các lệnh cơ bản trong DDL là:*
    - ***Create:** **tạo** các đối tượng dữ liệu (create table <table_name>).*
    - ***Alter:** **sửa** các đối tượng dữ liệu (alter table, alter function...)*
    - ***Drop:** **xóa** các đối tượng dữ liệu (drop table, drop database).*

TẠO ĐỐI TƯỢNG	SỬA ĐỔI ĐỐI TƯỢNG	XÓA ĐỐI TƯỢNG
CREATE DATABASE	ALTER DATABASE	DROP DATABASE
CREATE DOMAIN	ALTER DOMAIN	DROP DOMAIN
CREATE INDEX	ALTER INDEX	DROP INDEX
CREATE PROCEDURE	ALTER PROCEDURE	DROP PROCEDURE
CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE TRIGGER	ALTER TRIGGER	DROP TRIGGER
CREATE VIEW		DROP VIEW

- ***Ngôn ngữ thao tác dữ liệu (Data manipulation language - DML):** cung cấp các cấu trúc lệnh để **lựa chọn thông tin từ các bảng, thêm, sửa, và xóa các dòng dữ liệu.***

Làm thay đổi dữ liệu trong bảng	Không làm thay đổi dữ liệu trong bảng
INSERT	SELECT
DELETE	
UPDATE	

- ***Ngôn ngữ điều khiển dữ liệu (Data control language - DCL):** cung cấp hai câu lệnh cho phép thiết lập các **chính sách bảo mật** trong cơ sở dữ liệu là lệnh **GRANT (cấp phát quyền cho người sử dụng) và REVOKE (Thu hồi quyền đã cấp phát).***

1.8 Microsoft SQL Server Management Studio

- SQL Server Management Studio là một công cụ trực quan để **quản lý SQL Server.** Với SQL Server Management Studio chúng ta có thể **thực hiện các tương tác với database** trên **giao diện người dùng hoặc bằng câu lệnh.**

- **Khởi động:**
  - ***Cách 1:*** Kích đúp vào biểu tượng SQL Server Management Studio
  - ***Cách 2:*** Mở Windows + R → gõ ssms

1.9 Các đối tượng trong SQL Server

1.9.1 Bảng (Table)

- Là đối tượng được Database sử dụng để tổ chức và lưu trữ dữ liệu.
- Mỗi Table trong Database có thể liên kết với một hoặc nhiều Table khác, ở một hoặc nhiều thuộc tính. Một Database bao gồm nhiều Table, giữa các Table có mối liên hệ với nhau thể hiện qua **KHÓA CHÍNH & KHÓA NGOẠI.**
- Mỗi bảng được **xác định duy nhất bởi tên bảng** → Các tên bảng không được trùng nhau.

- **Mỗi Table bao gồm:**
  - Cột hay còn gọi là các TRƯỜNG THUỘC TÍNH. Biểu diễn cho một tính chất của thực thể.
  - Dòng hay còn gọi là các BẢN GHI. Biểu diễn cho một thực thể (ứng với một đối tượng)

1.9.2 Khung nhìn (View)

- VIEW được xem như một bảng ảo trong CSDL có nội dung được định nghĩa thông qua một câu lệnh truy vấn (SELECT).
- **Tại sao được sử dụng?** Trong các loại truy vấn, ta cần một bảng tập hợp nhiều thông tin từ nhiều bảng khác, lúc đó ta không thể tạo một bảng mới và lưu tất cả các thông tin đó được, vì quy mô rất lớn. Lúc này view **giúp người sử dụng nhìn thấy được các thông tin đó được thể hiện như một bảng,** nhưng thực chất không có bảng nào được tạo, và thực hiện truy vấn trên bảng ảo đó.

1.9.3 Chỉ mục (Index)

Chỉ mục (INDEX) là bảng tra cứu đặc biệt mà công cụ tìm kiếm dữ liệu (Database Search Engine) có thể sử dụng để tăng nhanh thời gian và hiệu suất thu thập dữ liệu. Hiểu đơn giản, một chỉ mục là một con trỏ trỏ tới dữ liệu trong một bảng. Một chỉ mục trong một Database là tương tự như một chỉ mục trong Mục lục của cuốn sách.

1.10 Lập trình trên SQL Server

1.10.1 Một số quy tắc viết lệnh trong T- SQL

- Các lệnh trong câu lệnh SQL thuộc loại không phân biệt chữ viết hoa hay thường.
  - Ví dụ: SeleCT = select
- Nội dung của một câu lệnh SQL có thể được trải dài trên nhiều dòng, cũng như nhiều câu lệnh SQL kết hợp lại thành một dòng. Các khoảng trắng và ký tự tách dòng được bỏ qua.
- Các từ khoá không được phép viết tắt, viết đứt đoạn hay phân cách trên nhiều dòng
  - Ví dụ: "se Lect" là sai
- Ta có thể sử dụng các ký tự đặc biệt như: +, -, l, *, ... để biểu diễn giá trị trong câu lệnh.
  - Ví dụ: where luong + thuong > 3000
- Một số RDBMS yêu cầu kết thúc câu phải có dấu chấm phẩy, tuy nhiên một số khác lại không yêu cầu, ví dụ như SQL Server của Microsoft.
  - Ví dụ: select hoten from sinhvien;
- Chú thích trong T-SQL:
  - Chú thích cho một dòng: --
  - Chú thích cho một khối: /* */

1.10.2 Biến

- Biến là một đối tượng có thể **lưu giữ một giá trị dữ liệu**. Dữ liệu có thể được chuyển đến câu lệnh T-SQL bằng việc sử dụng biến cục bộ. Biến có thể được phân thành 2 loại: biến cục bộ và toàn cục.

1.10.2.1 Biến cục bộ

- Trong T-SQL biến cục bộ được tạo và được sử dụng cho việc **lưu trữ tạm thời trong khi câu lệnh SQL được thực hiện.** Tên của biến cục bộ phải bắt đầu với dấu '@'

- **Khai báo**

```
DECLARE { @local_variable [AS] data_type }
```

Trong đó: @local_variable: xác định **tên của biến**, tên của biến phải bắt đầu với 1 dấu '@'.
Data_type: là kiểu dữ liệu được định nghĩa bởi **người sử dụng hoặc hệ thống.**

- Câu lệnh **SET** hoặc **SELECT** được sử dụng để **gán giá trị đến cho biến cục bộ.**

```
SET @local_variable=value
SELECT @local_variable=value
```

  - Lệnh SET chỉ để gán **giá trị cụ thể** hoặc các **biểu thức tính toán** hoặc **giá trị tính toán từ các biến khác**, hoặc **tên của một biến khác;**
  - ngược lại lệnh SELECT dùng để gán các **giá trị được lấy ra hoặc tính toán** từ **dữ liệu của các cột bên trong các bảng** dữ liệu.
- Giá trị gán cho biến phải phù hợp với kiểu dữ liệu của biến.

Ví dụ 1: để gán giá trị ngày 25/1/2007 vào biến ngày xuất hàng

```
DECLARE @ngayxh DATETIME
SET @ngayxh = '25-1-2007'

Set @MaLop = „TH2001"
Set @SoSV = (select count (*) from SinhVien) --Câu truy vấn phải trả ra đúng 1 dòng có đúng 1 cột
Set @MaLop = „TH"+Year(@NgayTuyenSinh)
```

Ví dụ 2:

```
DECLARE @TongSLDat int
SELECT @TongSLDat = SUM(SLDAT) FROM CTHDON
```

Ví dụ 3:

```
DECLARE @MinSLDat int, @MaxSLDat int
SELECT @MinSLDat = MIN(SLDAT), @MaxSLDat = MAX(SLDAT), FROM CTHDON
```

Ví dụ 4:

```
SV(MaSV: int; HoTen: nvarchar(30), Tuoi int)
Select @Var1 = HoTen, @Var1 = Tuoi from SV where MaSV = 1
```

Lưu ý: nếu câu truy vấn trả về nhiều dòng, **các biến chỉ nhận giá trị tương ứng của dòng đầu tiên**

1.10.2.1.1 Xem giá trị hiện hành của biến

Cú pháp: PRINT @Tên_biến hoặc PRINT Biểu_thức_chuỗi

Ví dụ:

```
DECLARE @MinSLDat int, @MaxSLDat int
SELECT @MinSLDat = MIN(SLDAT), @MaxSLDat = MAX(SLDAT), FROM CTHDON
PRINT “Số lượng đặt thấp nhất là: ”
PRINT @MinSLDat
PRINT “Số lượng đặt cao nhất là: ”
PRINT @MaxSLDat
```

1.10.2.1.2 Phạm vi hoạt động của biến

- Phạm vi hoạt động của biến **chỉ nằm trong một thủ tục nội tại (stored procedure) hoặc một lô (batch)** chứa các câu lệnh mà **biến đã được khai báo bên trong đó.**
- Lô được xem như một nhóm **tập hợp của một hoặc nhiều câu lệnh** Transaction-SQL sẽ được **biên dịch đồng thời** tại SQL Server và sau đó hệ thống sẽ **thực thi các câu lệnh này ngay sau khi đã biên dịch thành công**
- Một từ khoá **GO** chỉ định **kết thúc 1 lô**
- Do các câu lệnh trong một lô được biên dịch tại SQL Server vì thế **khi có ít nhất 1 lệnh bên trong lô có lỗi về cú pháp** lúc **biên dịch** thì **hệ thống sẽ không có lệnh nào được thực thi bên trong lô đó.**

Ví dụ:

```
SELECT * FROM NHACC
ORDER BY TenNhaCC
INSERT INTO NHACC VALUES (‘C01’, ‘Nguyen Van A’, ’87 Ly Tu Trong’, ‘0903.123456’)
SELECT * FROM VATTU
ORDER BY Tenvtu DESC
GO
```

→ **thiếu từ khoá VALUES** thì các lệnh SELECT bên trong lô này không được thực hiện.

- Đối với các lỗi khi thực hiện (run-time) bên trong 1 lô nếu trường hợp các **lỗi vi phạm ràng buộc toàn vẹn dữ liệu** thì hệ thống SQL Server chỉ **ngưng lại câu lệnh gây lỗi và thực hiện tiếp các lệnh bên trong lô đó.**

Ví dụ:

```
SELECT * FROM NHACC
ORDER BY TenNhaCC

INSERT INTO NHACC
VALUES (‘C01’, ‘Nguyen Van A’, ’87 Ly Tu Trong’, ‘0903.123456’)

SELECT * FROM VATTU
ORDER BY Tenvtu DESC
GO
```

→ mặc dù vi phạm ràng buộc toàn vẹn trong INSERT **(giả sử trùng khoá chính ở cột MaNCC)** nhưng **các lệnh SELECT bên trong lô này vẫn được thực hiện bình thường.**

Ví dụ:

```
DECLARE @NgayDH datetime
SELECT @NgayDH = MAX(NGAYDH) FROM DONDH
GO
PRINT “ngay dat hang gan nhat: ” + convert(char(12), @ngaydh)
GO
```

Hệ thống sẽ báo lỗi vì có thêm từ khoá GO ở giữa 2 lệnh SELECT và PRINT. Bởi vì khi đó các lệnh này được **chia làm 2 lô và lô thứ hai sẽ không hiểu biến @ngaydh đã được khai báo trong lô thứ 1.**

→ Các batch tách biệt nhau không chia sẻ biến cục bộ.

1.10.2.2 Biến hệ thống/ Biến toàn cục

- Các biến hệ thống trong SQL Server luôn **bắt đầu bằng 2 chữ @@** và giá trị mà chúng ta đang lưu trữ do hệ thống SQL cung cấp, người lập trình **không can thiệp trực tiếp để gán giá trị** vào các biến hệ thống.
- Bản chất là 1 hàm (function)

Tên biến	Kiểu trả về	Dùng để trả về
Connections	Số nguyên	**Tổng số các kết nối** vào SQL Server từ khi nó được khởi động.
Error	Số nguyên	**Số mã lỗi** của câu lệnh thực hiện gần nhất. Khi một lệnh thực hiện thành công thì biến này có giá trị là 0.
Language	Chuỗi	**Tên ngôn ngữ mà hệ thống SQL đang sử dụng.** Mặc định là US_English.
RowCount	Số nguyên	**Tổng số mẩu tin** được tác động vào câu lệnh truy vấn **gần nhất.**
ServerName	Chuỗi	**Tên của máy tính cục bộ** được cài đặt trong SQL Server.
ServiceName	Chuỗi	**Tên dịch vụ kèm theo** bên dưới SQL Server.
Fetch_Status	Số nguyên	**Trạng thái** của việc đọc dữ liệu trong bảng theo **cơ chế dòng mẩu tin (cursor).** Khi dữ liệu đọc mẩu tin thành công thì biến này có giá trị là 0.
Version	Chuỗi	**Phiên bản,** ngày của sản phẩm SQL Server và loại CPU.

Ví dụ:

```
SELECT * FROM NHACC
SELECT @@ROWCOUNT
```

(trả về **tổng số mẩu tin đang hiện có trong bảng NHACC**)

Ví dụ:

```
UPDATE VATTU
SET PHANTRAM = PHANTRAM + WHERE MAVTU like “TV%”
SELECT @@ROWCOUNT
```

(Trả về **tổng số mẩu tin có MAVTU bắt đầu bằng chữ “TV”** trong **bảng VATTU**)

1.10.3 Các câu lệnh truy vấn dữ liệu

- Truy vấn lồng là những câu lệnh mà trong thành phần **WHERE** có chứa thêm một câu lệnh **SELECT** khác nữa.
- Nói cách khác, truy vấn lồng là câu truy vấn mà bên trong của nó chưa 1 câu truy vấn khác
- Câu lệnh này thường gặp khi dữ liệu cần thiết phải duyệt qua nhiều lần.

+ **Lồng phân cấp**: Mệnh đề **WHERE** của câu truy vấn con **không tham chiếu** đến thuộc tính các mối quan hệ trong mệnh đề **FROM** ở câu truy vấn cha ***(cha-con độc lập thực hiện).*** Khi thực hiện, **câu truy vấn con sẽ được thực hiện trước.**

- **Bài toán**: Tìm danh sách các nhân viên có mức lương cao nhất trong công ty.

```
+----+-------------+--------------+--------+
| ID | Name        | DepartmentID | Salary |
+----+-------------+--------------+--------+
| 1  | John Smith  | 101          | 5000   |
| 2  | Alice Doe   | 101          | 5000   |
| 3  | Bob Johnson | 102          | 4500   |
| 4  | Jane Doe    | 102          | 4500   |
| 5  | Mike Brown  | 103          | 4000   |
| 6  | Sarah White | 103          | 3800   |
+----+-------------+--------------+--------+
```

```
SELECT Name, Salary
FROM Employees
WHERE Salary = (SELECT MAX(Salary) FROM Employees);
```

- **Giải thích**:
  - Subquery (SELECT MAX(Salary) FROM Employees) được thực hiện **trước**, trả về mức lương cao nhất, ví dụ: 5000.
  - Truy vấn cha sau đó so sánh giá trị Salary của từng hàng với 5000 để tìm nhân viên phù hợp.
  - **Kết quả**:

```
+----+-------------+--------------+--------+
| ID | Name        | DepartmentID | Salary |
+----+-------------+--------------+--------+
| 1  | John Smith  | 101          | 5000   |
| 2  | Alice Doe   | 101          | 5000   |
+----+-------------+--------------+--------+
```

+ **Lồng tương quan**: Mệnh đề **WHERE** của câu truy vấn con **tham chiếu ít nhất một thuộc tính** của các quan hệ trong mệnh đề **FROM ở câu truy vấn cha.**

- **Bài toán**: Tìm danh sách nhân viên có mức lương cao nhất trong từng phòng ban.

```
SELECT e1.Name, e1.DepartmentID, e1.Salary
FROM Employees e1
WHERE e1.Salary = (
    SELECT MAX(e2.Salary)
    FROM Employees e2
    WHERE e2.DepartmentID = e1.DepartmentID
);
```

- **Giải thích**:
  - Subquery tham chiếu e1.DepartmentID từ câu truy vấn cha (e1 là bí danh của bảng Employees trong truy vấn cha).
  - Với mỗi hàng trong truy vấn cha, subquery được thực hiện để tìm mức lương cao nhất trong phòng ban tương ứng.
  - Quá trình lặp lại cho mỗi hàng của e1.
  - **Kết quả**:

```
+----+-------------+--------------+--------+
| ID | Name        | DepartmentID | Salary |
+----+-------------+--------------+--------+
| 1  | John Smith  | 101          | 5000   |
| 3  | Bob Johnson | 102          | 4500   |
| 5  | Mike Brown  | 103          | 4000   |
+----+-------------+--------------+--------+
```

- **Truy vấn con: chỉ là một câu lệnh truy vấn lựa chọn (SELECT) được lồng vào các câu lệnh truy vấn khác** nhằm thực hiện các truy vấn tính toán phức tạp. Khi sử dụng đến truy vấn con chúng ta cần lưu tâm đến một vài yếu tố sau:
  - Cần mở và đóng ngoặc đơn cho câu lệnh truy vấn con.
  - Chúng ta **chỉ được phép tham chiếu đến tên một cột hoặc một biểu thức sẽ trả về giá trị** trong truy vấn con.
  - Kết quả truy vấn con có thể trả về là **một giá trị đơn lẻ hoặc một danh sách các giá trị.**
  - **Cấp độ lồng nhau** của các truy vấn con bên trong SQL Server là **không giới hạn.**

- **Truy vấn con trả về một giá trị đơn:** là truy vấn mà kết quả trả về của nó luôn đảm bảo chỉ là một giá trị đơn.

Ví dụ: để biết được danh sách các đơn đặt hàng gần đây nhất.

```
SELECT MAX(NGAYDH)
FROM DONDH
```

Kết quả trả về: 2017-01-25 00:00:00

```
SELECT *
FROM DONDH
WHERE NGAYDH = “2017-01-25”
```

Kết hợp 2 câu truy vấn trên

```
SELECT *
FROM DONDH
WHERE NGAYDH = (SELECT MAX(NGAYDH)
                FROM DONDH)
```

- **Truy vấn con trả về danh sách các giá trị:** trả về của nó là danh sách các giá trị hay còn gọi là **một tập hợp các phần tử.** Toán tử **IN** sẽ được sử dụng để so sánh truy vấn con dạng này

Ví dụ 1: để biết nhà cung cấp nào mà công ty đã đặt hàng trong tháng 01/2017.

```
SELECT MaNCC
FROM DONDH
WHERE Convert(char(7), NgayDH, 21) = ‘2017-01’
```

Kết quả trả về

MaNCC
C03
C01

Hàm Convert:

- Dùng để chuyển đổi một giá trị sang kiểu dữ liệu cụ thể.

- Trong trường hợp này, NgayDH được chuyển thành kiểu chuỗi (char(7)), giữ lại 7 ký tự đầu tiên từ định dạng ngày.

Định dạng kiểu 21:

- Định dạng ngày kiểu 21 trong SQL Server biểu thị kiểu **YYYY-MM-DD HH:MI:SS.mmm** (năm-tháng-ngày giờ:phút:giây.millisecond).
- Khi chỉ lấy 7 ký tự đầu tiên (theo char(7)), kết quả sẽ là **YYYY-MM**.

```
SELECT TenNCC, DienThoai
FROM NHACC
WHERE MaNCC IN('C01', 'C03')
```

Toán tử IN được sử dụng trong SQL để kiểm tra xem một giá trị có nằm trong một tập hợp giá trị cụ thể hay không. Nó thường được sử dụng trong mệnh đề WHERE để thay thế cho nhiều điều kiện OR.

- Tìm tất cả các nhân viên có mức lương là 4000, 4500 hoặc 5000.

```
SELECT Name, Salary
FROM Employees
WHERE Salary IN (4000, 4500, 5000);
```

- **Kết quả**:

```
+-------------+--------+
| Name        | Salary |
+-------------+--------+
| John Smith  | 5000   |
| Alice Doe   | 5000   |
| Bob Johnson | 4500   |
| Jane Doe    | 4500   |
| Mike Brown  | 4000   |
+-------------+--------+
```

Nếu danh sách trong IN chứa NULL, hãy cẩn thận, vì SQL sẽ trả về FALSE khi so sánh bất kỳ giá trị nào với NULL.
Truy vấn với IN có thể kém hiệu quả hơn so với các phương pháp khác (như JOIN), đặc biệt khi danh sách giá trị lớn.

- Không đảm bảo rằng trong tháng 01/2017 công ty chỉ đặt hàng cho 2 nhà cung cấp C01 và C03. Do đó để **luôn luôn có được danh sách họ tên các nhà cung cấp mà công ty đã đặt trong tháng 01-2017** chúng ta thực hiện truy vấn con sau:

```
SELECT TenNCC, DienThoai
FROM NHACC
WHERE MaNCC IN(SELECT MaNCC
FROM DONDH
WHERE Covert(char(7), NgayDH, 21) = '2017-01' )
```

- **Hoặc dùng EXISTS**

```
SELECT TenNCC, DienThoai
FROM NHACC
WHERE EXISTS (SELECT *
FROM DONDH
WHERE Covert(char(7), NgayDH, 21) = "2017-01" AND NHACC.MaNCC = DONDH.MaNCC)
```

- **WHERE EXISTS**: Kiểm tra xem có **ít nhất một dòng** trong bảng DONDH thỏa mãn điều kiện không. Nếu có, điều kiện EXISTS trả về TRUE và nhà cung cấp tương ứng được chọn.
- **NHACC.MaNCC = DONDH.MaNCC**: Kết nối giữa bảng NHACC và bảng DONDH dựa trên khóa ngoại MaNCC.

- Trong câu truy vấn sử dụng IN, **kết nối tường minh (explicit join)** giữa bảng NHACC và DONDH không cần thiết vì bản thân **toán tử IN** đã ngầm định tạo mối liên kết giữa hai bảng.
- Trong câu truy vấn EXISTS, điều kiện kiểm tra có dòng nào thỏa mãn liên kết NHACC.MaNCC = DONDH.MaNCC trong truy vấn con hay không. Kết nối là bắt buộc vì EXISTS chỉ quan tâm đến việc tồn tại dòng thỏa mãn điều kiện.
- If the argument sub-query is non-empty, exists construct returns the value true, otherwise false. To check whether a row is returned through this sub-query or not, it is used. True is returned if **one or more rows are returned by executing the sub-query,** otherwise False when **no rows are returned.**

Ví dụ 2: Để biết danh sách các nhà cung cấp nào mà **công ty chưa bao giờ đặt hàng.** Chúng ta có thể thực hiện câu truy vấn như sau:

```
SELECT TenNhaCC, DienThoai
FROM NHACC
WHERE MaNCC NOT IN (SELECT Distinct MaNCC FROM DONDH)
```

Hoặc

```
SELECT TenNhaCC, DienThoai
FROM NHACC
WHERE MaNCC <> ALL (SELECT Distinct MaNCC FROM DONDH)
```

Lưu ý:

IN tương đương = ANY
NOT IN tương đương <>ALL

DISTINCT là từ khóa được sử dụng trong SQL để **loại bỏ các dòng trùng lặp** trong kết quả của câu truy vấn, đảm bảo **mỗi dòng trong kết quả chỉ xuất hiện một lần.**

Cú pháp:

```
SELECT DISTINCT column1, column2, ... FROM table_name;
```

SELECT DISTINCT MaNCC FROM DONDH: Trả về *danh sách các mã nhà cung cấp (MaNCC) đã xuất hiện trong bảng DONDH.* Việc sử dụng DISTINCT **đảm bảo mỗi mã chỉ xuất hiện một lần,** ngay cả khi nhà cung cấp có nhiều đơn đặt hàng.

1.10.4 Lệnh INSERT

Cách 1: Thêm trực tiếp một bộ

```
INSERT INTO bảng [<cột 1>, <cột 2>,…, cột n]
VALUES (<giá trị 1>, <giá trị 2>,…, <giá trị n>)
```

Ví dụ: thêm dữ liệu vào HOCVIEN

```
INSERT INTO HOCVIEN(MaHV, TenHV, NgaySinh, Malop)
VALUES('001', 'Nguyen Van Tam', '05/06/1986', 'CT01')
```

Cách 2: Thêm nhiều bộ giá trị lấy từ các bộ giá trị của các bảng khác của CSDL

```
INSERT INTO bảng[<cột 1>, <cột 2>,…, <cột n>]
SELECT … FROM …. WHERE
```

Ví dụ:

```
INSERT INTO LOP-HV(TenLop, SiSo)
SELECT LOP.TenLop, count(*)
FROM HOCVIEN
JOIN LOP ON HOCVIEN.MaLop = LOP.MaLop
GROUP BY LOP.TenLop;
```

- **INSERT INTO LOP_HV (TenLop, SiSo)**
  - Chỉ định bảng đích là LOP_HV và các cột cụ thể cần thêm dữ liệu (TenLop, SiSo).
- **SELECT LOP.TenLop, COUNT(*)**
  - LOP.TenLop: Lấy tên lớp từ bảng LOP.
  - COUNT(*): Đếm số lượng học viên trong từng lớp từ bảng HOCVIEN.
- **FROM HOCVIEN JOIN LOP ON HOCVIEN.MaLop = LOP.MaLop**
  - Sử dụng phép nối (JOIN) để kết nối hai bảng HOCVIEN và LOP dựa trên cột chung MaLop.
- **GROUP BY LOP.TenLop**
  - Nhóm các kết quả theo tên lớp (TenLop) để tính tổng số học viên (SiSo) cho từng lớp.

1.10.5 Lệnh UPDATE

Cú pháp:

```
UPDATE <tên bảng>
SET <cột 1> = <giá trị 1>,
<cột 2> = <giá trị 2>,
……,
<cột n> = <giá trị n>
[WHERE <biểu thức điều kiện>]
```

<biểu thức điều kiện> có thể là

$$\begin{bmatrix} WHERE\ col1 = 100; \\ WHERE\ col2\ IN\ (1,2,3); \\ WHERE\ col3\ LIKE\ 'An\%' \\ WHERE\ col4\ BETWEEN\ 10\ AND\ 20; \end{bmatrix}$$

$$WHERE\ col3\ LIKE\ 'An\%'$$

- Lọc các giá trị của cột col3 bắt đầu bằng chữ "An".
- Ký tự % là ký tự đại diện trong SQL, biểu thị bất kỳ chuỗi nào (bao gồm cả chuỗi rỗng) theo sau chữ "An".

$$WHERE\ col4\ BETWEEN\ 10\ AND\ 20;$$

- Lọc các giá trị của cột c2 nằm trong khoảng từ 10 đến 20 (bao gồm cả 10 và 20).
- Toán tử BETWEEN bao gồm cả giá trị giới hạn.

Ví dụ: Sửa nhân viên Dư Thanh Loan ở phòng HC sang phòng KD

```
UPDATE NHANVIEN
SET Phong= “KD”
```

```
WHERE HoTen = “Dư Thanh Loan”
```

1.10.6 Lệnh DELETE

Cú pháp:

```
DELETE [FROM] Bảng
[WHERE <điều kiện>]
```

Ví dụ: Xóa tất cả các nhân viên có LCB<700

```
DELETE FROM NHANVIEN
WHERE LCB<700
```

1.10.7 Biểu thức CASE

Cú pháp:

```
Case biểu_thức:
        When giá_trị_1|Bt_logic_1 Then Biểu_thức_kết_quả_1
        [When giá_trị_2|Bt_logic_2 Then Biểu_thức_kết_quả_2…]
        [ELSE biểu_thức_kết_quả_N]
End
```

Ví dụ 1: hiển thị danh sách các vật tư có trong bảng VATTU **theo từng loại hàng**

```
SELECT MAVTU, TenVTU,
      Loai = CASE
            WHEN LEFT(MAVTU, 2) = 'DD' THEN 'Đầu DVD'
            WHEN LEFT(MAVTU, 2) = 'VD' THEN 'Đầu VCD'
            WHEN LEFT(MAVTU, 2) = 'TV' THEN 'Tivi'
            WHEN LEFT(MAVTU, 2) = 'TL' THEN 'Tủ lạnh'
            WHEN LEFT(MAVTU, 2) = 'LO' THEN 'Loa thùng'
            ELSE 'chưa phân loại'
          END,
      DVTinh
FROM VATTU;
```

Cách này dễ bảo trì hơn cách dưới đây.

Hoặc

```
SELECT MAVTU, TenVTU, Loai = CASE LEFT(MAVTU,2)
   WHEN 'DD' THEN 'Đầu DVD'
   WHEN 'VD' THEN 'Đầu VCD'
   WHEN 'TV' THEN 'Tivi'
   WHEN 'TL' THEN 'Tủ lạnh'
   WHEN 'LO' THEN 'Loa thùng'
   ELSE 'chưa phân loại'
END, DVTinh
FROM VATTU
```

LEFT(): Hàm LEFT(chuỗi, độ_dài) trong SQL Server được sử dụng để trích xuất một số lượng ký tự nhất định **từ bên trái của một chuỗi.** Trong trường hợp này, LEFT(MAVTU, 2) sẽ lấy hai ký tự đầu tiên từ cột MAVTU.

Ví dụ 2: Hiển thị danh sách các vật tư trong bảng VATTU, thông tin **bổ sung thêm chuỗi ghi chú, tuỳ thuộc vào giá trị của cột phần trăm giá bán.**

```
SELECT MaVT, TenVT, DVTinh, PhanTram,
GhiChu = CASE
        WHEN PhanTram < 20 Then “Lời ít”
        WHEN PhanTram Between 20 And 40 Then “Lời nhiều”ELSE “Rất lời”
        END
FROM VATTU
```

Ví dụ 3: Giảm giá bán hàng **trong tháng 2-2017** theo quy tắc sau: Nếu số lượng hàng <= 2 thì không giảm giá, Nếu số lượng hàng từ 3 đến 10 thì giảm 10%, Nếu số lượng hàng > 10 thì giảm 20%

```
UPDATE CTPXUAT
SET DGXuat = CASE
   WHEN SLXUAT <= 2 THEN DGXuat
   WHEN SLXUAT BETWEEN 3 AND 10 THEN DGXuat * 0.9
   ELSE DGXuat * 0.8
END
FROM CTPXUAT
INNER JOIN PXUAT ON CTPXUAT.SoPX = PXUAT.SoPX
WHERE CONVERT(char(7), NgayXuat, 23) = '2017-02'
```

Hoặc

```
UPDATE CTPXUAT
SET DGXuat = CASE
   WHEN SLXUAT <= 2 THEN DGXuat
   WHEN SLXUAT BETWEEN 3 AND 10 THEN DGXuat * 0.9
   ELSE DGXuat * 0.8
END
FROM CTPXUAT A, PXUAT B
WHERE A.SoPX = B.SoPX
AND CONVERT(char(7), B.NgayXuat, 23) = '2017-02'
```

1.10.8 Cấu trúc điều khiển

1.10.8.1 Cấu trúc rẽ nhánh IF…ELSE

```
IF Biểu_thức_luận_lý
        Câu_lệnh1|khối_lệnh1
ELSE
        Câu_lệnh2|khối_lệnh2
```

Ví dụ: Cho biết **có vật tư nào đã bán ra với số lượng nhiều hơn 4 không**? Nếu có thì hiển thị danh sách đó ra, ngược lại thì thông báo chưa bán vật tư nào nhiều hơn 4

```
IF (SELECT COUNT(*) FROM CTPXUAT WHERE SLXUAT > 4) > 0
BEGIN
        PRINT 'Danh sách các hàng hoá bán ra với số lượng lớn hơn 4'
        SELECT CTPXUAT.MAVT, TENVT, SLXUAT
        FROM CTPXUAT A , VATTU B
        WHERE A.MaVT = B.MaVT
        AND A.SLXUAT>4
END
```

```
ELSE
        PRINT 'Chưa bán hàng nào với số lượng lớn hơn 4'
```

SELECT COUNT(*) FROM CTPXUAT WHERE SLXUAT > 4:

- COUNT(*) đếm số dòng trong bảng CTPXUAT mà thỏa mãn điều kiện SLXUAT > 4.
- WHERE SLXUAT > 4 lọc ra các bản ghi có giá trị trong cột SLXUAT lớn hơn 4.

Kết quả của SELECT COUNT(*):

- COUNT(*) trả về một giá trị số nguyên, cụ thể là số lượng các bản ghi thỏa mãn điều kiện SLXUAT > 4.
- Ví dụ, nếu có 5 bản ghi trong bảng CTPXUAT có giá trị SLXUAT > 4, câu lệnh này sẽ trả về giá trị 5.

IF ... > 0:

- giúp kiểm tra giá trị trả về từ truy vấn con.
- Điều kiện > 0 kiểm tra xem số lượng bản ghi thỏa mãn điều kiện (SLXUAT > 4) có lớn hơn 0 không.
    - Nếu số lượng bản ghi thỏa mãn điều kiện là lớn hơn 0, điều kiện IF sẽ **đúng** và thực thi các câu lệnh trong khối BEGIN...END.
    - Nếu không có bản ghi nào thỏa mãn điều kiện SLXUAT > 4, điều kiện IF sẽ **sai** và thực thi câu lệnh trong phần ELSE (nếu có).

Trong SQL Server, **BEGIN** và **END** được sử dụng để nhóm các câu lệnh lại với nhau thành một khối lệnh (block of statements). Điều này giúp đảm bảo rằng nhiều câu lệnh sẽ được thực thi trong một ngữ cảnh duy nhất, đặc biệt trong các câu lệnh điều kiện như IF, vòng lặp WHILE, hoặc khi bạn thực hiện các giao dịch (transactions).

- Cú pháp IF kết hợp với từ khoá EXISTS dùng để kiểm tra sự tồn tại của các dòng dữ liệu bên trong bảng.

Cú pháp:

```
IF EXISTS(Câu_lệnh_SELECT)
        Câu_lệnh1|khối_lệnh1
ELSE
        Câu_lệnh2|khối_lệnh2
```

Ví dụ:

```
IF EXISTS(SELECT COUNT(*) FROM CTPXUAT WHERE SLXUAT>4)
BEGIN
        Print ‘Danh sách các hàng hoá bán ra với số lượng lớn hơn 4’
        SELECT CTPXUAT.MAVT, TENVT, SLXUAT
        FROM CTPXUAT A, VATTU B
        WHERE A.MaVT = B.MaVT
        AND A.SLXUAT>4
END
        ELSE
        Print ‘chưa bán hàng nào với số lượng lớn hơn 4’
```

1.10.8.2 Cấu trúc lặp WHILE

Cú pháp:

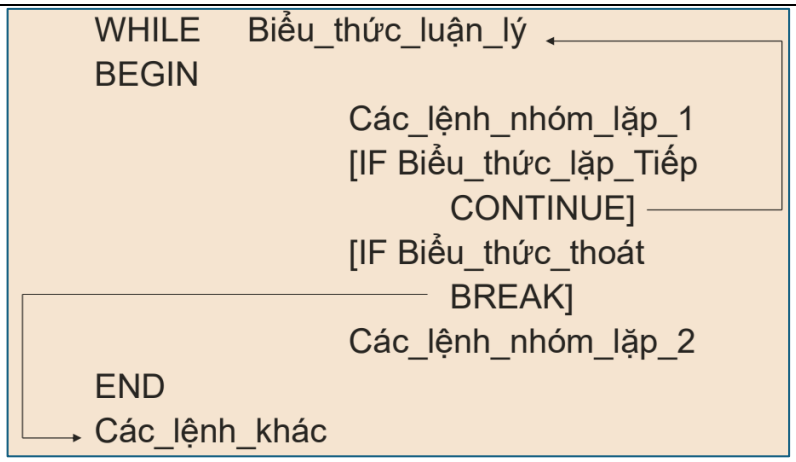
```
WHILE Biểu_thức_luận_lý
        BEGIN
              Các_lệnh_lặp
        END
```

Ví dụ 1: Để in ra 10 số nguyên dương bắt đầu từ 100.

```
DECLARE @Songuyen INT
SET @Songuyen = 100
WHILE (@Songuyen < 110)
        BEGIN
            Print 'Số nguyên: ' + convert(char(3), @songuyen)
            SET @Songuyen = @Songuyen +1
        END
```

- Từ khoá **BREAK** lồng vào cấu trúc WHILE để có thể **kết thúc việc lặp** của các lệnh bên trong vòng lặp

```
DECLARE @Songuyen int
SET @Songuyen = 100
WHILE (@Songuyen < 110)
        BEGIN
            Print "So nguyen: " + Convert(char(3), @songuyen)
            IF @Songuyen = 105
                Break
            SET @Songuyen = @Songuyen +1
        END
```



- Thực hiện giống ví dụ trước, nhưng muốn **in sót số nguyên 105.** Chúng ta sử dụng cấu trúc lặp WHILE như sau

```
DECLARE @Songuyen int
SET @Songuyen = 100
WHILE (@Soguyen < 110)
        BEGIN
            SET @Songuyen = @Songuyen + 1
            IF @Songuyen = 105
                CONTINUE
            Print "Số nguyên: " + Convert(char(3), @songuyen)
        END
```

1.10.9 Biến kiểu dữ liệu cursor

- CSDL quan hệ thường làm việc trên dữ liệu của nhiều dòng mẫu tin – còn gọi là các bộ mẫu tin. Ví dụ lệnh SELECT kết quả luôn trả về nhiều dòng dữ liệu hơn là một dòng dữ liệu.
- Tuy nhiên có một số ngôn ngữ lập trình việc xử lý và tính toán dữ liệu trên **từng dòng riêng lẻ.** Để đáp ứng được yêu cầu này SQL Server tạo ra một kiểu dữ liệu đó chính là kiểu **cursor (~ con trỏ)**.
- Cursor is a database object used by applications to manipulate data in a set on a row-by-row basis, instead of the typical SQL commands that operate on all the rows in the set at one time.

- Các bước sử dụng kiểu dữ liệu cursor
  - **Định nghĩa biến** kiểu cursor bằng lệnh **DECLARE**.
  - Sử dụng **lệnh OPEN** để **mở ra cursor** đã định nghĩa trước đó.
  - **Đọc và xử lý** trên từng dòng dữ liệu bên trong cursor.
  - **Đóng cursor bằng lệnh CLOSE và DEALLOCATE.**

- Cú pháp định nghĩa biến có kiểu cursor

```
DECLARE Tên_cursor CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | DYNAMIC | KEYSET]
[READ_ONLY | SCROLL_LOCK]
FOR Câu_lệnh SELECT
[FOR UPDATE [OF danh_sách_cột_cập_nhật]]
```

- **Trong đó:**
  - **Tên cursor:** tên của biến kiểu cursor
  - **Từ khoá LOCAL | GLOBAL:** dùng chỉ **phạm vi hoạt động** của biến cursor hoặc là cục bộ (local) bên trong một thủ tục.
  - **FORWARD_ONLY:** đọc dữ liệu trong cursor theo chiều đi tới **duyệt từ đầu mẫu tin đầu tiên đến mẫu tin cuối cùng. Không thể quay lại trước các dòng đã duyệt.**
  - **SCROLL**: đọc dữ liệu trong cursor được phép **di chuyển tới lui, qua lại các dòng mẫu tin** bên trong cursor **tuỳ thích.**
  - **STATIC**: đọc dữ liệu bên trong cursor **tĩnh**. Khi đó nếu những *người dùng khác có thay đổi bên dưới dữ liệu gốc* thì các thay đổi đó sẽ **không được cập nhật tự động** trong dữ liệu của cursor. Bởi vì khi đó **dữ liệu trong cursor chính là dữ liệu của bảng tạm** đã được hệ thống sao chép và lưu trữ trong CSDL tempdb của hệ thống khi định nghĩa cursor.
  - **DYNAMIC**: dùng chỉ định dữ liệu trong cursor là **động**. Khi đó việc *cập nhật dữ liệu trong bảng cơ sở bởi những người dùng khác sẽ **được cập nhật tự động** trong dữ liệu cursor* có kiểu là DYNAMIC.
  - **KEYSET**: **hoạt động giống với kiểu DYNAMIC**, các thay đổi dữ liệu trên các cột **không** là khoá chính trong bảng cơ sở bởi những người dùng khác sẽ được cập nhật trong dữ liệu cursor. Tuy nhiên đối với **mẫu tin vừa thêm mới hoặc các mẫu tin đã bị huỷ bỏ** bởi những người dùng khác sẽ **không được hiển thị trong dữ liệu cursor** có kiểu là KEYSET.
    - ❖ Một keyset là một *tập hợp các chỉ mục duy nhất (các key) mà cursor sẽ sử dụng để duyệt qua các dòng.*
    - ❖ Khi sử dụng KEYSET, dữ liệu của cursor sẽ không bị thay đổi khi bạn duyệt qua (ví dụ: nếu có các *thay đổi trong bảng, những thay đổi đó sẽ không được phản ánh trong cursor*).

- **READ_ONLY:** chỉ định dữ liệu trong cursor **chỉ đọc** nhằm hạn chế việc sửa đổi dữ liệu bên trong cursor. Khi khai báo cursor với kiểu dữ liệu tĩnh (**STATIC**) thì dữ liệu trong cursor xem như chỉ đọc.
- **SCROLL_LOCK:** chỉ định hệ thống SQL Server **tự động khoá các dòng mẩu tin cần phải thay đổi giá trị hoặc huỷ bỏ** bên trong bảng nhằm bảo đảm các hành động cập nhật luôn thành công.
- **Danh sách cột cập nhật:** chỉ định danh sách tên các cột sẽ được phép thay đổi giá trị trong cursor.

Ví dụ 1: để định nghĩa một biến cursor chứa toàn bộ các dòng dữ liệu bên trong bảng MAT_HANG, **các dòng dữ liệu trong cursor cho phép được cập nhật.**

```
DECLARE Cur_MAT_HANG CURSOR
DYNAMIC
FOR SELECT * FROM MATHANG
```

Ví dụ 2: Định nghĩa một biến cursor chứa toàn bộ các dòng dữ liệu bên trong bảng NHACC, các dữ liệu trong cursor **chỉ được phép đọc** và việc đọc dữ liệu trong cursor **chỉ theo một chiều đi tới.**

```
DECLARE Cur_NhaCC CURSOR
FORWARD_ONLY
STATIC
READ_ONLY
FOR SELECT * FROM NHACC
```

- **Cursor sẽ lấy một tập hợp dữ liệu** từ câu lệnh SELECT, và dữ liệu này có thể chứa **nhiều dòng**.
- **Cursor sẽ duyệt qua các dòng dữ liệu** một cách tuần tự, mỗi lần chỉ trả về một dòng (record) để xử lý.
- Bạn có thể **lặp qua từng dòng** của cursor bằng cách sử dụng các lệnh như **FETCH** để lấy các dòng dữ liệu trong cursor.

1.10.9.1 Mở Cursor

Cú pháp:

```
OPEN Tên_cursor
```

Trong đó **Tên cursor:** tên của biến cursor đã được định nghĩa trước đó bằng lệnh DECLARE

Ví dụ: Mở các cursor đã định nghĩa ở ví dụ 1 trên. Chúng ta sử dụng lệnh OPEN như sau:

```
OPEN cur_MAT_HANG
```

Việc **mở cursor** trong SQL Server là một bước quan trọng để chuẩn bị và sử dụng cursor để duyệt qua dữ liệu. Dưới đây là lý do tại sao bạn cần mở cursor trước khi sử dụng nó:

1. Cấp phát bộ nhớ và chuẩn bị dữ liệu

Khi bạn khai báo một cursor, SQL Server cần phải chuẩn bị bộ nhớ và thực thi câu lệnh SELECT mà bạn đã chỉ định trong cursor. Việc mở cursor đảm bảo rằng SQL Server sẽ:

- **Thực thi câu lệnh SELECT** để lấy dữ liệu.
- **Cấp phát bộ nhớ** để lưu trữ các dòng dữ liệu mà cursor sẽ duyệt qua.

2. Khởi tạo con trỏ (Pointer) cho dữ liệu

Khi mở cursor, một con trỏ (pointer) được tạo ra. Con trỏ này giúp bạn di chuyển qua các dòng dữ liệu trong tập kết quả của câu lệnh SELECT. Nếu không mở cursor, không có con trỏ nào để di chuyển qua các dòng dữ liệu.

3. Bắt đầu duyệt qua dữ liệu

Sau khi cursor được mở, bạn có thể bắt đầu sử dụng các lệnh như FETCH để lấy dữ liệu từng dòng một từ cursor.

4. Giúp SQL Server xác định phạm vi dữ liệu cần duyệt

Khi bạn mở cursor, SQL Server sẽ bắt đầu theo dõi dòng dữ liệu mà cursor đã chọn và có thể xử lý dữ liệu một cách tuần tự. Điều này giúp:

- **Giới hạn phạm vi**: SQL Server chỉ giữ dữ liệu cần thiết trong bộ nhớ và chỉ duyệt qua các dòng được chọn bởi câu lệnh SELECT của cursor.
- **Cải thiện hiệu suất**: Dữ liệu không cần được xử lý đồng thời hoặc toàn bộ, chỉ cần duyệt từng dòng khi cần thiết.

5. Bắt đầu các thao tác với cursor

Sau khi cursor được mở, bạn có thể thực hiện các thao tác với dữ liệu được trả về từ cursor, như:

- **Duyệt và xử lý dữ liệu**.
- **Cập nhật dữ liệu** nếu cursor cho phép (tùy thuộc vào loại cursor).
- **Thực hiện các hành động khác** như chèn hoặc xóa dữ liệu (với cursor có quyền thay đổi).

1.10.9.2 Đọc và xử lý dữ liệu trong cursor

```
FETCH [Next | Prior | First | Last | Absolute n | Relative n ]
FROM Tên_cursor
[INTO danh_sách_biến]
```

Trong đó:

- **Next, Prior, First, Last:** dùng để đọc dữ liệu kế tiếp, trước, đầu, sau cùng.
- **Absolute:** đọc dòng **dữ liệu chính xác thứ n** trong cursor. **n>0** chỉ định việc đọc dữ liệu tại dòng thứ n đếm từ dòng đầu tiên, **n<0** dùng chỉ định việc đọc dữ liệu tại dòng thứ n được đếm ngược từ dòng cuối trở lên.
- **Relative:** dùng chỉ định việc đọc dữ liệu tại **một dòng tương đối so với dòng dữ liệu hiện hành.** N là một số nguyên có thể dương có thể âm để chỉ định theo chiều tới (xuống) hoặc lui (lên) so với dòng dữ liệu hiện hành.
- **Tên_cursor:** tên của biến cursor đã định nghĩa trước đó bằng lệnh DECLARE.
- **Danh sách biến:** danh sách tên các biến cục bộ đã được định nghĩa trước đó. Các biến này sẽ **lưu trữ các giá trị dữ liệu được đọc từ lệnh FETCH.**
  - **Số lượng biến trong danh sách phải bằng số lượng cột mà cursor lấy từ lệnh FOR. Nếu không sẽ báo lỗi:**

```
Msg 16924, Level 16, State 1, Line <line_number>
Cursor FETCH statement failed because the number of variables in the INTO clause does
not match the number of columns in the SELECT statement of the cursor.
```

Ví dụ minh họa

Cursor với 3 cột nhưng chỉ 2 biến

Sai cú pháp:

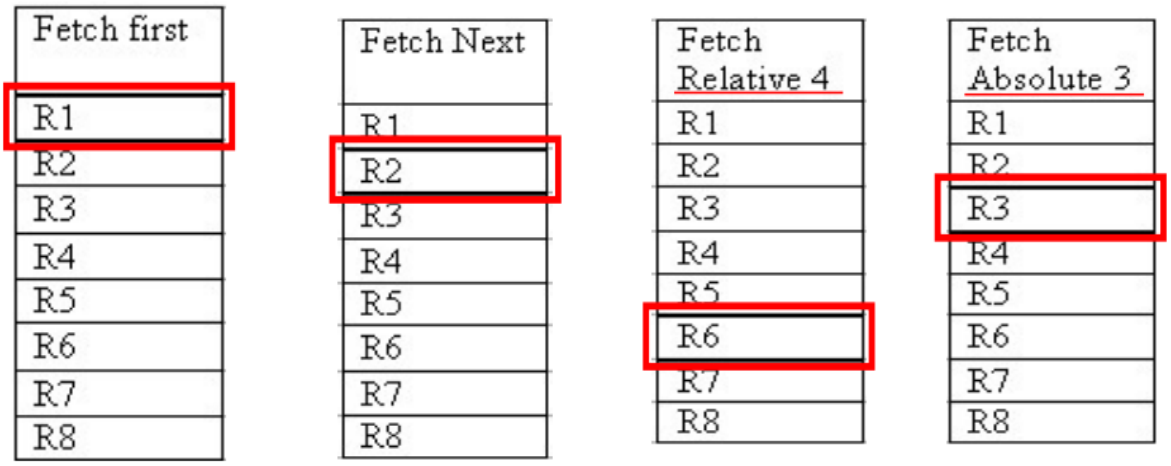
```
DECLARE @Col1 INT, @Col2 NVARCHAR(50)
DECLARE cur_Example CURSOR FOR
SELECT Col1, Col2, Col3 FROM TableName

-- Sẽ báo lỗi
FETCH NEXT FROM cur_Example INTO @Col1, @Col2
```

Đúng cú pháp:

```
DECLARE @Col1 INT, @Col2 NVARCHAR(50), @Col3 DATE
DECLARE cur_Example CURSOR FOR
SELECT Col1, Col2, Col3 FROM TableName

-- Hoạt động đúng
FETCH NEXT FROM cur_Example INTO @Col1, @Col2, @Col3
```



1.10.9.3 Đóng cursor

Cú pháp:

```
CLOSE Tên_cursor
DEALLOCATE Tên_cursor
```

Trong đó

- **CLOSE giải phóng <u>các dòng dữ liệu</u>** tham chiếu bên trong cursor.
- Lệnh **DEALLOCATE <u>giải phóng thật sự biến</u> cursor** ra khỏi bộ nhớ

- SQL Server cung cấp một biến hệ thống **@@FETCH_STATUS** dùng để kiểm tra trình trạng đọc dữ liệu thành công hay thất bại.
- **Giá trị trả về 0 khi việc đọc dữ liệu là <u>thành công</u>.**

Cho lược đồ quan hệ như sau:

```
MAT_HANG(MaMH, TenMH, DVT, MaNCC)
PNHAP(MaPN, NgayNhap, ThanhTien)
CTPNHAP(MaMH, MaPN, SLNhap, DonGia)
```

Ví dụ 1: Đọc dữ liệu cursor của bảng MAT_HANG **chỉ đọc các vật tư là Tivi**

```
-- Khai báo biến cursor
DECLARE cur_MatHang CURSOR
KEYSET
FOR
        SELECT * FROM MAT_HANG
        WHERE MaMH like 'TV%'
        ORDER BY MaMH

-- Mở cursor
OPEN cur_MatHang
```

```
-- Đọc dữ liệu
FETCH NEXT FROM cur_MatHang
WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Đọc tiếp dòng kế
            FETCH NEXT FROM cur_MatHang
        END

-- Đóng cursor
CLOSE cur_MatHang
DEALLOCATE cur_MatHang
```

- **ORDER BY MaMH**: Các dòng dữ liệu trong kết quả của câu lệnh SELECT sẽ được sắp xếp theo cột MaMH theo **thứ tự tăng dần.**
- **OPEN cur_MatHang**: Lệnh này mở cursor và thực thi câu lệnh SELECT đã được chỉ định, truy xuất dữ liệu từ bảng MAT_HANG vào cursor cur_MatHang. Khi cursor được mở, **con trỏ bắt đầu truy cập dữ liệu từ câu lệnh SELECT.**
- **FETCH NEXT FROM cur_MatHang**: Lệnh FETCH NEXT lấy dòng tiếp theo từ cursor cur_MatHang và gán dữ liệu vào các biến (nếu có khai báo). Mỗi lần FETCH NEXT được gọi, con trỏ sẽ di chuyển tới dòng kế tiếp.

- *Một ví dụ cụ thể với bảng dữ liệu giả định, sau đó thực hiện các lệnh SQL với cursor và trả về kết quả.*

1. Bảng dữ liệu giả định:

Giả sử bạn có bảng MAT_HANG với dữ liệu như sau:

```
CREATE TABLE MAT_HANG (
   MaMH VARCHAR(10),
   TenMH VARCHAR(100),
   Gia INT
);
```

Dữ liệu mẫu trong bảng MAT_HANG có thể là:

MaMH	TenMH	Gia
TV001	Tivi Sony	500
TV002	Tivi Samsung	400
TVA100	Tivi LG	600
TV12	Tivi Panasonic	700
TV99	Tivi Sharp	450

2. Thực hiện các lệnh sử dụng cursor:

Đây là đoạn mã SQL để khai báo, mở, duyệt và xử lý dữ liệu từ cursor, rồi trả về kết quả.

```
-- Khai báo biến cursor
DECLARE cur_MatHang CURSOR
KEYSET
FOR
        SELECT * FROM MAT_HANG
        WHERE MaMH LIKE 'TV%'
        ORDER BY MaMH;

-- Mở cursor
OPEN cur_MatHang;
```

```
-- Đọc dữ liệu và in ra kết quả
DECLARE @MaMH VARCHAR(10), @TenMH VARCHAR(100), @Gia INT;

FETCH NEXT FROM cur_MatHang INTO @MaMH, @TenMH, @Gia;

-- Vòng lặp để duyệt qua từng dòng dữ liệu
WHILE @@FETCH_STATUS = 0
BEGIN
   PRINT 'Mã MH: ' + @MaMH + ', Tên MH: ' + @TenMH + ', Giá: ' + CAST(@Gia AS VARCHAR(10));

   -- Đọc tiếp dòng kế
   FETCH NEXT FROM cur_MatHang INTO @MaMH, @TenMH, @Gia;
END;

-- Đóng cursor
CLOSE cur_MatHang;

-- Giải phóng cursor
DEALLOCATE cur_MatHang;
```

Giải thích các lệnh:

- **Khai báo cursor**:
  - **DECLARE cur_MatHang CURSOR**: Tạo cursor cur_MatHang.
  - **KEYSET**: Đây là kiểu cursor, có thể đọc dữ liệu và **dữ liệu trong cursor sẽ không thay đổi trong quá trình duyệt.**
  - **SELECT * FROM MAT_HANG WHERE MaMH LIKE 'TV%' ORDER BY MaMH**: Lọc tất cả các mặt hàng có mã bắt đầu với 'TV' và sắp xếp theo mã sản phẩm.
- **Mở cursor**:
  - **OPEN cur_MatHang**: Mở cursor và thực thi câu lệnh SELECT.
- **Duyệt qua dữ liệu**:
  - **FETCH NEXT FROM cur_MatHang INTO @MaMH, @TenMH, @Gia**: Lấy dòng dữ liệu tiếp theo vào các biến @MaMH, @TenMH, và @Gia.
  - **WHILE @@FETCH_STATUS = 0**: Lặp qua các dòng dữ liệu trong cursor. @@FETCH_STATUS trả về 0 khi có dòng dữ liệu tiếp theo để lấy, và -1 khi không còn dòng nào.
- **In ra kết quả**:
  - Trong vòng lặp, các giá trị của các cột MaMH, TenMH và Gia được in ra bằng câu lệnh PRINT.
- **Đóng và giải phóng cursor**:
  - **CLOSE cur_MatHang**: Đóng cursor sau khi hoàn thành duyệt.
  - **DEALLOCATE cur_MatHang**: Giải phóng tài nguyên của cursor.

Kết quả trả về (Giả sử dữ liệu trong bảng là như trên):

```
Mã MH: TV001, Tên MH: Tivi Sony, Giá: 500
Mã MH: TV002, Tên MH: Tivi Samsung, Giá: 400
Mã MH: TVA100, Tên MH: Tivi LG, Giá: 600
Mã MH: TV12, Tên MH: Tivi Panasonic, Giá: 700
Mã MH: TV99, Tên MH: Tivi Sharp, Giá: 450
```

Nếu bạn muốn thực hiện các thao tác khác với dữ liệu (như cập nhật hoặc xóa), bạn có thể thêm các lệnh vào bên trong vòng lặp WHILE.

Lưu ý:

- Mặc dù cursor hữu ích trong một số tình huống, việc sử dụng cursor có thể gây ảnh hưởng đến hiệu suất nếu dữ liệu quá lớn, vì **nó duyệt qua dữ liệu tuần tự** thay vì xử lý đồng thời.
- **Nếu không cần thao tác dữ liệu theo từng dòng, có thể thay thế cursor bằng các thao tác set-based như UPDATE, DELETE, hoặc INSERT** để xử lý dữ liệu hiệu quả hơn.

Ví dụ 2: *cập nhật dữ liệu cho **cột ThanhTien** trong bảng **PNHAP** bằng cách duyệt qua từng phiếu nhập,* **tính ra trị giá nhập của từng phiếu** căn cứ vào **số lượng nhập và đơn giá nhập của từng vật tư** trong bảng **CTPNHAP**, sau cùng cập nhật vào cột **ThanhTien**.

```
-- Khai báo biến cursor, các biến cục bộ
DECLARE @Sopn char(4), @TongTT Money
DECLARE cur_Pnhap CURSOR
FORWARD_ONLY
FOR
        SELECT SOPN -- cursor chỉ lấy cột SOPN từ bảng PNHAP
        FROM PNHAP

-- Mở cursor
OPEN cur_Pnhap

-- Đọc dữ liệu và cập nhật giá trị
FETCH NEXT FROM cur_Pnhap INTO @SoPN
WHILE @@FETCH_STATUS = 0
        BEGIN
                SELECT @TongTT = SUM(SLNhap*dongia) //cập nhật giá trị biến
                FROM CTPNHAP
                WHERE MaPN = @SoPN
                Print 'dang cap nhat phieu nhap: ' + @SoPN

                UPDATE PNHAP SET Thanhtien = @TongTT
                WHERE sopn = @SOPN -- Current OF cur_Pnhap

                -- dịch con trỏ đến dòng kế tiếp
                FETCH NEXT FROM cur_Pnhap INTO @Sopn
        END

-- Đóng cursor
CLOSE cur_Pnhap
DEALLOCATE cur_Pnhap
```

- Kiểu dữ liệu **MONEY** trong SQL Server được sử dụng để **lưu trữ các giá trị tiền tệ hoặc giá trị số có độ chính xác cố định**, thường được sử dụng trong các ứng dụng tài chính.
- **Tính toán với kiểu MONEY có thể gây ra sai số nhỏ:** Vì MONEY giới hạn ở 4 chữ số thập phân, các phép tính phức tạp có thể dẫn đến sai số tích lũy.
- Giá trị kiểu MONEY tự động được định dạng với **dấu phân cách hàng ngàn và dấu thập phân**, ví dụ: 12,345.6789.
- **Chuyển đổi sang kiểu chuỗi hoặc số khác:** Sử dụng CAST hoặc CONVERT để định dạng hoặc chuyển kiểu dữ liệu.

```
SELECT CAST(Price AS VARCHAR) AS PriceText
FROM Sales
```

Ý nghĩa của từng phần trong mã

- **DECLARE cur_Pnhap CURSOR**:
  - Khai báo cursor cur_Pnhap để duyệt qua tất cả các phiếu nhập trong bảng PNHAP.
- **FETCH NEXT FROM cur_Pnhap INTO @SoPN**:
  - Lấy từng dòng dữ liệu trong cursor và gán giá trị SoPN cho biến @SoPN.
- **SELECT @TongTT = SUM(SLNhap * DonGia)**:
  - Tính tổng trị giá nhập (tổng số lượng × đơn giá) từ bảng CTPNHAP cho phiếu nhập hiện tại (MaPN = @SoPN).
- **UPDATE PNHAP SET ThanhTien = @TongTT**:
  - Cập nhật giá trị ThanhTien trong bảng PNHAP bằng tổng trị giá đã tính.
- **WHILE @@FETCH_STATUS = 0**:
  - Vòng lặp tiếp tục duyệt qua các dòng dữ liệu cho đến khi không còn dòng nào trong cursor.
- **CLOSE cur_Pnhap**:
  - Đóng cursor sau khi hoàn thành việc duyệt dữ liệu.
- **DEALLOCATE cur_Pnhap**:
  - Giải phóng tài nguyên liên quan đến cursor.

Bảng dữ liệu ban đầu

Bảng PNHAP (Phiếu nhập):

SoPN	ThanhTien
PN01	NULL
PN02	NULL
PN03	NULL

Bảng CTPNHAP (Chi tiết phiếu nhập):

MaPN	SLNhap	DonGia
PN01	10	5000
PN01	5	8000
PN02	20	3000
PN03	15	10000
PN03	10	7000

Kết quả của lệnh PRINT

Khi chạy lệnh trong SQL Server Management Studio (SSMS), các thông báo PRINT được hiển thị trong tab **Messages** như sau:

Đang cập nhật phiếu nhập: PN01
Đang cập nhật phiếu nhập: PN02
Đang cập nhật phiếu nhập: PN03

Bảng PNHAP sau khi cập nhật:

SoPN	ThanhTien
PN01	90000
PN02	60000
PN03	220000

Giải thích từng bước cập nhật

1. **Đang cập nhật phiếu nhập: PN01**
   - Lấy dữ liệu từ CTPNHAP:
     - (10 × 5000) + (5 × 8000) = 50000 + 40000 = 90000.

- ThanhTien của PN01 được cập nhật thành **90000**.

2. **Đang cập nhật phiếu nhập: PN02**
   - Lấy dữ liệu từ CTPNHAP:
     - (20 × 3000) = 60000.
   - ThanhTien của PN02 được cập nhật thành **60000**.
3. **Đang cập nhật phiếu nhập: PN03**
   - Lấy dữ liệu từ CTPNHAP:
     - (15 × 10000) + (10 × 7000) = 150000 + 70000 = 220000.
   - ThanhTien của PN03 được cập nhật thành **220000**.

Khi nào nên dùng WHERE CURRENT OF?

- WHERE CURRENT OF được sử dụng để cập nhật hoặc xóa dòng hiện tại mà cursor đang trỏ đến.
- Dùng khi bạn muốn cập nhật dòng hiện tại mà cursor đang trỏ đến mà không cần chỉ định khóa chính hoặc điều kiện WHERE chi tiết.
- Điều kiện tiên quyết:
  - Cursor phải được **khai báo với FOR UPDATE OF. (FOR UPDATE OF ThanhTien)**
  - Dòng hiện tại phải là dòng hợp lệ mà cursor đang duyệt.

Cách viết khác:

```
DECLARE @Sopn char(4), @TongTT Money
DECLARE cur_Pnhap CURSOR
FORWARD_ONLY
FOR
        SELECT SOPN
        FROM PNHAP
-- Mở cursor
OPEN cur_Pnhap

-- Đọc dữ liệu và cập nhật giá trị
WHILE 0 = 0
        BEGIN
                FETCH NEXT FROM cur_Pnhap INTO @Sopn
                IF @@FETCH_STATUS<>0
                        BREAK
                SELECT @TongTT = SUM(SLNhap*dongia)
                FROM CTPNHAP
                WHERE MaPN = @SoPN
                Print 'dang cap nhat phieu nhap: ' + @SoPN
                UPDATE PNHAP
                SET Thanhtien = @TongTT
                Where sopn = @SOPN
        END
-- Đóng cursor
CLOSE cur_Pnhap
DEALLOCATE cur_Pnhap
```

```
--1. Khai báo biến cursor
DECLARE Tên_cursor CURSOR {kiểu đọc | cập nhật dữ liệu}
FOR
        Câu lệnh SELECT

--2. Mở cursor
OPEN Tên_cursor

--3. Đọc dữ liệu và cập nhật giá trị
WHILE 0=0
        Begin
                FETCH NEXT FROM <Tên_cursor> [INTO danh_sách_biến]
                IF @@FETCH_STATUS <> 0
                        Break
                --cập nhật dữ liệu trong cursor
        End

--4. Đóng cursor
CLOSE Tên_cursor
DEALLOCATE Tên_cursor
```

- **Khi nào chúng ta cần sử dụng kiểu dữ liệu cursor** trong Transaction-SQL để giải quyết các vấn đề:
  - SQL Server là một hệ quản trị CSDL quan hệ (Relational Database Management System) do đó chúng ta nên chọn giải pháp làm việc trên các bộ mẩu tin.
  - Khi cần giải quyết vấn đề cập nhật dữ liệu thì luôn ưu tiên chọn các hướng giải quyết trên **bộ mẩu tin** bởi vì khi đó làm cho các bộ xử lý được nhanh hơn.
  - Sau cùng là **hướng giải quyết theo kiểu cursor là giải pháp sau cùng nhất** để chọn lựa **khi không còn giải pháp nào tốt hơn**

```
-- Cập nhật toàn bộ bảng PNHAP chỉ với một câu lệnh
UPDATE PNHAP
SET ThanhTien = (
   SELECT SUM(SLNhap * DonGia)
   FROM CTPNHAP
   WHERE CTPNHAP.SoPN = PNHAP.SoPN
```

1.10.10 Các hàm thường dùng

1.10.10.1 Các hàm chuyển đổi kiểu dữ liệu

a/ Hàm CAST: chuyển đổi **một biểu thức** nào đó **sang một kiểu dữ liệu** mong muốn.

Cú pháp:

```
CAST(Biểu_thức AS kiểu_dữ_liệu)
```

Ví dụ:

```
SELECT MaVTU, TenVT, TyLe = CAST(PHANTRAM AS VARCHAR(3)) + '%'
FROM VATTU
```

- Trong SQL Server, bạn **có thể cộng chuỗi** bằng cách sử dụng toán tử +.
- Nếu **PHANTRAM** là **NULL**, thì phép nối + sẽ trả về **NULL**. Để xử lý, bạn có thể dùng hàm **ISNULL**:

```
SELECT MaVTU, TenVT,
TyLe = ISNULL(CAST(PHANTRAM AS VARCHAR(3)), '0') + '%'
FROM VATTU
```

Cú pháp

ISNULL(expression, replacement_value)

- **expression**: Biểu thức hoặc cột cần kiểm tra giá trị NULL.
- **replacement_value**: Giá trị thay thế nếu expression là NULL.

Cách hoạt động

- Nếu **expression** không phải NULL, hàm **ISNULL** trả về chính giá trị của **expression**.
- Nếu **expression** là NULL, hàm **ISNULL** trả về **replacement_value**.

Bảng dữ liệu mẫu:

Bảng VATTU:

MaVTU	TenVT	PHANTRAM
VT001	Vật tư A	50
VT002	Vật tư B	NULL
VT003	Vật tư C	75

Kết quả:

MaVTU	TenVT	TyLe
VT001	Vật tư A	50%
VT002	Vật tư B	0%
VT003	Vật tư C	75%

Hàm thay thế tương tự:

- Trong MySQL: **IFNULL(expression, replacement_value)**
- Trong PostgreSQL: **COALESCE(expression, replacement_value)**

TyLe = CAST(PHANTRAM AS VARCHAR(3)) + '%'

cột PHANTRAM không bị thay thế bởi cột TyLe. Thay vào đó:

- **TyLe là một cột được tạo tạm thời (computed column)** trong kết quả của truy vấn, TyLe chỉ xuất hiện trong kết quả truy vấn. Nó không ghi đè hoặc thay thế giá trị của cột PHANTRAM trong bảng VATTU.
- Dữ liệu thực tế của bảng VATTU vẫn giữ nguyên, không thay đổi.

Nếu bạn muốn cập nhật cột PHANTRAM thành cột TyLe (ghi đè):

Nếu thực sự muốn thay đổi dữ liệu trong bảng VATTU (ghi đè PHANTRAM bằng giá trị có dạng 50%), bạn phải:

- Đảm bảo cột PHANTRAM được đổi kiểu thành VARCHAR.
- Thực hiện lệnh UPDATE. Ví dụ:

```
UPDATE VATTU
SET PHANTRAM = CAST(PHANTRAM AS VARCHAR(3)) + '%'
```

b/ Hàm CONVERT: chuyển đổi một biểu thức nào đó sang một kiểu dữ liệu bất kỳ mong muốn nhưng **có thể theo một định dạng nào đó.**

Cú pháp:

CONVERT (*Kiểu_dữ_liệu [(length)]*, Biểu_thức[,định_dạng])

- **(length): Optional**. The length of the resulting data type (for **char, varchar, nchar, nvarchar, binary and varbinary**)
- **định_dạng *(tùy chọn)*:** Mã định dạng dùng để chuyển đổi định dạng ngày/giờ hoặc kiểu khác. Mã này *chủ yếu áp dụng khi chuyển đổi DATETIME sang chuỗi (VARCHAR hoặc CHAR) hoặc ngược lại.*
- **Bảng dưới đây mô tả các mã định dạng:**

STT	Mã định dạng (style code)	Hiển thị dữ liệu
1	101	Mm/dd/yy
2	102	yy.mm.dd
3	103	Dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	Dd mon yy
7	107	Mon dd, yy
8	108	Hh:mm:ss
9	109	Mon dd yyyy hh:mm:ss
10	110	mm-dd-yy
11	111	Yy/mm/dd
12	112	Yymmdd
13	113	Dd mon yyyy hh:mm:ss
14	114	Hh:mm:ss:mmm
15	21 hoặc 121	Yyyy-mm-dd hh:mi:ss.mmm
16	20 hoặc 120	Yyyy-mm-dd hh:mi:ss

Ví dụ:

```
SELECT SoHD,
CONVERT(char(10),NgayHD, 103) AS NGAYHDFROM DONDH
```

c/ Hàm STR: chuyển đổi kiểu dữ liệu **số** sang kiểu dữ liệu **chuỗi**. Phải đảm bảo *đủ khoảng trắng để chứa các ký số khi chuyển sang kiểu dữ liệu chuỗi.*

Cú pháp:

STR(*number*, [*length*], [*decimals*])

Parameter	Description
number	**Required**. The number to convert to a string
length	Optional. The **length of the returning string**. Default value is **10**
Decimals *(phần thập phân)*	**Optional**. The **number of decimals** to display in the returning string. Default value is 0

Ví dụ:

```
SELECT STR(185.476, 6, 2);
```

Result: 185.48

```
SELECT TenVT, SLNhap = STR(SLNhap, 5) + " " + DVTinh
FROM VATTU, CTPNHAP
WHERE VATTU.MaVT = CTPNHAP.MaVT
```

- **STR(SLNhap, 5)**: Hàm STR() chuyển đổi giá trị số thành chuỗi, với chiều dài tối đa là 5 ký tự (bao gồm cả dấu cách nếu có).
- **" " + DVTinh**: Kết hợp chuỗi số lượng nhập (SLNhap) và đơn vị tính (DVTinh) với một dấu cách giữa chúng.
- **FROM VATTU, CTPNHAP WHERE VATTU.MaVT = CTPNHAP.MaVT**: Thực hiện nối (join) giữa hai bảng VATTU và CTPNHAP thông qua trường MaVT để lấy thông tin tên vật tư (TenVT), số lượng nhập (SLNhap) và đơn vị tính (DVTinh).

Ví dụ về Bảng Dữ Liệu:

Bảng VATTU:

MaVT	TenVT	DVTinh
VT01	Máy tính	Cái
VT02	Bút bi	Cây
VT03	Sách	Quyển

Bảng CTPNHAP:

MaVT	SLNhap
VT01	10
VT02	50
VT03	30

Kết Quả:

TenVT	SLNhap
Máy tính	10 Cái
Bút bi	50 Cây
Sách	30 Quyển

Chú ý: Hàm STR **tạo ra một khoảng trắng ở phía trước số** nếu **chiều dài của số nhỏ hơn 5 ký tự.**

1.10.10.2 Các hàm về ngày

a/ DATEDIFF: trả về **1 số nguyên** khoảng cách của hai ngày *theo một đơn vị thời gian* bất kỳ

```
DATEDIFF(don_vi, ngay1, ngay2)
```

don_vi (*interval)* **Required**. The part to return. Can be one of the following values:

- year, yyyy, yy = Year
- quarter, qq, q = Quarter
- month, mm, m = month
- dayofyear = Day of the year
- day, dy, y, DD = Day
- week, ww, wk = Week
- weekday, dw, w = Weekday
- hour, hh = hour
- minute, mi, n = Minute
- second, ss, s = Second
- millisecond, ms = Millisecond

Ví dụ:

```
SELECT MADH, SONGAY = DATEDIFF(DD, NGAYDH, NGAYGH)
FROM DONDH
```

Giả sử bảng DONDH có dữ liệu như sau:

MADH	NGAYDH	NGAYGH
DH001	2024-12-01	2024-12-05
DH002	2024-12-10	2024-12-12
DH003	2024-12-15	2024-12-18

Kết quả của câu lệnh sẽ là:

MADH	SONGAY
DH001	4
DH002	2
DH003	3

b/ DATENAME: trả về một **chuỗi** thời gian **đại diện của 1 ngày chỉ đinh theo một đơn vi thời gian bất kỳ**

```
DATENAME(Don_vi, ngay)
```

Trích xuất **một phần của ngày** (như ngày, tháng, năm, giờ, phút, v.v.) trả về **dạng chuỗi** (số nguyên), chẳng hạn như tên ngày trong tuần hoặc tên tháng.

Ví dụ:

```
SELECT DATENAME(MONTH, '2024-03-15')  -- Trả về: 'March'
SELECT DATENAME(DAY, '2024-03-15')    -- Trả về: '15'
SELECT DATENAME(YEAR, '2024-03-15')   -- Trả về: '2024'
SELECT DATENAME(DW, '2024-03-15')     -- Trả về: 'Saturday'
```

Ví dụ:

```
SELECT MADH, THU = DATENAME(DW, NGAYDH)FROM DONDH
```

DATENAME(DW, NGAYDH) sẽ lấy tên ngày trong tuần từ cột NGAYDH, với các giá trị có thể là "Sunday", "Monday", "Tuesday", v.v., tùy thuộc vào ngày trong cột NGAYDH.
THU: Là tên cột mới sẽ chứa kết quả là tên ngày trong tuần (ví dụ: 'Monday', 'Tuesday', v.v.).

Mặc định, SQL Server sẽ **bắt đầu từ Chủ nhật (Sunday = 1)** và **kết thúc với thứ Bảy (Saturday = 7),** nhưng bạn có thể thay đổi thứ tự ngày trong tuần bằng cách thay đổi cấu hình của hệ thống, nếu cần.

Giả sử bảng DONDH có dữ liệu sau:

MADH	NGAYDH
DH001	2024-12-01
DH002	2024-12-02
DH003	2024-12-03
DH004	2024-12-04

Kết quả của câu lệnh sẽ là:

MADH	THU
DH001	Sunday
DH002	Monday
DH003	Tuesday
DH004	Wednesday

Trích xuất một phần của ngày (như ngày, tháng, năm, giờ, phút, v.v.) dưới dạng giá trị số nguyên.

c/ GETDATE: trả về ngày giờ hiện hành của hệ thống **GETDATE()**

d/ DATEPART: trả về **1 số nguyên** chỉ định **thời gian đai diện** của **1 ngày** theo một đơn vị thời gian bất kỳ

```
DATEPART(Don_vi, ngay)
```

Trích xuất phần của ngày và trả về giá trị **dưới dạng số nguyên.**

Ví dụ:

```
SELECT DATEPART(MONTH, '2024-03-15')  -- Trả về: 3
SELECT DATEPART(DAY, '2024-03-15')    -- Trả về: 15
SELECT DATEPART(YEAR, '2024-03-15')   -- Trả về: 2024
```

Ví dụ:

```
SELECT SODH, THANG = DATEPART(MM, NGAYDH)
FROM DONDH
```

Giả sử bảng DONDH có dữ liệu sau:

SODH	NGAYDH
DH001	2024-01-10
DH002	2024-02-15
DH003	2024-03-20
DH004	2024-04-25

Kết quả của câu lệnh sẽ là:

SODH	THANG
DH001	1
DH002	2
DH003	3
DH004	4

1.11 STORE PROCEDURE – Thủ tục lưu trữ

1.11.1 Khái niệm

- A stored procedure is a named **group of SQL statements** that **have been previously created and stored in the server database.**
- là một **tập hợp chứa các dòng lệnh, các biến và các cấu trúc điều khiển** trong ngôn ngữ Transaction-SQL dùng để **thực hiện một hành động** nào đó.
- **Các nét đặc trưng**
  - **Tên thủ tục**
  - **Tham số** truyền giá trị vào
    - Stored procedures accept **input parameters** so that a single procedure *can be used over the network by several clients using different input data.*
  - Tham số đón nhận giá trị ra
  - Trong thủ tục nội tại được phép **gọi thực thi một thủ tục nội tại khác (nested stored procedures).**
- And when the **procedure is modified, all clients automatically get the new version.**
- **Lợi ích của thủ tục**
  - Tốc độ xử lý của các thủ tục nội tại rất nhanh. *(Với thủ tục hệ thống, tốc độ nhanh do được tối ưu hóa sâu bởi Microsoft. Với thủ tục do người dùng định nghĩa, tốc độ nhanh do được biên dịch và lưu trữ kế hoạch thực thi.)*
  - Việc tổ chức và phân chia các xử lý thành hai nơi khác nhau: tại máy chủ hoặc tại máy trạm sẽ giúp giảm thời gian xây dựng ứng dụng. *(Việc di chuyển logic xử lý lên máy chủ giúp giảm tải cho máy trạm và tăng hiệu suất.)*
  - Stored procedures **reduce network traffic and improve performance.**
  - Stored pocedures can be used to help **ensure the integrity** of the database.
- **e.g.** sp _ helpdb. sp _ renamedb, sp _depends etc.
- ***Difference between procedure and function:*** **A function must return only one value back to the caller**. While procedure can never return a value back to the caller.

Có 2 loại thủ tục lưu trữ

Thủ tục hệ thống (System stored procedures)

- Thủ tục mà những người sử dụng **chỉ có quyền thực hiện, không được phép thay đổi.** Các tác vụ quản trị bao gồm: **liệt kê, thêm, cập nhập, xóa.**
- Hầu hết tất cả các thủ tục hệ thống được **lưu trữ bên trong CSDL Master.**
- Kí hiệu: **sp_..., xp_...**
- Được chia thành các nhóm sau:
  - SP dùng để **liệt kê thông tin** (liệt kê danh sách database ,...)
  - SP dùng để **trình bày thông tin** (trình bày thông tin table, ...)
  - SP dùng để **thêm, xóa, cập nhật thông tin** (đổi mật khẩu, ...)

Thủ tục nội tại

- Thủ tục do người dùng tạo và thực hiện.
- Có thể chỉnh sửa, xóa hoặc thêm mới.
- Có **tính cục bộ** bên trong một cơ sở dữ liệu lưu trữ thủ tục đó
  - Thủ tục hệ thống được lưu trong cơ sở dữ liệu **master** và có sẵn cho tất cả các cơ sở dữ liệu, do đó **không hoàn toàn "cục bộ".**

1.11.2 Tạo mới thủ tục

Cú pháp:

```
CREATE PROC[EDURE] Tên_thủ_tục AS [Declare biến_cục_bộ]
các_lệnh
```

AS: Xác định rằng phần sau là nội dung (body) của stored procedure.
[Declare biến_cục_bộ]: Tùy chọn khai báo các biến cục bộ trong stored procedure.

Ví dụ: cho lược đồ CSDL như sau:

```
MAT_HANG(MaMH, TenMH, DVT, MaNCC)
PHIEU_XUAT(SoPX, NgayXuat, #SoDH)
CTPX(Ma_MH, SoPX, SLXuat, DGXuat)
HOA_DONDH(MaDH, NgayDat)
CTDH(MaDH, MaMH, SLDH, DonGia)
```

Cho biết mặt hàng nào có **doanh số bán cao nhất** trong tháng **01/2017.**

```
-- Khai báo thủ tục
CREATE PROC sp_MaxSLHang
AS
        -- Khai báo biến cục bộ
        Declare     @TenMH varchar(50), @MaxSL int

        -- Lấy max doanh số 1/2017
        Select      @TenMH=RTRIM(TenMH),      @MaxSL=SLXuat
        From        CTPX, PHIEU_XUAT, MAT_HANG
        Where       CTPX.SoPX= PHIEU_XUAT.SoPX
        And         MAT_HANG.MaMH=CTPX.MaMH
        And         convert(char(7), NgayXuat, 21)= '2017-01'
        And         SLXuat = (Select Max(SLXuat)
                              From      CTPX, PHIEU_XUAT
                              Where     CTPX.SoPX=PHIEU_XUAT.SoPX
                              And       convert(char(7), NgayXuat, 21)= '2017-01')
-- In kết quả
Print @TenMH + 'Co doanh so cao nhat la' + Cast(@MaxSL as char(10))
```

- **Select @TenMH = RTRIM(TenMH), @MaxSL = SLXuat**:
  - Gán giá trị cột TenMH (loại bỏ khoảng trắng dư ở bên phải bằng RTRIM) vào biến @TenMH.
  - Gán giá trị cột SLXuat vào biến @MaxSL.
- **CTPX.SoPX = PHIEU_XUAT.SoPX**: Kết nối chi tiết phiếu xuất với phiếu xuất thông qua SoPX.
- **MAT_HANG.MaMH = CTPX.MaMH**: Kết nối chi tiết phiếu xuất với mặt hàng thông qua MaMH.
- **convert(char(7), NgayXuat, 21) = '2017-01'**: Lọc phiếu xuất có ngày xuất nằm trong tháng 01 năm 2017.
  - **convert(char(7), NgayXuat, 21)**: Chuyển cột NgayXuat thành chuỗi định dạng yyyy-MM.
- **SLXuat = (Select Max(SLXuat)**:
  - Lấy dòng có số lượng xuất (SLXuat) bằng số lượng xuất lớn nhất (dùng hàm Max) trong tháng 01/2017.
- **Cast(@MaxSL as char(10))**: Chuyển giá trị số lượng xuất lớn nhất từ kiểu int sang kiểu chuỗi char để nối chuỗi với phần còn lại.

Có 2 điều kiện **And convert(char(7), NgayXuat, 21)= '2017-01'**, nhưng mỗi điều kiện có công dụng khác nhau:

- **Điều kiện ngoài** đảm bảo rằng bạn chỉ lấy các mặt hàng trong tháng 01/2017 để gán giá trị cho biến @TenMH và @MaxSL.
- **Điều kiện trong (subquery)** đảm bảo rằng bạn tìm giá trị **số lượng xuất lớn nhất** (dùng trong so sánh SLXuat = ...) cũng chỉ trong tháng 01/2017.

Ví dụ dữ liệu của các bảng liên quan:

Bảng MAT_HANG

MaMH	TenMH
MH01	Điện thoại
MH02	Laptop
MH03	Tai nghe

Bảng PHIEU_XUAT

SoPX	NgayXuat
PX01	2017-01-15
PX02	2017-01-20
PX03	2017-02-05

Bảng CTPX (Chi tiết phiếu xuất)

SoPX	MaMH	SLXuat
PX01	MH01	100
PX01	MH02	50
PX02	MH01	70
PX02	MH03	30
PX03	MH02	90

Thực thi thủ tục sp_MaxSLHang:

- Thủ tục sẽ thực hiện các bước sau:
  1. Lọc các phiếu xuất có ngày thuộc tháng 01/2017 (PX01 và PX02).
  2. Tìm mặt hàng có số lượng xuất (SLXuat) cao nhất trong tháng này.
  3. Kết quả là mặt hàng "Điện thoại" với số lượng 100.

Kết quả đầu ra:

Điện thoại Co doanh so cao nhat la 100

1.11.3 Gọi thực hiện thủ tục

Cú pháp:

```
EXEC[UTE] Tên_thủ_tục
```

Ví dụ:

```
EXEC sp_MaxSLHang
```

1.11.4 Thay đổi nội dung thủ tục

Cú pháp:

```
ALTER PROC[EDURE] Tên_thủ_tục
AS
[Declare biến_cục_bộ]
Các_lệnh.
```

Ví dụ:

```
ALTER PROC sp_MaxSLHang
AS
        Declare @TenMH varchar(50), @MaxSL int
        IF NOT EXISTS(Select Ma_mh
                From CTPX, PHIEU_XUAT
                Where CTPX.SoPX= PHIEU_XUAT.SoPX
                And convert(char(7), NgayXuat, 21)= '2007-01')
Begin
        Print 'thang 01 nam 2007 chưa bán mặt hàng nào cả'
        Return
End
        Select @TenMH=RTRIM(TenMH), @MaxSL=SLXuat
        From CTPX, PHIEU_XUAT, MAT_HANG
        Where CTPX.SoPX= PHIEU_XUAT.SoPX
        And MAT_HANG.MaMH=CTPX.MaMH1
        And convert(char(7), NgayXuat, 21)= '2007-01'
        And SLXuat = (Select Max(SLXuat)
                From CTPX, PHIEU_XUAT
                Where CTPX.SoPX=PHIEU_XUAT.SoPX
                And convert(char(7), NgayXuat, 21)= '2007-01')
        Print @TenMH + 'Co doanh so cao nhat la' + Cast(@MaxSL as char(10))
```

Phân tích thay đổi trong thủ tục sp_MaxSLHang

So với phiên bản trước, thủ tục này có một thay đổi quan trọng: sử dụng kiểm tra điều kiện IF NOT EXISTS trước khi thực hiện việc tìm kiếm mặt hàng. Dưới đây là phân tích từng thay đổi:

---

1. Kiểm tra điều kiện trước khi thực hiện logic chính

```
IF NOT EXISTS(Select Ma_mh
    From CTPX, PHIEU_XUAT
    Where CTPX.SoPX = PHIEU_XUAT.SoPX
    And convert(char(7), NgayXuat, 21) = '2007-01')
```

- **Ý nghĩa:** Kiểm tra xem có bất kỳ mặt hàng nào được bán trong tháng 01/2007 không.
- **Lý do:** Tránh thực hiện các truy vấn nặng (như tìm mặt hàng có doanh số cao nhất) nếu không có dữ liệu trong khoảng thời gian yêu cầu.
- **Cách hoạt động:**
  - Câu lệnh SELECT Ma_mh sẽ trả về dòng dữ liệu nếu có mặt hàng được xuất.
  - IF NOT EXISTS sẽ thực thi khối lệnh bên trong (BEGIN ... END) nếu **không có dòng nào được trả về**.

2. Xử lý khi không có dữ liệu

```
Begin
        Print 'thang 01 nam 2007 chưa bán mặt hàng nào cả'
        Return
End
```

- **Ý nghĩa:** Thông báo rằng không có dữ liệu trong tháng 01/2007 và dừng thực thi thủ tục.
- **Lý do:** Tránh việc thực hiện các truy vấn không cần thiết khi không có dữ liệu phù hợp.
- **Hành vi:**
  - Print xuất thông báo "tháng 01 năm 2007 chưa bán mặt hàng nào cả".
  - Return kết thúc thủ tục sớm, không thực thi các phần tiếp theo.

1.11.5 Thủ tục với tham số đầu vào

Cú pháp:

```
CREATE PROC[EDURE] Tên_thủ_tục
        @Tên_tham_số   kiểu_dữ_liệu   [= giá_trị]
AS
        [Declare biến_cục_bộ]
        các_lệnh
```

Ví dụ 1: Tạo thủ tục tính tổng giá trị của một phiếu xuất hàng hoá có một **tham số vào là số phiếu xuất với kiểu dữ liệu là chuỗi.**

```
CREATE PROC sp_TongTGXuat @SoPX char(4)
AS
        Declare @TongTG money
        Select @TongTG=SUM(SLXuat*DGXuat)
        From CTPX
        Where @SoPX=SoPX
        Print 'Tri gia phieu xuat' + CAST(@SoPX AS char(4))
        Print 'là: ' + CAST(@TongTG as Varchar(15))
```

- Gọi thực hiện thủ tục

```
        Exec sp_TongTGXuat 'PX01'
```

- Hoặc:

```
        Exec sp_TongTGXuat @SoPX='PX01'
```

Ví dụ 2: tạo thủ tục hiển thị số đặt hàng của một mặt hàng trong một đơn đặt hàng có 2 tham số vào là số đặt hàng và mã mặt hàng.

```
CREATE PROC sp_TinhSLDat @SoDH char(4), @MaMH char(4)
AS
        Declare @Sldat int
        IF NOT EXISTS(Select MoDH From CTDH
                Where MaDH=@SoDH And MaMH=@MaMH)
        Begin
                Print 'khong hop le, xem lai don dat hang'
                Return
        End
        Select @SLDat = SLDat
        From CTDH
        Where SoDH = @SoDH And MaMH = @MaMH
```

```
        Print 'Don dat hang ' + @SoDH
        Print 'Voi ma mat hang ' + @MaMH
        Print 'Co so luong dat la: ' + Cast(@SLDat as varchar(10))
```

- Gọi thực hiện thủ tục:
  Exec sp_TinhSLDat **'DH01', 'Fe'**
- Hoặc
  Exec sp_TinhSLDat **@MaMH = 'Fe', @SoDH = 'DH01'**

Ví dụ: tạo thủ tục **thêm mới dữ liệu vào bảng MAT_HANG** với tên **sp_MATHANG_Them** gồm có **4 tham số vào** chính là các giá trị thêm mới cho các cột trong bảng MAT_HANG: **mã mặt hàng, tên mặt hàng, đơn vị tính…** Trong đó cần kiểm tra **các ràng buộc dữ liệu phải hợp lệ trước khi thực hiện lệnh INSERT INTO** để thêm dữ liệu vào bảng MAT_HANG.

- Mã mặt hàng phải **duy nhất**
- **Tỷ lệ phần trăm** phải nằm trong miền giá trị **0 đến 100**

```
CREATE PROC SP_MATHANG_Them
        @MaMH char(4), @TenMH varchar(50), @DVT varchar(50),@PhanTram INT
AS

        --Định nghĩa chuỗi lỗi
        DECLARE @ErrMsg varchar(200)

        --Kiểm tra có mặt hàng chưa?
        IF EXISTS(SELECT MaMH FROM MAT_HANG WHERE MaMH=@MaMH)
        BEGIN
                SET @ErrMsg = 'Mã mặt hàng [' + @MaMH + '] đã có'
                RAISERROR(@ErrMsg, 16, 1)
                RETURN
        END

        --Kiểm tra tỷ lệ phần trăm nằm ngoài 0…100
        IF @PhamTram NOT BETWEEN 0 AND 100
        BEGIN
                SET @ErrMsg = 'Tỷ lệ phần trăm nằm trong đoạn [0, 100]'
                RAISERROR(@ErrMsg, 16, 1)
                Return
        END

--Khi các RBTV hợp lệ thì thêm dữ liệu vào bảng MatHang
        INSERT INTO MAT_HANG(MaMH, TenMH, DVT, PhanTram)
        VALUES(@MaMH, @TenMH, @DvTinh, @PhanTram)
```

RAISERROR trong SQL Server

Lệnh RAISERROR được sử dụng để tạo và **trả về một thông báo lỗi** trong SQL Server. Nó có thể được dùng để thông báo lỗi tùy chỉnh hoặc tạo ngoại lệ trong các stored procedure, trigger, hoặc batch script.

1. Cú pháp cơ bản

```
RAISERROR ( { msg_id | msg_str | @local_variable }
        , severity
        , state
        [, argument [ ,...n ] ] )
```

- **msg_id**: ID của thông báo lỗi được định nghĩa trước trong sys.messages.
- **msg_str**: Chuỗi lỗi tùy chỉnh. Có thể sử dụng định dạng chuỗi với các tham số thay thế.
- **@local_variable**: Biến chứa chuỗi lỗi hoặc mã lỗi.
- **severity**: Mức độ nghiêm trọng của lỗi.
  - Giá trị từ 1 đến 25.
  - **16** thường được sử dụng cho lỗi người dùng.
- **state**: Một số nguyên từ 0 đến 255. Dùng để cung cấp thông tin bổ sung về trạng thái lỗi.
- **argument**: Các tham số thay thế cho chuỗi lỗi (tương tự printf trong C).

2. Ý nghĩa dòng lệnh

RAISERROR(@ErrMsg, 16, 1)

- **@ErrMsg**:
  - Là một biến chứa chuỗi thông báo lỗi được xác định trước đó.
  - Ví dụ:

```
DECLARE @ErrMsg NVARCHAR(255) = N'Lỗi: Số lượng không hợp lệ!'
RAISERROR(@ErrMsg, 16, 1)
```

- **16**:
  - Mức độ nghiêm trọng.
  - **16** đại diện cho lỗi do người dùng hoặc ứng dụng gây ra.
- **1**:
  - Trạng thái của lỗi (state).
  - Dùng để phân loại lỗi chi tiết hơn, mặc dù thường **ít được sử dụng.**

1.11.6 Thủ tục với tham số đầu ra

Cú pháp:

```
CREATE PROC Tên_thủ_tục
        @Tên_tham_số   kiểu_dữ_liệu OUTPUT[,...]
AS
        [Declare Biến cục bộ]
        Các_lệnh
```

- Khi gọi thủ tục, phải sử dụng từ khóa OUTPUT đi kèm tham số để nhận giá trị trả về.

Ví dụ: tạo thủ tục tính **số đặt hàng** của một mặt hàng trong một đơn đặt hàng có **2 tham số vào là số đặt hàng và mã mặt hàng**, **trả ra số lượng đặt hàng** của một vật tư tương ứng trong đơn đặt hàng thông qua tham số đầu ra.

```
CREATE PROC sp_TinhSLDat
        @Sodh char(4), @MaMH char(4), @SLDat int OUTPUT

AS
        IF NOT EXISTS(Select MaDH From CTDH
                Where MaDH=@SoDH And MaMH=@MaMH)
        Begin
                Print 'khong hop le, xem lai don dat hang'
```

```
        Return
    End

    Select @SLDat = SLDH
    From CTDH
    Where MaDH = @SoDH And MaMH = @MaMH
```

Gọi thực hiện thủ tục

```
DECLARE @SLDatHang int
EXEC sp_TinhSLDat @MaMH = 'Fe', @SoDH = 'DH01', @SLDat = @SLDatHang OUTPUT
Print 'Don dat hang DH01 với mặt hàng Fe'
Print 'co so luong dat la: ' + CAST(@SLDatHang AS varchar(10))
```

*** Có bao nhiêu tham số thì cần bấy nhiêu từ khóa "OUTPUT".**

1.11.7 Thủ tục có lệnh trả về Return

- Return **không có giá trị chỉ định** thì thủ tục sẽ trả về giá trị là **không (0).**
- **Return [Số_nguyên]**

Ví dụ: Tạo thủ tục tính tổng số lượng đặt hàng của một mặt hàng đối với một nhà cung cấp chỉ định, kiểm tra xem **giá trị của mặt hàng và mã nhà cung cấp** mà người dùng **truyền vào** thủ tục **có đúng hay không?** Qui định thủ tục **trả về 1 khi mã mặt hàng không tồn tại, trả về 2 khi mã nhà cung cấp không tồn tại.**

```
CREATE PROC sp_TinhTongSLDat
@MaNCC char(3), @MaMH char(4), @TongSLdat INT OUTPUT
AS
    IF NOT EXISTS(Select * From Mat_Hang Where MaMH=@MaMH)
        Return 1
    IF NOT EXISTS(Select * From Mat_Hang Where MaNCC=@MaNCC)
        Return 2

    Select @TongSLdat = SUM(SLDat)
    From HoaDon_DH, CTDH
    Where   HoaDon_DH.MaDH = CTDH.MaDH
    And     MaNCC=@MaNCC
    And     MaVTu=@MaVTu

    IF @TongSLdat IS NULL
        Set @TongSLdat=0
    Return
```

Gọi thực hiện thủ tục:

```
Declare @TongSLD INT, @Ketqua INT
EXEC @ketqua = sp_TinhTongSLDat 'NCCA', 'Fe', @TongSLdat=@TongSLD output

IF @ketqua =1
    Print 'Mã mặt hàng không hợp lệ'
ELSE IF @ketqua=2
    Print 'Mã nhà cung cấp không hợp lệ'
ELSE
    Print 'Tổng số lượng đặt là: ' + CAST(@TongSLD as char(10))
```

1.11.8 Sử dụng bảng tạm trong thủ tục

Cú pháp:

```
SELECT danh_sách_các_cột INTO #Tên_bảng_tạm
FROM Tên_bảng_dữ_liệu
```

(#): tạo ra các bảng tạm cục bộ

(##): tạo ra các bảng tạm toàn cục

Ví dụ: Tạo thủ tục tính **mặt hàng nào có doanh thu bán ra cao nhất** trong **một năm tháng bất kỳ.**

```
CREATE PROC sp_TinhDTCaoNhat
        @namThang char(7),
        @TenMH char(50) OUTPUT, @TongTien Money OUTPUT
AS
        Select MH.MaMH, TenMH, Sum(SLXuat*DGXuat) AS TT
        INTO #DoanhThu
        From Phieu_xuat PX, CTPX, Mat_Hang MH
        Where       PX.SoPX = CTPX.SoPX
        And         CTPX.MaMH = MH.MaMH
        And         Convert(char(7), ngayxuat, 21) = @namthang
        Group By    MH.MaMH, TenMH
        Order by    3 DESC

        Select Top 1 @TenMH=TenMH, @Tongtien = TT
        From #DoanhThu
```

Order by 3 DESC: Sắp xếp theo cột thứ 3 (TT) giảm dần, để mặt hàng có doanh thu cao nhất đứng đầu.

Gọi thực hiện thủ tục

```
Declare @TenMH char(50), @TongTien Money
EXEC sp_TinhDTCaoNhat '2017-01', @TenMH OUTPUT, @TongTien OUTPUT

IF @TenMH IS NULL
        Print 'không có dữ liệu tính toán'
ELSE
        Begin
                Print Rtrim(@TenMH) + 'có doanh thu cao nhất'
                Print 'là ' + CAST(@TongTien AS Varchar(20)) + 'VND'
        End
```

Lý Do Dùng Bảng Tạm

- Bảng tạm giúp lưu trữ các kết quả trung gian của truy vấn, trong trường hợp này là danh sách các mặt hàng và doanh thu của chúng trong tháng được chỉ định.
- Việc sử dụng bảng tạm giúp truy vấn trở nên đơn giản hơn và dễ quản lý, đặc biệt khi cần thực hiện các phép toán hoặc sắp xếp trước khi lấy kết quả cuối cùng.

Ví Dụ Dữ Liệu và Bảng Tạm

Giả sử bạn có dữ liệu trong các bảng Phieu_xuat, CTPX, Mat_Hang như sau:

Phieu_xuat:

SoPX	NgayXuat
1	2017-01-05
2	2017-01-06
3	2017-01-10
4	2017-01-15

CTPX:

SoPX	MaMH	SLXuat	DGXuat
1	MH01	10	5000
2	MH02	20	4000
3	MH01	30	5000
4	MH03	15	3000

Mat_Hang:

MaMH	TenMH
MH01	Mặt hàng 1
MH02	Mặt hàng 2
MH03	Mặt hàng 3

Kết quả truy vấn vào bảng tạm #DoanhThu:

MaMH	TenMH	TT (Tổng doanh thu)
MH01	Mặt hàng 1	170000
MH02	Mặt hàng 2	80000
MH03	Mặt hàng 3	45000

Sau khi bảng tạm #DoanhThu được tính toán và sắp xếp, thủ tục sẽ chọn mặt hàng có doanh thu cao nhất và trả về:

- **TenMH**: "Mặt hàng 1"
- **TongTien**: 170000 VND

MH01 có doanh thu cao nhất
là 170000 VND

1.11.9 Tham số cursor bên trong thủ tục

- Tham số cursor **trả về danh sách các dòng dữ liệu theo điều kiện chọn lọc nào đó.**
- Cursor được chia làm 2 phần: **bên trong thủ tục và bên ngoài thủ tục.**
- Các hành động **trong** thủ tục: **định nghĩa dữ liệu** cho biến kiểu cursor và **mở cursor.**
- Các hành động **bên ngoài** thủ tục: **đọc từng dòng dữ liệu bên trong cursor** và sau cùng là **đóng cursor** lại.

Ví dụ: tạo thủ tục trả về danh sách các **mã vật tư** đã **bán ra nhiều nhất** trong năm tháng nào đó.

Bước 1: tạo thủ tục có tham số kiểu dữ liệu cursor chứa danh sách các vật tư đã bán ra **nhiều nhất.**

```
CREATE PROC sp_TinhDsoBan
        @NamThang char(6),
        @cur_Dsmh CURSOR VARYING OUTPUT
AS

        --Tạo bảng tạm tính ra tổng số lượng bán
        SELECT CTDH.MAMH, SUM(SLDH) AS TongSLBan
        INTO #TongSLBan
        FROM CTDH, MAT_HANG, HOA_DONDH
        WHERE Convert(char(6), NgayDat, 112) = @NamThang
        AND             CTDH.MaMH=MAT_HANG.MaMH
        AND             CTDH.MaDH=HOA_DONDH.MaDH
        Group By        CTDH.MaMH

        -- Kiểm tra dữ liệu có phát sinh - Kiểm tra xem bảng có chứa dữ liệu hay không.
        IF EXISTS(SELECT MaMH FROM #TongSLBan)
        Begin
                -- Khởi tạo giá trị biến CURSOR
                SET @cur_Dsmh = CURSOR Forward_Only
                FOR
```

```
            SELECT MAMH, TongSLBan
            From #TongSLBan
            Where TongSLBan = (SELECT MAX(TongSLBan)
                               FROM #TongSLBan)
        --Mở cursor
        OPEN @cur_Dsmh
        DROP TABLE #TongSLBan -- Xóa bảng tạm để giải phóng tài nguyên.
        Return
    End

    -- Khi không có dữ liệu phát sinh
    DROP Table #TongSLBan
    Return 1
```

@cur_Dsmh CURSOR VARYING OUTPUT: Tham số đầu ra, một con trỏ sẽ chứa kết quả danh sách các mặt hàng và tổng số lượng bán nhiều nhất.

- **Con trỏ VARYING là gì?**
  - Trong SQL Server, từ khóa VARYING cho phép con trỏ (CURSOR) được gán lại với một con trỏ khác trong phạm vi của thủ tục hoặc bên ngoài nó.
  - Điều này làm cho con trỏ linh hoạt hơn so với con trỏ tĩnh.
- **Tại sao sử dụng VARYING?**
  - Khi thủ tục trả về một con trỏ động, bạn cần sử dụng **VARYING** để cho phép gán kết quả của truy vấn (dòng kết quả từ SELECT) vào tham số con trỏ.
  - Điều này giúp quản lý kết quả động hoặc có thể tái sử dụng con trỏ trong các thủ tục hoặc logic khác nhau.
  - Khi con trỏ được sử dụng như tham số đầu ra trong thủ tục.

Bước 2: đọc cursor, đón nhận danh sách các mã mặt hàng đã bán ra nhiều nhất trong **tháng 01 năm 2002**

```
DECLARE @cur_Dsmh CURSOR, @Gtmh INT, @MaMH char(4), @TongslBan INT
EXEC @Gtmh = sp_TinhDsoBan '200702', @cur_Dsmh OUTPUT

--Xử lý tiếp sau đó
IF @Gtmh = 0 -- Nếu thủ tục return
Begin
    Print 'danh sách các mặt hàng'
    While(0=0)
    Begin
        Fetch Next From @cur_Dsmh INTO @MaMH, @TongslBan
        IF @@Fetch_status<>0
            Break;
        Print 'Mã vật tư: ' + @MaMH
        Print 'Tổng số lượng: ' + CAST(@TongslBan AS varchar(10))
        Print Replicate('-', 50)
    End
End

ELSE-- Nếu thủ tục return 1
    Print 'không có bán hàng trong năm tháng chỉ định'
```

- **'200702':** Tháng 2 năm 2007.
- Biến @Gtmh được sử dụng để nhận **giá trị trả về** từ thủ tục sp_TinhDsoBan **(thông qua return trong thủ tục).** Giá trị này giúp kiểm tra xem thủ tục đã thực thi thành công hay chưa và có dữ liệu cần xử lý hay không.

REPLICATE(character_expression, integer_expression)

- **character_expression:** Một chuỗi ký tự hoặc biểu thức ký tự cần được lặp lại.
- **integer_expression:** Số lần lặp lại chuỗi ký tự.

Mục đích: Hàm REPLICATE tạo ra một chuỗi mới bằng cách lặp lại chuỗi ký tự được chỉ định một số lần.

REPLICATE('-', 50)

- Hàm này tạo ra một chuỗi gồm 50 ký tự dấu gạch ngang (-), thường được dùng để tạo các dòng phân cách trong kết quả in ra.

Dữ liệu mẫu:

1. Bảng MAT_HANG (Danh sách mặt hàng)

MaMH	TenMH
MH01	Mặt hàng 1
MH02	Mặt hàng 2
MH03	Mặt hàng 3
MH04	Mặt hàng 4

2. Bảng HOA_DONDH (Hóa đơn đặt hàng)

MaDH	NgayDat
DH01	2007-02-10
DH02	2007-02-15
DH03	2007-03-01
DH04	2007-02-20

3. Bảng CTDH (Chi tiết đơn hàng)

MaDH	MaMH	SLDH
DH01	MH01	20
DH01	MH02	15
DH02	MH01	25
DH02	MH03	10
DH03	MH02	30
DH04	MH01	10
DH04	MH04	5

Khi gọi thủ tục với @NamThang = '200702' (tháng 02 năm 2007), dữ liệu cần tính tổng số lượng bán (SLDH) cho từng mặt hàng có ngày đặt hàng thuộc tháng này.

- **Bảng #TongSLBan (Tính tổng số lượng bán):**

MaMH	TongSLBan
MH01	55
MH02	15
MH03	10
MH04	5

- Mặt hàng có tổng số lượng bán nhiều nhất là **MH01 (55)**.
- **Kết quả in ra:**

```
Danh sách các mặt hàng
Mã vật tư: MH01
Tổng số lượng: 55
----------------------------------------------
```

PHỤ LỤC

A. Dữ liệu

Dữ liệu: Thông tin về đối tượng được lưu trữ trên máy tính.

- **Cơ sở dữ liệu:** Tập hợp dữ liệu rời rạc được tổ chức có cấu trúc & có liên quan đến nhau, được lưu trữ trên các thiết bị lưu trữ thông tin, nhằm phục vụ cho nhiều mục đích.

B. Toàn vẹn dữ liệu

- **Ràng buộc toàn vẹn:** Các **điều kiện, quy tắc** trong bài toán/vấn đề cần xét, nếu không đủ RBTV thì CSDL tìm ẩn nhiều rủi ro.
- **RBTV quan tâm đến:**
  - **1/ Bối cảnh** (vd: ngày sinh của sv > 18t => Các quan hệ bị tác động bởi các ĐK);
  - **2/ Ngôn ngữ đại số quan hệ;**
  - **3/ Bảng tầm ảnh hưởng** (Khi thêm vào dữ liệu thì KT xem dữ liệu có vi phạm điều kiện hay không? => HQTCSDL tự động chạy để KT).
- **Toàn vẹn dữ liệu** được chia làm 2 loại
  - **Physical integrity**
    - Physical integrity means protecting the **accuracy, correctness, and wholeness of data** when it is stored and retrieved. This is typically compromised (bị xâm phạm) by issues like *power outages, storage erosion (xói mòn), hackers targeting database functions, and natural disasters,* which prevent accurate data storage and retrieval.
  - **Logical integrity**
    - Logical integrity ensures that **data remains unchanged** while being used in different ways through relational databases. This approach also aims to protect data from hacking or human error issues but does so differently than physical integrity.
    - Logical integrity comes in four different formats:
- **Các loại toàn vẹn dữ liệu:**
  - **Toàn vẹn miền giá trị (Domain Integrity):**
    - Domain integrity is a **series of processes that guarantee (bảo đảm) the accuracy of pieces of data within a domain.**
    - **A domain** is classified (phân loại) by a set of values that a table's columns are allowed to contain, along with constraints and measures that limit the amount, format, and type of data that can be entered.
    - Đảm bảo rằng giá trị của dữ liệu nằm trong **một phạm vi hoặc kiểu dữ liệu xác định.** Ví dụ:
      - Cột "Ngày sinh" không được chấp nhận giá trị ngày lớn hơn ngày hiện tại.
      - Cột "Số lượng" chỉ chứa giá trị số nguyên dương.
  - **Toàn vẹn thực thể (Entity Integrity):**
    - Entity integrity is a feature of relation systems that store data within tables, which can be used and linked in various ways. It **relies on primary keys and unique values** being created to identify a piece of data. This ensures data cannot be *listed multiple times, and fields in a table cannot be null.*
    - Mỗi bản ghi (record) trong một bảng phải có thể được nhận dạng duy nhất.
  - **Toàn vẹn tham chiếu (Referential Integrity):**
    - Referential integrity is a **series of processes** that ensure **data remains stored and used in a uniform manner (theo cách thống nhất).** Database structures are embedded with rules that define how foreign keys are used, which ensures

only appropriate data deletion, changes, and amendments (sửa đổi) can be made. This can prevent **data duplication** and guarantee data accuracy.

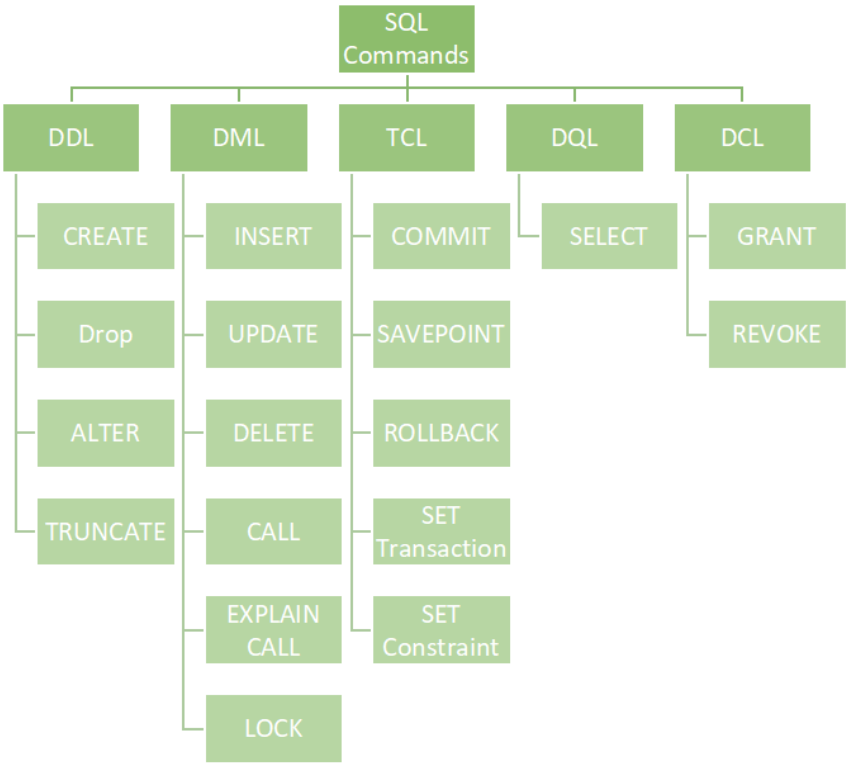
- Đảm bảo mối quan hệ giữa các bảng được duy trì. Ví dụ:
  - Một khóa ngoại (Foreign Key) phải tham chiếu đến một bản ghi tồn tại trong bảng liên quan.

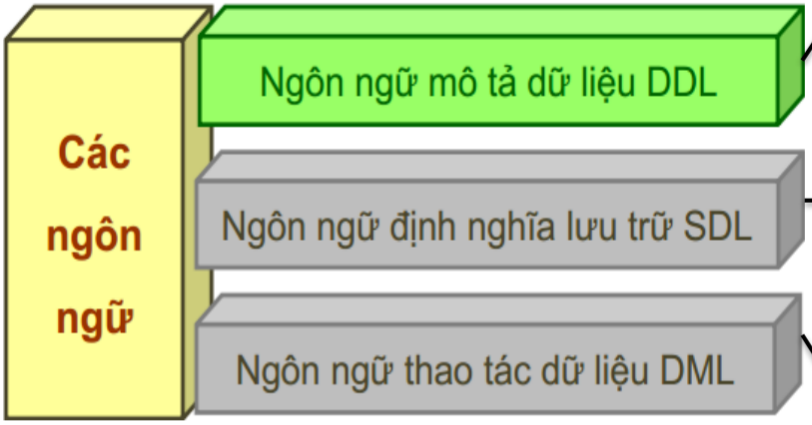
- **Toàn vẹn nghiệp vụ/ theo người dùng (Business Integrity/ User-defined integrity):**
  - User-defined integrity means that **rules and constraints** around data are **created by users** to align with their specific requirements. This is usually used when *other integrity processes will not safeguard an organization's data,* allowing for the creation of rules that incorporate (kết hợp) an organization's data integrity measures (cho phép tạo các quy tắc kết hợp các biện pháp toàn vẹn dữ liệu của tổ chức).
  - Ví dụ: Tổng số tiền chi tiêu không được vượt quá hạn mức tín dụng.

- Nhất quán dữ liệu là **trạng thái mà dữ liệu được lưu trữ trong hệ thống luôn đúng và tuân theo các quy tắc hoặc ràng buộc xác định**. Điều này có nghĩa là tại mọi thời điểm, dữ liệu phải không mâu thuẫn và phù hợp với nhau trên toàn hệ thống.
  - The **same data across all related systems, applications, and databases** is when we say that data is consistent. Inconsistent data can lead to **incorrect analysis, decision-making, and outcomes.**
  - **Ví dụ về nhất quán dữ liệu:**
    - **Nhất quán trong giao dịch:**
      - Trong một giao dịch ngân hàng chuyển tiền:
      - Số tiền bị trừ từ tài khoản A phải bằng số tiền được cộng vào tài khoản B.
      - Nếu hệ thống ghi số tiền trừ từ tài khoản A nhưng không cộng vào tài khoản B, dữ liệu sẽ không nhất quán.
    - **Nhất quán giữa các bảng:**
      - Nếu bảng "Đơn hàng" tham chiếu đến bảng "Khách hàng", thì tất cả đơn hàng phải thuộc về khách hàng tồn tại.
    - **Nhất quán thời gian thực:**
      - Trong một hệ thống phân tán, dữ liệu trên các máy chủ phải giống nhau tại mọi thời điểm (tùy thuộc vào mô hình nhất quán: chặt chẽ, cuối cùng, hoặc lỏng lẻo).

C. Truy vấn SQL

- **5 loại truy vấn SQL được sử dụng rộng rãi.**
  - Ngôn ngữ định nghĩa dữ liệu - Data Definition Language (DDL)
  - Ngôn ngữ thao tác dữ liệu – Data Manipulation Language (DML)
  - Ngôn ngữ kiểm soát dữ liệu - Data Control Language (DCL)
  - Ngôn ngữ kiểm soát giao dịch - Transaction Control Language (TCL)
  - Ngôn ngữ truy vấn dữ liệu - Data Query Language (DQL)





DDL (Data Definition Language)
– định nghĩa cấu trúc của CSDL
– định nghĩa các mối liên hệ của dữ liệu, các quy tắc

SDL (Storage Definition Language)
– định nghĩa cách tổ chức, lưu trữ vật lý của CSDL

DML (Data Manipulation Language)
– cập nhật dữ liệu: thêm, xóa, sửa

DML thủ tục (Low-level hay Procedural DML): Được nhúng trong một ngôn ngữ lập trình khác

DML phi thủ tục (High-level hay Nonprocedural DML): Ngôn ngữ truy vấn thủ tục riêng