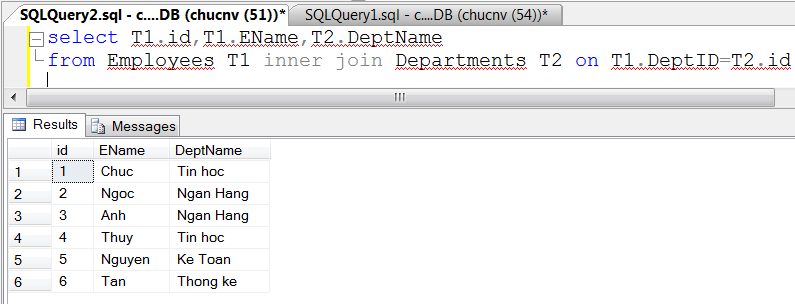
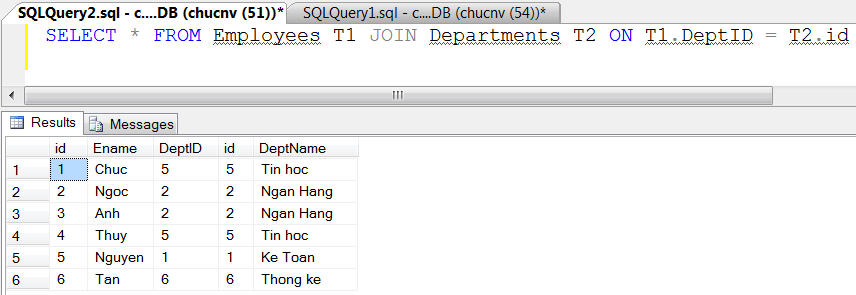
**CÁC CÂU HỎI PHỎNG VẤN**

**Các phép nối trong SQL**

1. Inner Join (Equi-join, Natural Join )
   1. Inner join



* 1. Equi-Join là phép nối mà trong đó điều kiện nối chỉ thực hiện phép so sánh bằng (=)



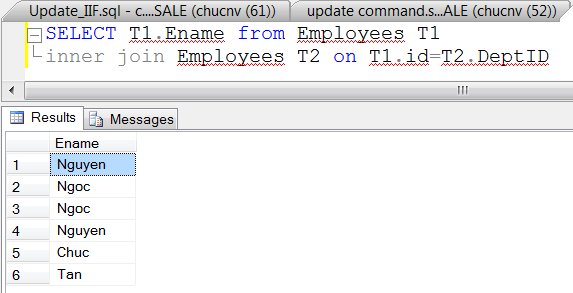
1. Outer Join (Left outer Join, Right outer join, Full outer join)
   1. Outer join trả về tất cả các dòng của cả 2 bảng cho dù các dòng dữ liệu có được so khớp (match) hay không

Outer join có 3 loại sau: Left outer join, Right outer join, Full outer join

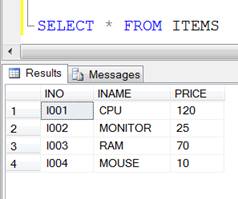
* + 1. Left Outer join: Hiển thị tất cả các dòng trong bảng đầu tiên (bên trái phép nối), các dòng trong bảng thứ 2 nếu không so khớp sẽ nhận giá trị null
    2. Right outer join: Ngược lại với Left Outer join
    3. Full outer join: Hiển thị tất cả các dòng trong cả 2 bảng cho dù dữ liệu có được so khớp (match) hay không

1. Cross Join (Cartesion Join)
   1. Mỗi hàng của bảng thứ nhất sẽ được kết hợp với mỗi hàng của bảng thứ 2. Do vậy số lượng các hàng trong bảng mới tạo ra bằng tích số lượng hàng giữa các bảng.
2. Self Join
   1. Self Join là phép nối 1 bảng với chính nó, Self Join thường được sử dụng khi muốn trích xuất những dòng dữ liệu có mối quan hệ hoặc tương tự với các dòng dữ liệu khác trong cùng 1 bảng

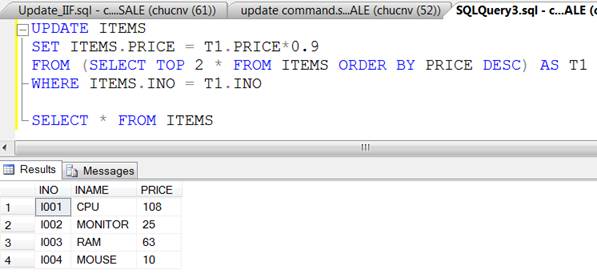
Truy vấn sau đây đưa ra các nhân viên có ID trùng với DeptID

1. 

Giả sử có bảng hàng hóa (ITEMS) với dữ liệu như sau

1. 

*Yêu cầu:* Giảm giá 2 mặt hàng có giá cao nhất xuống 10%



**Transaction trong SQL**

* Transaction (giao dịch) được dung để đảm bảo tính toàn vẹn dữ liệu khi xảy ra cập nhật (cập nhật xin được hiểu theo nghĩa rộng là các hành động sửa đổi dữ liệu, như INSERT, UPDATE, DELETE…). Khi một transaction bao gồm nhiều lệnh cập nhật, nó đảm bảo tất cả các cập nhật đều được thực hiện thành công, hoặc trong trường hợp một lệnh gặp sự cố toàn bộ transaction bị hủy bỏ. Khi đó dữ liệu trở về trạng thái như trước khi xảy ra transaction. Nói cách khác transaction ngăn chặn tình huống dữ liệu được cập nhật nửa chừng, trong đó một phần được cập nhật còn một phần bị bỏ qua.

Example:

ALTER Procedure [dbo].[DeleteStudentTransaction]

@Id INT

AS

BEGIN TRY

BEGIN TRANSACTION

DELETE FROM Student WHERE Id=@Id

-- some other codes

COMMIT

END TRY

BEGIN CATCH

ROLLBACK

END CATCH

4 đặc tính của Transaction:

* + Bảo toàn – đảm bảo rằng tất cả các câu lệnh trong nhóm lệnh được thực thi thành công. Nếu không transaction bị hủy bỏ tại thời điểm thất bại và tất các thao tác trước đó được khôi phục về trạng thái cũ.
  + Nhất quán – đảm bảo rằng cơ sở dữ liệu thay đổi chính xác các trạng thái khi một transaction được thực thi thành công.
  + Độc lập – cho phép các transaction hoạt động độc lập và minh bạch với nhau.
  + Bền bỉ - đảm bảo rằng kết quả của một transaction được commit vẫn tồn tại trong trường hợp lỗi hệ thông

Một số vấn đề xuất hiện khi có 2 transaction cùng hoạt đông:

* Dùng các mức Isolation Level:
  + Isolation level là một thuộc tính của transaction, qui định mức độ cô lập của dữ liệu mà transaction có thể truy nhập vào khi dữ liệu đó đang được cập bởi một transaction khác. Khi một transaction cập nhật dữ liệu đang diễn ra, một phần dữ liệu sẽ bị thay đổi (ví dụ một số bản ghi của bảng được sửa đổi hoặc bị xóa bỏ, một số được thêm mới), vậy các transaction hoặc truy vấn khác xảy ra đồng thời và cùng tác động vào các bản ghi đó sẽ diễn ra thế nào? Chúng sẽ phải đợi đến khi transaction đầu hoàn thành hay có thể thực hiện song song, kết quả dữ liệu nhận được là trong khi hay sau khi cập nhật? Bạn có thể điều khiển những hành vi này thông qua việc đặt isolation level của từng transaction. SQL Server cung cấp các mức isolation level sau xếp theo thứ tự tăng dần của mức độ cô lập của dữ liệu: **Read Uncommitted, Read Commited, Repeatable Read, và Serializable**.
* Read Uncommitted: Khi transaction thực hiện ở mức này, các truy vấn vẫn có thể truy nhập vào các bản ghi đang được cập nhật bởi một transaction khác và nhận được dữ liệu tại thời điểm đó mặc dù dữ liệu đó chưa được commit (uncommited data). Nếu vì lý do nào đó transaction ban đầu rollback lại những cập nhật, dữ liệu sẽ trở lại giá trị cũ. Khi đó transaction thứ hai nhận được dữ liệu sai.
* Read Commited: Đây là mức isolation mặc định, nếu bạn không đặt gì cả thì transaction sẽ hoạt động ở mức này. Transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải đợi đến khi việc cập nhật thực hiện xong. Vì thế nó tránh được dirty read như ở mức trên
* Repeatable Read: Đây là mức isolation mặc định, nếu bạn không đặt gì cả thì transaction sẽ hoạt động ở mức này. Transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải đợi đến khi việc cập nhật thực hiện xong. Vì thế nó tránh được dirty read như ở mức trên
* Serializable: Mức isolation này tăng thêm một cấp nữa và khóa toàn bộ dải các bản ghi có thể bị ảnh hưởng bởi một transaction khác, dù là UPDATE/DELETE bản ghi đã có hay INSERT bản ghi mới

Đánh INDEX trong SQL:

* Các loại index
  1. Clustered Index: thường được tạo ra khi bảng có primary key và chỉ có 1 clustered index
  2. Non-Clustered Index
  3. Clustered index columnStore
  4. Non-Clustered Index columnStore

Entity Framework

1. DB first
2. Code first
3. Model first

Giải thích

|  |  |
| --- | --- |
| **Code First** | **Database First** |
| Viết các class Model bằng code C# Generate Database từ class Model | Generate Model từ Database Các class Model sẽ ko được phép chỉnh sửa |

1. Ưu điểm và nhược điểm

|  |  |  |
| --- | --- | --- |
|  | **Code First** | **Database First** |
| Ưu điểm | Rất phổ biến (vì các lập trình viên thường không thích thiết kế DB, nhưng thích thiết kế class)  Kiểm soát hoàn toàn code model, thêm xóa sửa thuộc tính vô cùng dễ dàng  Thuận tiện khi làm việc theo nhóm, mỗi cá nhân có thể chạy 1 database độc lập trên máy tính của mình, ko cần server trung gian. Dữ liệu test do mình tự tạo ra nên không ảnh hưởng tới người khác  Có thể version control Database | Ít phổ biến hơn  Dùng được DB có sẵn   * Code tự động sinh ra dựa vào các bảng trong database |
| Nhược điểm | * Bạn phải viết mọi thứ liên quan tới database vào trong code, bao gồm cả store procedure   Các thay đổi cấu trúc trực tiếp trên DB sẽ mất  Khó kiểm soát những column sẽ tạo trên Db  Hơi khó khi kết hợp với Db có sẵn | * Không quản lý được version của database * Bất tiện khi làm việc theo nhóm, cả nhóm thường phải chung 1 server để database thống nhất cho cả team |

* Trong trường hợp cơ sở dữ liệu của bạn có thay đổi về một số trường trong một bảng hoặc thêm mới bảng thì sẽ xử lý như thế nào? Thì ở bài này đối với ứng dụng web sử dụng Entity Framework Code First bạn có thể sử dụng **Migration** để khắc phục vấn đề này.

Khác biệt giữa asp web form và asp MVC

1. Web forms có ViewState, MVC thì không

* Như tôi đã đề cập ở trên, ViewState là một cơ chế quản lý trạng thái cấp trang. Nếu bạn có nhiều server-side web control trên một trang, ViewState của bạn sẽ trở nên cực kỳ lớn. Phần mã HTML trả về cho client nên nhỏ nhất có thể, bởi vậy không nên có kích thước biến ViewState lớn.
* MVC không có một ViewState. Nó sử dụng model/ViewModels để chuyển dữ liệu qua lại giữa các Views.

1. Web forms có code-behind model, MVC chỉ có các model

* *Code-Behind là một cách để đính kèm code C# hoặc VB với một trang web và hoạt động dựa trên những hành động cụ thể khi chúng xuất hiện trên trang*. Khi bạn biên dịch một Web Form có chứa code-behind, nó trở thành một phần của assembly (hay DLL) để tạo ra chức năng của trang web đó.

1. Web forms có các Web Control, MVC không có

* Có rất nhiều lập trình viên đã phàn nàn rằng MVC không có bất kỳ server-side control nào cả. Điều này là bởi vì họ đã quen thuộc với cách Web Forms làm việc trên một trang. Các control luôn có sẵn cho họ sử dụng. Cuối cùng, họ bắt đầu xây dựng các custom server control của riêng mình.
* Trong khi Web Forms có một số lượng lớn các server control, thì MVC lại có một cách tiếp cận khác. Sau một thời gian phát triển, các lập trình viên bắt đầu hiểu rõ mô hình MVC và đòn bẩy bởi JavaScript, nơi họ có thể xây dựng các custom control riêng của họ hoặc thậm chí sử dụng một thư viện của bên thứ ba như Bootstrap cho giao diện người dùng.

1. Web forms có IsPostBack, MVC có GET và POST

* Các lập trình viên Web Form mới vào nghề đôi khi thường quên thêm một dòng lệnh "if( IsPostBack) return;" ở sự kiện Page\_Load trong các trang web. Điều này có vẻ lạ đối với một số lập trình viên web kỳ cựu đến từ [PHP](https://techmaster.vn/khoa-hoc/25486/lap-trinh-php-7-qua-du-an-thuc-te) hoặc ngôn ngữ web khác.
* Toàn bộ ý tưởng của Web Forms là khi bạn đang request một trang, nó được coi là một GET (truy cập vào trang lần đầu tiên) và trang được khởi tạo. Khi bạn submit một form, thì Page\_Load sẽ phát hiện ra rằng đó là một Postback và sự kiện Page\_Load sẽ không được thực thi.
* MVC có các chuẩn HTTP Protocol thông qua các lệnh GET, POST, DELETE, và redirect. Khi bạn bấm vào một nút submit, form (và dữ liệu) sẽ được gửi tới controller và controller sẽ xử lý nó.
* Một lần nữa, cách mà MVC xử lý là tự nhiên hơn so với Web Forms.

1. Web Forms không dễ để sử dụng unit-test, MVC được xây dựng hỗ trợ test
2. Web Forms có một thẻ form lớn, MVC có thể có nhiều thẻ form trên một trang

CURSOR trong SQL

* Cursor dùng để lấy dữ liệu ra một bảng và bạn có thể duyệt qua từng dòng trong bảng đó với lệnh while. Tôi thấy rằng dữ liệu được cập nhật lên các bảng khi chạy delete,Update,insert tại mỗi vòng lặp này. Vì vậy có một số trường hợp bạn sẽ dùng cursor.  
  Ví dụ : Tôi cần xóa các danh mục vật tư có cùng tên nhưng khác mã và chỉ để lại một tên vật tư duy nhất. Lúc đó bạn không thể dùng :  
  delete from Item where not Exists(select \* from Item IT where IT.No<>Item.No\_ and IT.Name=Item.Name )  
  Vì Dòng trùng lặp chỉ được xóa ở bảng tạm. Sau khi câu lệnh xóa hoàn tất rồi thì mới cập nhật lên bảng chính. Mà trong lúc đó nó lại duệt đến dòng trùng lặp đã bị xóa và xóa luôn dòng mà trước đó nó để lại. Tóm lại là nó sẽ xóa hết vật tư cùng tên.  
  cursor giải quyết được vấn đề này. Vấn đề nên hay không nên dùng Cursor không quan trọng. Quan trọng là phải hiểu được cách thức xử lý của SQL để tối ưu hóa ứng dụng.

**SUBQUERY trong SQL**

* Truy vấn con SUBQUERY là truy vấn con nằm trong truy vấn khác, hoặc gọi là truy vấn lồng.

**PIVOT trong SQL**

* Trong [SQL Server](https://quantrimang.com/sql-server) (Transact-SQL), mệnh đề PIVOT cho phép phân tích bảng chéo (cross tabulation) chuyển dữ liệu từ bảng này sang bảng khác, tức là lấy kết quả tổng hợp rồi chuyển từ dòng thành cột.

**Truyền dữ liệu từ controller sang View có mấy cách:**

1. ViewData (dạng key) “ViewData[Danh\_muc]”
2. ViewBag (dạng dynamic) “ViewData.Danh\_muc”
3. TempData (dạng Dictionary) dạng giống Session, có phiên sống ngắn muốn gia hạn sống them phải dùng từ khóa “keep”

* TempData thường được dùng để di chuyển, move từ Controller này sang controller khác hoặc từ Action này sang Action khác. Hay dùng làm câu thông báo, validate.

MVC trong sql

* Model giúp lưu trữ dữ liệu của ứng dụng và trạng thái của nó. Nó là một cơ sở dữ liệu hoặc cũng có thể chỉ là một tập tin XML.
* View được coi là một giao diện người dùng được sử dụng bởi khách truy cập trang web của bạn để nhìn thấy các dữ liệu. Các trang ASPX thường được sử dụng để hiển thị view trong các ứng dụng ASP.NET MVC.
* Controller chịu trách nhiệm xử lý các tương tác của người dùng với trang web. Nó được sử dụng để xác định loại view nào cần phải được hiển thị. Controller cũng được sử dụng cho mục đích giao tiếp với model.

Note: Xử lý get, set, update data là trong Model.