MỤC LỤC

Overview of SQL Server 2000

Để đọc và hiểu bài viết này bạn phải có kiến thức căn bản về SQL và Access Database

Giới Thiệu SQL Server 2000

SQL Server 2000 là một hệ thống quản lý cơ sở dữ liệu (Relational Database Management System (RDBMS)) sử dụng **Transact-SQL** để trao đổi dữ liệu giữa Client computer và SQL Server computer. Một RDBMS bao gồm databases, database engine và các ứng dụng dùng để quản lý dữ liệu và các bộ phận khác nhau trong RDBMS.

SQL Server 2000 được tối ưu để có thể chạy trên môi trường cơ sở dữ liệu rất lớn (Very Large Database Environment) lên đến Tera-Byte và có thể phục vụ cùng lúc cho hàng ngàn user. SQL Server 2000 có thể kết hợp "ăn ý" với các server khác như Microsoft Internet Information Server (IIS), E-Commerce Server, Proxy Server....

SQL Server có 7 editions:

- **Enterprise** : Chứa đầy đủ các đặc trưng của SQL Server và có thể chạy tốt trên hệ thống lên đến 32 CPUs và 64 GB RAM. Thêm vào đó nó có các dịch vụ giúp cho việc phân tích dữ liệu rất hiệu quả (Analysis Services)
- **Standard** : Rất thích hợp cho các công ty vừa và nhỏ vì giá thành rẻ hơn nhiều so với Enterprise Edition, nhưng lại bị giới hạn một số chức năng cao cấp (advanced features) khác, edition này có thể chạy tốt trên hệ thống lên đến 4 CPU và 2 GB RAM.
- **Personal**: được tối ưu hóa để chạy trên PC nên có thể cài đặt trên hầu hết các phiên bản windows kể cả Windows 98.
- **Developer** : Có đầy đủ các tính năng của Enterprise Edition nhưng được chế tạo đặc biệt như giới hạn số lượng người kết nối vào Server cùng một lúc.... Đây là edition mà các bạn muốn học SQL Server cần có. Chúng ta sẽ dùng edition này trong suốt khóa học. Edition này có thể cài trên Windows 2000 Professional hay Win NT Workstation.
- **Desktop Engine** (MSDE): Đây chỉ là một engine chạy trên desktop và không có user interface (giao diện). Thích hợp cho việc triển khai ứng dụng ở máy client. Kích thước database bị giới hạn khoảng 2 GB.

- **Win CE** : Dùng cho các ứng dụng chạy trên Windows CE
- **Trial**: Có các tính năng của Enterprise Edition, download free, nhưng giới hạn thời gian sử dụng.

Cài Đặt SQL Server 2000 (Installation)

Các bạn cần có **Developer Edition** và ít nhất là 64 MB RAM, 500 MB hard disk để có thể install SQL Server. Bạn có thể install trên Windows Server hay Windows XP Professional, Windows 2000 Professional hay NT Workstation nhưng không thể install trên Win 98 family.

Vì một trong những đặc điểm của các sản phẩm Microsoft là dễ install nên chúng tôi không trình bày chi tiết về cách install hay các bước install mà chỉ trình bày các điểm cần lưu ý khi install mà thôi. Nếu các bạn gặp trở ngại trong việc install thì có thể đưa lên forum để hỏi thêm. Khi install bạn cần lưu ý các điểm sau:

Ở màn hình thứ hai bạn chọn **Install Database Server**. Sau khi install xong SQL Server bạn có thể install thêm Analysis Service nếu bạn thích.

Ở màn hình **Installation Definition** bạn chọn **Server and Client Tools**.

Sau đó bạn nên chọn kiểu **Custom** và **chọn tất cả** các bộ phận của SQL Server. Ngoài ra nên **chọn các giá trị mặc định** (default)

Ở màn hình **Authentication Mode** nhớ chọn **Mixed Mode** . Lưu ý vì SQL Server có thể dùng chung chế độ bảo mật (security) với Win NT và cũng có thể dùng chế độ bảo mật riêng của nó. Trong Production Server người ta thường dùng Windows Authetication vì độ an toàn cao hơn và dễ dàng cho người quản lý mạng và cả cho người sử dụng. Nghĩa là một khi bạn được chấp nhận (authenticated) kết nối vào domain thì bạn có quyền truy cập dữ liệu (access data) trong SQL Server. Tuy nhiên ta nên chọn Mixed Mode để dễ dàng cho việc học tập.

Sau khi install bạn sẽ thấy một icon nằm ở góc phải bên dưới màn hình, đây chính là Service Manager. Bạn có thể Start, Stop các SQL Server services dễ dàng bằng cách double-click vào icon này.

Một chút kiến thức về các Version của SQL Server

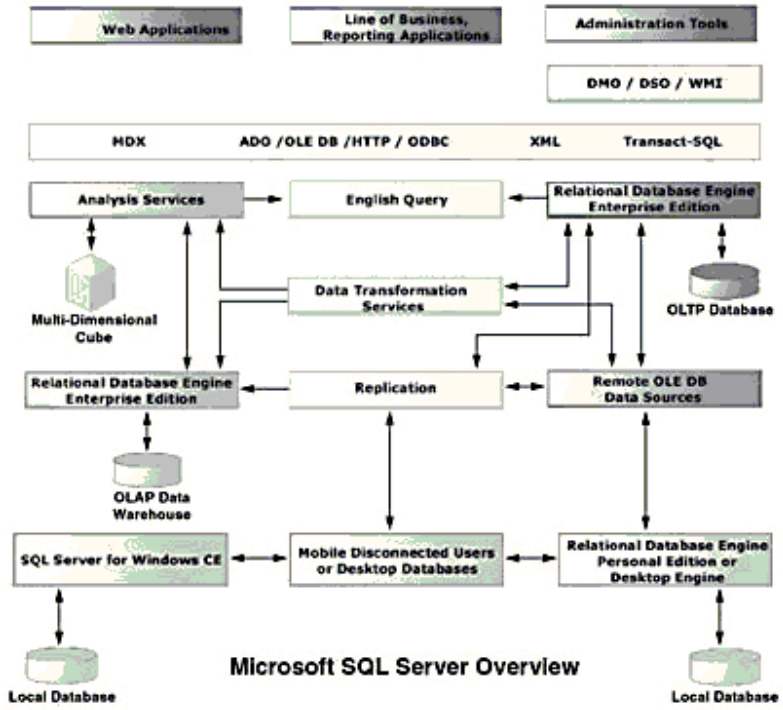
SQL Server của Microsoft được thị trường chấp nhận rộng rãi kể từ version 6.5. Sau đó Microsoft đã cải tiến và hầu như viết lại một engine mới cho SQL Server 7.0. Cho nên có thể nói từ version 6.5 lên version 7.0 là một bước nhảy vọt. Có một số đặc tính của SQL Server 7.0 không tương thích với version 6.5. Trong khi

đó từ Version 7.0 lên version 8.0 (SQL Server 2000) thì những cải tiến chủ yếu là mở rộng các tính năng về web và làm cho SQL Server 2000 đáng tin cậy hơn.

Một điểm đặc biệt đáng lưu ý ở version 2000 là **Multiple-Instance**. Nói cho dễ hiểu là bạn có thể install version 2000 chung với các version trước mà không cần phải uninstall chúng. Nghĩa là bạn có thể chạy song song version 6.5 hoặc 7.0 với version 2000 trên cùng một máy (điều này không thể xảy ra với các version trước đây). Khi đó version cũ trên máy bạn là **Default Instance** còn version 2000 mới vừa install sẽ là **Named Instance**.

Các thành phần quan trọng trong SQL Server 2000

SQL Server 2000 được cấu tạo bởi nhiều thành phần như Relational Database Engine, Analysis Service và English Query.... Các thành phần này khi phối hợp với nhau tạo thành một giải pháp hoàn chỉnh giúp cho việc lưu trữ và phân tích dữ liệu một cách dễ dàng.



Relational Database Engine - Cái lõi của SQL Server:

Đây là một engine có khả năng chứa data ở các quy mô khác nhau dưới dạng table và support tất cả các kiểu kết nối (data connection) thông dụng của Microsoft như ActiveX Data Objects (ADO), OLE DB, and Open Database Connectivity (ODBC). Ngoài ra nó còn có khả năng tự điều chỉnh (tune up) ví dụ

như sử dụng thêm các tài nguyên (resource) của máy khi cần và trả lại tài nguyên cho hệ điều hành khi một user log off.

Replication - Cơ chế tạo bản sao (Replica):

Giả sử bạn có một database dùng để chứa dữ liệu được các ứng dụng thường xuyên cập nhật. Một ngày đẹp trời bạn muốn có một cái database giống y hệt như thế trên một server khác để chạy báo cáo (report database) (cách làm này thường dùng để tránh ảnh hưởng đến performance của server chính). Vấn đề là report server của bạn cũng cần phải được cập nhật thường xuyên để đảm bảo tính chính xác của các báo cáo. Bạn không thể dùng cơ chế back up and restore trong trường hợp này. Thế thì bạn phải làm sao? Lúc đó cơ chế replication của SQL Server sẽ được sử dụng để bảo đảm cho dữ liệu ở 2 database được đồng bộ (synchronized). Replication sẽ được bàn kỹ trong bài 12

Data Transformation Service (DTS) - Một dịch vụ chuyển dịch data vô cùng hiệu quả

Nếu bạn làm việc trong một công ty lớn trong đó data được chứa trong nhiều nơi khác nhau và ở các dạng khác nhau cụ thể như chứa trong Oracle, DB2 (của IBM), SQL Server, Microsoft Access....Bạn chắc chắn sẽ có nhu cầu di chuyển data giữa các server này (migrate hay transfer) và không chỉ di chuyển bạn còn muốn định dạng (format) nó trước khi lưu vào database khác, khi đó bạn sẽ thấy DTS giúp bạn giải quyết công việc trên dễ dàng như thế nào. DTS sẽ được bàn kỹ trong bài 8.

Analysis Service - Một dịch vụ phân tích dữ liệu rất hay của Microsoft

Dữ liệu (Data) chứa trong database sẽ chẳng có ý nghĩa gì nhiều nếu như bạn không thể lấy được những thông tin (Information) bổ ích từ đó. Do đó Microsoft cung cấp cho bạn một công cụ rất mạnh giúp cho việc phân tích dữ liệu trở nên dễ dàng và hiệu quả bằng cách dùng khái niệm hình khối nhiều chiều (multi-dimension cubes) và kỹ thuật "đào mỏ dữ liệu" (data mining) sẽ được chúng tôi giới thiệu trong bài 13.

English Query - Một dịch vụ mà người Việt Nam chắc là ít muốn dùng :-) (?)

Đây là một dịch vụ giúp cho việc query data bằng tiếng Anh "trơn" (plain English).

Meta Data Service:

Dịch vụ này giúp cho việc chứa đựng và "xào nấu" Meta data dễ dàng hơn. Thế thì Meta Data là cái gì vậy? Meta data là những thông tin mô tả về cấu trúc của data trong database như data thuộc loại nào String hay Integer..., một cột nào

đó có phải là Primary key hay không....Bởi vì những thông tin này cũng được chứa trong database nên cũng là một dạng data nhưng để phân biệt với data "chính thống" người ta gọi nó là Meta Data. Phần này chắc là bạn phải xem thêm trong một thành phần khác của SQL Server sắp giới thiệu sau đây là **SQL Server Books Online** vì không có bài nào trong loạt bài này nói rõ về dịch vụ này cả.

SQL Server Books Online - Quyển Kinh Thánh không thể thiếu:

Cho dù bạn có đọc các sách khác nhau dạy về SQL server thì bạn cũng sẽ thấy books online này rất hữu dụng và không thể thiếu được(cho nên Microsoft mới hào phóng đính kèm theo SQL Server).

SQL Server Tools - Đây là một bộ đồ nghề của người quản trị cơ sở dữ liệu (DBA)

Ái chà nếu kể chi tiết ra thì hơi nhiều đấy cho nên bạn cần đọc thêm trong books online. Ở đây người viết chỉ kể ra một vài công cụ thông dụng mà thôi.

- Đầu tiên phải kể đến **Enterprise Manager**. Đây là một công cụ cho ta thấy toàn cảnh hệ thống cơ sở dữ liệu một cách rất trực quan. Nó rất hữu ích đặc biệt cho người mới học và không thông thạo lắm về SQL.
- Kế đến là **Query Analyzer**. Đối với một DBA giỏi thì hầu như chỉ cần công cụ này là có thể quản lý cả một hệ thống database mà không cần đến những thứ khác. Đây là một môi trường làm việc khá tốt vì ta có thể đánh bất kỳ câu lệnh SQL nào và chạy ngay lập tức đặc biệt là nó giúp cho ta debug mấy cái stored procedure dễ dàng.
- Công cụ thứ ba cần phải kể đến là **SQL Profiler**. Nó có khả năng "chụp" (capture) tất cả các sự kiện hay hoạt động diễn ra trên một SQL server và lưu lại dưới dạng text file rất hữu dụng trong việc kiểm soát hoạt động của SQL Server.
- Ngoài một số công cụ trực quan như trên chúng ta cũng thường hay dùng **osql** và **bcp** (bulk copy) trong command prompt.

Tóm lại trong bài này chúng ta đã dạo qua một vòng để tìm hiểu về SQL Server. Trong bài sau chúng ta cũng sẽ tiếp tục dạo chơi thêm một chút với Transact-SQL trước khi đi sâu vào các đề tài khác.

Transact SQL

Giới Thiệu Sơ Lược Về Transact SQL (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.

Trong bài này chúng ta sẽ tìm hiểu sơ qua về T-SQL. Chúng được chia làm 3 nhóm:

Data Definition Language (DDL):

Đây là những lệnh dùng để quản lý các thuộc tính của một database như định nghĩa các hàng hoặc cột của một table, hay vị trí data file của một database...thường có dạng

- Create *object_Name*
- Alter *object_Name*
- Drop *object_Name*

Trong đó *object_Name* có thể là một table, view, stored procedure, indexes...

Ví dụ:

Lệnh Create sau sẽ tạo ra một table tên Importers với 3 cột CompanyID,CompanyName,Contact

```
USE Northwind

CREATE TABLE Importers(
        CompanyID int NOT NULL,
        CompanyName varchar(40) NOT NULL,
        Contact varchar(40) NOT NULL
        )
```

Lệnh Alter sau đây cho phép ta thay đổi định nghĩa của một table như thêm(hay bớt) một cột hay một Constraint...Trong ví dụ này ta sẽ thêm cột ContactTitle vào table Importers

```
USE Northwind
```

```
ALTER TABLE Importers
ADD ContactTitle varchar(20) NULL
```

Lệnh Drop sau đây sẽ hoàn toàn xóa table khỏi database **nghĩa là cả định nghĩa của table và data bên trong table đều biến mất** (khác với lệnh Delete chỉ xóa data nhưng table vẫn tồn tại).

```
USE Northwind

DROP TABLE Importers
```

Data Control Language (DCL):

Đây là những lệnh quản lý các quyền truy cập lên từng object (table, view, stored procedure...). Thường có dạng sau:

- Grant
- Revoke
- Deny

Ví dụ:

Lệnh sau sẽ cho phép user trong Public Role được quyền Select đối với table Customer trong database Northwind (**Role** là một khái niệm giống như Windows Group sẽ được bàn kỹ trong phần Security)

```
USE Northwind
GRANT SELECT
ON Customers
TO PUBLIC
```

Lệnh sau sẽ từ chối quyền Select đối với table Customer trong database Northwind của các user trong Public Role

```
USE Northwind
DENY SELECT
ON Customers
TO PUBLIC
```

Lệnh sau sẽ xóa bỏ tác dụng của các quyền được cho phép hay từ chối trước đó

```
USE Northwind
REVOKE SELECT
ON Customers
TO PUBLIC
```

Data Manipulation Language (DML):

Đây là những lệnh phổ biến dùng để xử lý data như Select, Update, Insert, Delete

Ví dụ:

Select

```
USE Northwind
SELECT CustomerID, CompanyName, ContactName
FROM Customers
WHERE (CustomerID = 'alfki' OR CustomerID = 'anatr')
ORDER BY ContactName
```

Insert

```
USE Northwind
INSERT INTO Territories
VALUES (98101, 'Seattle', 2)
```

Update

```
USE Northwind
UPDATE Territories
SET TerritoryDescription = 'Downtown Seattle'
WHERE TerritoryID = 98101
```

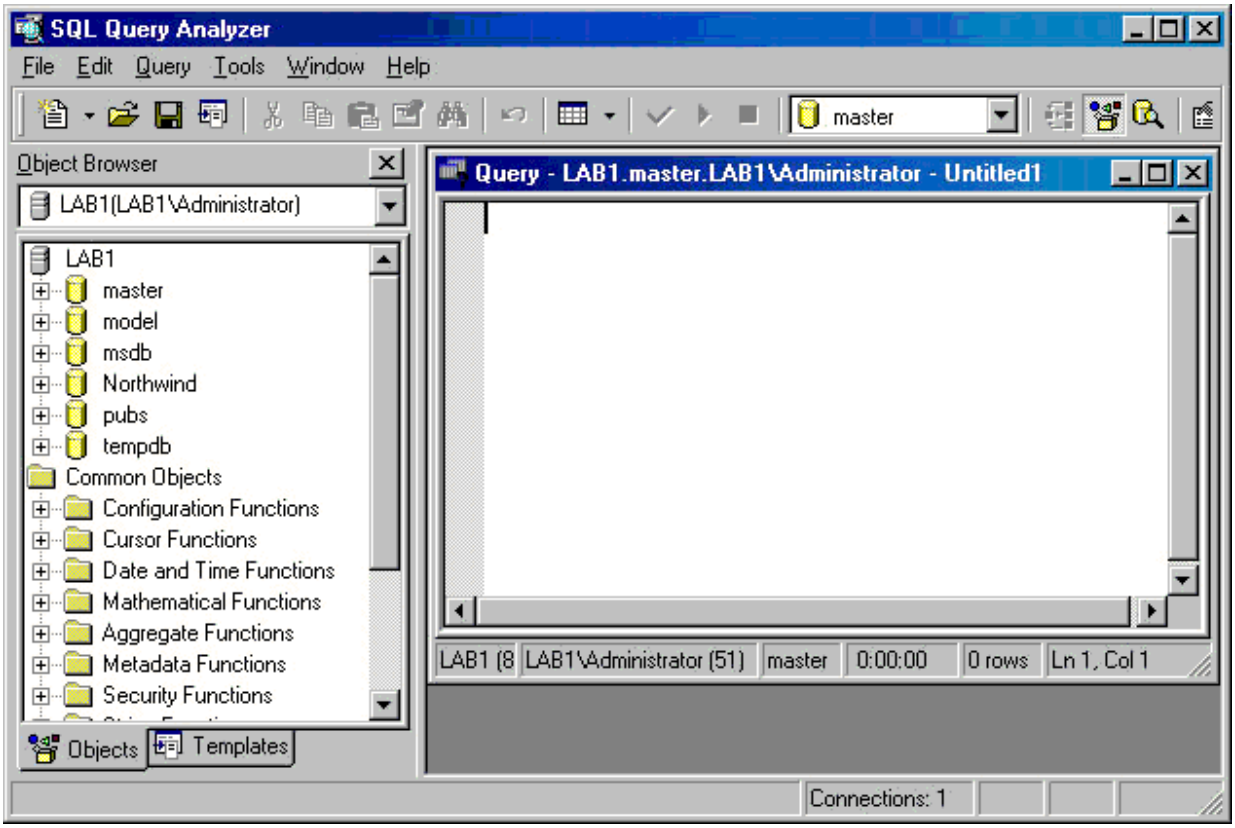
Delete

```
USE Northwind
DELETE FROM Territories
WHERE TerritoryID = 98101
```

Vì phần này khá căn bản nên chúng tôi thiết nghĩ không cần giải thích nhiều. Chú ý trong lệnh Delete bạn có thể có chữ From hay không đều được.

Nhưng mà chúng ta sẽ chạy thử các ví dụ trên ở đâu?

Để chạy các câu lệnh thí dụ ở trên bạn cần sử dụng và làm quen với **Query Analyser**



Cú Pháp Của T-SQL:

Phần này chúng ta sẽ bàn về các thành phần tạo nên cú pháp của T-SQL

Identifiers

Đây chính là tên của các database object. Nó dùng để xác định một object. (Chú ý khi nói đến Object trong SQL Server là chúng ta muốn đề cập đến table, view, stored procedure, index.....Vì hầu như mọi thứ trong SQL Server đều được thiết kế theo kiểu hướng đối tượng (object-oriented)). Trong ví dụ sau TableX, KeyCol, Description là những identifiers

```
CREATE TABLE TableX
(KeyCol INT PRIMARY KEY, Description NVARCHAR(80))
```

Có hai loại Identifiers một loại thông thường (**Regular Identifier**) và một loại gọi là **Delimited Identifier**, loại này cần có dấu "" hay dấu [] để ngăn cách. Loại Delimited được dùng đối với các chữ trùng với từ khóa của SQL Server (reserved keyword) hay các chữ có khoảng trống.

Ví dụ:

```
SELECT * FROM [My Table]
WHERE [Order] = 10
```

Trong ví dụ trên chữ Order trùng với keyword Order nên cần đặt trong dấu ngoặc vuông [].

Variables (Biến)

Biến trong T-SQL cũng có chức năng tương tự như trong các ngôn ngữ lập trình khác nghĩa là cần khai báo trước loại dữ liệu trước khi sử dụng. Biến được bắt đầu bằng dấu @ (Đối với các global variable thì có hai dấu @@)

Ví dụ:

```
USE Northwind
DECLARE @EmpIDVar INT
SET @EmpIDVar = 3
SELECT * FROM Employees
WHERE EmployeeID = @EmpIDVar + 1
```

Functions (Hàm)

Có 2 loại hàm một loại là built-in và một loại user-defined

Các hàm **Built-In** được chia làm 3 nhóm:

- **Rowset Functions** : Loại này thường trả về một object và được đối xử như một table. Ví dụ như hàm OPENQUERY sẽ trả về một recordset và có thể đứng vị trí của một table trong câu lệnh Select.
- **Aggregate Functions** : Loại này làm việc trên một số giá trị và trả về một giá trị đơn hay là các giá trị tổng. Ví dụ như hàm AVG sẽ trả về giá trị trung bình của một cột.
- **Scalar Functions** : Loại này làm việc trên một giá trị đơn và trả về một giá trị đơn. Trong loại này lại chia làm nhiều loại nhỏ như các hàm về toán học, về thời gian, xử lý kiểu dữ liệu String....Ví dụ như hàm MONTH('2002-09-30') sẽ trả về tháng 9.

Các hàm **User-Defined** (được tạo ra bởi câu lệnh CREATE FUNCTION và phần body thường được gói trong cặp lệnh BEGIN...END) cũng được chia làm các nhóm như sau:

- **Scalar Functions** : Loại này cũng trả về một giá trị đơn bằng câu lệnh RETURNS.
- **Table Functions** : Loại này trả về một table

Data Type (Loại Dữ Liệu)

Các loại dữ liệu trong SQL Server sẽ được bàn kỹ trong các bài sau

Expressions

Các Expressions có dạng **Identifier** + **Operators** (như +,-,*,/,=...) + **Value**

Các thành phần Control-Of Flow

Như BEGIN...END, BREAK, CONTINUE, GOTO, IF...ELSE, RETURN, WHILE.... Xin xem thêm Books Online để biết thêm về các thành phần này.

Comments (Chú Thích)

T-SQL dùng dấu -- để đánh dấu phần chú thích cho câu lệnh đơn và dùng /*...*/ để chú thích cho một nhóm

Thực Thi Các Câu Lệnh SQL

Thực thi một câu lệnh đơn:

Một câu lệnh SQL được phân ra thành các thành phần cú pháp như trên bởi một parser, sau đó SQL Optimizer (một bộ phận quan trọng của SQL Server) sẽ phân tích và tìm cách thực thi (Execute Plan) tối ưu nhất ví dụ như cách nào nhanh và tốn ít tài nguyên của máy nhất... và sau đó SQL Server Engine sẽ thực thi và trả về kết quả.

Thực Thi một nhóm lệnh (Batches)

Khi thực thi một nhóm lệnh SQL Server sẽ phân tích và tìm biện pháp tối ưu cho các câu lệnh như một câu lệnh đơn và chứa execution plan đã được biên dịch (compiled) trong bộ nhớ sau đó nếu nhóm lệnh trên được gọi lại lần nữa thì SQL Server không cần biên dịch mà có thể thực thi ngay điều này giúp cho một batch chạy nhanh hơn.

Lệnh GO

Lệnh này chỉ dùng để gởi một tín hiệu cho SQL Server biết đã kết thúc một batch job và yêu cầu thực thi. Nó vốn không phải là một lệnh trong T-SQL.

Tóm lại trong phần này chúng ta đã tìm hiểu về Transact- SQL là ngôn ngữ chính để giao tiếp với SQL Server. Trong bài sau chúng ta sẽ tiếp tục bàn về cấu trúc bên trong của SQL Server .

Design and Implement a SQL Server Database

Cấu Trúc Của SQL Server

Như đã trình bày ở các bài trước một trong những đặc điểm của SQL Server 2000 là **Multiple-Instance** nên khi nói đến một (SQL) Server nào đó là ta nói đến một Instance của SQL Server 2000, thông thường đó là Default Instance. Một Instance của SQL Server 2000 có 4 system databases và một hay nhiều user database. Các system databases bao gồm:

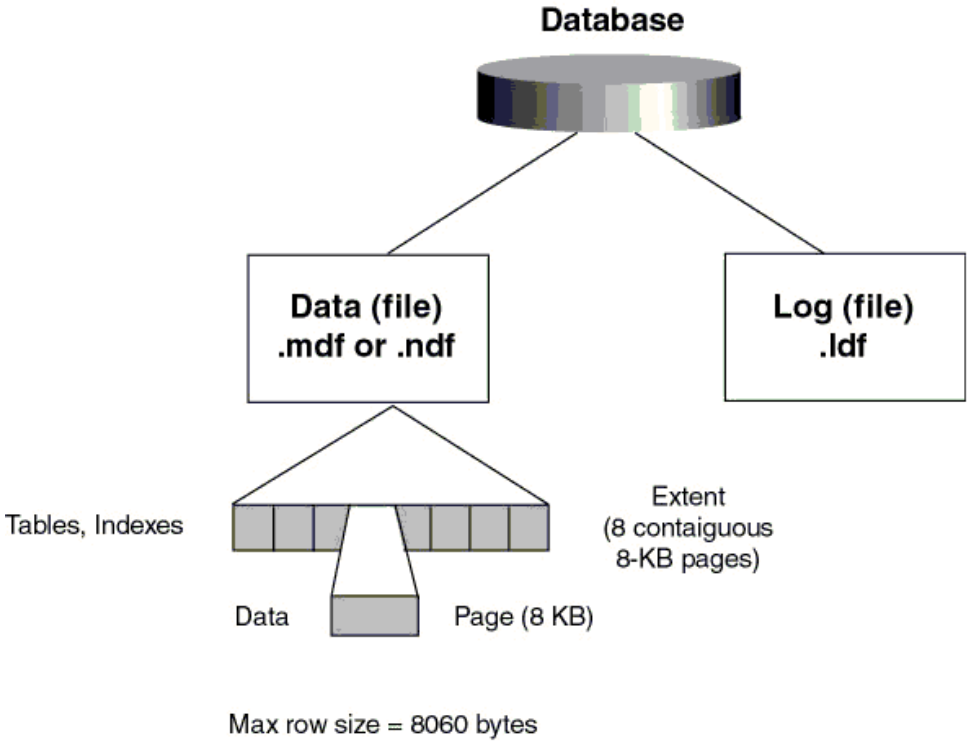
- **Master** : Chứa tất cả những thông tin cấp hệ thống (system-level information) bao gồm thông tin về các database khác trong hệ thống như vị trí của các data files, các login account và các thiết đặt cấu hình hệ thống của SQL Server (system configuration settings).
- **Tempdb** : Chứa tất cả những table hay stored procedure được tạm thời tạo ra trong quá trình làm việc bởi user hay do bản thân SQL Server engine. Các table hay stored procedure này sẽ biến mất khi khởi động lại SQL Server hay khi ta disconnect.
- **Model** : Database này đóng vai trò như một bảng kẻm (template) cho các database khác. Nghĩa là khi một user database được tạo ra thì SQL Server sẽ copy toàn bộ các system objects (tables, stored procedures...) từ Model database sang database mới vừa tạo.
- **Msdb** : Database này được SQL Server Agent sử dụng để hoạch định các báo động và các công việc cần làm (schedule alerts and jobs).

Cấu Trúc Vật Lý Của Một SQL Server Database

Mỗi một database trong SQL Server đều chứa ít nhất một data file chính (primary), có thể có thêm một hay nhiều data file phụ (Secondary) và một transaction log file.

- **Primary data file** (thường có phần mở rộng **.mdf**) : đây là file chính chứa data và những system tables.
- **Secondary data file** (thường có phần mở rộng **.ndf**) : đây là file phụ thường chỉ sử dụng khi database được phân chia để chứa trên nhiều dĩa.
- **Transaction log file** (thường có phần mở rộng **.ldf**) : đây là file ghi lại tất cả những thay đổi diễn ra trong một database và chứa đầy đủ thông tin để có thể roll back hay roll forward khi cần.

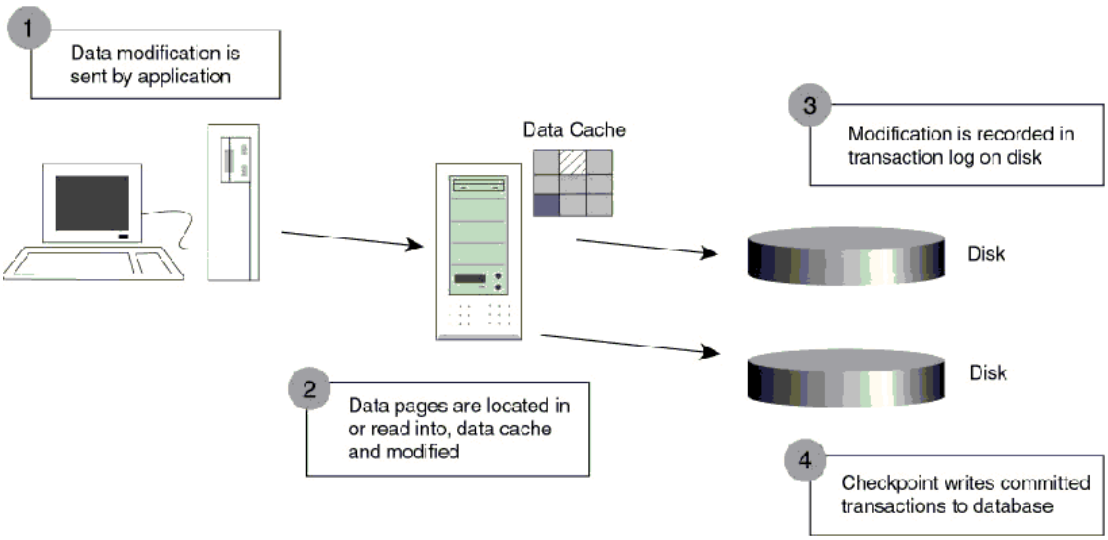
Data trong SQL Server được chứa thành từng **Page** 8KB và 8 page liên tục tạo thành một **Extent** như hình vẽ dưới đây:



Trước khi SQL Server muốn lưu data vào một table nó cần phải dành riêng một khoảng trống trong data file cho table đó. Những khoảng trống đó chính là các extents. Có 2 loại Extents: **Mixed Extents** (loại hỗn hợp) dùng để chứa data của nhiều tables trong cùng một Extent và **Uniform Extent** (loại thuần nhất) dùng để chứa data của một table. Đầu tiên SQL Server dành các Page trong Mixed Extent để chứa data cho một table sau đó khi data tăng trưởng thì SQL dành hẳn một Uniform Extent cho table đó.

Nguyên Tắc Hoạt Động Của Transaction Log Trong SQL Server

Transaction log file trong SQL Server dùng để ghi lại các thay đổi xảy ra trong database. Quá trình này diễn ra như sau: đầu tiên khi có một sự thay đổi data như Insert, Update, Delete được yêu cầu từ các ứng dụng, SQL Server sẽ tải (load) data page tương ứng lên memory (vùng bộ nhớ này gọi là data cache), sau đó data trong data cache được thay đổi(những trang bị thay đổi còn gọi là *dirty-page*). Tiếp theo mọi sự thay đổi đều được ghi vào transaction log file cho nên người ta gọi là *write-ahead* log. Cuối cùng thì một quá trình gọi là **Check Point Process** sẽ kiểm tra và viết tất cả những transaction đã được commited (hoàn tất) vào dĩa cứng (flushing the page).

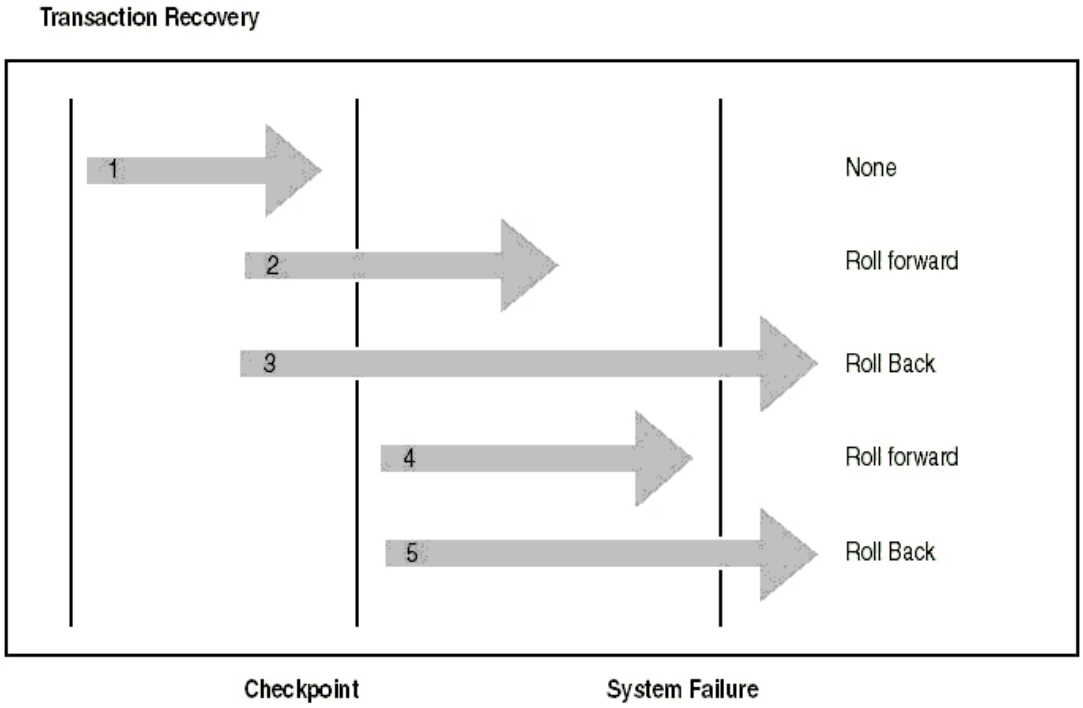


Ngoài Check Point Process những dirty-page còn được đưa vào dĩa bởi một **Lazy writer**. Đây là một anh chàng làm việc âm thầm chỉ thức giấc và quét qua phần data cache theo một chu kỳ nhất định sau đó lại ngủ yên chờ lần quét tới.

Xin giải thích thêm một chút về khái niệm transaction trong database. Một transaction hay một giao dịch là một loạt các hoạt động xảy ra được xem như một công việc đơn (unit of work) nghĩa là hoặc thành công toàn bộ hoặc không làm gì cả (all or nothing). Sau đây là một ví dụ cổ điển về transaction:

```
Chúng ta muốn chuyển một số tiền $500 từ account A sang
account B như vậy công việc này cần làm các bước sau:
         1.    Trừ $500 từ account A
         2.    Cộng $500 vào account B
Tuy nhiên việc chuyển tiền trên phải được thực hiện dưới dạng một
transaction nghĩa là giao dịch chỉ được
xem là hoàn tất (commited) khi cả hai bước trên đều thực hiện thành
công. Nếu vì một lý do nào đó ta chỉ
có thể thực hiện được bước 1 (chẳng hạn như vừa xong bước 1 thì điện
cúp hay máy bị treo) thì xem như giao
dịch không hoàn tất và cần phải được phục hồi lại trạng thái ban đầu
(roll back).
```

Thế thì Check Point Process hoạt động như thế nào để có thể đảm bảo một transaction được thực thi mà không làm "dơ" database.



Trong hình vẽ trên, một transaction được biểu diễn bằng một mũi tên. Trục nằm ngang là trục thời gian. Giả sử một Check Point được đánh dấu vào thời điểm giữa transaction 2 và 3 như hình vẽ và sau đó sự cố xãy ra trước khi gặp một Check point kế tiếp. Như vậy khi SQL Server được restart nó sẽ dựa trên những gì ghi trong transaction log file để phục hồi data (xem hình vẽ).

Điều đó có nghĩa là SQL Server sẽ không cần làm gì cả đối với transaction 1 vì tại thời điểm Check point data đã được lưu vào dĩa rồi. Trong khi đó transaction 2 và 4 sẽ được roll forward vì tuy đã được commited nhưng do sự cố xảy ra trước thời điểm check point kế tiếp nên data chưa kịp lưu vào dĩa. Tức là dựa trên những thông tin được ghi trên log file SQL Server hoàn toàn có đầy đủ cơ sở để viết vào dĩa cứng. Còn transaction 3 và 5 thì chưa được commited (do bị down bất ngờ) cho nên SQL Server sẽ roll back hai transaction này dựa trên những gì được ghi trên log file.

Cấu Trúc Logic Của Một SQL Server Database

Hầu như mọi thứ trong SQL Server được tổ chức thành những objects ví dụ như tables, views, stored procedures, indexes, constraints.... Những system objects trong SQL Server thường có bắt đầu bằng chữ *sys* hay *sp*. Các objects trên sẽ được nghiên cứu lần lượt trong các bài sau do đó trong phần này chúng ta chỉ bàn sơ qua một số system object thông dụng trong SQL Server database mà thôi.

Một số Sytem objects thường dùng:

System Stored Procedure	Ứng dụng
Sp_help *['object']*	Cung cấp thông tin về một database object (table, view...) hay một data type.
Sp_helpdb *['database']*	Cung cấp thông tin về một database cụ thể nào đó.
Sp_monitor	Cho biết độ bận rộn của SQL Server
Sp_spaceused *['object', 'updateusage']*	Cung cấp thông tin về các khoảng trống đã được sử dụng cho một object nào đó
Sp_who ['*login*']	Cho biết thông tin về một SQL Server user

Ví dụ:

sp_helpdb 'Northwind' sẽ cho kết quả có dạng như bảng dưới đây

```
name        db_size    owner   dbid   created       status .....
------------------------------------------------------------------------------------------------------
---------------------------- -------
Northwind   3.94 MB    sa      6      Aug 6 2000    Status=ONLINE,
Updateability=READ_WRITE, .....
```

stored procedure sp_spaceused như ví dụ sau

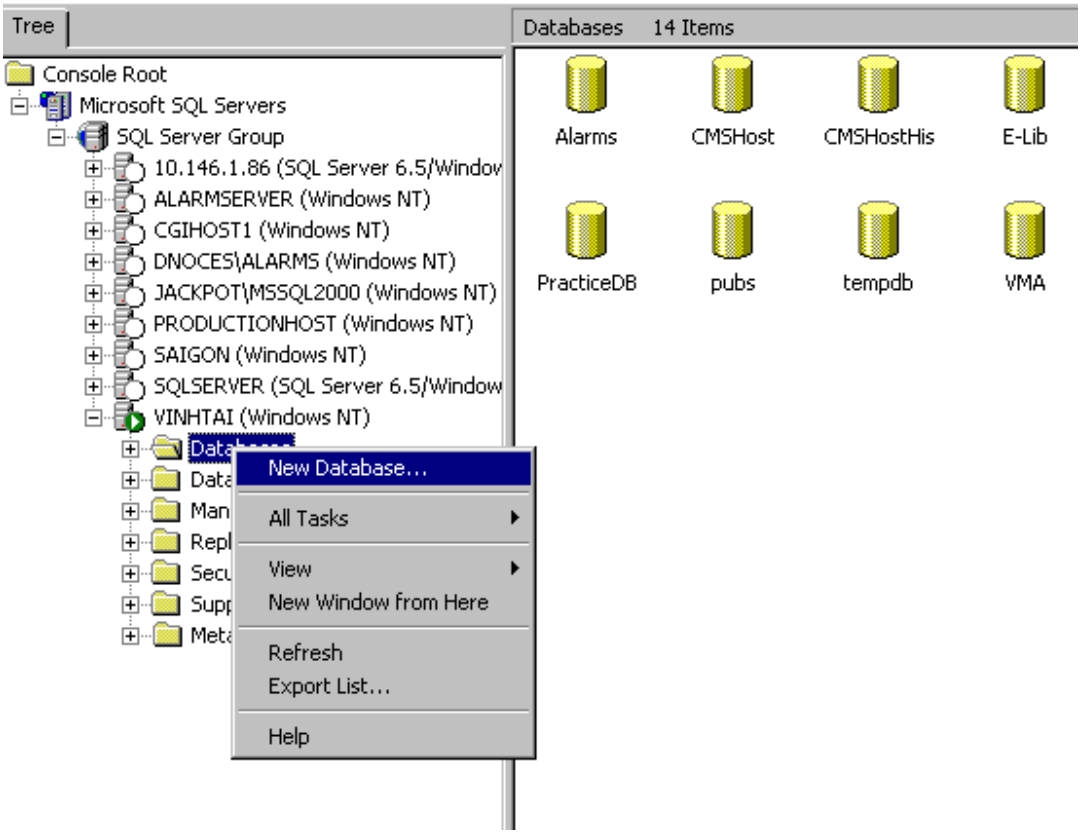
```
USE Northwind
Go
sp_spaceused 'Customers'
```

sẽ cho biết thông tin về table Customer:

```
name        rows    reserved     data      index_size    unused
----------------------------------- ----------------------------------------
Customers   91      104 KB       24 KB     80 KB          0 KB
```

Tạo Một User Database

Chúng ta có thể tạo một database dễ dàng dùng SQL Server Enterprise bằng cách right-click lên trên "database" và chọn "New Database" như hình vẽ sau:



Sau đó chúng ta chỉ việc đánh tên của database và click OK.

Ngoài ra đôi khi chúng ta cũng dùng SQL script để tạo một database. Khi đó ta phải chỉ rõ vị trí của primary data file và transaction log file.

Ví dụ:

```
USE master
GO
CREATE DATABASE Products
ON
(  NAME = prods_dat,
   FILENAME = 'c:\program files\microsoft SQL
server\mssql\data\prods.mdf',
   SIZE = 4,
   MAXSIZE = 10,
   FILEGROWTH = 1
)
GO
```

Trong ví dụ trên ta tạo một database tên là Products với logical file name là prods_dat và physical file name là prods.mdf, kích thước ban đầu là 4 MB và data file sẽ tự động tăng lên mỗi lần 1 MB cho tới tối đa là 10 MB. Nếu ta không

chỉ định một transaction log file thì SQL sẽ tự động tạo ra 1 log file với kích thước ban đầu là 1 MB.

Lưu Ý:

Khi tạo ra một database chúng ta cũng phải lưu ý một số điểm sau: Đối với các hệ thống nhỏ mà ở đó vấn đề tốc độ của server không thuộc loại nhạy cảm thì chúng ta thường chọn các giá trị mặc định (default) cho **Initial size**, **Automatically growth file**. Nhưng trên một số production server của các hệ thống lớn kích thước của database phải được người DBA ước lượng trước tùy theo tầm cỡ của business, và thông thường người ta không chọn Autogrowth(tự động tăng trưởng) và Autoshrink(tự động nén). Câu hỏi được đặt ra ở đây là vì sao ta không để SQL Server chọn một giá trị khởi đầu cho datafile và sau đó khi cần thì nó sẽ tự động nới rộng ra mà lại phải ước lượng trước? Nguyên nhân là nếu chọn Autogrowth (hay Autoshrink) thì chúng ta có thể sẽ gặp 2 vấn đề sau:

- **Performance hit:** Ảnh hưởng đáng kể đến khả năng làm việc của SQL Server. Do nó phải thường xuyên kiểm tra xem có đủ khoảng trống cần thiết hay không và nếu không đủ nó sẽ phải mở rộng bằng cách dành thêm khoảng trống từ dĩa cứng và chính quá trình này sẽ làm chậm đi hoạt động của SQL Server.
- **Disk fragmentation** : Việc mở rộng trên cũng sẽ làm cho data không được liên tục mà chứa ở nhiều nơi khác nhau trong dĩa cứng điều này cũng gây ảnh hưởng lên tốc độ làm việc của SQL Server.

Trong các hệ thống lớn người ta có thể dự đoán trước kích thước của database bằng cách tính toán kích thước của các tables, đây cũng chỉ là kích thước ước đoán mà thôi (xin xem "Estimating the size of a database" trong SQL Books Online để biết thêm về cách tính) và sau đó thường xuyên dùng một số câu lệnh SQL (thường dùng các câu lệnh bắt đầu bằng **DBCC** .Phần này sẽ được bàn qua trong các bài sau) kiểm tra xem có đủ khoảng trống hay không nếu không đủ ta có thể chọn một thời điểm mà SQL server ít bận rộn nhất (như ban đêm hay sau giờ làm việc) để nới rộng data file như thế sẽ không làm ảnh hưởng tới performance của Server.

Chú ý giả sử ta dành sẵn 2 GB cho datafile, khi dùng Window Explorer để xem ta sẽ thấy kích thước của file là 2 GB nhưng data thực tế có thể chỉ chiếm vài chục MB mà thôi.

Những Điểm Cần Lưu Ý Khi Thiết Kế Một Database

Trong phạm vi bài này chúng ta không thể nói sâu về lý thuyết thiết kế database mà chỉ đưa ra một vài lời khuyên mà bạn nên tuân theo khi thiết kế.

Trước hết bạn phải nắm vững về các loại **data type**. Ví dụ bạn phải biết rõ sự khác biệt giữa **char(10)**, **nchar(10) varchar(10)**, **nvarchar(10)**. Loại dữ liệu Char là một loại string có kích thước cố định nghĩa là trong ví dụ trên nếu data đưa vào "This is a really long character string" (lớn hơn 10 ký tự) thì SQL Server sẽ tự động cắt phần đuôi và ta chỉ còn "This is a". Tương tự nếu string đưa vào nhỏ hơn 10 thì SQL sẽ thêm khoảng trống vào phía sau cho đủ 10 ký tự. Ngược lại loại varchar sẽ không thêm các khoảng trống phía sau khi string đưa vào ít hơn 10. Còn loại data bắt đầu bằng chữ **n** chứa dữ liệu dạng unicode.

Một lưu ý khác là trong SQL Server ta có các loại Integer như : **tinyint**, **smallint**, **int**, **bigint**. Trong đó kích thước từng loại tương ứng là 1,2,4,8 bytes. Nghĩa là loại **smallint** tương đương với **Integer** và loại **int** tương đương với **Long** trong VB.

Khi thiết kế table nên:

- Có ít nhất một cột thuộc loại **ID** dùng để xác định một record dễ dàng.
- Chỉ chứa data của một entity (một thực thể)

Trong ví dụ sau thông tin về Sách và Nhà Xuất Bản được chứa trong cùng một table

Books

BookID	Title	Publisher	PubState	PubCity	PubCountry
1	Inside SQL Server 2000	Microsoft Press	CA	Berkely	USA
2	Windows 2000 Server	New Riders	MA	Boston	USA
3	Beginning Visual Basic 6.0	Wrox	CA	Berkely	USA

Ta nên tách ra thành table Books và table Publisher như sau:

Books

BookID	Title	PublisherID
1	Inside SQL Server 2000	P1
2	Windows 2000 Server	P2
3	Beginning Visual Basic 6.0	P3

và

Publishers

PublisherID	Publisher	PubState	PubCity	PubCountry
P1	Microsoft Press	CA	Berkely	USA
P2	New Riders	MA	Boston	USA
P3	Wrox	CA	Berkely	USA

- Tránh dùng cột có chứa **NULL** và nên luôn có giá trị **Default** cho các cột
- Tránh lập lại một giá trị hay cột nào đó

Ví dụ một cuốn sách có thể được viết bởi hơn một tác giả và như thế ta có thể dùng một trong 2 cách sau để chứa data:

Books

BookID	Title	Authors
1	Inside SQL Server 2000	John Brown
2	Windows 2000 Server	Matthew Bortniker, Rick Johnson
3	Beginning Visual Basic 6.0	Peter Wright, James Moon, John Brown

hay

Books

BookID	Title	Author1	Author2	Author3
1	Inside SQL Server 2000	John Brown	Null	Null
2	Windows 2000 Server	Matthew Bortniker	Rick Johnson	Null
3	Beginning Visual Basic 6.0	Peter Wright	James Moon	John Brown

Tuy nhiên việc lập đi lập lại cột Author sẽ tạo nhiều vấn đề sau này. Chẳng hạn như nếu cuốn sách có nhiều hơn 3 tác giả thì chúng ta sẽ gặp phiền phức ngay....Trong ví dụ này ta nên chặt ra thành 3 table như sau:

Books

BookID	Title
1	Inside SQL Server 2000
2	Windows 2000 Server
3	Beginning Visual Basic 6.0

Authors

AuthID	First Name	Last Name
A1	John	Brown
A2	Matthew	Bortniker
A3	Rick	Johnson
A4	Peter	Wright
A5	James	Moon

AuthorBook

BookID	AuthID
1	A1
2	A2
2	A3
3	A4
3	A5
3	A1

Ngoài ra một trong những điều quan trọng là phải biết rõ quan hệ (**Relationship**) giữa các table:

- **One-to-One Relationships** : trong mối quan hệ này thì một hàng bên table A không thể liên kết với hơn 1 hàng bên table B và ngược lại.
- **One-to-Many Relationships** : trong mối quan hệ này thì một hàng bên table A có thể liên kết với nhiều hàng bên table B.
- **Many-to-Many Relationships** : trong mối quan hệ này thì một hàng bên table A có thể liên kết với nhiều hàng bên table B và một hàng bên table B cũng có thể liên kết với nhiều hàng bên table A. Như ta thấy trong ví dụ trên một cuốn sách có thể được viết bởi nhiều tác giả và một tác giả cũng có thể viết nhiều cuốn sách. Do đó mối quan hệ giữa Books và Authors là quan hệ Many to Many. Trong trường hợp này người ta thường dùng một table trung gian để giải quyết vấn đề (table AuthorBook).

Để có một database tương đối hoàn hảo nghĩa là thiết kế sao cho data chứa trong database không thừa không thiếu bạn cần biết thêm về các thủ thuật **Normalization**. Tuy nhiên trong phạm vi khóa học này chúng tôi không muốn bàn sâu hơn về đề tài này, bạn có thể xem thêm trong các sách dạy lý thuyết cơ sở dữ liệu.

Tóm lại trong bài này chúng ta đã tìm hiểu về cấu trúc của một SQL Server database và một số vấn đề cần biết khi thiết kế một database. Trong bài sau chúng ta sẽ bàn về Backup và Restore database như thế nào.

Backup And Restore SQL Server

Chiến Lược Phục Hồi Dữ Liệu (Data Restoration Strategy)

Có một điều mà chúng ta phải chú ý là hầu như bất kỳ database nào cũng cần được phục hồi vào một lúc nào đó trong suốt chu kỳ sống của nó. Là một người Database Administrator bạn cần phải giảm tối đa số lần phải phục hồi dữ liệu, luôn theo dõi, kiểm tra thường xuyên để phát hiện các trục trặc trước khi nó xảy ra. Phải dự phòng các biến cố có thể xảy ra và bảo đảm rằng có thể nhanh chóng phục hồi dữ liệu trong thời gian sớm nhất có thể được.

Các dạng biến cố hay tai họa có thể xảy ra là:

- Đĩa chứa data file hay Transaction Log File hay system file bị mất
- Server bị hư hỏng
- Những thảm họa tự nhiên như bão lụt, động đất, hỏa hoạn
- Toàn bộ server bị đánh cắp hoặc phá hủy
- Các thiết bị dùng để backup - restore bị đánh cắp hay hư hỏng
- Những lỗi do vô ý của user như lỡ tay delete toàn bộ table chẳng hạn
- Những hành vi mang tính phá hoại của nhân viên như cố ý đưa vào những thông tin sai lạc.
- Bị hack (nếu server có kết nối với internet).

Bạn phải tự hỏi khi các vấn đề trên xảy ra thì bạn sẽ làm gì và phải luôn có biện pháp đề phòng cụ thể cho từng trường hợp cụ thể. Ngoài ra bạn phải xác định thời gian tối thiểu cần phục hồi dữ liệu và đưa server trở lại hoạt động bình thường.

Các Loại Backup

Để có thể hiểu các kiểu phục hồi dữ liệu khác nhau bạn phải biết qua các loại backup trong SQL Server

- **Full Database Backups** : Copy tất cả data files trong một database . Tất cả những user data và database objects như system tables, indexes, user-defined tables đều được backup.

- **Differential Database Backups** : Copy những thay đổi trong tất cả data files kể từ lần full backup gần nhất.
- **File or File Group Backups** : Copy một data file đơn hay một file group.
- **Differential File or File Group Backups** : Tương tự như differential database backup nhưng chỉ copy những thay đổi trong data file đơn hay một file group.
- **Transaction Log Backups** : Ghi nhận một cách thứ tự tất cả các transactions chứa trong transaction log file kể từ lần transaction log backup gần nhất. Loại backup này cho phép ta phục hồi dữ liệu trở ngược lại vào một thời điểm nào đó trong quá khứ mà vẫn đảm bảo tính đồng nhất (consistent).

Trong lúc backup SQL Server cũng copy tất cả các hoạt động của database kể cả hoạt động xảy ra trong quá trình backup cho nên ta có thể backup trong khi SQL đang chạy mà không cần phải ngưng lại.

Recovery Models

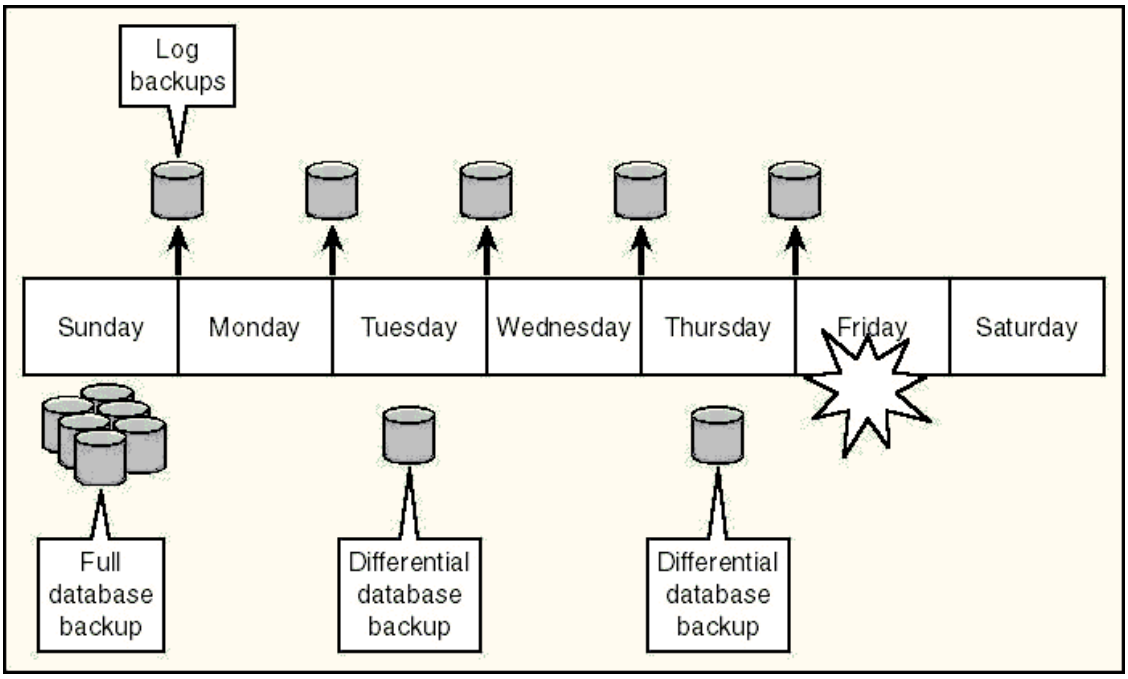
- **Full Recovery Model** : Đây là model cho phép phục hồi dữ liệu với ít rủi ro nhất. Nếu một database ở trong mode này thì tất cả các hoạt động không chỉ insert, update, delete mà kể cả insert bằng **Bulk Insert**, hay **bcp** đều được log vào transaction log file. Khi có sự cố thì ta có thể phục hồi lại dữ liệu ngược trở lại tới một thời điểm trong quá khứ. Khi data file bị hư nếu ta có thể backup được transaction log file thì ta có thể phục hồi database đến thời điểm transaction gần nhất được commited.
- **Bulk-Logged Recovery Model** : Ở mode này các hoạt động mang tính hàng loạt như Bulk Insert, bcp, Create Index, WriteText, UpdateText chỉ được log minimum vào transaction log file đủ để cho biết là các hoạt động này có diễn ra mà không log toàn bộ chi tiết như trong Full Recovery Mode. Các hoạt động khác như Insert, Update, Delete vẫn được log đầy đủ để dùng cho việc phục hồi sau này.
- **Simple Recovery Model** : Ở mode này thì Transaction Log File được truncate thường xuyên và không cần backup. Với mode này bạn chỉ có thể phục hồi tới thời điểm backup gần nhất mà không thể phục hồi tới một thời điểm trong quá khứ.

Muốn biết database của bạn đang ở mode nào bạn có thể **Right-click lên một database nào đó** trong SQL Server Enterprise Manager chọn **Properties->Options->Recovery**

Tuy nhiên có thể tới đây bạn cảm thấy rất khó hiểu về những điều trình bày ở trên. Chúng ta hãy dùng một ví dụ sau để làm rõ vấn đề.

Ví dụ:

Chúng ta có một database được áp dụng chiến lược backup như hình vẽ sau:



Trong ví dụ này ta schedule một Full Database Backup vào ngày Chủ Nhật và Differential Backup vào các ngày thứ Ba và Thứ Năm. Transaction Log Backup được schedule hằng ngày. Vào một ngày Thứ Sáu "đen tối" một sự cố xảy ra đó là đĩa chứa data file của database bị hư và là một DBA bạn được yêu cầu phải phục hồi dữ liệu và đưa database trở lại hoạt động bình thường. Bạn phải làm sao?

Trước hết bạn phải backup ngay Transaction Log File (Trong ví dụ này Transaction Log File được chứa trong một đĩa khác với đĩa chứa Data File nên không bị hư và vẫn còn hoạt động). Người ta còn gọi file backup trong trường hợp này là " the tail of the log" (cái đuôi). Nếu Log File được chứa trên cùng một đĩa với Data file thì bạn có thể sẽ không backup được "cái đuôi" và như vậy bạn phải dùng đến log file backup gần nhất. Khi backup "cái đuôi" này bạn cần phải dùng option **NO_TRUNCATE** bởi vì thông thường các Transaction Log Backup sẽ truncate(xoá) những phần không cần dùng đến trong transaction log file, đó là những transaction đã được commited và đã được viết vào database (còn gọi là inactive portion of the transaction log) để giảm kích thước của log file. Tuy nhiên khi backup phần đuôi không được truncate để đảm bảo tính consistent (nhất quán) của database.

Kế đến bạn phải restore database từ Full Backup File của ngày Chủ Nhật. Nó sẽ làm 2 chuyện : copy data, log, index... từ đĩa backup vào Data Files và sau đó sẽ lần lượt thực thi các transaction trong transaction log. Lưu ý ta phải dùng option **WITH NORECOVERY** trong trường hợp này (tức là option thứ 2 "**Leave database nonoperational but able to restore additional transaction logs**" trong Enterprise Manager). Nghĩa là các transaction chưa hoàn tất (incomplete transaction) sẽ **không được roll back**. Như vậy database lúc này sẽ ở trong tình trạng **inconsistent** và **không thể dùng được**. Nếu ta chọn **WITH RECOVERY** (hay "**Leave database operational. No additional transaction logs can be restored** " trong Enterprise Manager) thì các incomplete transaction sẽ được roll back và database ở trạng thái **consistent nhưng ta không thể nào restore các transaction log backup được nữa**.

Tiếp theo bạn phải restore Differential Backup của ngày Thứ Năm. Sau đó lần lượt restore các Transaction Log Backup kể từ sau lần Differential Backup cuối cùng nghĩa là restore Transaction Log Backup của ngày Thứ Năm và "Cái Đuôi". Như vậy ta có thể phục hồi data trở về trạng thái trước khi biến cố xảy ra. Quá trình này gọi là **Database Recovery**.

Cũng xin làm rõ cách dùng từ **Database Restoration** và **Database Recovery** trong SQL Server. Hai từ này nếu dịch ra tiếng Việt đều có nghĩa là phục hồi cơ sở dữ liệu nhưng khi đọc sách tiếng Anh phải cẩn thận vì nó có nghĩa hơi khác nhau.

Như trong ví dụ trên Khi ta **restore** database từ một file backup nghĩa là chỉ đơn giản tái tạo lại database từ những file backup và thực thi lại những transaction đã được commit nhưng database có thể ở trong trạng thái inconsistent và không sử dụng được. Nhưng khi nói đến **recover** nghĩa là ta không chỉ phục hồi lại data mà còn bảo đảm cho nó ở trạng thái consistent và sử dụng được (usable).

Có thể bạn sẽ hỏi **consistent** là thế nào? Phần này sẽ được nói rõ trong bài sau về Data Integrity. Nhưng cũng xin dùng một ví dụ đơn giản để giải thích. Trong ví dụ về thế nào là một transaction ở bài 3 : Giả sử số tiền $500 được trừ khỏi account A nhưng lại không được cộng vào account B và nếu database không được quá trình khôi phục dữ liệu tự động (automatic recovery process) của SQL rollback thì nó sẽ ở trạng thái inconsistent. Nếu database ở trạng thái giống như trước khi trừ tiền hoặc sau khi đã cộng $500 thành công vào account B thì gọi là consistent.

Cho nên việc backup Transaction Log File sẽ giúp cho việc recovery data tới bất kỳ thời điểm nào trong quá khứ. Đối với Simple Recovery Model ta chỉ có thể recover tới lần backup gần nhất mà thôi.

Như vậy khi restore database ta có thể chọn option WITH RECOVERY để roll back các transaction chưa được commited và database có thể hoạt động bình thường nhưng ta không thể restore thêm backup file nào nữa, thường option này được chọn khi restore file backup cuối cùng trong chuỗi backup. Nếu chọn option WITH NORECOVERY các transaction chưa được commited sẽ không được roll back do đó SQL Server sẽ không cho phép ta sử dụng database nhưng ta có thể tiếp tục restore các file backup kế tiếp, thường option này được chọn khi sau đó ta còn phải restore các file backup khác.

Không lẽ chỉ có thể chọn một trong hai option trên mà thôi hay sao? Không hoàn toàn như vậy ta có thể chọn một option trung lập hơn là option WITH STANDBY (tức là option 3 "**Leave database read-only and able to restore additional transaction logs**" trong Enterprise Manager). Với option này ta sẽ có luôn đặc tính của hai option trên : các incomplete transaction sẽ được roll back để đảm bảo database consistent và có thể sử dụng được nhưng chỉ dưới dạng Read-only mà thôi, đồng thời sau đó ta có thể tiếp tục restore các file backup còn lại (SQL Server sẽ log các transaction được roll back trong undo log file và khi ta restore backup file kế tiếp SQL Server sẽ trả lại trạng thái no recovery từ những gì ghi trên undo file). Người ta dùng option này khi muốn restore database trở lại một thời điểm nào đó (a point in time) nhưng không rõ là đó có phải là thời điểm mà họ muốn không, cho nên họ sẽ restore từng backup file ở dạng Standby và kiểm chứng một số data xem đó có phải là thời điểm mà họ muốn restore hay không (chẳng hạn như trước khi bị delete hay trước khi một transaction nào đó được thực thi) trước khi chuyển sang Recovery option.

Backup Database

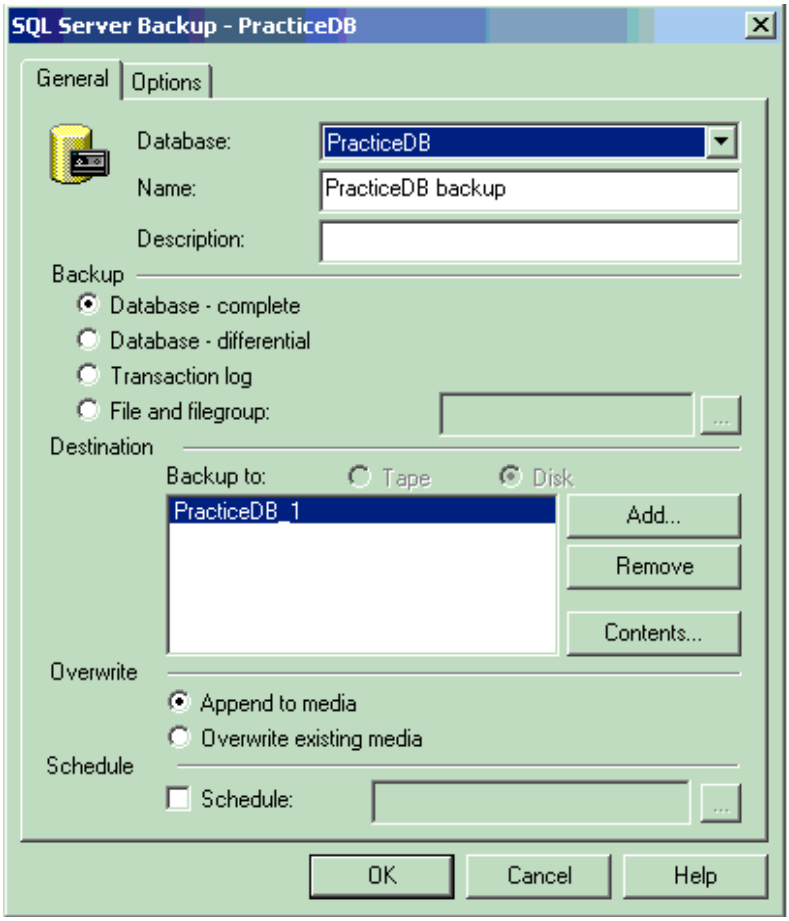
Trong phần này chúng ta sẽ bàn về cách backup database. Nhưng trước hết chúng ta hãy làm quen với một số thuật ngữ dùng trong quá trình backup và restore. Có những từ ta sẽ để nguyên tiếng Anh mà không dịch.

Thuật Ngữ	Giải Thích
Backup	Quá trình copy toàn bộ hay một phần của database, transaction log, file hay file group hình thành một backup set. Backup set được chứa trên backup media (tape or disk) bằng cách sử dụng một backup device (tape drive name hay physical filename)
Backup Device	Một file vật lý (như C:\SQLBackups\Full.bak) hay tape drive cụ thể (như \\.\Tape0) dùng để record một backup vào một backup media.
Backup File	File chứa một backup set
Backup Media	Disk hay tape được sử dụng để chứa một backup set. Backup media có thể chứa nhiều backup sets (ví dụ như từ nhiều SQL Server 2000 backups và từ nhiều Windows 2000 backups).
Backup Set	Một bộ backup từ một lần backup đơn được chứa trên backup media.

Chúng ta có thể tạo một backup device cố định (permanent) hay tạo ra một backup file mới cho mỗi lần backup. Thông thường chúng ta sẽ tạo một backup device cố định để có thể dùng đi dùng lại đặc biệt cho việc tự động hóa công việc backup. Để tạo một backup device dùng Enterprise Manager bạn chọn **Management->Backup** rồi **Right-click->New Backup Device**. Ngoài ra bạn có thể dùng **sp_addumpdevice** system stored procedure như ví dụ sau:

```
USE Master
Go
Sp_addumpdevice 'disk' , 'FullBackupDevice' , 'E:\SQLBackups\Full.bak'
```

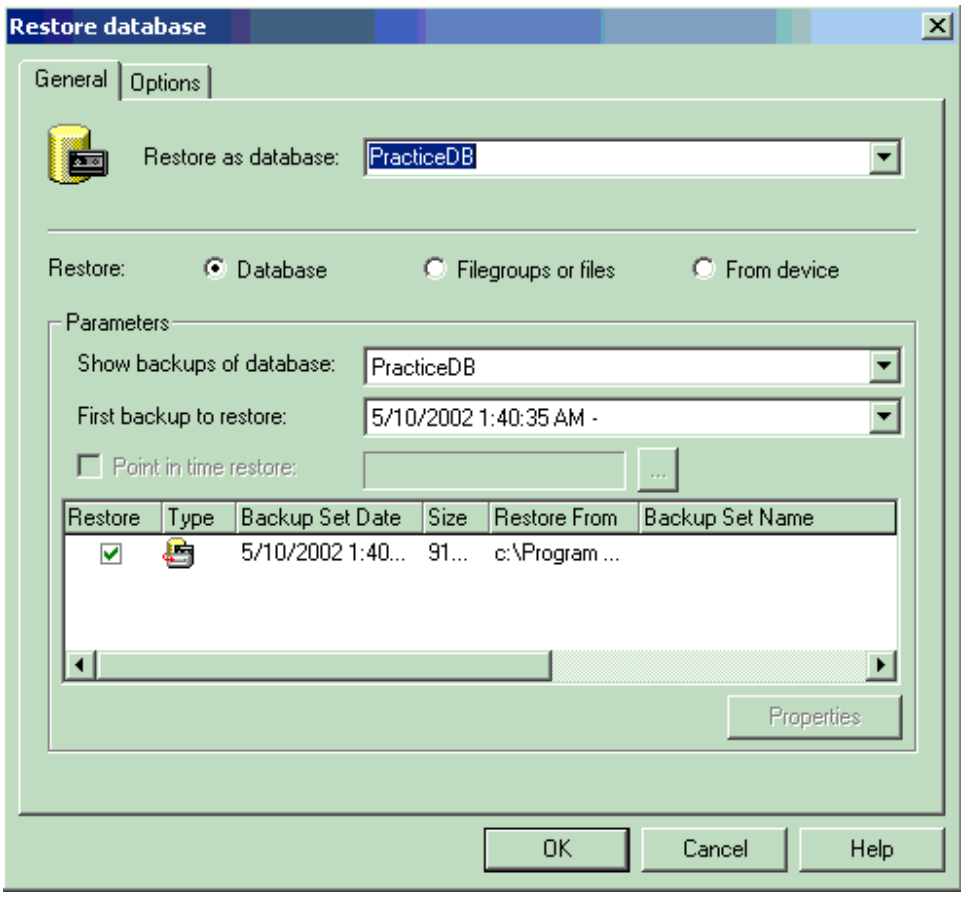
Để backup database bạn có thể dùng Backup Wizard hoặc click lên trên database muốn backup sau đó **Right-click->All Tasks->Backup Database...** sẽ hiện ra window như hình vẽ sau:



Sau đó dựa tùy theo yêu cầu của database mà chọn các option thích hợp. Ta có thể schedule cho SQL Server backup định kỳ.

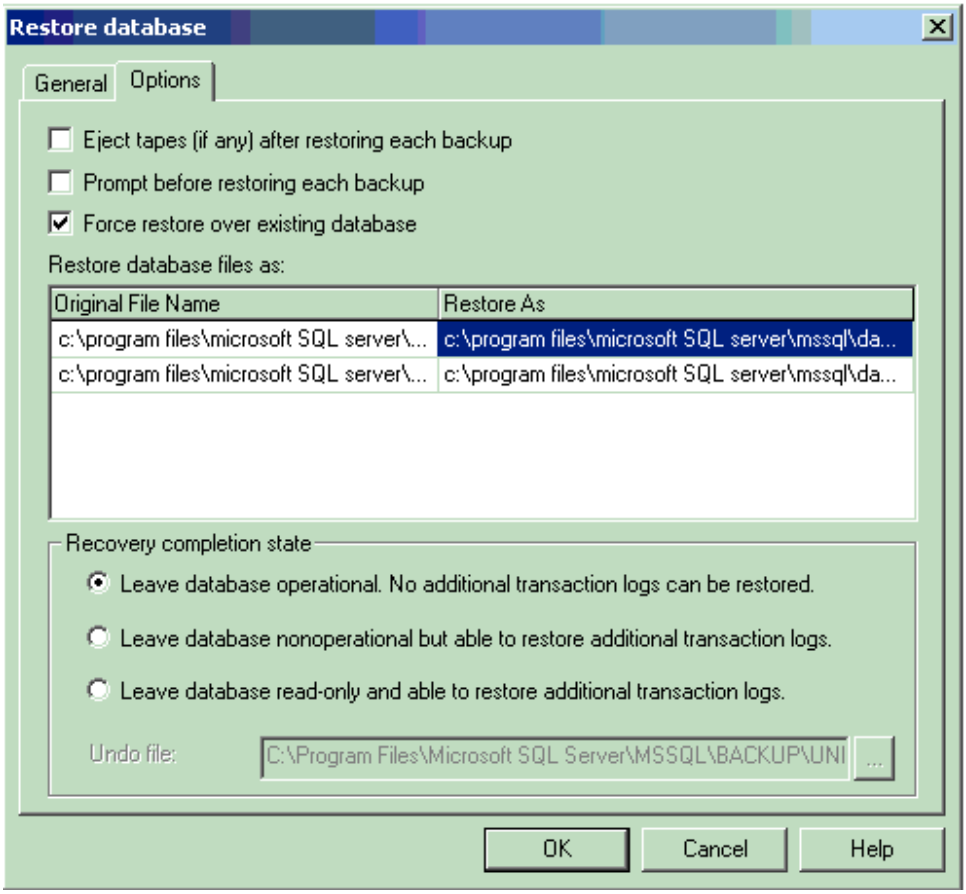
Restore Database

Trước khi restore database ta phải xác định được thứ tự file cần restore. Các thông tin này được SQL Server chứa trong **msdb** database và sẽ cho ta biết backup device nào, ai backup vào thời điểm nào. Sau đó ta tiến hành restore. Để restore bạn **Right-click->All Tasks->Restore database...** sẽ thấy window như hình vẽ sau:



Nếu bạn restore từ một instance khác của SQL Server hay từ một server khác bạn có chọn **From device** option và chọn backup device (file backup) tương ứng .

Lưu ý nếu bạn muốn overwrite database có sẵn với data được backup bạn có thể chọn option **Force restore over existing database** như hình vẽ sau:



Bạn có thể chọn leave database operational hay nonoperational tùy theo trường hợp như đã giải thích ở trên.

Tóm lại trong bài này chúng ta đã tìm hiểu một chút lý thuyết về backup và restore database trong SQL Server. Để có thể hiểu rõ hơn bạn cần phải thực tập hay làm thử để có thêm kinh nghiệm. Trong bài sau chúng ta sẽ bàn về đề tài Data Integrity nghĩa là làm sao để đảm bảo data chứa trong database là đáng tin cậy và không bị "lũng lỗ" như cách nói bình dân mà tôi thường hay dùng.

Data Integrity and Advanced Query Technique

Nói đến Data Integrity là ta nói đến tính toàn vẹn của một database hay nói một cách khác là data chứa trong database phải chính xác và đáng tin cậy. Nếu data chứa trong database không chính xác ta nói database mất tính toàn vẹn (lost data integrity). Trong bài này chúng ta sẽ bàn qua các phương pháp để giữ cho database được toàn vẹn.

Các Phương Pháp Đảm Bảo Data Integrity

SQL Server dùng một số cách để đảm bảo Data Integrity. Một số cách như Triggers hay Index sẽ được bàn đến trong các bài sau tuy nhiên trong phạm vi bài này chúng ta cũng nói sơ qua các cách trên.

- **Data Type** : Data type cũng có thể đảm bảo tính toàn vẹn của data ví dụ bạn khai báo data type của một cột là Integer thì bạn không thể đưa giá trị thuộc dạng String vào được.
- **Not Null Definitions** : Null là một loại giá trị đặc biệt, nó không tương đương với zero, blank hay empty string " " mà có nghĩa là không biết (unknown) hay chưa được định nghĩa (undefined). Khi thiết kế database ta nên luôn cẩn thận trong việc cho phép một cột được Null hay Not Null vì việc chứa Null data có thể làm cho một số ứng dụng vốn không xửa lý null data kỹ lưỡng bị "té".
- **Default Definitions** : Nếu một cột được cho một giá trị default thì khi bạn không đưa vào một giá trị cụ thể nào thì SQL Server sẽ dùng giá trị mặc định này. Bạn phải dùng Default đối với Not Null definition.
- **Identity Properties** : Data thuộc dạng ID sẽ đảm bảo tính duy nhất của data trong table.
- **Constraints** : Đây sẽ là phần mà ta đào sâu trong bài này. Constraint tạm dịch là những ràng buộc mà ta dùng để đảm bảo tính toàn vẹn của data. Constraints là những quy luật mà ta áp đặt lên một cột để đảm bảo tính chính xác của dữ liệu được nhập vào.
- **Rules** : Đây là một object mang tính backward-compatible chủ yếu để tương thích với các version trước đây của SQL Server. Rules tương đương với CHECK Constraint trong SQL Server 2000 nhưng người ta có xu hướng sử dụng CHECK Constraint vì nó chính xác hơn và có thể đặt nhiều Constraints lên một cột trong khi đó chỉ có một rule cho một cột mà thôi. Chú ý rule là một object riêng và sau đó liên kết với một cột nào đó của table trong khi CHECK constraint là một thuộc tính của table nên có thể được tạo ra với lệnh CREATE TABLE.
- **Triggers** : Một loại stored procedure đặc biệt được thực thi một cách tự động khi một table được Update, Insert, hay Delete. Ví dụ ta muốn khi một món hàng được bán ra thì tổng số hàng hóa trong kho phải được giảm xuống (-1) chẳng hạn khi đó ta có thể dùng trigger để đảm bảo chuyện đó. Triggers sẽ được bàn kỹ trong các bài sau.
- **Indexes** : sẽ được bàn đến trong bài nói về Indexes.

Constraints

Constraints là những thuộc tính (property) mà ta áp đặt lên một table hay một cột để tránh việc lưu dữ liệu không chính xác vào database (invalid data). Thật ra NOT NULL hay DEFAULT cũng được xem là một dạng constraint nhưng chúng ta không bao gồm hai loại này ở đây mà chỉ trình bày 4 loại constraints là Primary Key Constraint, Unique Constraint, Foreign Key Constraint và Check Constraint.

Primary Key Constraint:

Một table thường có một hay nhiều cột có giá trị mang tính duy nhất để xác định một hàng bất kỳ trong table. Ta thường gọi là Primary Key và được tạo ra khi ta Create hay Alter một table với Primary Key Constraint.

Một table chỉ có thể có một Primary Key constraint. Có thể có nhiều cột tham gia vào việc tạo nên một Primary Key, các cột này không thể chứa Null và giá trị trong các cột thành viên có thể trùng nhau nhưng giá trị của tất cả các cột tạo nên Primary Key phải mang tính duy nhất.

Khi một Primary Key được tạo ra một Unique Index sẽ được tự động tạo ra để duy trì tính duy nhất. Nếu trong table đó chưa có Clustered Index thì một Unique + Clustered Index sẽ được tạo ra.

Có thể tạo ra Primary Key Constraints như sau:

```
CREATE TABLE Table1
        (Col1 INT PRIMARY KEY,
         Col2 VARCHAR(30)
        )
```

hay

```
CREATE TABLE Table1
        (Col1 INT,
         Col2 VARCHAR(30),
         CONSTRAINT table_pk PRIMARY KEY (Col1)
        )
```

Unique Constraint

Bạn có thể tạo Unique Constraint để đảm bảo giá trị của một cột nào đó không bị trùng lập. Tuy Unique Constraint và Primary Key Constraint đều đảm bảo tính duy nhất nhưng bạn nên dùng Unique Constraint trong những trường hợp sau:

- **Nếu một cột (hay một sự kết hợp giữa nhiều cột) không phải là primary key**. Nên nhớ chỉ có một Primary Key Constraint trong một table trong khi ta có thể có nhiều Unique Constraint trên một table.
- **Nếu một cột cho phép chứa Null**. Unique constraint có thể áp đặt lên một cột chứa giá trị Null trong khi primary key constraint thì không.

Cách tạo ra Unique Constraint cũng tương tự như Primary Key Constraint chỉ việc thay chữ Primary Key thành Unique. SQL Server sẽ tự động tạo ra một non-clustered unique index khi ta tạo một Unique Constraint.

Foreign Key Constraint

Foreign Key là một cột hay một sự kết hợp của nhiều cột được sử dụng để áp đặt mối liên kết data giữa hai table. Foreign key của một table sẽ giữ giá trị của Primary key của một table khác và chúng ta có thể tạo ra nhiều Foreign key trong một table.

Foreign key có thể reference (tham chiếu) vào Primary Key hay cột có Unique Constraints. Foreign key có thể chứa Null. Mặc dù mục đích chính của Foreign Key Constraint là để kiểm soát data chứa trong table có Foreign key (tức table con) nhưng thực chất nó cũng kiểm soát luôn cả data trong table chứa Primary key (tức table cha). Ví dụ nếu ta delete data trong table cha thì data trong table con trở nên "mồ côi" (orphan) vì không thể reference ngược về table cha. Do đó Foreign Key constraint sẽ đảm bảo điều đó không xảy ra. Nếu bạn muốn delete data trong table cha thì trước hết bạn phải drop hay disable Foreign key trong table con trước.

Có thể tạo ra Foreign Key Constraints như sau:

```
CREATE TABLE Table1
        (Col1 INT PRIMARY KEY,
         Col2 INT REFERENCES Employees(EmployeeID)
         )
```

hay

```
CREATE TABLE Table1
        (Col1 INT PRIMARY KEY,
         Col2 INT,
         CONSTRAINT col2_fk FOREIGN KEY (Col2)
         REFERENCES Employees (EmployeeID)
         )
```

Đôi khi chúng ta cũng cần Disable Foreign Key Constraint trong trường hợp:

- **Insert hay Update**: Nếu data insert vào sẽ vi phạm những ràng buộc có sẵn (violate constraint) hay constraint của ta chỉ muốn áp dụng cho data hiện thời mà thôi chứ không phải data sẽ insert.
- **Tiến hành quá trình replicate**. Nếu không disable Foreign Key Constraint khi replicate data thì có thể cản trở quá trình copy data từ source table tới destination table một cách không cần thiết.

Check Constraint

Check Constraint dùng để giới hạn hay kiểm soát giá trị được phép insert vào một cột. Check Constraint giống Foreign Key Constraint ở chỗ nó kiểm soát giá trị đưa vào một cột nhưng khác ở chỗ Foreign Key Constraint dựa trên giá trị ở table cha để cho phép một giá trị được chấp nhận hay không trong khi Check Constraint dựa trên một biểu thức logic (logic expression) để kiểm tra xem một giá trị có hợp lệ không. Ví dụ ta có thể áp đặt một Check Constraint lên cột salary để chỉ chấp nhận tiền lương từ $15000 đến $100000/năm.

Ta có thể tạo ra nhiều Check Constraint trên một cột. Ngoài ra ta có thể tạo một Check Constraint trên nhiều cột bằng cách tạo ra Check Constraint ở mức table (table level).

Có thể tạo ra Check Constraint như sau:

```
CREATE TABLE Table1
        (Col1 INT PRIMARY KEY,
         Col2 INT
         CONSTRAINT limit_amount CHECK (Col2 BETWEEN 0 AND
1000),
         Col3 VARCHAR(30)
         )
```

Trong ví dụ này ta giới hạn giá trị chấp nhận được của cột Col2 từ 0 đến 1000. Ví dụ sau sẽ tạo ra một Check Constraint giống như trên nhưng ở table level:

```
CREATE TABLE Table1
        (Col1 INT PRIMARY KEY,
         Col2 INT,
         Col3 VARCHAR(30),
         CONSTRAINT limit_amount CHECK (Col2 BETWEEN 0 AND 1000)
         )
```

Tương tự như Foreign Key Constraint đôi khi ta cũng cần disable Check Constraint trong trường hợp Insert hay Update mà việc kiểm soát tính hợp lệ của data không áp dụng cho data hiện tại. Trường hợp thứ hai là replication.

Muốn xem hay tạo ra Constraint bằng Enterprise Manager thì làm như sau:

Click lên trên một table nào đó và **chọn Design Table-> Click vào icon bên phải "Manage Constraints..."**

Advanced Query Techniques

Trong phần này chúng ta sẽ đào sâu một số câu lệnh nâng cao như SELECT, INSERT...

Có thể nói hầu như ai cũng biết qua câu lệnh căn bản kiểu như "SELECT * FROM TABLENAME WHERE..." nhưng có thể có nhiều người không biết đến những tính chất nâng cao của nó.

Cú pháp đầy đủ của một câu lệnh SELECT rất phức tạp tuy nhiên ở đây chỉ trình bày những nét chính của lệnh này mà thôi:

SELECT select_list
[INTO new_table]
FROM table_source [WHERE search_condition]
[GROUP BY group_by_expression]
[HAVING search_condition]
[ORDER BY order_expression [ASC | DESC]]

Chúng ta sẽ lần lượt nghiên cứu từng clause (mệnh đề) trong câu lệnh này.

SELECT Clause

Sau keyword (từ khóa) SELECT ta sẽ có một danh sách các cột mà ta muốn select được cách nhau bằng dấu ",". Có 3 Keywords cần nhấn mạnh trong phần SELECT.

- **Distinct** : Khi có keyword này vào thì sẽ cho kết quả các cột không trùng nhau. Ví dụ trong Orders table của Norwind database (database mẫu di kèm với SQL Server) chứa giá trị trùng lập (duplicate value) trong cột ShipCity. Nếu ta muốn select một danh sách ShipCity trong đó mỗi city chỉ xuất hiện một lần trong kết quả nhận được ta dùng như sau:

```
SELECT DISTINCT ShipCity, ShipRegion
FROM Orders
ORDER BY ShipCity
```

- **Top *n*** : Nếu ta muốn select *n* hàng đầu tiên mà thôi ta có thể dùng Top keyword. Nếu có thêm ORDER BY thì kết quả sẽ được order trước sau đó mới select. Chúng ta cũng có thể select số hàng

dựa trên phần trăm bằng cách thêm Keyword Percent vào. Ví dụ sau sẽ select 10 hàng đầu tiên theo thứ tự:

```
SELECT DISTINCT TOP 10 ShipCity, ShipRegion
FROM Orders
ORDER BY ShipCity
```

- **As** : Đôi khi chúng ta muốn cho SELECT statement dễ đọc hơn một chút ta có thể dùng một alias (tức là từ thay thế hay từ viết tắt) với keyword As hay không có keyword As: *table_name* As *table_alias* hay *table_name table_alias*. Ví dụ:

```
USE pubs
SELECT p.pub_id, p.pub_name AS PubName
FROM publishers AS p
```

Ngoài ra trong Select list ta có thể select dưới dạng một expression như sau:

```
SELECT FirstName + ' ' + LastName AS "Employee Name",
        IDENTITYCOL AS "Employee ID",
        HomePhone,
        Region
FROM Northwind.dbo.Employees
ORDER BY LastName, FirstName ASC
```

Trong ví dụ trên ta select cột "Employee Name" là sản phẩm ghép lại của cột FirstName và LastName được cách nhau bằng một khoảng trắng. Một giá trị thuộc loại identity để làm cột "Employee ID". Kết quả sẽ được sắp theo thứ tự từ nhỏ tới lớn (ASC) (còn DESC là từ lớn tới nhỏ) trong đó cột LastName được sắp trước rồi mới tới cột FirstName.

The INTO Clause

INTO Clause cho phép ta select data từ một hay nhiều table sau đó kết quả sẽ được insert vào một table mới. Table này được tạo ra do kết quả của câu lệnh SELECT INTO. Ví dụ:

```
SELECT FirstName, LastName
INTO EmployeeNames
FROM Employers
```

Câu lệnh tên sẽ tạo ra một table mới có tên là EmployeeNames với 2 cột là FirstName và LastName sau đó kết quả select được từ table Employers sẽ được insert vào table mới này. Nếu table EmployeeNames tồn tại SQL Server sẽ báo lỗi. Câu lệnh này thường hay được sử dụng để select một lượng data lớn từ nhiều table khác nhau vào một table mới (thường dùng cho mục đích tạm thời (temporary table)) mà khỏi phải thực thi câu lệnh Insert nhiều lần.

Một cách khác cũng select data từ một hay nhiều table và insert vào một table khác là dùng "**Insert Into...Select...**". Nhưng câu lệnh này không tạo ra một table mới. Nghĩa là ta table đó phải tồn tại trước. Ví dụ:

```
INSERT INTO EmployeeNames
SELECT FirstName, LastName
FROM Employers
```

Chú ý là không có chữ "**Value**" trong câu Insert này.

The GROUP BY and HAVING Clauses

GROUP BY dùng để tạo ra các giá trị tổng (aggregate values) cho từng hàng trong kết quả select được. Chỉ có một hàng cho từng giá trị riêng biệt (distinct) của từng cột. Các cột được select đều phải nằm trong GROUP BY Clause. Hãy xem ví dụ phức tạp sau:

```
SELECT OrdD1.OrderID AS OrderID,
       SUM(OrdD1.Quantity) AS "Units Sold",
       SUM(OrdD1.UnitPrice * OrdD1.Quantity) AS Revenue
FROM [Order Details] AS OrdD1
WHERE OrdD1.OrderID in (SELECT DISTINCT OrdD2.OrderID
                        FROM [Order Details] AS OrdD2
                        WHERE OrdD2.UnitPrice > $100)
GROUP BY OrdD1.OrderID
HAVING SUM(OrdD1.Quantity) > 100
```

Trong ví dụ trên đầu tiên ta select những order riêng biệt (distinct) từ Order Details table với giá > 100. Sau đó tiếp tục select OrderID, "Units Sold", Revenue từ kết quả trên trong đó "Units Sold" và Revenue là những aggregate columns (cho giá trị tổng một cột của những hàng có cùng OrderID). HAVING Clause đóng vai trò như một filter dùng để lọc lại các giá trị cần select mà thôi. HAVING Clause thường đi chung với GROUP BY mặc dù có thể xuất hiện riêng lẻ.

UNION

Uninon keyword có nhiệm vụ ghép nối kết quả của 2 hay nhiều queries lại thành một kết quả.

Ví dụ:

Giả sử có table1(ColumnA varchar(10), ColumnB int) và table2(ColumnC varchar(10), ColumnD int). Ta muốn select data từ table1 và ghép với data từ table2 để tạo thành một kết quả duy nhất ta làm như sau:

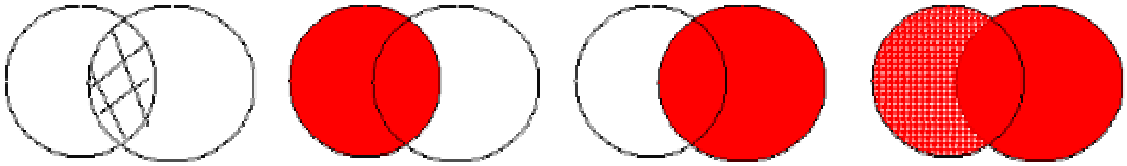
```
SELECT * FROM Table1
UNION ALL
SELECT * FROM Table2
```

Nếu không có keyword ALL thì những hàng giống nhau từ 2 table sẽ chỉ xuất hiện một lần trong kết quả. Còn khi dùng ALL thì các hàng trong 2 table đều có trong kết quả bất chấp việc lập lại.

Khi Dùng Union phải chú ý hai chuyện: số cột select ở 2 queries phải bằng nhau và data type của các cột tương ứng phải compatible (tương thích).

Using JOINS

Trong phần này chúng ta sẽ tìm hiểu về các loại Join trong SQL Server. Bằng cách sử dụng JOIN bạn có thể select data từ nhiều table dựa trên mối quan hệ logic giữa các table (logical relationships). Có thể tóm tắt các loại Join thông dụng bằng các hình sau:



Thứ tự từ trái sang phải: Inner Join, Left Outer Join, Right Outer Join, Full Outer Join

Inner Joins

Dùng Inner Join để select data từ 2 hay nhiều tables trong đó giá trị của các cột được join phải xuất hiện ở cả 2 tables tức là phần gạch chéo trên hình. Ví dụ:

```
SELECT t.Title, p.Pub_name
FROM Publishers AS p INNER JOIN Titles AS t
ON p.Pub_id = t.Pub_id
ORDER BY Title ASC
```

Left Outer Joins

Dùng Left Outer Join để select data từ 2 hay nhiều tables trong đó tất cả cột bên table thứ nhất và không tồn tại bên table thứ hai sẽ được select cộng với các giá trị của các cột được inner join. Số cột select được sẽ bằng với số cột của table thứ nhất. Tức là phần tô màu đỏ trên hình. Ví dụ:

```
USE Pubs
SELECT a.Au_fname, a.Au_lname, p.Pub_name
FROM Authors a LEFT OUTER JOIN Publishers p
ON a.City = p.City
ORDER BY p.Pub_name ASC, a.Au_lname ASC, a.Au_fname ASC
```

Right Outer Joins

Dùng Right Outer Join để select data từ 2 hay nhiều tables trong đó tất cả cột bên table thứ hai và không tồn tại bên table thứ nhất sẽ được select cộng với các giá trị của các cột được inner join. Số cột select được sẽ bằng với số cột của table thứ hai. Tức là phần tô màu đỏ trên hình. Ví dụ:

```
USE Pubs
SELECT a.Au_fname, a.Au_lname, p.Pub_name
FROM Authors a RIGHT OUTER JOIN Publishers p
ON a.City = p.City
ORDER BY p.Pub_name ASC, a.Au_lname ASC, a.Au_fname ASC
```

Full Outer Joins

Dùng Full Outer Join để select data từ 2 hay nhiều tables trong đó tất cả cột bên table thứ nhất và thứ hai đều được chọn các giá trị bên hai table bằng nhau thì chỉ lấy một lần. Tức là phần tô màu đỏ trên hình. Ví dụ:

```
USE Pubs
SELECT a.Au_fname, a.Au_lname, p.Pub_name
FROM Authors a FULL OUTER JOIN Publishers p
ON a.City = p.City
ORDER BY p.Pub_name ASC, a.Au_lname ASC, a.Au_fname ASC
```

Cross Joins

Dùng Cross Join ghép data từ hai table trong đó số hàng thu được bằng với số hàng của table thứ nhất nhân với số hàng của table thứ hai. Ví dụ:

```
USE pubs
SELECT au_fname, au_lname, pub_name
FROM authors CROSS JOIN publishers
WHERE authors.city = publishers.city
ORDER BY au_lname DESC
```

Để ý là trong câu lệnh này không có keyword "On".

Muốn hiểu rõ hơn về các loại join bạn cho chạy thử trên SQL Server và làm phần bài tập số 1.

Tóm lại trong bài này chúng ta đã tìm hiểu data integrity trong SQL Server bằng cách dùng các loại Constraint. Ngoài ra Chúng ta cũng biết qua về một số kỹ thuật query nâng cao. Sau bài học này các bạn cần làm bài tập số 1 để hệ thống hóa lại kiến thức đã học từ bài 1 đến bài 5 trước khi bạn học tiếp bài số 6. Khi làm bài tập nhớ phải làm theo thứ tự và tuân thủ theo các yêu cầu của bài tập đặt ra. Không nên bỏ qua bước nào.

Stored Procedure and Advanced T-SQL

Trong bài này chúng ta sẽ tìm hiểu một số cách import và export data trong SQL Server. Sau đó sẽ bàn qua các loại Stored Procedure và Cursor.

Sử dụng bcp và BULK INSERT để import data

bcp là một command prompt dùng để import hay export data từ một data file (Text file hay Excel File) vào SQL Server hay ngược lại. Thường khi muốn chuyển một số lượng lớn data từ một database system khác như Oracle, DB2...sang SQL Server trước hết ta sẽ export data ra một text file sau đó import vào SQL Server dùng bcp command. Một trường hợp thông dụng hơn là ta export data từ SQL Server sang một Microsoft Excel file và Excel file này có thể là input cho một program hay một database system khác.

Chúng ta cũng có thể chuyển data vào SQL Server dùng câu lệnh **BULK INSERT**. Tuy nhiên BULK INSERT **chỉ có thể import data** vào trong SQL Server chứ không thể export data ra một data file như bcp.

Để có thể insert data vào SQL Server Database, data file phải có dạng bảng nghĩa là có cấu trúc hàng và cột. Chú ý khi data được bulk copy (copy hàng loạt dùng bcp hay BULK INSERT) vào một table trong SQL Server thì table đó phải tồn tại và data được cộng thêm vào (append). Ngược lại khi export data ra một data file thì một file mới sẽ được tạo ra hoặc data file sẽ bị overwrite nếu nó tồn tại.

Cú pháp đầy đủ của lệnh bcp có thể xem trong SQL Server Books Online. Ở đây chỉ trình bày một số ví dụ đơn giản về cách sử dụng bcp command và BULK INSERT.

Ví dụ 1: Giả sử bạn muốn export data từ table Orders trong PracticeDB (đây là database được tạo ra trong bài tập số 1) ra một text file trong đó các cột được phân cách bằng dấu ";". Bạn có thể làm như sau: mở DOS command prompt và đánh vào dòng lệnh sau:

```
bcp PracticeDB..Orders out c:\Orders.txt -c -T -t;
```

Trong ví dụ trên ta muốn bulk copy table Orders ra một text file trong đó :

out: copy data từ table hay view ra một data file (c:\Orders.txt). Ngược lại ta có thể dùng switch **in** để import data từ text file vào SQL Server.

-c: bulk copy dùng kiểu dữ liệu Character (Char) (nếu không chỉ rõ thì SQL Server sẽ dùng "TAB" character (\t) để phân định các cột và dùng new line character (\n) để phân định các hàng như các giá trị default).

-t;: dấu ";" đi sau switch "t" cho biết ta muốn dùng ";" để phân định các cột (nếu không sẽ dùng giá trị mặc định như trên)

-T: dùng (NT) Trust connection để kết nối với database. Nghĩa là nếu user đã authenticated (cho phép) vào được Windows system thì đương nhiên được sử dụng SQL Server mà không cần dùng thêm username và password nào khác.

Ví dụ 2: Thay vì copy toàn bộ table ta có thể dùng query để select một phần data và export ra text file như sau:

```
        bcp "Select * From practiceDB..Orders" queryout c:\Orders.txt -c
-SVinhtai -Usa -Pabc
```

Trong ví dụ này ta select toàn bộ data trong Orders table ra một text file dùng query và SQL Server authentication.

queryout : cho biết đây là một query chứ không phải là table.

-S : tên của SQL Server (hay tên của một Instance)

-U : SQL user name dùng để log on

-P : password dùng để log on.

Ví dụ 3 : dùng BULK INSERT để bulk copy data từ text file vào SQL Server database. Mở Query Analyser (BULK INSERT là một T-SQL command chứ không phải là một command prompt utility) và đánh vào các dòng sau :

```
        BULK INSERT PracticeDB..Orders FROM 'c:\Orders.txt ' WITH
(DATAFILETYPE = 'CHAR')
```

Trong ví dụ trên DATAFILETYPE= 'CHAR' cho biết data được chứa dạng Char data type. Nêú muốn dùng data type dạng unicode thì dùng 'WIDECHAR'

Chú ý: Các switch trong **bcp** command là case-sensitive. Nghĩa là chữ hoa và chữ thường sẽ có ý nghĩa khác nhau.

Distributed Queries

Đôi khi chúng ta muốn select data từ những database system khác như MS Access, Oracle, DB2... hay thậm chí từ một SQL Server khác ta cần phải dùng distributed query. SQL Server sẽ dùng kỹ thuật OLEDB và các API để chuyển các query này tới các database system khác. Có 2 cách để truy cập vào các database system khác là dùng LINKED SERVER và Ad Hoc Computer Name.

Linked Server:

Linked Server là một server ảo được dùng để truy cập vào các database system khác. Một khi đã setup thì ta có thể query data dùng four-part name : linked_server_name.catalog.schema.object_name . Trong đó catalog thường tương đương với database name, Schema tương đương với database owner và object_name tương đương với table hay view.

Ví dụ: Giả sử ta setup một Linked Server vào Access database "PracticeDB.mdb" trong đó các table đều tương tự như PracticeDB database trong SQL Server (được tạo ra trong phần bài tập số 1).

Mở **Enterprise Manager -> Chọn node Security của local server -> Right-Click lên node Linked Server chọn New Linked Server**. Sau đó nhập vào tên của Linked Server *LinkedPracticeDB*, trong phần **Provider Name** chọn *Microsoft Jet 4.0 OLEDB Provider*. Trong phần **Data Source** nhập vào vị trí của Access database (C:\PracticeDB.mdb) và click OK.

Ta sẽ có Linked Server tên LinkedPracticeDB xuất hiện dưới phần Security/Linked Server. Giả sử ta muốn select data từ Linked Server này ta có thể dùng Query Analyser như sau:

```
Select * from LinkedPracticeDB...Customers
```

Trong ví dụ trên ta dùng tên của Linked Server và theo sau là ba chấm (vì để truy cập vào database ta phải dùng four-part name nhưng trong trường hợp này ta dùng default value nên không cần cho biết tên của Catalog và Schema nhưng phải dùng dấu chấm để phân biệt từng phần).

Ngoài cách trên ta có thể dùng pass-through query với **OPENQUERY** function như sau:

```
Select * from OPENQUERY(LinkedPracticeDB,'Select * from
Customers')
```

Trong ví dụ trên ta thấy function OPENQUERY sẽ trả về một data set và có thể nằm sau keyword FROM như một table. Khi dùng OPENQUERY function ta cần cho biết tên của Linked Server và query mà ta muốn thực hiện.

Lưu ý: function trong SQL Server được dùng tương tự như là stored procedure.

Ad Hoc Computer Name

Ngoài cách dùng Linked Server như đã trình bày ở trên ta có thể dùng ad hoc computer name (ad hoc nghĩa là lâm thời, tạm thời). Nghĩa là đối với những database system mà ta thường xuyên query thì dùng Linked Server còn đối với những query lâu lâu mới dùng đến thì ta có thể select data bằng **OPENROWSET** hay **OPENDATASOURCE** functions

Ví dụ: ta cũng sẽ select data từ Access database như trên dùng **OPENROWSET**

```
        Select * from
OPENROWSET('Microsoft.jet.oledb.4.0','C:\PracticeDB.mdb'; 'admin'; '',
Customers)
```

Trong ví dụ trên khi dùng OPENROWSET ta cần phải đưa vào tất cả những thông tin cần thiết để connect vào database như tên của Provider, vị trí của file, username, password (trường hợp này không có password) và tên của table mà ta muốn select. Mỗi lần ta thực thi câu lệnh trên SQL Server đều kiểm tra security trong khi đó nếu dùng Linked Server thì chỉ kiểm ta một lần mà thôi. OPENROWSET tương tự như OPENQUERY ở chỗ nó trả về một rowset và có thể đặt vào vị trí của một table trong câu lệnh query.

Ngoài cách dùng trên ta cũng có thể dùng OPENDATASOURCE để query như sau:

```
        Select * from OPENDATASOURCE('Microsoft.jet.oledb.4.0',
                                'Data Source = C:\PracticeDB.mdb; User ID
= Admin; Password = ')
                                ...Customers
```

Trong ví dụ trên ta thấy OPENDATASOURCE trả về một phần của four-part name (nghĩa là tương đương với tên của Linked Server) cho nên ta phải dùng thêm ba dấu chấm.

Cursors

Nếu giải thích một cách ngắn gọn thì cursor tương tự như recordset hay dataset trong programming. Nghĩa là ta select một số data vào memory sau đó có thể lần lượt làm việc với từng record bằng cách Move Next...

Có 3 loại cursors là Transact- SQL Cursors, API Cursors và Client Cursors. Trong đó Transact-SQL và API thuộc loại Server Cursors nghĩa là cursors được load lên và làm việc bên phía server. Trong khuôn khổ bài học này ta chỉ nghiên cứu Transact-SQL cursors.

Transact-SQL cursors được tạo ra trên server bằng các câu lệnh Transact-SQL và chủ yếu được dùng trong stored procedures và triggers. Trước hết hãy xem qua một ví dụ về cursor:

```
        DECLARE @au_lname varchar(40), @au_fname varchar(20)

        DECLARE Employee_Cursor CURSOR FOR
        SELECT LastName, FirstName FROM Northwind.dbo.Employees

        OPEN Employee_Cursor

        FETCH NEXT FROM Employee_Cursor INTO @au_lname, @au_fname
        WHILE @@FETCH_STATUS = 0
        BEGIN
                PRINT 'Author:' + @au_fname + ' ' + @au_lname
                FETCH NEXT FROM Employee_Cursor INTO @au_lname,
@au_fname
        END

        CLOSE Employee_Cursor

        DEALLOCATE Employee_Cursor
```

Trong ví dụ ở trên ta sẽ select LastName và FirstName từ Employees table của Northwind database và load vào Employee_Cursor sau đó lần lượt in tên của các employee ra màn hình. Để làm việc với một cursor ta cần theo các bước sau:

1. Dùng câu lệnh DECLARE CURSOR để khai báo một cursor. Khi khai báo ta cũng phải cho biết câu lệnh SELECTsẽ được thực hiện để lấy data.
2. Dùng câu lệnh OPEN để đưa data lên memory (populate data). Đây chính là lúc thực hiện câu lệnh SELECT vốn được khai báo ở trên.
3. Dùng câu lệnh FETCH để lấy từng hàng data từ record set. Cụ thể là ta phải gọi câu lệnh FETCH nhiều lần. FETCH tương tự như lệnh Move trong ADO recordset ở chỗ nó có thể di chuyển tới lui bằng câu lệnh FETCH FIRST, FETCH NEXT, FETCH PRIOR, FETCH LAST, FETCH ABSOLUTE n, FETCH RELATIVE n nhưng khác ở chỗ là nó lấy data bỏ vào trong variable (FETCH...FROM...INTO variable_name). Thông thường ta FETCH data trước sau đó loop cho tới record cuối của Cursor bằng vòng lặp WHILE bằng cách kiểm tra global variable @@FETCH_STATUS (=0 nghĩa là thành công).

4. Khi ta viếng thăm từng record ta có thể UPDATE hay DELETE tùy theo nhu cầu (trong thí dụ này chỉ dùng lệnh PRINT)
5. Dùng câu lệnh CLOSE để đóng cursor. Một số tài nguyên (memory resource) sẽ được giải phóng nhưng cursor vẫn còn được khai báo và có thể OPEN trở lại.
6. Dùng câu lệnh DEALLOCATE để phóng thích hoàn toàn các tài nguyên dành cho cursor (kể cả tên của cursor).

Lưu ý là trong ví dụ ở trên trước khi dùng Cursor ta cũng declare một số variable (@au_fname và @au_lname) để chứa các giá trị lấy được từ cursor. Bạn có thể dùng Query Analyzer để chạy thử ví dụ trên.

Stored Procedures

Trong những bài học trước đây khi dùng Query Analyzer chúng ta có thể đặt tên và save các nhóm câu lệnh SQL vào một file dưới dạng script để có thể sử dụng trở lại sau này. Tuy nhiên thay vì save vào text file ta có thể save vào trong SQL Server dưới dạng Stored Procedure. **Stored Procedure là một nhóm câu lệnh Transact-SQL đã được compiled (biên dịch) và chứa trong SQL Server dưới một tên nào đó và được xử lý như một đơn vị** (chứ không phải nhiều câu SQL riêng lẻ).

Ưu Điểm Của Stored Procedure

Stored Procedure có một số ưu điểm chính như sau:

- **Performance** : Khi thực thi một câu lệnh SQL thì SQL Server phải kiểm tra permission xem user gởi câu lệnh đó có được phép thực hiện câu lệnh hay không đồng thời kiểm tra cú pháp rồi mới tạo ra một execute plan và thực thi. Nếu có nhiều câu lệnh như vậy gởi qua network có thể làm giảm đi tốc độ làm việc của server. SQL Server sẽ làm việc hiệu quả hơn nếu dùng stored procedure vì người gởi chỉ gởi một câu lệnh đơn và SQL Server chỉ kiểm tra một lần sau đó tạo ra một execute plan và thực thi. Nếu stored procedure được gọi nhiều lần thì execute plan có thể được sử dụng lại nên sẽ làm việc nhanh hơn. Ngoài ra cú pháp của các câu lệnh SQL đã được SQL Sever kiểm tra trước khi save nên nó không cần kiểm lại khi thực thi.
- **Programming Framework** : Một khi stored procedure được tạo ra nó có thể được sử dụng lại. Điều này sẽ làm cho việc bảo trì (maintainability) dễ dàng hơn do việc tách rời giữa business rules (tức là những logic thể hiện bên trong stored procedure) và database. Ví dụ nếu có một sự thay đổi nào đó về mặt logic thì ta chỉ việc thay đổi code bên trong stored procedure mà thôi. Những

ứng dụng dùng stored procedure này có thể sẽ không cần phải thay đổi mà vẫn tương thích với business rule mới. Cũng giống như các ngôn ngữ lập trình khác stored procedure cho phép ta đưa vào các input parameters (tham số) và trả về các output parameters đồng thời nó cũng có khả năng gọi các stored procedure khác.

- **Security** : Giả sử chúng ta muốn giới hạn việc truy xuất dữ liệu trực tiếp của một user nào đó vào một số tables, ta có thể viết một stored procedure để truy xuất dữ liệu và chỉ cho phép user đó được sử dụng stored procedure đã viết sẵn mà thôi chứ không thể "đụng" đến các tables đó một cách trực tiếp. Ngoài ra stored procedure có thể được encrypt (mã hóa) để tăng cường tính bảo mật.

Các Loại Stored Procedure

Stored procedure có thể được chia thành 5 nhóm như sau:

1. **System Stored Prcedure** : Là những stored procedure chứa trong Master database và thường bắt đầu bằng tiếp đầu ngữ **sp_** . Các stored procedure này thuộc loại built-in và chủ yếu dùng trong việc quản lý database (administration) và security. Ví dụ bạn có thể kiểm tra tất cả các processes đang được sử dụng bởi user DomainName\Administrators bạn có thể dùng `sp_who @loginame='DomainName\Administrators'` . Có hàng trăm system stored procedure trong SQL Server. Bạn có thể xem chi tiết trong SQL Server Books Online.
2. **Local Stored Procedure** : Đây là loại thường dùng nhất. Chúng được chứa trong user database và thường được viết để thực hiện một công việc nào đó. Thông thường người ta nói đến stored procedure là nói đến loại này. Local stored procedure thường được viết bởi DBA hoặc programmer. Chúng ta sẽ bàn về cách tạo stored prcedure loại này trong phần kế tiếp.
3. **Temporary Stored Procedure** : Là những stored procedure tương tự như local stored procedure nhưng chỉ tồn tại cho đến khi connection đã tạo ra chúng bị đóng lại hoặc SQL Server shutdown. Các stored procedure này được tạo ra trên TempDB của SQL Server nên chúng sẽ bị delete khi connection tạo ra chúng bị cắt đứt hay khi SQL Server down. **Temporary stored procedure được chia làm 3 loại** : **local** (bắt đầu bằng #), **global** (bắt đầu bằng ##) và stored procedure được **tạo ra trực tiếp trên TempDB**. Loại local chỉ được sử dụng bởi connection đã tạo ra chúng và bị xóa khi disconnect, còn loại global có thể được sử dụng bởi bất kỳ connection nào. Permission cho loại global là dành cho mọi người

(public) và không thể thay đổi. Loại stored procedure được tạo trực tiếp trên TempDB khác với 2 loại trên ở chỗ ta **có thể set permission**, chúng **tồn tại kể cả sau khi connection tạo ra chúng bị cắt đứt** và **chỉ biến mất khi SQL Server shut down**.

4. **Extended Stored Procedure** : Đây là một loại stored procedure sử dụng một chương trình ngoại vi (external program) vốn được compiled thành một DLL để mở rộng chức năng hoạt động của SQL Server. Loại này thường bắt đầu bằng tiếp đầu ngữ **xp_** .Ví dụ, `xp_sendmail` dùng đề gởi mail cho một người nào đó hay `xp_cmdshell` dùng để chạy một DOS command... Ví dụ `xp_cmdshell 'dir c:\'` . Nhiều loại extend stored procedure được xem như system stored procedure và ngược lại.
5. **Remote Stored Procedure** : Những stored procedure gọi stored procedure ở server khác.

Viết Stored Procedure

Tên và những thông tin về Stored Procedure khi được tạo ra sẽ chứa trong SysObjects table còn phần text của nó chứa trong SysComments table. Vì Stored Procedure cũng được xem như một object nên ta cũng có thể dùng các lệnh như CREATE, ALTER, DROP để tạo mới, thay đổi hay xóa bỏ một stored procedure. Chúng ta hãy xem một ví dụ sau về Stored Procedure: Để tạo một stored procedure bạn có thể dùng Enterprise Manager **click lên trên Stored Procedure -> New Stored Procedure...**. Trong ví dụ này ta sẽ tạo ra một stored procedure để insert một new order vào Orders table trong Practice DB. Để insert một order vào database ta cần đưa vào một số input như OrderID, ProductName (order món hàng nào) và CustomerName (ai order). Sau đó ta trả về kết quả cho biết việc insert đó có thành công hay không. Result = 0 là insert thành công.

```
        CREATE PROCEDURE AddNewOrder
        @OrderID smallint,
        @ProductName varchar(50),
        @CustomerName varchar(50),
        @Result smallint=1 Output
        AS

        DECLARE @CustomerID smallint
        BEGIN TRANSACTION
        If not Exists(SELECT CustomerID FROM Customers WHERE
[Name]=@CustomerName)
        --This is a new customer. Insert this customer to the database
                BEGIN
                        SET @CustomerID= (SELECT Max(CustomerID) FROM
Customers)
                        SET @CustomerID=@CustomerID+1
```

```
                        INSERT INTO Customers
VALUES(@CustomerID,@CustomerName)
                        If Exists(SELECT OrderID FROM [Orders] WHERE
OrderID=@OrderID)
                        --This order exists and could not be added any
more so Roll back
                                BEGIN
                                        SELECT @Result=1
                                        ROLLBACK TRANSACTION
                                END
                        Else
                        --This is a new order insert it now
                                BEGIN
                                        INSERT INTO
[Orders](OrderID,ProductName,CustomerID)
VALUES(@OrderID,@ProductName,@CustomerID)
                                        SELECT @Result=0
                                        COMMIT TRANSACTION
                                END
                END
        Else
        --The customer exists in DB go ahead and insert the order
                BEGIN
                        If Exists(SELECT OrderID FROM [Orders] WHERE
OrderID=@OrderID)
                        --This order exists and could not be added any
more so Roll back
                                BEGIN
                                        SELECT @Result=1
                                        ROLLBACK TRANSACTION
                                END
                        Else
                        --This is a new order insert it now
                                BEGIN
                                        INSERT INTO
[Orders](OrderID,ProductName,CustomerID)
VALUES(@OrderID,@ProductName,@CustomerID)
                                        SELECT @Result=0
                                        COMMIT TRANSACTION
                                END
                END
        Print @Result
        Return
```

Để tạo ra một stored procedure ta dùng lệnh CREATE PROCEDURE theo sau là tên của nó (nếu là temporary stored procedure thì thêm dấu # trước tên của procedure. Nếu muốn encrypt thì dùng WITH ENCRYPTION trước chữ AS) và các input hoặc ouput parameters. Nếu là output thì thêm keyword OUTPUT đằng sau parameter. Ta có thể cho giá trị default cùng lúc với khai báo data type của parameter. Kể từ sau chữ AS là phần body của stored procedure.

Trong ví dụ ở trên trước hết ta khai báo một biến @CustomerID sau đó bắt đầu một transaction bằng BEGIN TRANSACTION (toàn bộ công việc insert này được thực hiện trong một Transaction nghĩa là hoặc là insert thành công hoặc là

không làm gì cả- all or nothing). Trước hết ta kiểm tra xem người khách hàng là người mới hay cũ. Nếu là người mới thì ta "tiện tay" insert vào Customers table luôn còn nếu không thì chỉ insert vào Orders table mà thôi. Nếu là người customer mới ta lấy CustomerID lớn nhất từ Customers table bỏ vào biến @CustomerID và sau đó tăng lên một đơn vị dùng cho việc Insert Customer ở dòng kế tiếp.

Sau khi insert người customer mới ta tiếp tục công việc bằng việc kiểm tra xem Order muốn insert có tồn tại chưa (vì nếu order đã tồn tại thì khi insert SQL Server sẽ báo lỗi do OrderID là Primary key). Nếu như order trên vì lý do nào đó đã có trong DB thì ta roll back và trả kết quả =1 còn nếu không thì ta insert một order mới vào và commit transaction với kết quả trả về =0.

Tương tự như vậy nếu người customer đã tồn tại (sau chữ else đầu tiên) thì ta chỉ việc insert order giống như trên. Trong mọi trường hợp kể trên ta đều in ra kết quả và return.

Ví dụ trên đây chỉ mang tính học hỏi còn trên thực tế database có thể phức tạp hơn nhiều nên việc viết stored procedure đòi hỏi kiến thức vững chắc về SQL và kỹ năng về programming.

Muốn hiểu rõ hơn về bài học này bạn cần làm bài tập số 2.

Tóm lại trong bài này chúng ta đã tìm hiểu một số kỹ thuật import và export data . Đồng thời biết qua các cách select data từ các database system khác dùng distributed query. Nhưng quan trọng nhất và thường dùng nhất là các stored procedures. Bạn cần hiểu rõ vai trò của stored procedure và biết cách tạo ra chúng.

Vì kiến thức về database nói chung và SQL Server nói riêng khá rộng nên trong khuôn khổ một bài học chúng tôi không thể trình bày cặn kẽ từng chi tiết và đôi khi có hơi dồn ép cho nên bạn cần đọc đi đọc lại nhiều lần để nắm được ý chính và phải xem thêm sách (nếu không có sách thì phải xem thêm SQL Books Online). Sau bài học này các bạn cần làm bài tập số 2 để hệ thống hóa lại kiến thức đã học. Khi làm bài tập nên làm theo thứ tự và tuân thủ theo các yêu cầu của bài tập đặt ra. Không nên bỏ qua bước nào.

Triggers And Views

Trong bài này chúng ta sẽ tìm hiểu ứng dụng của một loại stored procedure đặc biệt gọi là Triggers và dùng Views để thể hiện data trong một hay nhiều table như thế nào.

Triggers

Trigger là một loại stored procedure đặc biệt được execute (thực thi) một cách tự động khi có một data modification event xảy ra như Update, Insert hay Delete. Trigger được dùng để đảm bảo Data Integrity hay thực hiện các business rules nào đó.

Khi nào ta cần sử dụng Trigger:

- Ta chỉ sử dụng trigger khi mà các biện pháp bảo đảm data intergrity khác như Constraints không thể thỏa mãn yêu cầu của ứng dụng. Nên nhớ Constraint thuộc loại **Declarative Data Integrity** cho nên sẽ **kiểm tra data trước khi cho phép nhập vào table** trong khi Trigger thuộc loại Procedural Data Integrity nên việc insert, update, delete đã xảy ra rồi mới kích hoạt trigger. Chính vì vậy mà ta cần cân nhắc trước khi quyết định dùng loại nào trong việc đảm bảo Data Integrity.

- Khi một database được denormalized (ngược lại quá trình normalization, là một quá trình thiết kế database schema sao cho database chứa data không thừa không thiếu) sẽ có một số data thừa (redundant) được chứa trong nhiều tables. Nghĩa là sẽ có một số **data được chứa cùng một lúc ở hai hay nhiều nơi khác nhau**. Khi đó để đảm bảo tính chính xác thì khi data được update ở một table này thì cũng phải được update một cách tự động ở các table còn lại bằng cách dùng Trigger.

  Ví dụ: ta có table Item trong đó có field Barcode dùng để xác định một mặt hàng nào đó. Item table có vai trò như một cuốn catalog chứa những thông tin cần thiết mô tả từng mặt hàng. Ta có một table khác là Stock dùng để phản ánh món hàng có thực trong kho như được nhập về này nào được cung cấp bởi đại lý nào, số lượng

bao nhiêu (tức là những thông tin về món hàng mà không thể chứa trong Item table được)...table này cũng có field Barcode để xác định món hàng trong kho. Như vậy thông tin về Barcode được chứa ở hai nơi khác nhau do đó ta cần dùng trigger để đảm bảo là Barcode ở hai nơi luôn được synchonize (đồng bộ).

- Đôi khi ta có nhu cầu thay đổi dây chuyền (cascade) ta có thể dùng Trigger để bảo đảm chuyện đó. Nghĩa là khi có sự thay đổi nào đó ở table này thì một số table khác cũng được thay đổi theo để đảm bảo tính chính xác. Ví dụ như khi một món hàng được bán đi thì số lượng hàng trong table Item giảm đi một món đồng thời tổng số hàng trong kho (Stock table) cũng phải giảm theo một cách tự động. Như vậy ta có thể tạo một trigger trên Item table để mỗi khi một món được bán đi thì trigger sẽ được kích hoạt và giảm tổng số hàng trong Stock table.

Đặc điểm của Trigger:

- Một trigger có thể làm nhiều công việc (actions) khác nhau và có thể được kích hoạt bởi nhiều hơn một event. Ví dụ ta có thể viết một trigger được kích hoạt bởi bất kỳ event nào như Update, Insert hay Delete và bên trong trigger ta sẽ viết code để giải quyết cho từng trường hợp.
-
- Trigger không thể được tạo ra trên temporary hay system table.

- Trigger chỉ có thể được kích hoạt một cách tự động bởi một trong các event Insert, Update, Delete mà không thể chạy manually được.

- Có thể áp dụng trigger cho View.

- Khi một trigger được kích hoạt thì data mới vừa được insert hay mới vừa được thay đổi sẽ được chứa trong **Inserted** table còn data mới vừa được delete được chứa trong **Deleted** table. Đây là 2 table tạm chỉ chứa trên memory và chỉ có giá trị bên trong trigger mà thôi (nghĩa là chỉ nhìn thấy và được query trong trigger mà thôi). Ta có thể dùng thông tin trong 2 table này để so sánh data cũ và mới hoặc kiểm tra xem data mới vừa thay đổi có hợp lệ trước khi commit hay roll back. (Xem thêm ví dụ bên dưới)

- Có 2 loại triggers (class) : INSTEAD OF và AFTER. Loại INSTEAD OF sẽ bỏ qua (bybass) action đã kích hoạt trigger mà thay vào đó sẽ thực hiện các dòng lệnh SQL bên trong Trigger. Ví dụ ta có một

Update trigger trên một table với câu INSTEAD OF thì khi table được update thay vì update SQL Server sẽ thực hiện các lệnh đã được viết sẵn bên trong trigger. Ngược lại loại AFTER (loại default tương đương với keyword FOR) sẽ thực hiện các câu lệnh bên trong trigger sau khi các action tạo nên trigger đã xảy ra rồi.

Tạo Một Trigger Như Thế Nào?

Cú pháp căn bản để tạo ra một trigger có dạng như sau:

CREATE TRIGGER trigger_name

ON table_name or view_name

FOR trigger_class and trigger_type(s)

AS Transact-SQL statements

Như vậy khi tạo ra một trigger ta phải chỉ rõ là tạo ra trigger trên table nào và được trigger khi nào (insert, update hay delete. Sau chữ AS là các câu lệnh SQL xử lý công việc.

Ta hãy nghiên cứu một ứng dụng thực tiễn sau. Giả sử ta viết một application cho phép user có thể Insert, Update và Delete những thông tin nằm trong database. User này thường là những người không thông thạo lắm về computer mà chúng tôi thường gọi đùa là "bà tám". Vào một ngày đẹp trời, "bà tám" mặt mày tái xanh đến cầu cứu ta vì đã lỡ tay "delete" những thông tin khá quan trọng và hy vọng ta có thể phục hồi dữ liệu dùm. Nếu chúng ta không phòng xa trước khi viết application thì coi như cũng vô phương cứu chữa vì data đã hoàn toàn bị delete.

Nhưng nếu bạn là một "guru" bạn sẽ gật gù "chuyện này khó lắm!" nhưng sau đó bạn chỉ tốn vài phút đồng hồ để rollback. Muốn làm được chuyện này chúng ta phải dùng một "chiêu" gọi là Audit (kiểm tra hay giám sát). Tức là ngoài các table chính ta sẽ thêm các table phụ gọi là Audit tables. Bất kỳ hoạt động nào đụng chạm vào một số table quan trọng trong database ta đều ghi nhận vào trong Audit table. Ví dụ khi user update hay delete một record trong table nào đó thì trước khi update hay delete ta sẽ âm thầm di chuyển record đó sang Audit table rồi mới update hay delete table chính. Như vậy nếu có chuyện gì xảy ra ta có thể dễ dàng rollback (trả record về chỗ cũ).

Ví dụ:

Ta có table Orders trong PracticeDB. Để audit các hoạt động diễn ra trên table này ta tạo ra một audit table với tên **Aud_Orders với các column giống y hệt với Orders table**. Ngoài ra ta thêm vào 2 columns :

- **Audit_Type** : với các giá trị có thể là 'I','U','D' để ghi nhận record được Insert, Update hay Delete
- **Date_Time_Stamp** : Data Type thuộc loại DateTime dùng để ghi nhận thời điểm xảy ra sự thay đổi, có vai trò như một con dấu.

(Nếu trong môi trường nhiều user thì ta thêm một column **UserID** để ghi nhận user nào thay đổi).

Sau đó ta sẽ tạo ra 3 trigger dùng cho việc audit như sau:

```
--Insert Trigger
CREATE TRIGGER [AuditInsertOrders]
ON [dbo].[Orders]
FOR Insert
AS
insert into aud_orders select *,'I',getdate() From inserted
--Update Trigger
CREATE TRIGGER [AuditUpdateOrders]
ON [dbo].[Orders]
for UPDATE
AS
insert into aud_orders select *,'U',Getdate() from deleted

--Delete Trigger
CREATE TRIGGER [AuditDeleteOrders]
ON [dbo].[Orders]
FOR DELETE
AS
insert into aud_orders select *,'D',getdate() From deleted
```

Trong ví dụ trên khi user insert một record thì record mới vừa được insert sẽ nằm trong **inserted** table như đã trình bày ở phần trên. Do đó ta sẽ select tất cả các column trong inserted table cộng thêm Audit Type "I" và dùng hàm GetDate() trong SQL Server để lấy system date time dùng cho Date_Time_Stamp column, sau đó insert vào Aud_Orders table. Tương tự với trường hợp Update và Delete, record đã được update hay delete nằm trong **deleted** table.

Như vậy trở lại trường hợp thí dụ ở trên nếu "bà tám" yêu cầu ta có thể vào tìm kiếm trong audit table để phục hồi lại record. Ngoài ra ta có thể dùng table này để tìm ra thủ phạm đã xoá hay sửa chữa data khi cần thiết.

Để tạo ra hay xem một trigger bằng **Enterprise Manager** bạn làm như sau: **Right-Click lên table mà bạn muốn tạo trigger->All Tasks-> Manage Triggers.**

Lưu ý: Đôi Khi ta chỉ muốn trigger thực sự hoạt động khi một hay vài column nào đó được Update chứ không phải bất kỳ column nào. Khi đó ta có thể dùng hàm **Update**(*Column_Name*) để kiểm tra xem column nào đó có bị update hay không.

Ví dụ:

Tạo một trigger cho Customer table. Bên trong Trigger (sau chữ AS) ta có thể kiểm tra xem nếu column First_Name hay Last_Name bị thay đổi thì mới hành động nếu không thì không làm gì cả

```
        IF UPDATE (first_name) OR UPDATE (Last_Name)
        BEGIN
                Do some conditional processing when either of these
columns are updated.
        END
```

Nếu muốn kiểm tra nhiều columns ta có thể dùng hàm khác là **Columns_Updated()** . Xin xem thêm trong SQL Server Books Online để biết thêm chi tiết về cách sử dụng.

Views

Định nghĩa một cách đơn giản thì view trong SQL Server tương tự như Query trong Access database. View có thể được xem như một table ảo mà data của nó được select từ một stored query. Đối với programmer thì view không khác chi so với table và có thể đặt ở vị trí của table trong các câu lệnh SQL. Đặc điểm của View là ta có thể join data từ nhiều table và trả về một recordset đơn. Ngoài ra ta có thể "xào nấu" data (manipulate data) trước khi trả về cho user bằng cách dùng một số logic checking như (if, case...).

Ví dụ:

```
        Create View OrderReport
        As
        Select OrderID,
                (case when [Name] is null then 'New Customer'
                else [Name]
                end )As CustomerName,
                ProductName,
                DateProcessed
        From Customers Right Outer Join Orders on
Customers.CustomerID=Orders.CustomerID
```

Trong ví dụ trên ta chủ yếu trả về data từ Orders table trong PracticeDB nhưng thay vì display CustomerID vốn không có ý nhiều ý nghĩa đối với user ta sẽ display tên của customer bằng cách join với Customer table. Nếu Customer Name là Null nghĩa là tên của customer đã đặt order không tồn tại trong system. Thay vì để Null ta sẽ display "New Customer" để dễ nhìn hơn cho user.

Nói chung câu lệnh SQL trong View có thể từ rất đơn giản như select toàn bộ data từ một table cho đến rất phức tạp với nhiều tính năng programming của T-SQL.

View Thường Được Dùng Vào Việc Gì?

View thường được sử dùng vào một số công việc sau:

- **Tập trung vào một số data nhất định** : ta thường dùng view để select một số data mà user quan tâm hay chịu trách nhiệm và loại bỏ những data không cần thiết.

Ví dụ: Giả sử trong table ta có column "Deleted" với giá trị là True hay False để đánh dấu một record bị delete hay không. Việc này đôi khi được dùng cho việc Audit. Nghĩa là trong một ứng dụng nào đó khi user delete một record nào đó, thay vì ta physically delete record ta chỉ logically delete bằng cách đánh dấu record là đã được "Deleted" để đề phòng user yêu cầu roll back. Như vậy chủ yếu ta chỉ quan tâm đến data chưa delete còn data đã được đánh dấu deleted chỉ được để ý khi nào cần roll back hay audit mà thôi. Trong trường hợp này ta có thể tạo ra một view select data mà Deleted=False và làm việc chủ yếu trên view thay vì toàn bộ table.

- **Đơn giản hóa việc xử lý data**: Đôi khi ta có những query rất phức tạp và sử dụng thường xuyên ta có thể chuyển nó thành View và đối xử nó như một table, như vậy sẽ làm cho việc xử lý data dễ dàng hơn.
-
- **Customize data**: Ta có thể dùng view để làm cho users thấy data từ những góc độ khác nhau mặc dù họ đang dùng một nguồn data giống nhau. Ví dụ: Ta có thể tạo ra views trong đó những thông tin về customer được thể hiện khác nhau tùy login ID là normal user hay manager.
-
- **Export và Import data**: Đôi khi ta muốn export data từ SQL Server sang các ứng dụng khác như Excel chẳng hạn ta có thể dùng view để join nhiều table và export dùng bcp.

Khi sử dụng view ta có thể select,insert, update, delete data bình thường như với một table.

Ví dụ:

```
Select * From OrderReport
Where DateProcessed <'2003-01-01'
```

Lưu ý: Trong Enterprise Edition (và Developer Edition) ta có thể tạo Index cho View như cho table. Index sẽ được bàn đến trong các bài sau.

Muốn hiểu rõ hơn về bài học này bạn cần làm bài tập số 3.

Như vậy trong bài này chúng ta đã tìm hiểu Trigger, View trong SQL Server và một số ứng dụng của nó. Nói chung view thường được dùng để trừu tượng hóa (abstract) hay lọc raw data (data thô) trước khi trả về cho user trong khi trigger thường được dùng để bảo đảm tính integrity của database.

Exercise 1: Advanced Query

Please follow those steps to practise:

1. Create a new database called **PracticeDB** (using Enterprise Manager (EP) or Query Analyser (QA))
2. Create 2 tables and insert data as follows (use EP or QA) :

Customers

CustomerID (Int)	Name (nVarChar(50))
1	John Nguyen
2	Bin Laden
3	Bill Clinton
4	Thomas Hardy
5	Ana Tran
6	Bob Carr

Orders

OrderID (Int)	CustomerID (Int)	ProductName (nvarchar(50))	DateProcessed (datetime)
1	2	Nuclear Bomb	'2002-12-01'
2	3	Missile	'2000-03-02'
3	2	Jet-1080	'2003-08-03'
4	1	Beers	'2001-05-12'
5	4	Asian Food	'2002-10-04'
6	7	Wine	'2002-03-08'

7	8	Milk	'2002-05-02'

3. Query data using Select statement with the following Join (use QA)
   a. **Inner Join**
   b. **Left Outter Join**
   c. **Right Outer Join**
   d. **Full Outer Join**
   e. **Cross Join**

The result of the query must be displayed in the following format

OrderID	CustomerName	ProductName	DateProcessed

(Hints: Based on the data supplied and your knowledge about various kind of joins guess the results then after each select compare it with your guessing to check if what you understand is correct. You also use As keyword or alias in the queries)

4. Using "**Select * Into...From**" statement to create a new table called "**Processed Orders**" and populate the new table with the data selecting from **Orders** table Where DateProcessed is earlier than '2002-10-05'. (use QA)
5. Using "**Insert Into...Select**" statement to get the top 1 record from "Orders" table and insert into the "ProcessedOrders" (use QA)
6. **Delete** a record from ProcessedOrders where the date processed is '2002-10-04'. (use QA)
7. Using **Union** to merge the two data set from Orders and ProcessedOrders into one data set. (use QA)
8. Apply **Constraints** (use EP or QA)

   a.Apply the **Primary Constraint** to the "ID" column in the tables

   b. Apply the **Foreign Key Constraint** in the Orders table.

   c. Apply the **Check Constraint** to the DateProcessed column so that the date is within '1970-01-01' – '2005-01-01' and try to insert invalid data to see if SQL rejects.

   d. Apply **Unique Constraint** to the CustomerName column of Customers

9**. Back up** database and **Restore** to somewhere else (on a different server or on the same server but with different database name) (use EP or QA)

10. **Truncate** and **Drop** the ProcessedOrders table (use QA)

Exercise 2: Manipulate Data and Stored Procedure

Please follow those steps to practise:

9. Use **bcp** to export all data from Orders table of PracticeDB to c:\Orders.txt (or to c:\Orders.csv)
10. Change some data in the c:\Orders.txt and save. Then import to Orders table from the text file using bcp
11. Import Orders.txt to Orders table using **BULK INSERT**
12. Create a **Linked Server** 'LinkedPracticeDB' which link to an Access database 'PracticeDB.mdb' (firstly you have to create an Access database similar to PracticeDB in SQL Server and input some data). Then do a select data using four-part name and **OPENQUERY**
13. Using **ad hoc computer name** with **OPENROWSET** and **OPENDATASOURCE** functions to select data from 'PracticeDB.mdb'
14. Create the following **Cursor**

```
DECLARE @au_lname varchar(40), @au_fname varchar(20)
DECLARE Employee_Cursor CURSOR FOR

SELECT LastName, FirstName FROM Northwind.dbo.Employees

OPEN Employee_Cursor

FETCH NEXT FROM Employee_Cursor  INTO @au_lname, @au_fname
WHILE @@FETCH_STATUS = 0
BEGIN
        PRINT 'Author:' + @au_fname + ' ' +  @au_lname
        FETCH NEXT FROM Employee_Cursor INTO @au_lname, @au_fname
END
CLOSE Employee_Cursor
DEALLOCATE Employee_Cursor
```

15. Create the following **stored procedure** and try to execute with some values

```
CREATE PROCEDURE AddNewOrder
    @OrderID smallint,
    @ProductName varchar(50),
    @CustomerName varchar(50),
    @Result  smallint=1 Output
AS
DECLARE @CustomerID smallint
    BEGIN TRANSACTION
    If not Exists(SELECT CustomerID FROM Customers WHERE
[Name]=@CustomerName)
            BEGIN
```

```
                    SET @CustomerID= (SELECT Max(CustomerID) FROM
Customers)
                    SET @CustomerID=@CustomerID+1
                    INSERT INTO Customers
VALUES(@CustomerID,@CustomerName)
                    If Exists(SELECT OrderID FROM [Orders] WHERE
OrderID=@OrderID)
                              BEGIN
                                        SELECT @Result=1
                                        ROLLBACK TRANSACTION
                              END
                    Else
                              BEGIN
                                        INSERT INTO
[Orders](OrderID,ProductName,CustomerID)
VALUES(@OrderID,@ProductName,@CustomerID)
                                        SELECT @Result=0
                                        COMMIT TRANSACTION
                              END
          END
     Else
          BEGIN
                    If Exists(SELECT OrderID FROM [Orders] WHERE
OrderID=@OrderID)
                              BEGIN
                                        SELECT @Result=1
                                        ROLLBACK TRANSACTION
                              END
                    Else
                              BEGIN
                                        INSERT INTO
[Orders](OrderID,ProductName,CustomerID)
VALUES(@OrderID,@ProductName,@CustomerID)
                                        SELECT @Result=0
                                        COMMIT TRANSACTION
                              END
          END
Print @Result
Return
```

9. Using **VB** 6 or VB.NET to execute the 'AddNewOrder' stored procedure
10. Using **xp_cmdshell** extended stored procedure to send a message (xp_cmdshell 'net send Hello')

Exercise 3: Triggers And Views

Please follow those steps to practise:

16. Create 3 triggers to audit the changes to the Orders table.

Tips:

- Create an audit table "aud_Orders" with the same colums as in the Orders table and 2 more colums AuditType(with values either 'I','U','D') and DateTimeStamp(which will record the date time stamp of changes)
- Create Update Triggers (Similar to Insert,Delete) : when a record in the Orders table is updated the trigger will move the old record to the audit table(record the date time and mark it as 'U')

17. Create a view that shows all the orders with the following colums:
    OrderID,CustomerName,ProductName,DateProcessed,Status

Business rules:

If CustomerName is a null value "New Customer" is returned

If DateProcessed is later than current date return "Pending", if DateProcessed is ealier return "History" in Status colum.

Tips:

a. Using **Case When ...Then** statement in the view
b. Using Getdate() function to get the current date time