

TỰ HỌC SQL Server²⁰⁰⁰

- Giới thiệu về SQL Server 2000 và ngôn ngữ SQL Server
- Cấu trúc Database trong SQL Server 2000
- Tính toán vện của dữ liệu,
ví dụ về cơ sở dữ liệu quản lý nhân sự...



TƯ HOC SQL SERVER 2000

NGUYỄN THANH QUANG - HOÀNG ANH QUANG

(Sưu tầm, biên soạn)

NHÀ XUẤT BẢN VĂN HÓA - THÔNG TIN

LỜI NÓI ĐẦU

Trong năm 2005 việc Microsoft giới thiệu phiên bản Visual studio 2005 và hệ quản trị cơ sở dữ liệu SQL Server 2005 được đánh giá là một trong mười sự kiện của công nghệ thông tin thế giới. Thật vậy SQL Server được nhắc tới rất nhiều khi người ta nói về hệ quản trị cơ sở dữ liệu. Bởi khả năng truy vấn thông tin một cách nhanh chóng, không giới hạn số lượng bản ghi, dễ dàng làm việc với những loại dữ liệu ký tự UNICODE việc mà nhiều cơ sở dữ liệu trước đây không thực hiện được.

Chúng tôi hân hạnh giới thiệu với các bạn cuốn sách "**Tự học SQL Server 2000**". Quyển sách sẽ lần lượt giúp bạn tìm hiểu về các khả năng của SQL Server 2000, đặc biệt là phần ngôn ngữ SQL. Kiến thức mà chúng tôi gửi đến các bạn rất cơ bản và dễ dàng tiếp thu, nó sẽ là nền tảng giúp các bạn thuận tiện nghiên cứu các phiên bản của SQL sau này cũng như các hệ quản trị dữ liệu khác. Để đọc sách có hiệu quả yêu cầu bạn có một kiến thức nhất định về cơ sở dữ liệu. Chúc các bạn thành công!

NGƯỜI BIÊN SOẠN

BÀI 1

GIỚI THIỆU VỀ SQL SERVER 2000

1. 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 thiết kế 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.

2. Một số kiến thức về các Version của SQL Server

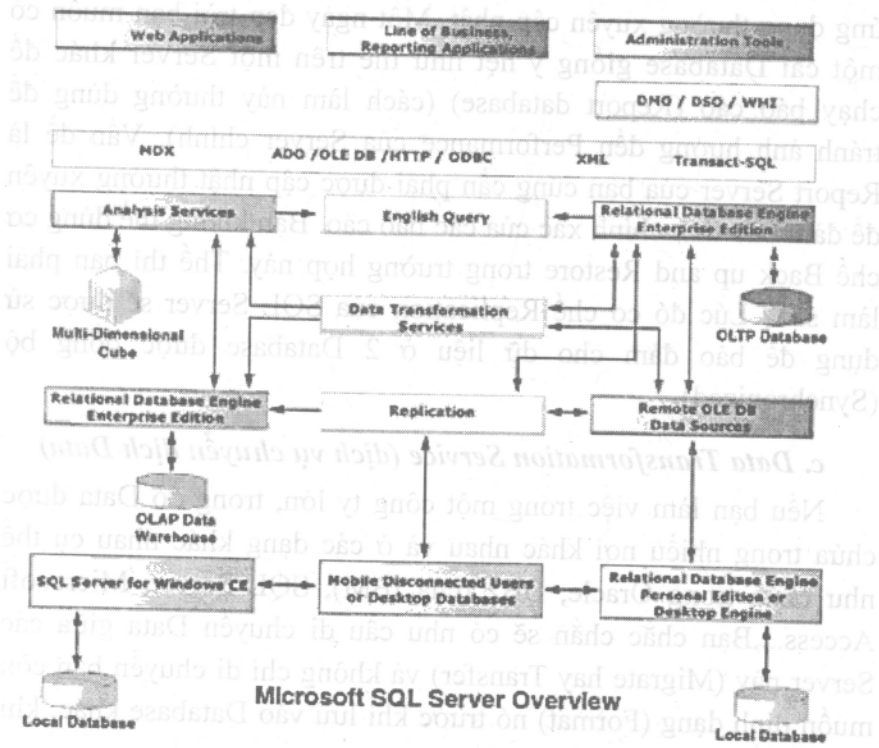
SQL Server của Microsoft được thị trường chấp nhận rộng rãi kể từ Version 6.5. Sau đó Microsoft đã cải tiến và hầu như viết lại một Engine mới cho SQL Server 7.0. Cho nên có thể nói từ Version 6.5 lên Version 7.0 là một bước nhảy vọt. Có một số đặc tính của SQL Server 7.0 không tương thích với Version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

3. 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.



Microsoft SQL Server Overview

a. Relational Database Engine

Được ví như cái lõi của Database, 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.

b. Replication (cơ chế tạo bản sao)

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).

c. Data Transformation Service (dịch vụ chuyển dịch Data)

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.

d. Analysis Service (dịch vụ phân tích dữ liệu)

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).

e. 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).

f. Meta Data Service

Dịch vụ này giúp cho việc chứa đựng và "xào nấu" Meta Data dễ dàng hơn. 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.

g. SQL Server Books Online

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.

h. SQL Server Tools

- 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.

- 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.

- 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.

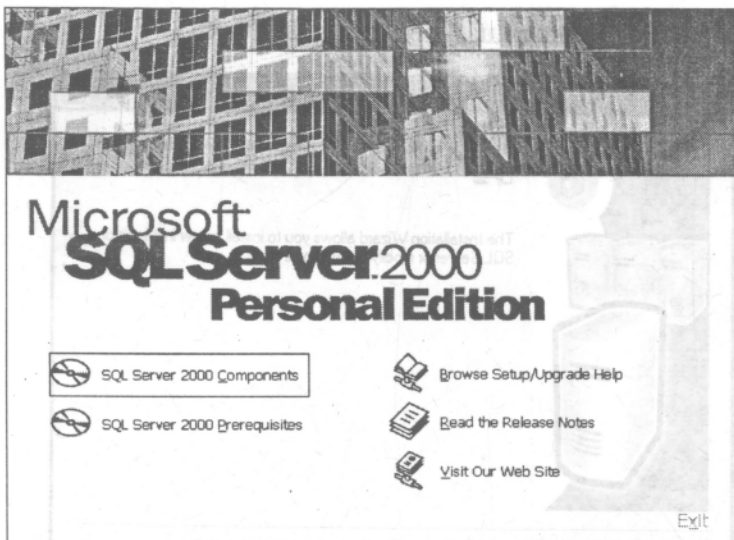
4. Cài đặt SQL Server 2000

a. Yêu cầu cấu hình máy tính

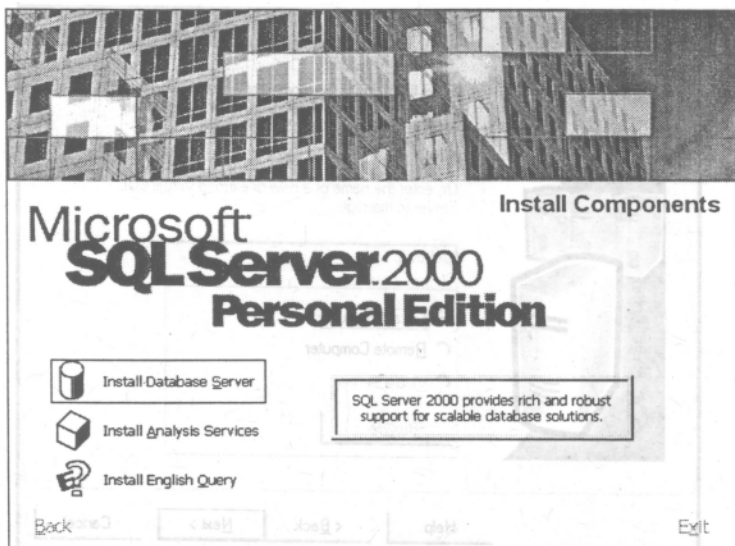
- Một bộ cài đặt SQL Server 2000
- Ít nhất là 64 MB RAM
- 500 MB hard disk để có thể Install SQL Server.
- Bạn có thể Install trên Windows Server hay Windows XP Professional, Windows 2000 Professional hay NT Workstation nhưng không thể Install trên Win 98 family.

b. Các bước tiến hành cài đặt SQL Server 2000

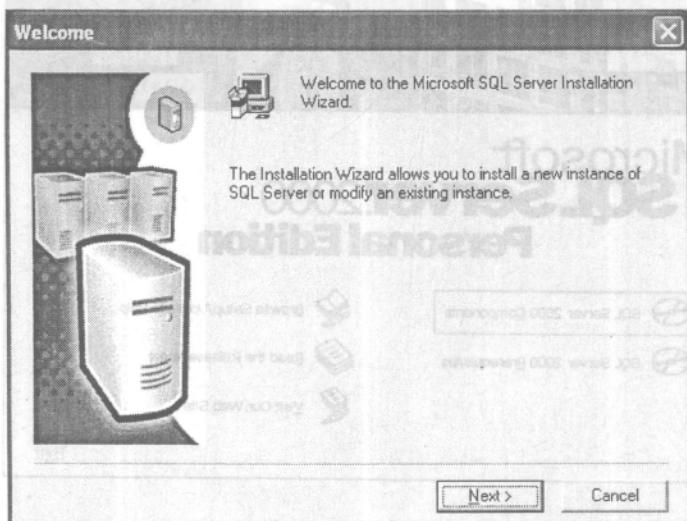
- Bạn chọn SQL Server 2000 Components



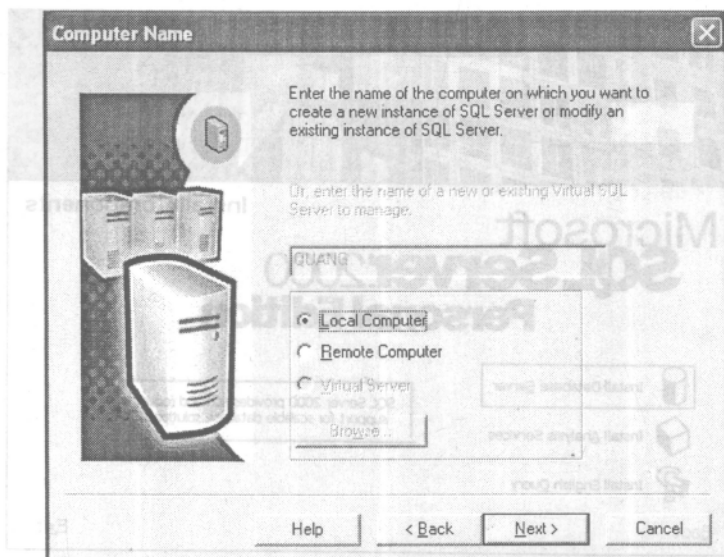
- Bạn chọn mục Install Database Server.



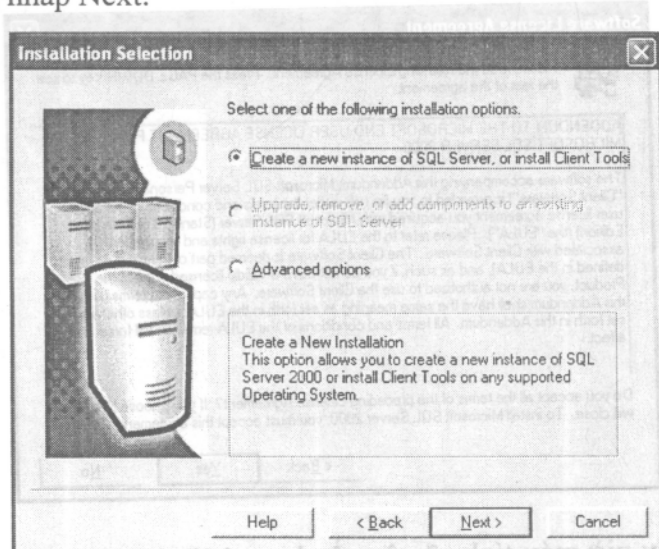
- Cửa sổ Wellcome xuất hiện bạn nhấp Next.



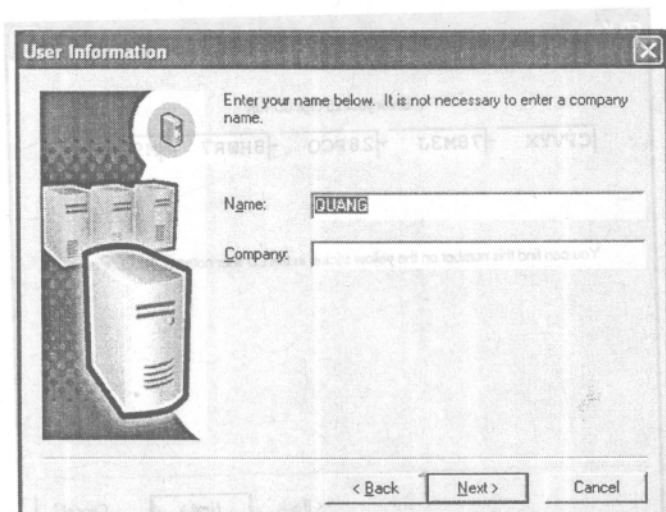
- Ở cửa sổ Computer Name bạn chọn Local Computer xong nhấp Next.



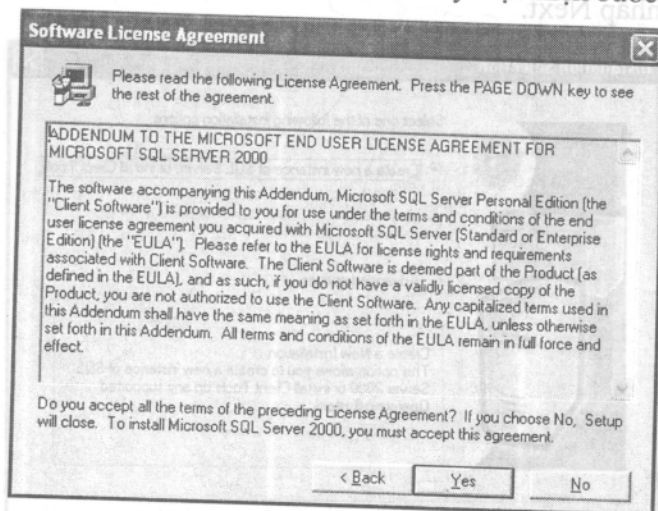
- Ở cửa sổ Installation Select bạn chọn Create a new Instance of .. rồi nhấp Next.



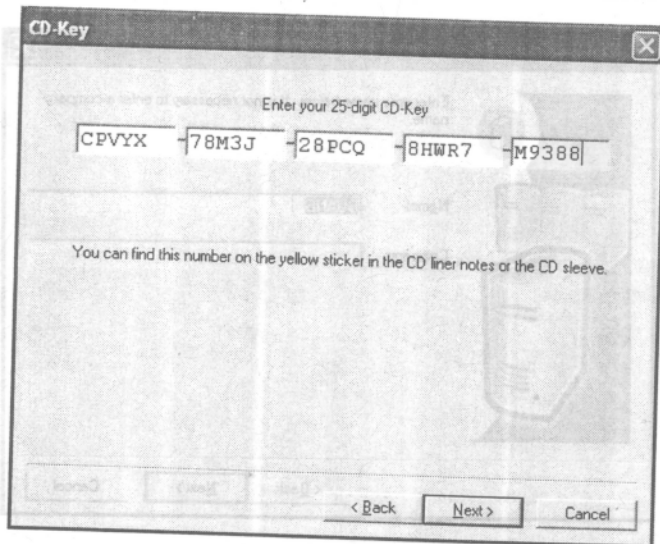
- Ở cửa sổ User information bạn nhập các thông tin về User sau đó nhấp Next.



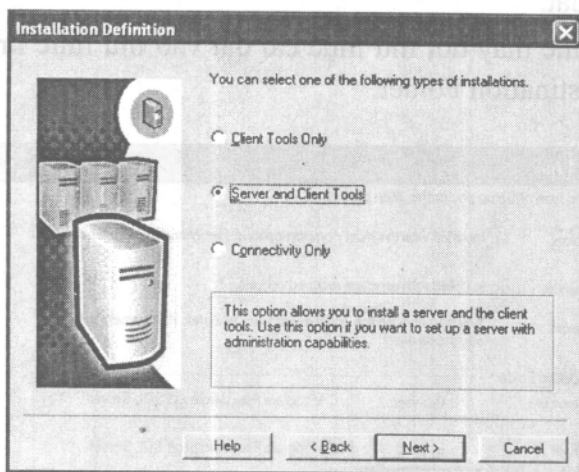
- Cửa sổ xuất hiện bạn chọn Yes để máy thực hiện bước tiếp theo.



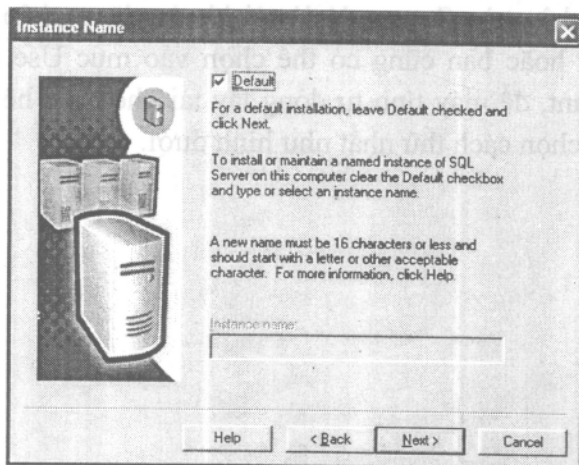
- Bây giờ máy tính sẽ yêu cầu bạn phải nhập mã đăng ký, các bạn có thể tìm mã này trong tệp Cdkey.txt lưu trong thư mục chứa bộ cài SQL Server 2000.



- Bạn nhấp Next, cửa sổ Installation Definition xuất hiện bạn nhấp vào mục Server and Client Tools. Rồi nhấp Next.

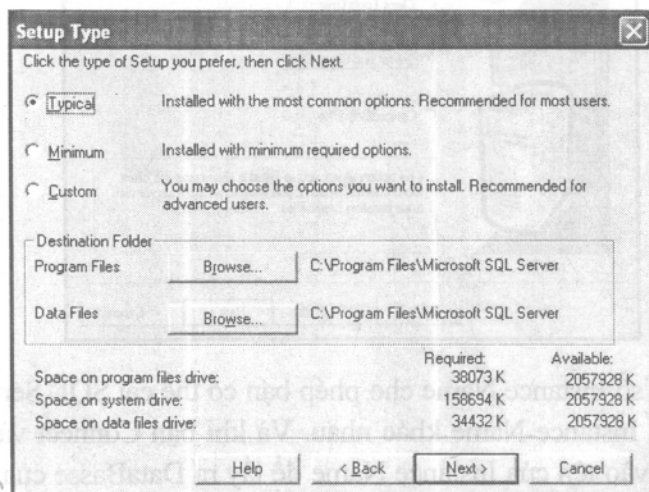


- Cửa sổ Instance Name cho phép bạn có thể cài SQL Server 2000 trên nhiều Instance Name khác nhau. Và khi bạn Connect vào Server sẽ căn cứ vào tên của Instance Name để lấy ra DataBasse của Instance Name. Nhưng ở đây chúng ta chỉ cài một Instance name duy nhất vì vậy các bạn tích vào mục Default. Sau đó bạn nhấp Next.



- Cửa sổ Setup Type cho phép bạn có các tùy chọn nếu bạn tích vào mục Custom, ở đây chúng ta chọn Typical tức là mặc định theo tiến trình cài đặt.

- Bạn có thể thay đổi thư mục cài đặt vào thư mục lưu dữ liệu trong mục Destination Folder.



- Cửa sổ Services Accounts cho phép bạn đăng nhập một tài khoản đăng nhập vào Server đó là tài khoản đăng nhập vào máy tính của bạn, hoặc bạn cũng có thể chọn vào mục Use the Local System account, để máy tính tự động lấy tài khoản từ hệ thống. Ở đây chúng ta chọn cách thứ nhất như hình dưới.

Services Accounts

☒ Use the same account for each service. Auto start SQL Server Service.
☐ Customize the settings for each service.

Services	Service Settings
☐ SQL Server ☐ SQL Server Agent	☐ Use the Local System account ☒ Use a Domain User account Username: Quang Password: xxxxxx Domain: BRAVO ☐ Auto Start Service

- SQL Server 2000 cũng sẽ tự động Add một tài khoản tự động là Sa để đăng nhập vào Server, và trong cửa sổ Authentication Mode sẽ yêu cầu bạn nhập mật khẩu cho tài khoản này, bạn gõ mật khẩu là Sa là xác nhận lại ở mục dưới. Sau đó nhấp Next.

Authentication Mode

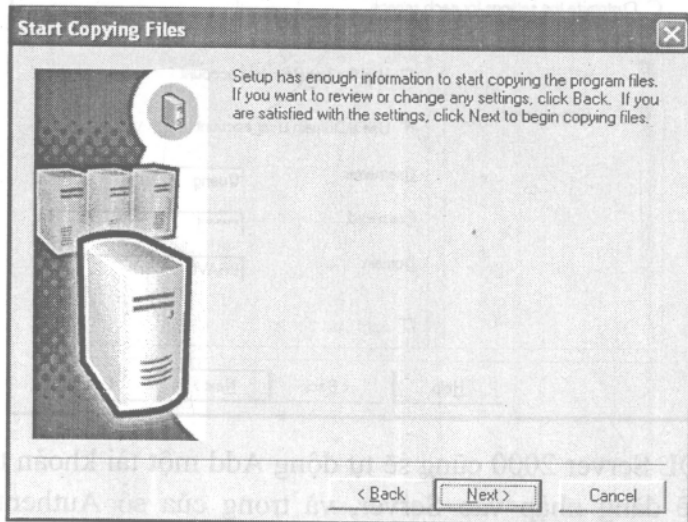
Choose the authentication mode.

☐ Windows Authentication Mode
☒ Mixed Mode (Windows Authentication and SQL Server Authentication)

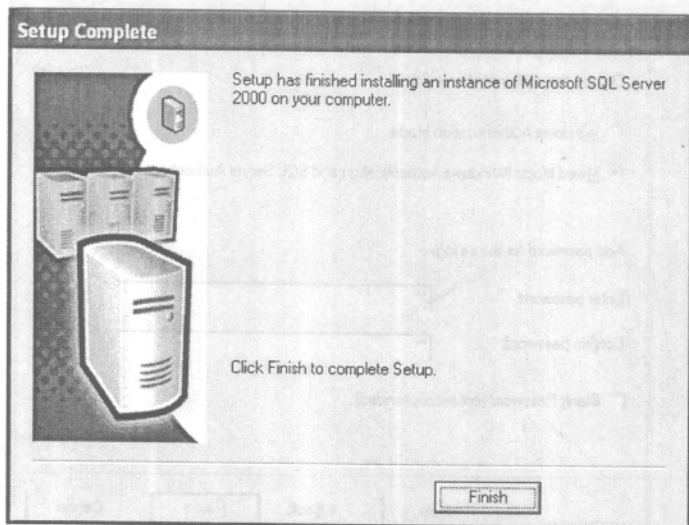
Add password for the sa login:

Enter password:
 Confirm password:
☐ Blank Password (not recommended)

- Đến đây bạn nhấp Next để máy tính tự động Copy những File cần thiết.



- Cuối cùng bạn nhấp Finish để kết thúc cài đặt.



- Sau khi Install bạn sẽ thấy một Icon nằm ở System tray, đây chính là Service Manager. Bạn có thể Start, Stop các SQL Server services dễ dàng bằng cách Double-click vào Icon này.

BÀI 2

NGÔN NGỮ 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.

1. Phân loại

a. 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

b. 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
```

c. 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
```

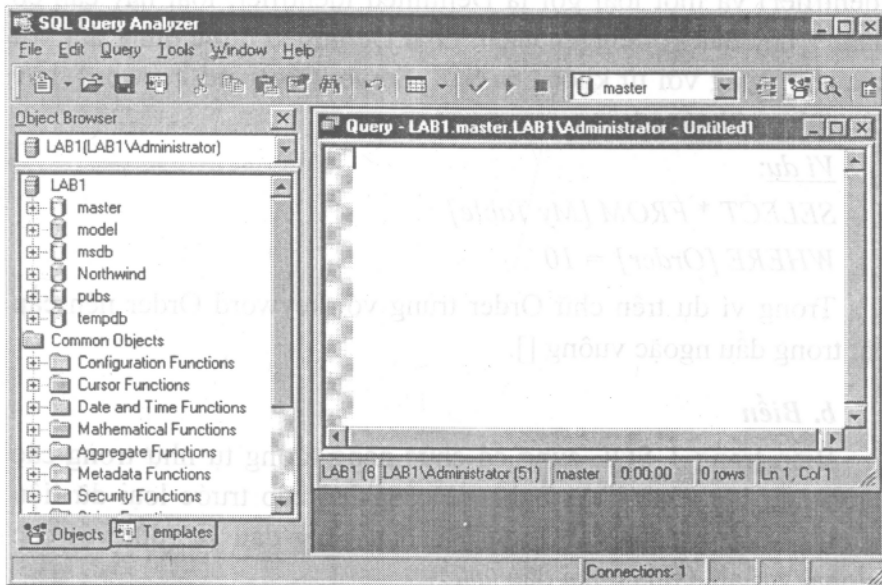
```
USE Northwind
```

```
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.

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



2. Cấu trúc câu lệnh SQL

a. 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 [].

b. 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
```

c. Hàm

Có 2 loại hàm một loại là Built-in và một loại User-defined
+ Built-in

- 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.

+ User-defined

Được tạo ra bởi câu lệnh CREATE FUNCTION và phần Body thường được gói trong cặp lệnh BEGIN...END

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

- Table Functions: Loại này trả về một Table

d. Kiểu dữ liệu

Các loại dữ liệu trong SQL Server sẽ được bàn kỹ trong các bài tiếp theo.

e. Expressions

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

f. 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.

g. Chú Thích

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

3. Thực thi câu lệnh SQL

a. 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.

b. 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.

c. Lệnh GO

Lệnh này chỉ dùng để gọi 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.

4. Kiểu dữ liệu, hàm

a. Kiểu dữ liệu

Kiểu dữ liệu dùng để xác định kiểu thông tin (ký tự, số, ...) và cần bao nhiêu không gian để chứa thông tin trong một cột. Bạn phải chọn kiểu dữ liệu một cách cẩn thận, vì chúng không dễ dàng thay đổi khi bảng được tạo xong và khi đã nhập dữ liệu. Một số kiểu dữ liệu có chiều dài thay đổi, trong khi một số khác có chiều dài cố định. Các kiểu dữ liệu của SQL Server có thể được phân thành các nhóm sau: String (Chuỗi), chuỗi Unicode binary (Nhị phân), Integer (Số nguyên), Approximate Numeric (Số gần đúng), và Exact numeric (Số chính xác), Special, date and Time (Ngày và giờ), Monney (Tiền tệ), Auto-incrementing datatypes (Kiểu dữ liệu tự tăng), Synonym (tương đương), User-defined (kiểu người sử dụng tự định nghĩa), và Computed column (Cột tính). Dưới đây là các kiểu dữ liệu khác nhau trong SQL Server 2000.

- + String (Chuỗi): Bao gồm các kiểu dữ liệu Char (n), Varchar, lưu giữ các chuỗi ký tự

- + Binary (nhị phân): Binary (n), Varbinary, lưu giữ thông tin nhị phân trong khoảng 2 byte.

- + Integer (Số nguyên): Int, Smallint, Tinyint, lưu giữ các giá trị nguyên.

- + Approximate Numeric (Số gần đúng): Float, real, lưu các số gần đúng.

- + Exact Numeric (Số chính xác): Decimal, Numeric, lưu giữ các số chính xác.

- + Special (Kiểu dữ liệu đặc biệt): Bit, Text, Image, lưu trữ Bit, văn bản nhiều hơn 8000 Byte, hay dữ liệu hình ảnh.

- + Date and Time (Ngày và giờ): Datetime, Smalldatetime, lưu trữ ngày và giờ.

- + Money (Tiền tệ): Money, Smallmoney, lưu trữ các giá trị tiền tệ.
- + Auto-incrementing datatypes (Kiểu tự động tăng): Dentyti, timestamp, lưu trữ các giá trị cho phép sẽ tăng tự động hay được gán bởi SQL Server.
- + Synonyms (Tương đương): Ánh xạ kiểu dữ liệu ANSI thành kiểu dữ liệu SQL Server
- + User-defined (Kiểu dữ liệu sử dụng định nghĩa): Bạn có thể tạo kiểu dữ liệu riêng cho bạn để lưu trữ thông tin.

b. Hàm

- Giống như bất kỳ ngôn ngữ máy tính khác, SQL hỗ trợ việc sử dụng các hàm để xử lý dữ liệu. Các hàm là các phép toán vốn thường được thực hiện trên dữ liệu, thường giúp thuận tiện trong quá trình chuyển đổi và xử lý.

- Một ví dụ về một hàm là RTRIM (), chúng ta đã sử dụng trong bài học trước để cắt bất kỳ các khoảng trống từ phía cuối của một chuỗi.

- Trước khi bạn làm việc qua bài học này và thử thực hiện các ví dụ, bạn nên biết về việc sử dụng các hàm SQL có thể rất khó hiểu.

- Không giống như các câu lệnh SQL (ví dụ SELECT), các câu lệnh được hỗ trợ bởi tất cả các hệ thống quản lý cơ sở dữ liệu (DBMS) như nhau, các hàm có khuynh hướng cụ thể cho mỗi DBMS. Thật ra, rất nhiều hàm được hỗ trợ giống nhau bởi tất cả các DBMS chính. Mặc dù tất cả các chức năng hàm thường có sẵn trong mỗi DBMS, nhưng sự thực thi chức năng đó có thể khác nhau rất lớn. Để thể hiện chỉ mức độ khó hiểu của vấn đề này, bảng liệt kê 3 hàm thường sử dụng và cú pháp của chúng khi được sử dụng bởi các DBMS khác nhau.

*** Sự khác biệt hàm của DBMS**

Chức năng	Cú pháp
Trích xuất một phần của một chuỗi	Access sử dụng MID (). DB2, Oracle, và PostgreSQL sử dụng SUBSTR (). MySQL, SQL Server, và Sybase sử dụng SUBSTRING ().
Chuyển đổi kiểu dữ liệu	Access và Oracle sử dụng nhiều hàm, một hàm cho mỗi kiểu chuyển đổi. DB2 và PostgreSQL sử dụng CAST (). MySQL, SQL Server, và Sybase use CONVERT ().
Lấy ngày hiện hành	Access sử dụng NOW (). DB2 và PostgreSQL sử dụng CURRENT_DATE. MySQL sử dụng CURDATE (). Oracle sử dụng SYSDATE. SQL Server và Sybase sử dụng GETDATE ().

- Như bạn có thể thấy, không giống như câu lệnh SQL, các hàm SQL không linh động. Điều này có nghĩa là đoạn mã bạn viết để thực thi SQL trên hệ thống cụ thể có thể không thực thi trên các hệ thống khác.

- Với tính năng linh động của mã (Code), nhiều nhà lập trình chọn không sử dụng bất kỳ tính năng thực thi chuyên biệt nào. Mặc dù đây là một cái nhìn hơi duy tâm, nhưng nó không luôn là các vấn đề quan tâm nhất của việc thực thi ứng dụng. Nếu bạn chọn không sử dụng những hàm này, bạn sẽ phải tạo cho ứng dụng đoạn mã làm việc vất vả hơn. Nó phải sử dụng phương pháp khác để làm những gì DBMS có thể làm hiệu quả hơn.

Nếu bạn quyết định sử dụng các hàm, đảm bảo rằng bạn ghi chú đoạn mã tốt, để vài ngày sau bạn (hoặc một nhà lập trình khác) sẽ biết câu lệnh SQL bạn đang viết thực hiện điều gì.

*** Sử dụng các hàm**

- Hầu hết các hoạt động thực thi SQL hỗ trợ các dạng hàm sau:
- + Các hàm xử lý Text được sử dụng để xử lý các chuỗi Text (ví

dụ, cắt hoặc chèn thêm các ký tự (chuỗi con) và chuyển đổi các ký tự thành dạng hoa hay thường)

- + Các hàm xử lý số được sử dụng để thực hiện các phép tính toán học trên dữ liệu số (ví dụ, trả về các số tuyệt đối và thực hiện các phép toán đại số).

- + Các hàm ngày và thời gian được sử dụng để xử lý các giá trị ngày và thời gian và trích xuất các thành phần cụ thể từ các giá trị (ví dụ, trả về các dạng khác nhau của ngày, và kiểm tra sự hợp lệ ngày).

- + Các hàm hệ thống trả về thông tin riêng cho DBMS đang sử dụng (ví dụ, trả về thông tin đăng nhập User).

- Trong bài học trước, bạn đã thấy một hàm được sử dụng dưới dạng một phần tử của một danh sách cột trong câu lệnh SELECT, nhưng đó không phải là tất cả các hàm số có thể thực hiện được. Bạn có thể sử dụng các hàm trong các phần khác của câu lệnh SQL (ví dụ trong mệnh đề WHERE), cũng như trong các câu lệnh SQL khác.

Các câu lệnh xử lý text

- Hàm RTRIM () được sử dụng để cắt bỏ khoảng trắng từ phía cuối của một giá trị cột. Đây là một ví dụ khác, lần này sử dụng hàm UPPER ().

```
SELECT vend_name, ;  
UPPER (vend_name) AS vend_name_upcase ;  
FROM Vendors ;  
ORDER BY vend_name;
```

Kết quả

Vend_name	vend_name_upcase
-----------	------------------

Bear Emporium	BEAR EMPORIUM
---------------	---------------

Bears R Us	BEARS R US
------------	------------

Doll House Inc. DOLL HOUSE INC.
 Fun and Games FUN AND GAMES
 Furball Inc. FURBALL INC.
 Jouets et ours JOUETS ET OURS

- Như bạn có thể thấy, UPPER () chuyển đổi Text sang dạng chữ hoa và vì vậy trong ví dụ này mỗi nhà sản xuất được liệt kê hai lần, thứ nhất chính xác như được lưu trữ trong Table Vendors, và sau đó được chuyển đổi thành dạng chữ hoa ở cột vend_name_upcase.

Các hàm xử lý Text thường được sử dụng:

Hàm	Mô tả
LEFT () (hay hàm substring ())	Trả về các ký tự bắt đầu từ bên trái của chuỗi
LENGTH () (DATALENGTH () hoặc LEN ())	Trả về chiều dài của một chuỗi.
LOWER ()	Chuyển đổi các chuỗi thành dạng chữ thường
LTRIM () (LCASE () nếu sử dụng Access)	Cắt bỏ khoảng trắng từ bên trái của chuỗi
RIGHT () (hoặc sử dụng hàm substring ())	Trả về các ký tự phía bên phải của chuỗi
RTRIM ()	Cắt bỏ khoảng trắng từ bên phải của chuỗi.
SOUNDEX ()	Trả về giá trị SOUNDEX của một chuỗi
UPPER () (UCASE () nếu sử dụng Access)	Chuyển đổi chuỗi thành dạng chữ hoa

- Một mục trong bảng đòi hỏi phải giải thích thêm SOUDEX là một thuật toán vốn chuyển đổi chuỗi Text bất kỳ sang dạng mẫu chữ số (Alphanumeric pattern) mô tả biểu diễn ngữ âm của Text

đó. SOUNDEX đưa vào bản mô tả (Account) các ký tự phát âm tương tự và các âm tiết, cho phép các chuỗi được so sánh theo cách chúng phát âm tốt hơn cách chúng được gõ nhập.

- Hỗ trợ SOUNDEX: Hàm SOUNDEX () không được hỗ trợ bởi Microsoft Access hay PostgreSQL, và vì thế ví dụ sau sẽ không hoạt động trên các DBMS đó.

- Đây là một ví dụ về việc sử dụng hàm SOUNDEX (). Khách hàng Kids Place nằm trong Table Customers và có một đối tác có tên là Michelle Green. Nhưng chuyện gì sẽ xảy ra nếu đó là một lỗi gõ phím, và người bạn thực sự được gõ là Michelle Green? Rõ ràng, việc tìm kiếm tên người bạn này chính xác sẽ không trả về dữ liệu nào cả, như được trình bày ở đây:

```
SELECT cust_name, cust_contact ;  
FROM Customers ;  
WHERE cust_contact = 'Michael Green';
```

Kết quả

Cust_name	cust_contact
-----------	--------------

Bây giờ, bạn thử tìm kiếm lại bằng cách sử dụng hàm SOUNDEX () để tìm tất cả các tên những người bạn phát âm tương tự Michael Green:

```
SELECT cust_name, cust_contact ;  
FROM Customers ;  
WHERE SOUNDEX (cust_contact) = SOUNDEX ('Michael  
Green');
```

Kết quả

Cust_name	cust_contact
-----------	--------------

Kids Place Michelle Green

- Trong ví dụ này, mệnh đề WHERE sử dụng hàm SOUNDEX () để chuyển đổi cả hai giá trị cột cust_contact và tìm kiếm chuỗi cho các giá trị SOUNDEX của chúng. Bởi vì Michael Green và Michelle phát âm như nhau, các giá trị SOUNDEX của chúng thoả mãn, và vì vậy mệnh đề WHERE lọc chính xác dữ liệu mong muốn.

Các hàm xử lý ngày và thời gian (Date and time)

- Ngày (ngày - tháng - năm) và thời gian được lưu trữ trong các Table bằng cách sử dụng các kiểu dữ liệu, và mỗi DBMS có dạng cụ thể riêng của nó. Các giá trị ngày và thời gian được lưu trữ ở các dạng đặc biệt để chúng có thể được sắp xếp hay được lọc một cách nhanh chóng và hiệu quả, cũng như tiết kiệm được không gian lưu trữ vật lý.

- Dạng được sử dụng để lưu trữ ngày và thời gian thường không sử dụng cho các ứng dụng của bạn, và vì thế các hàm ngày và thời gian hầu hết luôn được sử dụng để lọc, mở rộng và xử lý các giá trị này. Vì vậy, các hàm xử lý ngày và thời gian là một trong số các hàm quan trọng trong ngôn ngữ SQL. Thật không may, chúng cũng có khuynh hướng kém nhất quán và linh động.

- Để thể hiện việc sử dụng hàm xử lý ngày, đây là một ví dụ đơn giản. Table Orders (đưa bằng hàng) chứa tất cả các đơn đặt hàng theo một cùng với một ngày đặt hàng. Để truy vấn một danh sách tất cả các đơn đặt hàng năm 2004 trong SQL Server và Sybase, thực hiện như sau:

```
SELECT order_num ;  
FROM Orders ;  
WHERE DATEPART (yy, order_date) = 2004;
```

Kết quả

Order_num

20005

20006

20007

20008

20009

Trong Access:

```
SELECT order_num ;
```

```
FROM Orders ;
```

```
WHERE DATEPART ('yyyy', order_date) = 2004;
```

- Ví dụ này (cả phiên bản SQL và Sybase, và Access) sử dụng hàm DATEPART (), hàm, cũng như tên của nó đề nghị, trả về một phần của ngày. DATEPART () nhận hai tham số, phần trả về, và ngày để nó trả về. Trong ví dụ của chúng ta, DATEPART () trả về chỉ năm từ cột order_date. Bằng cách so sánh nó với năm 2004, mệnh đề WHERE có thể lọc chỉ các đơn đặt hàng cho năm đó.

- Đây là phiên PostSQL vốn sử dụng một hàm tương tự có tên là DATE_PART ():

```
SELECT order_num ;
```

```
FROM Orders ;
```

```
WHERE DATE_PART ('year', order_date) = 2004;
```

- MySQL có tất cả các hàm xử lý một phần của ngày, nhưng không phải là DATEPART (). Các User của MySQL có thể sử dụng một hàm có tên là YEAR () để trích năm từ một ngày.

```
SELECT order_num ;
```

```
FROM Orders ;
```

```
WHERE YEAR (order_date) = 2004;
```

Oracle không có hàm DATEPART (), nhưng nó có nhiều hàm xử lý ngày khác vốn có thể được sử dụng để thực hiện cùng chức

năng truy vấn đó. Đây là ví dụ:

```
SELECT order_num ;
```

```
FROM Orders ;
```

```
WHERE to_number (to_char (order_date,'YY')) =2004;
```

- Trong ví dụ này, hàm `to_char ()` được sử dụng để trích một phần của ngày, và `to_number ()` được sử dụng để chuyển đổi nó sang dạng số để có thể so sánh với số 2004.

- Một cách khác để thực hiện cùng tác vụ trên là sử dụng toán tử `BETWEEN`:

```
SELECT order_num ;
```

```
FROM Orders ;
```

```
WHERE order_date ;
```

```
BETWEEN to_date ('01-JAN-2004') AND;
```

```
to_date ('31-DEC-2004');
```

- Trong ví dụ này, hàm `to_date ()` của Oracle được sử dụng để chuyển đổi hai chuỗi thành dạng ngày. Một chứa ngày 01-02-2004 (ngày 1 tháng 2 năm 2004), và một chứa ngày 31-12-2004. Một toán tử `BETWEEN` chuẩn được sử dụng để tìm tất cả các đơn đặt hàng giữa hai ngày đó. Điều đáng chú ý là cùng một đoạn mã như vậy không thực hiện được trong SQL Server bởi vì nó không hỗ trợ hàm `to_date ()`. Tuy nhiên, nếu bạn đã thay thế `to_date` bằng `DATEPART ()`, bạn thực sự có thể sử dụng loại câu lệnh này.

- Các ngày của Oracle. Các ngày theo dạng DD-MM-YYYY (như trong ví dụ được trình bày ở trên) thường được xử lý bởi Oracle tốt mặc dù không đổi thành dạng ngày rõ ràng bằng cách sử dụng `to_date ()`, tuy nhiên, vẫn an toàn, hàm đó luôn được sử dụng.

- Ví dụ trình bày ở đây đã trích và sử dụng một phần của ngày (năm). Để lọc một tháng cụ thể, xử lý tương tự có thể được thực

hiện, bằng cách chỉ định một toán tử AND và cả hai phép so sánh năm và tháng.

- Các DBMS thường đưa ra thêm các phép trích đơn giản một phần của một ngày. Hầu hết có các hàm để so sánh các ngày, biểu diễn ngày dựa trên số học, các tùy chọn để định dạng ngày, v.v... Nhưng bạn đã thấy, các hàm xử lý ngày và thời gian đặc biệt cụ thể cho mỗi DBMS cho danh sách các hàm xử lý thời gian.

Các hàm xử lý số

- Các hàm xử lý số chỉ xử lý dữ liệu kiểu số. Những hàm này có khuynh hướng được sử dụng chủ yếu cho các phép tính đại số, lượng giác, hoặc hình học và vì vậy không thường được sử dụng như các hàm xử lý ngày và thời gian và các hàm xử lý chuỗi.

- Một điều chăm biên là tất cả các hàm đã tìm thấy trong các DBMS chính, các hàm số là các hàm hầu như là đồng nhất và nhất quán.

Hàm	Mô tả
ABS ()	Lấy giá trị tuyệt đối của một số
COS ()	Lấy Cosin trong lượng giác
EXP ()	Lấy số mũ
PI ()	Lấy giá trị PI
SIN ()	Lấy Sin trong lượng giác
SQRT ()	Lấy căn bậc hai
TAN ()	Lấy tang trong lượng giác

- Tham khảo tài liệu DBMS của bạn cho một danh sách các hàm xử lý toán học được hỗ trợ.

Các hàm tập hợp (Aggregate Function)

- Rất cần thiết khi tổng kết dữ liệu mà không thực sự truy vấn tất cả dữ liệu. Và SQL cung cấp các hàm đặc biệt cho mục đích

này. Sử dụng các hàm này, các Query SQL (truy vấn) thường được sử dụng truy vấn dữ liệu để phân tích và báo cáo các kết quả.

- Các ví dụ về kiểu truy vấn này là:

- + Xác định số dòng trong một Table (hoặc số dòng vốn bất gặp một số điều kiện hoặc chứa một giá trị xác định).

- + Thu được tổng của một tập các dòng trong một Table.

- + Tìm các giá trị lớn nhất, nhỏ nhất, và trung bình trong một cột Table (mỗi cột cho tất cả các dòng hoặc cho tất cả các dòng cụ thể).

- Trong mỗi ví dụ này, bạn muốn một bản tổng kết dữ liệu trong một Table, không là dữ liệu thực sự của chính nó. Vì vậy, việc trả về dữ liệu Table thực sẽ là một sự lãng phí thời gian và tài nguyên (không đề cập tới băng thông). Nhắc lại, tất cả các bạn thực sự muốn là thông tin tổng kết.

- Để giúp thuận tiện cho kiểu truy vấn này. SQL đưa ra một tập 5 hàm tập hợp, được liệt kê trong bảng dưới. Những hàm này cho phép bạn thực hiện tất cả các kiểu truy vấn, chỉ được liệt kê. Bạn sẽ cảm thấy bớt căng thẳng khi biết rằng không giống như các hàm xử lý dữ liệu trong bài học trước, các hàm tập hợp của SQL hầu hết được các chương trình triển khai SQL hỗ trợ khá nhất quán.

Các hàm tập hợp

Hàm	Mô tả
AVG ()	Trả về giá trị trung bình của cột
COUNT ()	Trả về tổng số dòng trong một cột
MAX ()	Trả về giá trị lớn nhất của cột
MIN ()	Trả về giá trị nhỏ nhất
SUM ()	Trả về tổng của một cột

- Cách sử dụng từng hàm này sẽ được giải thích trong các phần sau.

Hàm AVG ()

AVG () được sử dụng để trả về giá trị trung bình của một cột cụ thể bằng cách đếm tổng số dòng trong Table và tính tổng các giá trị của chúng. AVG () có thể được sử dụng để trả về giá trị trung bình của tất cả các cột hoặc các cột hay các dòng cụ thể.

- Ví dụ đầu tiên này sử dụng AVG () để trả về giá trị trung bình của tất cả các sản phẩm trong Table Products:

```
SELECT AVG (prod_price) AS avg_price ;
```

```
FROM Products;
```

Kết quả

Avg_price

6.823333

- Câu lệnh trên trả về một giá trị đơn, avg_price vốn chứa giá trị trung bình của tất cả các sản phẩm trong Table Products, avg_price là một bí danh.

- AVG () còn có thể được sử dụng để xác định giá trị trung bình của các cột hoặc các dòng cụ thể. Ví dụ sau trả về giá trị trung bình của các sản phẩm của một nhà sản xuất cụ thể:

```
SELECT AVG (prod_price) AS avg_price ;
```

```
FROM Products ;
```

```
WHERE vend_id = 'DLL01';
```

Kết quả

Avg_price

3.8650

- Câu lệnh SELECT này khác câu lệnh trước chỉ ở chỗ câu lệnh này chỉ chứa một mệnh đề WHERE. Mệnh đề WHERE chỉ lọc các

sản phẩm có vendor_id là DLL01, và vì vậy, giá trị được trả về trong avg_price là giá trị trung bình của chỉ các sản phẩm của nhà sản xuất đó mà thôi.

- Chỉ các cột riêng lẻ AVG () có thể chỉ được sử dụng để xác định giá trị trung bình của một cột có kiểu số cụ thể, và tên của cột đó phải được chỉ định dưới dạng tham số hàm. Để lấy giá trị trung bình của nhiều cột, phải sử dụng nhiều hàm AVG ().

- Các giá trị NULL. Các dòng của cột có chứa các giá trị NULL thì hàm AVG () bỏ qua.

Hàm COUNT ()

COUNT () chỉ thực hiện đếm. Sử dụng COUNT (), bạn có thể xác định số dòng trong một Table hoặc số dòng vốn thoả mãn một điều kiện cụ thể.

COUNT () có thể được sử dụng theo 2 cách:

- + Sử dụng COUNT (*) để đếm số dòng trong một Table, dù các cột có chứa giá trị hoặc giá trị NULL.

- + Sử dụng COUNT () để đếm số dòng chứa các giá trị trong một cột cụ thể, bỏ qua các giá trị NULL.

- Ví dụ đầu tiên này sẽ trả về tổng số khách hàng trong Table Customers (khách hàng):

```
SELECT COUNT (*) AS num_cust ;
```

```
FROM Customers;
```

Kết quả

Num_cust

5

- Trong ví dụ này, COUNT (*) được sử dụng để đếm tất cả các dòng, không quan tâm tới là các giá trị nào. Kết quả trả về trong num_cust.

Ví dụ sau đếm chỉ các khách hàng có một địa chỉ email:

```
SELECT COUNT (cust_email) AS num_cust ;
```

```
FROM Customers;
```

Kết quả

Num_cust

3

- Câu lệnh SELECT sử dụng COUNT (cust_email) để đếm chỉ các dòng có một giá trị trong cột cust_email. Trong ví dụ này, cust_email là 3 (nghĩa là chỉ có 3 trong 5 khách hàng có email).

- Các giá trị NULL: Các dòng của cột chứa các giá trị NULL được hàm COUNT () bỏ qua nếu tên cột được chỉ định, và không bỏ qua nếu (*) được sử dụng.

Hàm MAX ()

MAX () trả về giá trị lớn nhất trong một cột cụ thể. MAX () bắt buộc tên cột phải được chỉ định, như bạn thấy sau đây:

```
SELECT MAX (prod_price) AS max_price ;
```

```
FROM Products;
```

Kết quả

Max_price

11.9900

- Đây là hàm MAX () trả về giá món hàng đắt nhất trong Table Products.

- Sử dụng MAX () với dữ liệu không phải là kiểu số. Mặc dù MAX () thường được sử dụng để tìm các giá trị kiểu ngày hay kiểu số lớn nhất, nhiều DBMS (nhưng không phải là tất cả) cho phép hàm này được sử dụng để trả về các giá trị lớn nhất trong bất kỳ các cột nào bao

gồm cả những cột có kiểu Text. Khi được sử dụng với dữ liệu Text, MAX () trả về dòng cuối cùng nếu dữ liệu của cột đó được sắp xếp.

- Các giá trị NULL: Các dòng của cột chứa các giá trị NULL được hàm MAX () bỏ qua.

Hàm MIN ()

MIN () thực hiện đảo ngược của hàm MAX (); nó trả về giá trị nhỏ nhất trong một cột cụ thể. Cũng giống như hàm MAX (), MIN () đòi hỏi tên cột phải được chỉ định, như sau:

```
SELECT MIN (prod_price) AS min_price;
```

```
FROM Products;
```

Kết quả

Min_price

3.4900

- Đây là hàm MIN () trả về món hàng rẻ nhất trong Table Products.

- Sử dụng MIN () với dữ liệu không phải kiểu số: Mặc dù MIN () thường được sử dụng để tìm các giá trị kiểu ngày hay kiểu số nhỏ nhất, nhiều DBMS (nhưng không phải là tất cả) cho phép hàm này được sử dụng để trả về các giá trị nhỏ nhất trong bất kỳ các cột nào bao gồm cả những cột có kiểu Text. Khi được sử dụng với dữ liệu Text, MIN () trả về dòng cuối cùng nếu dữ liệu cột đó được sắp xếp.

- Các giá trị NULL: Các dòng của cột chứa các giá trị NULL được hàm MIN () bỏ qua.

Hàm SUM ()

SUM () trả về tổng các giá trị trong một cột cụ thể.

Đây là một ví dụ để minh họa điều này. Table OrderItems chứa các món hàng thực sự trong một đơn đặt hàng, và mỗi món hàng có

một số lượng (Quantity). Tổng các món hàng được đặt hàng (tổng tất cả các giá trị Quantity) có thể được truy vấn như sau:

```
SELECT SUM (Quantity) AS item_ordered ;
```

```
FROM OrderItems ;
```

```
WHERE order_num = 20005;
```

Kết quả

Item_ordered

200

- Hàm SUM (Quantity) trả về tổng số tất cả các số lượng món hàng trong một đơn đặt hàng, và mệnh đề WHERE bảo đảm rằng chỉ các món hàng của đúng đơn đặt hàng mới được đưa vào.

- SUM () còn có thể được sử dụng cho các giá trị được tính tổng. Trong ví dụ kế tiếp, tổng số đơn đặt hàng được truy vấn bằng cách tính tổng item_price*Quantity (Đơn giá*số lượng) cho mỗi món hàng.

```
SELECT SUM (item_price*Quantity) AS total_price ;
```

```
FROM OrderItems ;
```

```
WHERE order_num = 20005;
```

Kết quả

total_price

1648.0000

- Hàm SUM (item_price*Quantity) trả về tổng các giá trị mở rộng trong một đơn đặt hàng, và một lần nữa mệnh đề WHERE bảo đảm rằng chỉ các món hàng của đúng đơn đặt hàng mới được đưa vào.

- Thực hiện các phép tính trên nhiều cột. Tất cả các hàm tập hợp có thể được sử dụng để thực hiện các phép tính trên nhiều cột bằng cách sử dụng các toán tử toán học chuẩn, như trong ví dụ này.

- Các giá trị NULL. Các dòng của cột chứa các giá trị NULL được hàm SUM () bỏ qua.

Tập hợp theo giá trị phân biệt

5 hàm tập hợp có thể được sử dụng theo hai cách:

+ Để thực hiện các phép tính trên tất cả các dòng, hãy chỉ định đối số ALL hoặc không chỉ định đối số nào cả (bởi vì ALL là hành vi mặc định).

+ Để chỉ đưa vào các giá trị chỉ có duy nhất, hãy chỉ định đối số DISTINCT.

- ALL là đối số mặc định. Đối số ALL không cần phải được chỉ định bởi vì nó là hành vi mặc định. Nếu DISTINCT không được chỉ định, xem như ALL sẽ được chỉ định.

- Không có trong Access Microsoft Access không hỗ trợ dùng DISTINCT trong các hàm tập hợp, và vì thế ví dụ sau sẽ không làm việc với Access.

- Ví dụ sau sẽ sử dụng hàm AVG () để trả về giá sản phẩm trung bình của một nhà sản xuất cụ thể. Nó giống như câu lệnh SELECT được sử dụng ở trên, nhưng ở đây đối số DISTINCT được sử dụng để giá trung bình chỉ tính trên các giá duy nhất (giá xuất hiện một lần):

```
SELECT AVG (DISTINCT prod_price) AS avg_price ;
```

```
FROM Products ;
```

```
WHERE vend_id = 'DLL01';
```

Kết quả

Avg_price

4.2400

- Trong ví dụ này avg_price lớn hơn khi DISTINCT được sử dụng bởi vì có nhiều món hàng có cùng mức giá thấp. Loại chúng ra sẽ tăng giá trung bình lên.

- Chú ý DISTINCT có thể chỉ được sử dụng với COUNT () nếu một tên cột được chỉ định. DISTINCT không thể được sử dụng với COUNT (*). Tương tự, DISTINCT phải được sử dụng với một tên cột và không được sử dụng với một phép tính hoặc biểu thức.

- Sử dụng DISTINCT với MIN () và MAX (). Mặc dù DISTINCT về mặt kỹ thuật có thể được sử dụng với MIN () và MAX (), nhưng thực sự không ai làm như vậy. Các giá trị lớn nhất và nhỏ nhất trong một cột cũng sẽ giống nhau cho dù có các giá trị duy nhất hay không.

- Các đối số tập hợp bổ sung. Ngoài DISTINCT và ALL, một số DBMS hỗ trợ các đối số bổ sung như TOP và TOP PERCENT cho phép bạn thực hiện các phép tính trên các tập con của các kết quả Query. Tham khảo tài liệu DBMS của bạn để xác định chính xác đối số nào có sẵn cho bạn.

Kết hợp các hàm tập hợp

Tất cả các ví dụ về hàm tập hợp được sử dụng cho đến bây giờ chỉ đưa ra một hàm đơn. Nhưng thực tế các câu lệnh SELECT có thể chứa nhiều hàm tập hợp.

Hãy xem ví dụ này:

```
SELECT COUNT (*) AS num_item, ;  
MIN (prod_price) AS price_min, ;  
MAX (prod_price) AS price_max, ;  
AVG (prod_price) AS price_avg ;  
FROM Products;
```

Kết quả

Num_items	price_min	price_max	price_avg
-----------	-----------	-----------	-----------

9	3.4900	11.9900	6.823333
---	--------	---------	----------

- Đây là một câu lệnh SELECT đơn giản thực hiện 4 phép tính tập hợp trong một bước và trả về 4 giá trị số (số món hàng trong Table Products, và các giá sản phẩm lớn nhất, nhỏ nhất và trung bình).

- Đặt tên các bí danh. Khi chỉ định các tên bí danh để chứa các

kết quả của một hàm tập hợp. Cố gắng đừng sử dụng tên của các cột thực trong Table. Mặc dù thực sự không có gì không hợp lệ khi làm như vậy, nhiều trình diễn khai SQL không hỗ trợ điều này và sẽ phát sinh các thông điệp lỗi.

5. Câu lệnh SQL

a. Câu lệnh Select

** Truy vấn các cột riêng lẻ*

Chúng ta bắt đầu với một câu lệnh đơn giản như sau:

Select prod_name ;

From Products;

- Câu lệnh trên sử dụng câu lệnh Select để truy vấn một cột có tên là Prod_Name từ Table có tên là Products. Tên cột bạn muốn được chỉ định sau từ khoá Select, sau từ khoá Form là tên của Table được truy vấn dữ liệu. Kết quả xuất từ câu lệnh này như sau:

Kết quả

Prod_name

Fish bean bag toy

Bird bean bag toy

Rabbit bean bag toy

8 inch teddy bear

12 inch teddy bear

18 inch teddy bear

Raggedy Ann

King doll

Queen doll

- Dữ liệu chưa được sắp xếp. Xem lại nếu bạn thử thực hiện

Query này, bạn có thể thấy rằng dữ liệu được hiển thị khác hơn so với ở đây. Nếu trong trường hợp này, đừng lo lắng – nó đang làm việc chính xác đó. Nếu kết quả truy vấn không được sắp xếp rõ ràng, khi đó dữ liệu trả về không theo thứ tự nào cả. Nó có thể theo thứ tự mà dữ liệu thêm vào bảng, nhưng cũng có thể là không. Miễn là Query của bạn trả về cùng số dòng như trên thì câu lệnh thực hiện là đúng.

- Một câu lệnh đơn giản như câu lệnh được sử dụng ở trên sẽ trả về tất cả các dòng trong một Table. Dữ liệu sẽ được lọc ra một tập con kết quả, cũng như không được sắp xếp.

- Kết thúc các câu lệnh SQL phải được kết thúc bằng dấu chấm phẩy (;). Hầu hết các DBMS (hệ thống quản lý cơ sở dữ liệu) không đòi hỏi một câu lệnh đơn giản kết thúc bằng một dấu chấm phẩy. Nếu như DBMS của bạn cần, bạn có thể phải thêm nó vào. Tất nhiên bạn luôn có thể thêm nó vào nếu bạn muốn. Ngoại trừ Sybase Adaptive Server, nó không kết thúc câu lệnh SQL bằng dấu chấm phẩy.

- Câu lệnh SQL và kiểu chữ (Case) điều quan trọng mà bạn phải lưu ý là các câu lệnh SQL không phân biệt kiểu chữ hoa hay chữ thường, thay vì SELECT bạn có thể viết select hay Select. Nhiều nhà phát triển SQL nhận thấy rằng việc sử dụng tất cả các chữ hoa cho từ khoá SQL và chữ thường cho các cột và các tên Table giúp cho mã dễ đọc và dễ gỡ rối hơn rất nhiều. Tuy nhiên, bạn phải lưu ý rằng trong khi ngôn ngữ SQL không phân biệt kiểu chữ thì các tên của Table, Column, và các giá trị cũng không (tùy thuộc MDBS của bạn và cách bạn cấu hình nó).

** Truy vấn nhiều cột*

- Để truy vấn nhiều cột từ một bảng, cũng dùng tương tự như câu lệnh SELECT đã sử dụng. Chỉ khác là tên của các cột phải được chỉ định sau từ khoá SELECT, và mỗi cột phải tách nhau bởi một dấu phẩy (,).

Câu lệnh SELECT sau truy vấn 3 cột từ bảng Products

SELECT prod_id, prod_name, prod_price ;

FROM Products;

- Như trong ví dụ trước, câu lệnh này sử dụng SELECT để truy vấn dữ liệu từ bảng Products. Trong ví dụ này, có đến 3 cột được chọn, tên mỗi cột cách nhau bởi dấu phẩy. Kết quả từ câu lệnh này như sau:

Kết quả

Prod_id	prod_name	prod_price
BNBG01	Fish bean bag toy	3.4900
BNBG02	Bird bean bag toy	3.4900
BNBG03	Rabbit bean bag toy	3.4900
BR01 8	inch teddy bear	5.9900
BR02 12	inch teddy bear	8.9900
BR01 18	inch teddy bear	11.9900
RGAN01	Raggedy Ann	4.9900
RYL01	King doll	9.4900
RYL02	Queen doll	9.4900

- Biểu diễn dữ liệu như bạn thấy kết quả như trên, các câu lệnh SQL thường trả về dữ liệu thô, dữ liệu chưa định dạng. Định dạng dữ liệu là một vấn đề biểu diễn dữ liệu, không phải là vấn đề truy vấn dữ liệu. Vì vậy, biểu diễn dữ liệu (trong ví dụ này là kiểu hiển thị giá ở trên dưới dạng số lượng tiền với các chữ số thập phân chính xác). Các dữ liệu được truy lục chưa được sắp xếp (việc định dạng không được cung cấp bởi trình ứng dụng) rất ít khi sử dụng.

** Truy vấn tất cả các cột*

Ngoài việc chỉ định các cột mong muốn các câu lệnh SELECT

cũng có thể được sử dụng để truy lục tất cả các cột mà không phải liệt kê chúng riêng lẻ. Điều này được thực hiện bằng cách sử dụng ký tự đại diện (*) thay vì phải gõ từng cột vào câu lệnh.

```
SELECT * ;
```

```
FROM Products;
```

- Khi sử dụng ký tự (*), tất cả các cột trong Table sẽ được trả về. Thứ tự cột thường sẽ theo thứ tự cột khác được xác định trong Table. Tuy nhiên, dữ liệu ít khi được hiển thị như vậy. Thông thường thì nó sẽ được trả về cho một trình ứng dụng vốn định dạng hoặc biểu diễn dữ liệu theo nhu cầu.

- Việc sử dụng các ký tự mở rộng là một quy tắc. Tốt hơn là bạn không nên sử dụng ký tự đại diện nếu bạn không thực sự cần thiết hiển thị tất cả các cột trong Table. Mặc dù dùng các ký tự đại diện sẽ tiết kiệm thời gian và công sức, nhưng việc truy vấn các cột không cần thiết thường làm chậm đi tốc độ truy vấn và việc thực thi trình ứng dụng của bạn.

b. Sắp xếp các dữ liệu được truy vấn

**** Sắp xếp dữ liệu***

Như bài học trước, câu lệnh SQL sau khi trả về một cột từ Table cơ sở dữ liệu. Nhưng bạn hãy xem kết quả hiển thị không theo thứ tự nào cả.

```
SELECT prod_name ;
```

```
FROM Products;
```

Kết quả

Prod_name

Fish bean bag toy

Bird bean bag toy

Rabbit bean bag toy

8 inch teddy bear

12 inch teddy bear

18 inch teddy bear

Raggedy Ann

King doll

Queen doll

- Thực sự dữ liệu được truy vấn không theo một thứ tự ngẫu nhiên duy nhất. Nếu chưa được sắp xếp, dữ liệu thường sẽ được hiển thị theo trật tự xuất hiện trong bảng. Đây có thể là trật tự dữ liệu được thêm vào bảng ban đầu. Tuy nhiên, nếu dữ liệu thường xuyên được cập nhật hoặc xóa, trật tự này sẽ bị ảnh hưởng bởi cách DBMS sử dụng lại không gian lưu trữ tái tạo. Lý thuyết về thiết kế cơ sở dữ liệu quan hệ phát biểu rằng trình tự dữ liệu được truy vấn không thể cho là có nghĩa nếu chỉ định việc tạo thứ tự rõ ràng.

- Để sắp xếp dữ liệu được truy vấn bằng câu lệnh SELECT, mệnh đề ORDER BY được sử dụng. ORDER BY lấy tên của một hoặc nhiều cột để sắp xếp kết quả xuất theo nó.

SELECT prod_name ;

FROM Products ;

ORDER BY prod_name;

- Câu lệnh này giống hệt câu lệnh trước, ngoại trừ nó còn chỉ định một mệnh đề ORDER BY chỉ thị phần mềm hệ thống quản lý cơ sở dữ liệu sắp xếp cột prod_name theo thứ tự Alphabet. Kết quả như sau:

Kết quả

12 inch teddy bear

18 inch teddy bear

8 inch teddy bear

Bird bean bag toy
Fish bean bag toy
King doll
Queen doll
Rabbit bean bag toy
Raggedy Ann

Vị trí của một mệnh đề ORDER BY: Khi chỉ định một mệnh đề ORDER BY, việc bảo đảm rằng nó là mệnh đề cuối cùng trong câu lệnh SQL. Việc sử dụng các mệnh đề không theo thứ tự sẽ phát sinh một số thông điệp lỗi.

- Sắp xếp các cột không được chọn. Thông thường, các cột được sử dụng trong mệnh đề ORDER BY sẽ là các cột vốn được hiển thị. Tuy nhiên, điều này thực sự không cần, và dữ liệu hoàn toàn có thể được sắp xếp theo cột không được truy vấn.

** Sắp xếp theo nhiều cột*

Đôi khi chúng ta cũng cần sắp xếp dữ liệu theo nhiều cột. Ví dụ, nếu bạn đang hiển thị một danh sách nhân viên, bạn có thể muốn dữ liệu hiển thị được sắp xếp theo Lastname và Fistname (đầu tiên theo Lastname, và sau đó mỗi Lastname được sắp xếp theo Fistname).

Để sắp xếp theo nhiều cột, đơn giản chỉ việc chỉ định các tên cột được tách biệt bằng dấu phẩy (thực hiện chỉ khi bạn đang chọn nhiều cột).

Đoạn mã sau đây truy vấn 3 cột và sắp xếp kết quả theo 2 cột đầu tiên là theo giá và sau đó là theo tên.

```
SELECT prod_id, prod_price, prod_name ;  
FROM Products ;
```

ORDER BY prod_price, prod_name;

Kết quả

Prod_id	prod_price	prod_name
---------	------------	-----------

BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy
RGAN01	4.9900	Raggedy Ann
BR01	5.9900 8	inch teddy bear
BR02	8.9900	12 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR03	11.9900	18 inch teddy bear

Điều quan trọng là bạn phải tìm hiểu khi bạn đang sắp xếp theo từng cột, đảm bảo xác định thứ tự chính xác. Nói cách khác, việc sử dụng kết quả xuất trong ví dụ trên, các sản phẩm được sắp xếp theo prod_price. Nếu tất cả trong cột prod_price đều duy nhất thì không có dữ liệu sắp xếp trong cột prod_name.

** Sắp xếp theo vị trí cột*

Ngoài việc có thể chỉ định sắp xếp bằng cách sử dụng tên cột, ORDER BY cũng cung cấp việc chỉ định việc sắp xếp theo vị trí cột liên quan. Cách tốt nhất để tìm hiểu xem qua một ví dụ.

```
SELECT prod_id, prod_price, prod_name ;
```

```
FROM Products;
```

```
ORDER BY 2, 3;
```

Kết quả

Prod_id	prod_price	prod_name

BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy
RGAN01	4.9900	Raggedy Ann
BR01	5.9900 8	inch teddy bear
BR02	8.9900	12 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR03	11.9900	18 inch teddy bear

- Bạn có thể thấy, kết quả xuất hiện trên giống với kết xuất của Query truy vấn trên. Cái khác biệt ở đây là trong mệnh đề ORDER BY. Thay vì chỉ định các tên cột, các vị trí cột được chọn liên quan trong danh sách SELECT được chỉ định. ORDER BY 2 nghĩa là sắp xếp theo-cột 2 trong danh sách SELECT, tức là cột prod_price. ORDER BY 2, 3. Nghĩa là sắp xếp theo cột prod_price, rồi sau đó sắp xếp theo cột prod_name.

- Thuận lợi chính của kỹ thuật này là đỡ mất thời gian gõ lại các tên cột. Nhưng cũng có một số điểm bất lợi. Trước hết, việc liệt kê tên các cột không rõ ràng sẽ gia tăng khả năng phạm lỗi tạo thứ tự lại khi bạn thay đổi danh sách SELECT (quên thực hiện các thay đổi tương ứng cho mệnh đề ORDER BY).

Cuối cùng, rõ ràng bạn không thể sử dụng kỹ thuật này khi sắp xếp các cột vốn không nằm trong danh sách SELECT.

- Sắp xếp theo cột không được rõ ràng, kỹ thuật này không thể được sử dụng khi sắp xếp theo các cột vốn không xuất hiện trong

danh sách SELECT. Tuy nhiên bạn có thể kết hợp sử dụng các tên cột thực và các vị trí cột liên quan trong một câu lệnh đơn khi cần thiết.

** Chỉ định hướng sắp xếp*

Việc sắp xếp dữ liệu không hạn chế ở các thứ tự sắp xếp tăng từ A đến Z. Mặc dù đây là thứ tự sắp xếp mặc định, mệnh đề ORDER BY cũng có thể được sử dụng để sắp xếp theo thứ tự giảm. Để sắp xếp theo thứ tự giảm bạn dùng từ khoá DESC.

Ví dụ: Sắp xếp các sản phẩm theo giá và theo thứ tự giảm dần

```
SELECT prod_id, prod_price, prod_name ;
```

```
FROM Products ;
```

```
ORDER BY prod_price DESC;
```

Kết quả

Prod_id	prod_price	prod_name
BR03	11.9900	18 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR02	8.9900	12 inch teddy bear
BR01	5.9900 8	inch teddy bear
RGAN01	4.9900	Raggedy Ann
BNBG01	3.4900	Fish bean bag toy
BNBG02	3.4900	Bird bean bag toy
BNBG03	3.4900	Rabbit bean bag toy

Nhưng điều gì xảy ra nếu bạn sắp xếp theo nhiều cột: Ví dụ sau sắp xếp các sản phẩm theo thứ tự tăng dần, cộng thêm tên sản phẩm.

```
SELECT prod_id, prod_price, prod_name ;
```

```
FROM Products;
```

```
ORDER BY prod_price DESC, prod_name;
```

Kết quả

Prod_id	prod_price	prod_name
BR03	11.9900	18 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR02	8.9900	12 inch teddy bear
BR01	5.9900	8 inch teddy bear
RGAN01	4.9900	Raggedy Ann
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy

- Từ khoá DESC chỉ áp dụng cho tên cột vốn đứng trước nó. Trong ví dụ trên, DESC được chỉ định cho cột prod_price, nhưng không chỉ định cho cột prod_name. Vì vậy, cột Prod_Name (trong mỗi giá) vẫn được sắp xếp theo thứ tự tăng dần.

- Sắp xếp tăng dần trên nhiều cột nếu bạn muốn sắp xếp giảm trên nhiều cột, phải đảm bảo mỗi cột phải có từ khoá DESC riêng.

- Điều đáng chú ý là DESC viết tắt của DESCENDING, và cả hai từ khoá này đều có thể sử dụng. Đối lập với DESC là ASC (hay ASCENDING), mà nó có thể được chỉ định để sắp xếp theo thứ tự tăng. Tuy nhiên, ASC không thường được sử dụng vì thứ tự tăng là thứ tự mặc định (và được sử dụng nếu cả Asc và DESC không được chỉ định).

- Phân biệt kiểu chữ và thứ tự sắp xếp khi bạn đang sắp xếp dữ liệu văn bản, A có giống a không? Và có đứng trước B và Z? Đây không phải là những câu hỏi lý thuyết và các câu trả lời phụ thuộc vào các cơ sở dữ liệu được thiết lập.

Trong thứ tự sắp xếp Dictionary, A được xem như là a, và đó là hành vi mặc định cho hầu hết các hệ thống quản lý cơ sở dữ liệu. Tuy nhiên, hầu hết các DBMS cho phép các nhà quản trị cơ sở dữ liệu thay đổi hành vi này khi cần thiết (nếu cơ sở dữ liệu chứa nhiều ký tự ngôn ngữ nước ngoài, điều này có thể trở nên cần).

c. Lọc dữ liệu

**** Sử dụng mệnh đề WHERE***

Các Table cơ sở dữ liệu thường chứa một lượng lớn dữ liệu, và bạn ít khi cần truy vấn tất cả các hàng trong Table. Bạn sẽ thường truy xuất một tập con dữ liệu của Table cho các hoạt động cụ thể hoặc các Report (bản báo cáo). Việc truy vấn chỉ dữ liệu bạn muốn đòi hỏi phải chỉ định vùng điều kiện tìm kiếm (Criteria), còn được gọi là điều kiện lọc.

Trong câu lệnh SELECT, dữ liệu được lọc bằng chỉ định vùng điều kiện tìm kiếm trong mệnh đề WHERE. Mệnh đề WHERE đặt sau tên Table (mệnh đề FORM) như sau:

```
SELECT prod_name, prod_price ;
```

```
FROM Products ;
```

```
WHERE prod_price = 3.49;
```

- Câu lệnh này truy vấn 2 cột trong Table Products, nhưng thay vì trả về tất cả các hàng với giá trị Prod_Price = 5.00 được trả về như sau:

Kết quả

<i>Prod_name</i>	<i>prod_price</i>
-----	-----
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900

Ví dụ này sử dụng một phép kiểm tra đơn giản. Nó kiểm tra xem một cột có giá trị được chỉ định hay không và nó đọc dữ liệu đó. Nhưng SQL cho phép bạn thực hiện không chỉ một phép kiểm tra bằng.

- PostgreSQL có các quy tắc chặt chẽ trong việc quản lý các giá trị được truyền cho câu lệnh SQL, đặc biệt gần gũi với kiểu số được sử dụng trong các cột thập phân. Ví dụ trước có thể không hoạt động trên PostgreSQL. Để ví dụ trên thực hiện bình thường, bạn có thể cần phải chỉ định rõ rằng 2.56 là một số bằng cách gõ nhập thêm trong mệnh đề Where, thay 2.56 thành = decimal '2.56'.

- Ứng dụng chống SQL lọc dữ liệu cũng có thể được lọc ở cấp ứng dụng. Để thực hiện điều này, câu lệnh SELECT của SQL truy vấn nhiều dữ liệu hơn thực sự đòi hỏi trong ứng dụng Client, và mã Client lập qua phần dữ liệu được trả về để trích xuất chỉ các dòng cần thiết.

- Theo quy tắc này, bài thực hành sẽ hết sức nhàm chán. Các cơ sở dữ liệu được tối ưu để thực hiện lọc một cách nhanh chóng và có hiệu quả. Việc tạo các ứng dụng Client thực hiện công việc của các cơ sở dữ liệu sẽ đột ngột ảnh hưởng đến sự thực thi ứng dụng và sẽ tạo ra các ứng dụng vốn không thể linh hoạt. Hơn nữa, nếu dữ liệu được lọc Client, Server phải gửi dữ liệu không cần thiết qua các kết nối mạng, gây ra sự lãng phí băng thông mạng.

- Vị trí mệnh đề Where khi sử dụng cả hai mệnh đề ORDER BY và WHERE, đảm bảo rằng ORDER BY phải đứng sau WHERE, nếu không sẽ phát sinh lỗi.

** Các toán tử trong mệnh đề WHERE*

Mệnh đề WHERE đầu tiên mà chúng ta đã xem qua ở các phép kiểm tra bằng – xác định xem một cột có chứa một giá trị nào đó hay không. SQL cung cấp một dãy toán tử điều kiện đầy đủ trong bảng sau:

Các toán tử trong mệnh đề WHERE

Toán tử mô tả

= Bằng

≠ Khác

!= Khác

< Nhỏ hơn

<= Nhỏ hơn hoặc bằng

!< Không nhỏ hơn

> Lớn hơn

>= Lớn hơn hoặc bằng

!> Không lớn hơn

BETWEEN Nằm giữa hai giá trị cụ thể

IS NULL Là một giá trị NULL

- Sự tương thích toán tử: Một số toán tử liệt kê trong bảng hơn thừa (Ví dụ < giống như !=. !< (Không nhỏ hơn) cũng giống >= (Lớn hơn hoặc bằng)). Không phải là tất cả những toán tử này đều hỗ trợ tất cả các DBMS. Tham khảo các tài liệu DBMS của bạn để xác định chính xác toán tử nào mà nó hỗ trợ.

* Kiểm tra một giá trị đơn

Chúng ta đã xem qua một ví dụ về phép kiểm tra bằng. Hãy xem qua một vài ví dụ thể hiện cách dùng toán tử khác.

Ví dụ đầu tiên này liệt kê tất cả các sản phẩm có giá trị nhỏ hơn 10\$.

```
SELECT prod_name, prod_price ;
```

```
FROM Products ;
```

```
WHERE prod_price <10;
```

Kết quả

Prod_name	prod_price
-----------	------------

Fish bean bag toy	3.4900
-------------------	--------

Bird bean bag toy	3.4900
-------------------	--------

Rabbit bean bag toy	3.4900
---------------------	--------

8 inch teddy bear	5.9900
-------------------	--------

12 inch teddy bear	5.9900
--------------------	--------

Raggedy Ann

King doll

Queen doll

Câu lệnh kế tiếp truy vấn tất cả các sản phẩm có giá 10\$ hoặc nhỏ hơn (mặc dù kết quả sẽ giống như trong ví dụ trước bởi vì không có món quà nào có giá trị chính xác là 10\$ cả).

```
SELECT prod_name, prod_price ;
```

```
FROM Products ;
```

```
WHERE prod_price <= 10;
```

** Kiểm tra các giá trị không phù hợp*

Câu lệnh kế tiếp liệt kê tất cả các sản phẩm không được chế tạo bởi nhà sản xuất DLL01:

```
SELECT vend_id, prod_name ;
```

```
FROM Products ;
```

```
WHERE vend_id <> 'DLL01';
```

Kết quả

Vend_id	prod_name
---------	-----------

BRS01	8 inch teddy bear
-------	-------------------

BRS01	12 inch teddy bear
BRS01	18 inch teddy bear
FNG01	King doll
FNG01	Queen doll

- Khi sử dụng các trích dẫn nếu bạn xem kỹ ở các điều kiện được sử dụng trong mệnh đề WHERE trên, bạn sẽ để ý rằng một số các giá trị được đính kèm trong dấu trích đơn (''), và những giá trị khác thì không. Dấu trích dẫn đơn sử dụng để phân định một chuỗi. Nếu bạn đang so sánh một giá trị đối với một cột là một kiểu dữ liệu (Datatype) chuỗi, đòi hỏi phải có dấu trích dẫn.

Sau đây là một ví dụ tương tự, ngoại trừ ví dụ này sử dụng toán tử != thay cho <>:

```
SELECT vend_id, prod_name ;
FROM Products ;
WHERE vend_id != 'DLL01';
```

- != hay <> có thể được sử dụng thay thế cho nhau. Tuy nhiên, không phải tất cả các DBMS hỗ trợ cả hai dạng toán tử khác này. Microsoft Access, ví dụ, hỗ trợ toán tử nhưng không hỗ trợ !=. Nếu bạn không chắc hãy tra cứu DBMS của bạn.

** Kiểm tra một miền các giá trị*

Để kiểm tra một dãy các giá trị, bạn có thể sử dụng toán tử BETWEEN. Cú pháp của nó hơi khác với các toán tử của mệnh đề WHERE bởi vì nó đòi hỏi hai giá trị: Giá trị bắt đầu và giá trị kết thúc của một miền.

Ví dụ, để kiểm tra tất cả các sản phẩm vốn nằm giữa 5\$ và 10\$ hoặc cho tất cả ngày rơi vào khoảng giữa ngày bắt đầu và ngày kết thúc cụ thể.

Ví dụ sau thể hiện cách dùng toán tử BETWEEN bằng cách

truy vấn tất cả các sản phẩm có giá trị nằm giữa 5\$ và 10\$

```
SELECT prod_name, prod_price ;  
FROM Products ;  
WHERE prod_price BETWEEN 5 AND 10;
```

Kết quả

Prod_name	prod_price
8 inch teddy bear	5.9900
12 inch teddy bear	8.9900
King doll	9.4900
Queen doll	9.4900

- Như bạn thấy ví dụ này, khi BETWEEN được sử dụng, hai giá trị phải được xác định – chặn dưới và chặn trên của miền bạn muốn. Hai giá trị này cũng phải được tách nhau bởi từ khóa AND. BETWEEN tương kết với tất cả các giá trị nằm trong miền, kèm theo các giá trị bắt đầu và kết thúc xác định.

** Kiểm tra các giá trị rỗng*

Khi một Table được tạo, nhà thiết kế Table có thể định các cột riêng lẻ có thể chứa giá trị rỗng hay không. Khi một cột chứa giá trị rỗng, thì ta nói rằng là nó chứa giá trị NULL.

Câu lệnh SELECT có một mệnh đề WHERE đặc biệt vốn có thể được sử dụng để kiểm tra các cột chứa các giá trị NULL – mệnh đề IS NULL. Cú pháp có dạng như sau:

```
SELECT prod_name ;  
FROM Products ;  
WHERE prod_price IS NULL;
```

Câu lệnh này trả về một danh sách các sản phẩm vốn không có

giá (một trường Prod_Price rỗng, nhưng không phải giá trị bằng 0), và bởi vì không có giá trị nào được trả về. Table Vendors, tuy nhiên, chắc chắn có chứa các cột có giá trị NULL – cột vend_state sẽ chứa NULL nếu không có bang (như trong trường hợp các địa chỉ không phải ở Mỹ).

```
SELECT vend_id ;  
FROM Vendors ;  
WHERE vend_state IS NULL;
```

Kết quả

Vend_id

FNG01

JTS01

- Các toán tử cụ thể của DBMS. Nhiều DBMS mở rộng tập hợp toán tử chuẩn, cung cấp các tùy chọn lọc nâng cao. Tham khảo tài liệu DBMS của bạn để biết thêm thông tin

d. Lọc dữ liệu nâng cao

**** Kết hợp sử dụng các mệnh đề WHERE***

Trong tất cả các mệnh đề WHERE được giới thiệu trong các phần trước, lọc dữ liệu bằng cách sử dụng một vùng điều kiện đơn (Single criteria). Đối với một mức độ điều khiển lọc lớn hơn, SQL cho phép bạn chỉ định nhiều mệnh đề WHERE. Các mệnh đề này phải được sử dụng theo hai cách: Ở dạng các mệnh đề AND hoặc ở dạng các mệnh đề ORDER BY.

**** Sử dụng toán tử AND***

Để lọc bởi nhiều cột, bạn sử dụng toán tử AND để thêm các điều kiện vào mệnh đề WHERE. Đoạn mã sau đây thể hiện điều này:

```
SELECT prod_id, prod_price, prod_name ;
FROM Products ;
WHERE vend_id = 'DLL01' AND prod_price <=4;
```

- Câu lệnh SQL trên truy vấn tên và giá sản phẩm cho tất cả các sản phẩm được chế tạo bởi nhà sản xuất DLL01 miễn là giá 4\$ hoặc nhỏ hơn. Mệnh đề WHERE trong câu lệnh SELECT được tạo thành từ hai điều kiện, và từ khóa AND được sử dụng để kết nhập chúng. AND chỉ thị phần mềm DBMS trả về chỉ các hàng vốn thỏa mã tất cả các điều kiện được chỉ định. Nếu một sản phẩm được tạo thành từ nhà sản xuất DLL01, nhưng giá lớn hơn 4\$, thì nó sẽ không được đưa ra. Tương tự, các sản phẩm có giá nhỏ hơn 4\$ mà được tạo thành từ một nhà sản xuất khác với nhà sản xuất được chỉ định cũng không được đưa ra. Kết quả câu lệnh SQL này như sau:

Kết quả

Prod_id	prod_price	prod_name
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy

** Sử dụng toán tử OR*

Toán tử ngược lại với AND. Toán tử AND chỉ thị phần mềm DBMS truy vấn các hàng phù hợp với một trong hai điều kiện. Thật ra, hầu hết các DBMS tốt hơn thậm chí sẽ không xét đến điều kiện thứ hai nếu điều kiện thứ nhất thỏa mãn (nếu điều kiện thứ nhất thỏa mãn, dòng sẽ được đưa ra không cần quan tâm đến điều kiện thứ hai).

Hãy xem ví dụ sau:

```
SELECT prod_name, prod_price ;
```

FROM Products ;

WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';

- Câu lệnh SQL trên truy vấn tên và giá sản phẩm cho bất kỳ các sản phẩm được chế tạo bởi một trong hai nhà sản xuất được chỉ định. Toán tử OR chỉ thị DBMS tìm các giá trị thỏa mãn một trong hai điều kiện. Nếu toán tử AND được sử dụng ở đây thì không có dữ liệu nào được trả về cả. Kết quả của câu lệnh SQL như sau:

Kết quả

prod_name	prod_price
-----	-----
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900
8 inch teddy bear	5.9900
12 inch teddy bear	8.9900
18 inch teddy bear	11.9900
Raggedy Ann	4.9900

** Tìm hiểu thứ tự xét*

Các mệnh đề WHERE có thể chứa bất kỳ số toán tử AND và OR. Kết hợp hai toán tử cho phép bạn thực hiện công việc lọc tinh vi và phức tạp.

Nhưng việc kết hợp hai toán tử này sẽ đưa ra một vấn đề cần quan tâm. Để giải thích điều này, hãy xem qua một ví dụ. Bạn cần một danh sách tất cả các sản phẩm có giá 10\$ hoặc nhiều hơn được chế tạo bởi các nhà sản xuất DLL01 và BRS01. Câu lệnh SELECT sau sử dụng một sự kết hợp giữa hai toán tử này để xây dựng một mệnh đề WHERE:


```
SELECT prod_name, prod_price ;
FROM Products ;
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01' ;
AND prod_price >= 10;
```

Kết quả

Prod_name	prod_price
-----	-----
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900
18 inch teddy bear	11.9900
Raggedy Ann	4.9900

- Hãy xem các kết quả trên: 4 hàng trả về các giá nhỏ hơn 10\$ - vì vậy, rõ ràng, các dòng này không được lọc như dự định. Tại sao lại xảy ra như vậy? Câu trả lời là do thứ tự xét. SQL (giống như hầu hết các ngôn ngữ) xử lý các toán tử AND trước toán tử OR. Khi SQL xem mệnh đề WHERE ở trên, nó đọc bất kỳ sản phẩm nào có giá trị 10\$ hoặc nhiều hơn được chế tạo từ nhà sản xuất BRS01, và bất cứ sản phẩm nào được chế tạo bởi nhà sản xuất DLL01 không quan tâm tới giá. Nói cách khác, cấp bậc của AND cao hơn trong thứ tự xét.

- Giải pháp cho vấn đề này là sử dụng các dấu ngoặc () để nhóm các toán tử có liên quan lại một cách rõ ràng. Xem qua câu lệnh SELECT và kết xuất như sau:

```
SELECT prod_name, prod_price ;
FROM Products ;
WHERE (vend_id = "DLL01" OR vend_id = 'BRS01') ;
AND prod_price >= 10;
```

Kết quả

Prod_name	prod_price
-----	-----
18 inch teddy bear	11.9900

- Sự khác biệt duy nhất giữa câu lệnh SELECT này và câu lệnh SELECT trước là, trong câu lệnh này, hai điều kiện mệnh đề WHERE đầu tiên được đặt trong cặp dấu ngoặc đơn (). Cặp dấu này có thứ tự xét cao hơn các toán tử AND hoặc OR. DBMS đầu tiên sẽ lọc điều kiện OR trong các dấu ngoặc đơn đó. Khi đó, câu lệnh SQL trở thành: bất kỳ các sản phẩm nào được chế tạo bởi nhà sản xuất DLL01 hay BRS01 có giá 10\$ hoặc lớn hơn.

- Sử dụng các dấu ngoặc đơn trong mệnh đề WHERE. Bất cứ lúc nào bạn viết các mệnh đề WHERE vốn sử dụng cả hai toán tử AND và OR, hãy sử dụng các dấu ngoặc đơn để nhóm toán tử lại rõ ràng. Đừng nên phụ thuộc vào thứ tự xét mặc định, cho dù nó chính xác. Không có điểm bất lợi nào khi sử dụng các dấu ngoặc đơn, và bạn luôn loại trừ được các tình trạng mù mờ.

** Sử dụng toán tử IN*

Toán tử IN sử dụng để chỉ định một dãy điều kiện, bất kỳ giá trị nào trong dãy thoả mãn. IN sử dụng một danh sách các giá trị được tách bởi dấu phẩy, tất cả được đặt trong các dấu ngoặc đơn. Ví dụ sau thể hiện điều này:

```
SELECT prod_name, prod_price ;  
FROM Products ;  
WHERE vend_id IN ('DLL01', 'BRS01') ;  
ORDER BY prod_name;
```

Kết quả

Prod_name	prod_price
-----	-----
12 inch teddy bear	8.9900
18 inch teddy bear	11.9900
8 inch teddy bear	5.9900
Bird bean bag toy	3.4900
Fish bean bag toy	3.4900
Rabbit bean bag toy	3.4900
Raggedy Ann	4.9900

- Câu lệnh SELECT truy vấn tất cả các sản phẩm được chế tạo bởi nhà sản xuất DLL01 và Vendor BRS01. Sau toán tử IN là một danh sách các giá trị được tách bởi dấu phẩy. Và toàn bộ danh sách phải được đặt trong dấu ngoặc đơn.

- Nếu bạn đang nghĩ rằng toán tử IN thực hiện cùng mục đích như toán tử OR, bạn đúng đấy. Câu lệnh SQL sau thực hiện một điều tương tự chính xác như ví dụ trên:

```
SELECT prod_name, prod_price ;  
FROM Products ;  
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01' ;  
ORDER BY prod_name;
```

Kết quả

Prod_name	prod_price
-----	-----
12 inch teddy bear	8.9900
18 inch teddy bear	11.9900
8 inch teddy bear	5.9900

Bird bean bag toy	3.4900
Fish bean bag toy	3.4900
Rabbit bean bag toy	3.4900
Raggedy Ann	4.9900

- Tại sao sử dụng toán tử IN? Những thuận lợi của nó là:

+ Khi bạn đang làm việc với danh sách các tùy chọn quá dài, cú pháp toán tử IN lại rất gọn và dễ đọc hơn.

+ Trật tự xét để quản lý hơn khi sử dụng IN (như ít sử dụng toán tử hơn).

+ Các toán tử IN hầu như thực thi nhanh hơn các danh sách các toán tử OR.

+ Thuận lợi lớn nhất của toán tử IN là có thể chứa các câu lệnh SQL khác, cho phép bạn xây dựng một mệnh đề WHERE động.

** Sử dụng toán tử NOT*

Toán tử NOT có duy nhất một chức năng - NOT phủ định lại bất cứ điều kiện nào đứng sau nó. Vì NOT không bao giờ được sử dụng bởi chính bản thân nó (nó luôn được sử dụng kèm với một số toán tử khác), cú pháp của nó hơi khác với tất cả các toán tử khác. Không giống như các toán tử khác NOT có thể đặt trước cột để lọc, không phải chỉ đặt sau cột.

Ví dụ sau thể hiện cách dùng NOT. Để liệt kê tất cả các sản phẩm được chế tạo bởi tất cả các nhà sản xuất ngoại trừ nhà sản xuất DLL01, bạn có thể viết như sau:

```
SELECT prod_name ;
FROM Products ;
WHERE NOT vend_id = 'DLL01' ;
ORDER BY prod_name;
```

Kết quả

prod_name

12 inch teddy bear

18 inch teddy bear

8 inch teddy bear

King doll

Queen doll

- NOT ở đây phủ định điều kiện vốn đứng sau nó, nên thay vì truy vấn vend_id thoả mãn DLL01, thì DBMS sẽ tìm các giá trị phù hợp với bất cứ giá trị nào không phải là DLL01.

Ví dụ trước cũng có thể được thực hiện bằng cách sử dụng toán tử <>, như sau:

```
SELECT prod_name ;
```

```
FROM Products ;
```

```
WHERE vend_id <> 'DLL01' ;
```

```
ORDER BY prod_name;
```

Kết quả

prod_name

12 inch teddy bear

18 inch teddy bear

8 inch teddy bear

King doll

Queen doll

- Tại sao sử dụng NOT? Đối với các mệnh đề đơn giản như các mệnh đề trình bày ở đây, thực sự không có thuận lợi nào cả khi sử

dùng NOT. NOT hữu dụng trong các mệnh đề phức tạp hơn. Ví dụ, sử dụng NOT đi kèm với một toán tử IN giúp đơn giản tìm tất cả các hàng vốn không thoả mãn một danh sách vùng điều kiện.

- NOT trong MySQL dạng NOT được mô tả ở đây không được hỗ trợ trong MySQL. Trong MySQL chỉ được sử dụng để phủ định EXITST (như NOT EXITST).

e. Lọc bằng các ký tự đại diện

**** Sử dụng toán tử LIKE***

Bản thân các ký tự đại diện là các ký tự có ý nghĩa đặc biệt bên trong mệnh đề SQL WHERE và SQL cung cấp các kiểu ký tự đại diện thông thường.

Để sử dụng các ký tự đại diện bên trong các mệnh đề tìm kiếm, toán tử LIKE phải được sử dụng. LIKE chỉ thị DBMS theo mẫu tìm kiếm được so sánh bằng cách sử dụng một ký tự đại diện phù hợp nhưng cách này ít được sử dụng hơn cách trực tiếp.

- Khi nào một toán tử không còn một toán tử? Khi nào nó là một thuộc tính. Về mặt kỹ thuật, LIKE là một thuộc tính chứ không phải là một toán tử. Kết quả cuối cùng là giống nhau, hãy lưu ý thuật ngữ này trong trường hợp này trong cơ sở dữ liệu SQL.

Về việc tìm kiếm ký tự đại diện chỉ có thể được sử dụng với các trường Text (các chuỗi); bạn không thể sử dụng các ký tự đại diện để tìm kiếm các trường không có dữ liệu kiểu Text.

**** Ký tự đại diện %***

Ký tự đại diện được sử dụng thường xuyên nhất là ký tự. Bên trong chuỗi tìm kiếm, dấu % đại diện cho phần còn lại của chuỗi. Ví dụ, để tìm tất cả các sản phẩm bắt đầu với từ Fish, ta có thể thực hiện như sau:

```
SELECT prod_id, prod_name ;
```

FROM Products ;

WHERE prod_name LIKE 'Fish%';

Kết quả

Prod_id	Prod_name
---------	-----------

BNBG01	Fish bean bag toy
--------	-------------------

- Nếu bạn đang sử dụng Microsoft Access, bạn có thể sử dụng dấu * thay thế cho dấu %.

- Phụ thuộc vào DBMS của bạn và cách cấu hình của nó như thế nào, việc tìm kiếm có thể phân biệt kiểu chữ. Trong trường hợp này Fish bean bag toy không thoả mãn "Fish %". Không phù hợp với Fish bean bag toy.

Các ký tự đại diện có thể được sử dụng tại bất kỳ nơi nào trong mẫu tìm kiếm, và nhiều ký tự đại diện cũng có thể được sử dụng. Ví dụ sau sử dụng 2 ký tự đại diện:

SELECT prod_id, prod_name ;

FROM Products ;

WHERE prod_name LIKE '%bean bag%';

Kết quả

Prod_id	prod_name
---------	-----------

BNBG01	Fish bean bag toy
--------	-------------------

BNBG02	Bird bean bag toy
--------	-------------------

BNBG03	Rabbit bean bag toy
--------	---------------------

- Các ký tự đại diện cũng có thể được sử dụng ở giữa mẫu tìm kiếm, tuy nhiên phương pháp này không được sử dụng thường xuyên.

SELECT prod_name ;

FROM Products ;

WHERE prod_name LIKE 'F%y';

Một lưu ý quan trọng khác đó là dấu % cũng có thể thay thế cho ký tự zero. Dấu % đại diện cho zero, cho một hoặc các ký tự tại vị trí xác định bên trong mẫu tìm kiếm.

- Xem xét các khoảng trắng đuôi. Nhiều DBMS, bao gồm Microsoft Access, các khoảng trắng vào phần đuôi. Ví dụ, nếu một cột dài 50 ký tự và Text được lưu trữ là Fish Bean Bag Toy (17 ký tự), 33 khoảng trống có thể được thêm vào Text để điền vào đầy cột đó. Thông thường thì không có sự tác động thực tế nào lên dữ liệu và cách thức nó được sử dụng ra sao, nhưng nó có thể tác động lên câu lệnh SQL trước đó. Mệnh đề *WHERE prod_name LIKE 'F%y'* chỉ phù hợp với Prod name nếu nó bắt đầu bằng chữ F và kết thúc bằng chữ y. Nếu giá trị được điền đầy bằng các khoảng trống, nó không kết thúc bằng chữ y, vậy Fish Bean Bag Toy không truy vấn thỏa mãn. Một giải pháp đơn giản cho vấn đề này là một dấu % thứ hai vào mẫu tìm kiếm: *'F%y%'* cũng phù hợp với các ký tự (hoặc các khoảng trống) sau chữ y. Một giải pháp tối ưu hơn là tách các khoảng trống bằng cách sử dụng các hàm.

** Ký tự đại diện ()*

Một ký tự đại diện hữu dụng khác là ký tự (). Ký tự gạch dưới được sử dụng giống như dấu %, nhưng ký tự đại diện này chỉ đại diện cho một ký tự đơn.

- Microsoft Access. Nếu bạn nên sử dụng Microsoft Access, bạn có thể sử dụng dấu (?) thay cho dấu gạch dưới này _.

Xem ví dụ:

SELECT prod_id, prod_name ;

FROM Products ;

WHERE prod_name LIKE '_ inch teddy bear';

Kết quả

Prod_id	prod_name
BNBG02	12 inch teddy bear
BNBG03	18 inch teddy bear

```
SELECT prod_id, prod_name ;  
WHERE prod_name LIKE '% inch teddy bear';
```

Kết quả

Prod_id	prod_name
BNBG01	8 inch teddy bear
BNBG02	12 inch teddy bear
BNBG03	18 inch teddy bear

Không giống với ký tự %, dấu gạch dưới chỉ đại diện cho một ký tự không thoả mãn ký tự zero.

** Ký tự đại diện ([])*

Ký tự đại diện ([]) được sử dụng để chỉ định một tập hợp các ký tự, một ký tự trong tập phải thoả mãn một ký tự tại một vị trí xác định trước (vị trí của ký tự đại diện).

- Các tập không luôn được hỗ trợ không giống như các ký tự đại diện được mô tả cho đến thời điểm này. Việc sử dụng ([]) để tạo các tập không được hỗ trợ bởi tất cả các DBMS. Hãy tham khảo tài liệu DBMS của bạn để xem có hỗ trợ hay không.

Ví dụ, để tìm toàn bộ các đối tác mà tên của họ bắt đầu bằng chữ cái J hoặc chữ cái M, bạn có thể thực hiện như sau:

```
SELECT cust_contact ;  
FROM Customers ;
```

WHERE Cust_contact LIKE '[JM]%' ;

ORDER BY cust_contact;

Kết quả

cust_contact

Jim Jones

John Smith

Michelle Green

- Mệnh đề WHERE trong câu lệnh này là '[JM]%' . Mẫu tìm kiếm này sử dụng hai ký tự đại diện khác nhau. [JM] phù hợp với bất kỳ tên đối tác nào bắt đầu bằng các chữ cái bên trong dấu ngoặc vuông, và nó cũng chỉ phù hợp với mọi ký tự đơn. Bởi vì các tên bất kỳ dài hơn một ký tự đều không phù hợp. Ký tự đại diện % sau [JM] phù hợp với số lượng các ký tự bất kỳ sau ký tự đầu tiên, trở về kết quả như mong muốn.

Ký tự đại diện này có thể được phủ định bằng cách thêm tiếp dấu ngã ^ (ký tự mũ). Ví dụ, mẫu sau đây phù hợp với bất kỳ tên nào không bắt đầu bằng chữ J hoặc chữ M (ví dụ này đối nghịch với đối nghịch với ví dụ trước đó):

SELECT cust_contact ;

FROM Customers ;

WHERE cust_contact LIKE '[^JM]%' ;

ORDER BY cust_contact;

- Ghi chú: Nếu bạn đang sử dụng Microsoft Access, bạn nên sử dụng ! thay vì ^ để phủ định một tập - có nghĩa là [! JM] thay vì [^ JM].

Tất nhiên, bạn có thể đạt được kết quả tương tự bằng cách sử dụng toán tử NOT. Chỉ một ưu điểm duy nhất của ^ là nó có thể

đơn giản hoá cấu trúc nếu bạn sử dụng nhiều mệnh đề WHERE:

```
SELECT cust_contact ;  
FROM Customers ;  
WHERE NOT cust_contact LIKE '[JM]%' ;  
ORDER BY cust_contact;
```

- Ký tự đại diện ([]) không được hỗ trợ bởi toàn bộ DBMS. Hãy dựa vào tài liệu DBMS của bạn để quyết định xem ký tự đại diện đặc biệt có được hỗ trợ hay không.

** Các thủ thuật trong việc sử dụng các ký tự đại diện*

Như bạn có thể thấy, các ký tự đại diện của SQL rất hữu dụng. Nhưng việc tìm kiếm của chúng mất nhiều thời gian hơn các kiểu tìm kiếm bất kỳ khác đã được thảo luận trước đây. Sau đây là một số thủ thuật cần nhớ khi bạn sử dụng các ký tự đại diện:

+ Không lạm dụng các ký tự đại diện. Hãy thay thế ký tự đại diện bằng toán tử tìm kiếm khác nếu có thể.

+ Khi bạn sử dụng các ký tự đại diện, nếu không thực sự cần thiết bạn không nên sử dụng các ký tự đại diện ở phần đầu của các mẫu tìm kiếm. Các mẫu tìm kiếm bắt đầu bằng các ký tự đại diện là các mẫu tìm kiếm chậm nhất.

+ Bạn phải chú ý đến vị trí của các biểu tượng ký tự đại diện. Nếu chúng được đặt sai vị trí, bạn có thể không được kết quả như mong muốn.

Tuy nhiên, các ký tự đại diện là một công cụ tìm kiếm quan trọng và hữu dụng mà bạn sử dụng thường xuyên.

f. Tạo các trường tính toán

** Tìm hiểu các trường được tính toán*

Dữ liệu được lưu trữ trong các Table cơ sở dữ liệu thường

không có định dạng chính xác mà bạn cần cho các ứng dụng của mình. Sau đây là một số ví dụ:

- + Bạn cần hiển thị một trường có tên của một công ty dọc theo vị trí của công ty đó, nhưng thông tin lưu trữ này được phân mảnh thành nhiều cột riêng lẻ.

- + Mã thành phố, mã nước và mã vùng được lưu trữ trong các cột riêng lẻ, nhưng chương trình in nhãn thư cần truy vấn đến một trường được định dạng chính xác của chúng.

- + Dữ liệu cột được hoà trộn giữa các ký tự thường và ký tự in, nhưng bản báo cáo của toàn bộ dữ liệu kiểu in.

- + Một bản Order Items (các món hàng của đơn đặt hàng) lưu trữ giá và số lượng của món hàng (Item) nhưng không phải là giá mở rộng (giá nhân với số lượng) của nhóm hàng. Để in các đơn hàng bạn cần giá mở rộng này.

- + Bạn cần tổng, trung bình và các phép tính khác đặt trên dữ liệu của mình.

- Trong mỗi ví dụ trên, dữ liệu được lưu trữ trong Table không chính xác với ứng dụng bạn cần. Cách tốt nhất là truy vấn dữ liệu và sau đó định dạng lại nó bên trong báo cáo hoặc ứng dụng Client của bạn, tất cả những gì bạn thực sự muốn là truy vấn dữ liệu đã chuyển đổi, dữ liệu được tính toán hoặc dữ liệu được định dạng lại từ cơ sở dữ liệu.

- Không giống với toàn bộ các cột chúng ta đã truy vấn trong các bài học trước, các trường tính toán không thực sự tồn tại trong các bảng cơ sở dữ liệu. Đôi khi trường tính toán được tạo bên trong một câu lệnh SELECT của SQL.

- Một lưu ý quan trọng là chỉ có cơ sở dữ liệu mới biết được các cột nào trong một câu lệnh SELECT là các cột Table thực sự và nào là các trường tính toán từ phối cảnh của một Client (ví dụ, ứng

dụng của bạn), dữ liệu của trường được tính toán được trả về - Client với Server Formatting: Có rất nhiều sự chuyển đổi và định dạng lại vừa được thực hiện bên trong các câu lệnh SQL vừa được thực hiện trực tiếp vào ứng dụng Client của bạn. Tuy nhiên, như là một quy luật, việc thực hiện các thao tác này Server cơ sở dữ liệu sẽ nhanh chóng hơn khi nó được thực hiện bên trong Client bởi vì các DBMS được xây dựng để thực hiện tiến trình loại này nhanh hơn và hiệu quả hơn.

** Ràng buộc các trường*

- Để hiểu rõ quá trình làm việc với các trường được tính toán, chúng ta hãy bắt đầu bằng một ví dụ đơn giản bằng cách tạo một tiêu đề được tạo thành từ hai cột.

- Table Vendor (nhà sản xuất) chứa thông tin về tên và địa chỉ Vendor. Hãy tưởng tượng bạn đang thực hiện một Table báo cáo Vendor và cần liệt kê vị trí Vendor tại phần tên Vendor trong định dạng name (Location).

Bảng báo cáo cần một giá trị đơn, và dữ liệu trong bảng được lưu trữ ở hai cột: Vend name và vend country. Hơn nữa, bạn cần bao xung quanh Vend country bằng một dấu ngoặc đơn, và chúng không được lưu trữ trong bảng cơ sở dữ liệu. Câu lệnh SELECT trả về các tên và các vị trí Vendor đầy đủ, nhưng bạn làm cách nào để tạo giá trị được kết hợp này?

Giải pháp ràng buộc hai cột. Trong các câu lệnh SELECT của SQL, bạn có thể ràng buộc các cột bằng cách sử dụng một toán tử đặc biệt. Phụ thuộc vào DBMS mà bạn đang sử dụng, toán tử này có thể là dấu (+) hoặc dấu (||).

- + hoặc || ? Cả Access, SQL Server lẫn Sybase đều cung cấp + cho ràng buộc. DB2, Oracle, PostgreSQL, và Sybase cung cấp ||. Dựa vào tài liệu DBMS của bạn để có thêm nhiều chi tiết.

|| thực sự là cú pháp ưu tiên, vì thế càng ngày càng nhiều DBMS đang triển khai hỗ trợ cho nó.

- Đây là một ví dụ sử dụng dấu + (cú pháp được sử dụng bởi hầu hết các DBMS):

```
SELECT cust_contact ;  
FROM Customers ;  
WHERE NOT cust_contact LIKE '[JM]%' ;  
ORDER BY cust_contact ;
```

Kết quả

-----	-----
Bear Emporium	(USA)
Bears R Us	(USA)
Doll House Inc.	USA)
Fun and Games	(England)
Furball Inc.	(USA)
Jouets et ours	(France)

Sau đây là cùng lệnh đó, nhưng cú pháp ||:

```
SELECT vend_name || '(' || vend_country || ') ' ;  
FROM Vendors ;  
ORDER BY vend_name ;
```

Kết quả

-----	-----
Bear Emporium	(USA)
Bears R Us	(USA)
Doll House Inc.	(USA)
Fun and Games	(England)
Furball Inc.	(USA)
Jouets et ours	(France)

- Các câu lệnh SELECT trước đó nối các thành phần sau:
 - + Tên được lưu trữ trong cột Vend name
 - + Một chuỗi chứa một khoảng trắng và một dấu ngoặc đơn mở.
 - + Đang được lưu trữ trong cột Vend country.
 - + Một chuỗi chứa dấu ngoặc đơn đóng.
- Như bạn có thể thấy trong kết quả xuất được trình bày trước đây, câu lệnh SELECT trả về một cột đơn (một cột đã được tính toán) chứa tất cả bốn thành phần này dưới dạng một đơn vị.
- Hoạt động nối trong MySQL, MySQL không hỗ trợ hoạt động nối bằng cách sử dụng + hoặc là ||. Đúng hơn, nó đòi hỏi phải sử dụng một hàm CONCAT (), dòng ví dụ đầu tiên sẽ là như sau:

```
SELECT CONCAT (vend_name, ' (' , vend_country, ')'
```

MySQL hỗ trợ sử dụng ||, nhưng không dùng để nối. Trong MySQL || tương đương với toán tử OR và & tương đương với toán tử AND.

Hãy xem qua một lần nữa dữ liệu xuất được trả về bằng câu lệnh SQL. Hai cột được xác nhập thành trường được tính được thêm vào các khoảng trắng. Nhiều cơ sở dữ liệu (mặc dù không phải là tất cả) lưu các giá trị Text được độn vào trong khoảng trắng cho đủ bề ngang của cột. Để trả về dữ liệu được định dạng hợp lý, bạn phải cách ly những khoảng trắng được thêm vào. Điều này có thể thực hiện bằng cách sử dụng hàm RTRIM () như sau:

```
SELECT RTRIM (vend_name) + ' (' + ;  
RTRIM (vend_country) + ')';  
FROM Vendors ;  
ORDER BY vend_name;
```

Kết quả

Bear Emporium USA)

Bears R Us (USA)

Doll House Inc. USA)

Fun and Games (England)

Furball Inc. (USA)

Jouets et ours (France)

- Hàm RTRIM cắt tất cả ô trống từ bên phải của một giá trị. Bằng cách sử dụng RTRIM (), các cột riêng lẻ tất cả được cắt thích hợp, một dấu phẩy và khoảng trắng tách biệt thành phố và bang, khoảng trắng tách biệt bang và mã ZIP (ZIP code).

- Các hàm cắt xén (RTRIM). Hầu hết các DBMS hỗ trợ hàm RTRIM () (hàm như bạn vừa thấy cắt từ phía bên phải của một chuỗi), cũng như LTRIM (cắt từ phía bên trái của chuỗi) và TRIM () (hàm cắt cả hai bên trái và phải).

** Sử dụng các bí danh (Aliase)*

Câu lệnh SELECT dùng để nối các trường địa chỉ làm hoạt động tốt, như bạn thấy trong kết xuất trước. Nhưng tên của cột được tính mới này là gì? Nó không có tên, nó đơn giản là một giá trị. Mặc dù điều này, có thể là tốt nếu bạn chỉ xem các kết quả trong một công cụ Query SQL, một cột không được đặt tên có thể được sử dụng trong một ứng dụng Client, bởi vì Client không có cách nào để tham chiếu đến cột đó. Để giải quyết vấn đề này, SQL cung cấp bí danh của cột (Alias). Một bí danh chỉ là một tên thay thế cho một trường hoặc một giá trị. Các bí danh được gán với từ khoá AS. Xem qua câu lệnh SELECT sau:

```
SELECT RTRIM (vend_name) + ' (' + ;
```



```

RTRIM (vend_country) + ')' AS vend_title ;
FROM Vendors ;
ORDER BY vend_name;

```

Kết quả

Vend_title

```

-----
Bear Emporium (USA)
Bears R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)

```

Sau đây là cùng câu lệnh nhưng sử dụng cú pháp ||:

```

SELECT RTRIM (vend_name) || ' (' RTRIM (vend_country);
|| ')' AS vend_title ;
FROM Vendors ;
ORDER BY vend_name;

```

Kết quả

Vend_title

```

-----
Bear Emporium (USA)
Bears R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)

```

- Lệnh SELECT bản thân nó cũng giống câu lệnh được sử dụng trong đoạn mã trước ngoại trừ ở đây các trường được tính toán thêm Text AS vend_title theo sau. Điều này chỉ thị SQL tạo một trường tính toán được đặt tên là vend_title chứa các phép tính được chỉ định. Như bạn thấy trong kết quả xuất, các kết quả giống như trước đó. Nhưng cột bây giờ đã có tên vend_title và ứng dụng Client bất kỳ có thể tham chiếu đến cột này, không kém bất kỳ cột Table thực nào cả.

- Các công dụng khác cho các bí danh: Các bí danh cũng có các công cụ khác. Một số công dụng bao gồm việc đặt tên lại một cột nếu tên một cột Table thực chứa các giá trị không hợp lệ (ví dụ, các khoảng trắng) và mở rộng các tên cột nếu các tên gốc tối nghĩa hoặc dễ đọc nhầm.

- Các tên bí danh: Các tên bí danh có thể là các từ đơn hoặc hoàn toàn là chuỗi. Nếu là chuỗi thì chuỗi nên được đánh kèm trong dấu ngoặc kép. Phần thực hành này hợp lệ nhưng không nên dùng. Mặc dù tên có nhiều từ thực sự dễ đọc, nhưng chúng có thể gây ra nhiều loại sự cố đối với các ứng dụng Client. Vì thế, để một trong các việc sử dụng thông thường nhất của các bí danh là đặt tên lại bằng các tên từ đơn (như được giải thích trước đó).

- Derived Column: Các bí danh (Alias) thỉnh thoảng còn được gọi là các Derived column, vì vậy không quan tâm đến từ nào mà bạn ngẫu nhiên gặp, chúng đều có nghĩa như nhau.

** Thực hiện các phép tính khoa học*

Một công cụ thường xuyên khác đối với các trường được tính toán là thực hiện các phép tính toán học trên dữ liệu được truy vấn. Hãy xem qua một ví dụ. Table Orders chứa tất cả các đơn đặt hàng nhận được, và Table OrderItems chứa các món hàng riêng lẻ trong mỗi đơn đặt hàng. Câu lệnh SQL sau truy vấn tất cả các món hàng trong đơn đặt hàng 20008:

```
SELECT prod_id, Quantity, item_price;
FROM OrderItems;
WHERE order_num = 20008;
```

Kết quả

Prod_id	Quantity	item_price
RGAN01	5	4.9900
BR03	5	11.9900
BNBG01	10	3.4900
BNBG02	10	3.4900
BNBG03	10	3.4900

Cột item_price chứa giá trên mỗi đơn vị cho mỗi món hàng trong một đơn đặt hàng. Để mở rộng giá món hàng (giá món hàng * số lượng đặt hàng), bạn đơn giản thực hiện như sau:

```
SELECT prod_id, Quantity, Item_price, ;
Quantity*item_price AS expanded_price ;
FROM OrderItems ;
WHERE order_num = 20008;
```

Kết quả

Prod_id	Quantity	item_price	expanded_price
RGAN01	5	4.9900	24.9500
BR03 5	11.	9900	59.9500
BNBG01	10	3.4900	34.9000
BNBG02	10	3.4900	34.9000
BNBG03	10	3.4900	34.9000

- Cột expanded_price được trình bày trong kết quả xuất trước là một trường được tính toán, phép tính đơn giản là Quantity*item_price

(số lượng * đơn giá). Ứng dụng Client bây giờ có thể sử dụng cột được tính mới này không kém bất kỳ cột nào.

- SQL hỗ trợ các toán tử cơ bản. Ngoài ra, sử dụng các dấu ngoặc đơn để thiết lập thứ tự ưu tiên.

Các phép toán SQL

Toán tử Mô tả

+ Cộng thêm vào

- Loại trừ ra

* Nhân

/ Chia

g. Góm nhóm dữ liệu

** Tìm hiểu việc nhóm dữ liệu*

Trong bài học trước, bạn đã học rằng các hàm tập hợp SQL có thể sử dụng để tổng kết dữ liệu, điều này cho phép bạn đến các dòng, tính tổng và tính trung bình, và lấy các giá trị lớn nhất và nhỏ nhất mà không cần phải truy vấn tất cả các dữ liệu.

- Tất cả các phép toán như thế được thực hiện trên tất cả các dữ liệu trong một Table hoặc trên dữ liệu vốn thoả mãn một mệnh đề WHERE cụ thể. Như là người nhắc nhở, ví dụ sau sẽ trả về số các sản phẩm được đưa ra bởi nhà sản xuất DKK01:

```
SELECT COUNT (*) AS num_prods ;
```

```
FROM Products ;
```

```
WHERE vend_id = 'DLL01';
```

Kết quả

Num_prods

- Nhưng điều gì xảy ra nếu bạn muốn trả về số sản phẩm được đưa ra bởi mỗi nhà sản xuất? Hoặc tất cả các sản phẩm được đưa ra

bởi các nhà sản xuất đưa ra một sản phẩm đơn, hoặc chỉ những nhà sản xuất đưa ra hơn mười sản phẩm?

- Điều này cần đến việc nhóm dữ liệu, việc nhóm dữ liệu sẽ cho phép bạn phân dữ liệu thành các tập con Logical để bạn có thể thực hiện các hàm tập hợp trên mỗi nhóm.

** Tạo các nhóm*

- Các nhóm được tạo bằng cách sử dụng mệnh đề GROUP BY trong câu lệnh SELECT của bạn. Cách tốt nhất để tìm hiểu điều này là xem qua một ví dụ:

```
SELECT vend_id, COUNT (*) AS num_prods ;  
FROM Products ;  
GROUP BY vend_id;
```

Kết quả

Vend_id	num_prods
-----	-----
BRS01	3
DLL01	4
FNG01	2

- Câu lệnh SQL trên chỉ thị hai cột, Vend_id, cột chứa ID của một nhà sản xuất của sản phẩm, Num_prods, cột là trường đã được tính trước [được tạo bằng cách sử dụng hàm COUNT (*)]. Mệnh đề GROUP BY chỉ thị DBMS sắp xếp dữ liệu và nhóm nó theo cột ID. Điều này có thể gây cho cột Num_prods được tính một lần trên mỗi cột Vend_id, tốt hơn là tính một lần trên toàn bộ Table. Như bạn có thể thấy trong kết quả xuất, nhà sản xuất BRS01 có ba sản phẩm được liệt kê, nhà sản xuất DLL01 có bốn sản phẩm được liệt kê, và nhà sản xuất FNG01 có hai sản phẩm được liệt kê.

- Bởi vì bạn đã sử dụng GROUP BY, bạn không phải chỉ định

mỗi nhóm để xét và tính. Từ đó được thực hiện tự động. Mệnh đề GROUP BY chỉ thị DBMS nhóm dữ liệu và sau đó thực hiện tập hợp trên mỗi nhóm tốt hơn là trên toàn bộ tập kết quả.

- Trước khi bạn sử dụng GROUP BY hãy tuân theo một số quy tắc quan trọng về việc sử dụng mà bạn cần biết:

- Các mệnh đề GROUP BY có thể chứa số các cột như bạn muốn. Điều này cho phép bạn xếp lồng các nhóm, cung cấp cho bạn nhiều dạng điều khiển qua cách dữ liệu được gom nhóm.

- Nếu bạn có các nhóm xếp lồng vào nhau trong mệnh đề GROUP BY của bạn, dữ liệu được tổng kết ở các nhóm được chỉ định cuối cùng. Nói cách khác tất cả nhóm được đưa ra ở đây được xét cùng với nhau khi việc gom nhóm được thiết lập (vì thế bạn sẽ không nhận dữ liệu trở lại cho mỗi cấp độ cột riêng rẽ).

- Mỗi cột được liệt kê trong GROUP BY phải là một cột được truy lục hoặc là một biểu thức hợp lệ (nhưng không phải là một tập hợp). Nếu một biểu thức được sử dụng trong SELECT có cùng biểu thức đó phải được chỉ định trong GROUP BY. Các bí danh không thể sử dụng ở đây. Hầu hết các trình triển khai SQL không cho phép GROUP BY các cột với kiểu dữ liệu có chiều dài thay đổi (chẳng hạn như Text).

- Ngoài các câu lệnh của phép toán tập hợp, mỗi cột trong câu lệnh SELECT phải có mặt trong mệnh đề GROUP BY.

- Nếu việc nhóm cột thiếu một dòng với một giá trị NULL, NULL sẽ được trả về dưới dạng một nhóm. Nếu có nhiều dòng với các giá trị NULL tất cả chúng sẽ được gom nhóm với nhau.

- Mệnh đề GROUP BY phải đứng sau mệnh đề WHERE bất kỳ và đứng trước mệnh đề ORDER BY bất kỳ.

- Mệnh đề ALL một số triển khai SQL (chẳng hạn như Microsoft SQL Server) hỗ trợ một mệnh đề ALL tùy ý trong

GROUP BY. Mệnh đề này có thể được sử dụng để trả về tất cả các nhóm, thậm chí các nhóm đó không có các dòng nào thỏa mãn (trong trường hợp này tập hợp sẽ trả về NULL). Tham khảo tài liệu DBMS của bạn để xem nếu nó hỗ trợ mệnh đề ALL.

- Chỉ định các cột theo vị trí. Một số trình triển khai SQL cho phép bạn chỉ định các cột GROUP BY theo vị trí trong danh sách SELECT. Ví dụ, GROUP BY 2,1 có thể có nghĩa là nhóm theo cột 2 được chọn và sau đó theo cột đầu tiên. Mặc dù cú pháp dạng ngắn này rất thuận tiện, nhưng nó không được hỗ trợ bởi tất cả các trình triển khai SQL. Việc sử dụng của nó cũng hơi mạo hiểm và dễ dẫn đến các lỗi khi hiệu chỉnh các câu lệnh SQL.

** Lọc các nhóm*

Để có thể nhóm dữ liệu bằng cách sử dụng GROUP BY, SQL cũng cho phép bạn lọc các nhóm nào được đưa vào và các nhóm nào loại ra. Ví dụ, bạn có thể muốn một danh sách tất cả các khách hàng mà được tạo ở tối thiểu hai đơn đặt hàng. Dữ liệu lần này bạn phải lọc dựa trên các nhóm đã hoàn thành, không phải dựa vào các dòng riêng lẻ.

- Bạn đã xem cách hoạt động của mệnh đề WHERE (được giới thiệu trong bài 4 "Lọc dữ liệu") nhưng mệnh đề WHERE ở đây không hoạt động bởi vì WHERE chỉ lọc các dòng cụ thể không phải là các nhóm. Thực tế là, WHERE không biết một nhóm là gì.

- Vì thế bạn sử dụng cái gì để thay thế cho WHERE? SQL còn cung cấp một mệnh đề khác cho mục đích này. Mệnh đề HAVING, HAVING là rất giống với WHERE. Thực ra, tất cả các loại mệnh đề WHERE mà bạn đã học như thế có thể được sử dụng với HAVING. Cái khác biệt duy nhất là WHERE lọc các dòng và HAVING lập các nhóm.

- HAVING hỗ trợ tất cả các toán tử của WHERE trong bài học

4 & 5 "Lọc dữ liệu nâng" bạn đã học về các điều kiện mệnh đề WHERE (bao gồm các điều kiện ký tự đại diện và các mệnh đề với nhiều toán tử). Tất cả kỹ thuật và các tùy chọn mà bạn đã học về WHERE có thể được áp dụng cho HAVING, cú pháp giống nhau, chỉ có từ khoá là khác nhau.

- Vì thế bạn lọc các dòng như thế nào? Xem qua ví dụ sau:

```
SELECT cust_id, COUNT (*) AS orders ;
```

```
FROM Orders ;
```

```
GROUP BY cusst_id ;
```

```
HAVING COUNT (*) >= 2;
```

Kết quả

Cust_id	orders
-----	-----
1000000001	2

- Ba dòng đầu tiên của câu lệnh SELECT giống với các lệnh được thấy ở trên. Dòng cuối cùng thêm một mệnh đề HAVING lọc trên các nhóm đó với COUNT (*) >=2 - hai hoặc nhiều đơn đặt hàng.

- Như bạn có thể thấy, một mệnh đề WHERE không động ở đây bởi vì việc lọc được dựa trên các giá trị tập hợp nhóm, không phải dựa vào các giá trị của các dòng cụ thể.

- Sự khác biệt giữa HAVING và WHERE ở đây có một cách để phân biệt: WHERE lọc trước khi dữ liệu được nhóm, và HAVING lọc sau khi dữ liệu được nhóm. Đây là một sự phân biệt quan trọng, các dòng vốn được loại ra bởi một mệnh WHERE sẽ không được đưa vào trong nhóm này. Điều này có thể thay đổi các giá trị được tính mà lần lượt có thể ảnh hưởng đến các nhóm nào được lọc dựa vào việc sử dụng các giá trị đó trong mệnh đề HAVING.

- Vì thế có bao giờ cần sử dụng cả hai mệnh đề WHERE và

HAVING trong một câu lệnh? Thực sự có. Tin rằng bạn sẽ muốn lọc nhiều hơn nữa trong câu lệnh trên để nó sẽ trả về bất kỳ các khách hàng nào được đặt trong hai đơn đặt hàng trong 12 tháng vừa qua. Để làm điều đó, bạn có thể thêm một mệnh đề WHERE vốn lọc ra chỉ các đơn đặt hàng được đặt trong 12 tháng vừa qua. Khi đó bạn sẽ thêm một mệnh đề HAVING để lọc chỉ các nhóm với hai hoặc nhiều dòng trong chúng. Để thể hiện điều này tốt hơn, xem qua ví dụ sau vốn liệt kê tất cả các nhà sản xuất có hai hoặc nhiều sản phẩm có giá trị lớn hơn hoặc bằng 4.

```
SELECT vend_id, COUNT (*) AS num_prods ;
FROM Products ;
WHERE prod_price >=4 ;
GROUP BY vend_id ;
HAVING COUNT (*) >= 2;
```

Kết quả

Vend_id	num_prods
-----	-----
BRS01	3
FNG01	2

- Dòng đầu tiên là một câu lệnh SELECT. Căn bản sử dụng một hàm tập hợp. Mệnh đề WHERE lọc tất cả dòng với cột Prod_price tối thiểu bằng 4. Dữ liệu khi đó được nhóm lại theo cột Vend_id và sau đó một mệnh đề HAVING lọc chỉ những nhóm này với một số lượng hai hoặc nhiều hơn. Nếu không có mệnh đề WHERE, một dòng phụ sẽ được truy vấn (nhà sản xuất DLL01 bán 4 sản phẩm tất cả có giá dưới 4USD) như được thấy dưới đây:

```
SELECT vend_id, COUNT (*) AS num_prods ;
FROM Products ;
```

GROUP BY vend_id ;
HAVING COUNT () >= 2;*

Kết quả

Vend_id	num_prods
-----	-----
BRS01	3
DLL01	4
FNG01	2

- Việc sử dụng HAVING và WHERE HAVING tương tự với WHERE mà hầu hết các DBMS xem chúng như cùng một thứ nếu không có GROUP BY được chỉ định. Hãy sử dụng HAVING chỉ khi đi chung với GROUP BY. Hãy sử dụng WHERE cho các hoạt động cấp dòng chuẩn.

** Nhóm và sắp xếp*

Điều quan trọng phải hiểu rằng GROUP BY và ORDER BY là khác nhau, mặc dù chúng thường thực hiện cùng một vấn đề. Bảng sau tóm tắt các sự khác biệt giữa chúng.

ORDER BY với GROUP BY

ORDER BY	GROUP BY
Sắp xếp các dữ liệu xuất được phát sinh.	Nhóm các dòng. Dữ liệu xuất có thể không được theo thứ tự nhóm
Bất kỳ các cột nào (thậm chí các cột không được chọn) có thể được sử dụng.	Bắt buộc nếu sử dụng các cột (hoặc các biểu thức) với các hàm tập hợp.

Sự khác biệt đầu tiên được liệt kê ở bảng vô cùng quan trọng. Thường bạn sẽ nhận thấy rằng dữ liệu được nhóm bằng cách sử dụng GROUP BY sẽ thực sự là cách xuất theo thứ tự nhóm. Nhưng không luôn đúng như thế, và nó thực sự đòi hỏi phải được chỉ định. Hơn nữa, mặc dù DBMS đặc biệt luôn sắp xếp dữ liệu theo mệnh

đề GROUP BY được chỉ định, bạn có thể thực sự muốn nó được sắp xếp theo một cách khác. Chỉ vì bạn nhóm dữ liệu theo một cách (để nhận được các giá trị tập hợp cụ thể của nhóm) không có nghĩa là bạn cũng muốn cách xuất được sắp xếp theo cùng cách đó. Bạn cũng nên cung cấp mệnh đề ORDER BY rõ ràng, mặc dù nó giống như mệnh đề GROUP BY.

- Đừng quên mệnh đề ORDER BY. Theo quy luật, bất cứ khi nào bạn sử dụng mệnh đề GROUP BY bạn cũng nên chỉ định một mệnh đề ORDER BY. Đây chỉ là cách để bảo đảm dữ liệu sẽ được sắp xếp thích hợp. Đừng bao giờ dựa vào GROUP BY để sắp xếp dữ liệu của bạn.

- Để thể hiện việc sử dụng cả hai mệnh đề GROUP BY và ORDER BY, hãy xem qua một ví dụ. Câu lệnh SELECT sau giống như các câu lệnh đã xem qua trước đó. Nó sẽ truy vấn số các đơn đặt hàng và số các món hàng được đặt hàng cho tất cả các đơn đặt hàng chứa ba hay nhiều món hàng:

```
SELECT order_num, COUNT (*) AS items ;
```

```
FROM OrderItems ;
```

```
GROUP BY order_num ;
```

```
HAVING COUNT (*) >= 3;
```

Kết quả

Order_num	items
-----	-----
20006	3
20007	5
20008	5
20009	3

- Để sắp xếp cách xuất theo số các món hàng được đặt hàng, tất

cả các bạn cần làm là thêm một mệnh đề ORDER BY, như sau:

```
SELECT order_num, COUNT (*) AS items ;  
FROM OrderItems ;  
GROUP BY order_num ;  
HAVING COUNT (*) >= 3 ;  
ORDER BY items, order_num;
```

- Sự không tương thích trong Access. Microsoft Access không cho phép bạn sắp xếp theo bí danh (Alias), và vì thế ví dụ này sẽ bị thất bại. Giải pháp để thay thế Items (trong mệnh đề ORDER BY) với phép toán thực sự hoặc với vị trí trường. Thông thường sử dụng ORDER BY COUNT (*); order_num hoặc ORDER BY 1.

Kết quả

Order_num	items
-----	-----
20006	3
20009	3
20007	5
20008	5

- Trong ví dụ này mệnh đề GROUP BY được sử dụng để nhóm dữ liệu theo thứ tự, theo số đơn đặt hàng (cột Order_num) để hàm COUNT (*) có thể trả về số món hàng trong mỗi đơn đặt hàng. Mệnh đề HAVING lọc các dữ liệu để chỉ các đơn hàng mà có ba hoặc nhiều món hàng được trả về.

** Sắp thứ tự mệnh đề SELECT*

Đây có lẽ là thời điểm tốt để ôn lại thứ tự mà các mệnh đề của câu lệnh SELECT được chỉ định. Liệt kê tất cả các mệnh đề mà chúng ta đã học cho đến bây giờ, chúng phải được sử dụng theo thứ tự này.

Các mệnh đề SELECT và thứ tự của chúng.

MỆNH ĐỀ	MÔ TẢ	BẮT BUỘC
SELECT	Các cột hoặc các biểu thức được trả về	Có
FROM	Table để truy lục dữ liệu	Chỉ nếu chọn dữ liệu từ một Table.
WHERE	Lọc cấp dòng	Không
GROUP BY	Hoạt động chỉ định nhóm	Chỉ nếu tính toán các tập hợp hợp bởi các nhóm
HAVING	Lọc các nhóm	Không
ORDER BY	Thứ tự sắp xếp dữ liệu xuất	Không

h. Tìm hiểu Query con

- Các câu lệnh SELECT là các query SQL (truy vấn). Tất cả các câu lệnh SELECT mà chúng ta thấy cho đến thời điểm này là các câu Query đơn giản: các câu lệnh truy vấn dữ liệu từ các Table cơ sở dữ liệu riêng lẻ.

- SQL cũng cho phép bạn tạo các Subqueries: Các Queries vốn được nhúng vào các Queries khác.

** Lọc theo các Query con*

Các Table cơ sở dữ liệu được sử dụng trong tất cả bài học trong sách này là các Table quan hệ.

Các đơn đặt hàng được lưu trữ trong hai Table. Table Orders lưu trữ một dòng đơn cho mỗi đơn đặt hàng chứa mã đơn đặt hàng, ID khách hàng, và ngày đặt hàng. Các món hàng trong đơn đặt hàng đơn lẻ được lưu trữ trong Table có quan hệ Order items. Table Orders không lưu trữ thông tin khách hàng. Nó chỉ lưu trữ một ID khách hàng. Thông tin khách hàng thực sự được lưu trong bảng Customers.

Bây giờ, cho rằng muốn có một danh sách tất cả khách hàng đã đặt món hàng RGAN01. Bạn phải làm điều gì để truy vấn các thông tin này. Đây là các bước thực hiện:

- Truy vấn các mã số đơn đặt hàng của tất cả các đơn đặt hàng chứa món hàng RGAN01.

- Truy vấn số ID khách hàng của tất cả các khách hàng có các đơn đặt hàng được liệt kê trong các mã số đơn đặt hàng được trả về trong bước trước.

- Truy vấn các thông tin khách hàng trong tất cả các ID của khách hàng được trả về trong bước trước.

Mỗi bước này có thể được thực thi theo một Query riêng biệt. Bằng cách thực hiện điều này bạn sẽ sử dụng các kết quả được trả về bởi câu lệnh SELECT được đặt ở mệnh đề WHERE của câu lệnh SELECT kế tiếp.

Bạn cũng có thể sử dụng các Query con để kết hợp tất cả ba câu Query này vào một câu lệnh đơn.

Câu lệnh SELECT đầu tiên nên được làm sáng tỏ lúc này. Nó truy vấn cột Order_num trong tất cả các món hàng của đơn đặt hàng với Prod_id là RGAN01. Kết quả xuất liệt kê hai đơn đặt hàng chứa món hàng này.

```
SELECT order_num ;  
FROM OrderItems ;  
WHERE Prod_id = 'RGAN01';
```

Kết quả

Order_num

20007

20008

Bước kế tiếp là truy vấn các số ID của khách hàng có các đơn đặt hàng là 20007 và 20008. Việc sử dụng mệnh đề IN được mô tả trong bài học 5 "Lọc dữ liệu nâng cao", bạn có thể tạo một câu lệnh SELECT như sau:

```

SELECT cust_id ;
FROM Orders ;
WHERE order_num IN (20007, 20008);

```

Kết quả

Cust_id

1000000004

1000000005

Bây giờ hãy kết hợp hai câu Queries bằng cách biến câu Query đầu tiên (là câu Query vốn trả về các mã số đơn đặt hàng) thành một Query con. Hãy xem qua câu lệnh SELECT sau:

```

SELECT cust_id ;
FROM Orders ;
WHERE order_num IN (SELECT ;
FROM OrderItems ;
WHERE prod_id = 'RGAN01');

```

Kết quả

Cust_id

1000000004

1000000005

- Các Query con luôn được xử lý bắt đầu bằng một câu lệnh SELECT ở trong cùng và thực hiện từ trong ra ngoài. Khi câu lệnh SELECT trước được xử lý, DBMS thực thi hai hoạt động.

Trước tiên nó chạy Query con:

```

SELECT Order_num ;
FROM Orderitems ;

```

WHERE prod_id = 'RAGN01'

Câu Query đó sẽ trả về hai mã số đơn đặt hàng là 20007 và 20008. Hai giá trị khi đó được truyền cho mệnh đề WHERE của câu Query ngoài và được tách nhau bởi dấu phẩy bởi toán tử IN. Câu Query ngoài bây giờ trở thành:

SELECT Cust_id ;

FROM Order ;

WHERE Order_num IN (20007, 20008)

Như bạn có thể thấy, kết quả xuất chính xác và đúng như cùng dữ liệu xuất được trả về bởi mệnh đề WHERE được viết ở trên.

- Việc định dạng SQL của bạn. Các câu lệnh SELECT chứa các Subqueries có thể khó đọc và dễ nhầm lẫn, đặc biệt khi chúng càng ngày trở nên phức tạp. Việc phân các Query thành nhiều dòng và thụt dòng thích hợp như được trình bày ở đây có thể giúp đơn giản hoá khi làm việc với các Query con.

- Bây giờ bạn có các ID của tất cả khách hàng đã đặt món hàng RAGN01. Bước kế tiếp là truy vấn thông tin các khách hàng cho mỗi mã số ID của khách hàng. Câu lệnh SQL để truy vấn hai cột đó là:

SELECT cust_name, cust_contact ;

FROM Customers ;

WHERE cust_id IN ('1000000004', '1000000005');

Thay vì mã hoá các số ID khách hàng này, bạn có thể biến mệnh đề WHERE này thành Query con.

SELECT cust_name, cust_contact ;

FROM Customers ;

WHERE cust_id IN (SELECT cust_id ;

FROM Orders ;

WHERE order_num IN (SELECT order_num ;


```
FROM orderItems ;
WHERE prod_id = 'RGAN01'));
```

Kết quả

Cust_name	Cust_contact
-----	-----
Fun4ALL	Denise L. Stephens
The Toy Store	Kim Howard

- Để thực thi câu lệnh SELECT. Trên DBMS phải thực sự thực hiện ba câu lệnh SELECT. Query con nằm trong cùng nhất trả về một danh sách các mã số đơn đặt hàng vốn được sử dụng làm mệnh đề WHERE cho Query con ở trên nó. Câu Query con đó trả về một danh sách các ID của khách hàng vốn sử dụng làm mệnh đề WHERE cho Query cấp trên nữa. Query cấp trên thực sự trả về dữ liệu mong muốn.

- Như bạn có thể thấy việc sử dụng các Subqueries trong một mệnh đề cho phép bạn viết câu lệnh SQL vô cùng mạnh mẽ và linh hoạt. Không lạm dụng số các Query con có thể được lồng vào nhau, mặc dù trong bài thực hành này bạn sẽ nhận thấy rằng bạn đang lồng các Query con quá sâu.

- Chỉ một cột đơn các câu lệnh SELECT của các Query con có thể chỉ truy vấn một cột đơn nếu bạn cố gắng truy vấn nhiều cột sẽ trả về một lỗi.

- Các Query con và sự thực thi: Đoạn Code được trình bày ở đây hoạt động, và nó đạt được các kết quả mong muốn. Tuy nhiên, việc sử dụng các Query con không phải luôn luôn là cách hiệu quả nhất để thực hiện việc truy vấn các dữ liệu này.

** Sử dụng các Query con làm trường tính toán*

Một cách khác để sử dụng các Query con là để tạo các trường

tính toán. Tin rằng bạn muốn hiển thị tổng số đơn đặt hàng được đặt hàng bởi mỗi khách hàng trong Table Customers của bạn. Các đơn đặt hàng được lưu trữ trong Table Orders cùng với số ID khách hàng thích hợp.

Để thực hiện hoạt động này, hãy theo các bước sau:

1. Truy vấn danh sách các khách hàng từ Table Customers.
2. Đối với mỗi khách hàng được truy vấn, hãy đếm số đơn đặt hàng ứng với nó trong Table Order.

- Như bạn đã học trong bài học trước, bạn có thể sử dụng SELECT COUNT (*) để đếm các dòng trong một Table, và bằng cách cung cấp một mệnh đề WHERE để lọc số ID khách hàng cụ thể, bạn có thể đếm chỉ các đơn đặt hàng của khách hàng đó. Ví dụ, đoạn Code Counts sẽ đếm số các đơn đặt hàng được đặt hàng theo số ID khách hàng là 1000000001:

```
SELECT COUNT (*) AS orders ;
```

```
FROM Orders ;
```

```
WHERE cust_id = '1000000001';
```

Để thực hiện phép tính COUNT (*) cho mỗi khách hàng, hãy sử dụng COUNT (*) làm một Query con, xem qua đoạn Code như sau.

```
SELECT cust_name, Cust_state, ;
```

```
(SELECT COUNT (*) ;
```

```
FROM Orders ;
```

```
WHERE Orders.cust_id = Customers.cust_id AS orders;
```

```
FROM Customers ;
```

```
ORDER BY cust_name;
```

Kết quả

Cust_name	cust_state	orders
-----	-----	-----
Fun4ALL	IN	1
Fun4ALL	AZ	1
Kids Place	OH	0
The Toy Store	IL	1
Village Toys	MI	2

- Câu lệnh SELECT trả về ba cột cho mỗi khách hàng trong Table Customers: Cust_name, Cust_State và Order. Order là một trường đã tính toán vốn được xác lập bởi một Query con vốn được cung cấp trong các dấu ngoặc đơn. Query con đó được thực thi một lần đối với mỗi khách hàng đã được truy vấn. Trong ví dụ trên, Query con được thực thi 5 lần bởi vì 5 khách hàng được truy vấn.

- Mệnh đề WHERE trong câu Query con hơi khác với mệnh đề WHERE được sử dụng trước đây bởi vì nó sử dụng các tên cột đầy đủ. Mệnh đề sau đề nghị SQL so sánh cust_id trong Table order với ID đang được truy vấn từ Table Customers:

WHERE Orders.cust_id = Customers.cust_id

- Theo cú pháp này, tên Table và tên cột được tách nhau bởi một dấu chấm. Phải được sử dụng bất cứ khi nào có thể có sự tối nghĩa về các tên cột. Trong ví dụ này có hai cột Cust_id, một là ở trong Customers và một ở bên trong Order. Nếu không phát biểu các tên cột rõ ràng, DBMS sẽ cho là bạn đang so sánh Cust_id trong Table Orders với bản thân nó. Bởi vì:

SELECT COUNT () ;*

FROM Order WHERE cust_id = cust_id

Sẽ luôn trả về tổng số đơn đặt hàng trong Table Order, kết quả này sẽ không phải là những gì bạn đang mong đợi:

```
SELECT cust_name, Cust_state, ;  
(SELECT COUNT (*) ;  
FROM Orders ;  
WHERE cust_id = cust_id) AS orders;  
FROM Customers ;  
ORDER BY cust_name;
```

Kết quả

Cust_name	cust_state	orders
-----	-----	-----
Fun4ALL	IN	5
Fun4ALL	AZ	5
Kids Place	OH	5
The Toy Store	IL	5
Village Toys	MI	5

Mặc dù các Query con vô cùng hữu dụng trong việc xây dựng loại câu lệnh SELECT này nhưng cũng hết sức cẩn thận khi viết tên cột có nghĩa một cách thích hợp.

- Luôn có nhiều hơn một giải pháp. Như được giải thích ở trước đó trong bài học này, mặc dù các đoạn Code mẫu được trình bày ở đây hoạt động, nó thường không phải là cách hiệu quả nhất để thực hiện truy vấn này. Bạn sẽ xem lại ví dụ này trong một bài học sau.

i. Kết các Table

**** Tìm hiểu các phép nối***

Một trong các tính năng mạnh mẽ nhất của SQL là khả năng kết các Table nhanh chóng trong các Query truy vấn dữ liệu. Phép

nổi là một trong những hoạt động mà bạn có thể thực hiện bằng cách sử dụng SELECT của SQL. Tìm hiểu tốt về các phép nối và cú pháp phép nối là phần vô cùng quan trọng trong việc học SQL.

- Trước khi bạn sử dụng các phép nối một cách có hiệu quả, bạn phải tìm hiểu các Table quan hệ và các khái niệm căn bản về thiết kế cơ sở dữ liệu quan hệ. Những gì sau đây không nói hết về toàn bộ môn này, nhưng nó cũng đủ để bạn nắm rõ.

** Tìm hiểu các quan hệ*

Cách tốt nhất để tìm hiểu các Table quan hệ là xem qua một ví dụ thực tế.

- Giả sử bạn đã có một Table cơ sở dữ liệu chứa một danh mục sản phẩm, với mỗi món hàng của danh mục nằm trong dòng riêng của nó. Loại thông tin bạn sẽ lưu trữ với mỗi món hàng bao gồm phần mô tả và giá của sản phẩm, cùng với thông tin nhà sản xuất về công ty vốn tạo ra sản phẩm đó.

- Bây giờ giả sử rằng bạn có nhiều món hàng trong danh mục được tạo cùng một nhà sản xuất. Bạn sẽ lưu trữ thông tin nhà sản xuất ở đâu (những thứ như tên nhà sản xuất, địa chỉ, thông tin liên lạc). Bạn sẽ không muốn lưu trữ dữ liệu đó chung với các sản phẩm vì các lý do sau:

+ Bởi vì thông tin nhà sản xuất là giống nhau đối với mỗi sản phẩm mà nhà sản xuất đó tạo ra, việc lặp lại thông tin này cho mỗi sản phẩm là một điều lãng phí thời gian và không gian lưu trữ.

+ Nếu thông tin nhà sản xuất thay đổi (ví dụ, nếu nhà sản xuất đó dời đi hoặc mã vùng của anh ta thay đổi), bạn sẽ cần cập nhật sau mỗi biến cố về thông tin nhà sản xuất.

+ Khi dữ liệu lặp lại (thông tin nhà sản xuất này được sử dụng với mỗi sản phẩm). Có khả năng lớn xảy ra việc nhập dữ liệu

không chính xác mỗi lần nhập. Dữ liệu không nhất quán vô cùng khó khăn khi sử dụng để báo cáo.

- Điểm chính ở đây là có nhiều biến cố chung dữ liệu không bao giờ là một điều tốt, và nguyên tắc đó là nền tảng để thiết kế cơ sở dữ liệu quan hệ. Các Table quan hệ được thiết kế để thông tin được phân thành nhiều Table, một Table cho mỗi loại dữ liệu. Các Table quan hệ với nhau qua những giá trị chung (và vì thế có quan hệ trong thiết kế quan hệ).

- Trong ví dụ của chúng ta, bạn có thể tạo hai Table, một cho thông tin nhà sản xuất và một cho thông tin sản phẩm. Table Vendors chứa tất cả thông tin nhà sản xuất, một dòng Table/một nhà sản xuất, cùng với một định danh (ID) duy nhất cho mỗi nhà sản xuất. Giá trị này, được gọi là khoá chính, có thể là một ID của nhà sản xuất (Vendor ID), hoặc giá trị duy nhất khác bất kỳ.

- Table Products chỉ lưu trữ thông tin nhà sản xuất, và không có thông tin cụ thể nào của nhà sản xuất khác hơn là Vendor ID đó (khoá chính của Table Vendors). Khoá này tạo quan hệ cho Table Vendors với Table Products, và việc sử dụng Vendors ID cho phép bạn sử dụng Table Vendors để tìm các chi tiết về nhà sản xuất thích hợp.

Điều này làm gì cho bạn? Vâng, hãy xem xét các vấn đề sau:

- + Thông tin nhà sản xuất không bao giờ được lặp lại, và vì vậy thời gian và không gian đĩa không bị lãng phí.

- + Nếu thông tin nhà sản xuất thay đổi, bạn có thể cập nhật một Record đơn, một trong Table Vendors. Dữ liệu trong Table có quan hệ không thay đổi.

Dòng cuối cùng là dữ liệu quan hệ có thể được lưu trữ một cách hiệu quả và xử lý dễ dàng. Do nguyên nhân này, cơ sở dữ liệu quan hệ co giãn tốt hơn cơ sở dữ liệu không quan hệ.

** Tại sao phải sử dụng các phép nối kết?*

Như vừa giải thích, việc phân dữ liệu thành nhiều Table cho phép lưu trữ hiệu quả hơn, dễ xử lý hơn, và khả năng co giãn lớn hơn.

Nếu dữ liệu được lưu trữ trong nhiều Table, bạn có thể truy vấn dữ liệu đó với một câu lệnh SELECT đơn giản như thế nào?

Câu trả lời là sử dụng một phép kết. Đơn giản, một phép kết là một cơ chế được sử dụng để liên kết các Table trong một câu lệnh SELECT. Việc sử dụng một cú pháp đặc biệt, nhiều Table có thể kết lại để một tập kết quả xuất đơn được trả về, và phép kết này liên kết các dòng đúng trong mỗi Table nhanh chóng.

- Sử dụng các công cụ DBMS tương tác (Interactive DBMS Tool). Điều quan trọng phải hiểu một phép kết không phải là một thực thể vật lý - nói cách khác, nó không tồn tại trong các Table cơ sở dữ liệu thực. Một phép kết được tạo bởi DBMS khi cần tồn tại chỉ trong lúc thực thi Query.

- Nhiều DBMS cung cấp giao diện đồ họa vốn có thể được sử dụng để định nghĩa các mối quan hệ Table tương tác lẫn nhau. Các công cụ này giúp đỡ rất nhiều trong việc duy trì tính toàn vẹn tham chiếu. Khi sử dụng các Table quan hệ, điều quan trọng là chỉ dữ liệu hợp lệ mới được chèn vào các cột quan hệ. Trở lại ví dụ này, nếu một Vendor ID không hợp lệ được lưu trữ trong Table Products, những sản phẩm đó sẽ không thể truy cập bởi vì chúng không có quan hệ với bất kỳ nhà sản xuất nào. Để tránh điều này xảy ra, cơ sở dữ liệu có thể được chỉ thị để chỉ cho phép các giá trị hợp lệ (các giá trị phải có trong Table Vendor) trong cột Vendor ID trong Table Products. Toàn vẹn tham chiếu có nghĩa là DBMS áp đặt các quy luật toàn vẹn dữ liệu. Và những quy luật này thường được quản lý qua giao diện được cung cấp của DBMS.

** Tạo một phép kết*

Tạo một phép kết rất đơn giản. Bạn phải chỉ định tất cả các Table được đưa vào và cách chúng được tạo quan hệ với nhau. Xem qua ví dụ:

```
SELECT vend_name, prod_name, prod_price ;  
FROM Vendors, Products ;  
WHERE Vendors.vend_id = Products.vend_id;
```

Kết quả

Vend_name	prod_name	prod_price
-----	-----	-----
Doll House Inc.	Fish bean bag toy	3.4900
Doll House Inc.	Bird bean bag toy	3.4900
Doll House Inc.	Rabbit bean bag toy	3.4900
Bears R Us	8 inch teddy bear	5.9900
Bears R Us	12 inch teddy bear	8.9900
Bears R Us	18 inch teddy bear	11.9900
Doll House Inc.	Raggedy Ann	3.4900
Fun and Games	King doll	9.4900
Fun and Games	Queen doll	9.4900

- Hãy xem qua đoạn mã trên. Câu lệnh SELECT bắt đầu theo cùng cách như tất cả các câu lệnh mà bạn đã xem qua cho đến thời điểm này, bằng cách chỉ định các cột được truy vấn. Sự khác biệt lớn ở đây là hai cột (prod_name và prod_price) nằm trong một Table, trong khi vend_name nằm trong Table khác.

- Bây giờ, hãy xem mệnh đề FROM. Không giống như các câu lệnh SELECT trước, câu lệnh này có hai Table được liệt kê trong mệnh đề FROM, Vendors và Products. Đây là tên của hai Table đang được kết trong câu lệnh SELECT này. Các Table được kết lại một cách

chính xác với một mệnh đề WHERE vốn chỉ thị DBMS tương kết vend_id trong Table Vendors với vend_id trong Table Products.

- Bạn sẽ dễ ý thấy các cột được chỉ định dưới dạng Vendors.vend_id và Products.vend_id. Tên cột được phát biểu đầy đủ này bị bắt buộc ở đây bởi vì nếu bạn chỉ định vend_id, DBMS không thể biết được cột vend_id nào mà bạn đang tham chiếu đến (có hai trong chúng, một trong mỗi Table). Như bạn có thể thấy trong kết quả xuất trước, một câu lệnh SELECT - đơn trả về dữ liệu từ hai Table khác nhau.

- Các tên cột đầy đủ: Bạn phải sử dụng tên cột đầy đủ (Table và cột được tách biệt bởi một dấu chấm) bất cứ khi nào mơ hồ về cột nào mà bạn đang tham chiếu đến. Hầu hết các DBMS sẽ trả về một thông điệp lỗi nếu bạn tham gia chiếu đến một tên cột mơ hồ mà không phát biểu nó đầy đủ với tên của một Table.

** Sự quan trọng của mệnh đề WHERE*

- Sử dụng một mệnh đề WHERE để xác lập mối quan hệ của phép kết đường như hơi lạ, nhưng thực sự, có lý do chính đáng. Hãy nhớ, khi các Table được kết lại trong một câu lệnh SELECT, mỗi quan hệ được xây dựng nhanh chóng. Không làm được gì trong các định nghĩa Table của cơ sở dữ liệu vốn có thể chỉ thị DBMS cách kết các Table. Bạn phải tự làm điều đó. Khi bạn kết hai Table, những gì bạn thực sự làm là ghép mỗi dòng trong Table đầu tiên với mỗi dòng trong Table thứ hai. Mệnh đề WHERE hoạt động dưới dạng một bộ lọc để chỉ đưa vào dòng thoả mãn điều kiện lọc cụ thể - điều kiện kết, trong trường hợp này. Nếu không có mệnh đề WHERE, mỗi dòng trong Table đầu tiên được ghép với mỗi dòng trong Table thứ hai, không quan tâm đến chúng có hợp logic với nhau hay không:

Để tìm hiểu vấn đề này, hãy xem câu lệnh SELECT sau và kết quả xuất:

```
SELECT vend_name, prod_name, prod_price ;
FROM Vendors, Products;
```

Kết quả

Vend_name	prod_name	prod_price
-----	-----	-----
Bears R Us	8 inch teddy bear	5.99
Bears R Us	12 inch teddy bear	8.99
Bears R Us	18 inch teddy bear	11.99
Bears R Us	Fish bean bag toy	3.49
Bears R Us	Bird bean bag toy	3.49
Bears R Us	Rabbit bean bag toy	3.49
Bears R Us	Raggedy Ann	4.99
Bears R Us	King doll	9.49
Bears R Us	Queen doll	9.49
Bear Emporium	8 inch teddy bear	5.99
Bear Emporium	12 inch teddy bear	8.99
Bear Emporium	18 inch teddy bear	11.99
Bear Emporium	Fish bean bag toy	3.49
Bear Emporium	Bird bean bag toy	3.49
Bear Emporium	Rabbit bean bag toy	3.49
Bear Emporium	Raggedy Ann	4.99
Bear Emporium	King doll	9.49
Bear Emporium	Queen doll	9.49
Doll House Inc.	8 inch teddy bear	5.99
Doll House Inc.	12 inch teddy bear	8.99
Doll House Inc.	18 inch teddy bear	11.99

Doll House Inc.	Fish bean bag toy	3.49
Doll House Inc.	Bird bean bag toy	3.49
Doll House Inc.	Rabbit bean bag toy	3.49
Doll House Inc.	Raggedy Ann	4.99
Doll House Inc.	King doll	9.49
Doll House Inc.	Queen doll	9.49
Furball Inc.	8 inch teddy bear	5.99
Furball Inc.	12 inch teddy bear	8.99
Furball Inc.	18 inch teddy bear	11.99
Furball Inc.	Fish bean bag toy	3.49
Furball Inc.	Bird bean bag toy	3.49
Furball Inc.	Rabbit bean bag toy	3.49
Furball Inc.	Raggedy Ann	4.99
Furball Inc.	King doll	9.49
Furball Inc.	Queen doll	9.49
Fun and Games	8 inch teddy bear	5.99
Fun and Games	12 inch teddy bear	8.99
Fun and Games	18 inch teddy bear	11.99
Fun and Games	Fish bean bag toy	3.49
Fun and Games	Bird bean bag toy	3.49
Fun and Games	Rabbit bean bag toy	3.49
Fun and Games	Raggedy Ann	4.99
Fun and Games	King doll	9.49
Fun and Games	Queen doll	9.49
Jouets et ours	8 inch teddy bear	5.99
Jouets et ours	12 inch teddy bear	8.99

Jouets et ours	18 inch teddy bear	11.99
Jouets et ours	Fish bean bag toy	3.49
Jouets et ours	Bird bean bag toy	3.49
Jouets et ours	Rabbit bean bag toy	3.49
Jouets et ours	Raggedy Ann	4.99
Jouets et ours	King doll	9.49
Jouets et ours	Queen doll	9.49

- Như bạn có thể thấy trong phần kết quả xuất, tích đề cát ít khi là những gì bạn cần. Dữ liệu được trả về ở đây tương kết mỗi sản phẩm với mỗi nhà sản xuất, bao gồm cả những sản phẩm không có nhà sản xuất đúng (và thậm chí các nhà sản xuất không có sản phẩm nào cả).

- Cảnh báo: Đừng quên mệnh đề WHERE: Đảm bảo rằng tất cả các phép kết của bạn đều có mệnh đề WHERE, hoặc DBMS sẽ trả về nhiều dữ liệu hơn bạn muốn. Tương tự, đảm bảo rằng mệnh đề WHERE chính xác. Một điều kiện lọc không chính xác gây cho DBMS trả về dữ liệu sai.

- Các phép kết chéo (Cross Join): Thỉnh thoảng, bạn sẽ nghe nói loại phép kết trả về tích đề cát được gọi là phép kết chéo.

** Inner Join (nội kết)*

- Phép kết mà bạn đã học cho đến thời điểm này được gọi là phép kết dựa trên phép kiểm tra bằng giữa hai Table. Loại phép kết này còn được gọi là Inner join (nội kết). Thật ra, bạn có thể sử dụng cú pháp hơi khác cho những phép kết này, bằng cách chỉ định loại phép kết rõ ràng. Câu lệnh sau trả về cùng dữ liệu chính xác như ví dụ trước:

```
SELECT vend_name, prod_name, prod_price ;
FROM Vendors INNER JOIN Products ;
ON Vendors.vend_id = Products.vend_id;
```

- Lệnh SELECT trong câu lệnh này giống như câu lệnh

SELECT trước, nhưng mệnh đề FROM hơi khác. Ở đây mối quan hệ giữa hai Table là một phần của mệnh đề FROM được chỉ định dưới dạng INNER JOIN. Khi sử dụng cú pháp này, điều kiện kết được chỉ định bằng cách sử dụng mệnh đề ON đặc biệt thay cho mệnh đề WHERE. Điều kiện thực được truyền cho ON cũng như được truyền cho WHERE.

Tham khảo tài liệu DBMS của bạn để xem cú pháp nào được ưu tiên.

- Ghi chú: Cú pháp "đúng" trên đặc tả ANSI SQL, việc dùng cú pháp INNER JOIN có thể ưu tiên.

** Kết nhiều table*

SQL không hạn chế số lượng Table được kết lại trong một câu lệnh SELECT. Các quy tắc cơ bản để tạo phép kết nối vẫn giống nhau. Đầu tiên liệt kê tất cả các Table, và sau đó định nghĩa mối quan hệ giữa mỗi Table. Đây là một ví dụ:

```
SELECT prod_name, vend_name, prod_price, Quantity;
FROM OrderItems, Products, Vendors ;
WHERE Products.vend_id = Vendors.vend_id ;
AND OrderItems.prod_id = Products.prod_id ;
AND order_num = 20007;
```

Kết quả

Prod_name	vend_name	prod_price	Quantity
-----	-----	-----	-----
18 inch teddy bear	Bears R Us	11.9900	50
Fish bean bah toy	Doll House Inc.	3.4900	100
Bird bean bah toy	Doll House Inc.	3.4900	100
Rabbit bean bah toy	Doll House Inc.	4.4900	100
Raggedy Ann	Doll House Inc.	3.4900	50

- Ví dụ này hiển thị các món hàng trong đơn đặt hàng 20007. Các món hàng của đơn đặt hàng được lưu trữ trong Table OrderItems. Mỗi sản phẩm được lưu trữ bởi product ID, tham chiếu một sản phẩm trong Table Products. Các sản phẩm được liên kết tới nhà sản xuất thích hợp trong Table Vendors bởi Vendor ID, được lưu trữ với mỗi Record sản phẩm. Mệnh đề FROM ở đây liệt kê 3 Table, và mệnh đề WHERE định nghĩa cả hai điều kiện này. Một điều kiện WHERE bổ sung được sử dụng khi nó để lọc chỉ những món hàng cho đơn đặt hàng 20007.

- Xem xét việc thực thi: Các DBMS xử lý các phép kết thực thi hành bằng cách tạo quan hệ cho mỗi Table như được chỉ định. Việc xử lý này đòi hỏi rất nhiều tài nguyên, vì thế hãy cẩn thận không nên kết các Table không cần thiết. Bạn hãy kết càng nhiều Table thì tốc độ thực thi sẽ càng giảm.

- Số lượng Table tối đa trong một phép kết: Sự thật là bản thân SQL không hạn chế số lượng Table trên một phép kết, nhưng nhiều DBMS thực sự có hạn chế. Tham khảo tài liệu để xác định xem có những hạn chế nào.

Bây giờ là thời điểm tốt nhất để xem lại ví dụ sau trong bài "Làm việc với các Query con". Câu lệnh SELECT này trả về một danh sách các khách hàng đã đặt sản phẩm RGAN01:

```
SELECT cust_name, cust_contact ;  
FROM Customers ;  
WHERE cust_id IN (SELECT cust_id ;  
FROM Orders WHERE order_num IN ;  
(SELECT order_num ;  
FROM orderItems ;  
WHERE prod_id = 'RGAN01'));
```

- Như đã đề cập ở bài trước, các Query con không luôn là cách hiệu quả nhất để thực thi các thao tác SELECT phức tạp, và vì vậy như đã hứa, đây là cùng câu Query nhưng sử dụng phép kết:

```
SELECT cust_name, cust_contact;  
FROM Customers, Orders, OrderItems;  
WHERE Customers.cust_id = Orders.cust_id;  
AND OrderItems.order_num = Orders.order_num;  
AND prod_id = 'RGN01';
```

Kết quả

Cust_name	cust_contact
Fun4ALL	L. Stephens
The Toy Store	Kim Howard

- Như được giải thích trong phần trước, việc trả về dữ liệu cần trong câu Query này đòi hỏi phải dùng 3 Table. Nhưng thay vì sử dụng chúng trong các câu Query lồng nhau, ở đây hai phép kết được sử dụng để kết nối các Table. Có 3 điều kiện mệnh đề WHERE ở đây. Đầu tiên là hai điều kiện kết các Table trong phép kết, và điều kiện cuối cùng là lọc dữ liệu cho sản phẩm RAGN01.

- Như bạn có thể thấy, thường có nhiều cách để thực hiện thao tác SQL đưa ra bất kỳ. Và hiếm có một cách có định nghĩa đúng hay sai. Tốc độ thực thi có thể bị ảnh hưởng bởi loại thao tác. DBMS đang sử dụng, lượng dữ liệu trong các Table, có các Index và các khoá hay không, và toàn bộ. Vì vậy, thường phải thử nghiệm với các cơ chế khác nhau để tìm ra một cách tốt nhất cho bạn.

j. Tạo các phép kết nâng cao

**** Sử dụng các bí danh Table***

Cú pháp để đặt bí danh một cột như sau:

```

SELECT RTRIM (vend_name) + σ' (' + ;
RTRIM (vend_country) + ') AS vend_title;
FROM Vendors;
ORDER BY vend_name;

```

Ngoài việc sử dụng các bí danh cho các tên cột và các trường đã được tính toán, SQL cũng cho phép bạn đặt bí danh cho các tên Table. Có hai lý do chủ yếu để thực hiện điều này:

- + Để rút ngắn cú pháp SQL
- + Để cho phép nhiều việc sử dụng cùng Table trong một câu lệnh SELECT đơn.

Xem qua câu lệnh SELECT sau. Về cơ bản thì cũng giống như câu lệnh của ví dụ đã được sử dụng trong bài học trước nhưng nó đã được chỉnh sửa để sử dụng các bí danh:

```

SELECT cust_name, cust_contact;
FROM Customers AS C, Orders AS O, OrderItems AS OI;
WHERE C.cust_id = O.cust_id;
AND OI.order_num = O.cust_id;
AND prod_id = 'RGAN01';

```

- Bạn hãy để ý rằng ba Table trong các mệnh đề FROM tất cả đều có bí danh. Customers AS C thiết lập C làm một bí danh cho Customers ... Điều này cho phép sử dụng dạng ngắn C thay cho Text Customers đầy đủ. Trong ví dụ này các bí danh Table được sử dụng chỉ trong mệnh đề WHERE, nhưng các bí danh không được giới hạn chỉ ở mệnh đề WHERE. Bạn có thể sử dụng các bí danh trong danh sách SELECT, mệnh đề ORDER BY và trong bất cứ phần nào của câu lệnh.

- Không có AS trong Oracle. Oracle không hỗ trợ từ khoá AS. Để sử dụng các bí danh trong Oracle, đơn giản chỉ định bí danh mà không cần AS (Sử dụng Customers C thay cho Customers AS C).

- Điều đáng chú ý là các bí danh Table chỉ được sử dụng trong khi thực thi Query. Không giống như các bí danh cột, các bí danh Table không bao giờ được trả về cho Client.

** Sử dụng các phép kết khác*

Cho đến thời điểm này, bạn đã sử dụng chỉ các kết đơn giản như nội kết (Inner Join) hoặc các kết bằng (Equijoin). Bây giờ bạn sẽ xem qua 3 loại phép kết bổ sung: Tự kết (Self Join), kết tự nhiên (Natural Join), và ngoại kết (Outer).

** Tự kết (Self Join)*

- Như đã được đề cập trước đó, một trong các lý do chủ yếu sử dụng các bí danh Table là để có thể ám chỉ đến Table tương tự nhiều hơn một lần trong một câu lệnh SELECT đơn. Một ví dụ minh họa vấn đề này.

- Giả sử bạn muốn gửi Email cho tất cả các người bạn của bạn của khách hàng mà làm việc cho cùng công ty của Jim Jones. Câu Query này đòi hỏi rằng bạn trước hết phải tìm ra tên công ty mà Jim Jones làm việc, và tiếp đó thì tìm những khách hàng làm việc cho công ty đó. Sau đây là một cách để tiếp cận vấn đề này:

```
SELECT cust_id, cust_name, cust_contact;  
FROM Customers ;  
WHERE cust_name = (SELECT cust_name;  
FROM Customers ;  
WHERE cust_contact = 'Jim Jones');
```

Kết quả

Cust_id	cust_name	cust_contact
-----	-----	-----
1000000003	Fun4ALL	Jim Jones
1000000004	Fun4ALL	Denise L. Stephens

- Giải pháp đầu tiên sử dụng các Query con. Câu lệnh SELECT bên trong thực hiện một truy vấn đơn giản trả về Cust_name của công ty mà Jim Jones làm việc. Tên nó là một tên được sử dụng trong mệnh đề WHERE của câu Query ngoài để tất cả các nhân viên làm việc cho công ty đó đều được truy vấn.

Bây giờ xem qua cùng câu truy vấn sử dụng một kết nối:

```
SELECT c1.cust_id, c1.cust_name, c1.cust_contact;
```

```
FROM Customers AS c1, Customers AS c2;
```

```
WHERE c1.cust_name = c2.cust_name;
```

```
AND c2.cust_contact = 'Jim Jones';
```

Kết quả

Cust_id	cust_name	cust_contact
-----	-----	-----
1000000003	Fun4ALL	Jim Jones
1000000004	Fun4ALL	Denise L. Stephens

- Không có AS trong Oracle những người sử dụng Oracle, hãy nhớ rằng là phải bỏ AS.

- Hai Table cần trong Query này thực sự là nằm trong cùng một Table, và vì thế Table Customers xuất hiện trong mệnh đề FROM hai lần. Mặc dù điều này hoàn toàn hợp lệ, nhưng bất cứ tham chiếu nào đến Table Customers đều mơ hồ bởi vì DBMS không biết Table Customers nào mà bạn đang tham chiếu.

- Để giải quyết vấn đề này các bí danh Table được sử dụng. Biến thể đầu tiên của Customers có một bí danh là C1 và biến thể thứ hai có bí danh là C2. Bây giờ các biến thể có thể được sử dụng làm tên các Table. Câu lệnh SELECT, ví dụ, sử dụng tiền tố C1 để phát biểu tên đầy đủ của các cột muốn. Nếu không làm như vậy, DBMS sẽ trả về một lỗi bởi vì hai được đặt tên Cust_id, Cust_name, và Cust_contact. Nó không thể biết được cái nào bạn

muốn sử dụng (mặc dù, thật ra, chúng là một và giống nhau). Mệnh đề WHERE đầu tiên kết các Table, và sau đó nó lọc dữ liệu theo Cust_contact trong Table thứ hai trả về chỉ dữ liệu mong muốn.

- Self Join (tự kết) thay cho các Query con, tự kết (Self Join) thường được sử dụng để thay thế cho các câu lệnh sử dụng các Query con muốn truy vấn dữ liệu từ cùng Table đó dưới dạng câu lệnh ngoài. Mặc dù kết quả cuối cùng là như nhau, nhiều DBMS sử lý các kết nối nhanh hơn nhiều so với chúng thực hiện các Query con. Chúng ta cần thử nghiệm với cả hai phương pháp để xác định xem phương pháp nào thực hiện tốt hơn.

** Kết tự nhiên (Natural Join)*

Bất cứ khi nào các Table được kết lại, tối thiểu một cột sẽ xuất hiện trong nhiều hơn một Table (các cột đang được kết lại). Các kết chuẩn (các Inner Join mà bạn đã học trong bài học trước) trả về tất cả dữ liệu, thậm chí nhiều biến thể của cùng một cột. Một phép kết tự nhiên đơn giản loại trừ các biến thể này để chỉ một biến thể trong mỗi cột được trả về. Cách nó làm điều này như thế nào? Câu trả lời là nó không làm gì - bạn làm điều đó. Kết tự nhiên là một phép kết mà trong đó bạn chọn chỉ các cột vốn duy nhất. Điều này thường thực hiện bằng cách sử dụng một ký tự đại diện (SELECT *) cho một Table và các tập con rõ ràng của các cột cho tất cả các Table khác. Sau đây là một ví dụ:

```
SELECT C.*, O.order_num, O.order_date, ;  
      OI.prod_id, OI.Quantity, OI.item_price;  
FROM Customers AS C, Orders AS O, OrderItems AS OI;  
WHERE C.cust_id = O.cust_id;  
      AND OI.order_num = O.order_num;  
      AND prod_id = 'RGAN01';
```

- Không có AS trong Oracle. Những người sử dụng Oracle, hãy nhớ rằng phải bỏ AS.

- Trong ví dụ này, một ký tự đại diện được sử dụng chỉ Table đầu tiên. Tất cả các cột khác được liệt kê rõ ràng để không có cột trùng nào được truy vấn.

- Sự thật là, mỗi phép kết Inner Join mà bạn đã tạo cho đến thời điểm này thực sự là một phép kết tự nhiên (Natural Join), và bạn sẽ có thể không bao giờ cần một Inner Join mà không phải là một phép kết tự nhiên.

** Outer Joins (ngoại kết)*

Hầu hết các phép kết tạo quan hệ cho các dòng trong một Table với các dòng trong một Table khác. Nhưng đôi khi, bạn sẽ muốn đưa vào các dòng vốn không có các dòng quan hệ với nó. Ví dụ, bạn có thể sử dụng các kết nối để thực hiện các chức vụ sau:

+ Đếm số đơn đặt hàng cho mỗi khách hàng, bao gồm các khách hàng còn đang phải đặt đơn đặt hàng và đã đặt đơn đặt hàng.

+ Liệt kê tất cả các sản phẩm, với tất cả các đơn đặt hàng, bao gồm các sản phẩm chưa có đơn đặt hàng.

+ Tính cỡ bán trung bình, báo cáo các khách hàng chưa đặt đơn đặt hàng.

- Trong mỗi ví dụ này, phép kết đưa vào các dòng Table vốn không có các dòng tương kết trong Table quan hệ với Table đó. Kiểu kết này được gọi là ngoại kết (Outer Join).

- Sự khác biệt cú pháp cần lưu ý rằng cú pháp được sử dụng để tạo một Outer Join có thể thay đổi hơi khác giữa các trình triển khai SQL. Các dạng khác nhau của cú pháp mô tả trong phần sau sử dụng hầu hết trong các trình triển khai, hãy tham khảo tài liệu DBMS của bạn để thẩm định lại các cú pháp của nó trước khi bắt đầu.

- Câu lệnh SELECT sau là một Inner Join đơn giản (nội kết). Nó truy vấn một danh sách các khách hàng và các đơn đặt hàng của họ:

```
SELECT Customers.cust_id, Orders.order_num;  
FROM Customers INNER JOIN Orders;  
ON Customers.cust_id = Orders.cust_id;
```

Cú pháp Outer Join tương tự. Để truy vấn một danh sách tất cả các khách hàng không đặt đơn đặt hàng nào, bạn có thể viết như sau:

```
SELECT Customers.cust_id, Orders.order_num;  
FROM Customers LEFT OUTER JOIN Orders;  
On Customers.cust_id = Orders.cust_id;
```

Kết quả

Cust_id	order_num
-----	-----
1000000001	20005
1000000001	20009
1000000002	NULL
1000000003	20006
1000000004	20007
1000000005	20008

- Giống như nội kết (Inner Join) đã thấy trong bài học trước, câu lệnh SELECT sử dụng các từ khoá OUTER JOIN để chỉ định loại kết (thay vì chỉ định nó trong mệnh đề WHERE). Nhưng không giống như Inner Join (nội kết), loại kết này tạo quan hệ cho các dòng trong cả hai Table, còn Outer Join (ngoại kết) đưa vào các dòng mà không có các dòng quan hệ. Khi sử dụng cú pháp OUTER JOIN bạn phải sử dụng các từ khoá RIGHT hoặc LEFT để chỉ định Table đưa tất cả các dòng vào (RIGHT cho Table nằm bên phải OUTER JOIN, và LEFT cho Table nằm bên trái). Ví dụ trước sử dụng LEFT

OUTER JOIN để chọn tất cả các dòng từ Table bên trái trong mệnh đề FROM (table Customers). Để chọn tất cả các dòng từ Table bên phải, bạn sử dụng RIGHT OUTER JOIN như trong ví dụ sau đây.

```
SÉLECT Customers.cust_id, Orders.order_num;  
FROM Customers RIGHT OUTER JOIN Orders;  
ON Orders.cust_id = Customers.cust_id;
```

SQL Sever cung cấp một cú pháp Outer Join bổ sung đơn giản. Để truy vấn một danh sách tất cả các khách hàng, bao gồm các khách hàng không đặt đơn đặt hàng nào, bạn có thể thực hiện như sau:

```
SELECT Customers.cust_id, Orders.order_num;  
FROM Customer, Orders;  
WHERE Customers.cust_id * = Orders.cust_id;
```

Kết quả

Cust_id	order_num
-----	-----
1000000001	20005
1000000001	20009
1000000002	NULL
1000000003	20006
1000000004	20007
1000000005	20008

- Ở đây điều kiện kết được chỉ định trong mệnh đề WHERE. Thay vì kiểm tra bằng với toán tử =, toán tử *= được sử dụng để cho biết rằng mỗi dòng trong Table Customers nên được đưa vào *= là toán tử Left Outer Join. Nó truy vấn tất cả các dòng từ Table bên trái.

- Ngược lại với Left Outer Join là Right Outer Join được chỉ định bởi toán tử =*. Nó có thể được sử dụng để trả về tất cả các dòng từ Table được liệt kê bên phải toán tử, như bạn thấy trong ví dụ sau:

```
SELECT Customers.cust_id, Orders.order_num;
FROM Customers, Orders;
WHERE Orders.cust_id =* Customers.cust_id;
```

- Còn một dạng khác của cú pháp Outer Join (được sử dụng chỉ bởi Oracle) đòi hỏi phải sử dụng toán tử + sau tên Table như sau:

```
SELECT Customers.cust_id, Orders.order_num;
FROM Customers, Orders;
WHERE Customers.cust_id (+) = Orders.cust_id;
```

- Các loại Outer Join không quan tâm đến các dạng Outer được sử dụng, luôn có hai dạng cơ bản của Outer Join - Left Outer Join và Right Outer Join. Khác biệt duy nhất giữa chúng là thứ tự các Table mà chúng đang tạo quan hệ. Nói cách khác, một phép kết Left Outer Join có thể trở thành một Right Outer Join đơn giản bằng cách đảo thứ tự các Table trong mệnh đề FROM hay WHERE. Thông thường, hai loại Outer Join này có thể được sử dụng thay thế cho nhau, và bạn quyết định chọn cái nào miễn là bạn cảm thấy thuận tiện.

- Có một dạng khác của Outer Join, và đó là dạng Outer Join đầy đủ vốn truy vấn tất cả các dòng từ cả hai Table. Không giống như phép kết Left Outer Join hay Right Outer Join, các phép kết này chỉ đưa vào các dòng không quan hệ từ một Table đơn, dạng phép kết Outer Join đầy đủ đưa vào các dòng không quan hệ cả hai Table. Cú pháp cho dạng kết Outer đầy đủ như sau:

```
SELECT Customers.cust_id, Orders.order_num;
FROM Orders FULL OUTER JOIN Customers ;
ON Orders.cust_id = Customers.cust_id;
```

** Sử dụng các phép kết với các hàm tập hợp*

Các hàm tập hợp được sử dụng để tổng kết dữ liệu. Mặc dù tất cả các ví dụ về các hàm tập hợp cho đến thời điểm này chỉ tóm tắt

dữ liệu từ một Table đơn, các hàm này còn có thể được sử dụng với các phép kết. Để minh họa vấn đề này, hãy xem qua một ví dụ dưới đây. Bạn muốn truy vấn một danh sách tất cả các khách hàng và số các đơn đặt hàng mà mỗi khách hàng đã đặt hàng. Đoạn mã sử dụng hàm COUNT để thực hiện điều này:

```
SELECT Customers.cust_id, ;
COUNT (Orders.order_num) AS num_ord;
FROM Customers INNER JOIN Orders;
ON Customers.cust_id = Orders.cust_id;
GROUP BY Customers.cust_id;
```

Kết quả

Cust_id	num_ord
-----	-----
1000000001	2
1000000003	1
1000000004	1
1000000005	1

- Câu lệnh SELECT sử dụng phép kết INNER JOIN để tạo quan hệ cho các Table Customers và Order với nhau. Mệnh đề GROUP BY nhóm dữ liệu theo khách hàng, và vì thế hàm này là COUNT (Orders.Order_num) đếm số lượng đơn đặt hàng cho mỗi khách hàng và trả nó về trong Num_ord.

Các hàm tập hợp có thể được sử dụng dễ dàng với các kiểu kết khác. Xem ví dụ sau:

```
SELECT Customers.cust_id, ;
COUNT (Orders.order_num) AS num_ord;
FROM Customers LEFT OUTER JOIN Orders;
ON Customers.cust_id = Orders.cust_id;
```


GROUP BY Customers.cust_id;

- Không có AS trong Oracle một lần nữa, những người sử dụng Oracle, hãy nhớ rằng phải bỏ AS.

Kết quả

Cust_id	num_ord
-----	-----
1000000001	2
1000000001	0
1000000003	1
1000000004	1
1000000005	1

- Ví dụ này sử dụng Left Outer Join để đưa vào tất cả các khách hàng, thậm chí khách hàng không đặt bất kỳ các đơn đặt hàng nào. Kết quả cho thấy rằng khách hàng là 1000000002 cũng được đưa vào, lần này với không đơn đặt hàng.

** Sử dụng các phép kết và các điều kiện kết*

Trước khi chúng ta thảo luận về các phép kết, tôi nghĩ là cần phải tóm tắt lại một số điểm chính về các phép kết và cách sử dụng chúng:

Hãy chú ý cẩn thận đến các kiểu kết đang được sử dụng. Thông thường, bạn sẽ muốn một phép kết Inner Join, nhưng thường cũng có các sử dụng hợp lý cho các phép kết Outer Join.

+ Kiểm tra tài liệu DBMS của bạn để có cú pháp chính xác mà nó hỗ trợ (hầu hết các DBMS sử dụng một trong các dạng cú pháp được tả trong hai bài học này).

+ Đảm bảo rằng bạn sử dụng điều kiện kết chính xác (không quan tâm đến cú pháp đang được sử dụng), hoặc bạn sẽ trả về dữ liệu sai.

+ Đảm bảo rằng bạn luôn cung cấp một điều kiện kết, bạn sẽ đưa kết luận sai.

+ Bạn phải đưa vào nhiều Table trong một phép kết và thậm chí có các loại phép kết khác cho mỗi Table. Mặc dù điều này là hợp lệ, đảm bảo rằng bạn đã thử nghiệm cho mỗi phép kết riêng rẽ trước khi kiểm tra chúng cùng lúc với nhau. Điều này sẽ giúp bạn khắc phục dễ dàng hơn nhiều sau này.

k. Kết hợp các Query

**** Tìm hiểu các Query kết hợp***

Hầu hết các Query SQL chứa một câu lệnh SELECT đơn vốn trả về dữ liệu từ một hoặc nhiều Table. SQL cũng cho phép bạn thực hiện nhiều câu Query (nhiều câu lệnh SELECT) và trả về các kết quả dưới dạng một tập phép kết Query đơn. Các Query kết hợp này thường được gọi là các UNION hoặc là các Query kết hợp.

Có hai tình huống mà bạn sử dụng các Query kết hợp:

+ Tương tự trả về dữ liệu có cấu trúc từ các Table khác trong một câu Query đơn.

+ Thực hiện nhiều câu Query đối với một Table đơn và trả về dữ liệu như là một Query.

- Kết hợp các Query và nhiều điều kiện WHERE: Trong hầu hết các phần, kết hợp hai Query cho cùng một Table sẽ thực hiện cùng một vấn đề như là một Query đơn với nhiều điều kiện của mệnh đề WHERE. Nói cách khác câu lệnh SELECT bất kỳ với nhiều mệnh đề WHERE cũng còn có thể được chỉ định dưới dạng một Query kết hợp, như bạn đã thấy trong phần sau.

**** Tạo các Query kết hợp***

Các Queries SQL được kết hợp bằng cách sử dụng toán tử UNION. Việc sử dụng UNION, nhiều câu lệnh SELECT có thể được chỉ định, và các kết quả của chúng có thể được kết hợp thành một tập kết quả đơn.

** Sử dụng UNION*

- Sử dụng UNION đủ đơn giản. Tất cả các bạn thực hiện là chỉ định mỗi câu lệnh SELECT và đặt từ khoá UNION giữa mỗi câu lệnh.

- Hãy xem qua một ví dụ. Bạn cần một báo cáo về tất cả các khách hàng của bạn trong Illinois, Idiana, và Michigan. Bạn còn muốn đưa vào tất cả các vị trí FUN4All, không quan tâm đến bạn. Tất nhiên, bạn có thể tạo một mệnh đề WHERE sẽ thực hiện điều này, nhưng lần này bạn sẽ sử dụng UNION thay vào đó.

- Như vừa giải thích, việc tạo một UNION dẫn đến viết nhiều câu lệnh SELECT. Đầu tiên hãy nhìn qua các câu lệnh riêng rẽ:

```
SELECT cust_name, cust_contact, cust_email;
```

```
FROM Customers ;
```

```
WHERE Customers_state IN ('IL', 'IN', 'MI');
```

Kết quả

Cust_name	cust_contact	cust_email
-----------	--------------	------------

Village Toys	John Smith	Sales@villagetoy.com
--------------	------------	----------------------

Fun4All	Jim Jones	jjones@fun4all.com
---------	-----------	--------------------

The Toy	Store Kim	Null
---------	-----------	------

```
SELECT cust_name, cust_contact, cust_email
```

```
FROM Customers
```

```
WHERE cust_name = 'Fun4All';
```

Kết quả

Cust_name	cust_contact	cust_email
-----------	--------------	------------

Fun4All	Jim Jones	jjones@fun4all.com
---------	-----------	--------------------

Fun4All	Denise	dstephens@
---------	--------	------------

- Câu lệnh SELECT đầu tiên truy vấn tất cả các dòng lệnh trong Illinois, Indiana, và Michigan bằng cách truyền các dạng viết tắt của những bang đó vào mệnh đề IN. Câu lệnh SELECT thứ hai sử dụng một phép kiểm tra bằng đơn giản để tìm các vị trí FUN4All.

Để kết hợp hai câu lệnh này, thực hiện như sau:

```
SELECT cust_name, cust_contact, cust_email
```

```
FROM Customers
```

```
WHERE cust_state IN ('IL', 'IN', 'MI')
```

```
UNION
```

```
SELECT cust_name, cust_contact, cust_email
```

```
FROM Customers
```

```
WHERE cust_name 'Fun4All';
```

Kết quả

Cust_name	cust_contact	cust_email
-----	-----	-----
Fun4All	Denise L.	dstephens@
Stephens	fun4all.com	dstephens@
Fun4All	Jim Jones	jjones@fun4all.com
Village	Toys John	Smith sales@
The Toy	Store Kim	Howard

- Các câu lệnh trước tạo thành từ hai câu lệnh SELECT trước đó được tách biệt bởi từ khoá UNION. UNION chỉ thị DBMS thực thi cả hai câu lệnh SELECT và kết hợp kết quả xuất vào một tập kết quả Query đơn.

Để tham khảo thêm, đây là cùng một Query sử dụng nhiều mệnh đề WHERE thay cho một UNION:

```
SELECT cust_name, cust_contact, cust_email  
FROM Customers  
WHERE cust_state IN ('IL', 'IN', 'MI')  
OR cust_name = 'Fun4All';
```

- Trong ví dụ đơn giản của chúng ta, UNION có thể thực sự phức tạp hơn việc sử dụng một mệnh đề WHERE. Nhưng với nhiều điều kiện lọc phức tạp, hoặc nếu dữ liệu đang được truy vấn từ nhiều Table (và không phải chỉ một Table đơn), UNION có thể thực hiện quá trình này đơn giản hơn nhiều.

- Các hạn chế UNION: Không có hạn chế SQL chuẩn nào cho số các câu lệnh SELECT vốn có thể được kết hợp với câu lệnh UNION. Tuy nhiên, hãy tham khảo tài liệu DBMS của bạn để đảm bảo rằng nó không hạn chế số câu lệnh tối đa.

- Hầu hết các DBMS tốt đều sử dụng một trình tối ưu Query bên trong để kết hợp các câu lệnh trước khi chúng được xử lý. Về lý thuyết, điều này có nghĩa là từ một bối cảnh thực thi, không có sự khác biệt thực sự nào giữa cách điều kiện của mệnh đề WHERE hoặc một UNION. Tôi nói theo lý thuyết, bởi vì, trong thực hành, hầu hết các trình tối ưu Query luôn luôn không thực hiện tốt một công việc như chúng cần nên làm. Cách tốt nhất là thử nghiệm cả hai phương pháp để xem phương pháp nào sẽ hoạt động tốt đối với bạn.

** Các quy luật UNION*

Như bạn có thể thấy, các UNION rất dễ sử dụng nhưng có một số quy luật cần nắm chính xác vốn có thể được kết hợp:

+ Một UNION phải có từ hai hay nhiều câu lệnh SELECT trở lên, mỗi câu lệnh phải được tách biệt bởi từ khoá UNION (vì vậy, nếu kết hợp bốn câu lệnh SELECT phải có ba từ khoá UNION được sử dụng).

+ Mỗi câu Query trong một UNION phải chứa cùng các cột, các biểu thức, hoặc các hàm tập hợp (mặc dù các cột không cần được liệt kê theo cùng thứ tự).

Các kiểu dữ liệu của cột phải tương thích: Chúng không cần phải cùng chính xác kiểu dữ liệu, nhưng chúng phải là một kiểu dữ liệu mà DBMS có thể chuyển đổi hoàn toàn (ví dụ, các kiểu dữ liệu số khác nhau hoặc các kiểu dữ liệu ngày khác nhau).

Ngoài những quy tắc cơ bản và những hạn chế này, các UNION có thể được sử dụng cho bất kỳ các tác vụ truy lục dữ liệu.

** Đưa vào hoặc loại các dòng thừa*

Trở lại phần trước "Sử dụng UNION" và hãy xem qua các câu lệnh SELECT mẫu được sử dụng. Bạn sẽ để ý rằng khi thực thi riêng biệt, câu lệnh SELECT đầu tiên trả về ba dòng, và câu lệnh SELECT thứ hai trả về hai dòng. Tuy nhiên, khi hai câu lệnh SELECT được kết hợp với một UNION, chỉ bốn dòng được trả về mà không phải năm dòng.

UNION tự động loại bỏ bất kỳ các dòng thừa khỏi tập kết quả Query (nói cách khác, nó thực hiện chỉ giống như thực hiện nhiều điều kiện của mệnh đề WHERE trong một câu lệnh SELECT đơn). Bởi vì có một vị trí `UNION ALL` trong UNION được sử dụng, dòng trùng này được loại bỏ.

Đây là hành vi mặc định của UNION, nhưng bạn có thể thay đổi điều này nếu bạn thực sự muốn. Nếu bạn muốn, thật ra, muốn tất cả số lần xuất hiện của các giá trị thoả mãn được trả về, bạn có thể sử dụng `UNION ALL` thay cho UNION.

Hãy xem qua ví dụ sau đây:

```
SELECT cust_name, cust_contact, cust_email
```

```
FROM customers
```

```
WHERE cust_state IN ('IL', 'IN', 'MI')
```

UNION ALL

SELECT cust_name, cust_contact, cust_email

FROM Customers

WHERE cust_name = 'Fun4All'

Kết quả

<i>Cust_name</i>	<i>cust_contact</i>	<i>cust_email</i>
-----	-----	-----
Village Toy	John Smith	Gettoys.com
Fun4All	Jim Jones	jjones@
The Toy	Store Kim	Howard NULL
Fun4All	Jim Jones	jjones@
Fun4All	Denise L.	dstephens

- Sử dụng UNION ALL, DBMS không loại bỏ các phần trùng. Vì vậy, ví dụ trước sẽ trả về 5 dòng, một trong chúng xuất hiện hai lần.

UNION luôn thực hiện cùng một vấn đề như nhiều điều kiện WHERE. UNION ALL là dạng của UNION vốn thực hiện những gì không thể thực hiện với mệnh đề mệnh WHERE. Nếu bạn muốn thực hiện, thực sự, muốn tất cả số lần xuất hiện của các giá trị thoả mãn cho mỗi điều kiện (đưa vào các phần trùng), bạn phải sử dụng UNION và không được sử dụng WHERE .

** Sắp xếp các phép kết Query kết hợp.*

Kết quả xuất của câu lệnh SELECT được sắp xếp bằng các sử dụng mệnh đề ORDER BY. Khi kết hợp các Query với một UNION chỉ có một mệnh đề ORDER BY có thể được sử dụng, và nó phải được đặt sau câu lệnh SELECT cuối cùng. Có một điểm rất nhỏ trong việc sắp xếp một phần của một tập kết quả theo một chiều khác, và vì vậy nhiều mệnh đề ORDER BY không được cho phép.

Ví dụ sau sắp xếp các kết quả được trả về bởi phép UNION được sử dụng trước đó.

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All'
ORDER BY cust_name, cust_contact;
```

Kết quả

Cust_name	cust_contact	cust_email
Fun4All	Denise L.	Stephens
Fun4All	Jim Jones	jjones@fun4all.com
The Toy	Store Kim	Howard
Village Toys	John Smith	sales@

- UNION này đưa ra một mệnh đề ORDER BY đơn sau câu lệnh SELECT cuối cùng. Mặc dù ORDER BY xuất hiện chỉ là một phần của các lệnh SELECT cuối cùng, DBMS thật ra sẽ sử dụng nó để sắp xếp tất cả các kết quả được trả về bởi tất cả các câu lệnh SELECT.

- Các loại UNION khác. Một số DNMS cung cấp hai loại bổ sung của UNION. EXCEPT (thỉnh thoảng được gọi là MINUS) có thể sử dụng để chỉ truy vấn các dòng vốn tồn tại trong Table đầu tiên không tồn tại trong Table thứ hai, và phần INTERSECT (phần giao) có thể được sử dụng để truy vấn chỉ các dòng vốn tồn tại trong cả hai Table. Thực tế, tuy nhiên, các kiểu UNION này ít khi

được sử dụng như cùng các kết quả có thể được thực hiện bằng cách sử dụng các phép kết.

1. Chèn dữ liệu

**** Tìm hiểu phép chèn dữ liệu***

SELECT là câu lệnh SQL thường được sử dụng nhất. Nhưng có ba câu lệnh SQL khác cũng thường được sử dụng mà bạn sẽ học. Câu lệnh đầu tiên là INSERT.

Như tên của nó đề nghị, Insert được sử dụng là để chèn (thêm) các dòng vào một Table cơ sở dữ liệu. Insert có thể được sử dụng theo nhiều cách.

- + Để chèn một dòng đơn đầy đủ.
- + Để chèn một phần của một dòng đơn.
- + Để chèn các kết quả của một Query.

Bây giờ bạn sẽ xem qua lần lượt các cách.

- INSERT và hệ thống bảo mật việc sử dụng câu lệnh Insert có thể đòi hỏi các đặc quyền bảo mật đặc biệt trong các DBMS Client-Server. Trước khi bạn cố sử dụng INSERT, đảm bảo rằng bạn có các quyền bảo mật thích đáng để làm điều này.

**** Chèn các dòng đầy đủ***

- Cách đơn giản nhất để chèn dữ liệu vào một Table là sử dụng cú pháp INSERT đơn giản, cú pháp này đòi hỏi rằng bạn phải chỉ định tên Table và các giá trị để được chèn vào dòng mới. Sau đây là một ví dụ:

```
INSERT INTO Customers  
VALUE ('10000000006',  
'Toy Land',  
'123 Any Street',
```

'New York',
'NY',
'11111'
'USA',
NULL,
NULL);

- Ví dụ trên chèn một khách hàng mới vào Table Customers. Dữ liệu được lưu trữ trong một cột Table được chỉ định trong mệnh đề VALUES và một giá trị phải được cung cấp cho mỗi cột. Nếu một cột không có giá trị (ví dụ, các cột Cust_contact và cust_email trên), giá trị NULL nên được sử dụng (giả sử Table cho phép không có giá trị được chỉ định cho cột đó). Các cột phải được đặt theo thứ tự mà chúng xuất hiện trong định nghĩa Table.

- Từ khoá INTO: Trong một số trình triển khai SQL, từ khoá INTO đứng sau INSERT là tùy ý. Tuy nhiên bạn nên luyện tập đưa ra từ khoá này mặc dù nó không cần. Làm như vậy sẽ bảo đảm rằng mã SQL của bạn có thể hợp lệ giữa các DBMS.

- Mặc dù cú pháp này thực sự đơn giản, nhưng nó không còn an toàn nữa và nên tránh.

- Câu lệnh SQL phụ thuộc khá nhiều vào thứ tự mà các cột được định nghĩa trong Table. Nó còn phụ thuộc vào thông tin về thứ tự đang có sẵn. Mặc dù nó có sẵn, nhưng không bảo đảm rằng các cột có cùng thứ tự chính xác mà lần sau Table được xây dựng lại. Vì vậy, việc viết các câu lệnh SQL phụ thuộc vào việc tạo thứ tự cột cụ thể rất không an toàn.

- Cách an toàn hơn để viết câu lệnh INSERT là như sau:

```
INSERT INTO Customers (cust_id,  
Cust_name,  
Cust_address,
```

```
Cust_city,  
Cust_state,  
Cust_zip,  
Cust_country,  
Cust_contact,  
Cust_email)  
VALUES ('10000000006',  
'Toy Land',  
'123 Any Street',  
'New York',  
'NY',  
'11111'  
'USA',  
NULL,  
NULL);
```

- Ví dụ này thực hiện chính xác điều tương tự như trong câu lệnh INSERT trước, nhưng lần này các tên cột được viết rõ ràng trong các dấu ngoặc đơn sau tên Table. Khi dòng được chèn vào, DBMS sẽ tìm mỗi món hàng thoả mãn trong các cột liệt kê với giá trị thích hợp trong danh sách VALUE. (Entry) Đầu tiên trong VALUE tương ứng với tên cột đầu tiên được chỉ định. Giá trị thứ hai tương ứng với tên cột thứ hai...

Bởi vì tên cột được đưa ra, VALUE phải tìm các tên cột cụ thể thoả trong thứ tự mà chúng được chỉ định, và không cần thiết theo thứ tự mà các cột xuất hiện trong Table thực. Thuận lợi của điều này là, mặc dù bố cục của Table thay đổi, câu lệnh INSERT vẫn làm việc chính xác.

Câu lệnh INSERT sau đưa ra tất cả các cột của dòng (như

trước), nhưng nó thực hiện theo một thứ tự khác. Bởi vì tên của các cột được chỉ định, phép chèn sẽ hoạt động chính xác:

```
INSERT INTO Customers (cust_id,  
Cust_contact,  
Cust_email,  
Cust_name,  
Cust_address,  
Cust_city,  
Cust_state,  
Cust_zip)  
VALUES ('10000000006',  
NULL,  
NULL,  
'Toy Land',  
'123 Any Street',  
'New York',  
'NY',  
'11111'),
```

- Luôn sử dụng một danh sách các cột theo quy tắc, không bao giờ sử dụng INSERT mà không chỉ định rõ ràng danh sách cột. Điều này sẽ tăng đáng kể khả năng SQL của bạn tiếp tục hoạt động khi gặp biến cố các thay đổi Table xảy ra.

- Sử dụng VALUE cẩn thận không quan tâm đến cú pháp của INSERT không sử dụng, số VALUE chính xác phải được chỉ định. Nếu không có tên cột nào được ra, một giá trị phải được đại diện cho mỗi cột Table. Nếu các tên của các cột được đưa ra, một giá trị phải đại diện cho mỗi cột được liệt kê. Nếu không có giá trị nào

được đại diện, một thông điệp lỗi sẽ phát sinh, và dòng đó sẽ không chèn vào.

Bạn nên sử dụng INSERT để chỉ định rõ ràng các tên cột của Table. Sử dụng cú pháp này, bạn cũng có thể bỏ qua các cột. Điều này có nghĩa là bạn chỉ cung cấp các giá trị cho một số cột, không cung cấp cho các cột khác.

```
INSERT INTO Customers (cust_id,  
Cust_name,  
Cust_address,  
Cust_city,  
Cust_state,  
Cust_zip,  
Cust_country)  
VALUES ('10000000006',  
'Toy Land',  
'123 Any Street',  
'New York',  
'NY',  
'11111'  
'USA');
```

- Trong ví dụ đưa ra trước đó trong bài học này, các giá trị không được cung cấp cho hai cột, Cust_contact và Cust_email. Điều này có nghĩa là không có lý do nào để đưa vào các cột đó trong câu lệnh INSERT. Câu lệnh INSERT, vì vậy, bỏ qua hai cột và hai giá trị tương ứng.

- Bỏ qua các cột bạn có thể bỏ các cột trong một hoạt động INSERT nếu định nghĩa Table cho phép. Một trong các điều kiện sau phải tồn tại:

- + Cột được định nghĩa dưới dạng cho phép các giá trị NULL (không có giá trị nào cả)

- + Một giá trị mặc định phải được chỉ định trong định nghĩa Table. Điều này có nghĩa là giá trị phải được sử dụng nếu không có giá trị nào được chỉ định.

Nếu bạn bỏ qua một giá trị từ một Table vốn không cho phép chép các giá trị NULL và có một giá trị mặc định, DBMS sẽ phát sinh một thông điệp lỗi và dòng sẽ không được chèn vào.

** Chèn dữ liệu được truy lục*

INSERT thường được sử dụng để thêm một dòng vào một Table bằng cách sử dụng các giá trị cụ thể. Có một dạng khác của INSERT vốn có thể được sử dụng để chèn kết quả của một câu lệnh SELECT vào một Table. Điều này được gọi là INSERT SELECT, và như tên của nó đề nghị, nó được tạo thành từ một câu lệnh INSERT và một câu lệnh SELECT.

Giả sử bạn muốn nối các danh sách khách hàng từ một Table khác vào trong Table khách hàng của bạn. Thay vì đọc một dòng ở một thời điểm và chèn nó với INSERT, bạn có thể thực hiện như sau:

- Các chỉ thị cần cho ví dụ kế tiếp. Ví dụ sau nhập dữ liệu từ một Table Customers. Hãy thử ví dụ này, tạo Table CustNew trước. Dạng Table CustNew nên giống như Table Customers. Khi tạo CustNew, bảo đảm không sử dụng giá trị Cust_id vốn đã tạo trong Customers (hoạt động INSERT thường sẽ thất bại nếu các giá trị khoá chính trùng).

```
INSERT INTO Customers (cust_id,  
Cust_contact,  
Cust_email,  
Cust_name,
```

```

Cust_address,
Cust_city,
Cust_state,
Cust_zip,
Cust_country)
SELECT cust_id,
Cust_contact,
Cust_email,
Cust_name,
Cust_address,
Cust_city,
Cust_state,
Cust_zip,
Cust_country)
FROM CustNew;

```

- Ví dụ này sử dụng câu lệnh SELECT hãy nhập tất cả dữ liệu từ CustNew và Customers. Thay vì liệt kê các giá trị được đưa vào, câu lệnh SELECT truy vấn chúng từ CustNew. Mỗi cột trong SELECT tương ứng với một cột trong danh sách các cột được chỉ định. Câu lệnh này sẽ chèn bao nhiêu dòng? Điều đó phụ thuộc vào số dòng nằm trong Table CustNew. Nếu Table rỗng, không có dòng nào sẽ được chèn vào (không có lỗi phát sinh bởi vì hoạt động này vẫn hợp lệ). Nếu Table chứa dữ liệu, tất cả các dữ liệu này được chèn vào Table Customers.

- Các tên cột trong INSERT SELECT. Ví dụ này sử dụng cùng các tên cột trong cả hai câu lệnh INSERT và SELECT để hợp tác. Nhưng không có đòi hỏi rằng các tên cột phải phù hợp. Thật ra, các DBMS thậm chí không chú ý đến các tên cột được trả về bởi

SELECT. Ở đây, vị trí cột được sử dụng, vì thế cột đầu tiên trong SELECT (không quan tâm đến tên nó) sẽ được sử dụng cho cột Table được chỉ định đầu tiên...

- Câu lệnh SELECT được sử dụng trong một INSERT SELECT có thể đưa vào một mệnh đề WHERE để lọc dữ liệu được chèn vào.

- Chèn nhiều dòng INSERT chèn chỉ một dòng đơn. Để chèn nhiều dòng, bạn phải thực thi nhiều câu lệnh INSERT. Ngoại lệ của quy luật này là INSERT SELECT, vốn có thể được sử dụng để chèn nhiều dòng với một câu lệnh đơn - dù câu lệnh SELECT trả về những gì cũng được chèn vào bởi INSERT.

- * Sao chép một Table sang một Table khác.*

- Có một dạng khác của phép chèn dữ liệu vốn không sử dụng câu lệnh SELECT nữa. Để Copy nội dung của một Table vào một Table mới (một Table vốn được tạo nhanh chóng) bạn có thể sử dụng câu lệnh SELECT INTO.

- Không giống như INSERT SELECT, vốn thêm dữ liệu vào Table đã có sẵn, SELECT INTO sao chép dữ liệu vào một Table mới (và phụ thuộc vào DBMS đang sử dụng, có thể ghi đè Table này nếu đã có).

Ví dụ sau minh hoạ cách sử dụng SELECT INTO

*SELECT **

INTO CustCopy

FROM Customers;

- Câu lệnh SELECT tạo một Table mới có tên là CustCopy và sao chép toàn bộ nội dung của Table Customers vào trong nó. Bởi vì SELECT * được sử dụng, mỗi cột trong Table Customers sẽ được tạo (và đưa vào) trong Table CustCopy. Để sao chép chỉ tập con có sẵn, tên các cột rõ ràng có thể được chỉ định thay vì ký tự đại diện *.

My SQL và Oracle sử dụng cú pháp.

CREATE Table CustCopy AS

*SELECT **

FROM Customers;

Đây là một số vấn đề cần xem xét khi sử dụng SELECT INTO.

+ Bất kỳ các tùy chọn và mệnh đề SELECT có thể được sử dụng bao gồm WHERE và GROUP BY.

+ Các phép kết có thể được sử dụng để chèn dữ liệu từ nhiều Table.

+ Dữ liệu chỉ có thể được chèn vào một Table không quan tâm đến số Table mà dữ liệu được truy vấn từ các Table đó.

- Tạo các bản sao của Table SELECT INTO là một cách tốt để tạo các bản sao Table trước khi thử nghiệm với các câu lệnh SQL mới. Bằng cách tạo một bản sao trước hết, bạn có thể thử nghiệm SQL của bạn trên bản sao đó thay cho dữ liệu sống.

- Trong bài học này bạn đã học cách chèn các dòng vào một Table cơ sở dữ liệu. Bạn đã học nhiều cách để sử dụng INSERT, và tại sao phép chỉ định cột rõ ràng được ưu tiên. Bạn cũng đã học cách sử dụng INSERT SELECT để nhập các dòng từ một Table khác, và cách sử dụng SELECT INTO để xuất các dòng sang một Table mới.

m. Cập nhật và xoá dữ liệu

**** Cập nhật dữ liệu***

Để cập nhật (chỉnh sửa) dữ liệu trong Table lệnh Update được sử dụng. Update có thể được sử dụng theo hai cách:

+ Để cập nhật các dòng cụ thể trong một Table.

+ Để cập nhật tất cả dòng trong một Table.

Hãy xem qua mỗi cách sử dụng của chúng.

- Đừng bỏ qua mệnh đề *Where*, phải cẩn thận khi sử dụng lệnh *Update*, bởi vì nó cũng rất dễ cập nhật sai mỗi dòng trong *Table* của bạn.

- *Update* và bảo mật việc sử dụng câu lệnh *Update* có thể đòi hỏi các đặc quyền bảo mật đặc biệt trong các DBMS Client - server. Trước khi bạn cố sử dụng *Update*, đảm bảo rằng bạn có các đặc quyền bảo mật thích đáng để làm này.

Câu lệnh *Update* rất dễ sử dụng. Dạng cơ bản của một câu lệnh *Update* được tạo thành ba phần:

- + *Table* để cập nhật.
- + Các tên cột và các giá trị mới của chúng.
- + Điều kiện lọc vốn xác định các dòng nào nên được cập nhật.

Hãy xem qua một ví dụ đơn giản. Khách hàng 1000000005 bây giờ có một địa chỉ email và vì vậy *Record* của anh ta cần cập nhật. Câu lệnh sau khi thực hiện việc cập nhật này:

```
Update Customers  
Set cust_email = 'kim@thetoystore.com'  
Where cust_id = '1000000005'
```

Câu lệnh *Update* luôn bắt đầu với tên của *Table* đang được cập nhật. Trong ví dụ này, nó là *Table Customers*. Lệnh *Set* khi đó được sử dụng để gán giá trị mới cho một cột. Như được sử dụng ở đây, mệnh đề *Set* xác lập cột *cust_email* với giá trị cụ thể:

```
Set cust_email = 'kim@thetoystore.com'
```

Lệnh *Update* hoàn thành với mệnh đề *Where* vốn nói cho DBMS những dòng nào được cập nhật. Nếu không có mệnh đề *Where*, DBMS sẽ cập nhật tất cả các dòng trong *Table Customers* với địa chỉ email mới này - rõ ràng không phải là hiệu quả như mong muốn.

Cập nhật nhiều cột đòi hỏi một cú pháp hơi khác.

Update Customers

Set cust_contact = 'Sam Roberts'.

Cust_email = 'sam@toyland.com'

Where cust_id = '1000000006';

Khi cập nhật nhiều cột, chỉ một lệnh Set đơn được sử dụng, và một cặp column = value được tách biệt bởi một dấu phẩy không có dấu phẩy được chỉ định sau cột cuối cùng). Trong ví dụ này, cột cust_contact và cust_email cả hai sẽ được cập nhật cho khách hàng 1000000006.

- Sử dụng các Query con trong câu lệnh Update các Query con có thể được sử dụng trong câu lệnh Update, cho phép bạn cập nhật các cột với dữ liệu được truy lục với một câu lệnh Select.

- Từ khoá From: Một số trình triển khai SQL hỗ trợ mệnh đề FROM trong câu lệnh Update vẫn có thể sử dụng để cập nhật các dòng trong một Table với dữ liệu từ một Table khác.

- Để xoá một giá trị của cột, bạn có thể xác lập nó là Null (giả sử rằng Table của bạn được định nghĩa để cho phép các giá trị Null). Bạn có thể thực hiện điều này như sau:

Update Customers

Set cust_email = NULL

WHERE cust_id = '1000000005';

Ở đây từ khoá Null được sử dụng để lưu không có giá trị cho cột cust_email

** Xoá dữ liệu*

- Để xoá (loại bỏ) dữ liệu khỏi một Table, câu lệnh Delete được sử dụng. Delete có thể được sử dụng theo hai cách:

- + Để xoá các dòng cụ thể khỏi một Table.

- + Để xoá tất cả các dòng cụ thể khỏi một Table.

Bây giờ bạn sẽ xem qua lần lượt các cấp sử dụng.

- Delete và bảo mật việc sử dụng câu lệnh Delete có thể đòi hỏi các đặc quyền bảo mật đặc biệt trong các DBMS Client - sever. Trước khi bạn cố sử dụng Delete, để đảm bảo rằng bạn có các đặc quyền bảo mật thích hợp để làm điều này.

Update rất dễ sử dụng và chúng tôi bảo đảm với bạn rằng Delete thậm chí còn dễ sử dụng hơn.

Câu lệnh sau xoá một dòng đơn khỏi Table Customers:

```
DELETE FROM Customers  
WHERE cust_id = '1000000006'
```

Câu lệnh này đã tự giải thích. Delete From đòi hỏi rằng bạn chỉ định tên của Table mà từ nó dữ liệu được xoá. Mệnh đề WHERE lập những dòng sẽ được xoá. Trong ví dụ này, chỉ khách hàng 1000000006 sẽ được xoá. Nếu mệnh đề WHERE bị bỏ sót, câu lệnh này sẽ xoá mỗi khách hàng trong Table.

- Từ khoá FROM trong một số trình triển khai SQL, từ khoá FROM đứng sau Delete là tùy ý. Tuy nhiên, bạn nên tập thói quen sử dụng từ khoá này, mặc dù nó không cần thiết. Điều này sẽ bảo đảm rằng mã SQL có thể tương thích giữa các DBMS.

Delete không nhận các tên cột hoặc các ký tự đại diện Delete sẽ xoá toàn bộ các dòng.

- Các nội dung Table, không phải các câu Table câu lệnh Delete xoá các dòng khỏi Table, thậm chí tất cả các dòng khỏi Table. Nhưng Delete không bao giờ xoá bản thân Table.

- Xoá nhanh hơn nếu bạn thật sự muốn xoá tất cả các dòng khỏi Table, đừng sử dụng Delete. Thay vào đó, hãy sử dụng lệnh Truncate Table vốn thực hiện điều tương tự nhưng thực hiện nhanh hơn nhiều.

** Các hướng dẫn để cập nhật và xoá dữ liệu*

- Các lệnh Update và Delete được sử dụng trong phần trước tất

cả đều có các mệnh đề WHERE, và có một lý do tốt cho vấn đề này. Nếu bạn bỏ qua mệnh đề WHERE, Update hoặc Delete sẽ áp dụng cho mỗi dòng trong Table. Nói cách khác, nếu bạn thực thi một lệnh Update mà không có mệnh đề WHERE, mỗi dòng trong Table sẽ được cập nhật với các giá trị mới. Tương tự nếu bạn thực thi lệnh Delete mà không có mệnh đề WHERE, tất cả các nội dung của Table sẽ bị xoá.

Đây là một số cách luyện tập tốt mà nhiều nhà lập trình SQL phải tuân thủ:

- + Không bao giờ thực thi một lệnh Update hoặc lệnh Delete mà không có mệnh đề WHERE nếu bạn thật sự không dự định cập nhật và xoá một dòng.

- + Bảo đảm rằng mỗi Table có một khóa chính và sử dụng nó như mệnh đề WHERE bất cứ khi nào có thể (bạn có thể chỉ định các khoá chính riêng lẻ, nhiều giá trị, hoặc các dãy giá trị).

- + Trước khi bạn sử dụng mệnh đề WHERE với một lệnh Update hoặc một lệnh Delete, đầu tiên hãy thử nghiệm nó với một lệnh SELECT để đảm bảo rằng nó đang lọc các record - rất dễ viết các mệnh đề WHERE sai.

- + Sử dụng cơ sở dữ liệu áp đặt tính toàn vẹn tham chiếu để DBMS sẽ không cho phép xoá các dòng vốn có dữ liệu trong các Table khác có quan hệ với chúng.

- + Một số DBMS cho phép các nhà quản trị cơ sở dữ liệu áp đặt các hạn chế vốn tránh thực thi Update và Delete mà không có mệnh đề WHERE. Nếu DBMS của bạn hỗ trợ tính năng này, xem xét cách sử dụng chúng.

- Chú ý khi sử dụng SQL không có nút Undo. Hãy cẩn thận khi sử dụng Update và Delete, hoặc bạn sẽ cập nhật và xoá dữ liệu sai.

6. Một số bài tập mẫu

BÀI 1: CHO CƠ SỞ DỮ LIỆU NHƯ SAU

HANGHOA

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_HANG	Character	3	Mã hàng
2	TEN_HG	Character	20	Tên hàng

DAILY: Mỗi đại lý có một số thứ tự, tên và một địa chỉ.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	STT_DL	Number	3	Số thứ tự đại lý
2	TEN_DL	Character	20	Tên đại lý
3	DCHI_DL	Character	20	Địa chỉ đại lý

MUA: Mỗi một ngày, đại lý sẽ tổng kết xem đã mua những mặt hàng nào với số lượng và trị giá bao nhiêu.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	STT_DL	Number	3	Số thứ tự đại lý
2	MA_HANG	Character	3	Mã hàng
3	NGAY_MUA	Date	8	Ngày mua
4	SOLG_MUA	Number	6	Số lượng mua
5	TRIGIA_MUA	Number	10	Trị giá mua

BAN: Sau mỗi ngày, đại lý sẽ tổng kết xem đã bán được những mặt hàng nào với số lượng và trị giá bán là bao nhiêu.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	STT_DL	Number	3	Số thứ tự đại lý
2	MA_HANG	Character	3	Mã hàng
3	NGAY_BAN	Date	8	Ngày bán
4	SOLG_BAN	Number	6	Số lượng bán
5	TRIGIA_BAN	Number	10	Trị giá bán

Yêu cầu: Viết các câu hỏi sau dưới dạng ngôn ngữ hỏi SQL

1. Tìm những mặt hàng đã bán trong tháng 1/95 tại đại lý số 3.
2. Tìm những mặt hàng đã mua trước năm 1995 và có trị giá mua > 500000. .
3. Tìm tên và địa chỉ đại lý có mua bia Heineken.
4. Tìm tất cả các mặt hàng mà đại lý số 2 đã bán trong năm 1994.
5. Tìm tên những mặt hàng mà đại lý Vạn Lợi đã mua trước 01/01/95 và có số lượng mua lớn hơn 150.
6. Tìm những mặt hàng đã được mua và bán trong cùng một ngày ở cùng một đại lý.
7. Tìm tên và địa chỉ đại lý có tổng giá trị mua trong một ngày lớn hơn 700000.
8. Tìm tổng giá trị mua và tổng giá trị bán của mặt hàng Coca Cola ở đại lý Tân Hiệp Hưng.
9. Tìm đơn giá mua trung bình của bia Heineken và bia TIGER trên các đại lý.
10. Tìm đơn giá mua trung bình của bia Sài Gòn trên các đại lý.
11. Tìm tên, địa chỉ của đại lý và những mặt hàng có số lượng mua và số lượng bán bằng nhau trong cùng một ngày.
12. Tìm tổng thu nhập từng ngày trên từng đại lý.
13. Tìm tổng giá trị mua trong tháng 1/95 tại đại lý Vạn Lợi.
14. Tìm số mặt hàng có bán ở từng đại lý.
15. Tìm tên và địa chỉ của đại lý có bán nhiều mặt hàng nhất.

Giải

Câu 1:

sele dist a.ma_hang, a.ten_hg, b.ngay_ban;

```
from hanghoa a, ban b;  
where a.ma_hang=b.ma_hang;  
and left (dtoc (b.ngay_ban),2)='01';  
and b.stt_dl=3
```

Câu 2:

```
sele dist a.ma_hang, a.ten_hg, b.ngay_mua, b.trigia_mua;  
from hanghoa a, mua b;  
where a.ma_hang=b.ma_hang;  
and right (dtoc (b.ngay_mua),2)<'95';  
and b.trigia_mua>500000
```

Câu 3:

```
sele dist a.*;  
from daily a, mua b, hanghoa c;  
where a.stt_dl=b.stt_dl;  
and b.ma_hang=c.ma_hang;  
and upper (c.ten_hg)='BIA HEINEKEN'
```

Câu 4:

```
sele dist a.ma_hang, a.ten_hg, b.ngay_ban;  
from hanghoa a, ban b;  
where a.ma_hang=b.ma_hang;  
and b.stt_dl=2;  
and right (dtoc (b.ngay_ban),2)='94';
```

Câu 5:

```
sele dist a.ma_hang, a.ten_hg, b.ngay_mua, b.solg_mua;  
from hanghoa a, mua b;  
where a.ma_hang=b.ma_hang;
```



```
and right (dtoc (b.ngay_mua),2)<'95';
group by b.ma_hang;
having sum (b.solg_mua)>150
```

Câu 6:

```
sele dist a.ma_hang, a.ten_hg, b.ngay_mua as ngay;
from hanghoa a, mua b, ban c;
where a.ma_hang=b.ma_hang;
and a.ma_hang=c.ma_hang;
and b.stt_dl=c.stt_dl;
and b.ngay_mua=c.ngay_ban
```

Câu 7:

```
sele dist a.*,b.ngay_mua, sum (b.trigia_mua) as tong_mua;
from daily a, mua b;
where a.stt_dl=b.stt_dl;
group by b.stt_dl, b.ngay_mua;
having sum (b.trigia_mua)>700000
```

Câu 8:

```
sele sum (a.trigia_mua) as tong_mua,
sum (b.trigia_ban) as tong_ban;
from mua a, ban b, daily c, hanghoa d;
where a.stt_dl=b.stt_dl;
and a.stt_dl=c.stt_dl;
and a.ma_hang=b.ma_hang;
and a.ma_hang=d.ma_hang;
and upper (d.ten_hg)='COCA COLA';
and upper (c.ten_dl)='TAN HIEP HUNG'
```

Câu 9:

```
sele dist a.*;  
from daily a, ban b, hanghoa c;  
where a.stt_dl=b.stt_dl;  
and b.ma_hang=c.ma_hang;  
and upper (c.ten_hg)='BIA TIGER';  
and b.stt_dl in;  
(sele b.stt_dl;  
from daily a, ban b, hanghoa c;  
where a.stt_dl=b.stt_dl;  
and b.ma_hang=c.ma_hang;  
and upper (c.ten_hg)='BIA HEINEKEN')
```

Câu 10:

```
sele dist b.ten_hg, avg (a.trigia_mua) as tgia_tb;  
from mua a, hanghoa b;  
where a.ma_hang=b.ma_hang;  
and upper (b.ten_hg)='BIA SAIGON';  
group by b.ma_hang
```

Câu 11:

```
sele dist a.ten_dl, a.dchi_dl, b.ma_hang, b.ten_hg;  
from daily a, hanghoa b, mua c, ban d;  
where a.stt_dl=c.stt_dl;  
and a.stt_dl=d.stt_dl;  
and b.ma_hang=c.ma_hang;  
and b.ma_hang=d.ma_hang;  
and c.solg_mua=d.solg_ban;  
and c.ngay_mua=d.ngay_ban
```

Câu 12:

```
sele stt_dl as ma_dl, ngay_mua as ngay_mua,  
sum (trigia_mua) as chi into dbf tam;  
from mua;  
group by stt_dl, ngay_mua  
sele stt_dl as ma_dl, ngay_ban as ngay_ban,  
sum (trigia_ban) as thu into dbf tam1;  
from ban;  
group by stt_dl, ngay_ban  
sele a.ma_dl, a.ngay_mua as ngay, ;  
  (b.thu-a.chi) as thunhap;  
from tam a, tam1 b;  
where a.ma_dl=b.ma_dl;  
and a.ngay_mua=b.ngay_ban
```

Câu 13:

```
sele sum (b.trigia_mua) as tong_mua;  
from daily a, mua b;  
where a.stt_dl=b.stt_dl;  
and upper (a.ten_dl)='VAN LOI';  
and b.ngay_mua>={01/01/95};  
and b.ngay_mua<={01/31/95};  
group by a.stt_dl
```

Câu 14:

```
sele dist a.stt_dl as stt_dl, b.ten_dl as ten_dl, ;  
b.dchi_dl as dchi_dl, a.ma_hang as ma_hang ;  
into dbf tam;
```

from ban a, daily b;

where a.stt_dl=b.stt_dl

sele stt_dl, ten_dl,count (ma_hang) as so_mat_hg;

from tam;

group by stt_dl

Câu 15:

sele stt_dl, ten_dl,dchi_dl, count (ma_hang) ;

as so_mat_hg into dbf tam1;

from tam;

group by stt_dl ;

sele max (so_mat_hg) as max_so_mat_hg into dbf tam2;

from tam1 ;

sele a.stt_dl, a.ten_dl, a.dchi_dl;

from tam1 a, tam2 b;

where a.so_mat_hg=b.max_so_mat_hg

BÀI 2: CHO CƠ SỞ DỮ LIỆU NHƯ SAU

CGTRINH: Mỗi một công trình có một số thứ tự, tên công trình, địa chỉ, kinh phí thực hiện, thuộc về một chủ nhân nào đó, do một chủ thầu thi công, được khởi công vào một ngày nào đó ở một tỉnh thành nào đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	STT_CTR	Number	4	Số thứ tự công trình
2	TEN_CTR	Character	20	Tên công trình
3	DIACHI_CTR	Character	20	Địa chỉ công trình
4	TINH_THANH	Character	15	Tỉnh thành phố

5	KINH_PHI	Number	4	Kinh phí thực hiện (Triệu)
6	TEN_CHU	Character	20	Tên chủ công trình
7	TEN_THAU	Character	20	Tên chủ thầu
8	NGAY_BD	Date	8	Ngày bắt đầu thực hiện

CHUNHAN: Mỗi một chủ công trình có một tên và địa chỉ (giả sử tên của các chủ nhân là không trùng lặp với nhau)

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	TEN_CHU	Character	20	Tên chủ công trình
2	DCHI_CHU	Character	20	Địa chỉ chủ công trình

CHU_THAU: Mỗi chủ thầu (có thể là đơn vị hay cá nhân) có một tên (giả sử rằng các tên không trùng nhau), một số điện thoại và một địa chỉ liên hệ.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	TEN_THAU	Character	20	Tên chủ thầu
2	DCHI_THAU	Character	20	Địa chỉ chủ thầu
3	TEL	Character	6	Số điện thoại liên hệ

CONGNHAN: Mỗi một công nhân có một họ tên (giả sử không trùng nhau), năm sinh, năm vào nghề và một chuyên môn riêng

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	HOTEN_CN	Character	20	Họ và tên công nhân
2	NAMS_CN	Number	2	Năm sinh
3	NAM_VAO_N	Number	2	Năm vào nghề
4	CH_MON	Character	10	Chuyên môn

KTRUCSU: Mỗi công trình được thiết kế bởi một kiến trúc sư, mỗi kiến trúc sư có một tên riêng (giả sử các tên không trùng nhau), năm sinh, nơi tốt nghiệp và một địa chỉ liên hệ.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	HOTEN_KTS	Character	20	Họ và tên kiến trúc sư
2	NAMS_KTS	Number	2	Năm sinh kiến trúc sư
3	PHAI	Character	1	Phái
4	NOI_TN	Character	15	Nơi tốt nghiệp
5	DCHI_LL_KT	Character	30	Địa chỉ liên lạc

THAMGIA: Mỗi một công nhân có thể tham gia vào một công trình từ một ngày nào đó, trong một khoảng thời gian tính bằng số ngày. Một công nhân có thể tham gia vào nhiều công trình khác nhau và một công trình có thể có nhiều công nhân tham gia.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	HOTEN_CN	Character	20	Họ và tên công nhân
2	STT_CTR	Number	4	Số thứ tự công trình
3	NGAY_TGIA	Date	8	Ngày bắt đầu tham gia
4	SO_NGAY	Number	2	Số ngày tham gia công trình

THIETKE: Mỗi kiến trúc sư khi thiết kế một công trình nào đó sẽ được hưởng một mức thù lao tương ứng. Một kiến trúc sư có thể thiết kế nhiều công trình khác nhau và một công trình có thể do nhiều kiến trúc sư đồng thiết kế.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	HOTEN_KTS	Character	20	Họ và tên kiến trúc sư
2	STT_CTR	Number	4	Số thứ tự công trình
3	THU_LAO	Number	7	Tiền thù lao (triệu đồng)

Yêu cầu: Viết các câu hỏi sau dưới dạng ngôn ngữ hỏi SQL

1. Tìm danh sách các công trình được thực hiện ở Hà Nội.
2. Tìm danh sách các công nhân có chuyên môn hàn hoặc điện và làm việc cho công ty xây dựng số 6.
3. Tìm các công trình mà công nhân Nguyễn Hồng Vân tham gia.
4. Tìm các kiến trúc sư thiết kế đồng thời cho công ty xây dựng số 6 và Lê Văn Sơn.
5. Tìm họ tên kiến trúc sư, tên công trình mà họ đã thiết kế và có mức thù lao lớn hơn 7000000.
6. Tìm tổng công nhân cho mỗi công trình.
7. Tìm danh sách các công nhân có làm chung công trình với kiến trúc sư Lê Thanh Tùng.
8. Tìm danh sách các kiến trúc sư đã tốt nghiệp tại thành phố Hồ Chí Minh và có thiết kế công trình ở Thành phố Hồ Chí Minh.
9. Tìm danh sách các kiến trúc sư có thiết kế cho các công trình mà kinh phí lớn hơn 50 triệu.
10. Tìm họ tên, địa chỉ của kiến trúc sư trẻ tuổi nhất.
11. Tính tổng số tiền thiết kế công trình đối với từng kiến trúc sư.
12. Tìm họ tên, năm sinh, chuyên môn của các công nhân có nhiều tuổi nghề nhất.
13. Tính tổng số ngày làm việc trong công trình của từng công nhân trong năm 1994.
14. Tìm công trình có số công nhân tham gia nhiều nhất từ trước đến nay.
15. Cho biết họ tên, địa chỉ, phái của các kiến trúc sư chỉ thiết kế các công trình ở thành phố Hồ Chí Minh.

Giải

Câu 1:

```
sele stt_ctr, ten_ctr, diachi_ctr, kinh_phi;  
from cgtrinh;  
where upper (tinh_thanh) = 'HA NOI'
```

Câu 2:

```
sele dist a.hoten_cn, a.nams_cn, nam_vao_n;  
from congnhan a, cgtrinh b, thamgia c;  
where a.hoten_cn = c.hoten_cn;  
and b.stt_ctr = c.stt_ctr;  
and upper (b.ten_thau)='CTY XD SO 6';  
and upper (a.ch_mon)='DIEN';  
or upper (a.ch_mon)='HAN'
```

Câu 3:

```
sele a.stt_ctr, a.ten_ctr, a.diachi_ctr;  
from cgtrinh a, thamgia b;  
where a.stt_ctr = b.stt_ctr;  
and upper (b.hoten_cn)='NGUYEN HONG VAN'
```

Câu 4:

```
sele a.hoten_kts, b.nams_kts, b.dchi_ll_kt;  
from thietke a, ktrucs b, cgtrinh c;  
where a.hoten_kts= b.hoten_kts;  
and a.stt_ctr=c.stt_ctr;  
and upper (c.ten_thau)='CTY XD SO 6';  
and a.hoten_kts in;
```



```
(sele a.hoten_kts;  
from thietke a, ktrucsus b, cgtrinh c;  
where a.hoten_kts= b.hoten_kts;  
and a.stt_ctr=c.stt_ctr;  
and upper (c.ten_thau)='PHONG DICH VU SO XD')
```

Câu 5:

```
sele a.stt_ctr, b.ten_ctr, a.hoten_kts;  
from thietke a, cgtrinh b;  
where a.stt_ctr=b.stt_ctr;  
and a.thu_lao>7.00
```

Câu 6:

```
sele a.stt_ctr, b.ten_ctr, count (a.hoten_cn) as so_cn;  
from thamgia a, cgtrinh b;  
where a.stt_ctr=b.stt_ctr;  
group by a.stt_ctr
```

Câu 7:

```
sele a.stt_ctr, a.hoten_cn;  
from thamgia a, thietke b;  
where a.stt_ctr=b.stt_ctr;  
and upper (b.hoten_kts)='LE THANH TUNG'
```

Câu 8:

```
sele a.hoten_kts;  
from ktrucsus a, thietke b, cgtrinh c;  
where a.hoten_kts=b.hoten_kts;  
and b.stt_ctr=c.stt_ctr;
```

and upper (a.noi_tn)='TP HCM';
and upper (c.tinh_thanh)='TP HCM'

Câu 9:

sele a.stt_ctr, a.ten_ctr, b.hoten_kts, a.kinh_phi;
from cgtrinh a, thietke b;
where a.stt_ctr=b.stt_ctr;
and a.kinh_phi>50

Câu 10:

sele max (nams_kts) as namsinh into dbf tam;
from ktrucsu
sele hoten_kts, dchi_ll_kt;
from ktrucsu a, tam b;
where a.nams_kts=b.namsinh

Câu 11:

sele hoten_kts, sum (thu_lao) as sotien;
from thietke;
group by hoten_kts

Câu 12:

sele min (nam_vao_n) as nam into dbf tam;
from cong Nhan
sele a.hoten_cn, a.nams_cn, a.ch_mon;
from cong Nhan a, tam b;
where a.nam_vao_n=b.nam

Câu 13:

sele hoten_cn, sum (so_ngay) as so_ngay;

```
from thamgia;  
where right (dtoc (ngay_tgia),2)='94';  
group by hoten_cn
```

Câu 14:

```
sele a.stt_ctr, b.ten_ctr, count (a.hoten_cn)  
as so_cn into dbf tam;  
from thamgia a, cgtrinh b;  
where a.stt_ctr=b.stt_ctr;  
group by a.stt_ctr  
sele stt_ctr,max (so_cn) as cn_ctr into dbf tam1;  
from tam  
sele a.stt_ctr, a.ten_ctr, a.diachi_ctr;  
from cgtrinh a, tam1 b;  
where a.stt_ctr=b.stt_ctr
```

Câu 15:

```
sele a.hoten_kts, a.dchi_ll_kt, a.phai;  
from ktrucsua a, thietke b, cgtrinh c;  
where a.hoten_kts=b.hoten_kts;  
and b.stt_ctr=c.stt_ctr;  
and upper (c.tinh_thanh)='TP HCM'
```

BÀI 3: CHO CƠ SỞ DỮ LIỆU NHƯ SAU

BAIBAO: Mỗi bài báo có một số thứ tự, một tựa bài và thuộc về một thể loại nào đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	STT_BAI	Number	3	Thứ tự bài viết
2	MA_THLOAI	Character	4	Mã thể loại
3	TUA	Character	60	Tựa bài

BAO_TCHI: Mỗi tờ báo hay tạp chí có một mã số, một tên, một địa chỉ tòa soạn, số điện thoại và có thời gian phát hành định kỳ của báo đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_BAO_TC	Character	4	Mã báo, tạp chí
2	TEN	Character	20	Tên báo, tạp chí
3	DCHI_TS	Character	30	Địa chỉ tòa soạn
4	DINH_KY	Character	8	Định kỳ xuất bản
5	TEL	Character	7	Điện thoại liên lạc

DANG_BAI: Một bài báo hoặc tạp chí được đăng trên một tờ báo hay tạp chí ở một hay nhiều số và ở một số được đăng ở một trang bắt đầu nào đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_BAO_TC	Character	4	Mã báo, tạp chí
2	SO_BAO_TC	Character	7*	Số báo, tạp chí
3	STT_BAI	Number	3	Số thứ tự bài
4	TRANG	Number	3	Trang đăng bài

PHATHANH: Mỗi một số của một tờ báo hay tạp chí được phát hành vào một ngày duy nhất.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_BAO_TC	Character	4	Mã báo, tạp chí
2	SO_BAO_TC	Character	7	Số báo, tạp chí
3	NGAY	Date	8	Ngày phát hành

THELOAI: Mỗi thể loại báo tạp chí có một mã số và một diễn giải tương ứng.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA	Character	4	Mã thể loại
2	DGIAI	Character	18	Diễn giải ý nghĩa

VIETBAI: Một bài báo có một mã số và bút hiệu của các tác giả (giả sử bút hiệu không trùng nhau). Một bài báo có thể do nhiều tác giả viết và một tác giả có thể viết nhiều bài báo khác nhau.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	STT_BAI	Number	3	Số thứ tự bài báo
2	BUT_HIEU	Character	20	Bút hiệu

Yêu cầu: Viết các câu hỏi sau dưới dạng ngôn ngữ hỏi SQL

1. Tìm các bài báo và tạp chí của Tô Phan.
2. Tìm tựa, số trang của các bài báo trong tạp chí Kiến thức ngày nay số 156.
3. Tìm các bài viết của Nguyễn Trọng Vinh trên các báo tuần.
4. Tìm các bài thơ được đăng trên các báo trong tuần lễ từ 23/04 đến 30/04/95.
5. Tìm tên, định kỳ của các tạp chí chi xuất bản trong năm 94.
6. Tìm tựa các bài bút ký đăng trên báo tuổi Trẻ chủ nhật.
7. Tìm các bài viết của Tô Phan được đăng trên báo Cựu chiến binh.

8. Tìm tựa, số trang của các bài bút ký được phát hành trong năm 1994.
9. Tìm số bài báo đăng trong tạp chí Kiến thức ngày nay số 156.
10. Tìm tên các tạp chí có đăng xã luận.

Giải

Câu 1:

```
sele dist a.stt_bai, b.tua;  
from vietbai a, baibao b;  
where a.stt_bai=b.stt_bai;  
and upper (a.but_hieu)='TO PHAN'
```

Câu 2:

```
sele b.tua, c.trang;  
from bao_tchi a, baibao b, dang_bai c;  
where a.ma_bao_tc=c.ma_bao_tc;  
and b.stt_bai=c.stt_bai;  
and upper (a.ten)='KIEN THUC NGAY NAY';  
and c.so_bao_tc='156'
```

Câu 3:

```
sele a.ma_bao_tc, a.stt_bai, b.ten;  
from dang_bai a, bao_tchi b, vietbai c;  
where a.ma_bao_tc=b.ma_bao_tc;  
and a.stt_bai=c.stt_bai;  
and upper (c.but_hieu)='NG TRONG VINH';  
and upper (b.dinh_ky)='1 TUAN'
```

Câu 4:

```
sele a.ma_bao_tc, b.ngay, c.tua;  
from dang_bai a, phathanh b, baibao c;  
where a.ma_bao_tc=b.ma_bao_tc;  
and a.so_bao_tc=b.so_bao_tc;  
and a.stt_bai=c.stt_bai;  
and upper (c.ma_thloai)='T';  
and b.ngay>={04/23/95};  
and b.ngay<={04/30/95}
```

Câu 5:

```
sele a.ma_bao_tc, a.ten, a.dinh_ky;  
from bao_tchi a, phathanh b;  
where a.ma_bao_tc=b.ma_bao_tc;  
and right (dtoc (b.ngay),2)='94'
```

Câu 6:

```
sele a.tua;  
from baibao a, bao_tchi b, dang_bai c;  
where a.stt_bai=c.stt_bai;  
and b.ma_bao_tc=c.ma_bao_tc;  
and upper (a.ma_thloai)='BK';  
and upper (b.ma_bao_tc)='TTCN'
```

Câu 7:

```
sele a.stt_bai, a.trang, c.tua;  
from dang_bai a, vietbai b, baibao c;  
where a.stt_bai=b.stt_bai;
```

```
and b.stt_bai=c.stt_bai;  
and upper (a.ma_bao_tc)='CCB';  
and upper (b.but_hieu)='TO PHAN'
```

Câu 8:

```
sele a.ma_bao_tc, a.stt_bai, b.ngay, c.tua, a.trang;  
from dang_bai a, phathanh b, baibao c;  
where a.ma_bao_tc=b.ma_bao_tc;  
and a.so_bao_tc=b.so_bao_tc;  
and a.stt_bai=c.stt_bai;  
and upper (c.ma_thloai)='BK';  
and right (dtoc (b.ngay),2)='94'
```

Câu 9:

```
sele count (a.ten) as sobai;  
from bao_tchi a, dang_bai b;  
where a.ma_bao_tc=b.ma_bao_tc;  
and upper (a.ten)='KIEN THUC NGAY NAY';  
and b.so_bao_tc='156';  
group by a.ten, b.so_bao_tc
```

Câu 10:

```
sele b.ten;  
from baibao a, bao_tchi b, dang_bai c;  
where a.stt_bai= c.stt_bai;  
and b.ma_bao_tc=c.ma_bao_tc;  
and upper (a.ma_thloai)='XL'
```


BÀI 4: CHO CƠ SỞ DỮ LIỆU NHƯ SAU

NHA_CCAP: Mỗi nhà cung cấp có một mã số, tên, vốn ban đầu và trụ sở đặt tại một thành phố nào đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_NCC	Character	2	Mã nhà cung cấp
2	TEN_NCC	Character	15	Tên nhà cung cấp
3	VON	Character	2	Vốn ban đầu (triệu đồng)
4	TH_PHO	Character	15	Tphố đặt trụ sở

HANG_HOA: Mỗi một mặt hàng có một mã số, một tên hàng, màu sắc, trọng lượng.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_HANG	Character	2	Mã hàng hóa
2	TEN_HANG	Character	15	Tên hàng
3	MAU	Character	5	Màu sắc
4	TR_LUONG	Number	3	Trọng lượng hàng

DU_AN: Mỗi một dự án có một dự án, một tên và thuộc về một thành phố duy nhất.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_DU_AN	Character	2	Mã dự án
2	TEN_DU_AN	Character	15	Tên dự án
3	TH_PHO	Character	20	Thành phố

CUNG_UNG: Mỗi nhà cung cấp cung ứng mặt hàng nào đó với một số lượng xác định.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MA_NCC	Character	2	Mã nhà cung cấp

2	MA_HANG	Character	2	Mã hàng
3	MA_DU_AN	Character	2	Mã dự án
4	SO_LG	Number	4	Số lượng hàng

Yêu cầu: Viết các câu hỏi sau dưới dạng ngôn ngữ hỏi SQL

1. Tìm thông tin về các dự án ở thành phố Hồ Chí Minh.
2. Tìm mã hàng, tên hàng cung cấp cho dự án Tân Phú ở Cần Thơ.
3. Tìm tên, mã nhà cung ứng mặt hàng P1 cho dự án J1
4. Tìm tên các dự án, mã dự án do nhà cung cấp Tân Phú ở thành phố Hồ Chí Minh cung ứng.
5. Tìm màu của các mặt hàng do hãng Tân An ở Cần Thơ cung ứng.
6. Tìm các nhà cung cấp cung ứng đồng thời cho cả 2 dự án J1 và J2
7. Tìm danh sách cung cấp cung ứng mặt hàng màu trắng cho dự án có mã J2
8. Tìm số mặt hàng cung ứng cho từng dự án.
9. Tìm danh sách các hãng có cung ứng cho dự án ở chính thành phố của mình.
10. Tìm tổng số từng mặt hàng đã cung ứng cho các dự án trên từng nhà cung cấp.
11. Tìm mã của những hãng nào cung ứng cùng một mặt hàng cho tất cả các dự án.
12. Tìm tất cả các dự án không sử dụng bất cứ mặt hàng nào do hãng S3 cung ứng.
13. Tìm tất cả các dự án có sử dụng tất cả các mặt hàng do hãng Tân An ở Cần Thơ cung ứng.
14. Tìm tổng số lượng của từng mặt hàng do hãng ở Cần Thơ cung ứng.

15. Tìm các dự án chi sử dụng duy nhất các mặt hàng do hãng S2 cung ứng

Giải

Câu 1:

```
sele dist a.ten_du_an, b.ten_ncc, c.ten_hang, d.so_luong;  
from du_an a, nha_ccap b, hang_hoa c, cung_ung d;  
where a.ma_du_an=d.ma_du_an;  
and b.ma_ncc=d.ma_ncc;  
and c.ma_hang=d.ma_hang;  
and upper (a.th_pho)='TP HO CHI MINH'
```

Câu 2:

```
sele dist a.ma_hang, a.ten_hang;  
from hang_hoa a, du_an b, cung_ung c;  
where a.ma_hang=c.ma_hang;  
and b.ma_du_an=c.ma_du_an;  
and upper (b.ten_du_an)='TAN PHU';  
and upper (b.th_pho)='CAN THO'
```

Câu 3:

```
sele dist a.ma_ncc, a.ten_ncc;  
from nha_ccap a, cung_ung b;  
where a.ma_ncc=b.ma_ncc;  
and upper (b.ma_hang)='P1';  
and upper (b.ma_du_an)='J1'
```

Câu 4:

```
sele dist a.ma_du_an, a.ten_du_an;  
from du_an a, nha_ccap b, cung_ung c;  
where a.ma_du_an=c.ma_du_an;  
and b.ma_ncc=c.ma_ncc;  
and upper (b.ten_ncc)='TAN PHU';  
and upper (b.th_pho)='TP HO CHI MINH'
```

Câu 5:

```
sele dist a.ma_hang, a.ten_hang, a.mau;  
from hang_hoa a, nha_ccap b, cung_ung c;  
where a.ma_hang=c.ma_hang;  
and b.ma_ncc=c.ma_ncc;  
and upper (b.ten_ncc)='TAN AN';  
and upper (b.th_pho)='CAN THO'
```

Câu 6:

```
sele dist a.ma_ncc, a.ten_ncc;  
from nha_ccap a, cung_ung b;  
where a.ma_ncc=b.ma_ncc;  
and upper (b.ma_du_an)='J1';  
and a.ma_ncc in;  
(sele a.ma_ncc;  
from nha_ccap a, cung_ung b;  
where a.ma_ncc=b.ma_ncc;  
and upper (b.ma_du_an)='J2')
```

Câu 7:

```
sele dist a.ma_ncc, a.ten_ncc;  
from nha_ccap a, hang_hoa b, cung_ung c;
```

```
where a.ma_ncc=c.ma_ncc;  
and b.ma_hang=c.ma_hang;  
and upper (b.mau)='TRANG';  
and upper (c.ma_du_an)='J2'
```

Câu 8:

```
sele b.ma_du_an, b.ten_du_an, count (a.ma_hang) as so_mat_hg;  
from hang_hoa a, du_an b, cung_ung c;  
where a.ma_hang=c.ma_hang;  
and b.ma_du_an=c.ma_du_an;  
group by b.ma_du_an
```

Câu 9:

```
sele dist a.ma_ncc, a.ten_ncc;  
from nha_ccap a, du_an b, cung_ung c;  
where a.ma_ncc=c.ma_ncc;  
and b.ma_du_an=c.ma_du_an;  
and a.th_pho=b.th_pho
```

Câu 10:

```
sele dist c.ten_du_an, a.ten_ncc, b.ten_hang,  
sum (d.so_luong) as so_mat_hg;  
from nha_ccap a, hang_hoa b, du_an c, cung_ung d;  
where a.ma_ncc=d.ma_ncc;  
and b.ma_hang=d.ma_hang;  
and c.ma_du_an=d.ma_du_an;  
group by d.ma_hang, d.ma_du_an, d.ma_ncc
```

Câu 11:

```
sele count (ma_du_an) as ts into dbf tam ;  
from du_an  
sele a.ma_ncc, c.ten_ncc;  
from cung_ung a,tam b, nha_ccap c;  
where a.ma_ncc=c.ma_ncc;  
group by a.ma_ncc,a.ma_hang;  
having count (a.ma_du_an)=ts
```

Câu 12:

```
sele ma_du_an as ma_du_an into dbf tam1;  
from cung_ung;  
where upper (ma_ncc)='S3'  
sele dist a.ma_du_an, a.ten_du_an;  
from du_an a, cung_ung b, tam1 c;  
where a.ma_du_an=b.ma_du_an;  
and a.ma_du_an not in;  
(sele all c.ma_du_an;  
from tam1 c)
```

Câu 13:

```
sele ma_ncc into dbf tam3;  
from nha_ccap;  
where upper (ten_ncc)='TAN AN';  
and upper (th_pho)='CAN THO'  
sele dist ma_hang into dbf tam4;  
from cung_ung a, tam3 b;  
where a.ma_ncc=b.ma_ncc ;
```

```

sele count (ma_hang) as ts into dbf tam6;
from tam4 ;
sele a.ma_du_an,a.ma_hang into dbf tam5;
from cung_ung a,tam3 b,tam4 c;
where a.ma_ncc=b.ma_ncc;
and a.ma_hang=c.ma_hang
sele a.ma_du_an, c.ten_du_an;
from tam5 a,tam6 b, du_an c;
where a.ma_du_an=c.ma_du_an;
group by a.ma_du_an;
having count (a.ma_hang)=b.ts

```

Câu 14:

```

sele a.ma_hang, a.ten_hang, sum (b.so_luong) as tg_so_lg;
from hang_hoa a, cung_ung b, nha_ccap c;
where b.ma_ncc=c.ma_ncc;
and a.ma_hang=b.ma_hang;
and upper (c.th_pho)='CAN THO';
group by b.ma_hang, c.th_pho

```

Câu 15:

```

sele a.ma_du_an,b.ten_du_an;
from cung_ung a, du_an b;
where upper (ma_ncc)='S2';
and a.ma_du_an=b.ma_du_an;
and a.ma_du_an not in;
(sele ma_du_an;
from cung_ung;
where upper (ma_ncc)<>'S2')

```

BÀI 5: CHO CƠ SỞ DỮ LIỆU NHƯ SAU

MACB: Mỗi cán bộ có một mã số, họ tên và mã chức danh.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MACB	Character	4	Mã cán bộ
2	HOTEN	Character	25	Họ tên cán bộ
3	MACD	Character	6	Mã chức danh

MACD: Mỗi một chức danh có một mã số, một tên gọi, được quy định một số giờ chuẩn nào đó và có giá tiền của 1 tiết vượt giờ (số tiết vượt giờ chuẩn) ứng với chức danh này.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MACD	Character	6	Mã chức danh
2	CHUCDANH	Character	15	Tên chức danh
3	GIOCHUAN	Number	3	Số giờ chuẩn
4	TIEN_1TIET	Number	5	Giá tiền 1 tiết vượt giờ

GD95_96: Mỗi một cán bộ giảng dạy khi dạy một môn nào đó, cho một lớp nào đó sẽ có một số tiết quy định và được hưởng hệ số quy đổi giờ chuẩn tùy theo số lượng sinh viên có trong lớp đó. Một cán bộ giảng dạy có thể dạy một môn cho nhiều lớp khác nhau và cũng có thể dạy nhiều môn cho cùng một lớp.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MACB	Character	4	Mã cán bộ
2	MONDAY	Character	20	Môn giảng dạy
3	LOPDAY	Character	15	Lớp dạy
4	SOTIET	Number	3	Số giờ chuẩn
5	SISO	Number	3	Sĩ số lớp đang dạy
6	HESO	Number	3	Hệ số được hưởng

TUNG: Khi cán bộ có tạm ứng thì sẽ có thể hiện trong tập tin này với các dữ kiện như sau: Mã cán bộ, số thứ tự lần tạm ứng của cán bộ và số tiền tạm ứng trong mỗi lần đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	MACB	Character	4	Mã cán bộ
2	LAN	Number	2	Số thứ tự lần tạm ứng
3	SOTIEN	Number	8	Số tiền tạm ứng trong lần đang xem xét.

Yêu cầu: Viết các câu hỏi sau dưới dạng ngôn ngữ hỏi SQL

1. Tìm mã số, họ tên các cán bộ có dạy môn Foxpro.
2. Tìm các lớp, môn học do Nguyễn Khanh phụ trách.
3. Tìm các lớp, môn học và có số tiết lớn hơn 30 tiết và do cán bộ có chức danh giảng viên phụ trách.
4. Tìm tên, mã số của cán bộ có dạy cho cả hai lớp tin học và điện tử 19.
5. Tìm mã cán bộ, họ tên, chức danh của các cán bộ có dạy vượt giờ.
6. Tính tổng số tiền tạm ứng cho từng cán bộ (chỉ tính cho các cán bộ có tạm ứng).
7. Tìm họ tên, chức danh của các cán bộ có dạy cho lớp, mà lớp này có học với giáo viên Nguyễn Khanh.
8. Tìm họ tên, chức danh, tổng số tiền tạm ứng của cán bộ có dạy vượt giờ.
9. Tính số tiền thực lĩnh của cán bộ có dạy vượt giờ.
10. Tìm tất cả các cán bộ có số tiền tạm ứng trung bình trong từng đợt lớn hơn 500000.
11. Tìm họ tên, chức danh của các cán bộ vừa dạy cho lớp điện tử, vừa dạy cho lớp tin học 20 học lớp tin học 19 và có tổng số tiền tạm ứng lớn hơn 1000000

12. Giả sử mỗi tiết dạy trong giờ chuẩn đều được trả tiền với giá như sau:

Chức danh	Số tiền một tiết
Giảng viên chính	11.000đ
Giảng viên	9.000đ
GV trung học	7.500đ

Hãy tính thực lĩnh của các cán bộ giảng dạy.

13. Tìm mã số, họ tên của cán bộ giảng dạy có thực lĩnh lớn nhất.

14. Tìm danh sách các môn học, số tiết chuẩn cho từng môn do các giáo viên có chức danh giáo viên trung học đảm trách.

15. Tìm họ tên, chức danh của cán bộ giảng dạy có hệ số tiết cao nhất.

Giải

Câu 1:

```
sele dist a.macb, a.hoten;  
from macb a, gd95-96 b;  
where a.macb=b.macb;  
and upper (b.monday)='FOXPRO'
```

Câu 2:

```
sele dist a.lopday, a.monday;  
from gd95-96 a, macb b;  
where a.macb=b.macb;  
and upper (b.hoten)='NGUYEN KHANH'
```

Câu 3:

```
sele dist a.lopday, a.monday, a.sotiet;
```

```

from gd95-96 a, macd b, macb c;
where a.macb=c.macb;
and b.macd=c.macd;
and a.sotiet>30;
and upper (b.chucdanh)='GIANG VIEN'

```

Câu 4:

```

sele dist a.macb, a.hoten;
from macb a, gd95-96 b;
where a.macb=b.macb;
and lower (b.lopday)='dt19';
or lower (b.lopday)='th-dt19';
and b.macb in;
(sele b.macb;
from macb a, gd95-96 b;
where a.macb=b.macb;
and lower (b.lopday)='th19';
or lower (b.lopday)='th-dt19')

```

Câu 5:

```

sele dist a.macb, a.hoten, b.giochuan,
sum (c.sotiet*c.heso) as sotiet;
from macb a, macd b, gd95-96 c;
where a.macb=c.macb;
and a.macd=b.macd;
group by c.macb;
having sum (c.sotiet*c.heso)>b.giochuan

```

Câu 6:

```
sele a.macb, a.hoten, sum (b.sotien) as tamung;  
from macb a, t_ung b;  
where a.macb=b.macb;  
group by b.macb;  
having sum (b.sotien)>0
```

Câu 7:

```
sele dist a.macb, a.hoten, b.chucdanh;  
from macb a, macd b, gd95-96 c;  
where a.macb=c.macb;  
and a.macd=b.macd;  
and c.lopday in;  
(sele c.lopday;  
from macb a, macd b, gd95-96 c;  
where a.macb=c.macb;  
and a.macd=b.macd;  
and upper (a.hoten)='NGUYEN KHANH')
```

Câu 8:

```
sele a.giochuan, b.macb, sum (b.sotiet*b.heso)  
as gioday into dbf tam;  
from macd a, gd95-96 b,macb c ;  
where b.macb=c.macb;  
and a.macd=c.macd;  
group by b.macb;  
having sum (b.sotiet*b.heso)>a.giochuan  
sele a.macb, b.hoten, c.chucdanh, sum (d.sotien) as tamung;
```

```

from tam a, macb b, macd c, t_ung d;
where a.macb=b.macb;
and a.macb=d.macb;
and b.macd=c.macd;
group by d.macb

```

Câu 9:

```

sele a.macb as macb, b.giochuan as tiet, sum (a.sotiet* ;
a.heso) - b.giochuan as sgvuot into dbf tam1;
from gd95-96 a, macd b, macb c;
where a.macb=c.macb;
and b.macd=c.macd;
group by a.macb;
having sum (a.sotiet*a.heso)>b.giochuan
sele a.macb as macb, sum (a.sotien) as tamung
into dbf tam2;
from t_ung a, tam1 b;
where a.macb=b.macb;
group by a.macb
close Data
crea Table trtien;
(chucdanh C (20), tien n (7,2))
insert into trtien;
value ('GV Chinh',11000)
insert into trtien;
value ('Giang Vien',9000)
insert into trtien;

```

```

value ('GV Trung hoc',7500)
sele dist a.macb, e.hoten, (a.tiet*c.tien+a.sgvuot*
d.tien_1tiet-b.tamung) as thuclinh;
from tam1 a, tam2 b, trtien c, macd d, macb e;
where a.macb=b.macb;
and a.macb=e.macb;
and d.macd=e.macd;
and c.chucdanh=d.chucdanh

```

Câu 10:

```

sele a.macb, a.hoten, avg (b.sotien) as tb_tung;
from macb a, t_ung b;
where a.macb=b.macb;
group by b.macb;
having avg (b.sotien)>500000

```

Câu 11:

```

sele macb as macb, sum (sotien) as tamung into dbf tam;
from t_ung;
group by macb
sele a.macb, a.hoten, b.chucdanh;
from macb a, macd b, tam c, gd95-96 d;
where a.macb=c.macb;
and a.macb=d.macb;
and a.macd=b.macd;
and c.tamung>1000000;
and upper (d.lopday)='DT-TH19';
union;

```

```

(sele a.macb, a.hoten, b.chucdanh;
from macb a, macd b, tam c, gd95-96 d;
where a.macb=c.macb;
and a.macb=d.macb;
and a.macd=b.macd;
and c.tamung>1000000;
and upper (d.lopday)='DT-TH20')

```

Câu 12:

```

sele macb as macb, sum (sotien) as tamung into dbf tam;
from t_ung;
group by macb
sele a.macb as macb, b.giochuan as tiet, sum (a.sotiet* ;
a.heso) - b.giochuan as sgvuot into dbf tam1;
from gd95-96 a, macd b, macb c;
where a.macb=c.macb;
and b.macd=c.macd;
group by a.macb;
having sum (a.sotiet*a.heso)>b.giochuan
sele macb into dbf tam2;
from macb;
where macb not in;
(sele macb;
from tam1)
close Data
crea Table trtien;
(chucdanh C (20), tien n (7,2))

```

```

insert into trtien;
value ('GV Chinh',11000)
insert into trtien;
value ('Giang Vien',9000)
insert into trtien;
value ('GV Trung hoc',7500)
sele d.hoten, e.chucdanh, b.tiet * c.tien + b.sgvuot*
e.tien_1tiet-a.tamung as tlinh;
from tam a, tam1 b, trtien c, macb d, macd e;
where a.macb=b.macb;
and a.macb=d.macb;
and d.macd=e.macd;
and c.chucdanh=e.chucdanh;
union;
(sele d.hoten, e.chucdanh, sum (f.sotiet*f.heso)* ;
c.tien-a.tamung as tlinh;
from tam a, tam2 b, trtien c, macb d, macd e, ;
gd95-96 f;
where e.macd=d.macd;
and a.macb=b.macb;
and a.macb=d.macb;
and f.macb=b.macb;
and c.chucdanh=e.chucdanh;
group by f.macb)

```

Câu 13:

```

sele d.hoten, e.chucdanh, b.tiet*c.tien + b.sgvuot* ;

```



```

e.tien_1 tiet-a.tamung as tlinh into dbf tam3;
from tam a, tam1 b, trtien c, macb d, macd e;
where a.macb=b.macb;
and a.macb=d.macb;
and d.macd=e.macd;
and c.chucdanh=e.chucdanh;
union;
(sele d.hoten, e.chucdanh, sum (f.sotiet*f.heso)* ;
c.tien-a.tamung as tlinh;
from tam a, tam2 b, trtien c, macb d, macd e, ;
gd95-96 f;
where e.macd=d.macd;
and a.macb=b.macb;
and a.macb=d.macb;
and f.macb=b.macb;
and c.chucdanh=e.chucdanh;
group by f.macb)
sele max (tlinh) as max_tlinh into dbf tam4;
from tam3
sele a.hoten, a.chucdanh;
from tam3 a, tam4 b;
where a.tlinh=b.max_tlinh

```

Câu 14:

```

sele dist a.monday, a.sotiet;
from gd95-96 a, macd b, macb c;
where a.macb=c.macb;

```

and b.macd=c.macd;
and upper (b.chucdanh)='GV TRUNG HOC'

Câu 15:

sele dist a.hoten, b.chucdanh, max (c.heso);
from macb a, macd b, gd95-96 c;
where a.macb=c.macb;
and a.macd=b.macd;
group by c.macb

BÀI 6: CHO CƠ SỞ DỮ LIỆU NHƯ SAU

CLB: Mỗi câu lạc bộ có một mã số để phân biệt với các câu lạc bộ khác, một tên gọi và địa chỉ của nó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	CLB_MA	Number	2	Mã câu lạc bộ
2	CLB_TEN	Character	20	Tên câu lạc bộ
3	CLB_DC	Character	30	Địa chỉ câu lạc bộ

DOI: Mỗi đội có một mã số để phân biệt với các đội khác của cùng một câu lạc bộ. Mỗi đội có một lứa tuổi duy nhất và có một phái (nam hoặc nữ). Tất cả các vận động viên của đội phải có cùng lứa tuổi hoặc nhỏ hơn.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	CLB_MA	Number	2	Mã câu lạc bộ
2	DOI_MA	Number	2	Mã đội
3	LT	Character	3	Mã lứa tuổi
4	PHAI	Character	3	Phái của đội

TD: Mỗi trận thi đấu có một mã số để phân biệt với các trận

đấu khác được diễn ra trong cùng một phòng, vào một ngày và một giờ bắt đầu đã quy định trước.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	TD_MA	Character	2	Mã trận đấu
2	PHG_MA	Character	2	Mã phòng
3	TD_GIO	Character	4	Giờ bắt đầu trận đấu (2 số đầu chỉ giờ, 2 số sau chỉ phút)
4	TD_NGAY	Character	6	Ngày diễn ra trận đấu (theo dạng yymmdd)

DOI_TD: Quan hệ này cho biết đội nào của câu lạc bộ nào tham gia vào trận đấu này. Mỗi trận đấu là sự gặp gỡ giữa 2 đội nào đó của câu lạc bộ nào đó. Hai đội phải có cùng lứa tuổi và thuộc cùng một phái. Có thể là 2 đội khác nhau của cùng một câu lạc bộ. Có thể là hai đội khác nhau của 2 câu lạc bộ khác nhau (mặc dù chúng có mã số đội giống nhau).

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	TD_MA	Character	2	Mã trận đấu
2	DOI_MA	Character	2	Mã đội tham gia thi đấu
3	CLB_MA	Character	2	Mã câu lạc bộ

PHONG: Mỗi phòng có một mã số để phân biệt các phòng với nhau, một địa chỉ và tổng số sân nhất định, nơi có thể diễn ra trận đấu.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	PHG_MA	Character	2	Mã số phòng
2	PHG_DC	Character	5	Địa chỉ phòng
3	PHG_TSS	Number	2	Tổng số sân hiện có.

LUA_TUOI: Tương ứng với mỗi lứa tuổi sẽ có một thời gian thi đấu riêng dành cho nó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	LT	Character	3	Mã lứa tuổi
2	LT_TGTD	Number	3	Thời gian thi đấu tương ứng (tính theo phút)

VDV: Mỗi vận động viên có một mã số dùng để phân biệt với các vận động viên khác, có một tên, một địa chỉ, một phái, một lứa tuổi và thuộc về một câu lạc bộ nào đó.

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	VDV_MA	Character	3	Mã vận động viên
2	VDV_TEN	Character	10	Tên gọi vận động viên
3	VDV_DC	Character	30	Địa chỉ vận động viên
4	CLB_MA	Character	2	Mã số câu lạc bộ
5	PHAI	Character	3	Phái
6	LT	Character	3	Mã số lứa tuổi

VDV_DOI: Mỗi vận động viên sẽ đăng ký chơi cho một đội nào đó (của câu lạc bộ mà họ đang tham gia). Mỗi vận động viên có thể tham gia vào nhiều đội khác nhau nhưng phải phù hợp với lứa tuổi (vận động viên chỉ có thể tham gia vào đội có lứa tuổi bằng hoặc lớn hơn lứa tuổi của họ).

STT	FIELD NAME	TYPE	WIDTH	DIỄN GIẢI
1	VDV_MA	Character	3	Mã vận động viên
2	DOI_MA	Character	2	Mã đội mà vận động viên tham gia

Yêu cầu: Viết các câu hỏi sau dưới dạng ngôn ngữ hỏi SQL

1. Tìm danh sách các vận động viên của câu lạc bộ mã 45.

2. Tìm danh sách các vận động viên thuộc đội 3 của câu lạc bộ 27.
3. Tính số lượng các trận đấu diễn ra trong ngày 900616.
4. Tìm tên, mã số câu lạc bộ và mã các đội mà các vận động viên ở lứa tuổi tham gia.
5. Tìm mã phòng, địa chỉ phòng mà câu lạc bộ Maika có thi đấu ở đó.
6. Tìm thời gian kết thúc trận đấu có mã là 09.
7. Tìm danh sách các vận động viên có tham gia thi đấu trong ngày 900109.
8. Tìm tên và địa chỉ của vận động viên có thi đấu với vận động viên Tâm.
9. Tìm tên các vận động viên có tham gia thi đấu trong khoảng thời gian từ 10-17 giờ.
10. Tìm địa chỉ, tên câu lạc bộ có ít nhất là một đội tham gia vào trận đấu diễn ra tại phòng mã số 17 trong ngày 901206.
11. Tìm mã, tên các câu lạc bộ và mã các đội có quy tụ các vận động viên có lứa tuổi nhỏ hơn lứa tuổi của đội.
12. Tìm tên, địa chỉ của các vận động viên có chơi cho đội đối lại của đội có mã là 01 của câu lạc bộ 50.
13. Tìm danh sách các trận đấu diễn ra trong khoảng thời gian từ 12 giờ đến 16 giờ tại phòng 49 trong ngày 900805.
14. Tìm số tối đa các trận đấu diễn ra trong cùng một ngày.
15. Tìm số các trận đấu diễn ra trong cùng một ngày ở cùng một phòng.

Giải

Câu 1:

```
sele vdv.*;  
from vdv ;  
where clb_ma='45'
```

Câu 2:

```
sele a.*;  
from vdv a, doi b;  
where a.clb_ma=b.clb_ma;  
and b.doi_ma='03';  
and a.clb_ma='27'
```

Câu 3:

```
sele count (td_ma) as so_tran;  
from td;  
where td_ngay='900616'
```

Câu 4:

```
sele a.clb_ma, a.clb_ten, b.doi_ma;  
from clb a, doi b;  
where a.clb_ma=b.clb_ma;  
and upper (b.lt)='LT3'
```

Câu 5:

```
sele a.phg_ma, a.phg_dc;  
from phong a, clb b, td c, doi_td d;  
where a.phg_ma=c.phg_ma;  
and b.clb_ma=d.clb_ma;
```

```
and c.td_ma=d.td_ma;  
and upper (b.clb_ten)='MAIKA'
```

Câu 6:

```
sele str (val (left (a.td_gio,2)) + int (b.lt_tgtd/60)) as gio,  
str(mod(val(right (a.td_gio,2)) +mod(bl_tgtd,60),60))as phut;  
from td a, lua_tuoi b, doi c, doi_td d;  
where a.td_ma=d.td_ma;  
and b.lt=c.lt;  
and c.doi_ma=d.doi_ma;  
and c.clb_ma=d.clb_ma;  
and a.td_ma='09'
```

Câu 7:

```
sele a.*;  
from vdv a, td b, doi_td c;  
where a.clb_ma=c.clb_ma;  
and b.td_ma=c.td_ma;  
and b.td_ngay='900109'
```

Câu 8:

```
sele dist a.vdv_ten, a.vdv_dc;  
from vdv a, vdv_doi b, doi_td c;  
where upper (alltrim (a.vdv_ten))<>'TAM';  
and a.vdv_ma=b.vdv_ma;  
and b.doi_ma=c.doi_ma;  
and a.clb_ma=c.clb_ma;  
and c.td_ma in;  
(sele d.td_ma;
```

```

from vdv a, vdv_doi b, doi_td c, td d;
where upper (alltrim (a.vdv_ten))='TAM';
and a.vdv_ma=b.vdv_ma;
and b.doi_ma=c.doi_ma;
and a.clb_ma=c.clb_ma;
and c.td_ma=d.td_ma)

```

Câu 9:

```

sele dist a.vdv_ma, a.vdv_ten, a.vdv_dc;
from vdv a, td b, doi_td c, doi d, lua_tuoi e;
where b.td_ma=c.td_ma;
and a.clb_ma=c.clb_ma;
and d.lt=e.lt;
and a.clb_ma=d.clb_ma;
and d.doi_ma=c.doi_ma;
and val (left (b.td_gio,2))>=10;
and val (left (b.td_gio,2)) + int ( (val (right (b.td_gio,2))+ ;
e.lt_tgtd)/60)<=17

```

Câu 10:

```

sele dist a.*;
from clb a, td b, doi_td c;
where a.clb_ma=c.clb_ma;
and b.td_ma=c.td_ma;
and b.phg_ma='17';
and b.td_ngay='901206'

```

Câu 11:

```

sele dist a.*;

```



```

from clb a, doi b, vdv c;
where a.clb_ma=b.clb_ma;
and b.clb_ma=c.clb_ma;
and c.lt<b.lt

```

Câu 12:

```

sele dist a.vdv_ten, a.vdv_dc;
from vdv a, vdv_doi b, doi_td c;
where c.doi_ma<>'01';
and a.vdv_ma=b.vdv_ma;
and b.doi_ma=c.doi_ma;
and a.clb_ma=c.clb_ma;
and c.td_ma in;
(sele d.td_ma;
from vdv a, vdv_doi b, doi_td c, td d;
where c.doi_ma='01';
and c.clb_ma='50';
and a.vdv_ma=b.vdv_ma;
and b.doi_ma=c.doi_ma;
and a.clb_ma=c.clb_ma;
and c.td_ma=d.td_ma)

```

Câu 13:

```

sele dist a.*;
from td a, doi_td b, lua_tuoi c, doi d;
where a.td_ma=b.td_ma;
and c.lt=d.lt;
and b.doi_ma=d.doi_ma;
and b.clb_ma=d.clb_ma;

```

```

and a.phg_ma='49';
and a.td_ngay='900805';
and val (left (a.td_gio,2))>=12;
and val (left (a.td_gio,2))<=16;
union;
(sele dist a.*;
from td a, doi_td b, lua_tuoi c, doi d;
where a.td_ma=b.td_ma;
and c.lt=d.lt;
and b.doi_ma=d.doi_ma;
and b.clb_ma=d.clb_ma;
and a.phg_ma='49';
and a.td_ngay='900805';
and int ( ( int (val (a.td_gio)/100))*60+mod (val
(a.td_gio),100)+c.lt_tgtd)/60)>=12;
and int ( ( int (val (a.td_gio)/100))*60+mod (val
(a.td_gio),100)+c.lt_tgtd)/60)<=16)

```

Câu 14:

```

sele count (td_ma) as sotran, td_ngay as ngay into dbf tam;
from td;
group by td_ngay
sele max (sotran) as sotran, ngay;
from tam

```

Câu 15:

```

sele count (td_ma) as st_dau, td_ngay as ngay, phg_ma as phong;
from td;
group by td_ngay, phg_ma

```

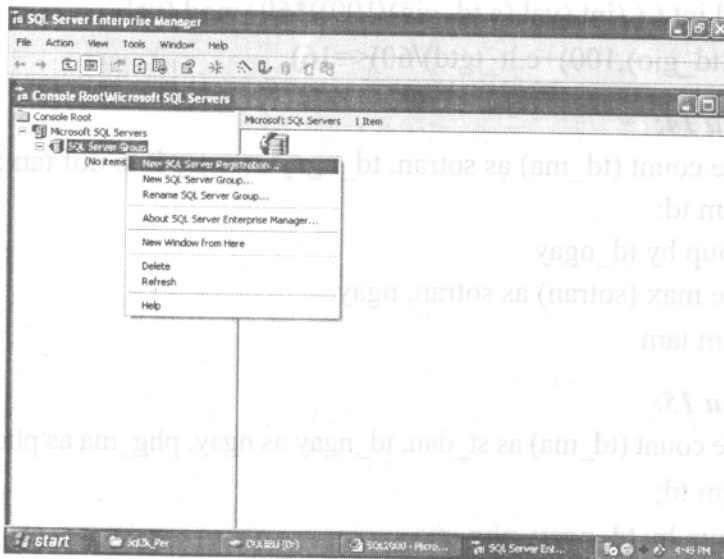
BÀI 3

LOGIN SQL SERVER, TẠO MỘT DATABASE ĐƠN GIẢN

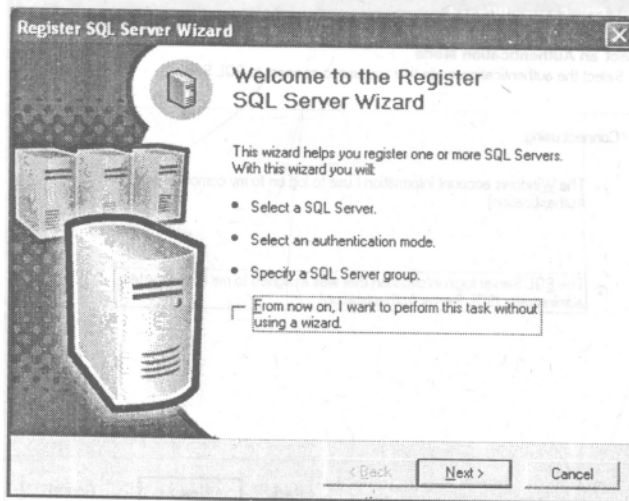
1. Login Server

Ở đây chúng ta sẽ tạo một Server ảo trên máy và Connect vào nó, để tạo thêm một Registration bạn có thể làm như sau:

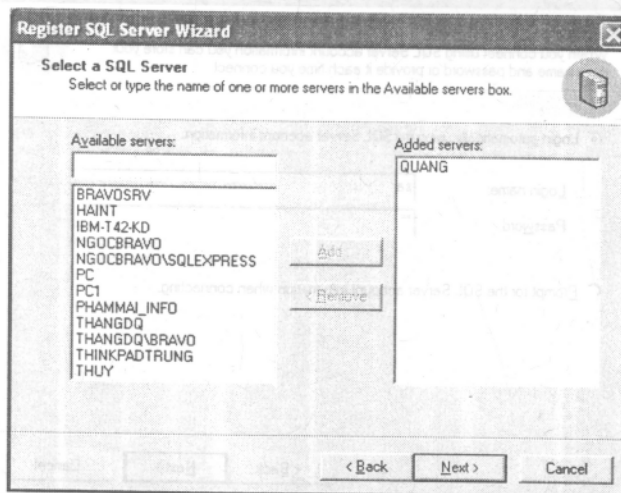
- Trong cửa sổ Enterprise Manager bạn nhấp chuột phải vào SQL Server Group/ New SQL Server Registration.



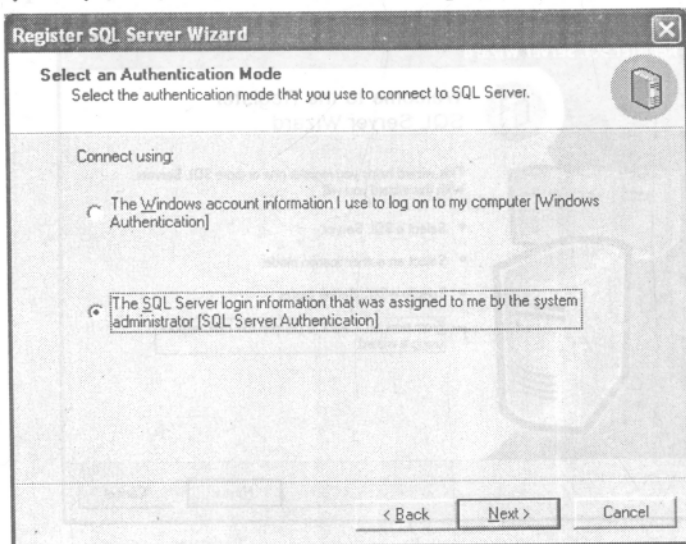
- Cửa sổ Register SQL Server Wizard xuất hiện bạn nhấp Next.



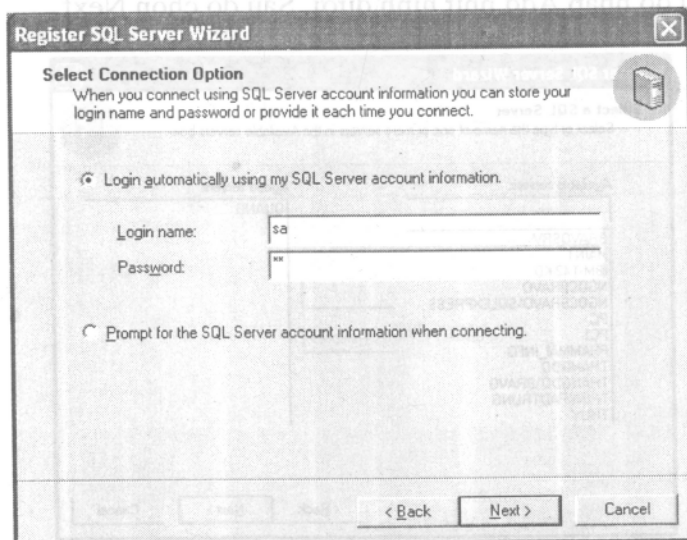
- Xuất hiện cửa sổ cho phép bạn gõ vào tên Server muốn Login, trong mục Available servers: Bạn gõ vào tên máy tính của mình sau đó nhấp Add như hình dưới. Sau đó chọn Next.



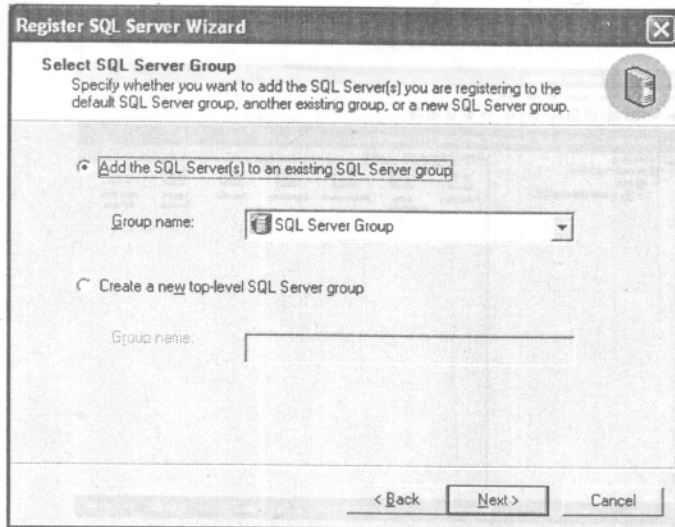
- Bạn chọn như hình dưới sau đó nhấn Next.



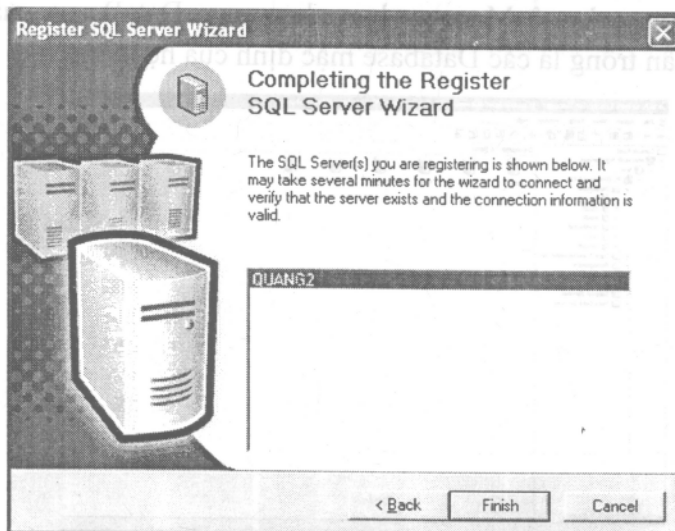
- Bạn chọn Login Automatically ...\ và gõ vào ô Login Name và Password là sa.



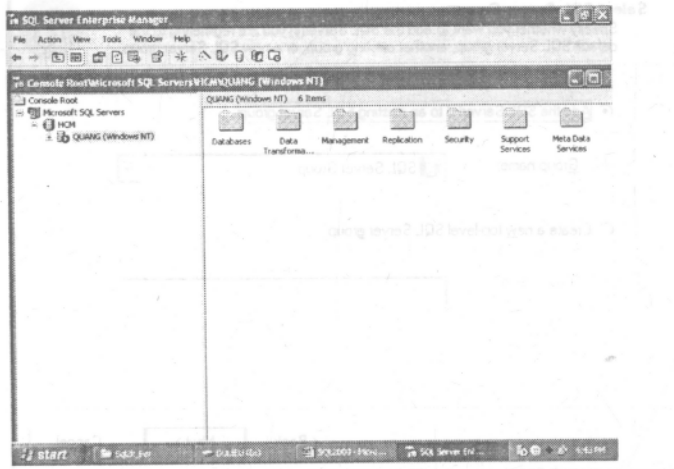
- Bạn nhấp Next.



- Chọn SQL Server Group, máy tính sẽ mặc định nếu chúng ta không có Group mới. Nhấp Next để chuyển bước tiếp.



- Cuối cùng bạn nhấp Finish để hoàn tất công việc. Sau khi Login xong bạn sẽ có một Server như hình dưới.



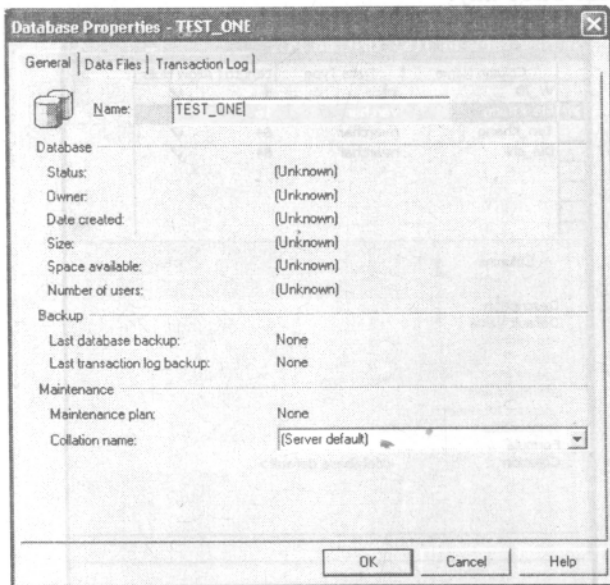
2. Tạo một Database đơn giản

a. Tạo Databases bằng Enterprise Manage

- Trong cửa sổ Manager bạn chọn mục DataBases xuất hiện trong phần trống là các Database mặc định của hệ thống.



- Bạn nhấp chuột phải vào phần trống trong cửa sổ chọn NewDataBases ... Cửa sổ DataBases Properties xuất hiện.

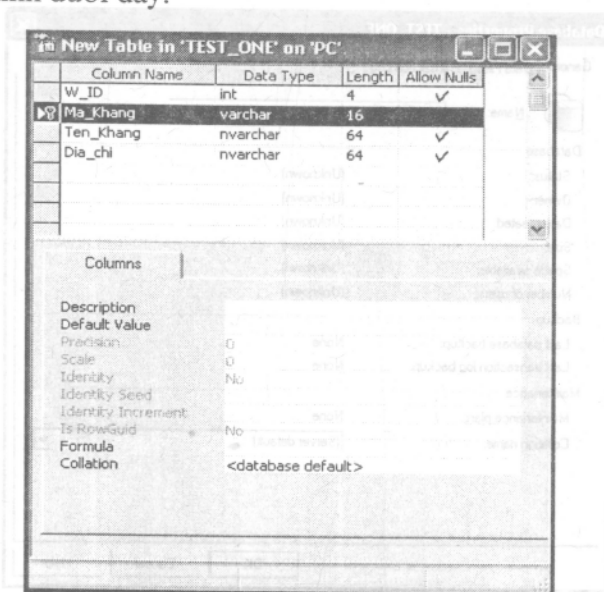


- Trong mục Name bạn gõ tên DataBases. Rồi nhấp OK, Sau bước này bạn đã có một dataBases với tên như trong ô Name bạn vừa gõ. Bây giờ trong cửa sổ DataBases bạn nhấp đúp vào DataBases vừa tạo cửa sổ thao tác sẽ xuất hiện. Trong đó bao gồm các thành phần của DataBases có Table, Trigger, Procedure, View..

- Bước tiếp theo chúng tôi sẽ hướng dẫn các bạn tạo một bảng đơn giản một thành phần chính không thể thiếu trong một cơ sở dữ liệu. Nơi sẽ lưu trữ toàn bộ các thông tin của CSDL.

- Bạn nhấp đúp vào biểu tượng Table. Cửa sổ làm việc với Table xuất hiện trong khung màn hình bạn sẽ thấy có rất nhiều bảng sẵn có đây là các bảng của hệ thống, và tất nhiên chúng không phải do chúng ta tạo ra.

- Bây giờ bạn hãy nhấp chuột phải lên phần trắng trong khung màn hình và chọn New Table.... Một cửa sổ New Table sẽ xuất hiện như hình dưới đây.



- Column Name: Khai báo các trường trong bản CSDL như hình trên chúng ta có bốn trường tất cả bao gồm: W_ID, Ma_Khang, Ten_Khang, Dia_Chi.

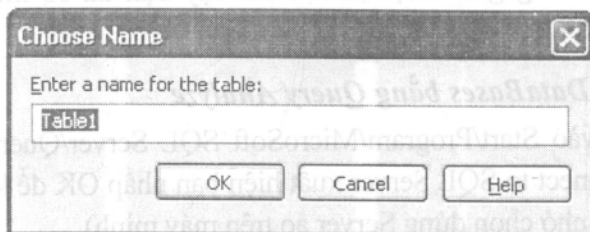
- DataType: Khai báo kiểu dữ liệu cho từng trường

- LengTh: Khai báo độ rộng của các trường

- AllNull: Nếu tích tức là bạn cho phép trường đó nhận giá trị rỗng khi nhập dữ liệu, như vậy khi một trường chúng ta luôn luôn muốn có dữ liệu thì chúng ta hãy bỏ dấu tích này đi.

- Đặt khóa cho bảng: Bạn nhấp chuột phải vào ô phía trái của trường muốn đặt làm khóa sau đó nhấp phải chuột và chọn Set Primary key. Để đặt nhiều khóa bạn đè phím Ctrt sau đó làm tương tự như trên.

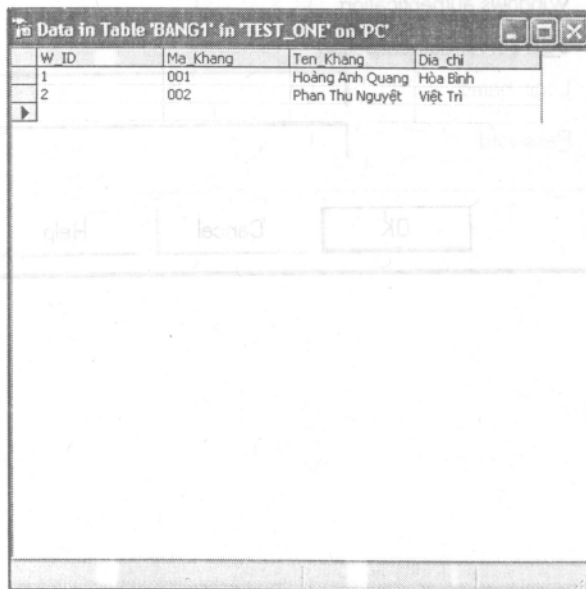
- Sau khi đã thiết kế đầy đủ dữ liệu bạn nhấn Save để lưu bảng lại. Chương trình sẽ hỏi bạn bằng cửa sổ dưới đây. Bạn hãy nhập tên cho bảng và nhấn OK.



A dialog box titled "Choose Name" with a close button (X) in the top right corner. It contains the text "Enter a name for the table:" followed by a text input field containing "Table1". At the bottom, there are three buttons: "OK", "Cancel", and "Help".

- Chỉnh sửa lại bảng dữ liệu: Sau khi đã thiết kế xong và ghi lại bảng nhưng do chưa ưng ý bạn hoàn toàn có thể chỉnh sửa lại bảng. Bạn nhấn phải chuột vào bảng muốn sửa chọn Design Table. Cửa sổ Design xuất hiện, bạn có thể thực hiện các thay đổi cần thiết.

- Thêm dữ liệu vào bảng: Trở lại cửa sổ thao tác với Table, bạn nhấn phải chuột vào bảng muốn thêm dữ liệu chọn Open Table/Return All Rows. Cửa sổ nhập liệu sẽ xuất hiện như hình dưới đây.



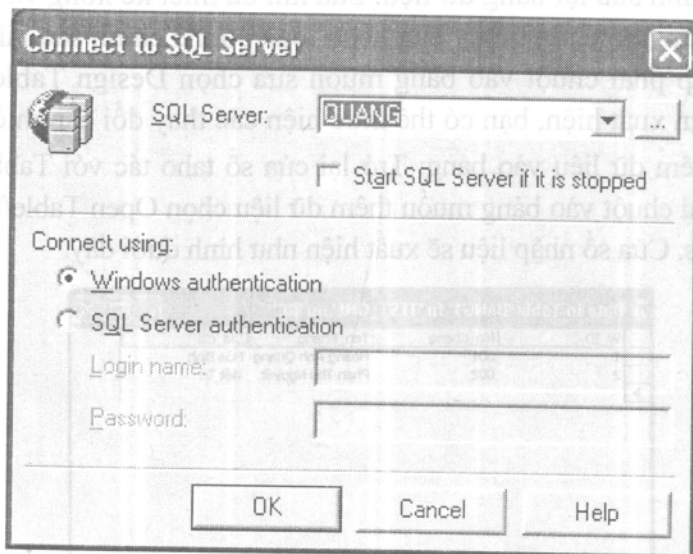
A window titled "Data in Table 'BANG1' in 'TEST_ONE' on 'PC'". It displays a table with 4 columns: "W ID", "Ma Khang", "Ten Khang", and "Dia chi". The table has 2 rows of data. Below the table, there are three buttons: "Help", "Cancel", and "OK".

W ID	Ma Khang	Ten Khang	Dia chi
1	001	Hoàng Anh Quang	Hòa Bình
2	002	Phan Thu Nguyệt	Việt Trì

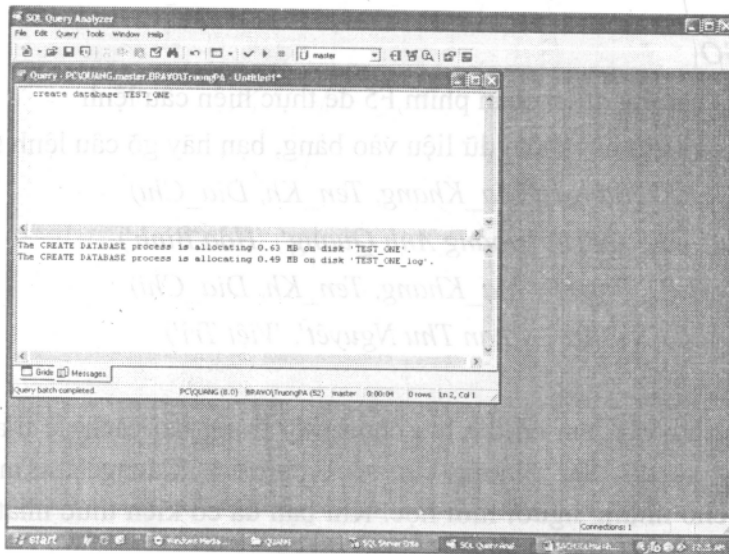
- Bạn lần lượt thêm dữ liệu cho các trường cần thiết các thao tác với bản ghi như Copy, xóa, sửa tương tự như trong Access nên chúng tôi không giới thiệu nữa. Như vậy bạn đã có một bảng dữ liệu hoàn chỉnh.

b. Tạo DataBases bằng Query Analyze

- Bạn vào Start/Program/MicroSoft SQL Server/Query Analyze. Cửa sổ Connect to SQL Server xuất hiện bạn nhập OK để kết nối vào Server (nên nhớ chọn đúng Server ảo trên máy mình)



- Cửa sổ Query xuất hiện



- Trong khung cửa sổ lệnh bạn gõ lệnh sau để tạo Databases *Create Database TEST_ONE*
- Sau đó nhấn phím F5 để thực thi câu lệnh
- Sau câu lệnh này một Databases có tên TEST_ONE sẽ được tạo.
- Bây giờ chúng ta sẽ tạo bảng cho Databases, bạn hãy gõ vào câu lệnh sau

USE TEST_ONE

CREATE Table [dbo].[Bang1]

(

W_ID [int] IDENTITY (1, 1) NOT NULL ,

Ma_KHang varchar (16) NOT NULL PRIMARY KEY,

Ten_KH nvarchar (64) NULL ,

Dia_chi nvarchar NULL

)

GO

- Và đừng quên nhấn phím F5 để thực hiện câu lệnh

- Bây giờ hãy thêm dữ liệu vào bảng, bạn hãy gõ câu lệnh sau:

INSERT Bang1 (Ma_Khang, Ten_Kh, Dia_Chi)

VALUES ('001', 'Hoàng Anh Quang', 'Hòa Bình')

INSERT Bang1 (Ma_Khang, Ten_Kh, Dia_Chi)

VALUES ('002', 'Phan Thu Nguyệt', 'Việt Trì')

GO

- Như vậy bạn có thể lựa chọn một trong hai cách để thao tác với cơ sở dữ liệu. Nhưng cửa sổ Enterprise Manage thường chỉ dùng cho những người mới học. Khi bạn đã có kiến thức nhất định với SQL thì bạn sẽ bỏ qua nó ngay.

BÀI 4

BACK UP & RESTORE, IMPORT SQL SERVER 2000

1. Backup & Restore SQL Server

a. Phục hồi dữ liệu

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 để 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.

b. 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.

c. 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 Committed.

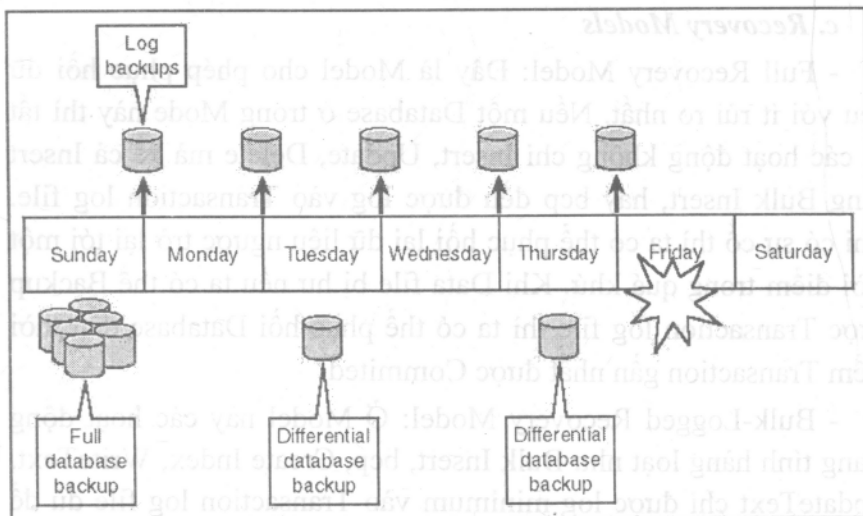
- Bulk-Logged Recovery Model: Ở Model này các hoạt động mang tính hàng loạt như Bulk Insert, bcp, Create Index, WriteText, UpdateText chỉ được log minimum vào Transaction log file đủ để cho biết là các hoạt động này có diễn ra mà không log toàn bộ chi tiết như trong Full Recovery Mode. Các hoạt động khác như Insert, Update, Delete vẫn được log đầy đủ để dùng cho việc phục hồi sau này.

- Simple Recovery Model: Ở Model 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 (xóa) những phần không cần dùng đến trong Transaction log file, đó là những Transaction đã được committed 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.

Kể 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).

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 Committed 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 Committed 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.

d. 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.

- 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:

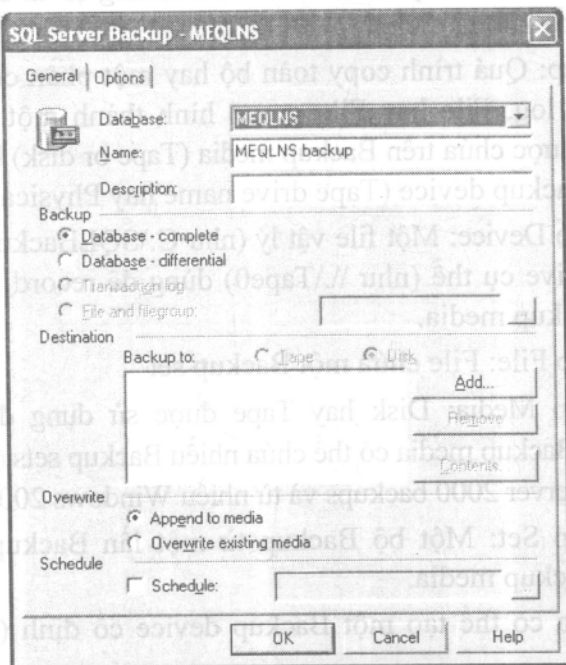
USE Master

Go

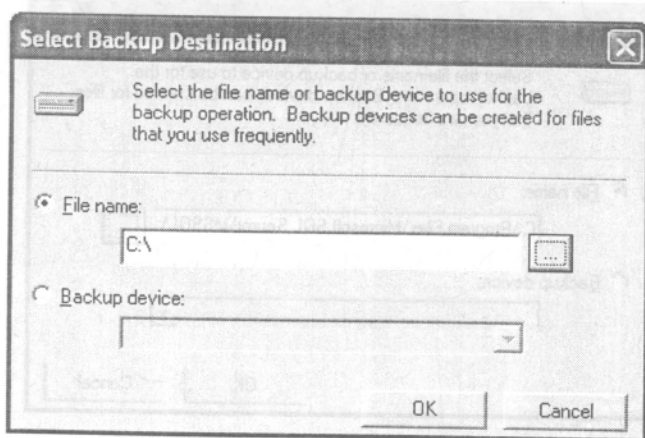
Sp_addumpdevice 'disk', 'FullBackupDevice',

'E:\SQLBackups\Full.bak'

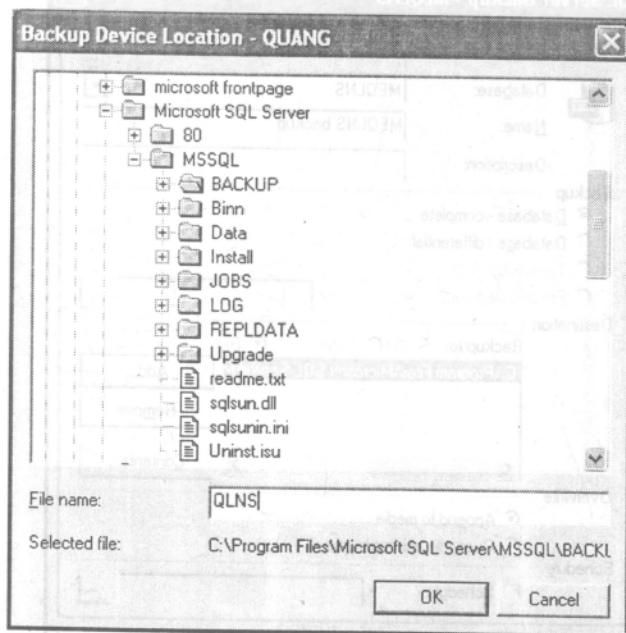
Để Backup Database bạn có thể dùng Backup Wizard hoặc click lên trên Database muốn Backup sau đó Right-click->All Tasks->Backup Database... sẽ hiện ra Window như hình vẽ sau:



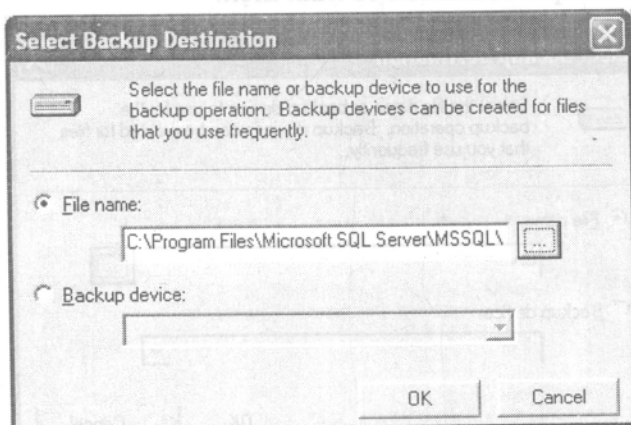
- Bạn nhấp Add để thêm vào đường dẫn chứa File Backup, cửa sổ Select Backup Destination sẽ xuất hiện.



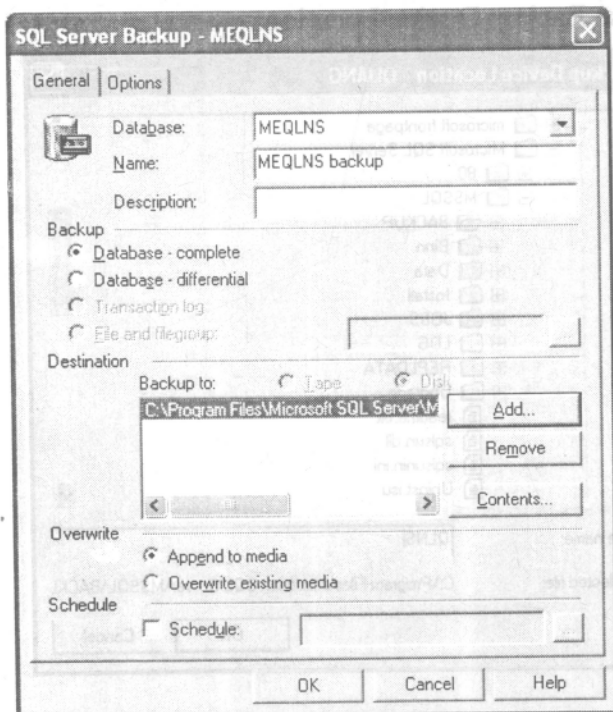
- Tiếp theo bạn bấm vào nút ... Sau đó hãy chọn đường dẫn trong cửa sổ Backup Device Location..



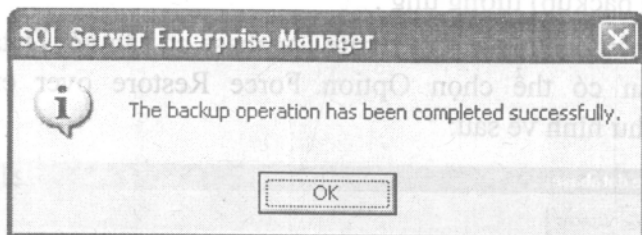
- Sau khi chọn xong ta sẽ được như hình dưới.



- Nhấp Ok để xác nhận.

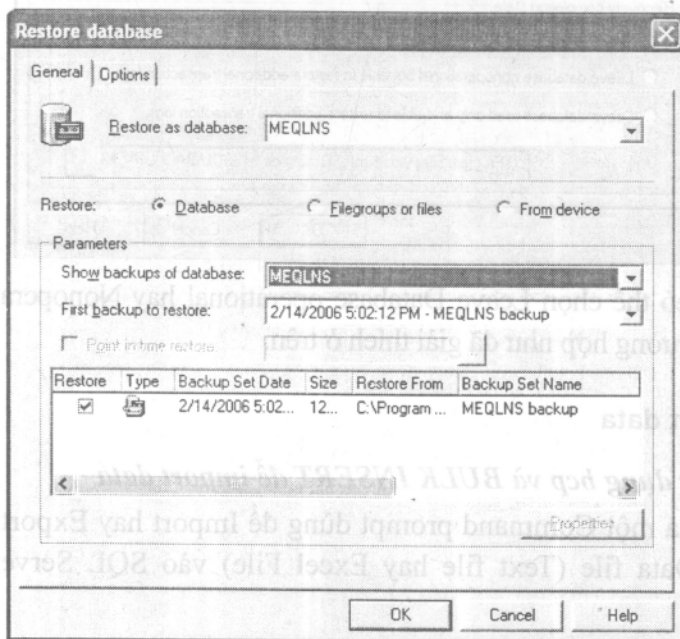


- Bạn nhấp Ok để hoàn tất nếu thành công hệ thống sẽ thông báo bằng hộp thoại dưới đây.



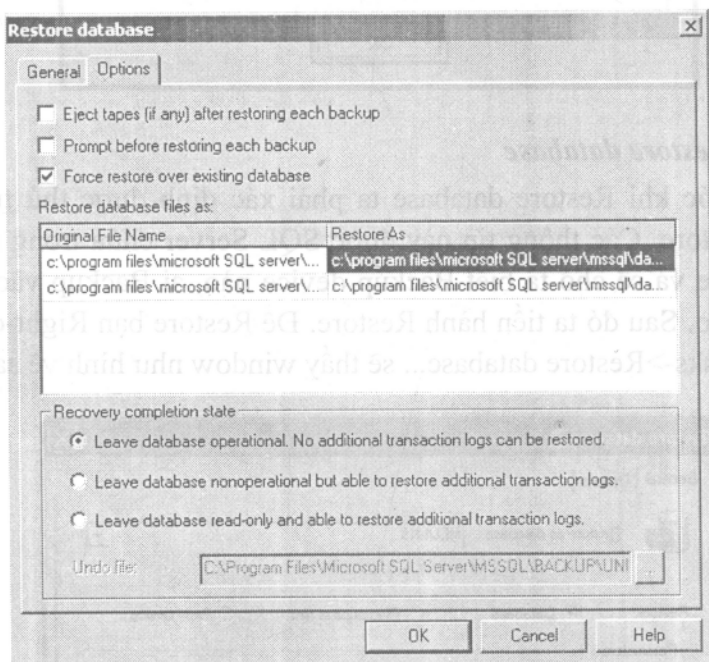
e. Restore database

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.

2 . Import data

a. 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:

```
hcp "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. Nếu 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.

b. 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 hai 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

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 tra 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.

BÀI 5

CẤU TRÚC DATABASE TRONG SQL SERVER 2000

1. 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 bản mẫu (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).

2. Cấu trúc vật lý của một SQL Server Database

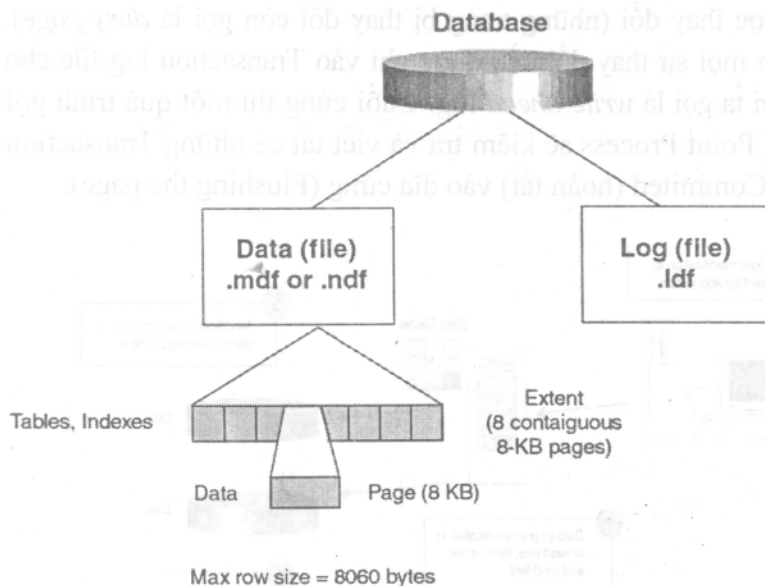
Mỗi một Database trong SQL Server đều chứa ít nhất một Data file chính (Primary), có thể có thêm một hay nhiều Data file phụ (Secondary) và một Transaction log file.

- 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 đĩa.

- 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.

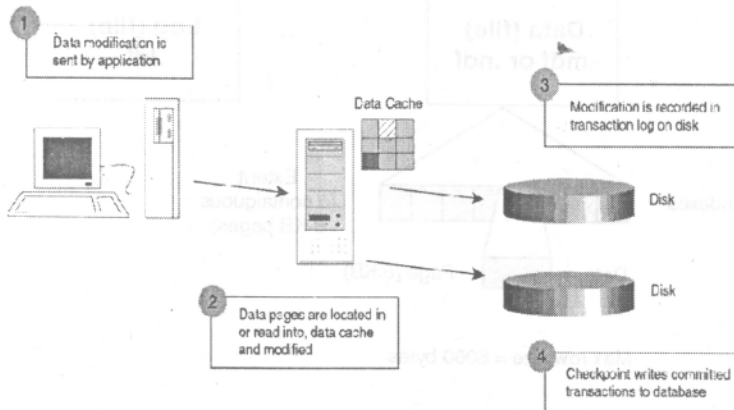
Data trong SQL Server được chứa thành từng Page 8KB và 8 Page liên tục tạo thành một Extent như hình vẽ dưới đây:



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.

3. 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 Committed (hoàn tất) vào đĩa cứng (Flushing the page).



Ngoài Check Point Process những Dirty-page còn được đưa vào đĩa bởi một Lazy writer.

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:

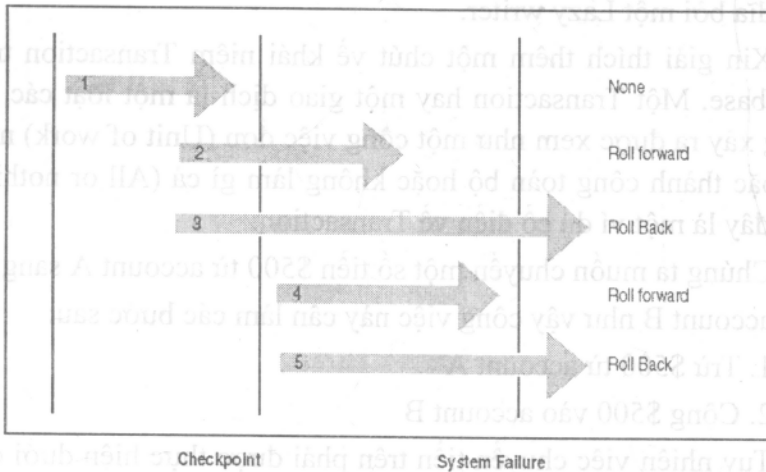
1. Trừ \$500 từ account A
2. Cộng \$500 vào account B

Tuy nhiên việc chuyển tiền trên phải được thực hiện dưới dạng một Transaction nghĩa là giao dịch chỉ được xem là hoàn tất (Committed) 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 xảy 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).

Transaction Recovery



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 đĩa rồi. Trong khi đó Transaction 2 và 4 sẽ được Roll forward vì tuy đã được Committed nhưng do sự cố xảy ra trước thời điểm Check point kế tiếp nên Data chưa kịp lưu vào đĩa. Tức là dựa trên những thông tin được ghi trên log file SQL Server hoàn toàn có đầy đủ cơ sở để viết vào đĩa cứng. Còn Transaction 3 và 5 thì chưa được Committed (do bị down bất ngờ) cho nên SQL Server sẽ Roll back hai Transaction này dựa trên những gì được ghi trên log file.

4. 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:

- 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.

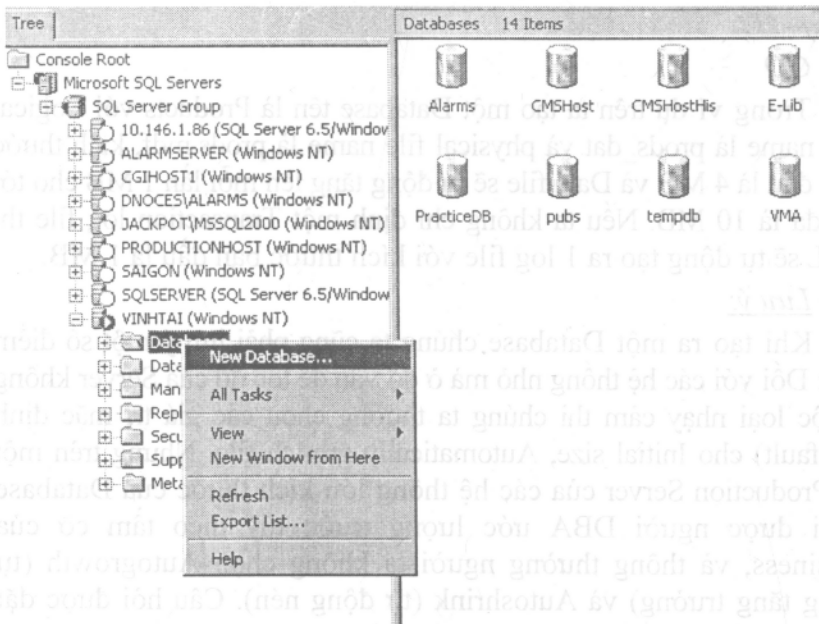
- Sp_monitor: Cho biết độ bận rộn của SQL Server

- 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

5. Cách 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 nói rộng ra mà lại phải ước lượng trước? Nguyên nhân là nếu chọn Autogrowth (hay Autoshrink) thì chúng ta có thể sẽ gặp hai 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 đĩa 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 đĩa 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 nói 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.

6. Những điểm cần lưu ý khi thiết kế 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 lặp 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 hai 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 Server 2000	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 mỗi quan hệ này thì một hàng bên Table A không thể liên kết với hơn một hàng bên Table B và ngược lại.

- One-to-Many Relationships: Trong mỗi quan hệ này thì một hàng bên Table A có thể liên kết với nhiều hàng bên Table B.

- Many-to-Many Relationships: Trong mỗi 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 mỗi 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.

BÀI 6

TÍNH TOÀN VỆ CỦA DỮ LIỆU

1. Các phương pháp bảo đảm toàn vẹn dữ liệu

SQL Server dùng một số cách để đảm bảo Data Integrity.

- 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 xử lý Null Data kỹ lưỡng bị "tê".

- 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 nghiên cứu kỹ 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.

2. 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 Databasc (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.

a. 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))
```

b. Unique Constraint

Bạn có thể tạo Unique Constraint để đảm bảo giá trị của một cột nào đó không bị trùng lặp. 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.

c. 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.

d. 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..."

BÀI 7

CURSORS, STORED PROCEDURE

1. 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 SELECT sẽ 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.

2. 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).

a. 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 gửi 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 gửi chỉ gửi 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 tra 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.

b. Các loại Stored procedure

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

- System Stored Procedure: 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.

- 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 procedure loại này trong phần kế tiếp.

- 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.

- 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 để gửi 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.

- Remote Stored procedure: Những Stored procedure gọi Stored procedure ở Server khác.

c. 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 Output 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.

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.

BÀI 8

TRIGGER & VIEW

1. 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.

a. Khi nào ta cần sử dụng Trigger

- Ta chỉ sử dụng Trigger khi mà các biện pháp bảo đảm Data integrity 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 Synchronize (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.

b. 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 hai 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.

- 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.

c. Cách tạo một Trigger

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.

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.

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

.....

END

Nếu muốn kiểm tra nhiều Columns ta có thể dùng hàm khác là Columns_Updated ().

2. View

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.

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 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 sử dụng 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.

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.

BÀI 9

VÍ DỤ VỀ CƠ SỞ DỮ LIỆU QUẢN LÝ NHÂN SỰ

1. Cơ sở dữ liệu quản lý nhân sự

- Danh mục tỉnh thành

Tên: DMTT

Khoá: Ma_Tthanh

Fields Name	Data Type	Fields Size	Description
W_ID	Int	4	Trường tự động
Ma_Tthanh	Varchar	16	Mã tỉnh thành
Ten_TThanh	Nvarchar	64	Tên tỉnh thành
Ghi_Chu	Nvarchar	128	Ghi chú

- Danh mục chức vụ

Tên: DMCV

Khoá: Ma_Cvu

Fields Name	Data Type	Fields Size	Description
W_ID	Int	4	Trường tự động
Ma_Cvu	Varchar	16	Mã chức vụ
Ten_CVu	Nvarchar	64	Tên chức vụ
Ghi_chu	Nvarchar	128	Ghi chú

- Danh mục phòng ban

Tên: DMPB

Khoá: Ma_Pban

Fields Name	Data Type	Fields Size	Description
W_ID	Int	4	Trường tự động
Ma_Pban	Varchar	16	Mã phòng ban
Ten_Pban	Nvarchar	64	Tên phòng ban
Ghi_chu	Nvarchar	128	Ghi chú

- Danh mục hợp đồng

Tên: DMHD

Khoá: So_Hdong, Ma_Nvien

Fields Name	Data Type	Fields Size	Description
W_ID	Int	4	Trường tự động
So_HDong	Varchar	16	Số hợp đồng
Dien_Giai	Nvarchar	64	Diễn giải
Ma_Nvien	Varchar	16	Mã nhân viên
Ngay_Ky	Date\Time	8	Ngày ký
Loai_Hdong	Char	1	Loại hợp đồng
Ghi_Chu	Nvarchar	128	Ghi chú

- Quá trình đào tạo

Tên: QTDT

Khoá: Ma_DTao, Ma_Nvien

Fields Name	Data Type	Fields Size	Description
W_ID	Int	4	Trường tự động
Ma_DTao	Varchar	16	Mã đào tạo
Dien_Giai	Nvarchar	64	Diễn giải

Mon_Dtao	Nvarchar	64	Môn đào tạo
Ngay_Dtao	Int	8	Ngày đào tạo
Ma_Nvien	Varchar	16	Mã nhân viên
Tinh_Trang	Nvarchar	64	Trình trạng đào tạo
Ghi_Chú	Nvarchar	128	Ghi chú

- Hồ sơ nhân viên

Tên: HSNV

Khoá: Ma_Nv

Fields Name	Data Type	Fields Size	Description
W_ID	Int	4	Trường tự động
Ma_Nvien	Varchar	16	Mã nhân viên
Ten_Nvien	Nvarchar	64	Tên nhân viên
Ngay_Sinh	Date\Time	8	Ngày sinh
Nguyen_Quan	Varchar	16	Quê quán
Dia_Chi	Nvarchar	64	Địa chỉ hiện nay
So_Dthoai	Varchar	15	Số điện thoại
Email	Varchar	64	Địa chỉ Email
Ma_Pban	Varchar	16	Phòng ban
Ma_Cvu	Varchar	16	Chức vụ
Ngay_Vcty	Date\Time	8	Ngày vào công ty
Hop_DLDong	Varchar	16	Số hợp đồng LĐ
Ghi_Chú	Nvarchar	128	Ghi chú

2. Thiết kế CSDL quản lý nhân sự bằng SQL Server 2000

- Trong phần này chúng tôi sẽ giới thiệu với các bạn cách thiết kế các bảng bằng câu lệnh SQL và sử dụng công cụ Query Analyze

- Tạo DataBases QLNS

CREATE Database QLNS

- Danh mục tỉnh thành

USE QLNS

CREATE Table DMTT

(

W_ID INT IDENTITY (1, 1) NOT NULL ,

Ma_TThanh varchar (16) Primary key NOT NULL,

Ten_TThanh nvarchar (64) NULL,

Ghi_Chua nvarchar (64) NULL

)

GO

- Danh mục chức vụ

USE QLNS

CREATE Table DMCV

(

W_ID INT IDENTITY (1, 1) NOT NULL ,

Ma_Cvu varchar (16) Primary key NOT NULL,

Ten_Cvu nvarchar (64) NULL,

Ghi_Chua nvarchar (64) NULL

)

GO

- Danh mục phòng ban

USE QLNS

CREATE Table DMTT

(

```

W_ID INT IDENTITY (1, 1) NOT NULL ,
Ma_PBan varchar (16) Primary key NOT NULL,
Ten_TBan nvarchar (64) NULL,
Ghi_Chua nvarchar (64) NULL
)
GO
- Danh mục hợp đồng
USE QLNS
CREATE Table DMHD
(
W_ID INT IDENTITY (1, 1) NOT NULL ,
So_HDong varchar (16) PRIMARY KEY NOT NULL,
Ma_NVien varchar (16) foreign key REFERENCES DMHD
(So_HDong),
Dien_Giai nvarchar (6 4) NULL,
Ngay_Ky DateTime NULL,
Loai_Hd char (1) NULL,
Ghi_Chua nvarchar (128) NULL
)
GO
- Quá trình đào tạo
USE QLNS
CREATE Table QTDT
(
W_ID INT IDENTITY (1, 1) NOT NULL ,
Ma_Dtao varchar (16) PRIMARY KEY NOT NULL,

```

*Ma_NVien varchar (16) foreign key REFERENCES DMHD
(So_HDong),*

*Dien_Giai nvarchar (6 4) NULL,
Ngay_Dtao DateTime NULL,
Mon_Dtao nvarchar (16) NULL,
Tinh_Trang nvarchar (64) NULL,
Ghi_Chua nvarchar (128) NULL
)*

GO

- Hồ sơ nhân viên

USE QLNS

CREATE Table HSNV (

*W_ID INT IDENTITY (1, 1) NOT NULL ,
Ma_NVien varchar (16) Primary key NOT NULL,
Ten_NVien nvarchar (64) NULL,
Ngay_Sinh DateTime NULL,
Nguyen_Quan Varchar (16) NULL,
Dia_Chi Nvarchar (64) NULL,
So_Dthoai Varchar (15) NULL,
Email Varchar (64) NULL
Ma_Pban Varchar (16) NULL,
Ma_Cvu Varchar (16) NULL,
Ngay_Vcty DateTime NULL,
Hop_DLDong Varchar (16) NULL,
Ghi_Chua nvarchar (64) NULL
)*

GO

MỤC LỤC

BÀI 1 - GIỚI THIỆU VỀ SQL Server 2000	7
1. SQL Server 2000	7
2. Một số kiến thức về các Version của SQL Server	8
3. Các thành phần quan trọng trong SQL Server 2000	9
a. Relational Database Engine	10
b. Replication (Cơ chế tạo bản sao)	10
c. Data Transformation Service (Dịch vụ chuyển dịch data)	10
d. Analysis Service (Dịch vụ phân tích dữ liệu)	11
e. English Query, Meta Data Service	11
g. SQL Server Books Online	11
h. SQL Server Tools	12
4. Cài đặt SQL Server 2000	12
a. Yêu cầu cấu hình máy tính	12
b. Các bước tiến hành cài đặt SQL Server 2000	12
 BÀI 2 - NGÔN NGỮ SQL	 22
1. Phân loại	22
a. Data Definition Language (DDL).	22
b. Data Control Language (DCL)	23
c. Manipulation Language (DML)	24
2. Cấu trúc câu lệnh SQL	26
a. Identifiers	26
b. Biến	26
c. Hàm	27
d. Kiểu dữ liệu	27
e. Expressions	28
f. Các thành phần Control-Of Flow	28
g. Chú Thích	28
3. Thực thi câu lệnh SQL	28
a. Thực thi một câu lệnh đơn	28
b. Thực Thi một nhóm lệnh (Batches)	28
c. Lệnh GO	28

4. Kiểu dữ liệu, Hàm	29
a. Kiểu dữ liệu	29
b. Hàm	30
5. Câu lệnh SQL	47
a. Câu lệnh Select	47
b. Sắp xếp các dữ liệu được truy vấn	50
c. Lọc dữ liệu	57
d. Lọc dữ liệu nâng cao	63
e. Lọc dữ liệu bằng toán tử đại diện	71
f. Tạo các trường tính toán	76
g. Gộp nhóm dữ liệu	85
h. Tìm hiểu Query con	94
i. Kết các Table	101
j. Tạo các phép kết nâng cao	112
k. Kết hợp các query	123
l. Chèn dữ liệu	130
m. Cập nhật và xóa dữ liệu	138
6. Một số bài tập mẫu	143
 BÀI 3 - LOGIN SQL SERVER, TẠO MỘT Database ĐƠN GIẢN	188
1. Login SQL Server	188
2. Tạo một Database đơn giản	192
a. Tạo Databases bằng Enterprise Manage	192
b. Tạo DataBases bằng Query Analyze	196
 BÀI 4 - BACK UP & RESTORE, IMPORT SQL Server 2000	199
1. Backup & Restore SQL Server	199
a. Phục hồi dữ liệu	199
b. Các Loại Backup	200
c. Recovery Models	201
d. Backup Database	205
e. Restore database	209
2. Import data	210
a. Sử dụng bcp và BULK INSERT để import data	210
b. Distributed Queries	213

BÀI 5 - CẤU TRÚC Database TRONG SQL Server 2000	216
1. Cấu trúc của SQL Server	216
2. Cấu trúc vật lý của một SQL Server Database	217
3. Nguyên tắc hoạt động của Transaction Log trong SQL Server	218
4. Cấu trúc Logic của một SQL Server Database	220
5. Cách tạo một User Database	221
6. Những điểm cần lưu ý khi thiết kế Database	223
 BÀI 6 - TÍNH TOÀN VỆN CỦA DỮ LIỆU	 228
1. Các phương pháp bảo đảm toàn vẹn dữ liệu	228
2. Constraints	229
a. Primary Key Constraint	229
b. Unique Constraint	230
c. Foreign Key Constraint	231
d. Check Constraint	232
 BÀI 7 - CURSORS, STORED PROCEDURE	 234
1. Cursors	234
2. Stored procedures	236
a. Ưu điểm của Stored procedure	236
b. Các loại Stored procedure	237
c. Viết Stored procedure	239
 BÀI 8 - Trigger & VIEW	 243
1. Triggers	243
a. Khi nào ta cần sử dụng Trigger.	243
b. Đặc điểm của Trigger	244
c. Cách tạo một Trigger.	245
2. View	247
 BÀI 9 - VÍ DỤ VỀ CƠ SỞ DỮ LIỆU QUẢN LÝ NHÂN SỰ	 251
1. Cơ sở dữ liệu quản lý nhân sự	251
2. Thiết kế CSDL quản lý nhân sự bằng SQL Server 2000	253

TỰ HỌC SQL SERVER 2000

Chịu trách nhiệm xuất bản:

BÙI VIỆT BẮC

Chịu trách nhiệm bản thảo:

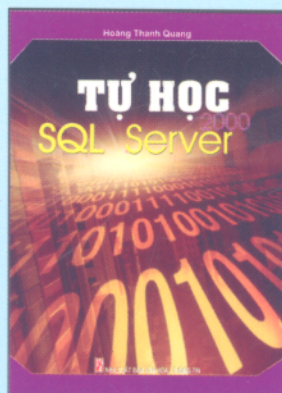
PHẠM NGỌC LUẬT

Biên tập: NGÔ HỒNG TÚ

Trình bày: NGỌC HUYỀN

Sửa bản in: TRUNG QUỐC

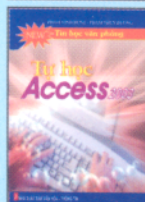
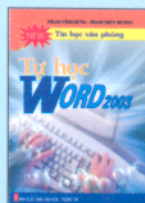
Bìa: LÊ BÁCH



Những cuốn sách sắp phát hành



Những cuốn sách đã phát hành



8 935077 013795

Giá: 30.000 đ