

Quản lý dữ liệu sử dụng Microsoft SQL Server

Quản lý dữ liệu sử dụng Microsoft SQL Server

Hướng dẫn người học

© 2014 Aptech Limited

Mọi quyền được bảo lưu.

Không có phần nào của cuốn sách này có thể được sao chép hoặc sao chép dưới mọi hình thức hoặc bằng bất kỳ phương tiện nào – đồ họa, điện tử hoặc cơ khí, bao gồm cả photocopy, ghi âm, ghi hình, hoặc lưu trữ trong hệ thống thông tin hoặc gửi hoặc truyền đi mà không có sự cho phép trước bằng văn bản của chủ sở hữu quyền tác giả Aptech Limited.

Mọi nhãn hiệu thương mại đã được công nhận.

APTECH LIMITED

E-mail: ov-support@onlinevarsity.com

Ấn bản đầu tiên - 2014



Học viên thân mến,

Chúng tôi xin chúc mừng về quyết định của bạn theo đuổi một khóa học của Aptech Worldwide.

Aptech Ltd. thiết kế các khóa học bằng một mô hình thiết kế hướng dẫn đúng đắn – từ khái niệm hóa đến thực thi, kết hợp các khía cạnh chính sau:

- Dò quét hệ thống người dùng và đánh giá nhu cầu

Đánh giá nhu cầu được tiến hành để phát hiện nhu cầu giáo dục và đào tạo của học viên

Các xu hướng công nghệ thường xuyên được tìm hiểu và theo dõi bởi các nhóm cốt lõi ở Aptech Ltd. TAG* phân tích các xu hướng này hàng tháng để hiểu nhu cầu đào tạo công nghệ mới nổi cho ngành.

Một cuộc khảo sát hồ sơ tuyển dụng của ngành công nghiệp hàng năm[#] được tiến hành vào tháng 8 - tháng 10 để hiểu các công nghệ mà các ngành công nghiệp sẽ thích nghi trong 2-3 năm tới. Phân tích về những xu hướng và nhu cầu tuyển dụng này sẽ được tiến hành để hiểu các yêu cầu kỹ năng cho các vai trò và cơ hội nghề nghiệp khác nhau.

Tiếp theo, các yêu cầu kỹ năng được đối chiếu với hồ sơ học viên (hệ thống người dùng) để xác định các mục tiêu học tập cho các vai trò khác nhau.

- Phân tích nhu cầu và thiết kế chương trình giảng dạy

Tiếp đó, các mục tiêu học tập được phân tích và biến thành các nhiệm vụ học tập. Từng nhiệm vụ hoặc hoạt động học tập được phân tích về kiến thức, kỹ năng và thái độ được yêu cầu để thực hiện nhiệm vụ đó. Các giáo viên và chuyên gia trong ngành cùng hợp tác thực hiện điều này. Sau đó những người này nhóm thành các nhóm nhỏ để xây dựng các chủ đề trong chương trình giảng dạy.

Ngoài ra, xã hội, giáo viên, và ngành công nghiệp hy vọng một số kiến thức và kỹ năng có liên quan đến khả năng như *học tập để tìm hiểu, suy nghĩ, khả năng thích ứng, giải quyết vấn đề, thái độ tích cực*, v.v.. Những năng lực này sẽ bao gồm cả hai lĩnh vực nhận thức và tình cảm.

Sơ đồ ưu tiên các chủ đề được vẽ ở nơi các điều kiện tiên quyết cho mỗi chủ đề được minh họa. Sơ đồ trong sơ đồ này được quyết định bởi thời lượng khóa học về mặt số học kỳ, v.v.. Nhờ sơ đồ ưu tiên và thời lượng cho mỗi chủ đề, chương trình giảng dạy được tổ chức hợp lý.

- Thiết kế và phát triển tài liệu hướng dẫn

Đề cương nội dung được phát triển nhờ đưa thêm vào các đề tài cần thiết để hoàn thiện lĩnh vực và để phát triển logic các năng lực đã xác định. Chiến lược và thể thức đánh giá được phát triển cho chủ đề. Các đề tài được sắp xếp/tổ chức theo thứ tự có ý nghĩa.

Tài liệu hướng dẫn chi tiết – Dụng cụ hỗ trợ đào tạo, tài liệu cho học viên, tài liệu tham khảo, nguyên tắc dự án, v.v. - được phát triển sau đó. Các kiểm tra chất lượng khắt khe được thực hiện ở mỗi giai đoạn.

➤ Các chiến lược cung cấp hướng dẫn

Sự phát triển toàn vẹn các khả năng như tư duy, giải quyết vấn đề, nhận thức học tập, v.v. được cân nhắc cẩn thận bằng cách chọn các chiến lược hướng dẫn thích hợp (phương pháp luận đào tạo), hoạt động hướng dẫn và tài liệu hướng dẫn.

Lĩnh vực CNTT đang thay đổi nhanh chóng và không rõ ràng. Do đó, quá trình hướng dẫn có tính linh hoạt cao nhờ đặc biệt đưa vào các hoạt động sáng tạo với tương tác nhóm giữa sinh viên và người đào tạo. Các khía cạnh tích cực của học trực tuyến – tiếp nhận thông tin, tổ chức thông tin và hành động trên cơ sở thông tin không đầy đủ là một số khía cạnh được kết hợp vào quá trình hướng dẫn.

➤ Đánh giá học tập

Học tập được đánh giá thông qua nhiều hình thức – kiểm tra, bài tập lớn và dự án. Hệ thống đánh giá được thiết kế nhằm đánh giá trình độ kiến thức và kỹ năng như các mục tiêu học tập đã xác định.

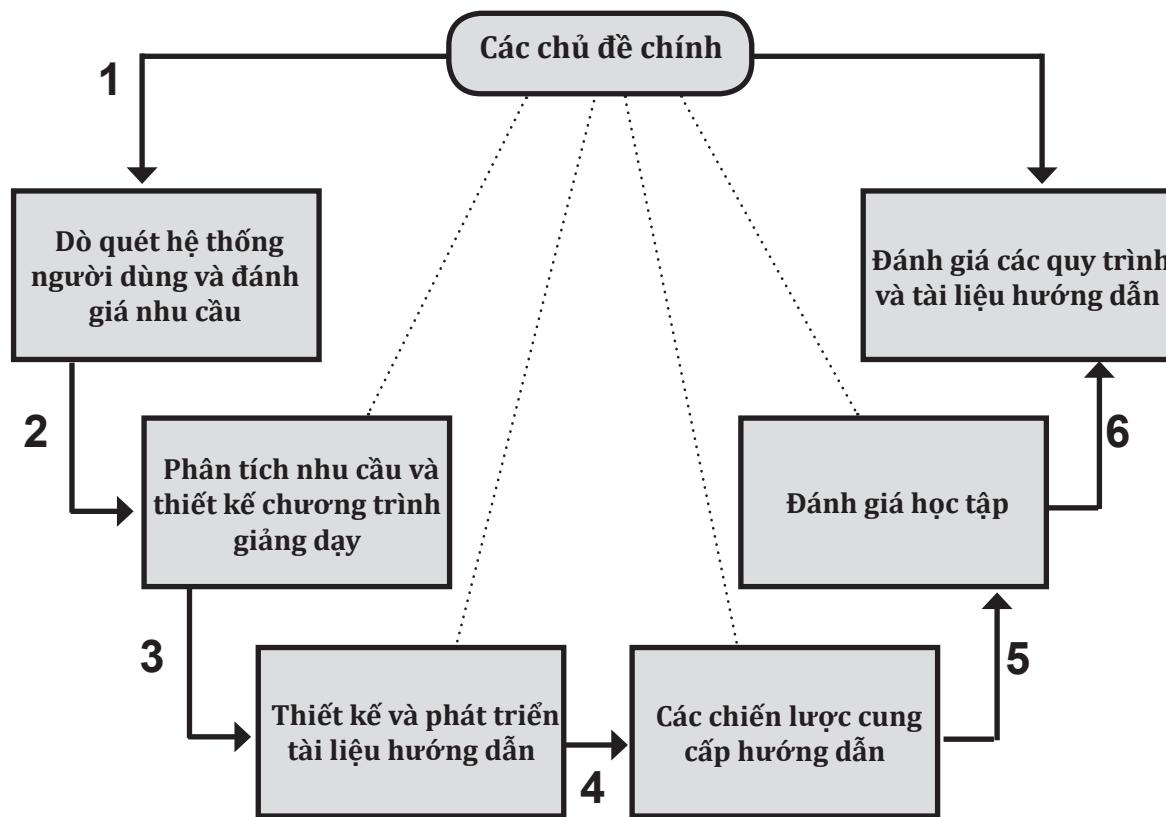
➤ Đánh giá quá trình hướng dẫn và tài liệu hướng dẫn

Quá trình hướng dẫn được hỗ trợ bởi một hệ thống giám sát tỉ mỉ để đánh giá - về nộp bài đúng hạn, hiểu học phần chủ đề, khả năng truyền đạt của người hướng dẫn. Là một phần toàn vẹn của quá trình này, chúng tôi muốn bạn gửi phản hồi của mình trong biểu mẫu trả lời đã dán tem đính ở cuối mỗi học phần.

*TAG – Nhóm Công nghệ và Hàn lâm gồm các thành viên từ Aptech Ltd., các giáo sư từ các Viện hàn lâm danh tiếng, các quản lý cấp cao trong ngành, các bậc thầy kỹ thuật từ các hãng phần mềm lớn và các đại diện từ các tổ chức/diễn đàn điều hành.

Các chuyên gia đầu ngành công nghệ của Aptech Ltd. gặp nhau hàng tháng để chia sẻ và đánh giá các xu hướng công nghệ. Nhóm gặp gỡ các đại diện của TAG ba lần một năm để đánh giá và hợp thức hóa các định hướng công nghệ và học thuật và nỗ lực của Aptech Ltd.

Mô hình thiết kế sản phẩm mới của Aptech



“

Ít kiến thức thật nguy hiểm,

nhưng sự ngu dốt còn nguy hiểm hơn

”

Lời nói đầu

SQL Server 2012 là Hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS) dựa trên máy khách-máy chủ mới nhất của Microsoft. Hệ thống cung cấp một nền quản lý dữ liệu mức doanh nghiệp cho tổ chức. SQL Server bao gồm rất nhiều tính năng và công cụ làm cho nó thành một cơ sở dữ liệu và nền tảng phân tích dữ liệu xuất sắc. Nó còn được nhắm tới Xử lý giao dịch trực tuyến (OLTP) quy mô lớn, lưu kho dữ liệu, và các ứng dụng thương mại điện tử. Một trong những tính năng chính của phiên bản SQL Server này là nó có sẵn trên nền tảng điện toán đám mây.

Cuốn sách này bắt đầu với phần giới thiệu về các khái niệm RDBMS và chuyển sang giới thiệu ngắn gọn về SQL Azure. Cuốn sách này sau đó đề cập đến nhiều chủ đề về SQL Server 2012 như là các loại dữ liệu, sử dụng Transact-SQL, và các đối tượng cơ sở dữ liệu như các chỉ mục, thủ tục đã lưu trữ, hàm, và vân vân. Cuốn sách này còn mô tả các giao dịch, các phần tử lập trình với Transact-SQL, và cuối cùng khắc phục sự cố các lỗi với các kỹ thuật xử lý lỗi.

Cuốn sách này là kết quả của một nỗ lực tập trung của Đội thiết kế, là sự phấn đấu liên tục để mang lại cho bạn điều tốt nhất và mới nhất trong Công nghệ thông tin. Quy trình thiết kế là một phần của chứng chỉ ISO 9001 cho Aptech-IT Division, Dịch vụ Hỗ trợ Giáo dục. Là một phần động lực chất lượng của Aptech, Nhóm thiết kế tiến hành nghiên cứu chuyên sâu và làm phong phú chương trình giảng dạy để bắt kịp các xu hướng của ngành.

Chúng tôi sẽ rất vui khi nhận được ý kiến đóng góp của bạn.

Nhóm thiết kế

“ Không có gì là mất thời gian
nếu bạn sử dụng kinh nghiệm thông minh ”

Mô-đun

1. Các khái niệm RDBMS
2. Mô hình và chuẩn hóa thực thể-mối quan hệ (E-R)
3. Giới thiệu về SQL Server 2012
4. SQL Azure
5. Transact-SQL
6. Tạo và quản lý cơ sở dữ liệu
7. Tạo các bảng
8. Truy cập dữ liệu
9. Truy vấn và phép nối nâng cao
10. Sử dụng giao diện, thủ tục đã lưu trữ và truy vấn siêu dữ liệu
11. Chỉ mục
12. Khởi phát
13. Lập trình Transact-SQL
14. Giao tác
15. Xử lý lỗi

“Học tập không phải là bắt buộc
nhưng đó lại là sự sống còn”

Phần - 1

Khái niệm RDBMS

Chào mừng bạn đến với phần **Khái niệm RDBMS**.

Phần này làm việc với các khái niệm liên quan đến cơ sở dữ liệu và hệ thống quản lý cơ sở dữ liệu, tìm hiểu các mô hình cơ sở dữ liệu khác nhau, và giới thiệu khái niệm về RDBMS.

Trong phần này, bạn sẽ học để:

- ➔ Giải thích khái niệm về dữ liệu và cơ sở dữ liệu
- ➔ Mô tả các phương pháp tiếp cận để quản lý dữ liệu
- ➔ Xác định một Hệ thống quản lý cơ sở dữ liệu (DBMS) và liệt kê các lợi ích của nó
- ➔ Giải thích các mô hình cơ sở dữ liệu khác nhau
- ➔ Xác định và giải thích RDBMS
- ➔ Mô tả các thực thể và bảng và liệt kê các đặc điểm của bảng
- ➔ Liệt kê sự khác biệt giữa DBMS và RDBMS

1.1 Giới thiệu

Các tổ chức thường duy trì số lượng dữ liệu lớn, chúng được tạo ra như là kết quả của các hoạt động hàng ngày. Cơ sở dữ liệu là một dạng có tổ chức của dữ liệu như vậy. Cơ sở dữ liệu gồm một hoặc nhiều các mục dữ liệu có liên quan được gọi là các bản ghi. Nghĩ về cơ sở dữ liệu như là một tập hợp dữ liệu mà có thể hỏi các câu hỏi khác nhau đối với tập hợp này. Ví dụ, “Các số điện thoại và địa chỉ của năm bưu điện gần nhất là gì?” hoặc “Chúng ta cuốn sách nào trong thư viện nói về thực phẩm có lợi cho sức khỏe hay không? Nếu có, chúng ở trên kệ nào?” hoặc “Cho tôi xem hồ sơ nhân viên và doanh số bán hàng của năm nhân viên bán hàng có thành tích tốt nhất trong quý hiện tại, nhưng không cần hiển thị các chi tiết địa chỉ của họ”.

1.2 Dữ liệu và cơ sở dữ liệu

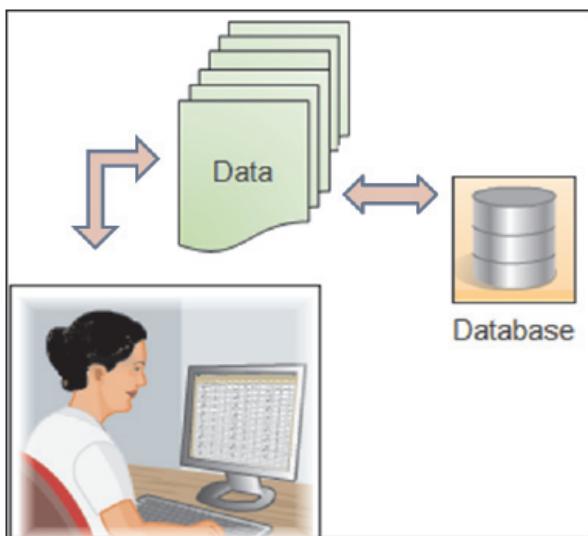
Dữ liệu có nghĩa là thông tin và đó là thành phần quan trọng nhất trong bất kỳ công việc nào đã xong. Trong hoạt động hàng ngày, hoặc dữ liệu hiện có được sử dụng hay thêm dữ liệu được tạo ra. Khi dữ liệu này được thu thập và phân tích, nó mang lại thông tin. Nó có thể là bất kỳ thông tin nào như là thông tin về xe cộ, thể thao, đường hàng không, và vân vân. Ví dụ, nhà báo của tạp chí thể thao (là một người đam mê bóng đá) tập hợp số điểm (dữ liệu) về thành tích của Đức trong 10 trận đấu cúp thế giới. Các tỉ số này cấu thành dữ liệu. Khi dữ liệu này được so sánh với dữ liệu của 10 trận đấu cúp thế giới do Brazil chơi, phóng viên có thể có được thông tin như là quốc gia nào có động bóng đá tốt hơn.

Thông tin giúp thấy trước và lên kế hoạch các sự kiện. Sự phiên dịch dữ liệu một cách thông minh mang lại thông tin. Trong thế giới kinh doanh, để có thể tiên đoán một sự kiện và lên kế hoạch cho sự kiện đó có thể tiết kiệm thời gian và tiền bạc. Xem xét ví dụ, khi công ty sản xuất xe hơi lên kế hoạch mua bán hàng năm một số bộ phận của xe hơi, những cái phải được nhập khẩu do không có sẵn trong nước. Nếu có sẵn dữ liệu về việc mua sắm các bộ phận này trong năm năm vừa qua, người đứng đầu công ty thậm chí có thể biên dịch thông tin về tổng số bộ phận đã nhập khẩu. Dựa trên các phát hiện này, có thể chuẩn bị kế hoạch sản xuất. Vì vậy, thông tin là một yếu tố quan trọng để lập kế hoạch.

Cơ sở dữ liệu là một tập hợp dữ liệu. Một số người thích nghĩa về cơ sở dữ liệu như là một cơ chế có tổ chức mà có khả năng lưu trữ thông tin. Người dùng có thể gọi ra thông tin này theo một cách thức hiệu quả và có hiệu suất cao.

Số điện thoại là một cơ sở dữ liệu. Dữ liệu được chứa bao gồm tên, địa chỉ và số điện thoại của cá nhân. Các danh sách này là theo thứ tự bảng chữ cái hoặc theo chỉ mục. Điều này cho phép Người dùng tham chiếu dễ dàng một cư dân địa phương cụ thể. Cuối cùng thì dữ liệu này được lưu trong cơ sở dữ liệu ở đâu đó trên máy tính. Bởi nhiều người dời đến các thành phố hoặc tiểu bang khác nhau, các mục nhập phải được thêm vào hoặc bỏ ra khỏi sổ điện thoại. Cũng như vậy, các mục nhập sẽ phải được sửa đổi cho những người thay đổi tên, điện thoại, hoặc số điện thoại, và vân vân.

Hình 1.1 minh họa khái niệm về cơ sở dữ liệu.



Hình 1.1: Cơ sở dữ liệu

Như vậy, cơ sở dữ liệu là một tập hợp dữ liệu được tổ chức làm sao để có thể dễ dàng truy cập, quản lý và cập nhật nội dung của chúng.

1.3 Quản lý dữ liệu

Quản lý dữ liệu nhằm giải quyết việc quản lý một số lượng lớn thông tin, trong đó có tham gia cả việc lưu trữ thông tin và cung cấp cơ chế cho việc thao tác thông tin. Ngoài ra, hệ thống cũng sẽ cung cấp sự an toàn cho thông tin được lưu trong các tình huống khác nhau, như là nhiều Người dùng truy cập, và vân vân.

Hai phương thức tiếp cận khác nhau trong việc quản lý dữ liệu là hệ thống dựa trên tập tin và hệ cơ sở dữ liệu.

1.3.1 Hệ thống dựa trên tập tin

Việc lưu trữ số lượng lớn dữ liệu đã luôn là vấn đề có được nhiều sự quan tâm. Trong những ngày đầu, các hệ thống dựa trên tập tin đã được sử dụng. Trong hệ thống này, dữ liệu được lưu trữ trong các tập tin rời rạc và một tập hợp các tập tin như vậy được lưu trữ trên một máy tính. Những tập tin này có thể được truy cập bởi trình điều hành máy tính. Các tập tin của dữ liệu đã cất giữ đã được gọi là bảng bởi vì chúng nhìn giống như các bảng được dùng trong việc giữ tập tin theo cách truyền thống. Các hàng trong bảng được gọi là bản ghi và các cột được gọi là trường.

Theo thông lệ, trước khi hệ cơ sở dữ liệu phát triển, dữ liệu trong các hệ thống phần mềm được lưu vào các tập tin phẳng.

Ví dụ về hệ thống dựa trên tập tin được minh họa trong bảng 1.1.

Họ	Tên	Địa chỉ	Số điện thoại
Eric	David	ericd@eff.org	213-456-0987
Selena	Sol	selena@eff.org	987-765-4321
Jordan	Lim	nadroj@otherdomain.com	222-3456-123

Bảng 1.1: Hệ thống dựa trên tập tin

→ Các điểm yếu của hệ thống dựa trên tập tin

Trong hệ thống dựa trên tập tin, các chương trình khác nhau trong cùng ứng dụng có thể đang tương tác với các tập tin dữ liệu riêng khác nhau. Không có hệ thống thúc ép bất kỳ sự điều khiển được chuẩn hóa nào lên việc tổ chức và cấu trúc của các tập tin dữ liệu này.

- **Dư thừa và không nhất quán dữ liệu**

Do dữ liệu nằm trong các tập tin dữ liệu riêng khác nhau, có khả năng xảy ra dư thừa và dẫn đến sự không nhất quán. Ví dụ, khách hàng có thể có một tài khoản tiết kiệm cũng như tiền vay thế chấp. Ở đây, các chi tiết khách hàng có thể được nhân đôi do các chương trình cho hai chức năng lưu trữ dữ liệu tương ứng của chúng trong hai tập tin dữ liệu khác nhau. Điều này đưa đến sự dư thừa trong dữ liệu của khách hàng. Do cùng dữ liệu được lưu trong hai tập tin, sự không nhất quán tăng lên nếu việc thay đổi được thực hiện ở dữ liệu của một tập tin không được phản ánh trong dữ liệu khác.

- **Truy vấn bất ngờ**

Trong một hệ thống dựa trên tập tin, có thể khó xử lý các truy vấn bất ngờ/không dự tính trước, do việc xử lý nào yêu cầu những thay đổi trong các chương trình hiện hữu. Ví dụ, nhân viên ngân hàng cần tạo ra một danh sách tất cả các khách hàng có bảng cân đối tài khoản là 20.000\$ hoặc nhiều hơn. Nhân viên ngân hàng này có hai lựa chọn: hoặc có được danh sách tất cả khách hàng và có thông tin cần thiết được trích ra bằng tay, hoặc thuê một lập trình viên hệ thống để thiết kế chương trình ứng dụng cần thiết. Cả hai sự thay thế hiển nhiên là không thỏa mãn. Giả sử là một chương trình như vậy được viết ra, và nhiều ngày sau, nhân viên này cần cắt bớt danh sách đó để chỉ gồm các khách hàng mà đã mở tài khoản của họ một năm trước. Bởi chương trình tạo ra danh sách như vậy không tồn tại, việc này dẫn đến khó khăn trong việc truy cập dữ liệu.

- **Cô lập dữ liệu**

Dữ liệu được phân tán trong các tập tin khác nhau, và các tập tin có thể ở định dạng khác nhau. Mặc dù dữ liệu được các chương trình khác nhau trong ứng dụng sử dụng có thể có liên quan, chúng lưu lại như các tập tin dữ liệu cô lập.

- **Tính bất thường truy cập đồng thời**

Trong các hệ thống lớn nhiều người dùng, cùng một tập tin hoặc bản ghi có thể cần phải được truy cập bởi nhiều người dùng cùng một lúc. Việc xử lý điều này trong hệ thống dựa trên tập tin là rất khó khăn.

- **Vấn đề an ninh**

Trong các ứng dụng cần nhiều dữ liệu, bảo mật dữ liệu là một mối quan ngại chính. Người dùng được cho quyền truy cập chỉ vào dữ liệu yêu cầu và không cho toàn bộ cơ sở dữ liệu.

Ví dụ, trong hệ thống ngân hàng, nhân viên trả lương chỉ cần xem một phần cơ sở dữ liệu có thông tin về các nhân viên khác nhau của ngân hàng. Họ không cần phải truy cập vào thông tin về tài khoản khách hàng. Do các chương trình ứng dụng được thêm vào hệ thống không theo thể thức, khó có thể bắt buộc các ràng buộc bảo mật như vậy. Trong hệ thống dựa trên tập tin, chỉ có thể xử lý việc này bằng việc lập trình bổ sung vào từng ứng dụng.

- **Vấn đề toàn vẹn**

Trong bất kỳ ứng dụng nào, sẽ có một số quy tắc về tính toàn vẹn dữ liệu cần phải được duy trì. Các quy tắc này có thể là ở dạng điều kiện/ràng buộc nào đó trên các thành tố của các bản ghi dữ liệu. Trong ứng dụng ngân hàng tiết kiệm, một quy tắc toàn vẹn như vậy có thể là "Mã khách hàng, đó là mã định danh duy nhất cho một hồ sơ khách hàng, không nên để trống". Có thể có nhiều các quy tắc tính toàn vẹn như vậy. Trong hệ thống dựa trên tập tin, tất cả các quy tắc này cần được lập trình riêng biệt trong chương trình ứng dụng.

Mặc dù tất cả những điều này là vấn đề phổ biến liên quan đến bất kỳ ứng dụng dùng nhiều đến dữ liệu, mỗi ứng dụng phải xử lý tất cả các vấn đề này trên chính ứng dụng đó. Lập trình viên ứng dụng cần phải bận tâm không chỉ về việc thực hiện các quy tắc kinh doanh ứng dụng, mà còn về việc xử lý những vấn đề chung này.

1.3.2 Hệ thống cơ sở dữ liệu

Hệ cơ sở dữ liệu đã phát triển trong những năm cuối của thập kỷ 60 để giải quyết các vấn đề phổ biến trong các ứng dụng xử lý số lượng lớn dữ liệu, mà cũng là các ứng dụng sử dụng thường xuyên dữ liệu. Một số trong những vấn đề này có thể là do nhược điểm của hệ thống dựa trên tập tin.

Cơ sở dữ liệu được dùng để lưu dữ liệu theo một cách hiệu quả và có tổ chức. Cơ sở dữ liệu cơ sở dữ liệu quản lý dữ liệu nhanh và dễ dàng. Ví dụ, công ty có thể duy trì chi tiết của các nhân viên trong các cơ sở dữ liệu khác nhau. Ở bất kỳ thời điểm nào, có thể gọi ra dữ liệu từ cơ sở dữ liệu, có thể thêm dữ liệu mới vào trong cơ sở dữ liệu và có thể tìm kiếm dữ liệu dựa trên một số tiêu chí trong các cơ sở dữ liệu này.

Việc lưu trữ dữ liệu có thể đạt được thậm chí bằng cách dùng các tập tin thủ công đơn giản. Ví dụ, một trường đại học phải duy trì thông tin về giáo viên, sinh viên, các môn học và các kỳ thi.

Có thể duy trì chi tiết về giáo viên trong một Ghi danh nhân viên và chi tiết về sinh viên có thể được nhập vào Ghi danh sinh viên và vân vân. Tuy nhiên, dữ liệu được lưu ở dạng này là không cố định. Chỉ có thể duy trì các bản ghi trong các tập tin thủ công như vậy trong vài tháng hoặc vài năm. Các bộ ghi hoặc tập tin cồng kềnh, tiêu tốn rất nhiều không gian, và do đó, không thể được giữ trong nhiều năm.

Thay vào đó, nếu cùng một dữ liệu đã được lưu trữ sử dụng hệ thống cơ sở dữ liệu, nó có thể là cố định và lâu dài hơn.

→ Các ưu điểm của hệ cơ sở dữ liệu

Thông tin hoặc dữ liệu có thể được lưu vĩnh viễn ở dạng các cơ sở dữ liệu được máy tính hóa. Hệ cơ sở dữ liệu là hệ thống thuận tiện bởi vì chúng cung cấp điều khiển trung tâm đối với dữ liệu.

Một số lợi ích của việc sử dụng hệ cơ sở dữ liệu trung tâm như vậy gồm:

- **Có thể giảm số lượng dư thừa trong dữ liệu được lưu**

Trong một tổ chức, nhiều phòng ban thường lưu trữ cùng dữ liệu. Việc duy trì một cơ sở dữ liệu trung tâm giúp nhiều phòng ban truy cập vào cùng dữ liệu. Do đó, sao chép dữ liệu hoặc “dư thừa dữ liệu” có thể được giảm.

- **Không có thêm tính không nhất quán trong dữ liệu**

Khi dữ liệu bị trùng lặp qua nhiều phòng ban, bất kỳ sự sửa đổi nào cho dữ liệu phải được phản ánh qua tất cả các phòng ban. Đôi khi, điều này có thể dẫn đến sự không nhất quán trong dữ liệu. Khi có một cơ sở dữ liệu trung tâm, có thể cho một người để tiếp nhận nhiệm vụ cập nhật dữ liệu một cách thường xuyên. Hãy xem xét rằng ông Larry Finner, nhân viên của một tổ chức được thăng chức làm Giám đốc cấp cao từ vị trí Giám đốc. Trong trường hợp này, đó chỉ là một bản ghi trong cơ sở dữ liệu cần phải được thay đổi. Kết quả là, tính không nhất quán của dữ liệu được giảm xuống.

- **Có thể chia sẻ dữ liệu đã lưu**

Có thể định vị cơ sở dữ liệu trung tâm trên máy chủ, nhiều Người dùng có thể chia sẻ cơ sở dữ liệu này. Theo cách này, tất cả Người dùng có thể truy cập vào thông tin chung và đã cập nhật bất cứ lúc nào.

- **Các tiêu chuẩn có thể được đặt và làm theo**

Điều khiển trung tâm bảo đảm là một tiêu chuẩn nào đó trong việc biểu thị dữ liệu có thể được đặt và làm theo. Ví dụ, tên của nhân viên phải được trình bày là “ông Larry Finner”. Sự biểu thị này có thể được chia nhỏ thành các thành phần sau đây:

- ◆ Danh hiệu (Mr.)
- ◆ Tên họ (Larry)
- ◆ Tên gọi (Finner)

Chắc chắn là tất cả tên được lưu trong cơ sở dữ liệu sẽ tuân theo cùng một định dạng nếu các tiêu chuẩn được đặt theo cách này.

- **Có thể duy trì tính toàn vẹn dữ liệu**

Tính toàn vẹn dữ liệu đề cập tới độ chính xác của dữ liệu trong cơ sở dữ liệu. Ví dụ, khi một nhân viên từ chức và rời khỏi tổ chức, xem xét là Phòng Kế toán đã cập nhật cơ sở dữ liệu của mình và Phòng Nhân sự đã không cập nhật các bản ghi của mình. Do đó, dữ liệu trong hồ sơ của công ty là không chính xác.

Việc điều khiển trung tâm cơ sở dữ liệu giúp tránh các lỗi này. Chắc chắn là nếu một bản ghi được xóa khỏi một bảng, bản ghi có liên kết trong bảng khác cũng được xóa.

- **Việc bảo mật dữ liệu có thể được thực hiện**

Trong hệ cơ sở dữ liệu trung tâm, đặc quyền sửa đổi cơ sở dữ liệu không được trao cho bất kỳ ai. Quyền này chỉ được trao cho một người có quyền điều khiển đầy đủ đối với cơ sở dữ liệu. Người này được gọi là Người quản trị cơ sở dữ liệu hay DBA. DBA có thể thực hiện việc bảo mật bằng cách đặt các hạn chế lên dữ liệu. Dựa trên các quyền được cấp cho họ, người dùng có thể thêm, sửa đổi, hoặc truy vấn dữ liệu.

1.4 Hệ thống quản lý cơ sở dữ liệu (DBMS)

DBMS có thể được định nghĩa là một tập hợp các bản ghi có liên quan và một tập hợp các chương trình truy cập và thao tác những bản ghi này. DBMS cho phép người dùng nhập vào, lưu trữ, và quản lý dữ liệu. Vấn đề chính với các gói DBMS trước đó là dữ liệu được lưu ở định dạng tập tin phẳng. Do vậy, thông tin về các đối tượng khác nhau được duy trì riêng trong các tập tin vật lý khác nhau. Từ đó, quan hệ giữa các đối tượng này, nếu có, phải được duy trì trong tập tin vật lý riêng. Như vậy, một gói đơn sẽ gồm quá nhiều tập tin và rất nhiều chức năng để tích hợp chúng vào trong một hệ thống đơn.

Giải pháp cho các vấn đề này có ở dạng hệ cơ sở dữ liệu trung tâm. Trong một hệ thống cơ sở dữ liệu tập trung, cơ sở dữ liệu được lưu trữ ở vị trí trung tâm. Mọi người có thể có quyền truy cập vào dữ liệu đã lưu ở vị trí trung tâm từ máy của họ. Ví dụ, một hệ thống dữ liệu trung tâm lớn sẽ gồm tất cả dữ liệu có liên quan đến nhân viên. Phòng Kế toán và Phòng Nhân sự sẽ truy cập vào dữ liệu yêu cầu bằng cách sử dụng các chương trình phù hợp. Các chương trình này hoặc toàn bộ ứng dụng sẽ nằm trên các đầu cuối máy tính riêng lẻ.

Cơ sở dữ liệu là tập hợp dữ liệu liên quan đến nhau, và DBMS là tập hợp các chương trình được sử dụng để thêm hoặc sửa đổi dữ liệu này. Do đó, DBMS là một tập hợp các chương trình phần mềm cho phép các cơ sở dữ liệu được định nghĩa, xây dựng, và thao tác.

DBMS mang lại một môi trường vừa thuận tiện vừa hiệu quả để sử dụng khi có một số lượng lớn dữ liệu và nhiều giao tác được xử lý. Có thể sử dụng các loại DBMS khác nhau, từ các hệ thống nhỏ chạy trên máy tính cá nhân đến các hệ thống lớn chạy trên các máy tính khổng lồ.

Ví dụ về các ứng dụng cơ sở dữ liệu bao gồm:

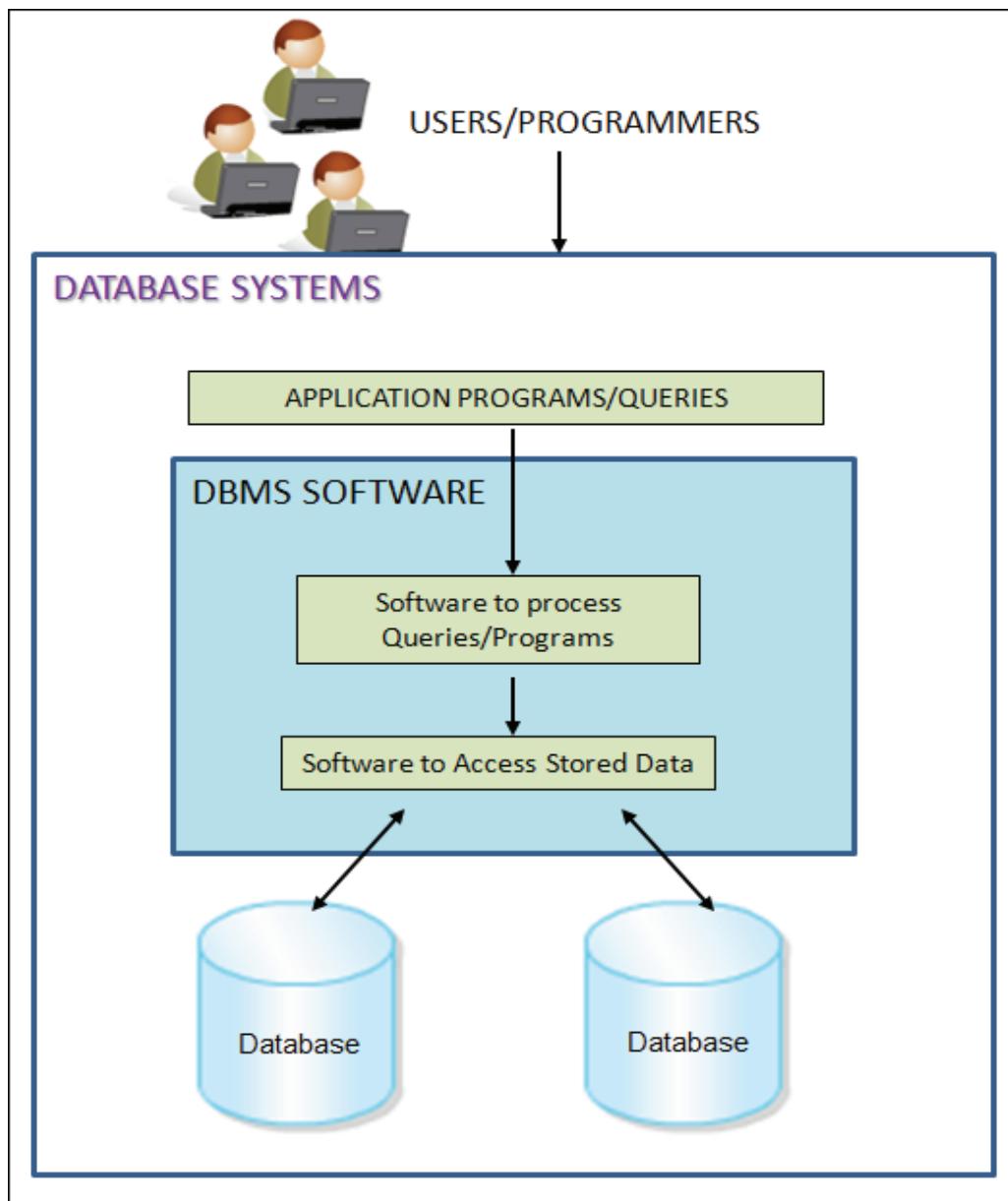
- ➔ Hệ thống thư viện được máy tính hóa
- ➔ Máy rút tiền tự động
- ➔ Hệ thống giữ chỗ bán vé máy bay
- ➔ Hệ thống tồn kho linh kiện được máy tính hóa

Từ quan điểm kỹ thuật, các sản phẩm DBMS có thể khác nhau rất nhiều. DBMS khác nhau hỗ trợ các ngôn ngữ truy vấn khác nhau, mặc dù có một ngôn ngữ truy vấn được tiêu chuẩn hóa một nửa, gọi là Ngôn ngữ truy vấn có cấu trúc (SQL). Các ngôn ngữ phức tạp dùng để quản lý các hệ cơ sở dữ liệu được gọi là Ngôn ngữ thế hệ thứ tư (4GL).

Thông tin từ cơ sở dữ liệu có thể được biểu thị ở rất nhiều các định dạng. Hầu hết các DBMS bao gồm một chương trình viết báo cáo để cho phép Người dùng đưa ra dữ liệu ở dạng báo cáo. Nhiều DBMS còn bao gồm thành phần đồ họa để cho phép Người dùng đưa ra thông tin ở dạng đồ họa và biểu đồ.

Không cần thiết phải sử dụng DBMS cho mục đích chung để thực hiện một cơ sở dữ liệu được máy tính hóa. Người dùng có thể viết tập hợp chương trình của riêng họ để tạo ra và duy trì cơ sở dữ liệu, trong việc tạo ra phần mềm DBMS cho mục đích đặc biệt của riêng họ. Cơ sở dữ liệu và phần mềm gộp lại được gọi là hệ cơ sở dữ liệu. Người dùng cuối truy cập vào hệ cơ sở dữ liệu qua các chương trình ứng dụng và các truy vấn. Phần mềm DBMS cho phép Người dùng xử lý các truy vấn và chương trình do Người dùng cuối đặt ra. Phần mềm truy cập dữ liệu từ cơ sở dữ liệu.

Hình 1.2 minh họa một hệ cơ sở dữ liệu.



Hình 1.2: Môi trường hệ cơ sở dữ liệu được đơn giản hóa

1.4.1 Lợi ích của DBMS

DBMS có trách nhiệm xử lý dữ liệu và chuyển đổi nó thành thông tin. Cho mục đích này, cơ sở dữ liệu phải được thao tác, trong đó bao gồm truy vấn cơ sở dữ liệu để lấy dữ liệu cụ thể, cập nhật cơ sở dữ liệu, và cuối cùng, tạo các báo cáo.

Những báo cáo này là nguồn thông tin, đó là, dữ liệu đã xử lý. DBMS còn chịu trách nhiệm cho sự bảo mật và tính toàn vẹn dữ liệu.

Những lợi ích của DBMS điển hình như sau:

→ **Lưu trữ dữ liệu**

Các chương trình cần có để lưu trữ dữ liệu về mặt vật lý, được xử lý bằng DBMS, được thực hiện bằng cách tạo ra các cấu trúc dữ liệu phức tạp, và quá trình này được gọi là quản lý lưu trữ dữ liệu.

→ **Định nghĩa dữ liệu**

DBMS cung cấp các hàm để xác định cấu trúc của dữ liệu trong ứng dụng. Những hàm này bao gồm việc xác định và thay đổi cấu trúc bản ghi, loại và kích thước của các trường, và các ràng buộc/điều kiện khác nhau được thỏa mãn bằng dữ liệu trong từng trường.

→ **Thao tác dữ liệu**

Một khi cấu trúc dữ liệu được định nghĩa, dữ liệu cần được chèn vào, sửa đổi, hoặc xóa. Những hàm, trong đó thực hiện những hoạt động này, cũng là một phần của DBMS. Các hàm này còn xử lý nhu cầu thao tác dữ liệu có kế hoạch và không có kế hoạch. Các truy vấn có kế hoạch là những cái hình thành nên một phần của ứng dụng. Các truy vấn không có kế hoạch là các truy vấn không theo thể thức, được thực hiện theo nhu cầu.

→ **Bảo mật và toàn vẹn dữ liệu**

Sự bảo mật dữ liệu là vấn đề hết sức quan trọng khi có nhiều Người dùng truy cập vào cơ sở dữ liệu. Cần phải duy trì kiểm tra đối với việc truy cập dữ liệu của Người dùng. Các quy tắc bảo mật chỉ ra Người dùng nào có quyền truy cập vào cơ sở dữ liệu, các thành phần dữ liệu nào Người dùng có quyền truy cập vào, và các phép tính dữ liệu Người dùng có thể thực hiện.

Dữ liệu trong cơ sở dữ liệu chứa càng ít lỗi càng tốt. Ví dụ, mã số của nhân viên để thêm một nhân viên mới không nên để trống. Số điện thoại chỉ gồm các số. Những lần kiểm tra như vậy được thực hiện bởi DBMS.

Như vậy, DBMS chứa các hàm, các hàm này xử lý sự bảo mật và tính toàn vẹn của dữ liệu trong ứng dụng. Những cái này có thể dễ dàng được ứng dụng gọi ra và do đó, lập trình viên ứng dụng không cần viết mã những hàm này trong các chương trình.

→ **Phục hồi và đồng thời dữ liệu**

Khôi phục dữ liệu sau khi có lỗi hệ thống và nhiều Người dùng truy cập đồng thời các bản ghi cũng được DBMS xử lý.

→ **Hiệu suất**

Việc tối ưu hiệu suất của các truy vấn là một trong các chức năng quan trọng của DBMS. Do đó, DBMS có một tập hợp các chương trình hình thành Query Optimizer, là trình đánh giá các cách triển khai truy vấn khác nhau và lựa chọn cái tốt nhất trong số đó.

→ Kiểm soát nhiều Người dùng truy cập

Vào bất cứ lúc nào, nhiều hơn một Người dùng có thể truy cập vào cùng dữ liệu. DBMS giải quyết việc chia sẻ dữ liệu giữa nhiều Người dùng, và duy trì tính toàn vẹn dữ liệu.

→ Các ngôn ngữ truy cập cơ sở dữ liệu và Giao diện lập trình ứng dụng (API)

Ngôn ngữ truy vấn của DBMS thực hiện truy cập dữ liệu. SQL là ngôn ngữ truy vấn được dùng phổ biến nhất. Ngôn ngữ truy vấn là một ngôn ngữ không theo thủ tục, nơi Người dùng cần vấn tin những gì cần có và không cần chỉ ra cách để được thực hiện. Một số ngôn ngữ thủ tục như là C, Visual Basic, Pascal, và những ngôn ngữ khác cung cấp quyền truy cập dữ liệu cho các lập trình viên.

1.5 Mô hình cơ sở dữ liệu

Có thể phân biệt các cơ sở dữ liệu dựa trên các hàm và mô hình của dữ liệu. Mô hình dữ liệu mô tả phần chứa để lưu trữ dữ liệu, và quá trình lưu trữ và gọi ra dữ liệu từ phần chứa đó. Phân tích và thiết kế các mô hình cơ sở dữ liệu đã là nền tảng phát triển các cơ sở dữ liệu. Mỗi mô hình đã được phát triển từ mô hình trước.

Các mô hình cơ sở dữ liệu được thảo luận ngắn gọn trong các phần sau đây.

1.5.1 Mô hình dữ liệu tập tin phẳng

Trong mô hình này, cơ sở dữ liệu gồm chỉ một bảng hoặc tập tin. Mô hình này được dùng cho các cơ sở dữ liệu đơn giản - ví dụ, để lưu trữ số đăng ký, tên, môn học và điểm của một nhóm sinh viên. Mô hình này không thể xử lý dữ liệu quá phức tạp. Nó có thể gây ra sự dư thừa khi dữ liệu bị lặp lại nhiều hơn một lần. Bảng 1.2 minh họa cấu trúc của một cơ sở dữ liệu tập tin phẳng.

Roll Number	FirstName	LastName	Subject	Marks
45	Jones	Bill	Toán học	84
45	Jones	Bill	Khoa học	75
50	Mary	Mathew	Khoa học	80

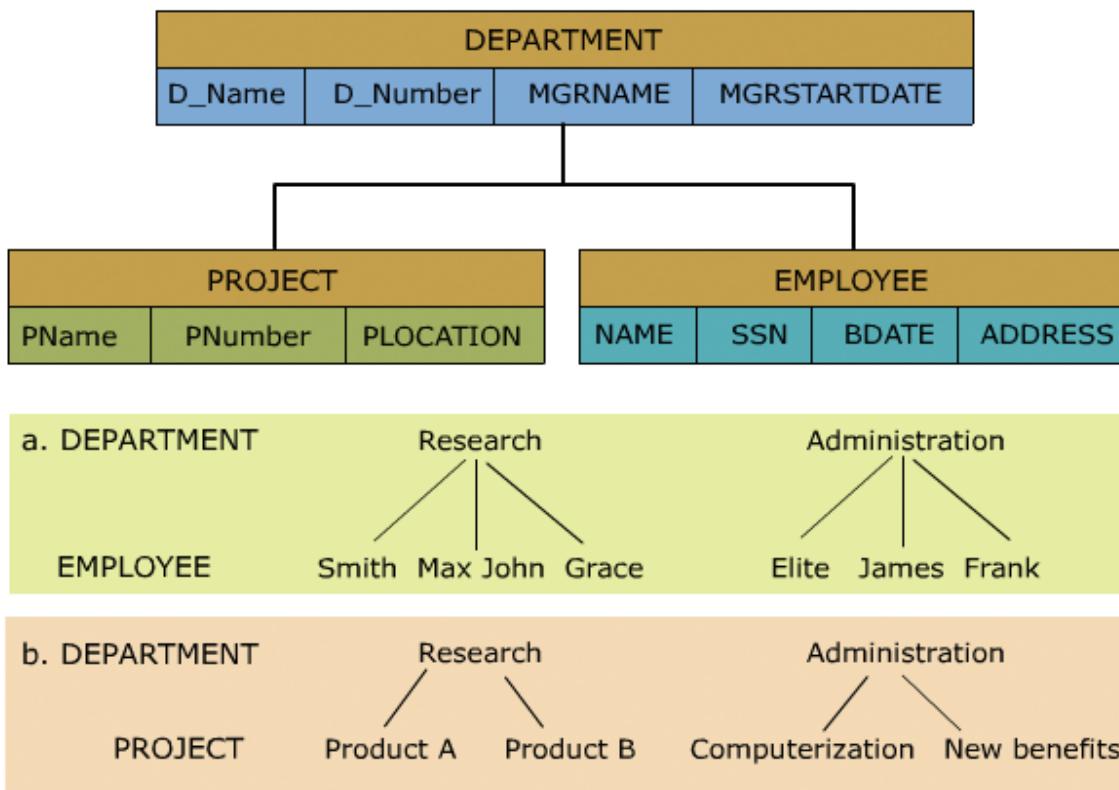
Bảng 1.2: Cấu trúc của mô hình dữ liệu tập tin phẳng

1.5.2 Mô hình dữ liệu phân cấp

Trong mô hình phân cấp, các bản ghi khác nhau được liên kết với nhau qua cấu trúc phân cấp hoặc dạng cây. Trong mô hình này, các mối quan hệ được nghĩ đến con và mẹ. Bản ghi mẹ có thể có nhiều bản ghi con, nhưng một bản ghi con chỉ có thể có một bản ghi mẹ. Để tìm ra dữ liệu được lưu trong mô hình này, Người dùng cần biết cấu trúc cây.

Windows Registry là một ví dụ về cơ sở dữ liệu phân cấp lưu trữ các thiết đặt và tùy chọn cấu hình trên hệ điều hành Microsoft Windows.

Hình 1.3 minh họa ví dụ về bản trình bày phân cấp.



Hình 1.3: Ví dụ về mô hình phân cấp

Ở trong mô hình phân cấp, Department được hiểu là mẹ của mảng. Các bảng Project và Employee là con. Đường vẽ theo các đoạn mẹ bắt đầu từ bên trái, định nghĩa cây. Sự sắp xếp các đoạn có thứ tự này dò theo cấu trúc phân cấp được gọi là đường phân cấp.

Chúng ta thấy rõ từ hình trong một phòng ban, có thể có nhiều nhân viên và một phòng ban có thể có nhiều dự án.

➔ Ưu điểm của mô hình phân cấp

Các ưu điểm của mô hình phân cấp như sau:

- Dữ liệu được tổ chức trong một cơ sở dữ liệu chung để chia sẻ dữ liệu trở nên dễ dàng hơn, và bảo mật được cung cấp và thực thi bởi DBMS.
- Tính độc lập dữ liệu được DBMS cung cấp, làm giảm công sức và chi phí trong việc duy trì chương trình.

Mô hình này rất hiệu quả khi cơ sở dữ liệu chứa một số lượng lớn dữ liệu. Ví dụ, hệ thống tài khoản khách hàng của ngân hàng thích hợp với mô hình phân cấp bởi vì tài khoản của từng khách hàng là đối tượng của một số giao dịch.

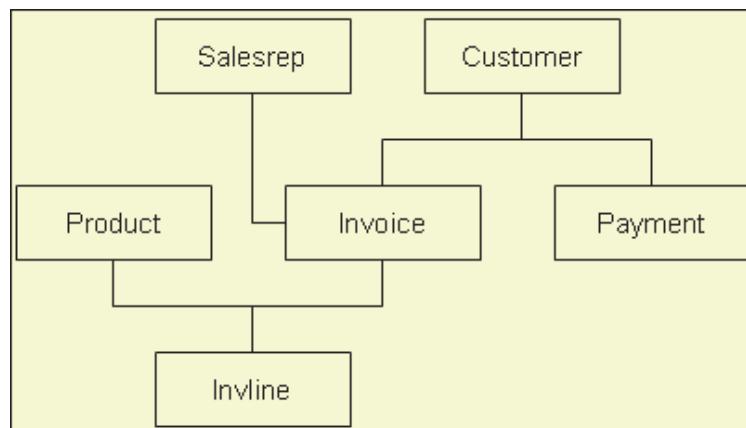
1.5.3 Mô hình dữ liệu mạng

Mô hình này tương tự như Mô hình dữ liệu phân cấp. Mô hình phân cấp thực tế là một tập hợp phụ của mô hình mạng. Tuy nhiên, thay bằng sử dụng một hệ thứ bậc dạng cây mẹ đơn, mô hình mạng sử dụng thuyết tập hợp để cung cấp hệ thứ bậc dạng cây với trường hợp ngoại lệ là các bảng con được cho phép có nhiều hơn một mẹ.

Trong mô hình mạng, dữ liệu được lưu thành các tập hợp, thay bằng định dạng cây phân cấp. Điều này giải quyết vấn đề dư thừa dữ liệu. Thuyết tập hợp của mô hình mạng không sử dụng hệ thứ bậc dạng cây mẹ đơn. Nó cho phép con có nhiều hơn một mẹ. Như vậy, các bản ghi được liên kết vật lý qua các danh sách có liên kết. Hệ thống quản lý cơ sở dữ liệu tích hợp (IDMS) của Associates International Computer Inc. và máy chủ Trình quản lý cơ sở dữ liệu Raima (RDM) của Raima Inc. là những ví dụ về DBMS mạng.

Mô hình mạng cùng với mô hình dữ liệu phân cấp là mô hình dữ liệu chính để thực hiện số lượng lớn DBMS thương mại. Các cấu trúc mô hình mạng và các hàm dựng ngôn ngữ được định nghĩa bằng Hội thảo Ngôn ngữ hệ thống dữ liệu (CODASYL).

Đối với mọi cơ sở dữ liệu, định nghĩa về tên của cơ sở dữ liệu, loại bản ghi cho từng bản ghi, và các thành phần tạo ra các bản ghi này được lưu lại. Cái này được gọi là lược đồ mạng. Một phần cơ sở dữ liệu như được thấy bởi các chương trình của ứng dụng thực sự sản xuất ra thông tin mong muốn từ dữ liệu chứa trong cơ sở dữ liệu được gọi là tiểu lược đồ. Nó cho phép các chương trình ứng dụng truy cập vào dữ liệu mong muốn từ cơ sở dữ liệu.



Hình 1.4: Mô hình mạng

Mô hình mạng được trình bày trong hình 1.4 minh họa một loạt các mối quan hệ một-nhiều, như sau:

1. Người đại diện kinh doanh có thể đã viết nhiều bản Hóa đơn, nhưng mỗi Hóa đơn được một người đại diện kinh doanh (Salesrep) viết.
2. Khách hàng có thể đã mua sắm vào các dịp khác nhau. Khách hàng có thể có nhiều tờ Hóa đơn, nhưng mỗi Hóa đơn chỉ thuộc về một khách hàng duy nhất.
3. Tờ Hóa đơn có thể có nhiều dòng mặt hàng (Invline), nhưng mỗi dòng Invline được tìm thấy trên một tờ Hóa đơn.
4. Sản phẩm có thể xuất hiện trong một số Invline khác nhau, nhưng mỗi Invline chỉ chứa một Sản phẩm.

Những thành phần của ngôn ngữ được sử dụng với các mô hình mạng như sau:

1. Ngôn ngữ định nghĩa dữ liệu (DDL) được dùng để tạo ra và gỡ bỏ các cơ sở dữ liệu và các đối tượng cơ sở dữ liệu. Ngôn ngữ này cho phép người quản trị cơ sở dữ liệu định nghĩa các thành phần của lược đồ.
2. DDL của lược đồ phụ mà cho phép người quản trị cơ sở dữ liệu định nghĩa các thành phần cơ sở dữ liệu.
3. Ngôn ngữ thao tác dữ liệu (DML), được sử dụng để chèn, lấy, và sửa đổi thông tin cơ sở dữ liệu. Tất cả Người dùng cơ sở dữ liệu sử dụng các lệnh này trong hoạt động thường lệ của cơ sở dữ liệu.
4. Ngôn ngữ điều khiển dữ liệu (DCL) được dùng để quản trị sự cho phép đối với các cơ sở dữ liệu và đối tượng cơ sở dữ liệu.

→ **Ưu điểm của mô hình mạng**

Các ưu điểm của cấu trúc như vậy được chỉ ra như sau:

- Dễ thực hiện các mối quan hệ trong mô hình cơ sở dữ liệu mạng hơn là trong mô hình phân cấp.
- Mô hình này cung cấp tính toàn vẹn cơ sở dữ liệu.
- Mô hình này đạt được đủ sự độc lập dữ liệu.

→ **Nhược điểm của mô hình mạng**

Các nhược điểm được chỉ ra như sau:

- Cơ sở dữ liệu trong mô hình này khó thiết kế.
- Người lập trình phải rất quen thuộc với các cấu trúc bên trong để truy cập vào cơ sở dữ liệu.
- Mô hình này cung cấp một môi trường truy cập dữ liệu có điều hướng. Do vậy, để di chuyển từ A đến E theo trình tự A-B-C-D-E, Người dùng phải di chuyển qua B, C, và D để đến E.

Mô hình này khó thực hiện và duy trì. Những người lập trình máy tính, hơn là những Người dùng cuối, sử dụng mô hình này.

1.5.4 Mô hình dữ liệu quan hệ

Bởi thông tin cần lớn dần lên và yêu cầu các ứng dụng và cơ sở dữ liệu rắc rối hơn, việc thiết kế, quản lý và sử dụng cơ sở dữ liệu trở lên quá cồng kềnh. Thiếu cơ sở tiện ích truy vấn sẽ mất rất nhiều thời gian của người lập trình để tạo ra thậm chí các báo cáo đơn giản nhất. Điều này đã dẫn đến việc phát triển cái được gọi là cơ sở dữ liệu Mô hình quan hệ.

Thuật ngữ “quan hệ” có nguồn gốc từ lý thuyết tập hợp của toán học. Trong mô hình quan hệ, không giống như mô hình phân cấp và mô hình mạng, không có các liên kết vật lý. Tất cả dữ liệu được duy trì ở dạng các bảng gồm các hàng và các cột. Dữ liệu trong hai bảng được liên kết qua các cột chung và không phải là các liên kết vật lý. Các toán tử được cung cấp để tính toán trên các hàng trong bảng.

Các DBMS quan hệ phổ biến là Oracle, Sybase, DB2, Microsoft SQL Server, và vân vân.

Mô hình này biểu diễn cơ sở dữ liệu như một tập hợp các quan hệ. Trong thuật ngữ của mô hình này, hàng được gọi là bộ, cột, thuộc tính, và bảng được gọi là mối quan hệ. Danh sách các giá trị áp dụng được cho một trường cụ thể được gọi là miền. Có khả năng nhiều thuộc tính có cùng miền. Số các thuộc tính của quan hệ được gọi là mức độ quan hệ. Số các bộ xác định bản số của quan hệ.

Để hiểu mô hình quan hệ, hãy xem xét bảng 1.3 và 1.4.

Roll Number	Student Name
1	Sam Reiner
2	John Parkinson
3	Jenny Smith
4	Lisa Hayes
5	Penny Walker
6	Peter Jordan
7	Joe Wong

Bảng 1.3: Bảng Students

Roll Number	Marks Obtained
1	34
2	87
3	45
4	90
5	36
6	65
7	89

Bảng 1.4: Bảng Marks

Bảng Students hiển thị Roll Number và Student Name, và bảng Marks hiển thị Roll Number và Marks học viên có được. Bây giờ cần thực hiện hai bước cho các sinh viên có điểm trên 50. Đầu tiên, xác định vị trí số đăng ký của những học viên có số điểm trên 50 từ bảng Marks. Thứ hai, tên của họ phải được đặt trong bảng Students bằng cách so khớp số đăng ký. Kết quả sẽ như được trình bày trong bảng 1.5.

Roll Number	Student Name	Marks Obtained
2	John	87
4	Lisa	90
6	Peter	65
7	Joe	89

Bảng 1.5: Hiển thị họ tên và số điểm của học viên

Có thể có được thông tin này bởi vì hai thực tế: Trước tiên, có một cột chung cho cả hai bảng - **Roll Number**. Thứ hai, dựa trên cột này, các bản ghi từ hai bảng khác nhau có thể được so khớp và có thể có được thông tin yêu cầu.

Trong mô hình quan hệ, dữ liệu được lưu trong các bảng. Bảng trong cơ sở dữ liệu có tên không trùng lặp để nhận dạng nội dung của chúng. Có thể định nghĩa mỗi bảng như một phần giao cắt của các hàng và các cột.

→ **Ưu điểm của mô hình quan hệ**

Mô hình cơ sở dữ liệu quan hệ cho người lập trình thời gian để tập trung vào cái nhìn logic của cơ sở dữ liệu hơn là quan tâm đến cái nhìn vật lý. Một trong các lý do về tính phổ biến về các cơ sở dữ liệu quan hệ là độ linh động trong việc truy vấn. Hầu hết các cơ sở dữ liệu quan hệ sử dụng Ngôn ngữ truy vấn có cấu trúc (SQL). RDBMS sử dụng SQL để biên dịch truy vấn của Người dùng thành mã kỹ thuật cần thiết để lấy dữ liệu yêu cầu. Mô hình quan hệ rất dễ xử lý, thậm chí những người không được đào tạo cũng thấy dễ tạo ra các báo cáo đẹp mắt và các truy vấn, mà không cần mất nhiều suy nghĩ để thiết kế một cơ sở dữ liệu phù hợp.

→ **Nhược điểm của mô hình quan hệ**

Mặc dù mô hình ẩn chứa tất cả các sự phức tạp của hệ thống, nó có xu hướng chậm hơn so với các hệ cơ sở dữ liệu khác.

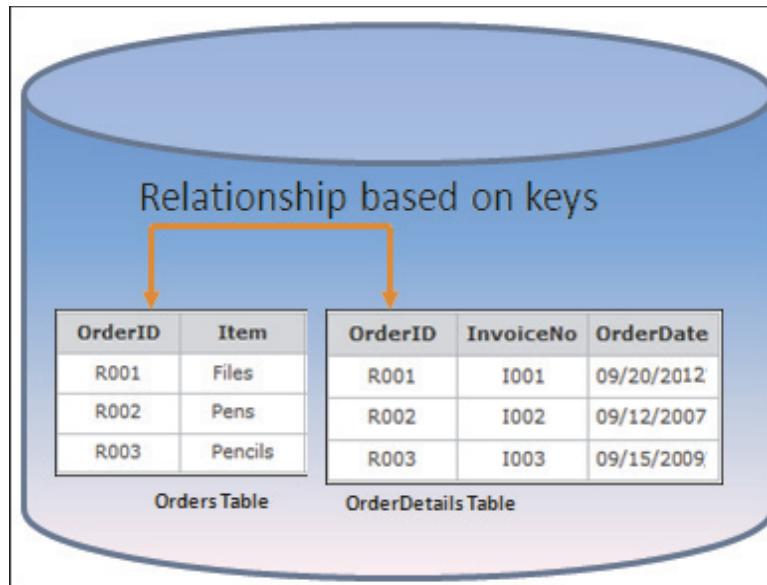
Khi so sánh với tất cả các mô hình khác, mô hình dữ liệu quan hệ được dùng rộng rãi và phổ biến nhất.

1.6 Hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS)

Mô hình quan hệ là một cố gắng để đơn giản hóa các cấu trúc cơ sở dữ liệu. Mô hình biểu diễn tất cả dữ liệu trong cơ sở dữ liệu như các bảng hàng-cột đơn giản của các giá trị dữ liệu. RDBMS là một chương trình phần mềm giúp bạn tạo ra, duy trì, và thao tác cơ sở dữ liệu quan hệ. Cơ sở dữ liệu quan hệ là cơ sở dữ liệu được chia thành các đơn vị logic được gọi là bảng, nơi các bảng được liên kết với nhau ở trong cơ sở dữ liệu.

Các bảng được liên kết trong một cơ sở dữ liệu quan hệ, cho phép lấy ra dữ liệu thích hợp trong một truy vấn đơn lẻ (mặc dù dữ liệu mong muốn có thể tồn tại trong nhiều hơn một bảng). Bởi có các khóa, hoặc trường chung trong các bảng cơ sở dữ liệu quan hệ, dữ liệu từ nhiều bảng có thể được nối lại để tạo thành một tập kết quả lớn.

Hình 1.5 trình bày hai bảng có liên kết với nhau qua một khóa chung (giá trị dữ liệu) trong một cơ sở dữ liệu quan hệ.



Hình 1.5: Mối quan hệ giữa các bảng

Như vậy, cơ sở dữ liệu quan hệ là một cơ sở dữ liệu được cấu trúc trên mô hình quan hệ. Đặc điểm cơ bản của một mô hình quan hệ là trong một mô hình quan hệ, dữ liệu được lưu trữ trong các quan hệ. Để hiểu các quan hệ, hãy xem xét ví dụ sau đây.

Bảng **Capitals** được trình bày trong bảng 1.6 hiển thị danh sách các nước và thủ đô của các nước đó, và bảng **Currency** được trình bày trong bảng 1.7 hiển thị các quốc gia và các loại tiền tệ địa phương được các quốc gia đó sử dụng.

Country	Capital
Greece	Athens
Italy	Rome
USA	Washington
China	Beijing
Japan	Tokyo
Australia	Sydney
France	Paris

Bảng 1.6: Capitals

Country	Currency
Greece	Drachma
Italy	Lira
USA	Dollar
China	Renminbi (Yuan)
Japan	Yen
Australia	Australian Dollar
France	Francs

Bảng 1.7: Currency

Cả hai bảng có một cột chung, có nghĩa là cột **Country**. Nay giờ, nếu Người dùng muốn hiển thị thông tin về loại tiền được dùng ở Rome, đầu tiên tìm tên của quốc gia mà Rome thuộc về nơi đó. Thông tin này có thể được lấy ra từ bảng 1.6. Tiếp theo, quốc gia đó cần được tra cứu trong bảng 1.7 để tìm ra loại tiền tệ.

Có thể có được thông tin này vì có thể thiết lập một mối quan hệ giữa hai bảng thông qua một cột chung được gọi là **Country**.

1.6.1 Các thuật ngữ liên quan đến RDBMS

Có một số thuật ngữ được sử dụng nhiều trong RDBMS. Những công cụ này được mô tả như sau:

- Dữ liệu được biểu diễn như một tập hợp các quan hệ.
- Mỗi quan hệ được minh họa như một bảng.
- Các cột là các thuộc tính.
- Hàng (“bộ dữ liệu”) thể hiện các thực thể.
- Mỗi bảng có một tập hợp các thuộc tính được gom lại làm “khóa” (về mặt kỹ thuật, là “siêu khóa”), trong đó xác định tính duy nhất cho mỗi thực thể.

Ví dụ, một công ty có thể có một bảng **Employee** với một hàng cho mỗi nhân viên. Những gì thuộc tính nào có thể đáng chú ý cho một bảng như vậy? Điều này sẽ phụ thuộc vào ứng dụng và loại hình sử dụng dữ liệu sẽ được đưa vào, và được xác định tại thời điểm thiết kế cơ sở dữ liệu.

Hãy xem xét kịch bản về một công ty duy trì thông tin khách hàng và đơn đặt hàng cho các sản phẩm đang được bán và thông tin chi tiết đặt hàng-khách hàng trong một tháng cụ thể, chẳng hạn như tháng Tám.

Bảng 1.8, 1.9, 1.10 và 1.11 được sử dụng để minh họa cho kịch bản này. Các bảng này minh họa các bộ dữ liệu và các thuộc tính ở dạng hàng và cột. Các thuật ngữ khác nhau liên quan đến những bảng này được đưa ra trong bảng 1.12.

Cust_No	Cust_Name	Phone_No
002	David Gordon	0231-5466356
003	Prince Fernandes	0221-5762382
003	Charles Yale	0321-8734723
002	Ryan Ford	0241-2343444
005	Bruce Smith	0241-8472198

Bảng 1.8: Customer

Item_No	Description	Price
HW1	Cấp nguồn	4000
HW2	Bàn phím	2000
HW3	Chuột	800
SW1	Office Suite	15000
SW2	Phần mềm tiền lương	8000

Bảng 1.9: Items

Ord_No	Item_No	Qty
101	HW3	50
101	SW1	150
102	HW2	10
103	HW3	50
104	HW2	25
104	HW3	100
105	SW1	100

Bảng 1.10 Order_Details

Ord_No	Ord_Date	Cust_No
101	02-08-12	002
102	11-08-12	003
103	21-08-12	003
104	28-08-12	002
105	30-08-12	005

Bảng 1.11 Order_August

Thuật ngữ	Nghĩa	Ví dụ từ kịch bản
Quan hệ	Bảng	Order_August, Order_Details, Customer và Items
Bộ dữ liệu	Một hàng hoặc một bản ghi trong một quan hệ	Một hàng từ quan hệ Customer là bộ Customer
Thuộc tính	Một trường hoặc một cột trong một quan hệ	Ord_Date, Item_No, Cust_Name, vân vân
Bản số của một quan hệ	Số lượng các bộ dữ liệu trong một quan hệ	Bản số của quan hệ Order_Details là 7
Mức độ của một quan hệ	Số các thuộc tính trong một quan hệ.	Mức độ quan hệ Customer là 3
Miền của một thuộc tính	Tập hợp tất cả các giá trị có thể được thuộc tính lấy	Miền của Qty trong Order_Details là tập hợp tất cả các giá trị có thể đại diện cho số lượng của một mục có thứ tự
Khóa chính của một quan hệ	Một thuộc tính hoặc tổ hợp các thuộc tính duy nhất định nghĩa mỗi bộ dữ liệu trong một quan hệ	Khóa chính của quan hệ Customer là Cust_No Tổ hợp Ord_No và Item_No tạo thành khóa chính của Order_Details
Khóa ngoại	Một thuộc tính hoặc tổ hợp các thuộc tính trong một quan hệ R1 cho biết mối quan hệ của R1 với quan hệ R2 khác Các thuộc tính phím ngoại tại R1 phải chứa các giá trị phù hợp với những giá trị trong R2	Cust_No trong quan hệ Order_August là khóa ngoại tạo ra tham chiếu từ Order_August cho Customer. Điều này là bắt buộc để chỉ ra mối quan hệ giữa các đơn đặt hàng trong Order_August và Customer

Bảng 1.12: Các thuật ngữ liên quan đến bảng

1.6.2 Người dùng RDBMS

Mục đích chính của hệ cơ sở dữ liệu là để cung cấp một môi trường để gọi ra thông tin từ và lưu trữ thông tin mới vào trong cơ sở dữ liệu.

Đối với cơ sở dữ liệu các nhân nhỏ, một người thường định nghĩa các hàm dụng và thao tác cơ sở dữ liệu. Tuy nhiên, nhiều người tham gia vào việc thiết kế, sử dụng và duy trì một cơ sở dữ liệu lớn với vài trăm Người dùng.

→ Người quản trị cơ sở dữ liệu (DBA)

DBA là người thu thập thông tin mà sẽ được lưu trữ trong cơ sở dữ liệu. Cơ sở dữ liệu được thiết kế để cung cấp đúng thông tin vào đúng thời điểm cho đúng người.

Việc quản trị các tài nguyên này là trách nhiệm của Người quản trị cơ sở dữ liệu. DBA chịu trách nhiệm cho việc cấp quyền truy cập vào cơ sở dữ liệu, để điều phối và giám sát việc sử dụng và để có được các tài nguyên phần mềm và phần cứng khi cần. DBA chịu trách nhiệm cho các vấn đề như sự vi phạm tính bảo mật hoặc thời gian phản hồi của hệ thống kém.

→ **Người thiết kế cơ sở dữ liệu**

Người thiết kế cơ sở dữ liệu chịu trách nhiệm cho việc nhận biết dữ liệu sẽ được lưu trữ trong cơ sở dữ liệu và cho việc chọn các cấu trúc thích hợp để biểu diễn và lưu trữ dữ liệu này. Trách nhiệm của người thiết kế cơ sở dữ liệu là liên lạc với tất cả những Người dùng cơ sở dữ liệu tương lai, để hiểu các yêu cầu của họ, và đưa ra thiết kế đáp ứng các yêu cầu này.

→ **Người phân tích hệ thống và người lập trình ứng dụng**

Người phân tích hệ thống xác định các yêu cầu của Người dùng cuối, và phát triển các thông số kỹ thuật cho các giao tác được xác định sẵn để đáp ứng các yêu cầu này. Các lập trình viên ứng dụng thực hiện những chi tiết kỹ thuật này làm các chương trình, sau đó, họ kiểm tra, gỡ lỗi, ghi tài liệu, và duy trì các giao tác được xác định trước.

Ngoài những người thiết kế, sử dụng và quản trị cơ sở dữ liệu, những người khác có liên quan đến thiết kế, phát triển và vận hành môi trường hệ thống và phần mềm DBMS.

→ **Người thiết kế và người triển khai DBMS**

Những người này thiết kế và thực hiện những mô-đun và giao diện DBMS như là một gói phần mềm. DBMS là một hệ thống phần mềm phức tạp có chứa nhiều thành phần hoặc mô-đun, bao gồm các mô-đun cho việc triển khai danh mục, ngôn ngữ truy vấn, bộ xử lý giao diện, truy cập dữ liệu, và bảo mật. DBMS phải có giao diện với phần mềm của hệ thống khác như là hệ điều hành và các trình biên dịch cho các ngôn ngữ lập trình khác nhau.

→ **Người dùng cuối**

Người dùng cuối gọi ra một ứng dụng để tương tác với hệ thống, hoặc viết một truy vấn để gọi ra, sửa đổi hoặc xóa dữ liệu dễ dàng.

1.7 Các thực thể và bảng

Các thành phần của một RDBMS là các thực thể và bảng, chúng sẽ được giải thích trong phần này.

1.7.1 Thực thể

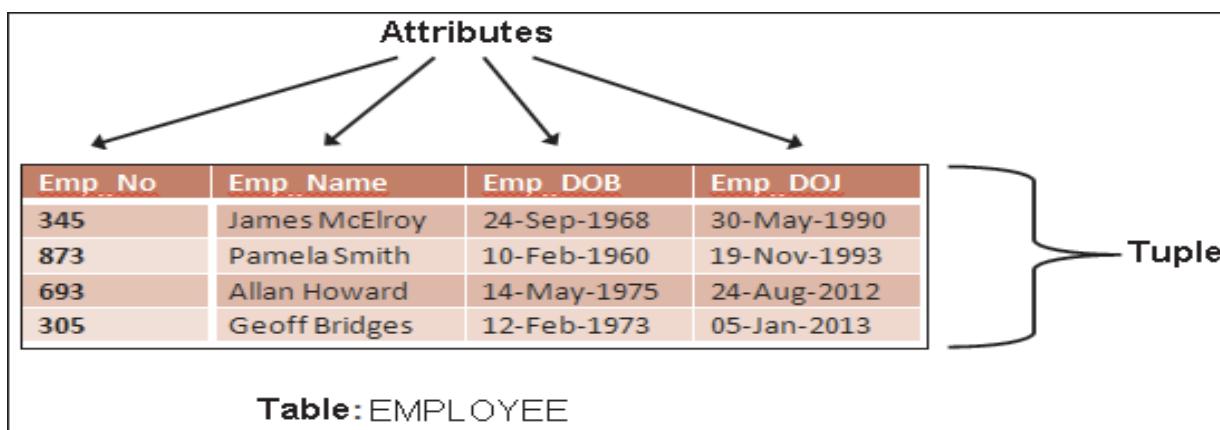
Thực thể là một người, nơi chốn, thứ, vật thể, sự kiện hoặc thậm chí một khái niệm, mà có thể được nhận dạng riêng biệt. Ví dụ, các thực thể trong một trường đại học là sinh viên, giảng viên, và các khóa học.

Mỗi thực thể có một số đặc tính được biết như là các thuộc tính. Ví dụ, thực thể học viên có thể bao gồm các thuộc tính như số học viên, tên, và lớp. Mỗi thuộc tính sẽ được đặt tên thích hợp.

Việc tạo nhóm các thực thể có liên quan tạo thành một tập hợp thực thể. Mỗi tập hợp thực thể được cho một cái tên. Tên của tập hợp thực thể phản ánh nội dung. Do đó, những thuộc tính của tất cả các sinh viên của trường đại học sẽ được lưu trữ trong một tập thực thể gọi là **Student**.

1.7.2 Bảng và các đặc điểm của bảng

Việc tạo ra các mối quan hệ dữ liệu dựa trên một hàm dựng được biết như là một bảng sẽ tạo điều kiện cho việc truy cập và thao tác dữ liệu. Bảng chứa một nhóm các thực thể có liên kết mà đó là một tập hợp thực thể. Các thuật ngữ tập hợp thực thể và bảng thường được dùng thay thế cho nhau. Bảng còn được gọi là quan hệ. Hàng được biết như là bộ dữ liệu. Cột được biết như là thuộc tính. Hình 1.6 làm nổi bật đặc điểm của một bảng.



Hình 1.6: Đặc điểm của một bảng

Các đặc điểm của một bảng như sau:

- Một cấu trúc hai chiều gồm các hàng và cột được biết đến như là bảng.
- Mỗi bộ dữ liệu biểu diễn một thực thể đơn ở trong tập hợp thực thể.
- Mỗi cột có một tên riêng.
- Mỗi giao cắt hàng/cột biểu diễn một giá trị dữ liệu đơn.
- Mỗi bảng phải có một khóa được biết như là khóa chính để nhận dạng duy nhất từng hàng.
- Mọi giá trị trong cột phải tuân thủ cùng định dạng dữ liệu. Ví dụ, nếu thuộc tính được gán một định dạng dữ liệu số thập phân, tất cả các giá trị trong cột thể hiện rằng thuộc tính phải là ở dạng số thập phân.
- Mỗi cột có một phạm vi giá trị cụ thể được biết như là miền thuộc tính.
- Mỗi hàng mang thông tin mô tả một lần xuất hiện thực thể.
- Thứ tự của các hàng và cột là không quan trọng trong DBMS.

1.8 Sự khác biệt giữa DBMS và RDBMS

Sự khác biệt giữa một DBMS và RDBMS được liệt kê trong bảng 1.13.

DBMS	RDBMS
Không cần phải có dữ liệu trong cấu trúc dạng bảng và cũng không bắt buộc các mối quan hệ dạng bảng giữa các mục dữ liệu.	Trong RDBMS, cấu trúc bảng là sự bắt buộc và các mối quan hệ bảng được thực thi bởi hệ thống. Các mối quan hệ này cho phép Người dùng áp dụng và quản lý các quy tắc kinh doanh với việc lập mã tối thiểu.
Có thể lưu trữ và gọi ra số lượng nhỏ dữ liệu.	RDBMS có thể lưu trữ và lấy ra số lượng lớn dữ liệu.
DBMS ít an toàn hơn so với RDBMS.	RDBMS an toàn hơn so với DBMS.
Đây là một hệ thống Người dùng đơn.	Nó là một hệ thống nhiều người dùng.
Hầu hết các DBMS không hỗ trợ kiến trúc máy khách/máy chủ.	Hệ thống này hỗ trợ kiến trúc máy khách/máy chủ.

Bảng 1.13: Sự khác biệt giữa DBMS và RDBMS

Trong RDBMS, quan hệ có tầm quan trọng hơn. Do đó, các bảng trong RDBMS là dạng phụ thuộc và người dùng có thể thiết lập các ràng buộc toàn vẹn khác nhau trên những bảng này để dữ liệu cuối cùng được người dùng sử dụng vẫn còn đúng. Trong trường hợp của DBMS, các thực thể có tầm quan trọng hơn và không có quan hệ được thiết lập giữa những thực thể này.

1.9 Kiểm tra tiến bộ của bạn

1. Mô hình dữ liệu _____ cho phép một nút con có nhiều hơn một cha mẹ.

(A)	Tập tin phẳng	(C)	Mạng
(B)	Phân cấp	(D)	Quan hệ

2. _____ được sử dụng để quản trị quyền truy cập vào các cơ sở dữ liệu và các đối tượng cơ sở dữ liệu.

(A)	Ngôn ngữ định nghĩa dữ liệu (DDL)	(C)	Lược đồ phụ
(B)	Ngôn ngữ thao tác dữ liệu (DML)	(D)	Ngôn ngữ điều khiển dữ liệu (DCL)

3. Trong thuật ngữ mô hình quan hệ, hàng được gọi là _____, cột là _____, và bảng là _____.

(A)	thuộc tính, bộ dữ liệu, quan hệ	(C)	thuộc tính, quan hệ, bộ dữ liệu
(B)	bộ dữ liệu, thuộc tính, quan hệ	(D)	hang, cot, bo du lieu

4. _____ có thể được định nghĩa là một tập hợp các bản ghi có liên quan và một tập hợp các chương trình truy cập và thao tác những bản ghi này.

(A)	Hệ thống quản lý cơ sở dữ liệu	(C)	Quản lý dữ liệu
(B)	Hệ thống quan hệ cơ sở dữ liệu quan hệ	(D)	Mô hình mạng

5. _____ mô tả một bộ chứa để lưu trữ dữ liệu và quá trình lưu trữ và lấy dữ liệu từ bộ chứa.

(A)	Mô hình mạng	(C)	Mô hình dữ liệu
(B)	Mô hình tập tin phẳng	(D)	Mô hình quan hệ

1.9.1 Câu trả lời

1.	C
2.	D
3.	B
4.	B
5.	C

Tóm tắt

- Cơ sở dữ liệu là một tập hợp các dữ liệu có liên quan được lưu trữ ở dạng bảng.
- Mô hình dữ liệu mô tả một bộ chứa để lưu trữ dữ liệu và quá trình lưu trữ và lấy dữ liệu từ bộ chứa.
- DBMS là một bộ sưu tập các chương trình cho phép người dùng lưu trữ, chỉnh sửa, và lấy thông tin từ một cơ sở dữ liệu.
- Hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS) là một bộ các chương trình phần mềm để tạo ra, duy trì, sửa đổi và thao tác một cơ sở dữ liệu quan hệ.
- Cơ sở dữ liệu quan hệ được chia thành các đơn vị logic được gọi là bảng. Các đơn vị logic này được liên kết với nhau ở trong cơ sở dữ liệu.
- Những thành phần chính của một RDBMS là các thực thể và bảng.
- Trong RDBMS, mỗi quan hệ có tầm quan trọng hơn, trong khi đó, trong trường hợp của DBMS, các thực thể có tầm quan trọng hơn và không có quan hệ nào được thiết lập giữa những thực thể này.

**Nền tảng của mọi quốc gia là
giáo dục thế hệ thanh niên**

Phần - 2

Mô hình và chuẩn hóa thực thể-mối quan hệ (E-R)

Chào mừng bạn đến với phần **Mô hình và chuẩn hóa thực thể-mối quan hệ (E-R)**.

Phần này nói về Tạo mô hình dữ liệu, mô hình E-R, các thành phần, biểu tượng, sơ đồ, mối quan hệ, chuẩn hóa dữ liệu, và các toán tử quan hệ của nó.

Trong phần này, bạn sẽ học để:

- ➔ Xác định và mô tả tạo mô hình dữ liệu
- ➔ Nhận dạng và mô tả các thành phần của mô hình E-R
- ➔ Nhận dạng các mối quan hệ có thể được hình thành giữa các thực thể
- ➔ Giải thích các sơ đồ E-R và sử dụng chúng
- ➔ Mô tả một sơ đồ E-R, các biểu tượng được sử dụng để vẽ, và hiển thị các mối quan hệ khác nhau
- ➔ Mô tả các Biểu mẫu Bình thường khác nhau
- ➔ Phác thảo những mục đích sử dụng của các toán tử quan hệ khác nhau

2.1 Giới thiệu

Mô hình dữ liệu là một nhóm các công cụ khái niệm mô tả dữ liệu, các mối quan hệ và ngữ nghĩa của nó. Nó còn bao gồm các ràng buộc tính nhất quán mà dữ liệu gắn chặt vào. Các mô hình Thực thể-mối quan hệ, Quan hệ, Mạng, và Phân cấp là các ví dụ về mô hình dữ liệu. Sự phát triển của tất cả các cơ sở dữ liệu bắt đầu với bước cơ bản là phân tích dữ liệu của nó để xác định mô hình dữ liệu thể hiện tốt nhất cho nó. Một khi bước này đã hoàn thành, mô hình dữ liệu được áp dụng cho dữ liệu.

2.2 Tạo mô hình dữ liệu

Quy trình này áp dụng mô hình dữ liệu phù hợp với dữ liệu, để tổ chức và cấu trúc nó, được gọi là mô hình dữ liệu.

Việc tạo mô hình dữ liệu hết sức quan trọng đối với phát triển cơ sở dữ liệu cũng như việc lên kế hoạch và thiết kế đối với bất kỳ phát triển dự án nào. Việc xây dựng một cơ sở dữ liệu mà không có mô hình dữ liệu tương tự như phát triển một dự án mà không có các kế hoạch và thiết kế. Mô hình dữ liệu giúp các nhà phát triển cơ sở dữ liệu định nghĩa các bảng quan hệ, khóa chính và khóa ngoại, các thủ tục đã lưu trữ, và các trigger cần thiết trong cơ sở dữ liệu.

Tạo mô hình dữ liệu có thể được chia thành ba bước rộng sau đây:

→ **Tạo mô hình dữ liệu khái niệm**

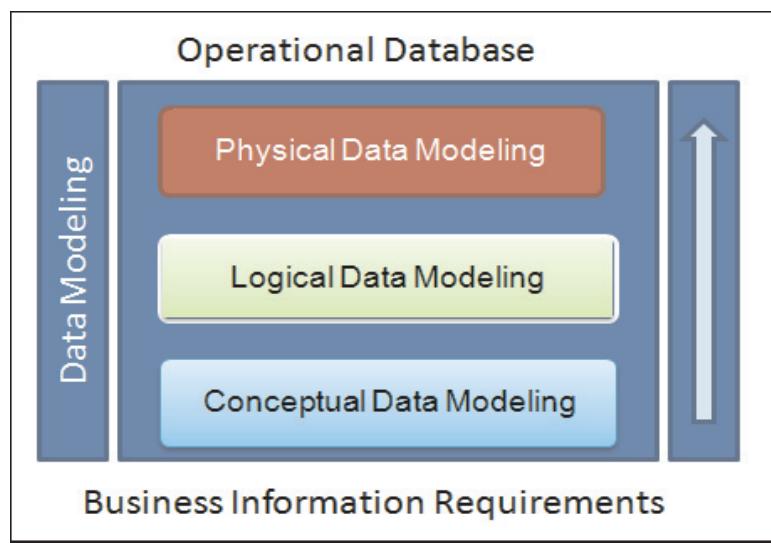
Bộ mô hình dữ liệu nhân sang mức mối quan hệ cao nhất trong dữ liệu.

→ **Tạo mô hình dữ liệu logic**

Trình mô hình dữ liệu mô tả dữ liệu và các mối quan hệ của nó một cách chi tiết. Bộ mô hình dữ liệu tạo ra một mô hình logic của cơ sở dữ liệu.

→ **Tạo mô hình dữ liệu vật lý**

Bộ mô hình dữ liệu chỉ ra cách mô hình logic được thực hiện về mặt vật lý. Hình 2.1 trình bày các bước khác nhau tham gia vào việc tạo mô hình dữ liệu.



Hình 2.1: Các bước tạo mô hình dữ liệu

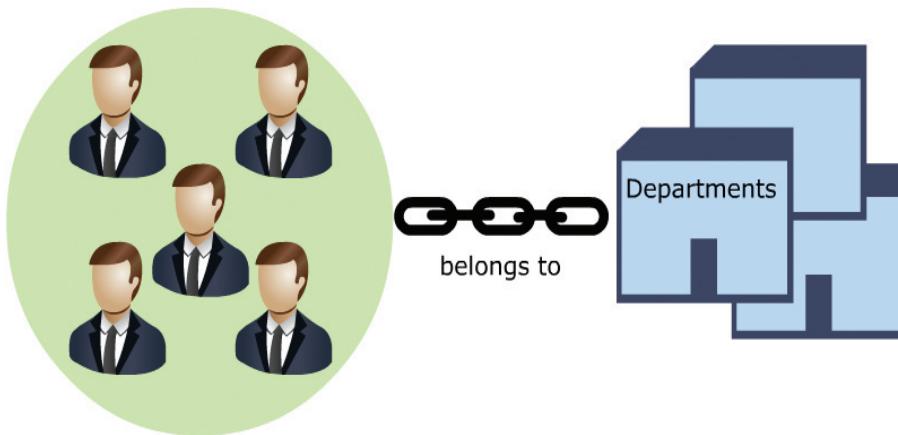
2.3 Mô hình Thực thể-mối quan hệ (E-R)

Có thể phân loại các mô hình dữ liệu thành ba nhóm khác nhau:

- Mô hình logic trên nền đối tượng
- Mô hình logic trên nền bản ghi
- Mô hình vật lý

Mô hình Thực thể-mối quan hệ (E-R) thuộc về phân loại đầu tiên. Mô hình được dựa trên ý tưởng đơn giản. Dữ liệu có thể được coi là đối tượng trong thế giới thực được gọi là các thực thể và các mối quan hệ tồn tại giữa chúng. Ví dụ, dữ liệu về nhân viên làm việc cho một tổ chức có thể được coi là tập hợp nhân viên và tập hợp các phòng ban khác nhau để hình thành lên một tổ chức. Cả nhân viên và phòng ban là các đối tượng trong thế giới thực. Nhân viên thuộc về một phòng ban. Vì vậy, mối quan hệ “thuộc về” liên kết nhân viên với một phòng ban cụ thể.

Có thể tạo mô hình quan hệ nhân viên-phòng ban như được trình bày trong hình 2.2.



Hình 2.2: Mô tả mô hình E-R của tổ chức

Mô hình E-R gồm năm thành phần cơ bản. Đó là:

→ **Thực thể**

Thực thể là một đối tượng trong thế giới thực tồn tại ở dạng vật chất và có thể phân biệt với các đối tượng khác. Ví dụ, nhân viên, phòng ban, học viên, khách hàng, xe cộ, và tài khoản là các thực thể.

→ **Mối quan hệ**

Mối quan hệ là một sự liên kết hoặc gắn kết tồn tại giữa một hoặc nhiều thực thể. Ví dụ, thuộc về, sở hữu, làm việc cho, tiết kiệm trong, được mua, và vân vân.

→ **Các thuộc tính**

Thuộc tính là tính năng mà một thực thể có. Các thuộc tính giúp phân biệt một thực thể với những thực thể khác. Ví dụ, các thuộc tính của học viên sẽ là **roll_number**, **name**, **stream**, **semester**, vân vân.

Các thuộc tính của xe hơi sẽ là `registration_number`, `model`, `manufacturer`, `color`, `price`, `owner`, vân vân.

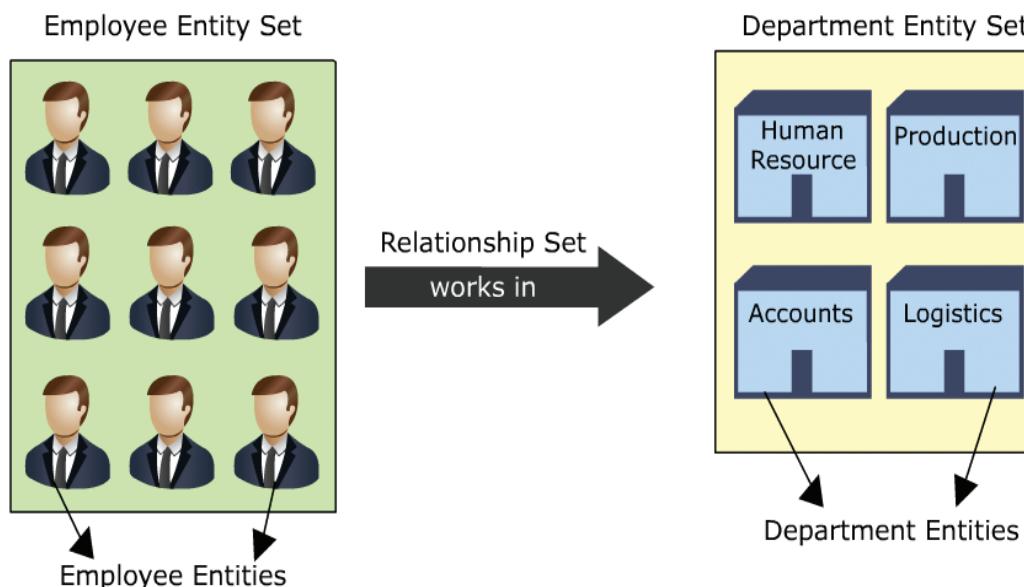
→ **Tập hợp thực thể**

Tập hợp thực thể là việc gom lại các thực thể tương tự. Ví dụ, các nhân viên của một tổ chức hình thành một tập hợp thực thể được gọi là tập hợp thực thể nhân viên.

→ **Tập hợp mối quan hệ**

Việc gom lại các mối quan hệ tương tự giữa hai hoặc nhiều tập hợp thực thể được gọi là tập hợp mối quan hệ. Ví dụ, các nhân viên làm việc trong một phòng ban cụ thể. Tập hợp tất cả các quan hệ “làm việc trong” tồn tại giữa nhân viên và phòng ban được gọi là tập hợp mối quan hệ “làm việc trong”.

Các thành phần mô hình E-R khác nhau có thể được thấy trong hình 2.3.



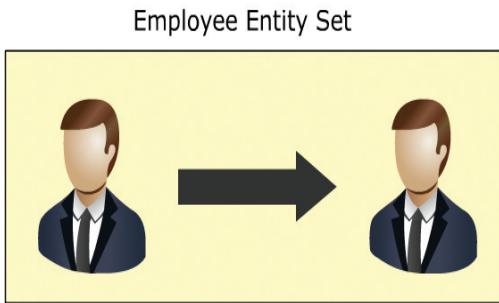
Hình 2.3: Các thành phần của mô hình E-R

Các mối quan hệ kết hợp một hoặc nhiều thực thể và có thể thuộc ba loại. Đó là:

→ **Các mối quan hệ tự thân**

Mối quan hệ giữa các thực thể của cùng một tập thực thể được gọi là mối quan hệ tự thân. Ví dụ, người quản lý và thành viên nhóm của anh ta, cả hai đều thuộc về tập hợp thực thể nhân viên. Thành viên nhóm làm việc cho người quản lý. Vì vậy, mối quan hệ “làm việc cho” tồn tại giữa hai thực thể nhân viên khác nhau thuộc cùng một tập thực thể nhân viên.

Có thể thấy mối quan hệ này trong hình 2.4.

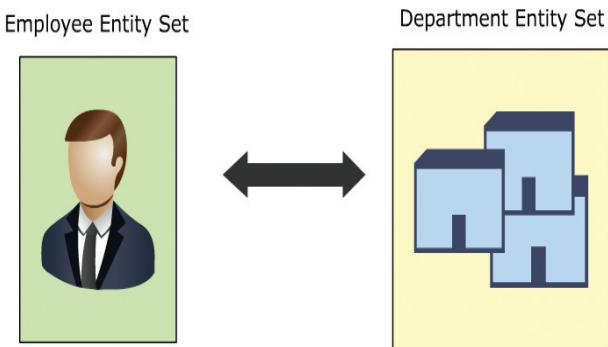


Hình 2.4: Mối quan hệ tự thân

→ **Mối quan hệ hai ngôi**

Mối quan hệ này tồn tại giữa các thực thể của hai tập hợp thực thể khác nhau được gọi là mối quan hệ hai ngôi. Ví dụ, một nhân viên thuộc về một phòng ban. Quan hệ tồn tại giữa hai thực thể khác nhau, thuộc hai tập thực thể khác nhau. Thực thể nhân viên thuộc về tập hợp thực thể phòng ban. Thực thể phòng ban thuộc về tập hợp thực thể phòng ban.

Có thể thấy mối quan hệ này trong hình 2.5.

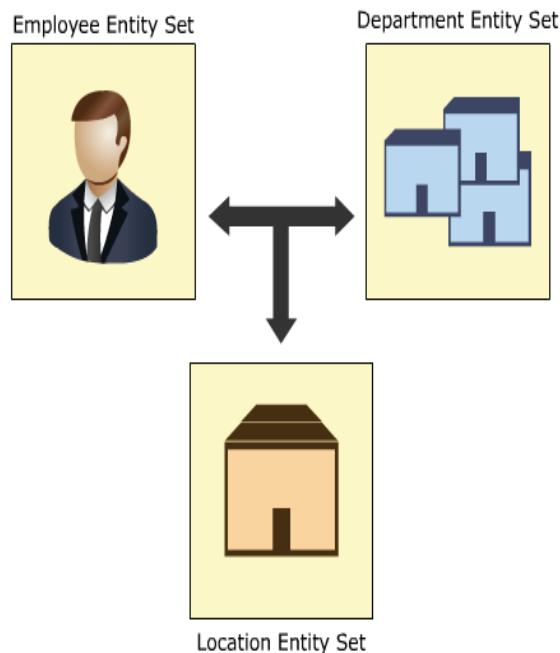


Hình 2.5: Mối quan hệ nhị phân

→ **Mối quan hệ ba ngôi**

Mối quan hệ tồn tại giữa ba thực thể của các tập hợp thực thể khác nhau được gọi là mối quan hệ ba ngôi. Ví dụ, nhân viên làm việc trong phòng kế toán ở chi nhánh khu vực. Quan hệ, “làm việc” tồn tại giữa cả ba: nhân viên, phòng ban, và vị trí.

Có thể thấy mối quan hệ này trong hình 2.6.



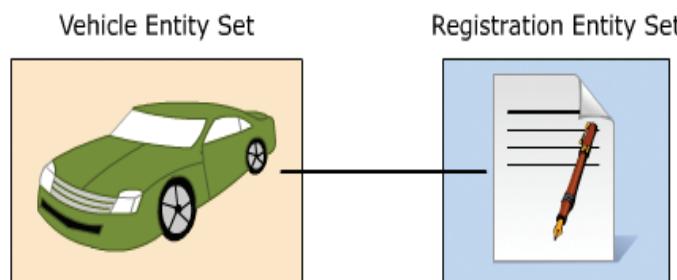
Hình 2.6: Mối quan hệ tam nguyên

Cũng có thể phân loại các mối quan hệ phù hợp với các yếu tố ánh xạ. Các bản số ánh xạ khác nhau như sau:

➔ Một-một

Loại ánh xạ này tồn tại khi thực thể của một tập hợp thực thể có thể được kết hợp với chỉ một thực thể của tập hợp khác.

Xem xét mối quan hệ giữa một chiếc xe và số đăng ký xe. Mỗi chiếc xe có một số đăng ký không trùng. Không có hai xe có cùng chi tiết đăng ký. Quan hệ này là một-một, đó là, một chiếc xe-một đăng ký. Có thể thấy bản số ánh xạ này trong hình 2.7.



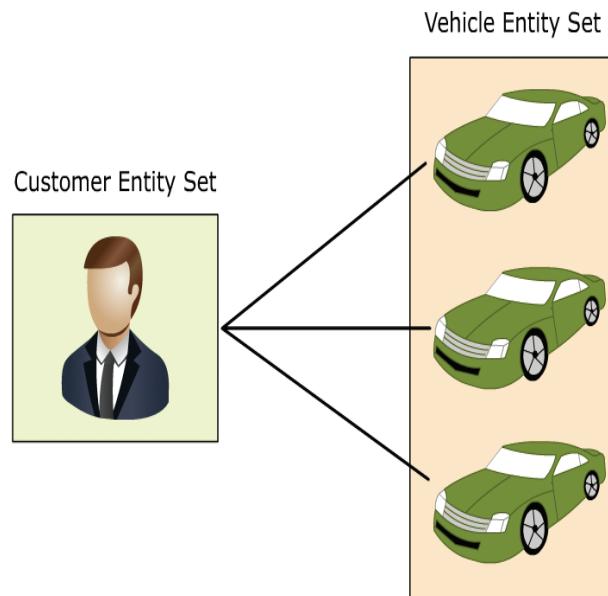
Hình 2.7: Bản số ánh xạ một-một

→ **Một-nhiều**

Loại ánh xạ này tồn tại khi thực thể của một tập hợp thực thể có thể được kết hợp với nhiều hơn một thực thể của tập hợp thực thể khác.

Xem xét quan hệ giữa khách hàng và các loại xe của khách hàng. Một khách hàng có thể có nhiều hơn một xe. Do vậy, việc ánh xạ này là ánh xạ một-nhiều, đó là một khách hàng - một hoặc nhiều xe.

Có thể thấy bản số ánh xạ này trong hình 2.8.



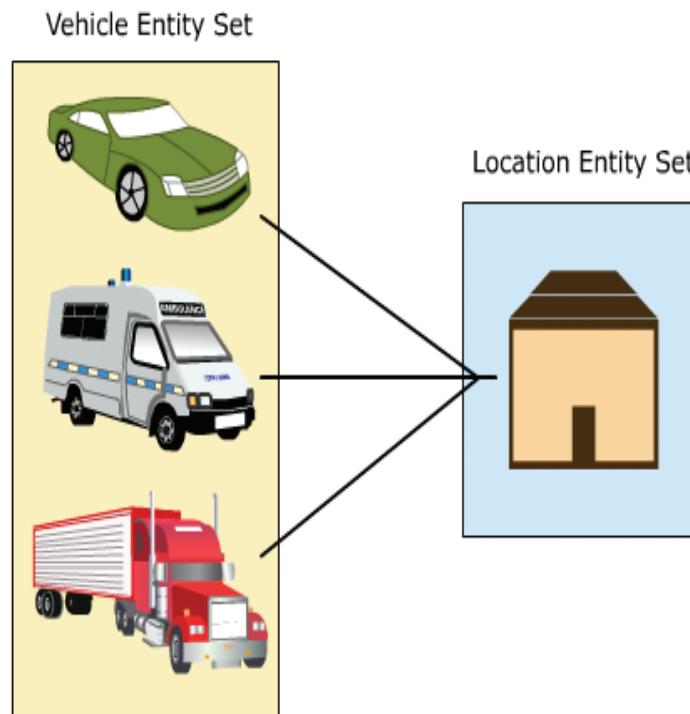
Hình 2.8: Bản số ánh xạ một-nhiều

→ **Nhiều-một**

Loại ánh xạ này tồn tại khi nhiều thực thể của một tập hợp được kết hợp với một thực thể của tập hợp khác. Sự kết hợp này được thực hiện bất kể là thực thể sau có được kết hợp với thực thể khác hoặc nhiều thực thể của tập hợp thực thể trước.

Xem xét quan hệ giữa một chiếc xe và nhà sản xuất của nó. Mỗi chiếc xe chỉ có một công ty hoặc liên minh sản xuất liên quan đến nó theo quan hệ “nhà sản xuất”, nhưng cùng một công ty hoặc liên minh có thể sản xuất nhiều hơn một loại xe.

Có thể thấy ánh xạ này trong hình 2.9.



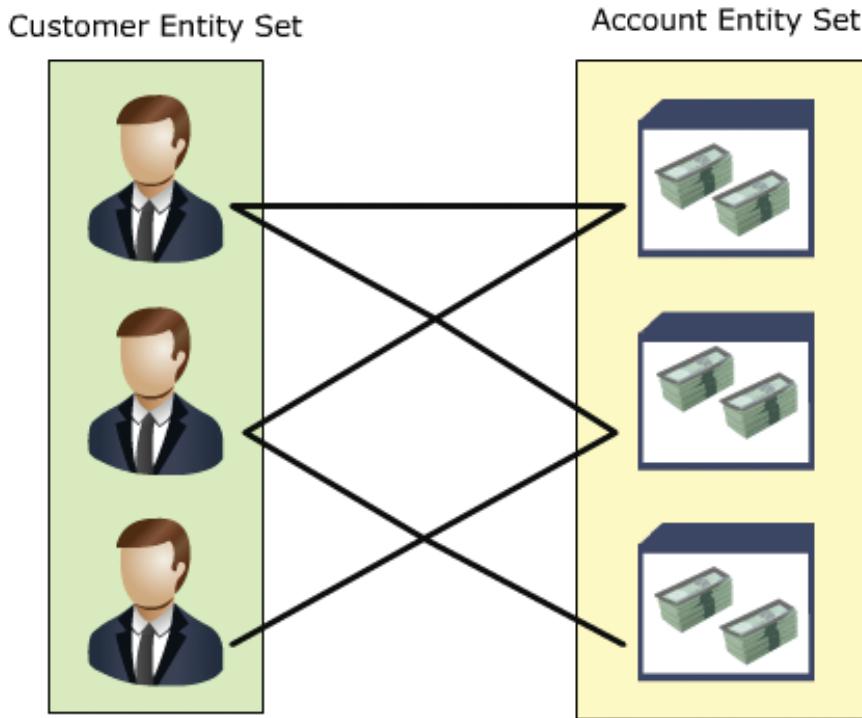
Hình 2.9: Bản sổ ánh xạ nhiều-một

→ **Nhiều-nhiều**

Loại ánh xạ này tồn tại khi bất kỳ số thực thể của một tập hợp có thể được kết hợp với bất kỳ số thực thể nào của tập hợp thực thể khác.

Xem xét mối quan hệ giữa khách hàng của ngân hàng và tài khoản của khách hàng. Khách hàng có thể có nhiều hơn một tài khoản và một tài khoản có thể có nhiều hơn một khách hàng có liên kết với tài khoản đó trong trường hợp đó là một tài khoản chung hoặc tương tự. Do vậy, ánh xạ này là nhiều-nhiều, đó là một hoặc nhiều khách hàng được kết hợp với một hoặc nhiều tài khoản.

Có thể thấy bản số ánh xạ này trong hình 2.10.



Hình 2.10: Bản số ánh xạ nhiều-nhiều

Một số khái niệm bổ sung trong mô hình E-R như sau:

→ **Khóa chính**

Khóa chính là một thuộc tính để có thể định nghĩa không trùng trong một tập hợp thực thể. Xem xét bảng 2.1 có chứa các chi tiết của học viên trong một trường học.

Enrollment_number	Name	Grade	Division
786	Ashley	Seven	B
957	Joseph	Five	A
1011	Kelly	One	A

Bảng 2.1: Các chi tiết của học sinh

Trong một trường học, mỗi học sinh có một **số đăng ký** duy nhất (như là **enrollment_number** trong bảng 2.1), là duy nhất cho học viên. Có thể nhận dạng bất kỳ học sinh nào dựa trên mã số học sinh. Do đó, thuộc tính **enrollment_number** đóng vai trò là khóa chính trong bảng **Student Details**.

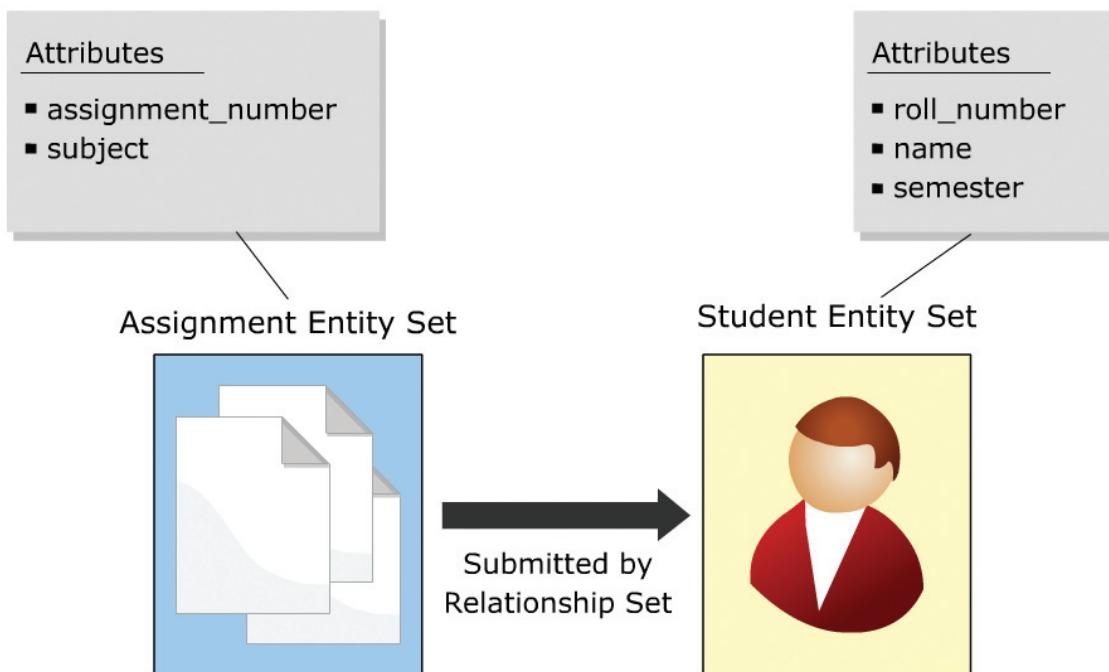
→ **Các tập hợp thực thể yếu**

Các tập hợp thực thể không có đủ các thuộc tính để thiết lập một khóa chính được gọi là các **tập hợp thực thể yếu**.

→ Các tập hợp thực thể mạnh

Các tập hợp thực thể có đủ các thuộc tính để thiết lập một khóa chính được gọi là các tập hợp thực thể mạnh.

Xem xét kịch bản của viện nghiên cứu giáo dục ở cuối mỗi học kỳ, học sinh được yêu cầu hoàn thành và nộp một loạt các bài tập. Giáo viên thường xuyên theo dõi các bài tập do học sinh nộp. Bây giờ, có thể xem một bài tập và một học sinh là hai thực thể riêng biệt. Thực thể gán được mô tả bằng các thuộc tính **assignment_number** và **subject**. Thực thể học viên được mô tả bằng **roll_number**, **name**, và **semester**. Có thể nhóm các thực thể bài tập để tạo thành một tập hợp thực thể bài tập và có thể nhóm các thực thể học sinh để tạo thành một tập hợp thực thể học viên. Các tập thực thể được liên kết bằng quan hệ “người gửi”. Quan hệ này được mô tả trong hình 2.11.



Hình 2.11: Quan hệ bài tập và học sinh

Các thuộc tính, **assignment_number** và **subject**, không đủ để xác định duy nhất một thực thể gán. Thực thể **roll_number** bản thân đã đủ để xác định duy nhất bất kỳ thực thể học viên. Do đó, **roll_number** là khóa chính cho tập thực thể học viên. Tập thực thể gán là một tập thực thể yếu vì nó thiếu một khóa chính. Tập thực thể học viên là một tập thực thể mạnh do có sự hiện diện của thuộc tính **roll_number**.

2.3.1 Biểu đồ thực thể-mối quan hệ

Biểu đồ E-R là một biểu diễn bằng đồ họa của mô hình E-R. Biểu đồ E-R, với sự giúp đỡ của các biểu tượng khác nhau, thể hiện hiệu quả các thành phần khác nhau của mô hình E-R.

Có thể thấy các biểu tượng được sử dụng cho các thành phần khác nhau trong bảng 2.2.

Thành phần	Ký hiệu	Ví dụ
Thực thể	Entity	Student
Thực thể yếu	Weak Entity	Assignments
Thuộc tính	Attribute	Roll_num
Mối quan hệ	Relationship	Saves in
Thuộc tính chính	Attribute	Acct_num

Bảng 2.2: Các biểu tượng của biểu đồ E-R

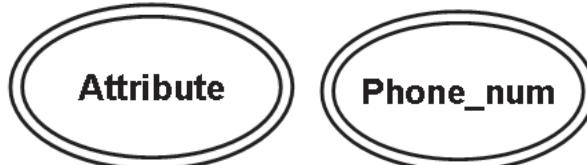
Các thuộc tính trong mô hình E-R có thể được phân loại thêm như sau:

➔ **Nhiều giá trị**

Thuộc tính nhiều giá trị được minh họa bằng một hình elip đôi dòng, trong đó có hơn một giá trị cho ít nhất một thể hiện của thực thể của nó. Thuộc tính này có thể có giới hạn trên và dưới được chỉ ra cho bất kỳ giá trị thực thể riêng lẻ nào.

Thuộc tính số điện thoại của một cá nhân có thể có một hoặc nhiều giá trị, đó là do cá nhân có thể có một hoặc nhiều số điện thoại. Do đó, thuộc tính điện thoại là một thuộc tính nhiều giá trị.

Có thể thấy biểu tượng và ví dụ về thuộc tính nhiều giá trị trong hình 2.12.

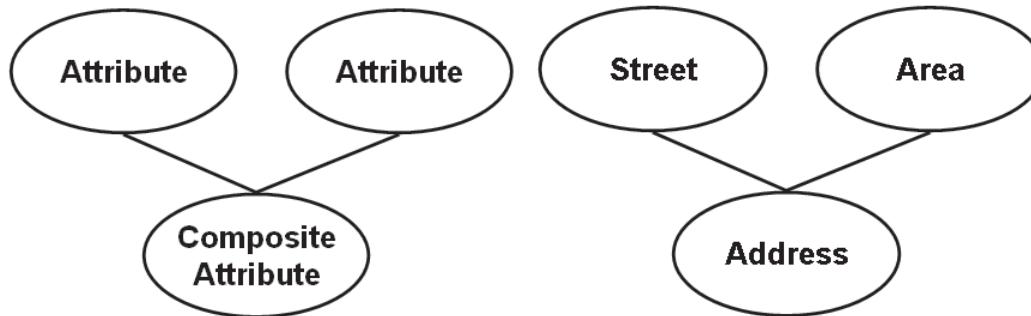


Hình 2.12: Biểu tượng và ví dụ về thuộc tính nhiều giá trị

→ **Tổng hợp**

Một thuộc tính tổng hợp bản thân nó có thể chứa hai hoặc nhiều thuộc tính, thể hiện các thuộc tính cơ bản có ý nghĩa độc lập của riêng mình.

Thuộc tính địa chỉ thường là một thuộc tính tổng hợp, bao gồm các thuộc tính như đường phố, khu vực, và vân vân. Có thể thấy biểu tượng và ví dụ về thuộc tính tổng hợp trong hình 2.13.



Hình 2.13: Biểu tượng và ví dụ về thuộc tính tổng hợp

→ **Được dẫn xuất**

Các thuộc tính được dẫn xuất là các thuộc tính mà giá trị của chúng phụ thuộc hoàn toàn vào thuộc tính khác và được chỉ ra bằng các hình elip đứt nét.

Thuộc tính tuổi của một người là ví dụ tốt nhất về các thuộc tính được dẫn xuất. Cho một thực thể người cụ thể, tuổi của một người có thể được xác định từ ngày hiện tại và ngày sinh của người đó. Có thể thấy biểu tượng và ví dụ về một thuộc tính dẫn xuất trong hình 2.14.



Hình 2.14: Biểu tượng và ví dụ về thuộc tính dẫn xuất

Bước để xây dựng một sơ đồ E-R như sau:

1. Thu thập tất cả các dữ liệu cần được mô hình.
2. Xác định dữ liệu có thể được mô hình hóa như các thực thể trong thế giới thực.
3. Nhận dạng các thuộc tính cho từng thực thể.

4. Sắp xếp các tập hợp thực thể thành các tập hợp thực thể yếu và mạnh.
5. Sắp xếp các thuộc tính thực thể như các thuộc tính khóa, thuộc tính nhiều giá trị, thuộc tính tổng hợp, thuộc tính dẫn xuất, và vân vân.
6. Nhận dạng các quan hệ giữa các thực thể khác nhau.

Việc sử dụng các biểu tượng khác nhau vẽ các thực thể, các thuộc tính của chúng, và mối quan hệ của chúng. Sử dụng các ký hiệu thích hợp trong khi vẽ các thuộc tính.

Xem xét kịch bản của một ngân hàng, với các khách hàng và tài khoản. Biểu đồ E-R cho kịch bản này có thể được dựng như sau:

Bước 1: Thu thập dữ liệu

Ngân hàng có một tập hợp các tài khoản được khách hàng sử dụng để gửi tiền.

Bước 2: Nhận dạng các thực thể

1. Khách hàng
2. Tài khoản

Bước 3: Nhận dạng các thuộc tính

1. Khách hàng: customer_name, customer_address, customer_contact
2. Tài khoản: account_number, account_owner, balance_amount

Bước 4: Sắp xếp các tập hợp thực thể

1. Tập hợp thực thể khách hàng: tập hợp thực thể yếu
2. Tập hợp thực thể tài khoản: tập hợp thực thể mạnh

Bước 5: Sắp xếp các thuộc tính

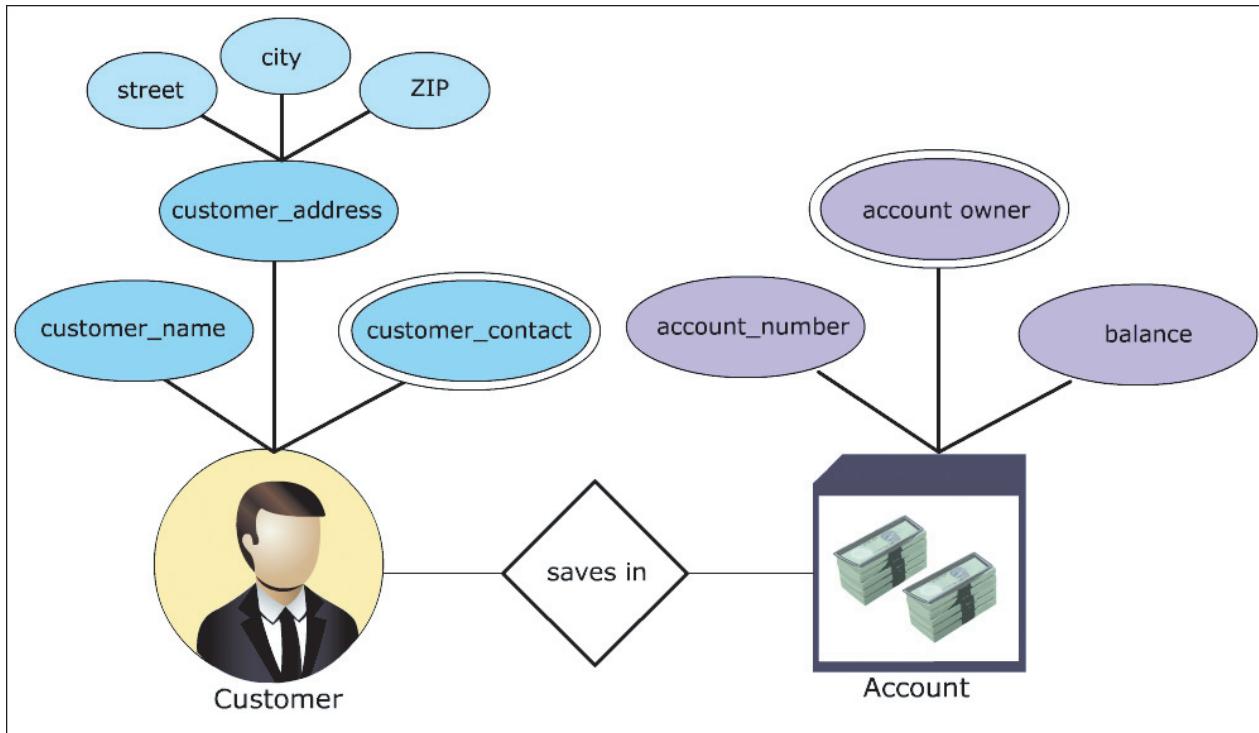
1. Tập thực thể khách hàng: customer_address - tổng hợp, customer_contact - nhiều giá trị
2. Tập thực thể tài khoản: account_number → khóa chính, account_owner – nhiều giá trị

Bước 6: Nhận dạng các quan hệ

Khách hàng “gửi tiết kiệm trong” một tài khoản. Quan hệ này là “gửi tiết kiệm trong”.

Bước 7: Vẽ biểu đồ bằng cách sử dụng các ký hiệu

Hình 2.15 trình bày sơ đồ E-R cho ngân hàng.



Hình 2.15: Sơ đồ E-R cho ngân hàng

2.4 Chuẩn hóa

Ban đầu, tất cả các cơ sở dữ liệu được biểu thị đặc điểm bằng số lượng lớn các cột và bản ghi. Phương thức này có một số nhược điểm. Xem xét những chi tiết sau đây của nhân viên trong một phòng ban. Bảng 2.3 bao gồm các chi tiết của nhân viên cũng như các chi tiết của dự án mà họ đang làm việc.

Emp_no	Project_id	Project_name	Emp_name	Grade	Salary
142	113, 124	BLUE STAR, MAGNUM	John	A	20.000
168	113	BLUE STAR	James	B	15.000
263	113	BLUE STAR	Andrew	C	10.000
109	124	MAGNUM	Bob	C	10.000

Bảng 2.3: Chi tiết về nhân viên của phòng

→ Sự dị thường lặp lại

Dữ liệu như là **Project_id**, **Project_name**, **Grade**, và **Salary** lặp lại nhiều lần. Việc lặp lại này cần trả cả hiệu suất trong khi truy vấn dữ liệu và cả dung lượng lưu trữ. Việc lặp lại dữ liệu này được gọi là **sự dị thường lặp lại**.

Sự lặp lại được trình bày trong bảng 2.4 với sự giúp đỡ của các ô bóng mờ.

Emp_no	Project_id	Project_name	Emp_name	Grade	Salary
142	113, 124	BLUE STAR, MAGNUM	John	A	20.000
168	113	BLUE STAR	James	B	15.000
263	113	BLUE STAR	Andrew	C	10.000
109	124	MAGNUM	Bob	C	10.000

Bảng 2.4: Chi tiết về nhân viên của phòng

→ **Sự dị thường chèn vào**

Giả sử phòng ban tuyển dụng một nhân viên mới có tên là **Ann**. Bây giờ, hãy xem xét **Ann** đã không được giao bất kỳ dự án nào. Việc chèn các chi tiết của cô ấy vào bảng sẽ để cột **Project_id** và **Project_name** trống. Việc để các cột trống có thể dẫn đến các vấn đề sau này. Sự dị thường do các lần chèn vào như vậy tạo ra được gọi là sự dị thường chèn vào. Có thể thấy sự bất thường này trong hình 2.5.

Emp_no	Project_id	Project_name	Emp_name	Grade	Salary
142	113, 124	BLUE STAR, MAGNUM	John	A	20.000
168	113	BLUE STAR	James	B	15.000
263	113	BLUE STAR	Andrew	C	10.000
109	124	MAGNUM	Bob	C	10.000
195	-	-	Ann	C	10.000

Bảng 2.5: Chi tiết về nhân viên của phòng

→ **Sự dị thường xóa**

Giả sử, Bob được rút ra khỏi dự án MAGNUM. Việc xóa bản ghi cũng sẽ xóa các chi tiết **Emp_no**, **Grade**, và **Salary** của Bob. Việc mất dữ liệu này sẽ có hại bởi tất cả các chi tiết cá nhân của Bob cũng bị mất như được thấy trong bảng 2.6. Loại mất dữ liệu này do việc xóa được gọi là sự dị thường xóa. Có thể thấy sự bất thường này trong hình 2.6.

Emp_no	Project_id	Project_name	Emp_name	Grade	Salary
142	113, 124	BLUE STAR, MAGNUM	John Smith	A	20.000
168	113	BLUE STAR	James Kilber	B	15.000
263	113	BLUE STAR	Andrew Murray	C	10.000

Bảng 2.6: Chi tiết dự án nhân viên

→ **Sự dị thường cập nhật**

Giả sử John được tăng lương trong **Salary** hay John bị giáng chức. Sự thay đổi trong **Salary** hoặc **Grade** của John cần phải được phản ánh trong tất cả các dự án mà John làm việc. Vấn đề trong việc cập nhật tất cả các sự cố này được gọi là sự dị thường cập nhật.

Bảng **Department Employee Details** được gọi là bảng không chuẩn hóa. Các nhược điểm này dẫn đến nhu cầu tiêu chuẩn hóa.

Tiêu chuẩn hóa là quá trình loại bỏ các sự dư thừa và các phần phụ thuộc không mong muốn.

Ban đầu, Codd (1972) trình bày ba dạng chuẩn (1NF, 2NF, và 3NF), tất cả đều dựa trên sự phụ thuộc giữa các thuộc tính của quan hệ. Dạng chuẩn thứ tư và thứ năm dựa trên sự phụ thuộc nhiều giá trị và kết hợp và đã được đề xuất sau.

2.4.1 Dạng chuẩn đầu tiên

Để đạt được dạng chuẩn đầu tiên, các bước sau đây cần phải được thực hiện:

- Tạo các bảng riêng cho từng nhó dữ liệu có liên quan
- Các cột của bảng phải có các giá trị nguyên tử
- Tất cả các thuộc tính chính phải được nhận dạng

Xem xét bảng **Employee Project Details** được trình bày trong bảng 2.7.

Emp_no	Project_id	Project_name	Emp_name	Grade	Salary
142	113, 124	BLUE STAR, MAGNUM	John	A	20.000
168	113	BLUE STAR	James	B	15.000
263	113	BLUE STAR	Andrew	C	10.000
109	124	MAGNUM	Bob	C	10.000

Bảng 2.7: Chi tiết dự án nhân viên

Bảng có dữ liệu liên quan đến các dự án và nhân viên. Bảng này cần phải được chia thành hai bảng, đó là bảng **Project Details** và bảng **Employee Details**. Các cột của bảng **Project_id** và **Project_names** có nhiều giá trị. Dữ liệu cần được tách ra trên các hàng khác nhau. Các bảng kết quả là **Project Details** và **Employee Details** như được trình bày trong bảng 2.8 và 2.9.

Project_id	Project_name
113	BLUE STAR
124	MAGNUM

Bảng 2.8: Chi tiết dự án

Emp_no	Emp_name	Grade	Salary
142	John	A	20.000
168	James	B	15.000
263	Andrew	C	10.000
109	Bob	C	10.000

Bảng 2.9: Chi tiết nhân viên

Thuộc tính **Project_id** là khóa chính cho bảng **Project Details**.

Thuộc tính `Emp_no` là khóa chính cho bảng `Employee Details`. Vì vậy, dạng chuẩn đầu tiên, bảng `Employee Project Details` ban đầu đã được giảm xuống thành bảng `Project Details` và `Employee Details`.

2.4.2 Dạng chuẩn thứ hai

Các bảng được gọi là ở dạng chuẩn thứ hai nếu:

- ➔ Chúng đáp ứng các yêu cầu của dạng chuẩn thứ nhất
- ➔ Không có các phần phụ thuộc một phần trong các bảng
- ➔ Các bảng được liên kết qua các khóa ngoại

Phụ thuộc một phần có nghĩa là thuộc tính không khóa không phải là phụ thuộc một phần vào nhiều hơn một thuộc tính khóa. Các bảng `Project Details` và `Employee Details` không thể hiện bất kỳ sự phụ thuộc một phần nào. `Project_name` chỉ phụ thuộc vào `Project_id` và `Emp_name`, `Grade`, và `Salary` chỉ phụ thuộc vào `Emp_no`. Các bảng còn cần được liên kết qua các khóa ngoại. Bảng thứ ba, có tên là `Employee Project Details`, được tạo ra chỉ có hai cột `Project_id` và `Emp_no`.

Vì vậy, các bảng chi tiết dự án và nhân viên khi chuyển đổi sang dạng chuẩn thứ hai tạo ra bảng `Project Details`, `Employee Details`, và `Employee Project Details` như được trình bày trong bảng 2.10, 2.11 và 2.12.

Project_id	Project_name
113	BLUE STAR
124	MAGNUM

Bảng 2.10: Chi tiết dự án

Emp_no	Emp_name	Grade	Salary
142	John	A	20.000
168	James	B	15.000
263	Andrew	C	10.000
109	Bob	C	10.000

Bảng 2.11: Chi tiết nhân viên

Emp_no	Project_id
142	113
142	124
168	113
263	113
109	124

Bảng 2.12: Chi tiết dự án nhân viên

Những thuộc tính **Emp_no** và **Project_id**, của bảng **Employee Project Details** kết hợp với nhau để tạo thành khóa chính. Các khóa chính như vậy được gọi là các khóa chính tổng hợp.

2.4.3 Dạng chuẩn thứ ba

Để đạt được dạng chuẩn thứ ba:

- Bảng phải đáp ứng các yêu cầu của dạng chuẩn thứ hai
- Các bảng không được có các phần phụ thuộc bổ trợ trong chúng

Bảng **Project Details**, **Employee Details**, và **Employee Project Details** là ở dạng chuẩn thứ hai. Nếu thuộc tính có thể xác định được bằng thuộc tính khác không phải khóa, nó được gọi là phần phụ thuộc bổ trợ. Để làm cho đơn giản hơn, mỗi thuộc tính không phải khóa sẽ chỉ được xác định bằng thuộc tính khóa. Nếu một thuộc tính không khóa có thể được xác định bằng một thuộc tính không khóa khác, nó cần phải đưa vào bảng khác.

Khi quan sát các bảng khác nhau, người ta thấy rằng các bảng **Project Details** và **Employee Project Details** không thể hiện bất kỳ sự phụ thuộc bắc cầu nào như vậy. Những thuộc tính không khóa này hoàn toàn được xác định bằng các thuộc tính khóa. **Project_name** chỉ được xác định bằng **Project_number**. Khi tiếp tục rà soát bảng **Employee Details**, thấy có sự không nhất quán nào đó. Thuộc tính **Salary** được xác định bởi thuộc tính **Grade** và không phải là thuộc tính khóa **Emp_no**. Vì vậy, phụ thuộc bắc cầu này cần phải được loại bỏ.

Bảng **Employee Details** có thể được tách thành bảng **Employee Details** và **Grade Salary Details** như được trình bày trong bảng 2.13 và 2.14.

Emp_no	Emp_name	Grade
142	John	A
168	James	B
263	Andrew	C
109	Bob	C

Bảng 2.13: Chi tiết nhân viên

Grade	Salary
A	20.000
B	15.000
C	10.000

Bảng 2.14: Bảng Grade Salary Details

Vì vậy, vào cuối của ba giai đoạn chuẩn hóa, bảng **Employee Project Details** ban đầu đã được giảm thành bảng **Project Details**, **Employee Project Details**, **Employee Details**, và **Grade Salary Details** như được trình bày trong bảng 2.15, 2.16, 2.17, và 2.18.

Project_id	Project_name
113	BLUE STAR
124	MAGNUM

Bảng 2.15: Chi tiết dự án

Emp_no	Project_id
142	113
142	124
168	113
263	113
109	124

Bảng 2.16: Chi tiết dự án nhân viên

Emp_no	Emp_name	Grade
142	John	A
168	James	B
263	Andrew	C
109	Bob	C

Bảng 2.17: Chi tiết nhân viên

Grade	Salary
A	20.000
B	15.000
C	10.000

Bảng 2.18: Chi tiết lương theo mức

2.4.4 Khử chuẩn hóa

Bằng cách tạo tiêu chuẩn hóa cơ sở dữ liệu, sự dư thừa đã được giảm. Điều này, đến lượt nó, làm giảm yêu cầu lưu trữ cho cơ sở dữ liệu và đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên, nó có một số nhược điểm. Đó là:

- ➔ Có thể thường xuyên phải viết các truy vấn kết hợp phức tạp để kết hợp dữ liệu trong nhiều bảng.
- ➔ Các kết hợp trên thực tế có thể tham gia nhiều hơn ba bảng, phụ thuộc vào nhu cầu thông tin.

Nếu các kết hợp như vậy được dùng rất thường xuyên, hiệu suất của cơ sở dữ liệu sẽ trở nên rất kém. Thời gian CPU cần để giải các truy vấn như vậy cũng sẽ rất lớn. Trong các trường hợp như vậy, có thể bỏ qua việc lưu một số trường không cần thiết để tăng hiệu suất của cơ sở dữ liệu. Những cơ sở dữ liệu có sự dư thừa nhỏ như vậy để tăng hiệu suất được gọi là cơ sở dữ liệu đã khử chuẩn hóa và quá trình làm như vậy được gọi là khử chuẩn hóa.

2.5 Các toán tử quan hệ

Mô hình quan hệ được dựa trên nền tảng vững chắc của Đại số quan hệ. Đại số quan hệ gồm một tập hợp các toán tử để tính toán trên các quan hệ. Mỗi toán tử lấy một hoặc hai quan hệ như là đầu vào và tạo ra một quan hệ mới như là đầu ra.

Hãy xem xét bảng **Branch Reserve Details** như được trình bày trong bảng 2.19.

Branch	Branch_id	Reserve (Tỷ €)
London	BS-01	9,2
London	BS-02	10
Paris	BS-03	15
Los Angeles	BS-04	50
Washington	BS-05	30

Bảng 2.19: Chi tiết dự trữ của chi nhánh

→ SELECT

Toán tử **SELECT** được sử dụng để trích xuất dữ liệu thỏa mãn một điều kiện nhất định. Chữ cái Hy Lạp sigma viết thường, ' σ ', được sử dụng để biểu thị lựa chọn. Hoạt động lựa chọn, trên bảng **Branch Reserve Details**, để hiển thị chi tiết của các chi nhánh ở London sẽ cho kết quả trong bảng 2.20.

Branch	Branch_id	Reserve (Tỷ €)
London	BS-01	9,2
London	BS-02	10

Bảng 2.20: Chi tiết của các chi nhánh ở London

Một lựa chọn trên bảng **Branch Reserve Details** để hiển thị các chi nhánh với trữ lượng lớn hơn 20 tỷ Euro sẽ cho kết quả trong bảng 2.21.

Branch	Branch_id	Reserve (Tỷ €)
Los Angeles	BS-04	50
Washington	BS-05	30

Bảng 2.21: Chi tiết của chi nhánh với trữ lượng lớn hơn 20 tỷ Euro

→ PROJECT

Toán tử PROJECT được sử dụng để đặt kế hoạch một số chi tiết của bảng quan hệ. Toán tử PROJECT chỉ hiển thị các chi tiết cần thiết để lại một số cột nhất định. Toán tử PROJECT được ký hiệu bằng chữ pi tiếng Hy Lạp, ' Π '. Giả định rằng chỉ có số lượng **Branch_id** và **Reserve** cần phải được hiển thị.

Phép toán lập kế hoạch để làm như vậy, trên bảng **Branch Reserve Details**, sẽ cho kết quả trong bảng 2.22.

Branch_id	Reserve (tỉ €)
BS-01	9,2
BS-02	10
BS-03	15
BS-04	50
BS-05	30

Bảng 2.22: Bảng kết quả với Branch_id và Số lượng dự trữ

→ PRODUCT

Toán tử PRODUCT, được ký hiệu bằng 'x' giúp kết hợp thông tin từ hai bảng quan hệ. Xem xét bảng 2.23.

Branch_id	Số tiền cho vay (tỷ €)
BS-01	0,56
BS-02	0,84

Bảng 2.23: Chi tiết tiền vay của chi nhánh

Phép tính product trên bảng **Branch Reserve Details** và **Branch Loan Details** sẽ cho ra kết quả trong bảng 2.24.

Branch	Branch_id	Reserve (tỉ €)	Số tiền cho vay (tỷ €)
London	BS-01	9,2	0,56
London	BS-01	9,2	0,84
London	BS-02	10	0,56
London	BS-02	10	0,84
Paris	BS-03	15	0,56
Paris	BS-03	15	0,84
Los Angeles	BS-04	50	0,56
Los Angeles	BS-04	50	0,84
Washington	BS-05	30	0,56
Washington	BS-05	30	0,84

Bảng 2.24: Tích số của Chi tiết dự trữ của chi nhánh và Chi tiết tiền vay của chi nhánh

Phép tính tích số kết hợp từng bản ghi từ bảng thứ nhất với tất cả bản ghi trong bảng thứ hai, đến lúc tạo ra tất cả các tổ hợp có thể giữa các bản ghi của bảng.

→ UNION

Giả sử một quan chức của ngân hàng có dữ liệu được đưa ra trong bảng 2.19 và 2.23 muốn biết chi nhánh nào có lượng dự trữ dưới 20 tỷ Euro hoặc cho vay. Bảng kết quả sẽ bao gồm các chi nhánh với lượng dự trữ dưới 20 tỷ Euro, cho vay hoặc cả hai.

Điều này cũng tương tự như sự kết hợp của hai bộ dữ liệu; đầu tiên, tập hợp các chi nhánh với lượng dự trữ ít hơn 20 tỷ Euro và thứ hai, các chi nhánh có cho vay. Chi nhánh với cả hai, dự trữ dưới 20 tỷ Euro và cho vay sẽ được hiển thị chỉ một lần. Toán tử **UNION** không chỉ như vậy, nó còn thu thập các dữ liệu từ các bảng khác nhau và trình bày một phiên bản hợp nhất của dữ liệu hoàn chỉnh. Phép tính hợp nhất được thể hiện bằng biểu tượng, 'U'. Phép tính union trên bảng **Branch Reserve Details** và **Branch Loan Details** sẽ tạo ra bảng 2.25.

Branch	Branch_id
London	BS-01
London	BS-02
Paris	BS-03

Bảng 2.25: Trình bày hợp nhất các chi nhánh với lượng dự trữ ít hoặc cho vay

→ INTERSECT

Giả sử cùng một viên chức sau khi nhìn thấy dữ liệu này muốn biết những chi nhánh nào trong số này có cả lượng dự trữ thấp và cho vay. Câu trả lời sẽ là phép tính quan hệ giao cắt. Toán tử **INTERSECT** tạo ra dữ liệu giữ đúng trong tất cả các bảng được áp dụng trên đó. Nó dựa trên lý thuyết tập giao điểm và được thể hiện bằng biểu tượng '∩'. Kết quả của giao điểm của bảng **Branch Reserve Details** và **Branch Loan Details** sẽ là danh sách các chi nhánh có cả lượng dự trữ dưới 20 tỷ Euro và cho vay trong tài khoản của họ. Bảng kết quả được tạo ra là bảng 2.26.

Branch	Branch_id
London	BS-01
London	BS-02

Bảng 2.26: Các chi nhánh có lượng dự trữ thấp và cho vay

→ DIFFERENCE

Nếu cùng một viên chức bây giờ muốn danh sách các chi nhánh có lượng dự trữ thấp nhưng không cho vay, sau đó viên chức này sẽ phải sử dụng phép toán khác biệt. Toán tử **DIFFERENCE**, được ký hiệu là '—SSSq, còn tạo ra dữ liệu từ các bảng khác nhau, nhưng nó tạo ra dữ liệu giữ true trong một bảng và không phải trong bảng kia. Do đó, chi nhánh sẽ phải có lượng dự trữ thấp và không có các khoản cho vay được hiển thị.

Bảng 2.27 là kết quả được tạo ra.

Branch	Branch_id
Paris	BS-03

Bảng 2.27: Các chi nhánh có lượng dự trữ thấp nhưng không cho vay

→ JOIN

Phép toán JOIN là phần nâng cao của phép toán sản phẩm. Nó cho phép một lựa chọn được thực hiện trên tích số của các bảng. Ví dụ, nếu các giá trị dự trữ và các khoản cho vay của chi nhánh có trữ lượng thấp và giá trị cho vay là cần thiết, sản phẩm của **Branch Reserve Details** và **Branch Loan Details** sẽ được yêu cầu. Một khi sản phẩm của bảng 2.19 và 2.23 được tạo ra, chỉ có những chi nhánh này sẽ được liệt kê trong đó có cả dự trữ dưới 20 tỷ Euro và cho vay. Bảng 2.28 được tạo ra như là kết quả của phép tính JOIN.

Branch	Branch_id	Reserve (tỉ €)	Số tiền cho vay (tỷ €)
London	BS-01	9,2	0,56
London	BS-02	10	0,84

Bảng 2.28: Danh sách chi tiết các chi nhánh với trữ lượng thấp và cho vay

→ DIVIDE

Giả sử một viên chức muốn xem tên các chi nhánh và lượng dự trữ của tất cả các chi nhánh có tiền vay. Quy trình này có thể được thực hiện rất dễ dàng bằng cách sử dụng toán tử DIVIDE. Tất cả những gì viên chức cần phải làm là chia bảng **Branch Reserve Details** (được trình bày trước đây trong bảng 2.19) bằng danh sách các chi nhánh, có nghĩa là, cột **Branch Id** của bảng **Branch Loan Details** (được trình bày trước đó trong bảng 2.23). Bảng 2.29 là kết quả được tạo ra.

Branch	Reserve (tỉ €)
London	9,2
London	10

Bảng 2.29: Bảng kết quả của phép tính chia

Lưu ý là các thuộc tính của bảng chia sẽ luôn là tập hợp phụ của bảng bị chia. Bảng kết quả sẽ luôn luôn không có giá trị của các thuộc tính của bảng số chia và các bản ghi không phù hợp với các bản ghi trong bảng số chia.

2.6 Kiểm tra tiến bộ của bạn

1. Một hoặc nhiều thuộc tính có thể định nghĩa duy nhất một thực thể từ một tập thực thể được gọi là khóa _____.

(A)	Chính	(C)	Luân phiên
(B)	Ngoại	(D)	Siêu

2. Một thuộc tính có chứa hai hoặc nhiều giá trị thuộc tính trong nó được gọi là thuộc tính _____.

(A)	Dẫn xuất	(C)	Nhiều giá trị
(B)	Tổng hợp	(D)	Mạng

3. Phụ thuộc bắc cầu được loại bỏ trong dạng chuẩn _____.

(A)	Đầu tiên	(C)	Thứ ba
(B)	Thứ hai	(D)	Thứ tư

4. Phép tính nào được tăng cường hơn nữa trong phép tính Sản phẩm?

(A)	Chia	(C)	Khác biệt
(B)	Giao cắt	(D)	Nối

5. Điều nào sau đây là những thành phần cơ bản của mô hình E-R?

- a. Thực thể
- b. Mối quan hệ
- c. Các thuộc tính
- d. Biểu đồ mối quan hệ
- e. Tập hợp mối quan hệ

(A)	a, b, c	(C)	a, c, e
(B)	a, d, c	(D)	a, b, c, e

2.6.1 Câu trả lời

1.	A
2.	B
3.	C
4.	D
5.	D

Tóm tắt

- ➔ Việc tạo mô hình dữ liệu là quá trình áp dụng một mô hình dữ liệu thích hợp cho dữ liệu có sẵn.
- ➔ Mô hình E-R xem thế giới thực như là một tập các đối tượng cơ bản và các mối quan hệ giữa chúng.
- ➔ Thực thể, thuộc tính, tập thực thể, mối quan hệ, và tập mối quan hệ hình thành năm thành phần cơ bản của mô hình E-R.
- ➔ Ánh xạ các bản sổ thể hiện số lượng các thực thể mà một thực thể có liên quan đến.
- ➔ Quá trình gỡ bỏ dữ liệu dư thừa khỏi các bảng của một cơ sở dữ liệu quan hệ được gọi là tiêu chuẩn hóa.
- ➔ Đại số quan hệ bao gồm tập hợp các toán tử để giúp truy vấn dữ liệu từ các cơ sở dữ liệu quan hệ.
- ➔ SELECT, PRODUCT, UNION, và DIVIDE là một số toán tử đại số quan hệ.

“
**Học cách để học là kỹ năng
quan trọng nhất trong cuộc sống.**
”

Phần - 3

Giới thiệu về SQL Server 2012

Chào mừng bạn đến với phần **Giới thiệu về SQL Server 2012**.

Phần này giải thích kiến trúc cơ bản của SQL Server 2012 và liệt kê các phiên bản và ấn bản của SQL Server. Nó còn giải thích vai trò và cấu trúc của SQL Server cùng với các tính năng mới được thêm vào SQL Server 2012. Cuối cùng, phần này giải thích quy trình kết nối với SQL Server, tạo và tổ chức các tập tin lệnh, và thực thi truy vấn Transact-SQL.

Trong phần này, bạn sẽ học để:

- ➔ Mô tả kiến trúc cơ bản của SQL Server 2012
- ➔ Liệt kê các phiên bản và ấn bản khác nhau của SQL Server
- ➔ Giải thích vai trò và cấu trúc của cơ sở dữ liệu SQL Server
- ➔ Liệt kê các tính năng mới của SQL Server 2012
- ➔ Giải thích quy trình kết nối với các thể hiện SQL Server
- ➔ Giải thích việc tạo và tổ chức tập tin lệnh
- ➔ Giải thích quy trình thực thi các truy vấn Transact-SQL

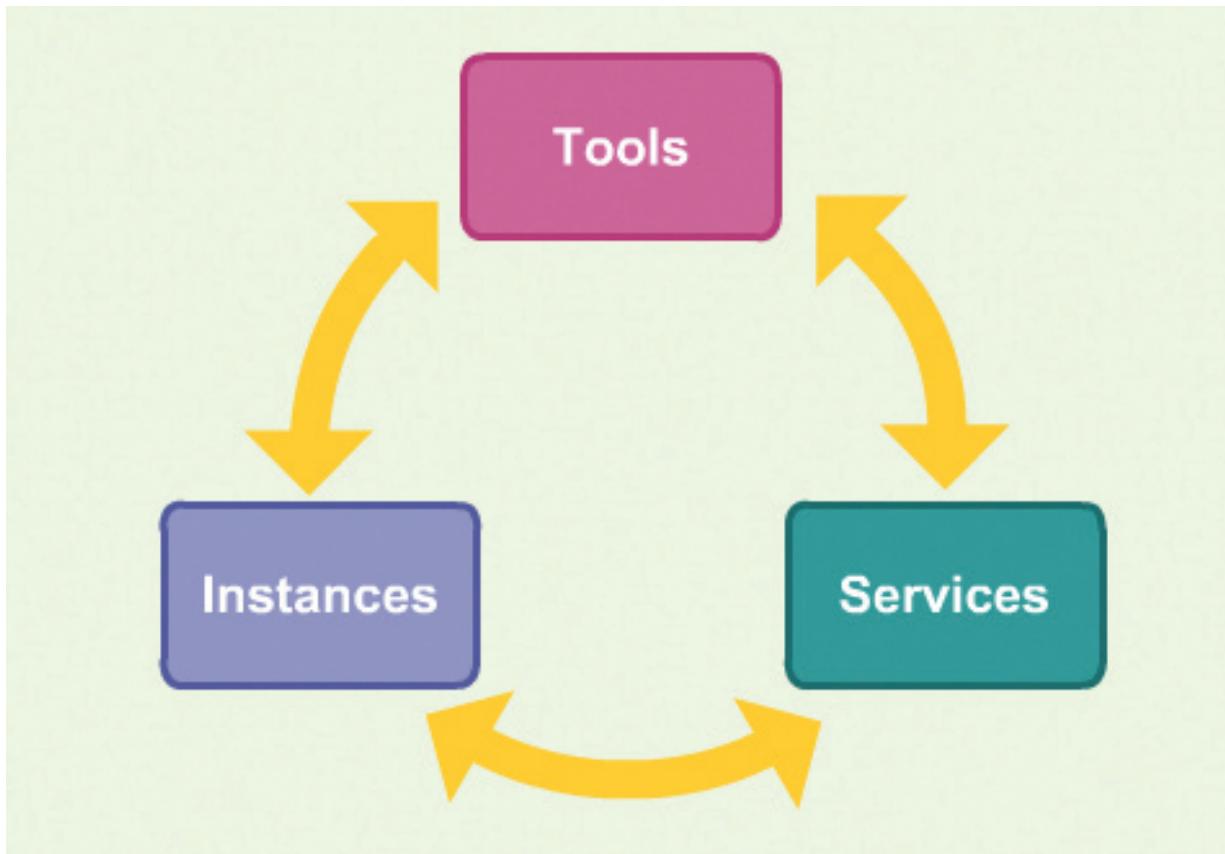
3.1 Giới thiệu

SQL Server là một RDBMS được phát triển bởi Microsoft. Hệ thống cung cấp một nền quản lý dữ liệu mức doanh nghiệp cho tổ chức. SQL Server bao gồm rất nhiều tính năng và công cụ làm cho nó thành một cơ sở dữ liệu và nền tảng phân tích dữ liệu xuất sắc. Nó còn được nhắm tới Xử lý giao tác trực tuyến (OLTP) quy mô lớn, lưu kho dữ liệu, và các ứng dụng thương mại điện tử.

SQL Server 2012 là phiên bản mới của SQL Server và đã được Microsoft đưa ra thị trường vào ngày 06 tháng 3 năm 2012. Một trong những tính năng chính của phiên bản SQL Server này là nó có sẵn trên nền tảng điện toán đám mây. Việc sử dụng SQL Server 2012 không chỉ giúp tổ chức lưu trữ và quản lý số lượng lớn thông tin, mà còn để bảo vệ và sử dụng dữ liệu này tại các địa điểm khác nhau theo yêu cầu.

3.2 Kiến trúc cơ bản của SQL Server 2012

Có những thành phần khác nhau hình thành một phần của SQL Server 2012. Tất cả các thành phần đến với nhau để tạo thành kiến trúc cơ bản của SQL Server 2012. Những thành phần này có thể được trình bày dưới ba đầu đề chính được trình bày trong hình 3.1.



Hình 3.1: Kiến trúc của SQL Server 2012

Công cụ

Có một số công cụ được cung cấp trong SQL Server 2012 để phát triển và quản lý truy vấn cơ sở dữ liệu. Trung tâm cài đặt SQL Server phải được sử dụng để cài đặt các tính năng và công cụ của chương trình SQL Server. Cũng có thể sửa đổi hoặc loại bỏ các tính năng bằng cách sử dụng Trung tâm cài đặt SQL Server. Bảng 3.1 liệt kê các công cụ khác nhau có sẵn trong SQL Server 2012.

Công cụ	Mô tả
SQL Server Management Studio (SSMS)	Một trong những công cụ quan trọng nhất có sẵn trong SQL Server 2012 là SSMS. SSMS là một ứng dụng được cung cấp với SQL Server 2012 giúp bạn tạo ra các cơ sở dữ liệu, đổi tượng cơ sở dữ liệu, truy vấn dữ liệu, và quản lý hoạt động tổng thể của SQL Server.
SQLCMD	SQLCMD là một công cụ dòng lệnh có thể được sử dụng thay cho SSMS. Nó thực hiện các chức năng tương tự như SSMS, nhưng chỉ ở định dạng lệnh.
Trung tâm cài đặt SQL Server	Công cụ Trung tâm cài đặt SQL Server cũng có thể được sử dụng để thêm, xóa bỏ, và sửa đổi các chương trình SQL Server.
SQL Server Configuration Manager	SQL Server Configuration Manager được các quản trị viên cơ sở dữ liệu sử dụng để quản lý các tính năng của phần mềm SQL đã cài đặt trong các máy khách. Công cụ này không có sẵn cho tất cả người dùng. Nó có thể được sử dụng để cấu hình những dịch vụ, các giao thức máy chủ, các giao thức máy khách, bí danh máy khách, và vân vân.
SQL Server Profiler	SQL Server Profiler được sử dụng để theo dõi một thể hiện của Công cụ Cơ sở dữ liệu hoặc Dịch vụ Phân tích.
Công cụ Dữ liệu SQL Server (SSDT)	SSDT là một Môi trường phát triển tích hợp (IDE) được sử dụng cho Các thành phần Nghiệp vụ Thông minh. Nó giúp thiết kế cơ sở dữ liệu sử dụng một công cụ có tên là Visual Studio.
Công cụ kết nối	Công cụ kết nối bao gồm DB-Library, Open Database Connectivity (ODBC), Object Linking and Embedding Database (OLE DB), vân vân. Những công cụ này được sử dụng để giao tiếp giữa các máy khách, máy chủ, và thư viện mạng.

Bảng 3.1: Các công cụ khác nhau trong SQL Server 2012

Dịch vụ

Có những dịch vụ khác nhau được thực thi trên một máy tính chạy SQL Server. Những dịch vụ này chạy cùng với các dịch vụ Windows khác và có thể được xem trong trình quản lý tác vụ. Một số dịch vụ SQL Server 2012 như sau:

- ➔ **SQL Server Database Engine** - Công cụ cơ sở dữ liệu là một dịch vụ cốt lõi được sử dụng để lưu trữ, xử lý và bảo mật dữ liệu. Nó còn được sử dụng cho nhân rộng, tìm kiếm toàn văn bản, và Chất lượng dịch vụ dữ liệu (DQS). Nó chứa các công cụ để quản lý dữ liệu quan hệ và eXtensible Markup Language (XML).
- ➔ **SQL Server Analysis Services** - Dịch vụ Phân tích chứa các công cụ giúp tạo và quản lý Online Analytical Processing (OLAP).

- ➔ Công cụ này được sử dụng cho mục đích cá nhân, nhóm, và nghiệp vụ thông minh của doanh nghiệp. Dịch vụ phân tích còn được sử dụng trong các ứng dụng khai thác dữ liệu. Những dịch vụ này còn giúp cộng tác với PowerPivot, Excel, và thậm chí Môi trường Máy chủ SharePoint.
- ➔ **SQL Server Reporting Services** - Dịch vụ báo cáo giúp tạo, quản lý, xuất bản, và triển khai các báo cáo. Những báo cáo này có thể là ở định dạng bảng, ma trận, đồ họa, hoặc dạng tự do. Ứng dụng báo cáo cũng có thể được tạo ra sử dụng Dịch vụ báo cáo.
- ➔ **SQL Server Integration Services** - Dịch vụ tích hợp được sử dụng để di chuyển, sao chép, và chuyển đổi dữ liệu sử dụng các công cụ đồ họa khác nhau và các đối tượng có thể lập trình. Thành phần DQS cũng được đưa vào trong Dịch vụ tích hợp. Dịch vụ tích hợp giúp xây dựng các giải pháp tích hợp dữ liệu hiệu suất cao.
- ➔ **SQL Server Master Data Services** - Dịch vụ dữ liệu chính (MDS) được sử dụng cho việc quản lý dữ liệu chính. MDS được sử dụng cho phân tích, quản lý và báo cáo thông tin như là hệ thống phân cấp, an ninh kỹ lưỡng, giao tác, quy tắc kinh doanh, vân vân.

Thể hiện

Tất cả các chương trình và phân bổ nguồn lực được lưu trong một thể hiện. Thể hiện có thể bao gồm bộ nhớ, các tập tin cấu hình, và CPU. Nhiều thể hiện có thể được sử dụng cho các người dùng khác nhau trong SQL Server 2012. Mặc dù nhiều thể hiện có thể có mặt trên một máy tính đơn lẻ, chúng không ảnh hưởng đến hoạt động của các thể hiện khác. Điều này có nghĩa rằng tất cả các thể hiện làm việc trong sự cô lập. Mỗi thể hiện có thể được tùy chỉnh theo yêu cầu. Thậm chí cho phép cho mỗi thể hiện có thể được cấp trên cơ sở riêng lẻ. Các nguồn tài nguyên cũng có thể được phân bổ cho thể hiện một cách phù hợp, ví dụ, số lượng các cơ sở dữ liệu được cho phép.

Nói cách khác, các thể hiện có thể được gọi là một bộ chứa lớn hơn có chứa các bộ chứa con ở dạng các cơ sở dữ liệu, tùy chọn bảo mật, đối tượng máy chủ, và vân vân.

Ghi chú - Sách trực tuyến (BOL) là một phần kiến trúc gián tiếp của SQL Server 2012 cung cấp thông tin về các khái niệm khác nhau liên quan đến SQL Server 2012. Điều này không chỉ bao gồm các tính năng và các thành phần mới của SQL Server 2012, mà còn có thông tin về các khía cạnh phát triển như là tạo cú pháp và tạo ra truy vấn. BOL có sẵn trực tuyến cũng như có thể được truy cập thông qua menu Help trong SSMS.

3.3 Các phiên bản của SQL Server

Phiên bản đầu tiên của SQL Server đã được phát hành trong năm 1989. Sau này, đã có các phiên bản mới phát hành gần như mỗi năm, với bản mới nhất là SQL Server 2012. Bảng 3.2 liệt kê các phiên bản khác nhau của SQL Server.

Phiên bản	Năm
SQL Server 1.0	1989
SQL Server 1.1	1991

Phiên bản	Năm
SQL Server 4.2	1992
SQL Server 6.0	1995
SQL Server 6.5	1996
SQL Server 7.0	1998
SQL Server 2000	2000
SQL Server 2005	2005
SQL Server 2008	2008
SQL Server 2008 R2	2010
SQL Server 2012	2012

Bảng 3.2: Các phiên bản khác nhau của SQL Server

3.4 Các phiên bản của SQL Server

Dựa trên các yêu cầu cơ sở dữ liệu, tổ chức có thể lựa chọn một trong ba phiên bản sau đây của SQL Server 2012 đã được phát hành. Những phiên bản chính này của SQL Server 2012 như sau:

- **Enterprise** – Đây là phiên bản được phát hành định kỳ trên hầu hết các phiên bản của SQL Server. Đây là phiên bản đầy đủ của SQL Server, trong đó có chứa tất cả các tính năng của SQL Server 2012. Phiên bản doanh nghiệp của SQL Server 2012 hỗ trợ các tính năng như là Power View, xVelocity, Dịch vụ Nghiệp vụ Thông minh, ảo hóa, và vân vân.
- **Standard** – Phiên bản tiêu chuẩn là phiên bản cơ bản của SQL Server hỗ trợ cơ sở dữ liệu cơ bản và chức năng báo cáo và phân tích. Tuy nhiên, nó không hỗ trợ phát triển ứng dụng quan trọng, bảo mật và lưu kho dữ liệu.
- **Business Intelligence** – Đây là một phiên bản mới được giới thiệu lần đầu tiên trong SQL Server 2012. Phiên bản này hỗ trợ cơ sở dữ liệu cơ bản, chức năng báo cáo và phân tích, và cả các dịch vụ nghiệp vụ thông minh. Phiên bản này hỗ trợ các tính năng như là PowerPivot, PowerView, mô hình ngữ nghĩa Nghiệp vụ Thông minh, Dịch vụ Dữ liệu Chính, và vân vân.

Bảng 3.3 cho thấy sự so sánh các tính năng có sẵn cho các phiên bản khác nhau của SQL Server 2012.

Các đặc điểm	Enterprise	Business Intelligence	Standard
Hỗ trợ không gian	Có	Có	Có
FileTable	Có	Có	Có
Quản lý dựa trên chính sách	Có	Có	Có
Báo cáo	Có	Có	Có
Phân tích	Có	Có	Có
Mô hình ngữ nghĩa Nghiệp vụ Thông minh đa chiều	Có	Có	Có

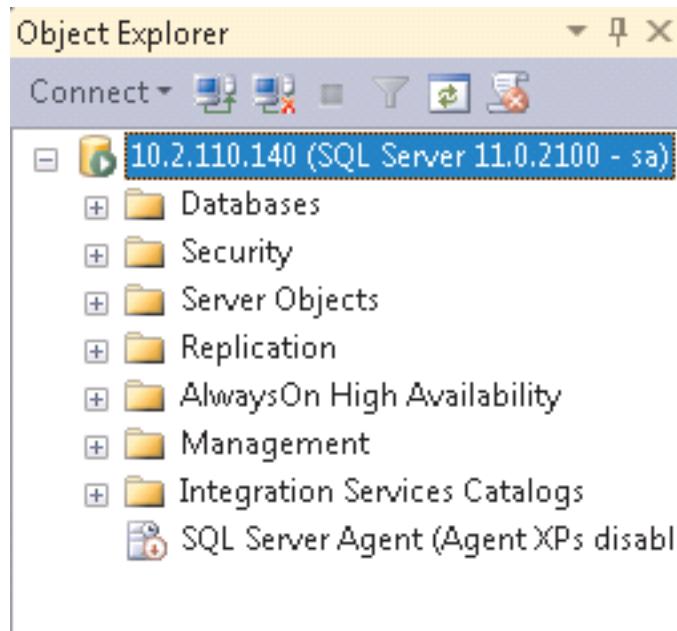
Các đặc điểm	Enterprise	Business Intelligence	Standard
Tính sẵn sàng cao cơ bản	Có	Có	Có
Khả năng tự phục vụ	Có	Có	
Cảnh báo	Có	Có	
Xem điện	Có	Có	
PowerPivot cho SharePoint Server	Có	Có	
Quản lý dữ liệu doanh nghiệp	Có	Có	
Dịch vụ chất lượng dữ liệu	Có	Có	
Dịch vụ dữ liệu chính	Có	Có	
Mô hình ngữ nghĩa Nghiệp vụ Thông minh dạng bảng trong bộ nhớ	Có	Có	
Ảo hóa không giới hạn	Có		
Lưu kho dữ liệu	Có		
Bảo mật nâng cao	Có		
Mã hóa dữ liệu trong suốt (TDE)	Có		
Nén và phân vùng	Có		
Tính sẵn sàng cao nâng cao	Có		

Bảng 3.3: So sánh các phiên bản khác nhau của SQL Server 2012

Khác với ba phiên bản này, cũng có các phiên bản khác có sẵn như là phiên bản Express, phiên bản Web, và phiên bản Developer. SQL Server 2012 Express là một phiên bản miễn phí của SQL Server 2012. Phiên bản Web được sử dụng cho môi trường các dịch vụ web dựa trên Internet. Phiên bản Developer được các lập trình viên sử dụng đặc biệt cho mục đích phát triển, thử nghiệm, và trình diễn.

3.5 Vai trò và cấu trúc của Object Explorer trong SQL Server

Cấu trúc của **Object Explorer** in SQL Server 2012 được trình bày trong hình 3.2. Cấu trúc bao gồm cơ sở dữ liệu, bảo mật, đối tượng máy chủ, nhân bản, và các tính năng như AlwaysOn High Availability, Management, Integration Services Catalogs, và vân vân.



Hình 3.2: Object Explorer

Các thành phần khác nhau trong **Object Explorer** như sau:

- **Databases** – Có một tập hợp các cơ sở dữ liệu lưu trữ một tập cụ thể các dữ liệu có cấu trúc.
- **Security** – Được sử dụng để cung cấp cấu hình bảo mật linh hoạt và đáng tin cậy trong SQL Server 2012. Phần này bao gồm thông tin đăng nhập, vai trò, chứng nhận, kiểm toán, và vân vân.
- **Server Objects** – Được sử dụng để giám sát hoạt động trong các máy tính chạy một thể hiện của SQL Server.
- **Replication** – Được sử dụng để sao chép và phân phối dữ liệu và các đối tượng cơ sở dữ liệu từ một cơ sở dữ liệu sang cơ sở dữ liệu khác, và sau đó, đồng bộ hóa giữa các cơ sở dữ liệu để duy trì tính nhất quán.
- **AlwaysOn High Availability** – Được sử dụng cho tính sẵn sàng cao và khôi phục thảm họa. Nó thường được sử dụng cho các ứng dụng đòi hỏi phải có thời gian hoạt động cao và bảo vệ chống sự cố.
- **Management** – Được sử dụng để quản lý các chính sách, nguồn lực, sự kiện, kế hoạch bảo trì, và vân vân.
- **Integration Services Catalogs** – Danh mục dịch vụ tích hợp lưu trữ tất cả các đối tượng của dự án sau khi dự án đã được triển khai.

3.6 Các tính năng mới của SQL Server 2012

Các tính năng mới được đưa vào trong SQL Server 2012 như sau:

- **Các thuộc tính thống kê** – Thông tin về số liệu thống kê của các đối tượng có thể được xem trong SQL Server 2012 bằng cách sử dụng hàm sys.dm_db_stats_properties.
- **Cải tiến tạo cụm chuyển đổi dự phòng** – SQL Server 2012 cung cấp các cụm chuyển đổi dự phòng đa subnet. Nó cũng đã giới thiệu các điểm kiểm tra gián tiếp và một chính sách chuyển đổi dự phòng linh hoạt để phát hiện sức khỏe cụm. Điều này đã tăng cường giải pháp khắc phục thảm họa hiện có trong SQL Server 2012.
- **SQL Azure** – Microsoft SQL Azure là một dịch vụ cơ sở dữ liệu quan hệ dựa trên đám mây, thúc đẩy các công nghệ SQL Server hiện có. Có thể sử dụng SQL Azure để lưu trữ và quản lý dữ liệu sử dụng các truy vấn và các hàm khác tương tự như SQL Server 2012. Dịch vụ báo cáo và tính năng sao lưu đã được thêm vào trong SQL Azure. Ngoài ra, bây giờ kích thước cơ sở dữ liệu trong SQL Azure có thể được tăng lên tối đa 150 Giga Byte (GB).
- **Các ứng dụng tầng dữ liệu** – Một phiên bản mới của Ứng dụng tầng dữ liệu (DAC) đã được đưa vào trong SQL Server 2012. DAC là một thực thể quản lý cơ sở dữ liệu logic định nghĩa các đối tượng SQL Server kết hợp với cơ sở dữ liệu của người dùng. Nâng cấp DAC này làm thay đổi cơ sở dữ liệu hiện có để so khớp sơ đồ với phiên bản mới của DAC.
- **Dịch vụ chất lượng dữ liệu** – Để duy trì tính toàn vẹn, sự phù hợp, tính nhất quán, tính chính xác, và tính hợp lệ của dữ liệu, Chất lượng dịch vụ dữ liệu (DQS) đã được tích hợp với SQL Server 2012. DQS sử dụng các kỹ thuật như giám sát, làm sạch dữ liệu, so khớp và lập hồ sơ, và vân vân để duy trì chất lượng và tính đúng đắn của dữ liệu.
- **Hỗ trợ dữ liệu lớn** – Microsoft đã tuyên bố hợp tác với Cloudera để sử dụng Hadoop là nền tảng hỗ trợ dữ liệu lớn. Dữ liệu lớn là một tập hợp dữ liệu lớn trong các tập dữ liệu được chia ra để xử lý dễ dàng hơn. Tập hợp này được lưu trữ trong dữ liệu lớn có thể bao gồm thông tin từ các website mạng xã hội, giao thông đường bộ và dữ liệu tín hiệu, và vân vân. Những loại dữ liệu này lớn và phức tạp đòi hỏi các ứng dụng cụ thể như Hadoop để xử lý dữ liệu. SQL Server 2012 phối hợp với Hadoop bây giờ sẽ có thể hỗ trợ dữ liệu lớn.
- **Cài đặt SQL Server** – Trung tâm cài đặt SQL Server hiện nay bao gồm SQL Server Data Tools (SSDT) và máy chủ tập tin Server Message Block (SMB). SSDT cung cấp IDE để xây dựng các giải pháp nghiệp vụ thông minh. Máy chủ tập tin SMB là một lựa chọn lưu trữ được hỗ trợ có thể được sử dụng cho các cơ sở dữ liệu hệ thống và các công cụ cơ sở dữ liệu.
- **Chế độ máy chủ** – Khái niệm chế độ máy chủ đã được thêm vào cho cài đặt Dịch vụ Phân tích. Một thể hiện Dịch vụ Phân tích có ba chế độ máy chủ đó là Multidimensional, Tabular và SharePoint.
- **Tính năng kiểm toán** – Thông số kỹ thuật kiểm toán chuyên dụng có thể được định nghĩa để viết các sự kiện tùy chỉnh trong nhật ký kiểm toán. Các tính năng lọc mới cũng đã được thêm vào trong SQL Server 2012.

- **Chỉ mục XML có lựa chọn** – Đây là một loại chỉ mục XML mới đã được giới thiệu trong SQL Server 2012. Chỉ số mới này có quá trình lập chỉ mục nhanh hơn, khả năng mở rộng cải thiện, và hiệu suất truy vấn nâng cao.
- **Dịch vụ dữ liệu chính** – Tính năng này cung cấp một trung tâm dữ liệu để đảm bảo tính thống nhất và toàn vẹn trên các ứng dụng khác nhau. Trong SQL Server 2012, một ứng dụng bổ sung của Excel đã được tạo ra để hỗ trợ dữ liệu chính khi làm việc với Excel. Ứng dụng bổ sung này làm cho nó dễ dàng chuyển giao và quản lý dữ liệu chính trong Excel. Dữ liệu này có thể dễ dàng được chỉnh sửa trên Excel và nó cũng có thể được xuất bản trở lại cơ sở dữ liệu.
- **PowerView** – Một bộ công cụ nghiệp vụ thông minh mới có tên PowerView đã được giới thiệu trong SQL Server 2012. Bộ công cụ này giúp bạn tạo ra các báo cáo Nghiệp vụ Thông minh cho toàn tổ chức. PowerView là một ứng dụng bổ sung của SQL Server 2012 làm việc trong sự hợp tác với Microsoft SharePoint Server 2010. Ứng dụng bổ sung này giúp trong trình bày và ảo hóa dữ liệu SQL Server 2012 trong một giao diện tương thích trên nền tảng SharePoint. PowerView là một công cụ nghiệp vụ thông minh có thể được sử dụng để làm các bản trình bày của khách hàng bằng cách sử dụng các mô hình, hình ảnh động, trực quan, và vân vân.
- **Tìm kiếm toàn văn bản** – Trong SQL Server 2012, dữ liệu được lưu trữ trong các thuộc tính mở rộng và siêu dữ liệu cũng được tìm kiếm và lập chỉ mục. Tất cả các thuộc tính bổ sung của một tài liệu được tìm kiếm cùng với dữ liệu hiện diện trong tài liệu. Những thuộc tính bổ sung này bao gồm Tên, Loại, Đường dẫn thư mục, Dung lượng, Ngày tháng, và vân vân.

3.7 Kết nối đến các thể hiện SQL Server

SSMS được sử dụng để kết nối với các thể hiện SQL Server. SSMS là một công cụ được dùng để tạo, truy vấn và quản lý cơ sở dữ liệu. Để mở SSMS, kết nối với SQL Server 2012 bằng cách chỉ ra thông tin máy chủ và các thông tin đăng nhập. Các thông tin đăng nhập sẽ bao gồm tên đăng nhập và mật khẩu. Các bước chi tiết để kết nối với thể hiện SQL Server như sau:

1. Bấm vào Start → All Programs → Microsoft SQL Server 2012 → SQL Server Management Studio.
2. Trong hộp thoại Connect to Server, chọn Server type as Database Engine.
3. Gõ **Tên máy chủ**.
4. Chọn **Windows Authentication** hoặc **SQL Server Authentication**, cung cấp **Login** và **Password** yêu cầu, và bấm vào **Connect**.

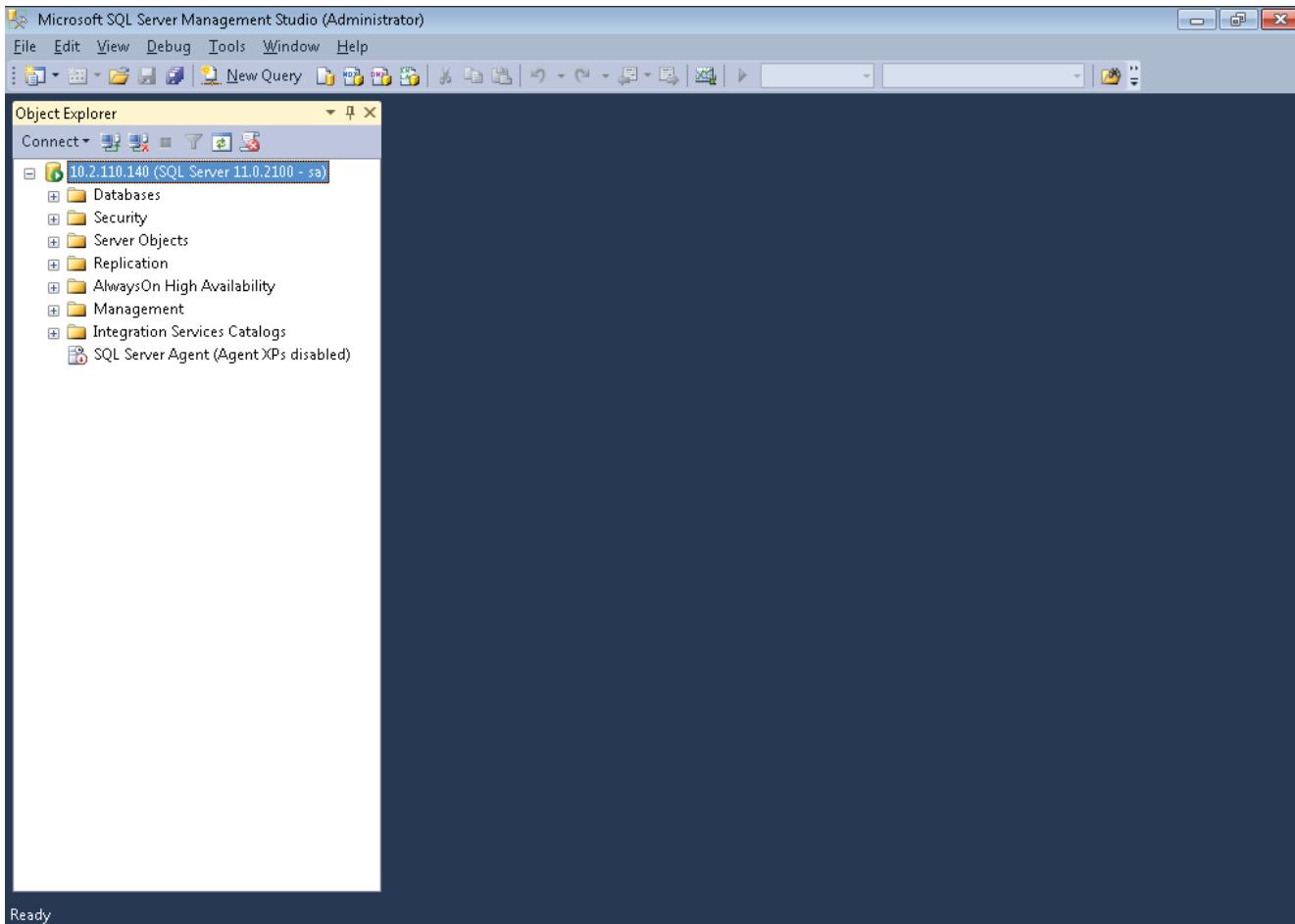
Hình 3.3 trình bày hộp thoại Connect to Server.



Hình 3.3: Hộp thoại Connect to Server

Ghi chú - Hai phương pháp xác thực được cung cấp từ SQL Server 2012 là SQL Server Authentication và Windows Authentication. SQL Server Authentication đòi hỏi một tài khoản người dùng để đăng nhập và mật khẩu. Do đó, nhiều tài khoản người dùng có thể truy cập thông tin sử dụng tên người dùng và mật khẩu tương ứng của họ. Với Windows Authentication, các thông tin đăng nhập hệ điều hành có thể được sử dụng để đăng nhập vào cơ sở dữ liệu SQL Server. Điều này sẽ chỉ làm việc trên máy tính đơn lẻ và không thể được sử dụng trong bất kỳ máy tính nào khác.

Hình 3.4 trình bày cửa sổ SSMS.

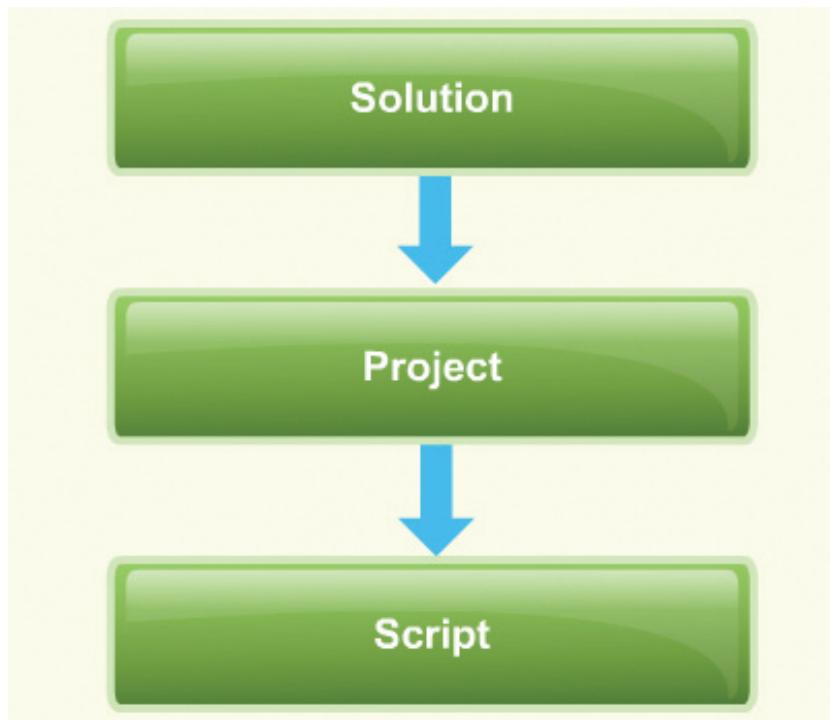


Hình 3.4: Cửa sổ SSMS

3.8 Tạo và tổ chức các tập tin tập lệnh

Tập tin tập lệnh là những tập tin có chứa một tập hợp các lệnh SQL. Tập tin tập lệnh có thể chứa một hoặc nhiều câu lệnh SQL. Các tập tin kịch bản được lưu trữ ở định dạng .sql trong SQL Server 2012.

Các lớp khái niệm trong đó các tập tin kịch bản phải được tổ chức được trình bày trong hình 3.5.



Hình 3.5: Các lớp khái niệm

Giải pháp là một tập tin trong đó tất cả các dự án trong SQL Server 2012 được lưu. Tính năng này hoạt động như một nút trên cùng trong hệ thống phân cấp. Tập tin giải pháp được lưu trữ như một tập tin văn bản với phần mở rộng .ssmssqln. Dự án nằm dưới nút giải pháp. Có thể có nhiều hơn một dự án trong SQL Server 2012. Tất cả dữ liệu liên quan đến siêu dữ liệu kết nối cơ sở dữ liệu và các tập tin linh tinh khác được lưu trữ trong một dự án. Nó được lưu trữ như một tập tin văn bản với phần mở rộng .ssmssqlproj. Các tập tin kịch bản là các tập tin cốt lõi trong đó các truy vấn được phát triển và thực thi. Các kịch bản có phần mở rộng .sql.

3.9 Truy vấn Transact-SQL

Những truy vấn này được nhập vào Transact-SQL và lưu lại dưới dạng tập tin .sql có thể được thực thi trực tiếp trong cửa sổ truy vấn SSMS. Các bước để thực thi các truy vấn Transact-SQL như sau:

1. Trong cửa sổ truy vấn, chọn mã để được thực thi.

2. Trên thanh công cụ **SSMS**, bấm vào **Execute**.

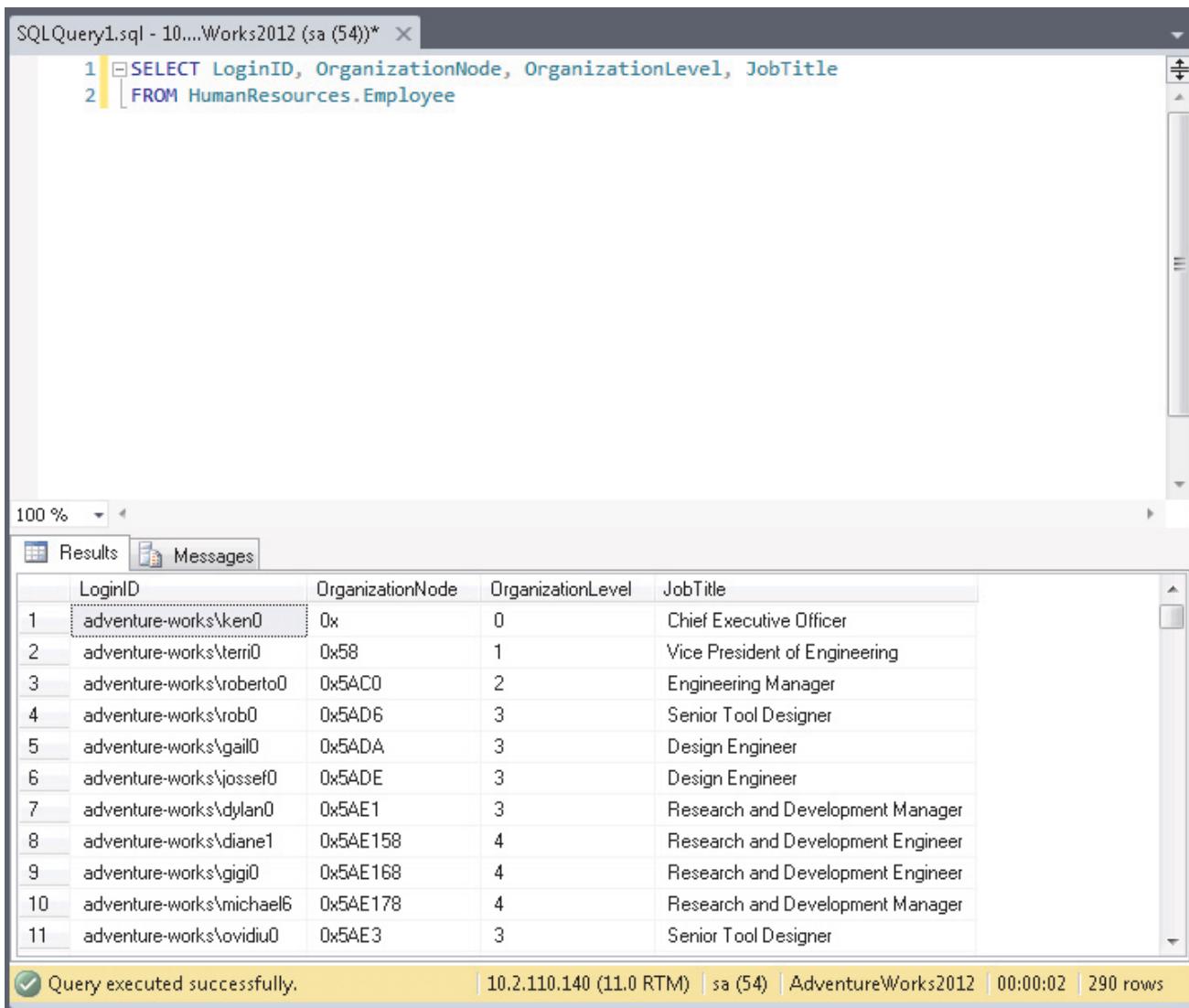
HOẶC

Trên menu **Query**, bấm vào **Execute**.

HOẶC

Nhấn **F5** hoặc **Alt+X** hoặc **Ctrl+E**.

Hình 3.6 trình bày việc thực thi một truy vấn mẫu.



The screenshot shows the SQL Server Management Studio interface. In the top-left corner, there is a tab labeled "SQLQuery1.sql - 10....Works2012 (sa (54))*". Below the tabs, the code window contains the following SQL query:

```
1 | SELECT LoginID, OrganizationNode, OrganizationLevel, JobTitle
2 | FROM HumanResources.Employee
```

Below the code window is a results grid titled "Results". The grid displays the following data:

	LoginID	OrganizationNode	OrganizationLevel	JobTitle
1	adventure-works\ken0	0x	0	Chief Executive Officer
2	adventure-works\terri0	0x58	1	Vice President of Engineering
3	adventure-works\roberto0	0x5AC0	2	Engineering Manager
4	adventure-works\rob0	0x5AD6	3	Senior Tool Designer
5	adventure-works\gail0	0x5ADA	3	Design Engineer
6	adventure-works\jossef0	0x5ADE	3	Design Engineer
7	adventure-works\dylan0	0x5AE1	3	Research and Development Manager
8	adventure-works\diane1	0x5AE158	4	Research and Development Engineer
9	adventure-works\gigi0	0x5AE168	4	Research and Development Engineer
10	adventure-works\michael6	0x5AE178	4	Research and Development Manager
11	adventure-works\ovidiu0	0x5AE3	3	Senior Tool Designer

At the bottom left of the results grid, there is a message: "Query executed successfully." To the right of the message, there is some system information: "10.2.110.140 (11.0 RTM) | sa (54) | AdventureWorks2012 | 00:00:02 | 290 rows".

Hình 3.6: Kết quả truy vấn

Có thể hiển thị các kết quả truy vấn ở ba định dạng khác nhau. Ba định dạng có sẵn là giao diện lưới, văn bản, và tập tin.

Kiểm tra kiến thức của bạn

1. _____ là một công cụ dòng lệnh trong SQL Server 2012.

(A)	SSMS	(C)	SSMSCMD
(B)	SQLCMD	(D)	SQLSSMS

2. Phiên bản đầu tiên của SQL Server đã được phát hành vào năm _____.

(A)	1989	(C)	1992
(B)	1991	(D)	1995

3. Phiên bản nào của SQL Server 2012 hỗ trợ các tính năng như là PowerPivot, PowerView, mô hình ngữ nghĩa Nghiệp vụ Thông minh, Dịch vụ Dữ liệu Chính, và vân vân?

(A)	Enterprise	(C)	Business Intelligence
(B)	Standard	(D)	Express

4. Thành phần nào sau đây có chứa Điểm cuối và Khởi phát?

(A)	Cơ sở dữ liệu	(C)	Đối tượng máy chủ
(B)	Bảo mật	(D)	Sự sao chép

5. Câu nào sau đây về các công cụ trong SQL Server 2012 là đúng?

- a. Công cụ Trung tâm cài đặt SQL Server có thể được sử dụng để thêm, xóa bỏ, và sửa đổi các chương trình SQL Server.
- b. SQLCMD là một IDE được sử dụng cho Các thành phần Nghiệp vụ Thông minh. Nó giúp thiết kế cơ sở dữ liệu sử dụng Visual Studio.
- c. SQL Server Profiler được sử dụng để theo dõi một thể hiện của Công cụ Cơ sở dữ liệu hoặc Dịch vụ Phân tích.
- d. Trung tâm Cài đặt SQL Server là một ứng dụng được cung cấp với SQL Server 2012 giúp bạn phát triển các cơ sở dữ liệu, truy vấn dữ liệu, và quản lý hoạt động tổng thể của SQL Server.

(A)	a và b	(C)	b, c, và d
(B)	a, b, và c	(D)	c và d

3.10.1 Câu trả lời

1.	B
2.	A
3.	C
4.	C
5.	A

Tóm tắt

- Kiến trúc cơ bản của SQL Server 2012 bao gồm các công cụ, dịch vụ, và thể hiện.
- Ba phiên bản của SQL Server là Enterprise, Standard, và Business Intelligence.
- Cấu trúc của SQL Database bao gồm cơ sở dữ liệu, bảo mật, đối tượng máy chủ, nhân bản, Tính sẵn sàng cao AlwaysOn, Quản lý, Danh bạ dịch vụ tích hợp, và vân vân.
- SSMS được sử dụng để kết nối với các thể hiện SQL Server. SSMS là một công cụ được dùng để phát triển, truy vấn và quản lý cơ sở dữ liệu.
- Các tập tin kịch bản sẽ được lưu trữ ở định dạng .sql trong SQL Server 2012.
- Những truy vấn này được nhập vào Transact-SQL và lưu lại dưới dạng tập tin .sql có thể được thực thi trực tiếp vào cửa sổ truy vấn SSMS.

“

**Để tránh chỉ trích, không làm gì cả,
không nói có gì, không là gì cả.**

”

Phần - 4

SQL Azure

Chào mừng bạn đến với phần **SQL Azure**.

Phần này giải thích về SQL Azure và các lợi ích của nó. Nó còn liệt kê các sự khác biệt giữa SQL Azure và SQL Server phía khách hàng. Cuối cùng, phần này giải thích quy trình kết nối SQL Azure với SSMS.

Trong phần này, bạn sẽ học để:

- ➔ Giải thích về SQL Azure
- ➔ Liệt kê các lợi ích của SQL Azure
- ➔ Nêu các sự khác biệt giữa SQL Azure và SQL Server phía khách hàng.
- ➔ Liệt kê các bước để kết nối SQL Azure với SSMS

4.1 Giới thiệu

Điện toán đám mây là một xu hướng công nghệ, có liên quan đến việc cung cấp các phần mềm, nền tảng và cơ sở hạ tầng như các dịch vụ thông qua Internet hoặc mạng. Windows Azure là một đề xuất quan trọng trong bộ sản phẩm và dịch vụ đám mây của Microsoft. Các hàm cơ sở dữ liệu của nền tảng đám mây của Microsoft được Windows Azure SQL Database cung cấp, thường được gọi là SQL Azure.

Có thể sử dụng SQL Azure để lưu trữ và quản lý dữ liệu sử dụng các truy vấn và các hàm khác tương tự như SQL Server 2012. Dữ liệu trên SQL Azure không có sự ràng buộc cụ thể theo địa điểm. Điều này có nghĩa rằng dữ liệu đã lưu trữ trong SQL Azure có thể được xem và chỉnh sửa từ bất kỳ vị trí nào, bởi toàn bộ dữ liệu được lưu trữ trên nền tảng lưu trữ đám mây.

4.2 SQL Azure

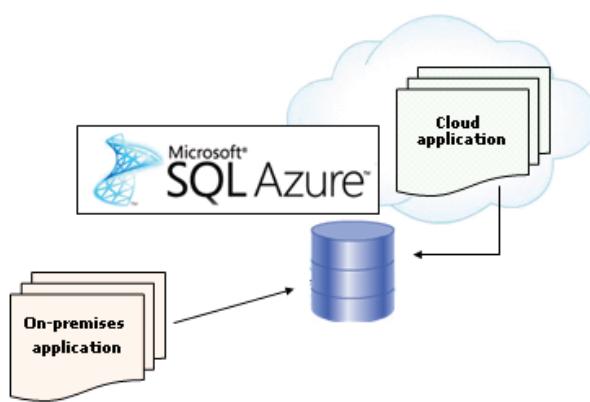
Hãy xem xét kịch bản của phòng Thuế thu nhập. Trong tháng Ba, phòng này tràn ngập với khối lượng công việc nặng nề. Trong phần còn lại của năm, khối lượng công việc có thể ít hơn. Kết quả là, các nguồn tài nguyên, máy chủ, và khả năng tính toán được sử dụng không đủ trong những tháng này và sử dụng quá mức trong những giai đoạn cao điểm. Trong một kịch bản như vậy, sử dụng dịch vụ cơ sở dữ liệu điện toán đám mây như SQL Azure có thể giúp sử dụng tối ưu các nguồn tài nguyên chỉ khi có yêu cầu.

SQL Azure là một dịch vụ cơ sở dữ liệu quan hệ dựa trên đám mây, thúc đẩy các công nghệ SQL Server hiện có. Microsoft SQL Azure mở rộng chức năng của Microsoft SQL Server để phát triển các ứng dụng dựa trên web, có khả năng mở rộng, và được phân phối.

SQL Azure cho phép người dùng thực hiện các truy vấn quan hệ, hoạt động tìm kiếm, và đồng bộ hóa dữ liệu với người dùng di động và các văn phòng hậu bị từ xa. SQL Azure có thể lưu trữ và lấy cả dữ liệu có cấu trúc và phi cấu trúc.

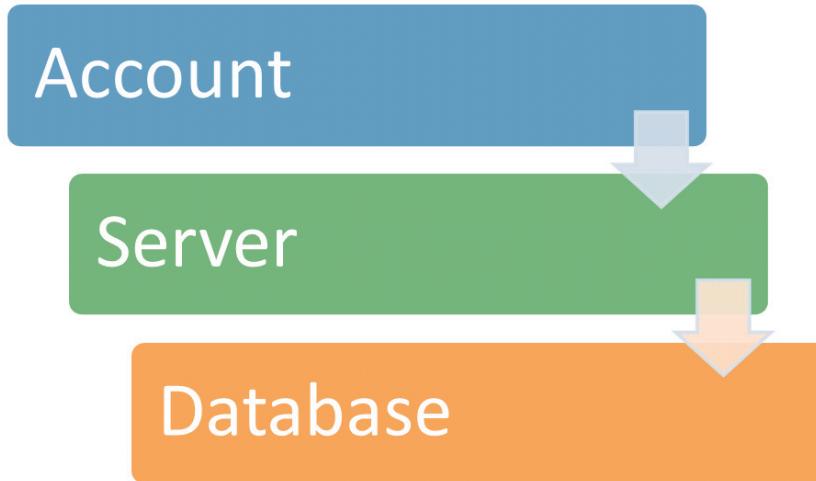
Cả hai ứng dụng dựa trên đám mây cũng như ở phía khách hàng có thể sử dụng cơ sở dữ liệu SQL Azure.

Ứng dụng lấy dữ liệu từ SQL Azure thông qua một giao thức được gọi là Tabular Data Stream (TDS). Giao thức này không phải là mới đối với SQL Azure. Bất cứ khi nào các ứng dụng ở phía khách hàng liên quan đến sự tương tác với SQL Server Database Engine, giao thức này được máy khách và máy chủ sử dụng. Hình 4.1 cho thấy giao diện đơn giản của kiến trúc SQL Azure.



Hình 4.1: Giao diện đơn giản của kiến trúc SQL Azure

Quy trình hoạt động của SQL Azure được giải thích trong mô hình như được trình bày trong hình 4.2.



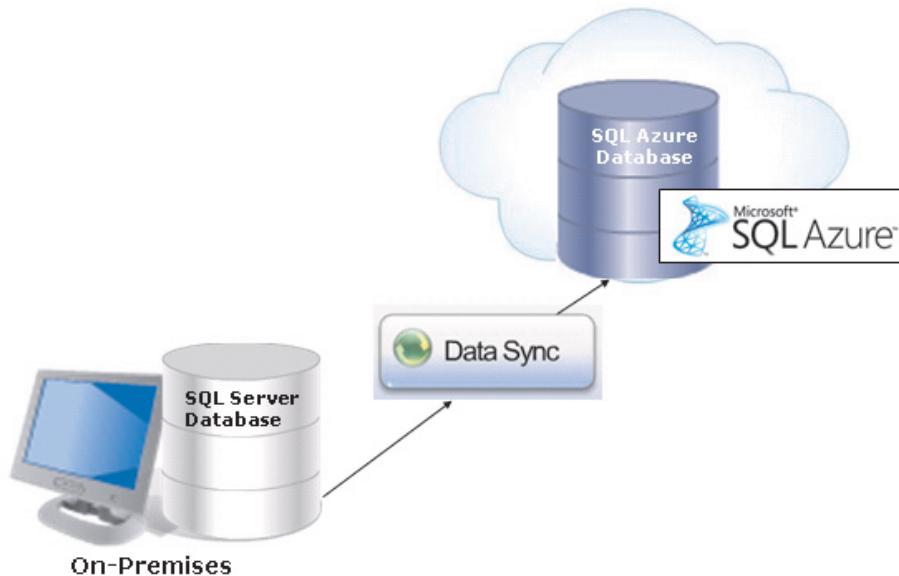
Hình 4.2: Mô hình hoạt động của SQL Azure

Ba đối tượng cốt lõi trong mô hình hoạt động của SQL Azure như sau:

- **Tài khoản** – Đầu tiên phải tạo một tài khoản SQL Azure trước khi thêm các máy chủ sẽ giúp lưu trữ và quản lý dữ liệu. Tài khoản này được tạo ra cho mục đích thanh toán. Thuê bao tài khoản được ghi lại và đo lường và một người được tính tiền theo lượng sử dụng. Để tạo một tài khoản, thông tin xác thực cần phải được cung cấp. Sau khi tài khoản người dùng được tạo ra, các yêu cầu cần phải được cung cấp cho cơ sở dữ liệu SQL Azure. Điều này bao gồm số lượng cơ sở dữ liệu cần thiết, kích thước cơ sở dữ liệu, và vân vân.
- **Máy chủ** – Máy chủ SQL Azure là đối tượng giúp tương tác giữa tài khoản và cơ sở dữ liệu. Sau khi tài khoản được đăng ký, cơ sở dữ liệu được cấu hình sử dụng máy chủ SQL Azure. Các thiết lập khác như thiết lập tường lửa và gán Hệ thống tên miền (DNS) cũng được cấu hình trong máy chủ SQL Azure.
- **Cơ sở dữ liệu** – Cơ sở dữ liệu SQL Azure lưu trữ tất cả dữ liệu theo một cách tương tự như bất kỳ cơ sở dữ liệu SQL Server tại chỗ nào lưu trữ dữ liệu. Mặc dù hiện diện trên đám mây, cơ sở dữ liệu SQL Azure có tất cả các chức năng của một RDBMS bình thường như bảng, dạng xem, truy vấn, hàm, thiết lập bảo mật, và vân vân.

Ngoài những đối tượng cốt lõi, có một đối tượng bổ sung trong SQL Azure. Đối tượng này là công nghệ Đồng bộ dữ liệu SQL Azure. Công nghệ Đồng bộ dữ liệu SQL Azure được xây dựng trên Microsoft Sync Framework và cơ sở dữ liệu SQL Azure.

SQL Azure Data Sync giúp đồng bộ hóa dữ liệu trên SQL Server cục bộ với các dữ liệu trên SQL Azure như được trình bày trong hình 4.3.



Hình 4.3: Data Sync

Data Sync còn có khả năng quản lý dữ liệu giúp chia sẻ dữ liệu dễ dàng giữa các cơ sở dữ liệu SQL khác nhau. Data Sync không chỉ được sử dụng để đồng bộ hóa tại chỗ với SQL Azure, mà còn để đồng bộ hóa một tài khoản SQL Azure với tài khoản khác.

4.3 Các lợi ích của SQL Azure

Những lợi ích của việc sử dụng SQL Azure như sau:

- ➔ **Chi phí thấp hơn** – SQL Azure cung cấp một số hàm tương tự như trên SQL Server tại chỗ với chi phí thấp hơn khi đem so sánh với các trường hợp tại chỗ của SQL Server. Ngoài ra, khi SQL Azure trên nền tảng đám mây, nó có thể được truy cập từ bất kỳ vị trí nào. Do đó, không có thêm chi phí cần thiết để phát triển một cơ sở hạ tầng CNTT chuyên dụng và phòng ban để quản lý cơ sở dữ liệu.
- ➔ **Sử dụng TDS** – TDS được sử dụng trong các cơ sở dữ liệu SQL Server tại chỗ cho các thư viện máy khách. Do đó, hầu hết các nhà phát triển đã quen thuộc với TDS và cách sử dụng tiện ích này. Cùng một loại giao diện TDS được sử dụng trong SQL Azure để xây dựng các thư viện máy khách. Do đó, các nhà phát triển làm việc trên SQL Azure dễ dàng hơn.
- ➔ **Biện pháp chuyển đổi dự phòng tự động** – SQL Azure lưu trữ nhiều bản sao dữ liệu trên các vị trí vật lý khác nhau. Thậm chí khi có lỗi phần cứng do sử dụng nhiều hoặc tải quá mức, SQL Azure giúp duy trì các hoạt động kinh doanh bằng cách cung cấp khả năng sẵn sàng của dữ liệu thông qua các địa điểm vật lý khác. Điều này được thực hiện bằng cách sử dụng các biện pháp chuyển đổi dự phòng tự động được cung cấp trong SQL Azure.

- **Tính linh hoạt trong việc sử dụng dịch vụ** – Ngay cả các tổ chức nhỏ cũng có thể sử dụng SQL Azure bởi mô hình định giá cho SQL Azure được dựa trên khả năng lưu trữ được tổ chức sử dụng. Nếu tổ chức cần lưu trữ nhiều hơn, giá có thể thay đổi cho phù hợp với nhu cầu. Điều này giúp các tổ chức có được sự linh hoạt trong việc đầu tư tùy thuộc vào việc sử dụng dịch vụ.
- **Hỗ trợ Transact-SQL** – Do SQL Azure hoàn toàn dựa trên mô hình cơ sở dữ liệu quan hệ, nó cũng hỗ trợ các hoạt động và truy vấn Transact-SQL. Khái niệm này cũng tương tự như hoạt động của các SQL Server tại chỗ. Do đó, các quản trị viên không cần bất kỳ đào tạo hoặc hỗ trợ bổ sung nào để sử dụng SQL Azure.

4.4 Sự khác biệt giữa SQL Azure và SQL Server tại chỗ.

Sự khác biệt lớn giữa SQL Azure và SQL Server tại chỗ là sự hiện diện của phần cứng và lưu trữ vật lý. Một số khác biệt quan trọng khác giữa SQL Azure và SQL Server phía khách hàng như sau:

- **Các công cụ** – SQL Server phía khách hàng cung cấp một số công cụ để theo dõi và quản lý. Tất cả những công cụ này có thể không được hỗ trợ bởi SQL Azure bởi có một số tập hợp công cụ hạn chế có sẵn trong phiên bản này.
- **Sao lưu** – Sao lưu và phục hồi chức năng phải được hỗ trợ trong SQL Server phía khách hàng để khắc phục thảm họa. Đối với SQL Azure, do tất cả các dữ liệu là trên nền tảng điện toán đám mây, sao lưu và phục hồi là không cần thiết.
- **Câu lệnh USE** – Câu lệnh USE không được SQL Azure hỗ trợ. Do đó, người dùng không thể chuyển đổi giữa các cơ sở dữ liệu trong SQL Azure so với SQL Server phía khách hàng.
- **Xác thực** – SQL Azure chỉ hỗ trợ xác thực SQL Server và SQL Server phía khách hàng hỗ trợ cả xác thực SQL Server và xác thực của Windows.
- **Hỗ trợ Transact-SQL** – Không phải tất cả các chức năng Transact-SQL đều được SQL Azure hỗ trợ.
- **Tài khoản và đăng nhập** – Trong SQL Azure, các tài khoản quản trị được tạo ra trong cổng thông tin quản lý Azure. Do đó, không có thông tin đăng nhập người dùng mức thể hiện cấp riêng biệt.
- **Tường lửa** – Các thiết lập tường lửa cho các cổng và địa chỉ IP cho phép có thể được quản lý trên máy chủ vật lý cho SQL Server phía khách hàng. Bởi cơ sở dữ liệu SQL Azure có mặt trên điện toán đám mây, xác thực thông qua các thông tin đăng nhập là phương pháp duy nhất để xác minh người dùng.

4.5 Kết nối với SQL Azure với SSMS

Để truy cập vào SQL Azure với SSMS, tài khoản Windows Azure phải được tạo ra. Quá trình kết nối SQL Azure với SSMS như sau:

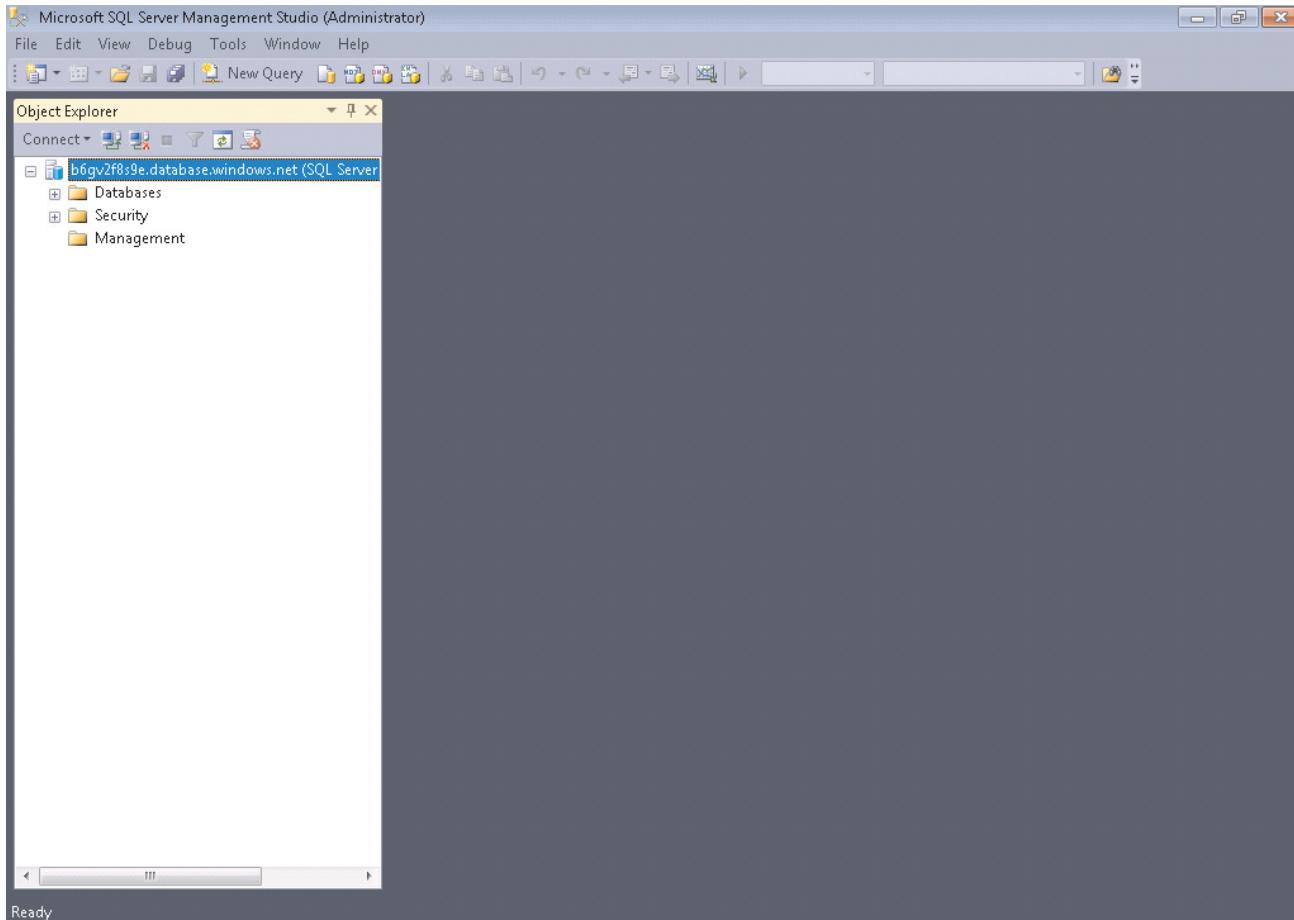
1. Tạo một tài khoản Windows Azure trực tuyến.
2. Mở **Microsoft SQL Server Management Studio**.
3. Trong hộp thoại **Connect to Server**, chỉ ra tên của máy chủ SQL Azure như được trình bày trong hình 4.4. Mỗi tài khoản người dùng của SQL Azure có một tên máy chủ cụ thể.



Hình 4.4: Hộp thoại Connect to Server

4. Trong hộp **Authentication**, chọn **SQL Server Authentication**.
5. Trong hộp **Login**, gõ tên của tài khoản quản trị SQL Azure và mật khẩu.

6. Bấm vào **Connect**. Kết nối với cơ sở dữ liệu được thiết lập thành công như được trình bày trong hình 4.5.



Hình 4.5: Cửa sổ cơ sở dữ liệu sau khi kết nối thành công

Ghi chú - Cơ sở dữ liệu chính là cơ sở dữ liệu mặc định để SQL Server kết nối thông qua SQL Azure. Để kết nối với cơ sở dữ liệu khác, trong hộp **Connect to Server**, bấm vào **Options** để lộ ra tab **Connection Properties** và nhập vào tên của cơ sở dữ liệu mong muốn trong hộp văn bản **Connect to database**. Sau khi một kết nối đến một cơ sở dữ liệu do người dùng định nghĩa được thiết lập, người dùng không thể chuyển sang cơ sở dữ liệu khác mà không ngắt kết nối và kết nối lại với cơ sở dữ liệu tiếp theo. Người dùng có thể chuyển đổi từ cơ sở dữ liệu chính sang cơ sở dữ liệu khác chỉ thông qua SSMS vì câu lệnh **USE** không được hỗ trợ.

4.6 Kiểm tra kiến thức của bạn

1. Giao thức nào sau đây được các ứng dụng sử dụng để lấy dữ liệu từ SQL Server?

(A)	ABS	(C)	TDS
(B)	DTS	(D)	WSQL

2. TDS là viết tắt của _____.

(A)	Dòng dữ liệu dạng bảng	(C)	Dòng phân biệt dạng bảng
(B)	Hệ thống dữ liệu dạng bảng	(D)	Dòng trực tiếp dạng bảng

3. Xác thực nào sau đây là bắt buộc để kết nối với SQL Azure?

(A)	Xác thực của Windows	(C)	Xác thực quản trị hệ thống
(B)	Xác thực SQL Server	(D)	Không có xác thực

4. Điều nào sau đây giúp đồng bộ hóa dữ liệu trên SQL Server cục bộ với các dữ liệu trên SQL Azure?

(A)	Sự xác thực	(C)	Data Sync
(B)	TDS	(D)	Máy chủ

4.6.1 Câu trả lời

1.	C
2.	A
3.	B
4.	C



- ➔ Microsoft SQL Azure là một dịch vụ cơ sở dữ liệu quan hệ dựa trên đám mây, thúc đẩy các công nghệ SQL Server hiện có.
- ➔ SQL Azure cho phép người dùng thực hiện các truy vấn quan hệ, hoạt động tìm kiếm, và đồng bộ hóa dữ liệu với người dùng di động và các văn phòng hậu bị từ xa.
- ➔ SQL Azure có thể lưu trữ và lấy cả dữ liệu có cấu trúc và phi cấu trúc.
- ➔ Ứng dụng lấy dữ liệu từ SQL Azure thông qua một giao thức được gọi là Tabular Data Stream (TDS).
- ➔ Ba đối tượng cốt lõi trong mô hình hoạt động của SQL Azure là tài khoản, máy chủ và cơ sở dữ liệu.
- ➔ SQL Azure Data Sync giúp đồng bộ hóa dữ liệu trên SQL Server cục bộ với các dữ liệu trên SQL Azure.
- ➔ Người dùng có thể kết nối với SQL Azure sử dụng SSMS.



1. Liệt kê ra một số kịch bản tổ chức nơi sử dụng cơ sở dữ liệu SQL Azure sẽ có nhiều thuận thế hơn so với sử dụng cơ sở dữ liệu SQL Server phía khách hàng.

“ Thực hành là

người thầy tốt nhất ”

”

”

Phần - 5

Transact-SQL

Chào mừng bạn đến với phần **Transact-SQL**.

Phần này giải thích Transact-SQL và những loại câu lệnh Transact-SQL khác nhau. Nó còn giải thích các kiểu dữ liệu khác nhau và các phần tử được Transact-SQL hỗ trợ. Cuối cùng, phần này giải thích lý thuyết tập hợp, logic xác nhận, và thứ tự hợp lý của các toán tử trong câu lệnh SELECT.

Trong phần này, bạn sẽ học để:

- ➔ Giải thích về Transact-SQL
- ➔ Danh sách các thể loại câu lệnh Transact-SQL khác nhau
- ➔ Giải thích các kiểu dữ liệu khác nhau được hỗ trợ bởi Transact-SQL
- ➔ Giải thích các phần tử ngôn ngữ Transact-SQL
- ➔ Giải thích về các tập hợp và logic xác nhận
- ➔ Mô tả thứ tự logic của các toán tử trong câu lệnh SELECT

5.1 Giới thiệu

SQL là ngôn ngữ phổ quát được sử dụng trong thế giới cơ sở dữ liệu. Hầu hết các sản phẩm RDBMS hiện đại sử dụng một số loại phương ngữ SQL làm ngôn ngữ truy vấn chính. Có thể sử dụng SQL để tạo ra hoặc hủy bỏ các đối tượng, như là bảng, trên máy chủ cơ sở dữ liệu và làm nhiều việc với các đối tượng này, như là đưa dữ liệu vào trong đó hoặc truy vấn dữ liệu. Transact-SQL là thực hiện SQL tiêu chuẩn của Microsoft. Thường được gọi là T-SQL, ngôn ngữ này thực hiện một cách chuẩn hóa để giao tiếp với cơ sở dữ liệu. Ngôn ngữ Transact-SQL là bản nâng cao của SQL, ngôn ngữ cơ sở dữ liệu quan hệ tiêu chuẩn của Viện Tiêu chuẩn Quốc gia Mỹ (ANSI). Họ cung cấp một ngôn ngữ toàn diện hỗ trợ định nghĩa các bảng định, chèn, xoá, cập nhật và truy cập dữ liệu trong bảng.

5.2 Transact-SQL

Transact-SQL là một ngôn ngữ mạnh mẽ đem lại các tính năng như là các kiểu dữ liệu, các đối tượng tạm thời, và các thủ tục đã lưu trữ mở rộng. Con trỏ cuộn, xử lý có điều kiện, kiểm soát giao tác, và ngoại lệ và xử lý lỗi cũng có một số tính năng được Transact-SQL hỗ trợ.

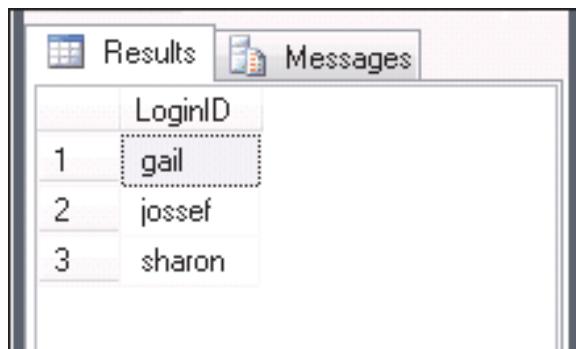
Ngôn ngữ Transact-SQL trong SQL Server 2012 cung cấp hiệu suất nâng cao, chức năng tăng lên và các đặc điểm tăng cường. Các cải tiến bao gồm các hàm vô hướng, phân trang, trình tự, khám phá siêu dữ liệu, và hỗ trợ xử lý lỗi tốt hơn.

Đoạn mã 1 trình bày câu lệnh Transact-SQL, `SELECT`, được sử dụng để lấy tất cả bản ghi của nhân viên với 'Kỹ sư thiết kế' là `JobTitle` từ bảng `Employee`.

Đoạn mã 1:

```
SELECT LoginID
FROM Employee
WHERE JobTitle = 'Kỹ sư thiết kế'
```

Hình 5.1 trình bày kết quả lấy tất cả bản ghi của nhân viên với 'Kỹ sư thiết kế' là `JobTitle` từ bảng `Employee`.



(LoginID)	(JobTitle)
1	gail
2	jossef
3	sharon

Hình 5.1: Kết quả của câu lệnh `SELECT`

Transact-SQL đưa vào nhiều thành tố cú pháp mà được dùng bởi hoặc ảnh hưởng đến hầu hết các câu lệnh. Những phần tử này bao gồm các kiểu dữ liệu, các xác nhận, các hàm, các biến, các biểu thức, điều khiển dòng chảy, chú thích, và dấu tách khối lệnh.

5.3 Các thể loại câu lệnh Transact-SQL khác nhau

SQL Server hỗ trợ ba loại câu lệnh Transact-SQL, cụ thể là: DDL, DML và DCL.

5.3.1 Ngôn ngữ định nghĩa dữ liệu (DDL)

DDL, thường là một phần của DBMS, được sử dụng để định nghĩa và quản lý tất cả các thuộc tính và tính chất của một cơ sở dữ liệu, bao gồm bố trí hàng, định nghĩa cột, cột khóa, địa điểm tập tin, và chiến lược lưu trữ. Các câu lệnh DDL được sử dụng để xây dựng và sửa đổi cấu trúc của bảng và các đối tượng khác như dạng xem, khởi phát, thủ tục đã lưu trữ, và vân vân. Đối với từng đối tượng, thường có các câu lệnh CREATE, ALTER, và DROP (như là CREATE TABLE, ALTER TABLE, và DROP TABLE). Hầu hết các câu lệnh DDL lấy dạng sau đây:

- ➔ CREATE object _ name
- ➔ ALTER object _ name
- ➔ DROP object _ name

Trong các câu lệnh DDL, object_name có thể là bảng, dạng xem, khởi phát, thủ tục đã lưu trữ, và vân vân.

5.3.2 Ngôn ngữ thao tác dữ liệu (DML)

DML được sử dụng để chọn, chèn, cập nhật, hoặc xóa dữ liệu trong các đối tượng được định nghĩa với DDL. Tất cả người dùng cơ sở dữ liệu có thể sử dụng những câu lệnh này trong các phép tính thông thường trên một cơ sở dữ liệu. Các câu lệnh DML khác nhau như sau:

- ➔ SELECT câu lệnh
- ➔ INSERT câu lệnh
- ➔ UPDATE câu lệnh
- ➔ DELETE câu lệnh

5.3.3 Ngôn ngữ kiểm soát dữ liệu (DCL)

Dữ liệu là một phần quan trọng của cơ sở dữ liệu, do vậy nên có những bước thích hợp để kiểm tra xem không có người dùng không hợp lệ nào truy cập vào dữ liệu. Ngôn ngữ kiểm soát dữ liệu được dùng để kiểm soát sự cho phép đối với các đối tượng cơ sở dữ liệu. Quyền được kiểm soát bằng cách sử dụng câu lệnh GRANT, REVOKE, và DENY. Các câu lệnh DCL cũng được sử dụng để bảo mật cơ sở dữ liệu. Ba câu lệnh DCL cơ bản như sau:

- ➔ GRANT câu lệnh
- ➔ REVOKE câu lệnh
- ➔ DENY câu lệnh

5.4 Các kiểu dữ liệu

Kiểu dữ liệu là một thuộc tính định nghĩa kiểu dữ liệu mà một đối tượng có thể có. Kiểu dữ liệu phải được cung cấp cho các cột, tham số, biến, và hàm trả về các giá trị dữ liệu, và các thủ tục lưu trữ có một mã trả lại. Transact-SQL đưa vào một số kiểu dữ liệu cơ bản, như là `varchar`, `text`, và `int`. Tất cả các dữ liệu được lưu trữ trong SQL Server phải tương thích với một trong các kiểu dữ liệu cơ sở.

Các đối tượng sau đây có kiểu dữ liệu:

- ➔ Cột có trong bảng và dạng xem
- ➔ Các tham số trong các thủ tục được lưu
- ➔ Các biến số
- ➔ Các hàm Transact-SQL mà trả lại một hoặc nhiều giá trị dữ liệu thuộc một kiểu dữ liệu cụ thể
- ➔ Thủ tục lưu trữ có một mã trả lại thuộc kiểu dữ liệu số nguyên

Các mục khác nhau trong SQL Server 2012 như cột, các biến, và biểu thức được gán các kiểu dữ liệu. SQL Server 2012 hỗ trợ ba kiểu dữ liệu:

➔ **Kiểu dữ liệu do hệ thống định nghĩa**

Những kiểu dữ liệu được cung cấp bởi SQL Server 2012. Bảng 5.1 trình bày các kiểu dữ liệu do hệ thống định nghĩa thường được sử dụng của SQL Server 2012.

Loại	Kiểu dữ liệu	Mô tả
Các số chính xác	int	Cột của loại này chiếm giữ 4 byte không gian bộ nhớ. Thường được dùng để giữ các giá trị số nguyên. Có thể giữ dữ liệu số nguyên từ -2^31 (-2.147.483.648) đến 2^31-1 (2.147.483.647).
	smallint	Cột của loại này chiếm giữ 2 byte không gian bộ nhớ. Có thể giữ dữ liệu số nguyên từ -32.768 đến 32.767.
	tinyint	Cột thuộc loại này chiếm giữ 1 byte không gian bộ nhớ. Có thể giữ dữ liệu số nguyên từ 0 đến 255.
	bigint	Cột của loại này chiếm giữ 8 byte không gian bộ nhớ. Có thể chứa dữ liệu trong khoảng -2^63 (-9.223.372.036.854.775.808) đến 2^63-1 (9.223.372.036.854.775.807).
	numeric	Cột thuộc loại này có độ chính xác và tỉ lệ cố định.
	money	Cột của loại này chiếm giữ 8 byte không gian bộ nhớ. Trình bày các giá trị dữ liệu tiền tệ trong khoảng từ -2^63/10000 (-922.337.203.685.477,5808) đến 2^63-1 (922.337.203.685.477,5807).
Các số gần đúng	float	Cột của loại này chiếm giữ 8 byte không gian bộ nhớ. Biểu diễn số có dấu thập phân thay đổi dao động từ -1.79E +308 đến 1.79E+308.
	real	Cột của loại này chiếm giữ 4 byte không gian bộ nhớ. Biểu diễn số có độ chính xác thay đổi dao động từ -3.40E+38 đến 3.40E+38.
Ngày và giờ	datetime	Biểu diễn ngày và giờ. Được lưu như hai số nguyên 4 byte.
	smalldatetime	Biểu diễn ngày và giờ.
Chuỗi ký tự	char	Lưu trữ dữ liệu ký tự có chiều dài cố định và không phải là Unicode.
	varchar	Lưu trữ dữ liệu ký tự có chiều dài thay đổi và không phải là Unicode với tối đa 8.000 ký tự.
	văn bản	Lưu trữ dữ liệu ký tự có chiều dài thay đổi và không phải là Unicode với chiều dài tối đa 2^31 - 1 (2.147.483.647) ký tự.
Loại Unicode	nchar	Lưu dữ liệu ký tự Unicode với độ dài cố định.
	nvarchar	Lưu dữ liệu ký tự Unicode có độ dài biến đổi.

Loại	Kiểu dữ liệu	Mô tả
Các kiểu dữ liệu khác	timestamp	Cột của loại này chiếm giữ 8 byte không gian bộ nhớ. Có thể giữ tự động các số nhị phân duy nhất, được tạo ra mà được tạo ra cho một cơ sở dữ liệu.
	binary(n)	Lưu trữ dữ liệu nhị phân có chiều dài cố định với chiều dài tối đa là 8000 byte.
	varbinary(n)	Lưu dữ liệu nhị phân có độ dài biến đổi với độ dài tối đa 8000 byte.
	image	Lưu dữ liệu nhị phân với độ dài biến đổi với độ dài tối đa là $2^{30}-1$ (1.073.741.823) byte.
	uniqueidentifier	Cột của loại này chiếm giữ 16 byte không gian bộ nhớ. Ngoài ra, lưu trữ mã định danh duy nhất toàn cầu (GUID).

Bảng 5.1: Các kiểu dữ liệu do hệ thống định nghĩa

→ Kiểu dữ liệu alias

Những cái này dựa trên các kiểu dữ liệu do hệ thống cung cấp. Các kiểu dữ liệu bí danh được sử dụng khi nhiều hơn một bảng lưu trữ cùng một kiểu dữ liệu trong một cột và có đặc điểm tương tự như là chiều dài, khả năng null, và loại. Trong các trường hợp như vậy, có thể tạo ra kiểu dữ liệu bí danh để có thể được dùng chung cho các bảng này.

Các kiểu dữ liệu bí danh có thể được tạo ra bằng cách sử dụng câu lệnh CREATE TYPE. Cú pháp cho câu lệnh CREATE TYPE như sau:

Cú pháp:

```
CREATE TYPE [ schema_name . ] type_name { FROM base_type [ ( precision [ , scale ]
) ] [ NULL | NOT NULL ] } [ ; ]
```

trong đó:

schema_name: xác định tên của sơ đồ trong đó kiểu dữ liệu bí danh đang được tạo ra. Sơ đồ là một tập hợp các đối tượng như bảng, dạng xem, và vân vân trong một cơ sở dữ liệu.

type_name: xác định tên của loại bí danh đang được tạo ra.

base_type: xác định tên của kiểu dữ liệu do hệ thống định nghĩa dựa trên đó kiểu dữ liệu bí danh đang được tạo ra.

precision và scale: xác định độ chính xác và tỉ lệ cho dữ liệu số.

NULL | NOT NULL: xác định xem kiểu dữ liệu có thể chứa giá trị null hay không.

Đoạn mã 2 trình bày cách để tạo ra một kiểu dữ liệu bí danh sử dụng câu lệnh CREATE TYPE.

Đoạn mã 2:

```
CREATE TYPE usertype FROM varchar(20) NOT NULL
```

Trong đoạn mã, kiểu dữ liệu dựng sẵn `varchar` được lưu giữ như một kiểu dữ liệu mới có tên là `usertype` bằng cách sử dụng câu lệnh `CREATE TYPE`.

→ Các loại do người dùng định nghĩa

Những loại này được tạo ra bằng cách sử dụng các ngôn ngữ lập trình được .NET Framework hỗ trợ.

5.5 Các phần tử ngôn ngữ Transact-SQL

Những phần tử ngôn ngữ Transact-SQL được sử dụng trong SQL Server 2012 để làm việc trên các dữ liệu được nhập vào cơ sở dữ liệu SQL Server. Những phần tử ngôn ngữ Transact-SQL bao gồm các xác nhận, toán tử, hàm, biến, biểu thức, điều khiển dòng chảy, lỗi, và giao tác, chú thích, và dấu tách khối lệnh.

5.5.1 Các xác nhận và toán tử

Các xác nhận được sử dụng để đánh giá liệu một biểu thức là `TRUE`, `FALSE`, hay `UNKNOWN`. Một số xác nhận có sẵn trong Transact-SQL như sau:

- `IN` - Xác định xem một giá trị đã chỉ định so khớp với bất kỳ giá trị trong một truy vấn con hoặc một danh sách.
- `BETWEEN` - Chỉ ra một loạt các giá trị để kiểm tra.
- `LIKE` - Được sử dụng để so khớp các ký tự so với một mô hình cụ thể.
- `CONTAINS` - Tìm kiếm các trùng khớp chính xác hoặc chưa chính xác đối với những từ đơn và cụm từ đơn lẻ, các từ trong một khoảng cách nhất định với nhau, hoặc các trùng khớp có trọng số.

Bảng 5.2 trình bày một số ví dụ về xác nhận.

Xác nhận	Ví dụ
<code>IN</code>	<code>SELECT UserID, FirstName, LastName, Salary FROM Employee WHERE Salary IN(5000,20000);</code>
<code>BETWEEN</code>	<code>Select UserID, FirstName, LastName, Salary FROM Employee WHERE Salary BETWEEN 5000 and 20000;</code>
<code>LIKE</code>	<code>Select UserID, FirstName, LastName, Salary FROM Employee WHERE FirstName LIKE '%h%'</code>
<code>CONTAINS</code>	<code>SELECT UserID, FirstName, LastName, Salary FROM Employee WHERE Salary CONTAINS(5000);</code>

Bảng 5.2: Ví dụ về xác nhận

Các toán tử được dùng để thực hiện tính toán số học, so sánh, ghép hoặc gán các giá trị. Ví dụ, dữ liệu có thể được kiểm tra để xác minh rằng cột `COUNTRY` cho dữ liệu khách hàng được tập kết (hoặc có giá trị `NOT NULL`). Trong các truy vấn, bất cứ ai có thể xem dữ liệu trong bảng yêu cầu toán tử cũng có thể thực hiện các phép tính. Yêu cầu có sự cho phép thích hợp trước khi có thể thay đổi thành công dữ liệu. SQL Server có bảy loại toán tử. Bảng 5.3 mô tả các toán tử khác nhau được hỗ trợ trong SQL Server 2012.

Toán tử	Mô tả	Ví dụ
So sánh	So sánh một giá trị đối với giá trị khác hoặc một biểu thức.	=, <, >, >=, <=, !=, !=
Logic	Thử về sự đúng sai của một điều kiện	AND, OR, NOT
Số học	Thực hiện các phép tính số học như cộng, trừ, nhân và chia	+ , - , * , / , %
Sự ghép nối	Kết hợp hai chuỗi thành một chuỗi	+
Gán	Gán một giá trị cho một biến số	=

Bảng 5.3: Các toán tử

Bảng 5.4 trình bày thứ tự ưu tiên của các xác nhận và toán tử.

Thứ tự	Toán tử
1	() dấu ngoặc
2	* , / , %
3	+ , -
4	=, <, >, >=, <=, !=, !=
5	NOT
6	AND
7	BETWEEN, IN, CONTAINS, LIKE, OR
8	=

Bảng 5.4: Thứ tự ưu tiên các xác nhận và toán tử

Đoạn mã 3 trình bày thực thi các toán tử theo thứ tự ưu tiên.

Đoạn mã 3:

```
DECLARE @Number int;
SET @Number = 2 + 2 * (4 + (5 - 3))
SELECT @Number
```

Ở đây, các bước để đi đến kết quả như sau:

1. $2 + 2 * (4 + (5 - 3))$
2. $2 + 2 * (4 + 2)$
3. $2 + 2 * 6$
4. $2 + 12$
5. 14

Do đó, đoạn mã sẽ hiển thị 14.

5.5.2 Các hàm

Hàm là tập hợp các câu lệnh Transact-SQL được dùng để thực hiện tác vụ nào đó. Transact-SQL bao gồm một số lượng lớn các hàm. Các hàm này có thể hữu dụng khi dữ liệu được tính toán hoặc thao tác. Trong SQL, các hàm làm việc với dữ liệu, hoặc nhóm dữ liệu, để trả lại một giá trị yêu cầu. Chúng có thể được sử dụng trong danh sách SELECT, hoặc bất cứ nơi nào trong một biểu thức. Bốn loại hàm trong SQL Server 2012 như sau:

- **Các hàm tập hàng** - Trong Transact-SQL, hàm tập hàng được sử dụng để trả về một đối tượng có thể được sử dụng thay cho tham khảo bảng. Ví dụ, OPENDATASOURCE, OPENQUERY, OPENROWSET, and OPENXML là các hàm tập hàng.
- **Các hàm tổng hợp** - Transact-SQL cung cấp các hàm tổng hợp để hỗ trợ việc tổng kết số lượng lớn dữ liệu. Ví dụ, SUM, MIN, MAX, AVG, COUNT, COUNTBIG, và vân vân là các hàm tổng hợp.
- **Các hàm xếp hạng** - Nhiều nhiệm vụ, chẳng hạn như tạo ra các mảng, tạo ra số thứ tự, tìm kiếm các hạng, và vân vân có thể được thực hiện một cách dễ dàng hơn và nhanh hơn bằng cách sử dụng các hàm xếp hạng. Ví dụ, RANK, DENSE_RANK, NTILE, và ROW_NUMBER là các hàm xếp hạng.
- **Các hàm vô hướng** - Trong các hàm vô hướng, đầu vào là một giá trị đơn và đầu ra nhận được cũng là một giá trị đơn.

Bảng 5.5 trình bày các hàm vô hướng thường được sử dụng trong SQL.

Tên hàm	Mô tả	Ví dụ
Hàm chuyển đổi	Hàm chuyển đổi được sử dụng để chuyển đổi giá trị của một kiểu dữ liệu sang loại khác. Ngoài ra, nó có thể được dùng để có được nhiều các định dạng đặc biệt của ngày.	CONVERT
Hàm ngày và thời gian	Hàm ngày và thời gian được sử dụng để thao tác các giá trị ngày và thời gian. Chúng rất hữu ích để thực hiện các tính toán dựa trên thời gian và ngày tháng.	GETDATE, SYSDATETIME, GETUTCDATE, DATEADD, DATEDIFF, YEAR, MONTH, DAY
Hàm toán học	Các hàm toán học thực hiện các phép tính đại số đối với các giá trị số.	RAND, ROUND, POWER, ABS, CEILING, FLOOR
Hàm hệ thống	SQL Server cung cấp các hàm hệ thống để trả về các thiết lập siêu dữ liệu hoặc cấu hình.	HOST_ID, HOST_NAME, ISNULL
Hàm chuỗi	Hàm chuỗi được sử dụng cho đầu vào chuỗi như char và varchar. Đầu ra có thể là một chuỗi hoặc một giá trị số.	SUBSTRING, LEFT, RIGHT, LEN, DATALENGTH, REPLACE, REPLICATE, UPPER, LOWER, RTRIM, LTRIM

Bảng 5.5: Các hàm vô hướng

Ngoài ra còn có các hàm vô hướng khác như hàm con trỏ, hàm logic, hàm siêu dữ liệu, hàm bảo mật, và vân vân có sẵn trong SQL Server 2012.

5.5.3 Các biến

Biến là một đối tượng mà có thể giữ một giá trị dữ liệu. Trong Transact-SQL, các biến có thể được phân thành các biến cục bộ và toàn cầu.

Trong Transact-SQL, các biến số nội bộ được tạo ra và sử dụng để lưu trữ tạm thời trong khi các câu lệnh SQL được thực thi. Có thể truyền dữ liệu cho các câu lệnh SQL bằng cách sử dụng các biến số nội bộ. Tên của biến cục bộ phải được bắt đầu bằng dấu '@'.

Các biến số toàn cục là các biến số dựng sẵn được hệ thống định nghĩa và duy trì. Biến toàn cầu trong SQL Server được bắt đầu với hai dấu '@'. Giá trị của bất kỳ biến nào cũng có thể được lấy ra với một truy vấn SELECT đơn giản.

5.5.4 Các biểu thức

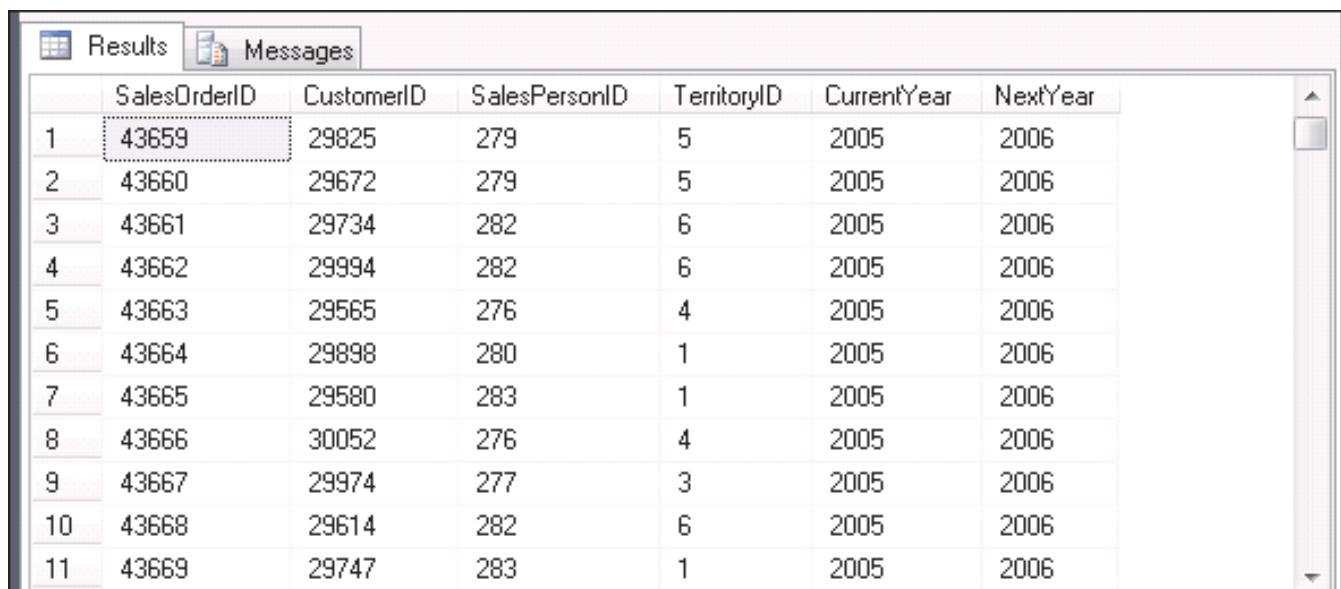
Biểu thức là một tổ hợp các ký hiệu nhận dạng, giá trị và toán tử mà SQL Server có thể đánh giá nhằm có được kết quả. Có thể sử dụng biểu thức ở một số nơi khác nhau khi truy cập hay thay đổi dữ liệu.

Đoạn mã 4 trình bày một biểu thức hoạt động trên một cột để thêm một số nguyên vào kết quả của hàm YEAR trên cột datetime.

Đoạn mã 4:

```
SELECT SalesOrderID, CustomerID, SalesPersonID, TerritoryID, YEAR(OrderDate)
AS CurrentYear, YEAR(OrderDate) + 1 AS NextYear
FROM Sales.SalesOrderHeader
```

Hình 5.2 trình bày các kết quả của biểu thức.



	SalesOrderID	CustomerID	SalesPersonID	TerritoryID	CurrentYear	NextYear
1	43659	29825	279	5	2005	2006
2	43660	29672	279	5	2005	2006
3	43661	29734	282	6	2005	2006
4	43662	29994	282	6	2005	2006
5	43663	29565	276	4	2005	2006
6	43664	29898	280	1	2005	2006
7	43665	29580	283	1	2005	2006
8	43666	30052	276	4	2005	2006
9	43667	29974	277	3	2005	2006
10	43668	29614	282	6	2005	2006
11	43669	29747	283	1	2005	2006

Hình 5.2: Kết quả của biểu thức

5.5.5 Kiểm soát dòng chảy, lỗi, và giao tác

Mặc dù Transact-SQL chủ yếu là một ngôn ngữ truy vấn dữ liệu, nó hỗ trợ các câu lệnh kiểm soát dòng chảy để thực thi và tìm lỗi. Ngôn ngữ kiểm soát dòng chảy xác định lưu lượng thực thi các câu lệnh Transact-SQL, khối câu lệnh, hàm do người dùng định nghĩa, và các thủ tục lưu trữ.

Bảng 5.6 trình bày một số câu lệnh kiểm soát dòng chảy thường được sử dụng trong Transact-SQL.

Câu lệnh kiểm soát dòng chảy	Mô tả
IF . . . ELSE	Cung cấp phân nhánh kiểm soát dựa trên một thử nghiệm hợp lý.
WHILE	Lặp lại một câu lệnh hoặc một khối lệnh miễn là điều kiện vẫn còn đúng.
BEGIN. . . END	Định nghĩa phạm vi của khối lệnh Transact-SQL.
TRY. . . CATCH	Định nghĩa cấu trúc cho ngoại lệ và xử lý lỗi.
BEGIN TRANSACTION	Đánh dấu một khối lệnh như là một phần của giao tác rõ ràng.

Bảng 5.6: Các câu lệnh kiểm soát dòng chảy

5.5 Chú thích

Chú thích là chuỗi văn bản mô tả, còn được gọi là nhận xét, trong mã chương trình sẽ được trình biên dịch bỏ qua. Chú thích có thể được đưa vào trong mã nguồn của một câu lệnh đơn lẻ, một lô, hoặc một thủ tục lưu trữ. Chú thích giải thích mục đích của chương trình, điều kiện thực thi đặc biệt, và cung cấp thông tin lịch sử sửa đổi. Microsoft SQL Server hỗ trợ hai loại kiểu dạng nhận xét:

→ - - (hai dấu nối)

Có thể đánh dấu dòng mã đầy đủ hoặc một phần của mã thành một nhận xét, nếu hai dấu nối (- -) được đặt ở đoạn bắt đầu. Phần còn lại của dòng trở thành nhận xét.

Đoạn mã 5 hiển thị việc sử dụng kiểu chú thích này.

Đoạn mã 5:

```
USE AdventureWorks2012
-- Bảng HumanResources.Employee chứa các chi tiết về một nhân viên.
--Câu lệnh này lấy tất cả các hàng của bảng
-- HumanResources.Employee.

SELECT * FROM HumanResources.Employee
```

→ /* ... */ (cặp ký tự dấu gạch théo-dấu sao)

Có thể sử dụng các ký tự nhận xét này trên cùng hàng như mã được thực thi, trên các dòng của chính chúng, hoặc thậm chí trong mã có thể thực thi. Mọi thứ trong dòng bắt đầu từ cặp chú thích mở /*) đến cặp chú thích đóng /*) được coi là một phần của chú thích. Đối với chú thích nhiều dòng, cặp ký tự chú thích mở mở comment /*) phải bắt đầu chú thích, và cặp ký tự nhận xét đóng /*) phải kết thúc chú thích.

Đoạn mã 6 hiển thị việc sử dụng kiểu chú thích này.

Đoạn mã 6:

```
USE AdventureWorks2012
/*
Bảng HumanResources.Employee chứa các chi tiết về một nhân viên.

Câu lệnh này lấy tất cả các hàng của bảng
HumanResources.Employee.*/
SELECT * FROM HumanResources.Employee
```

5.5.7 Dấu tách khối lệnh

Khối lệnh là tập hợp một hoặc nhiều câu lệnh Transact-SQL được gửi một lúc từ một ứng dụng vào SQL Server để thực thi. Các câu lệnh Transact-SQL trong một khối lệnh được biên dịch thành một đơn vị thực thi đơn lẻ, được gọi là kế hoạch thực thi. Các câu lệnh trong kế hoạch thực thi sau đó được thực thi từng cái một. Quá trình trong đó tập hợp các lệnh được xử lý từng cái một từ một khối lệnh được gọi là việc xử lý khối lệnh.

Dấu tách khối lệnh được xử lý bằng các công cụ máy khách SQL Server như SSMS để thực thi các lệnh. Ví dụ, bạn cần phải chỉ ra GO như một dấu tách khối lệnh trong SSMS.

Một ví dụ về câu lệnh khối lệnh được đưa ra trong Đoạn mã 7.

Đoạn mã 7:

```
USE AdventureWorks2012
SELECT * FROM HumanResources.Employee
GO
```

Trong Đoạn mã 7, hai câu lệnh sẽ được nhóm lại thành một kế hoạch thực thi, nhưng được thực hiện mỗi lần một câu lệnh. Từ khóa GO báo hiệu sự kết thúc của một khối lệnh.

5.6 Các tập hợp và logic xác nhận

Các tập hợp và logic xác nhận là hai nguyên tắc cơ bản toán học được sử dụng trong SQL Server 2012. Cả hai lý thuyết này được sử dụng trong truy vấn dữ liệu trong SQL Server 2012.

5.6.1 Lý thuyết tập hợp

Lý thuyết tập hợp là nền tảng toán học được sử dụng trong mô hình cơ sở dữ liệu quan hệ. Tập hợp là một bộ sưu tập các đối tượng riêng biệt được coi như là toàn bộ. Ví dụ, tất cả các nhân viên dưới bảng Employee có thể được coi là một tập hợp. Nhân viên là những đối tượng khác nhau tạo thành một phần của tập hợp trong bảng Employee.

Bảng 5.7 cho thấy các ứng dụng khác nhau trong lý thuyết tập hợp và ứng dụng tương ứng của chúng trong các truy vấn SQL Server.

Các ứng dụng lý thuyết tập hợp	Ứng dụng trong các truy vấn SQL Server
Hành động trên toàn bộ tập hợp cùng một lúc.	Truy vấn toàn bộ bảng cùng một lúc.
Sử dụng khai báo, xử lý dựa trên tập hợp.	Sử dụng các thuộc tính trong SQL Server để lấy dữ liệu cụ thể.
Các phần tử trong tập hợp phải là duy nhất.	Định nghĩa các khóa duy nhất trong bảng.
Không có hướng dẫn phân loại.	Kết quả truy vấn không được lấy theo bất kỳ thứ tự nào.

Bảng 5.7: Ứng dụng trong Lý thuyết tập hợp với các truy vấn SQL Server

5.6.2 Logic xác nhận

Logic xác nhận là một khuôn khổ toán học bao gồm các bài kiểm tra logic cung cấp kết quả. Kết quả luôn được hiển thị như đúng hoặc sai. Trong Transact-SQL, các biểu thức như biểu thức WHERE và CASE được dựa trên logic xác nhận. Logic xác nhận cũng được sử dụng trong các tình huống khác trong Transact-SQL. Một số ứng dụng logic xác nhận trong Transact-SQL như sau:

- Thực thi toàn vẹn dữ liệu sử dụng ràng buộc CHECK
- Kiểm soát dòng chảy sử dụng câu lệnh IF
- Nối các bảng sử dụng bộ lọc ON
- Dữ liệu lọc trong các truy vấn sử dụng mệnh đề WHERE và HAVING
- Cung cấp logic có điều kiện cho các biểu thức CASE
- Định nghĩa các truy vấn con

5.7 Thứ tự logic của các toán tử trong câu lệnh SELECT

Cùng với cú pháp của các phần tử SQL Server khác nhau, người dùng SQL Server cũng phải biết quy trình toàn bộ truy vấn được thực thi như thế nào. Quy trình này là một quá trình logic chia nhỏ truy vấn và thực hiện truy vấn đó theo một trình tự được định nghĩa trước trong SQL Server 2012. Câu lệnh SELECT là một truy vấn sẽ được sử dụng để giải thích quy trình logic về thực hiện truy vấn. Sau đây là cú pháp của câu lệnh SELECT.

Cú pháp:

```
SELECT <chọn danh sách>
FROM <nguồn bảng>
WHERE <điều kiện tìm kiếm>
GROUP BY <nhóm theo danh sách>
HAVING <điều kiện tìm kiếm>
ORDER BY <thứ tự theo danh sách>
```

Bảng 5.8 giải thích những phần tử của câu lệnh SELECT.

Phần tử	Mô tả
SELECT <chọn danh sách>	Định nghĩa các cột được trả về
FROM <nguồn bảng>	Định nghĩa bảng được truy vấn
WHERE <điều kiện tìm kiếm>	Lọc các hàng bằng cách sử dụng các xác nhận
GROUP BY <nhóm theo danh sách>	Sắp xếp các hàng theo nhóm
HAVING <điều kiện tìm kiếm>	Lọc các nhóm sử dụng các xác nhận
ORDER BY <thứ tự theo danh sách>	Sắp xếp đầu ra

Bảng 5.8: Các phần tử của câu lệnh SELECT

Đoạn mã 8 trình bày câu lệnh SELECT.

Đoạn mã 8:

```
SELECT SalesPersonID, YEAR(OrderDate) AS OrderYear
FROM Sales.SalesOrderHeader
WHERE CustomerID = 30084
GROUP BY SalesPersonID, YEAR(OrderDate)
HAVING COUNT(*) > 1
ORDER BY SalesPersonID, OrderYear;
```

Trong ví dụ này, thứ tự theo đó SQL Server sẽ thực hiện câu lệnh SELECT như sau:

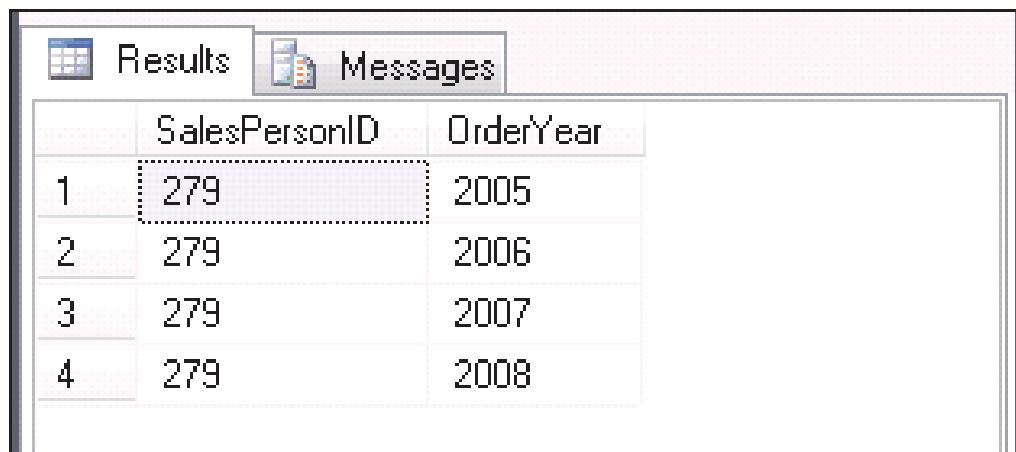
- Đầu tiên, mệnh đề FROM được đánh giá để định nghĩa bảng nguồn sẽ được truy vấn.
- Tiếp theo, mệnh đề WHERE được đánh giá để lọc các hàng trong bảng nguồn. Việc lọc này được định nghĩa bằng xác nhận đã đề cập trong mệnh đề WHERE.
- Sau đó, mệnh đề GROUP BY được đánh giá. Mệnh đề này sắp xếp các giá trị đã lọc nhận được từ mệnh đề WHERE.
- Tiếp theo, mệnh đề HAVING được đánh giá dựa trên xác nhận được cung cấp.
- Tiếp theo, mệnh đề SELECT được thực hiện để xác định các cột sẽ xuất hiện trong kết quả truy vấn.

6. Cuối cùng, câu lệnh ORDER BY được đánh giá để hiển thị đầu ra.

Trình tự thực hiện cho câu lệnh SELECT trong đoạn mã 8 sẽ như sau:

```
5. SELECT SalesPersonID, YEAR(OrderDate) AS OrderYear
1. FROM SalesOrderHeader
2. WHERE CustomerID = 30084
3. GROUP BY SalesPersonID, YEAR(OrderDate)
4. HAVING COUNT(*) > 1
6. ORDER BY SalesPersonID, OrderYear;
```

Hình 5.3 trình bày kết quả của câu lệnh SELECT.



	SalesPersonID	OrderYear
1	279	2005
2	279	2006
3	279	2007
4	279	2008

Hình 5.3: Kết quả của câu lệnh SELECT

5.8 Kiểm tra tiến bộ của bạn

1. Cái nào sau đây được sử dụng để định nghĩa và quản lý tất cả các thuộc tính và tính chất của cơ sở dữ liệu, bao gồm bố trí hàng, định nghĩa cột, cột khóa, địa điểm tập tin, và chiến lược lưu trữ.

(A)	DDL	(C)	DCL
(B)	DML	(D)	DPL

2. Cái nào sau đây không được sử dụng trong DCL?

(A)	GRANT câu lệnh	(C)	UPDATE câu lệnh
(B)	REVOKE câu lệnh	(D)	DENY câu lệnh

3. Cái nào sau đây chỉ ra phạm vi giá trị để kiểm tra.

(A)	IN	(C)	LIKE
(B)	BETWEEN	(D)	CONTAINS

4. So khớp nội dung sau.

Câu lệnh kiểm soát dòng chảy		Mô tả	
a.	IF... ELSE	1.	Đánh dấu một khối lệnh như là một phần của giao tác rõ ràng.
b.	WHILE	2.	Định nghĩa cấu trúc cho ngoại lệ và xử lý lỗi.
c.	BEGIN... END	3.	Lặp lại một câu lệnh hoặc một khối lệnh khi điều kiện vẫn còn đúng.
d.	TRY... CATCH	4.	Định nghĩa phạm vi của khối lệnh Transact-SQL.
e.	BEGIN TRANSACTION	5.	Cung cấp phân nhánh kiểm soát dựa trên một thử nghiệm hợp lý.

(A)	a-4, b-2, c-3, d-1, e-5	(C)	a-1, b-4, c-5, d-3, e-2
(B)	a-1, b-2, c-4, d-3, e-5	(D)	a-5, b-3, c-4, d-2, e-1

5. Điều nào sau đây là hai nguyên tắc cơ bản toán học được sử dụng trong SQL Server 2012.

(A)	Phân số và tập hợp	(C)	Logic xác nhận và phân số
(B)	Tập hợp và logic xác nhận	(D)	Xác suất và phân số

6. Điều nào sau đây sẽ là kết quả của đoạn mã?

```
SET @Number = 2 * (4 + 5) + 2 * (4 + (5 - 3))
```

(A)	120	(C)	42
(B)	62	(D)	26

5.8.1 Câu trả lời

1.	A
2.	C
3.	B
4.	D
5.	B
6.	C

Tóm tắt

- ➔ Transact-SQL là một ngôn ngữ mạnh mẽ đem lại các tính năng như là các kiểu dữ liệu, các đối tượng tạm thời, và các thủ tục đã lưu trữ mở rộng.
- ➔ SQL Server hỗ trợ ba loại câu lệnh Transact-SQL, cụ thể là: DDL, DML và DCL.
- ➔ Kiểu dữ liệu là một thuộc tính định nghĩa kiểu dữ liệu mà một đối tượng có thể có.
- ➔ Những phần tử ngôn ngữ Transact-SQL bao gồm các xác nhận, toán tử, hàm, biến, biểu thức, điều khiển dòng chảy, lỗi, và giao tác, chú thích, và dấu tách khối lệnh.
- ➔ Các tập hợp và logic xác nhận là hai nguyên tắc cơ bản toán học được sử dụng trong SQL Server 2012.
- ➔ Lý thuyết tập hợp là nền tảng toán học được sử dụng trong mô hình cơ sở dữ liệu quan hệ, nơi tập hợp là một bộ sưu tập các đối tượng riêng biệt được coi như toàn thể.
- ➔ Logic xác nhận là một khuôn khổ toán học bao gồm các bài kiểm tra logic cung cấp kết quả.

Hãy thử tự làm

1. Sử dụng Query Editor để thực thi một truy vấn. Bảo đảm là việc kết nối vào một đối tượng định hình máy chủ mới được thiết lập. Sau đó, trong cơ sở dữ liệu AdventureWorks2012, thực hiện một truy vấn để chọn các cột cụ thể là ProductID, Name, và ProductNumber từ bảng Production.Product, và Product ID và ModifiedDate từ bảng Production.ProductDocument.

Phần - 6

Tạo ra và quản lý các cơ sở dữ liệu

Chào mừng bạn đến với phần **Tạo và quản lý cơ sở dữ liệu**.

Phần này mô tả cơ sở dữ liệu do hệ thống và người dùng định nghĩa. Phần này còn liệt kê các tính năng chính của cơ sở dữ liệu AdventureWorks2012. Cuối cùng, phần này mô tả các loại sửa đổi cơ sở dữ liệu.

Trong phần này, bạn sẽ học để:

- ➔ Mô tả cơ sở dữ liệu do hệ thống và người dùng định nghĩa
- ➔ Liệt kê các tính năng chính của cơ sở dữ liệu mẫu AdventureWorks2012.
- ➔ Mô tả thêm các nhóm tập tin và các nhật ký giao tác
- ➔ Mô tả các thủ tục để tạo ra một cơ sở dữ liệu
- ➔ Liệt kê và mô tả các loại sửa đổi cơ sở dữ liệu
- ➔ Mô tả thủ tục để thả một cơ sở dữ liệu
- ➔ Mô tả bức ảnh tức thời cơ sở dữ liệu

6.1 Giới thiệu

Cơ sở dữ liệu là một tập hợp dữ liệu được lưu trong các tập tin dữ liệu trên đĩa hoặc phương tiện có thể tháo rời nào đó. Cơ sở dữ liệu gồm các tập tin dữ liệu để giữ dữ liệu thực tế.

Cơ sở dữ liệu SQL Server được tạo thành một bộ sưu tập các bảng lưu trữ tập hợp dữ liệu có cấu trúc cụ thể. Bảng bao gồm một tập hợp các hàng (còn gọi là các bản ghi hoặc bộ dữ liệu) và cột (còn gọi là thuộc tính). Mỗi cột trong bảng được thiết kế để lưu trữ một loại thông tin cụ thể, ví dụ, ngày tháng, tên, số lượng tiền tệ, và các con số.

Người dùng có thể cài đặt nhiều thể hiện của SQL Server trên một máy tính. Mỗi thể hiện của SQL Server có thể bao gồm nhiều cơ sở dữ liệu. Trong một cơ sở dữ liệu, có các nhóm quyền sở hữu đối tượng khác nhau được gọi là các lược đồ. Trong mỗi lược đồ, có các đối tượng cơ sở dữ liệu như là các bảng, khung nhìn, và thủ tục đã lưu trữ. Một số đối tượng như các chứng nhận và khóa bất đối xứng được chứa trong cơ sở dữ liệu, nhưng không được chứa trong lược đồ.

Cơ sở dữ liệu SQL Server được lưu trữ như các tập tin trong hệ thống tập tin. Những tập tin này được nhóm lại thành các nhóm tập tin. Khi mọi người có được quyền truy cập vào một thể hiện của SQL Server, họ được xác định là một đăng nhập. Khi mọi người có được quyền truy cập vào một cơ sở dữ liệu, họ được xác định là người dùng cơ sở dữ liệu.

Người dùng có quyền truy cập vào một cơ sở dữ liệu có thể được phép truy cập vào các đối tượng trong cơ sở dữ liệu. Mặc dù sự cho phép có thể được cấp cho người dùng riêng lẻ, tốt nhất nên tạo ra các vai trò cơ sở dữ liệu, thêm người dùng cơ sở dữ liệu cho các vai trò, và sau đó, cấp phép quyền truy cập cho những vai trò đó. Cấp quyền truy cập cho các vai trò thay vì cho người dùng làm cho dễ duy trì cấp phép thống nhất và dễ hiểu hơn bởi số lượng người dùng tăng lên và liên tục thay đổi.

SQL Server 2012 hỗ trợ ba loại cơ sở dữ liệu, như sau:

- Cơ sở dữ liệu hệ thống
- Cơ sở dữ liệu do người dùng định nghĩa
- Cơ sở dữ liệu mẫu

6.2 Cơ sở dữ liệu hệ thống

SQL Server sử dụng cơ sở dữ liệu hệ thống để hỗ trợ các phần khác nhau của DBMS. Mỗi cơ sở dữ liệu có một vai trò cụ thể và lưu trữ thông tin công việc đòi hỏi phải được thực hiện bởi SQL Server. Các cơ sở dữ liệu hệ thống lưu trữ dữ liệu trong các bảng, trong đó có các khung nhìn, thủ tục đã lưu trữ, và các đối tượng cơ sở dữ liệu khác. Chúng cũng có các tập tin cơ sở dữ liệu liên kết (ví dụ, các tập tin .mdf and .ldf) được định vị vật lý trên máy SQL Server.

Bảng 6.1 trình bày cơ sở dữ liệu hệ thống được SQL Server 2012 hỗ trợ.

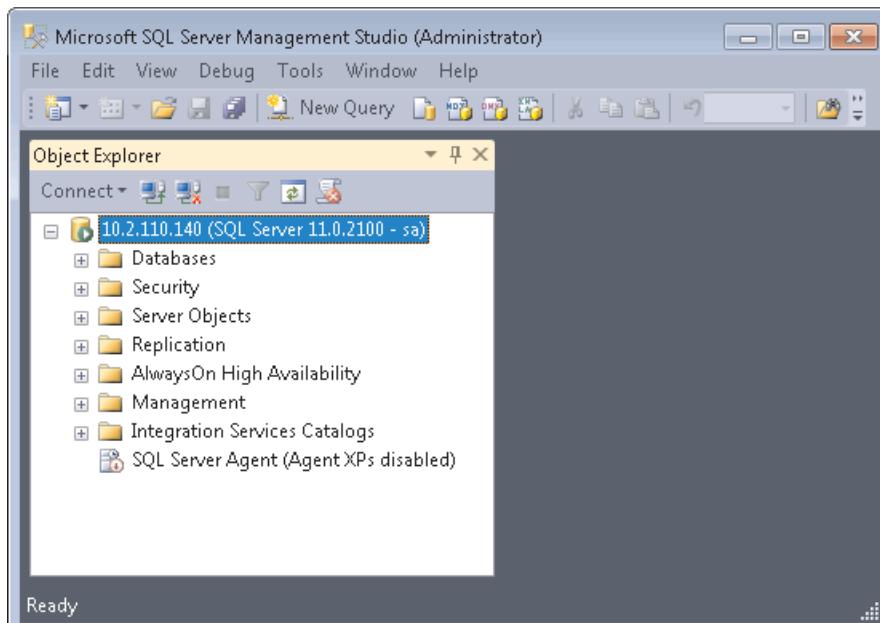
Cơ sở dữ liệu	Mô tả
master	Cơ sở dữ liệu ghi tất cả thông tin mức hệ thống của đối tượng định hình của SQL Server.
msdb	Cơ sở dữ liệu được SQL Server Agent sử dụng để lên lịch gửi các cảnh báo cơ sở dữ liệu và các công việc khác nhau.
model	Cơ sở dữ liệu được dùng làm khuôn mẫu cho tất cả các cơ sở dữ liệu sẽ được tạo ra trên đối tượng định hình cụ thể của SQL Server 2012.
resource	Cơ sở dữ liệu là loại chỉ đọc. Nó chứa các đối tượng hệ thống được đưa vào trong SQL Server 2012.
tempdb	Cơ sở dữ liệu giữ các đối tượng tạm hoặc các tập kết quả trung gian.

Bảng 6.1: Cơ sở dữ liệu hệ thống

6.2.1 Sửa đổi dữ liệu hệ thống

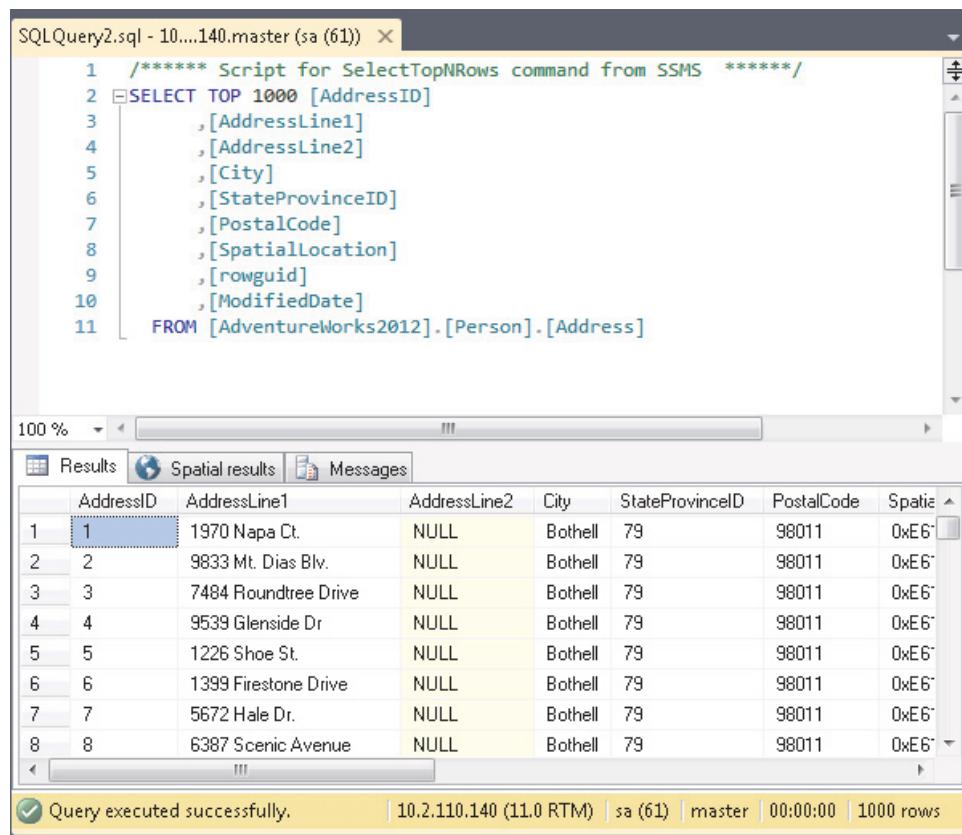
Người dùng không được phép trực tiếp cập nhật thông tin trong các đối tượng cơ sở dữ liệu hệ thống, chẳng hạn như các bảng hệ thống, thủ tục đã lưu trữ của hệ thống, và khung nhìn danh mục. Tuy nhiên, người dùng có thể tận dụng một bộ hoàn chỉnh các công cụ quản trị cho phép họ quản trị đầy đủ hệ thống và quản lý tất cả người dùng và các đối tượng cơ sở dữ liệu. Chúng bao gồm:

- ➔ **Tiện ích quản trị:** Từ SQL Server 2005 trở đi, một số tiện ích quản trị SQL Server được tích hợp vào SSMS. Đây là giao diện quản trị cốt lõi cho việc cài đặt SQL Server. Nó cho phép thực hiện các chức năng quản trị cao cấp, các công việc bảo trì định kỳ theo lịch, và vân vân. Hình 6.1 trình bày cửa sổ SQL Server 2012 Management Studio.



Hình 6.1: Cửa sổ SQL Server Management Studio

- **SQL Server Management Objects (SQL-SMO) API:** Bao gồm chức năng đầy đủ để quản trị SQL Server trong các ứng dụng.
- **Tập lệnh và thủ tục đã lưu trữ của Transact-SQL:** Những cái này sử dụng các thủ tục đã lưu trữ của hệ thống và câu lệnh Transact-SQL DDL. Hình 6.2 trình bày cửa sổ truy vấn Transact-SQL.



The screenshot shows the SSMS interface with a query window titled "SQLQuery2.sql - 10...140.master (sa (61))". The query is:

```

1  /***** Script for SelectTopNRows command from SSMS *****/
2  SELECT TOP 1000 [AddressID]
3      ,[AddressLine1]
4      ,[AddressLine2]
5      ,[City]
6      ,[StateProvinceID]
7      ,[PostalCode]
8      ,[SpatialLocation]
9      ,[rowguid]
10     ,[ModifiedDate]
11  FROM [AdventureWorks2012].[Person].[Address]

```

Below the query window is a results grid showing 10 rows of address data from the AdventureWorks2012 database. The columns are: AddressID, AddressLine1, AddressLine2, City, StateProvinceID, PostalCode, and SpatialLocation. The first few rows are:

AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode	SpatialLocation
1	1970 Napa Ct.	NULL	Bothell	79	98011	0xE6...
2	9833 Mt. Dias Blvd.	NULL	Bothell	79	98011	0xE6...
3	7484 Roundtree Drive	NULL	Bothell	79	98011	0xE6...
4	9539 Glenside Dr	NULL	Bothell	79	98011	0xE6...
5	1226 Shoe St.	NULL	Bothell	79	98011	0xE6...
6	1399 Firestone Drive	NULL	Bothell	79	98011	0xE6...
7	5672 Hale Dr.	NULL	Bothell	79	98011	0xE6...
8	6387 Scenic Avenue	NULL	Bothell	79	98011	0xE6...

At the bottom of the results grid, a message says "Query executed successfully." followed by connection information: 10.2.110.140 (11.0 RTM) | sa (61) | master | 00:00:00 | 1000 rows.

Hình 6.2: Cửa sổ truy vấn Transact-SQL

Những công cụ này còn bảo vệ các ứng dụng khỏi việc thay đổi trong các đối tượng hệ thống.

6.2.2 Xem dữ liệu của cơ sở dữ liệu hệ thống

Các ứng dụng cơ sở dữ liệu có thể xác định thông tin hệ thống và danh mục bằng cách dùng bất kỳ phương pháp nào sau đây:

- **Khung nhìn danh mục hệ thống:** Khung nhìn hiển thị siêu dữ liệu để mô tả các đối tượng cơ sở dữ liệu trong một thể hiện SQL Server.
- **SQL-SMO:** Mô hình đối tượng lập mã có quản lý mới, cung cấp một tập hợp đối tượng được dùng để quản lý Microsoft SQL Server.
- **Các hàm danh mục, phương thức, thuộc tính, hoặc tính chất của dữ liệu API:** Được sử dụng trong các ứng dụng ActiveX Data Objects (ADO), OLE DB, hoặc ODBC.
- **Thủ tục đã lưu trữ và hàm:** Được sử dụng trong Transact-SQL làm các thủ tục đã lưu trữ và hàm dụng sẵn.

6.3 Cơ sở dữ liệu do người dùng định nghĩa

Khi sử dụng SQL Server 2012, người dùng có thể tạo ra cơ sở dữ liệu riêng của họ, còn được gọi là cơ sở dữ liệu do người dùng định nghĩa, và làm việc với chúng. Mục đích của những cơ sở dữ liệu này là để lưu dữ liệu của người dùng.

6.3.1 Tạo các cơ sở dữ liệu

Để tạo ra một cơ sở dữ liệu do người dùng định nghĩa, thông tin cần thiết như sau:

- ➔ Tên của cơ sở dữ liệu
- ➔ Chủ nhân hoặc người tạo ra cơ sở dữ liệu
- ➔ Kích thước của cơ sở dữ liệu
- ➔ Các tập tin và nhóm tập tin được dùng để lưu dữ liệu

Sau đây là cú pháp để tạo ra một cơ sở dữ liệu do người dùng định nghĩa.

Cú pháp:

```
CREATE DATABASE DATABASE_NAME
[ ON
[ PRIMARY ] [ <filespec> [ ,...n ]
[ , <filegroup> [ ,...n ] ]
[ LOG ON { <filespec> [ ,...n ] } ]
]
[ COLLATE collation_name ]
]
[ ; ]
```

trong đó:

DATABASE_NAME: là tên của cơ sở dữ liệu được tạo ra.

ON: chỉ ra các tập tin trên đĩa được sử dụng để lưu trữ các phần dữ liệu của cơ sở dữ liệu và các tập tin dữ liệu.

PRIMARY: là danh sách <filespec> có liên quan định nghĩa tập tin chính.

<filespec>: kiểm soát các thuộc tính tập tin.

<filegroup>: kiểm soát các thuộc tính nhóm tập tin.

LOG ON: chỉ ra các tập tin trên đĩa sẽ được sử dụng để lưu trữ nhật ký cơ sở dữ liệu và các tập tin nhật ký. **COLLATE collation_name:** là đối chiếu mặc định cho cơ sở dữ liệu. Sự đổi chiếu định nghĩa các quy tắc để so sánh và sắp xếp dữ liệu ký tự dựa trên tiêu chuẩn về ngôn ngữ và locale cụ thể. Tên đối chiếu có thể là tên đối chiếu Windows hoặc tên đối chiếu SQL.

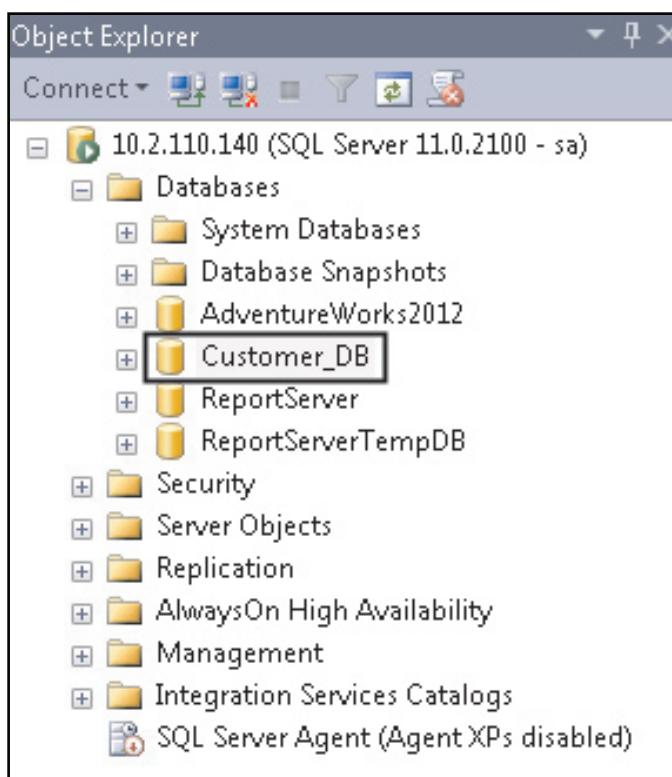
Đoạn mã 1 trình bày cách để tạo ra một cơ sở dữ liệu với tập tin cơ sở dữ liệu và tập tin nhật ký giao tác với tên đối chiếu.

Đoạn mã 1:

```
CREATE DATABASE [Customer_DB] ON PRIMARY
( NAME = 'Customer_DB', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Customer_DB.mdf')
LOG ON
( NAME = 'Customer_DB_log', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Customer_DB_log.ldf')
COLLATE SQL_Latin1_General_CI_AS
```

Sau khi thực thi mã trong Đoạn mã 1, SQL Server 2012 hiển thị thông báo 'Command(s) completed successfully' (Lệnh được thực hiện thành công).

Hình 6.3 trình bày cơ sở dữ liệu **Customer_DB** được liệt kê trong **Object Explorer**.



Hình 6.3: Cơ sở dữ liệu Customer_DB

6.3.2 Sửa đổi các cơ sở dữ liệu

Bởi cơ sở dữ liệu do người dùng định nghĩa tăng lên hoặc giảm đi, kích thước cơ sở dữ liệu sẽ được mở rộng hoặc được thu nhỏ tự động hoặc thủ công. Dựa trên các yêu cầu thay đổi theo thời gian, nó có thể được thấy cần thiết phải sửa đổi cơ sở dữ liệu.

Sau đây là cú pháp để sửa đổi cơ sở dữ liệu:

Cú pháp:

```
ALTER DATABASE database_name
{
<add_or_modify_files>
| <add_or_modify_filegroups>
| <set_database_options>
| MODIFY NAME = new_database_name
| COLLATE collation_name
}
[;]
```

trong đó:

`database_name`: là tên ban đầu của cơ sở dữ liệu.

`MODIFY NAME = new_database_name`: là tên mới của cơ sở dữ liệu sẽ được đổi tên.

`COLLATE collation_name`: là tên đối chiếu của cơ sở dữ liệu.

`<add_or_modify_files>`: là tập tin được thêm vào, gỡ bỏ, hoặc sửa đổi.

`<add_or_modify_filegroups>`: là nhóm tập tin được thêm vào, chỉnh sửa hoặc loại bỏ khỏi cơ sở dữ liệu.

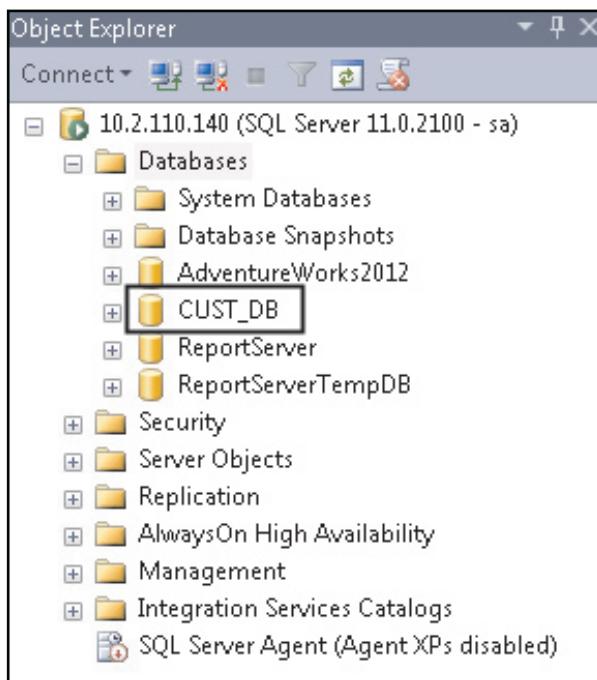
`<set_database_options>`: là tùy chọn mức cơ sở dữ liệu ảnh hưởng đến các đặc tính của cơ sở dữ liệu có thể được thiết lập cho mỗi cơ sở dữ liệu. Các tùy chọn này là duy nhất cho từng cơ sở dữ liệu và không ảnh hưởng đến các cơ sở dữ liệu khác.

Đoạn mã 2 trình bày cách để đổi tên cơ sở dữ liệu `Customer_DB` bằng một tên cơ sở dữ liệu mới `CUST_DB`.

Đoạn mã 2:

```
ALTER DATABASE Customer_DB MODIFY NAME = CUST_DB
```

Hình 6.4 trình bày cơ sở dữ liệu Customer_DB được đổi tên thành tên cơ sở dữ liệu mới CUST_DB.



Hình 6.4: Tên cơ sở dữ liệu mới CUST_DB

6.3.3 Quyền sở hữu cơ sở dữ liệu

Trong SQL Server 2012, có thể thay đổi quyền sở hữu một cơ sở dữ liệu do người dùng định nghĩa. Không thể thay đổi chủ sở hữu của các cơ sở dữ liệu hệ thống. Thủ tục hệ thống `sp_changedbowner` được sử dụng để thay đổi quyền sở hữu của một cơ sở dữ liệu. Cú pháp như sau:

Cú pháp:

```
sp_changedbowner [ @loginame = ] 'login'
```

trong đó:

`login` là tên sử dụng cơ sở dữ liệu hiện có.

Sau khi `sp_changedbowner` được thực thi, chủ sở hữu mới được gọi là người dùng `dbo` bên trong cơ sở dữ liệu đã chọn. `dbo` nhận được cấp phép để thực hiện tất cả các hoạt động trong cơ sở dữ liệu. Không thể thay đổi chủ sở hữu của các cơ sở dữ liệu hệ thống master, model, hoặc tempdb.

Đoạn mã 3, khi thực thi, làm cho đăng nhập SSSqsaSSSq thành chủ sở hữu của cơ sở dữ liệu hiện tại và ánh xạ 'sa' cho các bí danh hiện có đã được gán cho chủ sở hữu cơ sở dữ liệu cũ, và sẽ hiển thị 'Command(s) completed successfully' (Lệnh đã thực hiện thành công).

Đoạn mã 3:

```
USE CUST_DB
EXEC sp_changedbowner 'sa'
```

6.3.4 Cài đặt các tùy chọn cơ sở dữ liệu

Các tùy chọn mức cơ sở dữ liệu xác định các đặc tính của cơ sở dữ liệu mà có thể được đặt cho từng cơ sở dữ liệu. Những tùy chọn này là duy nhất cho mỗi cơ sở dữ liệu, vì vậy chúng không ảnh hưởng đến các cơ sở dữ liệu khác. Những tùy chọn cơ sở dữ liệu này được đặt thành các giá trị mặc định khi một cơ sở dữ liệu đầu tiên được tạo ra, và có thể sau đó, được thay đổi bằng cách sử dụng mệnh đề SET của câu lệnh ALTER DATABASE.

Bảng 6.2 trình bày các tùy chọn cơ sở dữ liệu được SQL Server 2012 hỗ trợ.

Loại tùy chọn	Mô tả
Các tùy chọn tự động	Kiểm soát hành vi tự động của cơ sở dữ liệu.
Các tùy chọn con trỏ	Kiểm soát hành vi con trỏ
Các tùy chọn phục hồi	Kiểm soát các mô hình phục hồi của cơ sở dữ liệu.
Các tùy chọn hỗn hợp	Kiểm soát tuân thủ ANSI.
Các tùy chọn tình trạng	Kiểm soát tình trạng của cơ sở dữ liệu, như là khả năng kết nối người dùng và trực tuyến/ngoại tuyến.

Bảng 6.2: Các tùy chọn cơ sở dữ liệu trong SQL Server 2012

Ghi chú - Các thiết đặt trên toàn máy chủ được thiết lập sử dụng thủ tục đã lưu trữ của hệ thống sp_configure hoặc SQL Management Studio.

Đoạn mã 4 khi thực thi sẽ đặt tùy chọn AUTO_SHRINK cho cơ sở dữ liệu CUST_DB thành ON. Tùy chọn AUTO_SHRINK khi đặt thành ON, thu nhỏ cơ sở dữ liệu có không gian trống.

Đoạn mã 4:

```
USE CUST_DB;
ALTER DATABASE CUST_DB
SET AUTO_SHRINK ON
```

6.4 Cơ sở dữ liệu mẫu

Cơ sở dữ liệu mẫu AdventureWorks đã được giới thiệu từ SQL Server 2005 trở đi. Cơ sở dữ liệu này cho thấy việc sử dụng các tính năng mới được giới thiệu trong SQL Server. Một công ty hư cấu có tên là Adventure Works Cycles được tạo ra như một kịch bản trong cơ sở dữ liệu. Adventure Works Cycles là một công ty sản xuất lớn, đa quốc gia. Công ty này sản xuất và bán xe đạp kim loại và hợp kim cho thị trường thương mại Bắc Mỹ, Châu Âu và Châu Á. Trong SQL Server 2012, phiên bản mới của cơ sở dữ liệu mẫu AdventureWorks2012 được sử dụng.

6.4.1 Cơ sở dữ liệu AdventureWorks2012

Cơ sở dữ liệu AdventureWorks2012 bao gồm khoảng 100 tính năng. Một số đặc điểm chính như sau:

- ➔ Công cụ cơ sở dữ liệu bao gồm các tiện ích quản trị, khả năng truy cập dữ liệu, tiện ích tìm kiếm toàn văn bản, lợi thế tích hợp Common Language Runtime (CLR), SMO, Service Broker, và XML.

- ➔ Các dịch vụ phân tích
- ➔ Các dịch vụ tích hợp
- ➔ Các dịch vụ thông báo
- ➔ Các dịch vụ báo cáo
- ➔ Các tiện ích sao chép
- ➔ Một tập hợp các mẫu được tích hợp cho hai mẫu dựa trên đa đặc điểm: HRResume và Storefront.

Cơ sở dữ liệu mẫu gồm các phần sau:

- ➔ AdventureWorks2012: Cơ sở dữ liệu OLTP mẫu
- ➔ AdventureWorks2012DW: Kho dữ liệu mẫu
- ➔ AdventureWorks2012AS: Cơ sở dữ liệu mẫu của Dịch vụ phân tích

6.4.2 Nhóm tập tin

Trong SQL Server, các tập tin dữ liệu được sử dụng để lưu trữ các tập tin cơ sở dữ liệu. Các tập tin dữ liệu được chia nhỏ thêm thành các nhóm tập tin để tăng hiệu suất. Mỗi nhóm tập tin được dùng để nhóm các tập tin có liên quan để cùng nhau lưu một đối tượng cơ sở dữ liệu. Mỗi cơ sở dữ liệu có một nhóm tập tin chính theo mặc định. Nhóm tập tin này chứa tập tin dữ liệu chính. Nhóm tập tin chính và các tập tin dữ liệu được tạo tự động với các giá trị thuộc tính mặc định vào thời điểm tạo ra cơ sở dữ liệu. Sau đó có thể tạo ra các nhóm tập tin do người dùng định nghĩa để nhóm các tập tin dữ liệu lại với nhau cho mục đích quản trị, phân bổ và bố trí dữ liệu.

Ví dụ, ba tập tin có tên là `Customer_Data1.ndf`, `Customer_Data2.ndf`, và `Customer_Data3.ndf`, có thể được tạo ra trên ba ổ đĩa tương ứng. Những tập tin này sau đó có thể được gán cho nhóm tập tin `Customer_fgroup1`. Sau đó bảng có thể được tạo ra đặc biệt trên nhóm tập tin `Customer_fgroup1`. Truy vấn dữ liệu từ bảng sẽ được trải qua ba ổ đĩa do đó, cải thiện hiệu suất.

Bảng 6.3 trình bày các nhóm tập tin được SQL Server 2012 hỗ trợ.

Nhóm tập tin	Mô tả
Chính	Nhóm tập tin này chứa tập tin chính. Tất cả các bảng hệ thống được đặt bên trong nhóm tập tin chính.
Do người dùng định nghĩa	Bất kỳ nhóm tập tin nào được người dùng tạo ra vào lúc tạo ra và sửa đổi các cơ sở dữ liệu.

Bảng 6.3: Các nhóm tập tin trong SQL Server 2012

Thêm các nhóm tập tin vào một cơ sở dữ liệu hiện hữu

Có thể tạo ra các nhóm tập tin khi cơ sở dữ liệu được tạo ra lần đầu tiên hoặc có thể được tạo ra sau này khi có nhiều tập tin được thêm vào cơ sở dữ liệu. Tuy nhiên, không thể dời các tập tin tới một nhóm tập tin khác sau khi các tập tin đã được thêm vào cơ sở dữ liệu.

Một tập tin mỗi lần không thể là thành viên của nhiều hơn một nhóm tập tin. Có thể tạo ra tối đa 32.767 nhóm tập tin cho mỗi cơ sở dữ liệu. Các nhóm tập tin có thể chỉ chứa các tập tin dữ liệu. Các tập tin nhặt ký giao tác không thể thuộc về một nhóm tập tin.

Sau đây là cú pháp để thêm các nhóm tập tin trong khi tạo cơ sở dữ liệu:

Cú pháp:

```
CREATE DATABASE database_name
[ ON
[ PRIMARY ] [ <filespec> [ ,...n ]
[ , <filegroup> [ ,...n ] ]
[ LOG ON { <filespec> [ ,...n ] } ]
]
[ COLLATE collation_name ]
]
[ ; ]
```

trong đó:

`database_name`: là tên của cơ sở dữ liệu mới.

`ON`: chỉ ra các tập tin trên đĩa để lưu trữ các phần dữ liệu của cơ sở dữ liệu và các tập tin dữ liệu.

`PRIMARY` và danh sách `<filespec>` có liên quan: định nghĩa tập tin chính. Tập tin đầu tiên được chỉ ra trong mục nhập `<filespec>` trong nhóm tập tin chính trở thành tập tin chính.

`LOG ON`: chỉ ra các tập tin trên đĩa được sử dụng để lưu trữ các tập tin nhật ký cơ sở dữ liệu.

`COLLATE collation_name`: là đối chiếu mặc định cho cơ sở dữ liệu.

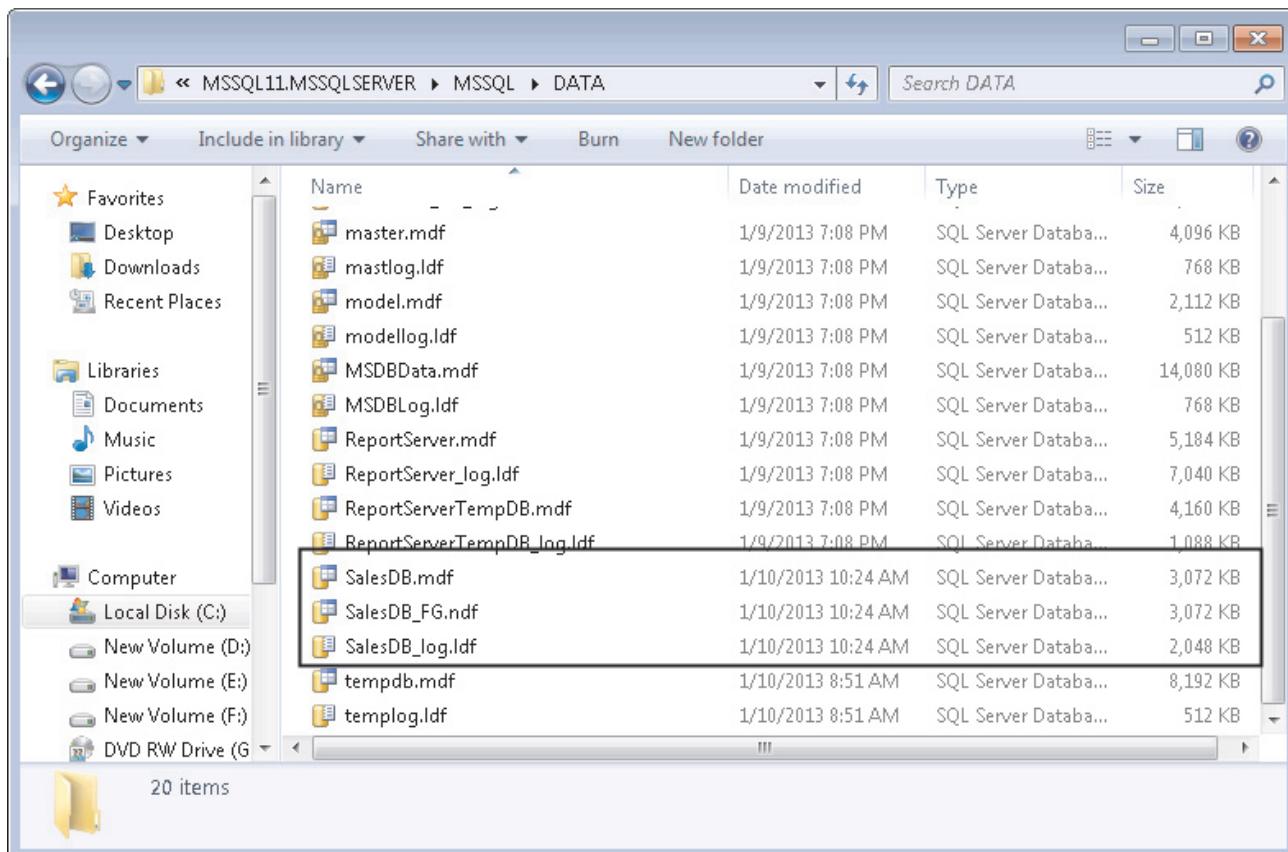
Đoạn mã 5 trình bày cách để thêm một nhóm tập tin (`PRIMARY` làm mặc định) trong khi tạo ra một cơ sở dữ liệu, được gọi là `SalesDB`.

Đoạn mã 5:

```
CREATE DATABASE [SalesDB] ON PRIMARY
( NAME = 'SalesDB', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\SalesDB.mdf' , SIZE = 3072KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 1024KB ) ,
FILEGROUP [MyFileGroup]
( NAME = 'SalesDB_FG', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\SalesDB_FG.ndf' , SIZE = 3072KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 1024KB )
```

```
LOG ON
( NAME = 'SalesDB_log', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\SalesDB_log.ldf' , SIZE = 2048KB , MAXSIZE = 2048GB , FILEGROWTH = 10% )
COLLATE SQL_Latin1_General_CI_AS
```

Hình 6.5 trình bày các nhóm tập tin khi tạo ra cơ sở dữ liệu **SalesDB**.



Hình 6.5: Nhóm tập tin đã thêm vào khi tạo cơ sở dữ liệu SalesDB

Sau đây là cú pháp để thêm nhóm tập tin vào cơ sở dữ liệu hiện có:

Cú pháp:

```
ALTER DATABASE database_name
{ <add_or_modify_files>
| <add_or_modify_filegroups>
| <set_database_options>
| MODIFY NAME = new_database_name
| COLLATE collation_name
}
[ ; ]
```

Đoạn mã 6 trình bày cách để thêm một nhóm tập tin cho một cơ sở dữ liệu hiện có, được gọi là **CUST_DB**.

Đoạn mã 6:

```
USE CUST_DB;
ALTER DATABASE CUST_DB
ADD FILEGROUP FG_ReadOnly
```

Sau khi thực thi mã, SQL Server 2012 hiển thị thông báo 'Command(s) completed successfully' (Lệnh đã thực hiện thành công) và nhóm tập tin **FG_ReadOnly** được thêm vào cơ sở dữ liệu hiện có **CUST_DB**.

Nhóm tập tin mặc định

Các đối tượng được gán cho nhóm tập tin mặc định khi chúng được tạo ra trong cơ sở dữ liệu. Nhóm tập tin **PRIMARY** là nhóm tập tin mặc định. Có thể thay đổi nhóm tập tin mặc định sử dụng câu lệnh **ALTER DATABASE**. Các đối tượng và bảng hệ thống vẫn còn trong nhóm tập tin **PRIMARY**, nhưng không đi vào nhóm tập tin mặc định mới.

Để làm nhóm tập tin **FG_ReadOnly** thành mặc định, cần có ít nhất một tập tin bên trong nó.

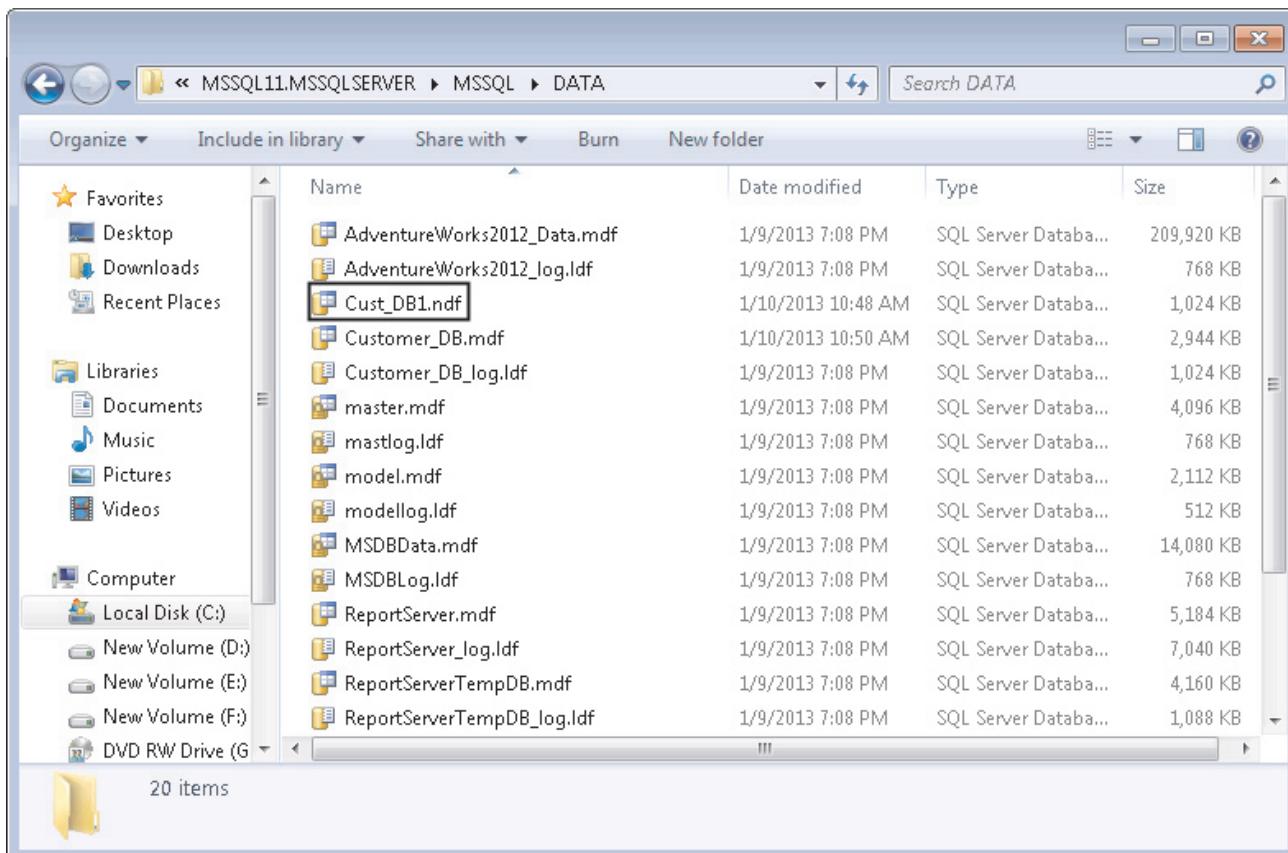
Đoạn mã 7 trình bày cách để tạo ra một tập tin mới, thêm nó vào nhóm tập tin **FG_ReadOnly** và làm cho nhóm tập tin **FG_ReadOnly** đã được tạo ra trong Đoạn mã 6 làm nhóm tập tin mặc định.

Đoạn mã 7:

```
USE CUST_DB
ALTER DATABASE CUST_DB
ADD FILE (NAME = Cust_DB1, FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Cust_DB1.ndf')
TO FILEGROUP FG_ReadOnly
ALTER DATABASE CUST_DB
MODIFY FILEGROUP FG_ReadOnly DEFAULT
```

Sau khi thực thi mã trong Đoạn mã 7, SQL Server 2012 hiển thị thông báo nói rằng thuộc tính nhóm tập tin 'DEFAULT' đã được đặt.

Hình 6.6 trình bày một tập tin mới **Cust_DB1** được tạo ra.



Hình 6.6: Tập tin mới **Cust_DB1** được tạo ra

6.4.3 Nhật ký giao tác

Nhật ký giao tác trong SQL Server ghi lại tất cả các giao tác và sửa đổi cơ sở dữ liệu do mỗi giao tác tạo ra. Nhật ký giao tác là một trong các thành phần hết sức quan trọng của cơ sở dữ liệu. Nó có thể là nguồn dữ liệu mới nhất trong trường hợp hệ thống bị lỗi.

Các nhật ký giao tác hỗ trợ các hoạt động như sau:

→ **Khôi phục các giao tác riêng lẻ**

Một giao tác không đầy đủ được phục hồi lại trong trường hợp ứng dụng ban hành câu lệnh ROLLBACK hoặc Công cụ Cơ sở dữ liệu phát hiện lỗi. Các bản ghi nhật ký này được dùng để quay ngược lại các sửa đổi.

→ **Khôi phục tất cả các giao tác chưa hoàn thành khi khởi động SQL Server**

Nếu máy chủ chạy SQL Server bị lỗi, các cơ sở dữ liệu có thể được giữ ở trạng thái không ổn định. Khi một đối tượng định hình của SQL Server được khởi động, nó chạy việc khôi phục của từng cơ sở dữ liệu.

→ **Việc quay ngược lại cơ sở dữ liệu đã phục hồi, tập tin, nhóm tập tin, hoặc trang chuyển tiếp đến điểm thất bại**

Có thể khôi phục cơ sở dữ liệu tới điểm bị lỗi sau khi lỗi đĩa hoặc mất phần cứng ảnh hưởng đến các tập tin cơ sở dữ liệu.

→ **Hỗ trợ việc sao chép giao tác**

Log Reader Agent giám sát nhật ký giao tác của từng cơ sở dữ liệu được cấu hình cho việc sao chép các giao tác.

→ **Hỗ trợ các giải pháp máy chủ dự phòng**

Các giải pháp máy chủ dự phòng, phản chiếu cơ sở dữ liệu, và vận chuyển nhật ký phụ thuộc vào nhật ký giao tác.

Phần tiếp theo giải thích hoạt động của các nhật ký giao tác và thêm tập tin nhật ký vào cơ sở dữ liệu.

→ **Hoạt động của nhật ký giao tác:**

Cơ sở dữ liệu trong SQL Server 2012 có ít nhất một tập tin dữ liệu và một tập tin nhật ký giao tác. Thông tin nhật ký giao tác và dữ liệu được giữ tách riêng trên cùng tập tin. Các tập tin riêng lẻ chỉ được một cơ sở dữ liệu sử dụng.

SQL Server sử dụng nhật ký giao tác của từng cơ sở dữ liệu để khôi phục các giao tác. Nhật ký giao tác là một bản ghi nối tiếp của tất cả các sửa đổi đã xuất hiện trong cơ sở dữ liệu cũng như các giao tác đã thực hiện các sửa đổi này. Nhật ký này giữ đủ thông tin để hoàn tác các sửa đổi đã thực hiện trong từng giao tác. Nhật ký giao tác ghi việc phân bổ và giải phóng các trang và việc cam kết hoặc quay lại của từng giao tác. Tính năng này cho phép SQL Server cuộn về phía trước hoặc để trở lại.

Việc cuộn ngược lại của mỗi giao tác được thực hiện bằng cách sử dụng những cách sau đây:

- Giao tác được cuộn về trước khi nhật ký giao tác được áp dụng.
- Giao dịch được quay lui khi giao tác không hoàn chỉnh bị rút lại.

→ Thêm các tập tin nhật ký vào cơ sở dữ liệu

Sau đây là cú pháp để sửa đổi một cơ sở dữ liệu và thêm các tập tin đăng nhập.

Cú pháp:

```
ALTER DATABASE database_name
{
...
}
[ ; ]
<add_or_modify_files>::=
{
ADD FILE <filespec> [ ,...n ]
```

```
[ TO FILEGROUP { filegroup_name | DEFAULT } ]
| ADD LOG FILE <filespec> [ ,...n ]
| REMOVE FILE logical_file_name
| MODIFY FILE <filespec>
}
```

Ghi chú - Theo mặc định, dữ liệu và các nhật ký giao tác được đặt trên cùng một ổ đĩa và đường dẫn để chứa các hệ thống đĩa đơn, nhưng có thể không được tối ưu cho các môi trường sản xuất.

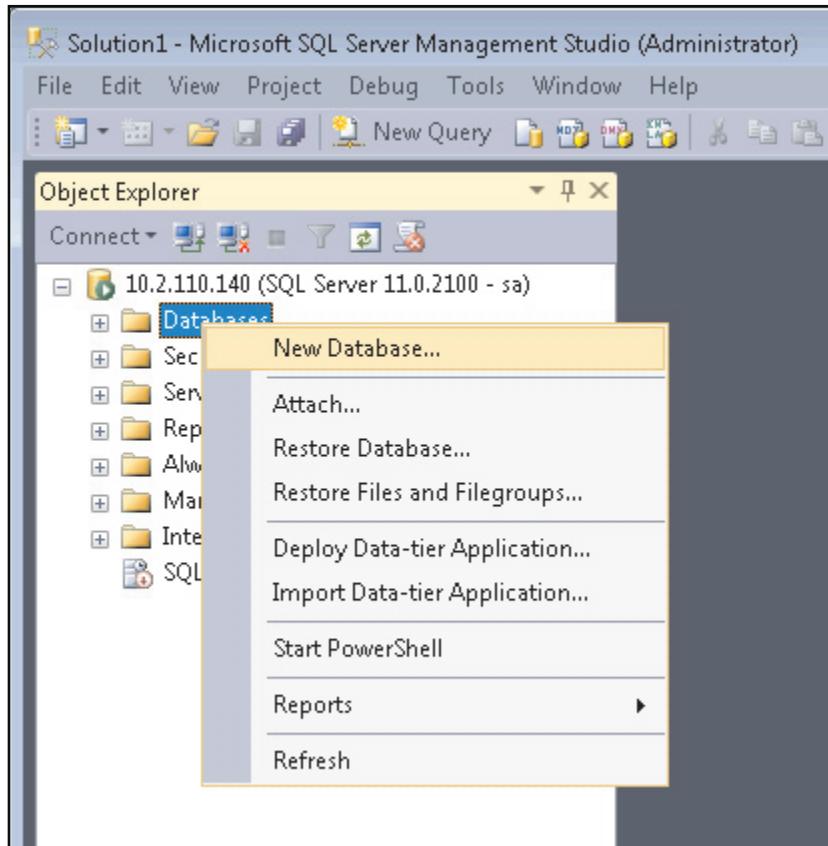
6.4.4 Tạo ra thủ tục cơ sở dữ liệu

Khi một cơ sở dữ liệu được tạo ra, các tập tin dữ liệu cần phải càng lớn càng tốt dựa trên số lượng dữ liệu tối đa, dự kiến trong cơ sở dữ liệu.

Các bước để tạo ra một cơ sở dữ liệu sử dụng SSMS như sau:

1. Trong **Object Explorer**, kết nối với một thể hiện của SQL Server Database Engine và sau đó, mở rộng thể hiện đó.

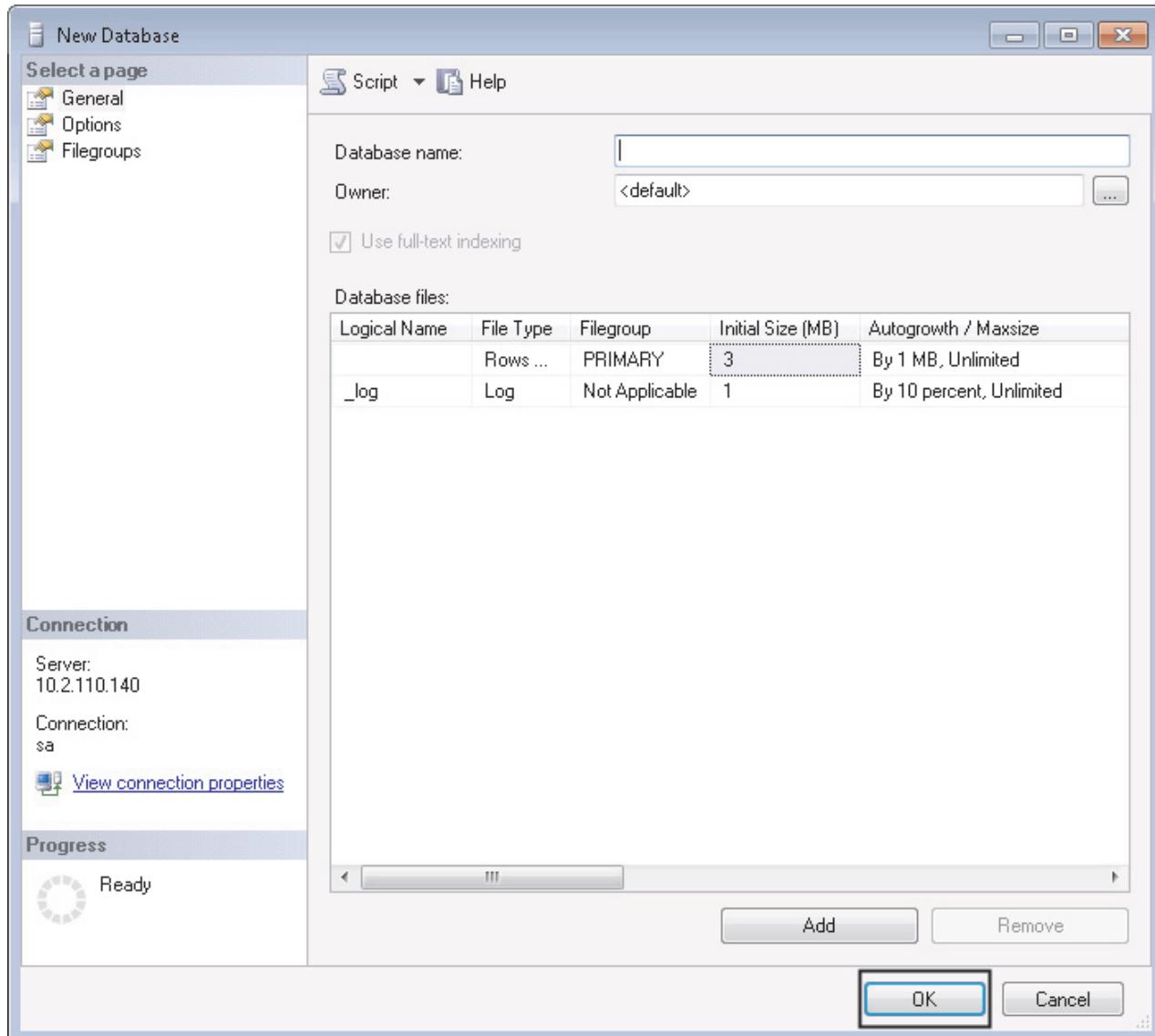
2. Bấm chuột phải vào **Databases**, và sau đó, bấm vào **New Database** như trong hình 6.7.



Hình 6.7: Bấm vào tùy chọn New Database

3. Trong **New Database**, nhập tên cơ sở dữ liệu.

4. Để tạo ra cơ sở dữ liệu bằng cách chấp nhận tất cả các giá trị mặc định, bấm vào **OK**, như được trình bày trong hình 6.8, nếu không, tiếp tục với các bước tùy chọn sau đây.



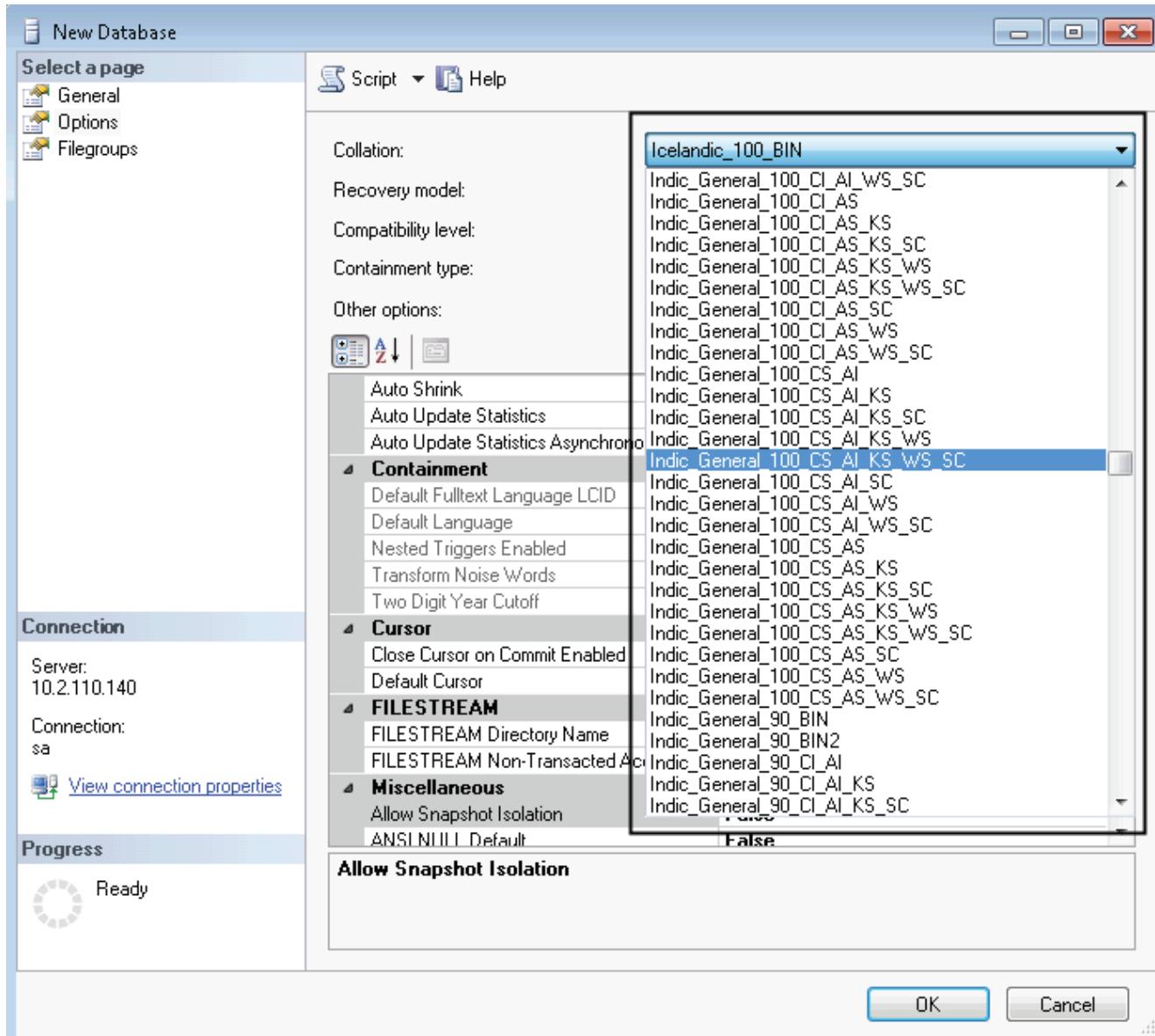
Hình 6.8: Cửa sổ New Database

5. Để thay đổi tên chủ sở hữu, bấm vào (...) để chọn chủ sở hữu khác.

Ghi chú - Tùy chọn 'Use full-text indexing' (Sử dụng tạo chỉ mục toàn văn bản) luôn được chọn và mờ đi bởi vì, từ SQL Server 2008 trở đi, tất cả các cơ sở dữ liệu do người dùng định nghĩa là cho phép toàn văn bản.

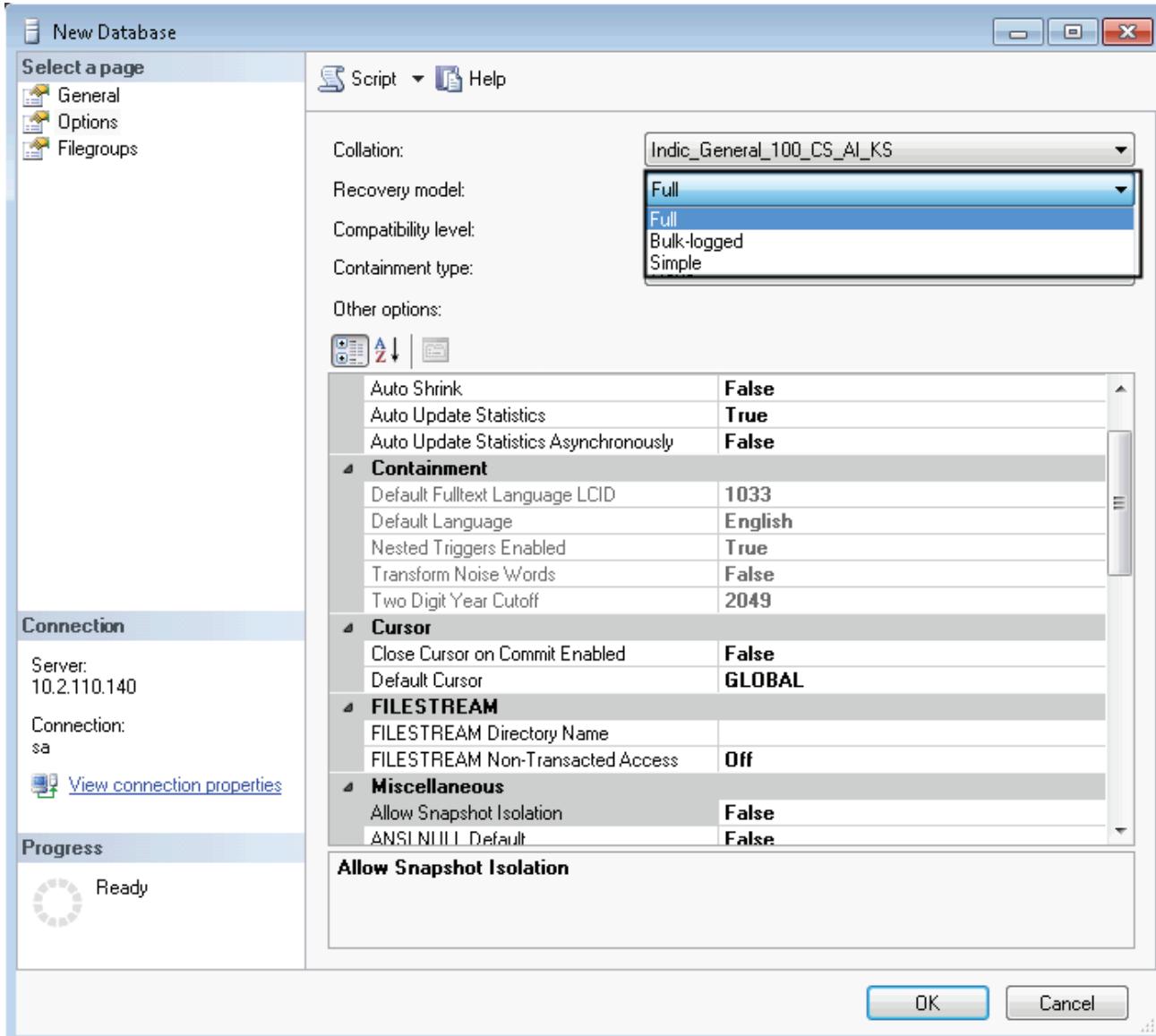
6. Để thay đổi các giá trị mặc định của dữ liệu chính và các tập tin nhật ký giao tác, trong lưới **Database files**, bấm vào ô thích hợp và nhập giá trị mới.

7. Để thay đổi đối chiếu của cơ sở dữ liệu, chọn trang **Options**, và sau đó, chọn một đối chiếu từ danh sách như được trình bày trong hình 6.9.



Hình 6.9: Danh sách đối chiếu

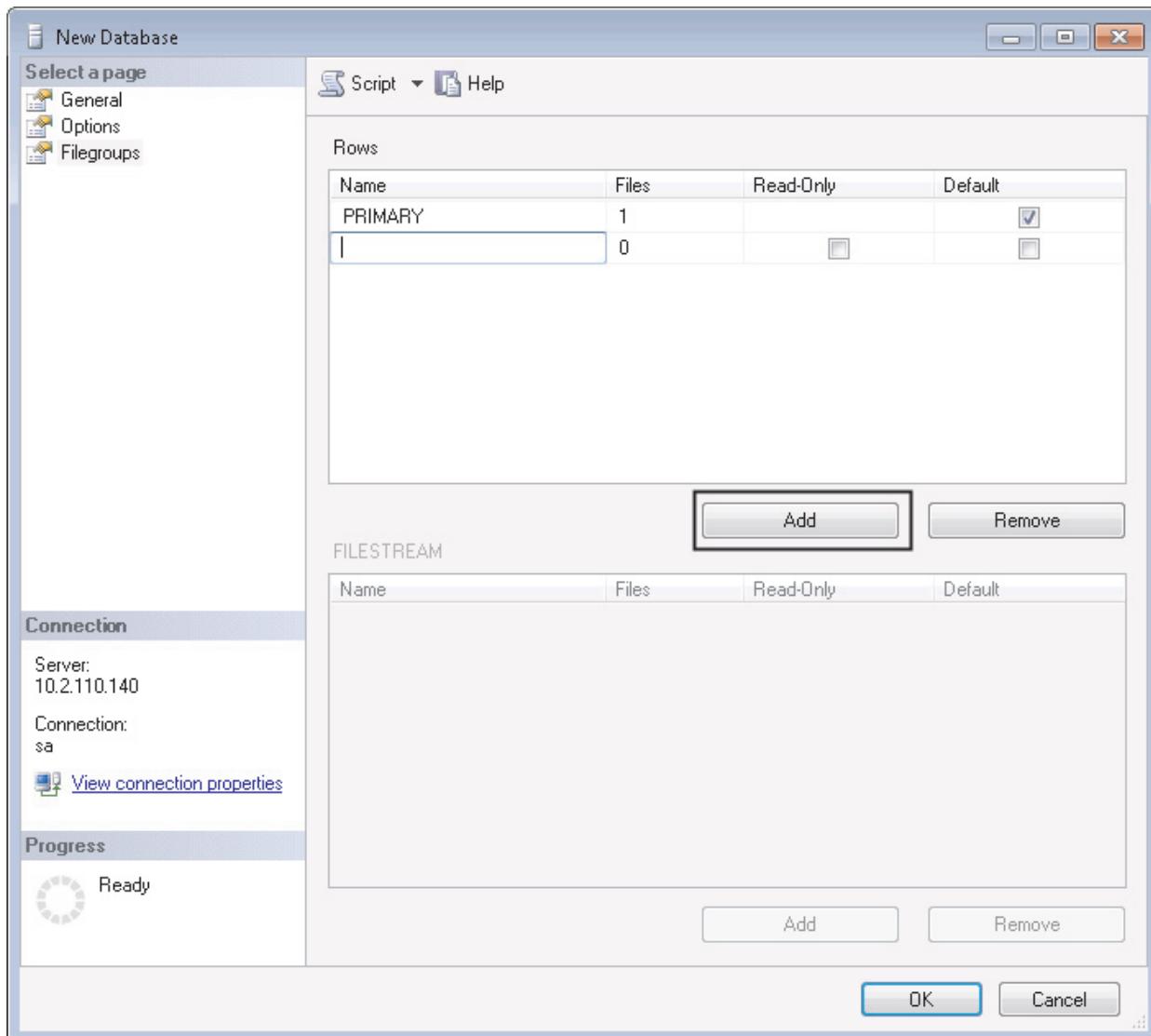
8. Để thay đổi mô hình phục hồi, chọn trang **Options**, và sau đó, chọn một mô hình phục hồi từ danh sách như được trình bày trong hình 6.10.



Hình 6.10: Mô hình phục hồi

9. Để thay đổi các tùy chọn cơ sở dữ liệu, chọn trang **Options**, và sau đó, thay đổi các tùy chọn cơ sở dữ liệu.

10. Để thêm một nhóm tập tin mới, nhấp vào trang **Filegroups**. Bấm vào **Add** và sau đó, nhập các giá trị cho nhóm tập tin như được trình bày trong hình 6.11.



Hình 6.11: Thêm một nhóm tập tin

11. Để thêm một thuộc tính mở rộng vào cơ sở dữ liệu, chọn trang **Extended Properties**.
- Trong cột **Name**, nhập tên cho thuộc tính mở rộng.
 - Trong cột **Value**, nhập văn bản thuộc tính mở rộng. Ví dụ, nhập một hoặc nhiều câu lệnh mô tả cơ sở dữ liệu.
12. Để tạo ra cơ sở dữ liệu, nhấp vào **OK**.

6.4.5 Các loại phương thức sửa đổi cơ sở dữ liệu

Bảng 6.4 liệt kê và mô tả các loại sửa đổi cơ sở dữ liệu và các phương thức sửa đổi.

Loại sửa đổi	Phương thức sửa đổi
Tăng kích thước của cơ sở dữ liệu	Câu lệnh ALTER DATABASE hoặc các thuộc tính cơ sở dữ liệu trong SSMS
Thay đổi vị trí vật lý của cơ sở dữ liệu	Câu lệnh ALTER DATABASE
Thêm các tập tin nhật ký giao tác hoặc dữ liệu	Câu lệnh ALTER DATABASE hoặc các thuộc tính cơ sở dữ liệu trong SSMS
Co lại cơ sở dữ liệu	Câu lệnh DBCC SHRINKDATABASE hay tùy chọn Co lại cơ sở dữ liệu trong SSMS, được truy cập thông qua nút cho cơ sở dữ liệu cụ thể
Co lại tập tin cơ sở dữ liệu	Câu lệnh DBCC SHRINKFILE
Xóa các tập tin nhật ký hoặc dữ liệu	Câu lệnh ALTER DATABASE hoặc các thuộc tính cơ sở dữ liệu trong SSMS
Thêm một nhóm tập tin vào cơ sở dữ liệu	Câu lệnh ALTER DATABASE hoặc các thuộc tính cơ sở dữ liệu trong SSMS
Thay đổi nhóm tập tin mặc định	Câu lệnh ALTER DATABASE
Thay đổi các tùy chọn cơ sở dữ liệu	Câu lệnh ALTER DATABASE hoặc các thuộc tính cơ sở dữ liệu trong SSMS
Thay đổi chủ nhân của cơ sở dữ liệu	Thủ tục lưu trữ hệ thống sp_changedbowner

Bảng 6.4: Các hàm do hệ thống định nghĩa và thủ tục lưu trữ trong SQL Server 2012

Có thể xóa cơ sở dữ liệu do người dùng định nghĩa khi nó không còn cần nữa. Các tập tin và dữ liệu liên quan đến cơ sở dữ liệu này sẽ tự động bị xóa khỏi đĩa khi cơ sở dữ liệu bị xóa.

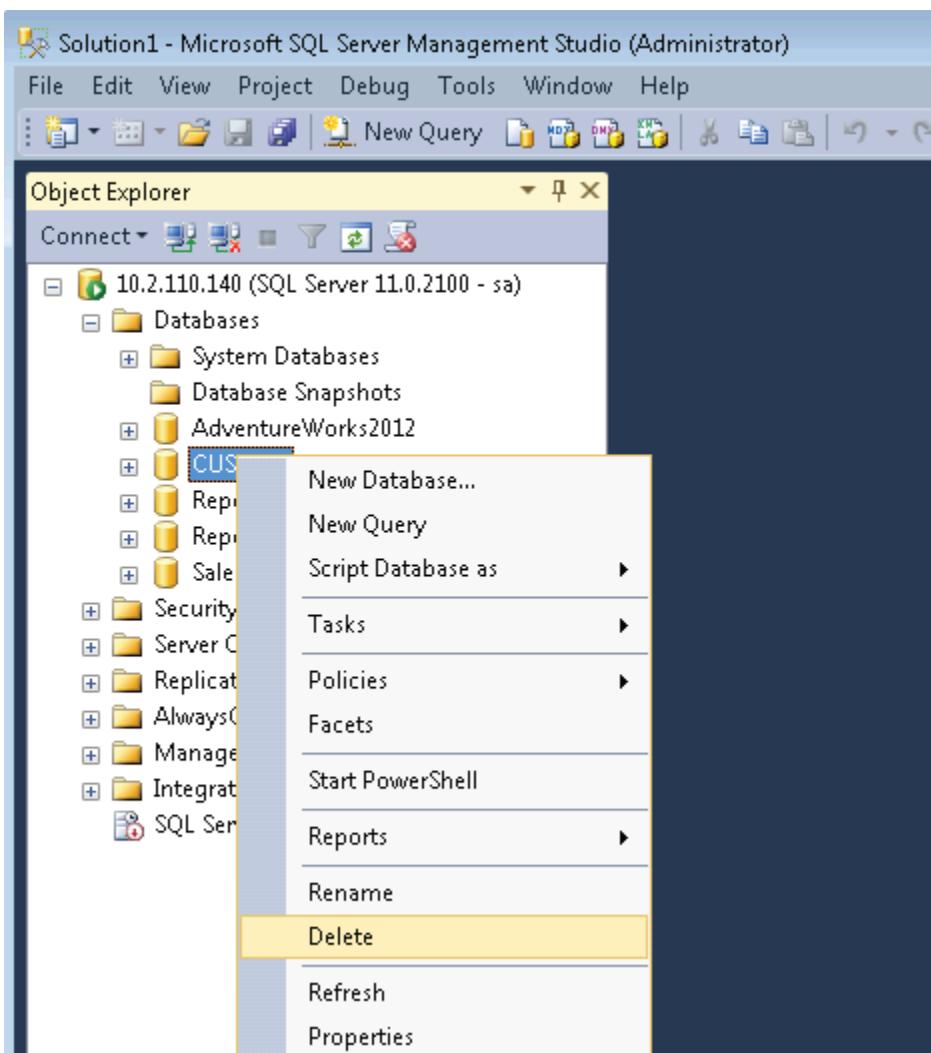
6.4.6 Xóa bỏ thủ tục cơ sở dữ liệu

Bản sao lưu đầy đủ của cơ sở dữ liệu cần phải được thực hiện trước khi xóa bỏ một cơ sở dữ liệu. Chỉ có thể được tái tạo cơ sở dữ liệu đã bị xóa bằng cách khôi phục bản sao lưu.

Những bước để xóa hoặc xóa bỏ một cơ sở dữ liệu sử dụng SSMS như sau:

- Trong **Object Explorer**, kết nối với một thể hiện của SQL Server Database Engine và sau đó, mở rộng thể hiện đó.

2. Mở rộng **Databases**, bấm chuột phải vào cơ sở dữ liệu để xóa, và sau đó, bấm vào **Delete**, như được trình bày trong hình 6.12.



Hình 6.12: Xóa một cơ sở dữ liệu

3. Xác nhận rằng đúng cơ sở dữ liệu được chọn, và sau đó, bấm vào **OK**.

Sau đây là cú pháp để xóa hoặc xóa bỏ một cơ sở dữ liệu sử dụng Transact-SQL.

Cú pháp:

```
CREATE DATABASE database_snapshot_name
ON
(
NAME = logical_file_name,
```

```
FILENAME = 'os_file_name'
) [ ,...n ]
AS SNAPSHOT OF source_database_name
[ ; ]
```

trong đó:

`database_snapshot_name`: là tên của ảnh chụp cơ sở dữ liệu mới.

`ON (NAME = logical_file_name, FILENAME = 'os_file_name') [,... n]`: là danh sách các tập tin trong cơ sở dữ liệu nguồn. Để có ảnh để làm việc, tất cả các tập tin dữ liệu phải được chỉ ra riêng lẻ.

`AS SNAPSHOT OF source_database_name`: là cơ sở dữ liệu đang được tạo ra là một ảnh chụp cơ sở dữ liệu của cơ sở dữ liệu nguồn được chỉ định bằng `source_database_name`.

Đoạn mã 9 tạo ra một ảnh chụp cơ sở dữ liệu trên cơ sở dữ liệu `CUST_DB`.

Đoạn mã 9:

```
CREATE DATABASE customer_snapshot01 ON
( NAME = Customer_DB, FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.
MSSQLSERVER\MSSQL\DATA\Customerdat_0100.ss')
AS SNAPSHOT OF CUST_DB;
```

6.5 Kiểm tra tiến bộ của bạn

1. Điều nào sau đây là cơ sở dữ liệu hệ thống được hỗ trợ bởi SQL Server 2012?

(A)	chính	(C)	msdt
(B)	tepdl	(D)	mod

2. Cơ sở dữ liệu SQL Server được lưu trữ như các tập tin trong _____.

(A)	FAT 16	(C)	Thư mục
(B)	Hệ thống tập tin	(D)	Đĩa cứng

3. Điều nào sau đây là một tiện ích quản trị?

(A)	SQL-SMO	(C)	Transact-SQL
(B)	SQL Server Management Studio	(D)	Thủ tục lưu trữ

4. Cái nào sau đây là các tính năng của cơ sở dữ liệu AdventureWorks2012?

- a. Các dịch vụ tích hợp
- b. Các dịch vụ báo cáo
- c. Các dịch vụ thông báo
- d. Dịch vụ ẩn

(A)	a, b, và c	(C)	a và b
(B)	b	(D)	c và d

5. Cái nào sau đây là cú pháp chính xác để tạo ra một cơ sở dữ liệu?

(A)	<pre>CREATE DATABASE (NAME = 'Customer_DB', FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL11.MSSQLSERVER\MSSQL\ DATA\Customer_DB.mdf')</pre>	(C)	<pre>CREATE DATABASE [Customer_ DB] ON PRIMARY (NAME = 'Customer_DB', FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL11.MSSQLSERVER\MSSQL\ DATA\Customer_DB.mdf')</pre>
(B)	<pre>CREATE Table [Customer_DB] ON PRIMARY (NAME = 'Customer_DB', FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL11.MSSQLSERVER\MSSQL\ DATA\Customer_DB.mdf')</pre>	(D)	Tất cả các câu trên trên

6.5.1 Câu trả lời

1.	A
2.	B
3.	B
4.	A
5.	C



Tóm tắt

- ➔ Cơ sở dữ liệu SQL Server được tạo thành một bộ sưu tập các bảng lưu trữ tập hợp dữ liệu có cấu trúc cụ thể.
- ➔ SQL Server 2012 hỗ trợ ba loại cơ sở dữ liệu:
 - Cơ sở dữ liệu hệ thống
 - Cơ sở dữ liệu do người dùng định nghĩa
 - Cơ sở dữ liệu mẫu
- ➔ SQL Server sử dụng cơ sở dữ liệu hệ thống để hỗ trợ các phần khác nhau của DBMS.
- ➔ Một công ty hư cấu là Adventure Works Cycles được tạo ra như một kịch bản và cơ sở dữ liệu AdventureWorks2012 được thiết kế cho công ty này.
- ➔ Các tập tin dữ liệu SQL Server được sử dụng để lưu trữ các tập tin cơ sở dữ liệu, được chia nhỏ thêm thành các nhóm tập tin vì mục đích hiệu suất.
- ➔ Các đối tượng được gán cho nhóm tập tin mặc định khi chúng được tạo ra trong cơ sở dữ liệu. Nhóm tập tin PRIMARY là nhóm tập tin mặc định.
- ➔ Ánh cơ sở dữ liệu là một khung nhìn tĩnh, chỉ đọc của cơ sở dữ liệu SQL Server.



1. Tạo ra một cơ sở dữ liệu có tên là **UnitedAir** sử dụng câu lệnh Transact-SQL với các thuộc tính sau:
 - Nhóm tập tin chính với các tập tin **UnitedAir1_dat** và **UnitedAir2_dat**. Kích thước, kích thước tối đa, và tỉ lệ tăng tập tin sẽ là 5, 10, và 15% tương ứng.
 - Nhóm tập tin có tên là **UnitedAirGroup1** với các tập tin **UnitedAirGrp1F1** và **UnitedAirGrp1F2**.
 - Nhóm tập tin với **UnitedAirGroup2** với các tập tin **UnitedAirGrp2F1** và **UnitedAirGrp2F2**.

“Khốn thay cho kẻ dạy người
nhanh hơn so với họ có thể học”

Phần - 7

Tạo các bảng

Chào mừng bạn đến với phần **Tạo các bảng**.

Phiên này khám phá các kiểu dữ liệu khác nhau được SQL Server 2012 cung cấp và mô tả cách sử dụng chúng. Những kỹ thuật để tạo, sửa đổi, và loại bỏ các bảng và cột cũng được thảo luận.

Trong phần này, bạn sẽ học để:

- ➔ Liệt kê các kiểu dữ liệu SQL Server 2012
- ➔ Mô tả thủ tục để tạo, chỉnh sửa, và thả bỏ các bảng trong cơ sở dữ liệu SQL Server
- ➔ Mô tả thủ tục để thêm, sửa đổi, và thả bỏ cột trong một bảng

7.1 Giới thiệu

Một trong những loại đối tượng cơ sở dữ liệu quan trọng nhất trong SQL Server 2012 là bảng. Bảng trong SQL Server 2012 chứa dữ liệu dưới dạng các hàng và cột. Mỗi cột có thể có dữ liệu thuộc một loại và kích thước cụ thể.

7.2 Kiểu dữ liệu

Kiểu dữ liệu là một thuộc tính chỉ ra loại dữ liệu một đối tượng có thể giữ, như là dữ liệu số, dữ liệu ký tự, dữ liệu tiền tệ, và vân vân. Kiểu dữ liệu còn chỉ ra dung lượng lưu trữ của một đối tượng. Một khi cột đã được định nghĩa để lưu trữ dữ liệu thuộc về một kiểu dữ liệu đặc biệt, dữ liệu của loại khác không thể được lưu trữ trong đó. Theo cách này, các kiểu dữ liệu bắt buộc sự toàn vẹn dữ liệu

Xem xét một cột tên là **BasicSalary** trong bảng **Employee**. Để đảm bảo rằng chỉ có dữ liệu số được nhập vào, cột này được định nghĩa thuộc về một kiểu dữ liệu số nguyên. Do đó, nếu cố nhập dữ liệu ký tự vào cột **BasicSalary** đó, nó sẽ không thành công.

7.2.1 Các loại kiểu dữ liệu khác nhau

SQL Server 2012 hỗ trợ ba loại dữ liệu:

- **Các kiểu dữ liệu hệ thống** - Chúng được cung cấp bởi SQL Server 2012.
- **Các kiểu dữ liệu bí danh** - Chúng dựa trên các kiểu dữ liệu do hệ thống cung cấp. Một trong những cách sử dụng điển hình của các kiểu dữ liệu bí danh là khi nhiều hơn một bảng lưu trữ cùng một loại dữ liệu trong một cột và có đặc điểm tương tự như là chiều dài, khả năng null, và loại. Trong các trường hợp như vậy, có thể tạo ra loại dữ liệu bí danh để có thể được dùng chung cho các bảng này.
- **Loại do người dùng định nghĩa** - Những loại này được tạo ra bằng cách sử dụng ngôn ngữ lập trình được .NET Framework hỗ trợ, đó là một khuôn khổ phần mềm được Microsoft phát triển.

Bảng 7.1 trình bày các kiểu dữ liệu khác nhau trong SQL Server 2012 cùng với thể loại và mô tả của chúng.

Loại	Loại dữ liệu	Mô tả
Các số chính xác	int	Trình bày một cột chiếm 4 byte không gian bộ nhớ. Thường được dùng để giữ các giá trị số nguyên.
	smallint	Trình bày một cột chiếm 2 byte không gian bộ nhớ. Có thể giữ dữ liệu số nguyên từ -32.768 đến 32.767.
	tinyint	Trình bày một cột chiếm 1 byte không gian bộ nhớ. Có thể giữ dữ liệu số nguyên từ 0 đến 255.

Loại	Loại dữ liệu	Mô tả
Các số chính xác	bigint	Trình bày một cột chiếm 8 byte không gian bộ nhớ. Có thể giữ dữ liệu trong phạm vi -2^{63} (-9.223.372.036.854.775.808) đến $2^{63}-1$ (9.223.372.036.854.775.807)
	numeric	Trình bày một cột thuộc loại này cố định độ chính xác và quy mô.
	money	Trình bày một cột chiếm 8 byte không gian bộ nhớ. Biểu diễn các giá trị dữ liệu tiền tệ dao động từ $(-2^{63}/10000)$ (-922.337.203.685.477,5808) đến $2^{63}-1$ (922.337.203.685.477,5807).
Các số gần đúng	float	Trình bày một cột chiếm 8 byte không gian bộ nhớ. Biểu diễn số có dấu thập phân thay đổi dao động từ $-1.79E+308$ đến $1.79E+308$.
	real	Trình bày một cột chiếm 4 byte không gian bộ nhớ. Biểu diễn số có độ chính xác thay đổi dao động từ $-3.40E+38$ đến $3.40E+38$.
Ngày và giờ	datetime	Biểu diễn ngày và giờ. Được lưu như hai số nguyên 4 byte.
	smalldatetime	Biểu diễn ngày và giờ.
Chuỗi ký tự	char	Lưu trữ dữ liệu ký tự có chiều dài cố định và không phải là Unicode.
	varchar	Lưu dữ liệu ký tự có độ dài biến đổi và không phải Unicode.
	text	Lưu dữ liệu ký tự có độ dài biến đổi và không phải Unicode.
Loại Unicode	nchar	Lưu dữ liệu ký tự Unicode với độ dài cố định.
	nvarchar	Lưu dữ liệu ký tự Unicode có độ dài biến đổi.
Các loại dữ liệu khác	Timestamp	Trình bày một cột chiếm 8 byte không gian bộ nhớ. Có thể giữ tự động các số nhị phân duy nhất, được tạo ra mà được tạo ra cho một cơ sở dữ liệu.
	binary(n)	Lưu trữ dữ liệu nhị phân có chiều dài cố định với chiều dài tối đa là 8000 byte.

Loại	Loại dữ liệu	Mô tả
Các loại dữ liệu khác	varbinary(n)	Lưu dữ liệu nhị phân có độ dài biến đổi với độ dài tối đa 8000 byte.
	image	Lưu dữ liệu nhị phân với độ dài biến đổi với độ dài tối đa là $2^{30}-1$ (1.073.741.823) byte.
	uniqueidentifier	Trình bày một cột chiếm 16 byte không gian bộ nhớ. Ngoài ra, lưu trữ mã định danh duy nhất toàn cầu (GUID).

Bảng 7.1: Các loại dữ liệu trong SQL Server 2012

Các loại dữ liệu bí danh có thể được tạo ra bằng cách sử dụng câu lệnh CREATE TYPE.

Cú pháp cho câu lệnh CREATE TYPE như sau:

Cú pháp:

```
CREATE TYPE [ schema_name.] type_name { FROM base_type [ (
precision [, scale] ) ] [NULL|NOT NULL] } [ ; ]
```

trong đó:

schema_name: chỉ ra tên của sơ đồ trong đó loại dữ liệu bí danh đang được tạo ra.

type_name: chỉ ra tên của loại bí danh đang được tạo ra.

base_type: chỉ ra tên của loại dữ liệu do hệ thống định nghĩa dựa trên đó loại dữ liệu bí danh đang được tạo ra.

precision và scale: chỉ ra độ chính xác và tỉ lệ cho dữ liệu số.

NULL|NOT NULL: chỉ ra xem loại dữ liệu có thể giữ giá trị null hay không.

Đoạn mã 1 trình bày cách để tạo ra một loại dữ liệu bí danh có tên là **usertype** sử dụng câu lệnh CREATE TYPE.

Đoạn mã 1:

```
CREATE TYPE usertype FROM varchar(20) NOT NULL
```

7.3 Tạo, sửa đổi, và thả bỏ các bảng

Hầu hết các bảng có một khóa chính, tạo thành từ một hoặc nhiều cột của bảng. Khóa chính luôn luôn là duy nhất. Database Engine sẽ thực thi hạn chế mà bất kỳ giá trị khóa chính nào không thể được lặp lại trong bảng. Do đó, khóa chính có thể được sử dụng để nhận dạng mỗi bản ghi một cách duy nhất.

7.3.1 Tạo ra các bảng

Câu lệnh CREATE TABLE được sử dụng để tạo ra các bảng trong SQL Server 2012. Cú pháp cho câu lệnh CREATE TABLE như sau:

Cú pháp:

```
CREATE TABLE [database_name].[schema_name].| schema_name.]table_name
    ([<column_name>] [data_type] Null/Not Null, )
    ON [filegroup | "default"]
GO
```

trong đó:

database_name: là tên của cơ sở dữ liệu trong đó bảng được tạo ra.

table_name: là tên của bảng mới. Snnbtable_nameSnnbcó thể có tối đa 128 ký tự.

column_name: là tên của một cột trong bảng. column_name có thể lên đến 128 ký tự. column_name không được chỉ định cho các cột được tạo ra với kiểu dữ liệu timestamp . Tên cột mặc định của cột timestamp là timestamp.

data_type: Nó chỉ ra kiểu dữ liệu của cột.

Đoạn mã 2 trình bày việc tạo ra một bảng có tên là `dbo.Customer_1`.

Đoạn mã 2:

```
CREATE TABLE [dbo].[Customer_1] (
    [Customer_id] number [numeric](10, 0) NOT NULL,
    [Customer_name] [varchar](50) NOT NULL
)
ON [PRIMARY]
GO
```

Một vài phần tiếp theo xem qua các đặc điểm khác nhau liên quan với các bảng như là tính chất không xác định của cột, định nghĩa mặc định, ràng buộc, và vân vân.

7.3.2 Sửa đổi các bảng

Câu lệnh ALTER TABLE được sử dụng để sửa đổi một định nghĩa bảng bằng cách thay đổi, thêm, hoặc thả bỏ các cột và ràng buộc, gán lại các phân vùng, hoặc vô hiệu hóa hoặc cho phép các ràng buộc và bộ kích khởi.

Cú pháp cho câu lệnh ALTER TABLE như sau:

Cú pháp:

```
ALTER TABLE [ [database_name. [schema_name].] schema_name.]table_name
    ALTER COLUMN ([<column_name>] [data_type] Null/Not Null,) ;
    | ADD ([<column_name>] [data_type] Null/Not Null,) ;
    | DROP COLUMN ([<column_name>]) ;
```

trong đó:

ALTER COLUMN: chỉ ra rằng cột cụ thể sẽ được thay đổi hoặc sửa đổi.

ADD: chỉ ra rằng một hoặc nhiều định nghĩa cột sẽ được thêm vào.

DROP COLUMN ([<column_name>]): chỉ ra rằng column_name sẽ được loại bỏ khỏi bảng.

Đoạn mã 3 trình bày việc thay đổi cột **Customer_id**.

Đoạn mã 3:

```
USE [CUST_DB]
ALTER TABLE [dbo].[Customer_1]
ALTER Column [Customer_id] numeric(12, 0) NOT NULL;
```

Đoạn mã 4 trình bày việc thêm cột **Contact_number**.

Đoạn mã 4:

```
USE [CUST_DB]
ALTER TABLE [dbo].[Table_1]
ADD [Contact_number] numeric(12, 0) NOT NULL;
```

Đoạn mã 5 trình bày việc thả rơi cột **Contact_number**.

Đoạn mã 5:

```
USE [CUST_DB]
ALTER TABLE [dbo].[Table_1]
DROP COLUMN [Contact_name];
```

Tuy nhiên, trước khi thử xóa các cột, quan trọng là bảo đảm các cột có thể xóa được. Trong điều kiện nhất định, không thể thả rơi các cột, chẳng hạn như khi chúng được sử dụng trong ràng buộc CHECK, FOREIGN KEY, UNIQUE, hoặc PRIMARY KEY, kết hợp với định nghĩa DEFAULT, và vân vân.

7.3.3 Thả rơi các bảng

Câu lệnh `DROP TABLE` loại bỏ một định nghĩa bảng, dữ liệu của nó, và tất cả các đối tượng liên quan như là các chỉ số, trigger, ràng buộc, và thông số kỹ thuật cho phép đổi với bảng đó.

Cú pháp cho câu lệnh `DROP TABLE` như sau:

Cú pháp:

```
DROP TABLE <Table_Name>
```

trong đó:

`<Table_Name>`: là tên của bảng sẽ được thả rơi.

Đoạn mã 6 trình bày cách để thả rơi một bảng.

Đoạn mã 6:

```
USE [CUST_DB]
DROP TABLE [dbo].[Table_1]
```

7.3.4 Các câu lệnh sửa đổi dữ liệu

Câu lệnh được sử dụng để sửa đổi dữ liệu là các câu lệnh `INSERT`, `UPDATE`, và `DELETE`. Điều này được giải thích như sau:

- ➔ **Câu lệnh `INSERT`** - Câu lệnh `INSERT` thêm một hàng mới vào bảng. Cú pháp cho câu lệnh `INSERT` như sau:

Cú pháp:

```
INSERT [INTO] <Table_Name>
VALUES <values>
```

trong đó:

`<Table_Name>`: là tên của bảng trong đó hàng sẽ được chèn vào.

`[INTO]`: là một từ khóa tùy chọn được sử dụng giữa `INSERT` và bảng mục tiêu.

`<values>`: chỉ ra những giá trị cho các cột của bảng.

Đoạn mã 7 trình bày việc thêm một hàng mới vào bảng **Table_2**.

Đoạn mã 7:

```
USE [CUST_DB]
INSERT INTO [dbo].[Table_2] VALUES (101, 'Richard Parker', 'Richy')
GO
```

Kết quả của điều này sẽ là một hàng với dữ liệu đã cho được đưa vào bảng.

- **Câu lệnh UPDATE** - Câu lệnh UPDATE sửa đổi dữ liệu trong bảng. Cú pháp cho câu lệnh UPDATE như sau:

Cú pháp:

```
UPDATE <Table_Name>
SET <Column_Name = Value>
[WHERE <Search condition>]
```

trong đó:

<Table_Name>: là tên của bảng trong đó các bản ghi sẽ được cập nhật.

<Column_Name>: là tên của cột trong bảng trong đó bản ghi sẽ được cập nhật.

<Value>: chỉ ra giá trị mới cho cột đã sửa đổi.

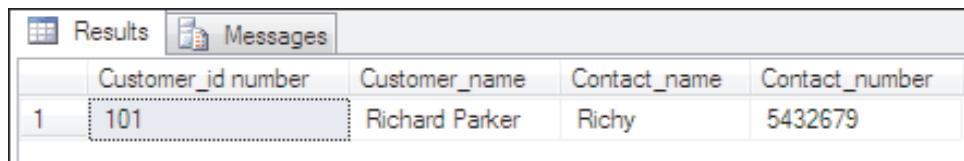
<Search condition>: chỉ ra điều kiện được đáp ứng cho các hàng bị xóa.

Đoạn mã 8 trình bày việc sử dụng câu lệnh UPDATE để sửa đổi giá trị trong cột Contact_number.

Đoạn mã 8:

```
USE [CUST_DB]
UPDATE [dbo].[Table_2] SET Contact_number = 5432679 WHERE Contact_name LIKE
'Richy'
GO
```

Hình 7.1 trình bày kết quả của câu lệnh UPDATE.



	Customer_id number	Customer_name	Contact_name	Contact_number
1	101	Richard Parker	Richy	5432679

Hình 7.1: Kết quả của câu lệnh UPDATE

- Câu lệnh **DELETE** - Câu lệnh **DELETE** loại bỏ các hàng khỏi bảng. Cú pháp cho câu lệnh **DELETE** như sau:

Cú pháp:

```
DELETE FROM <Table_Name>
[WHERE <Search condition>]
```

trong đó:

<**Table_Name**>: là tên của bảng mà từ đó các bản ghi sẽ được xóa.

Mệnh đề **WHERE** được sử dụng để chỉ ra tình trạng này. Nếu mệnh đề **WHERE** không được đưa vào trong câu lệnh **DELETE**, tất cả các bản ghi trong bảng sẽ bị xóa.

Đoạn mã 9 trình bày cách xóa một hàng khỏi bảng **Customer_2** nơi giá trị **Contact_number** bằng 5.432.679.

Đoạn mã 9:

```
USE [CUST_DB]
DELETE FROM [dbo].[Customer_2] WHERE Contact_number = 5432679
GO
```

7.3.5 Cột khả năng null

Đặc điểm tính chất không xác định của một cột xác định xem các hàng trong bảng có thể chứa giá trị null cho cột đó hay không. Trong SQL Server, giá trị null không giống như bằng không, trống, hoặc một chuỗi ký tự có độ dài bằng không (chẳng hạn như ' '). Ví dụ, giá trị null trong cột **Color** của bảng **Production.Product** của cơ sở dữ liệu AdventureWorks2012 không có nghĩa là sản phẩm không có màu, nó chỉ có nghĩa là màu của sản phẩm là không rõ hoặc chưa được đặt.

Có thể định nghĩa tính chất không xác định khi tạo ra bảng hoặc khi sửa đổi bảng.

Từ khóa **NULL** được sử dụng để chỉ ra rằng các giá trị null được cho phép trong cột, và từ khóa **NOT NULL** được sử dụng để chỉ ra rằng các giá trị null không được phép.

Khi chèn một hàng, nếu không có giá trị được đưa ra cho một cột nullable (có nghĩa là, nó cho phép các giá trị null), khi đó, SQL Server tự động cung cấp cho nó một giá trị null trừ khi cột đã được đưa ra một định nghĩa mặc định. Cũng có thể nhập riêng một giá trị null vào trong một cột bất kể nó thuộc loại dữ liệu nào hoặc là nó có giá trị mặc định nào được liên kết với nó. Việc làm cho một cột thành không thể không xác định (đó là không cho phép các giá trị null) làm tăng cường tính toàn vẹn dữ liệu bằng cách bảo đảm là cột có chứa dữ liệu trong mọi hàng.

Trong Đoạn mã 10, câu lệnh CREATE TABLE sử dụng từ khóa NULL và NOT NULL với định nghĩa cột.

Đoạn mã 10:

```
USE [CUST_DB]
CREATE TABLE StoreDetails ( StoreID int NOT NULL, Name varchar(40) NULL)
GO
```

Kết quả của mã là bảng **StoreDetails** được tạo ra với cột **StoreID** và **Name** bổ sung vào bảng này.

7.3.6 Định nghĩa DEFAULT

Xem xét kịch bản trong đó các chi tiết về sản phẩm cần được lưu trong bảng của SQL Server 2012 nhưng tất cả các giá trị cho chi tiết sản phẩm có thể không rõ thậm chí vào thời điểm chèn vào dữ liệu. Tuy nhiên, theo các quy tắc tính toàn vẹn và sự nhất quán dữ liệu, các cột trong một bản ghi thường chứa một giá trị. Việc lưu các giá trị null vào trong các cột như vậy khi giá trị chính xác của dữ liệu không rõ có thể là không thích hợp hoặc không thực tế.

Trong các tình huống như vậy, định nghĩa **DEFAULT** có thể được đưa ra cho cột để gán nó làm giá trị mặc định nếu không có giá trị nào được đưa ra vào thời điểm tạo ra. Ví dụ, người ta thường chỉ ra số không làm giá trị mặc định cho các cột số hoặc 'N/A' hoặc 'Unknown' làm giá trị mặc định cho các cột chuỗi khi không có giá trị nào được chỉ ra.

Định nghĩa **DEFAULT** cho một cột có thể được tạo ra vào thời điểm tạo bảng hoặc thêm ở giai đoạn sau vào bảng hiện có. Khi định nghĩa **DEFAULT** được thêm vào một cột hiện có trong bảng, SQL Server áp dụng các giá trị mặc định mới chỉ cho những dòng dữ liệu đó, đã mới được thêm vào bảng.

Trong Đoạn mã 11, câu lệnh CREATE TABLE sử dụng từ khóa **DEFAULT** để định nghĩa giá trị mặc định cho **Price**.

Đoạn mã 11:

```
USE [CUST_DB]
CREATE TABLE StoreProduct ( ProductID int NOT NULL, Name varchar(40) NOT NULL,
Price money NOT NULL DEFAULT (100))
GO
```

Khi một hàng được đưa vào sử dụng câu lệnh như trong Đoạn mã 12, giá trị của **Price** sẽ không được để trống, nó sẽ có giá trị 100.00 mặc dù người dùng đã không nhập bất kỳ giá trị nào cho cột đó.

Đoạn mã 12:

```
USE [CUST_DB]
INSERT INTO dbo.StoreProduct (ProductID, Name) VALUES (111, 'Rivets')
GO
```

Hình 7.2 trình bày kết quả của Đoạn mã 12, nơi mặc dù các giá trị được thêm vào chỉ cho cột **ProductID** và **Name**, cột **Price** vẫn sẽ hiển thị giá trị 100.00. Điều này là do định nghĩa **DEFAULT**.

ProductID	Name	Price
1	Rivets	100.00

Hình 7.2: Các giá trị được thêm vào cột ProductID và Name

Không thể tạo ra phần sau đây trên các cột với định nghĩa **DEFAULT**:

- ➔ Loại dữ liệu timestamp
- ➔ Thuộc tính **IDENTITY** hoặc **ROWGUIDCOL**
- ➔ Một định nghĩa mặc định hiện hữu hoặc đối tượng mặc định

7.3.7 Thuộc tính **IDENTITY**

Thuộc tính **IDENTITY** của SQL Server được sử dụng để tạo ra các cột mã định danh có thể chứa các giá trị liên tục tự động tạo ra để xác định duy nhất mỗi hàng trong một bảng. Ví dụ, một cột định danh có thể được tạo ra để tạo ra các số đăng ký học viên duy nhất một cách tự động bất cứ khi nào các hàng mới được đưa vào bảng **Students**. Số nhận dạng cho hàng đầu tiên được chèn vào bảng được gọi là giá trị hạt giống (seed) và lượng tăng, còn được gọi là thuộc tính Lượng tăng đồng nhất, được thêm vào hạt giống để tạo ra thêm các số nhận dạng theo tuần tự. Khi một hàng mới được chèn vào một bảng có một cột nhận dạng, giá trị nhận dạng tiếp theo được SQL Server tạo ra tự động bằng cách thêm lượng tăng vào hạt giống. Cột nhận dạng thường được dùng cho các giá trị khóa chính.

Các đặc tính của thuộc tính **IDENTITY** như sau:

- ➔ Một cột có thuộc tính **IDENTITY** phải được định nghĩa bằng cách sử dụng một trong những kiểu dữ liệu sau: **decimal**, **int**, **numeric**, **smallint**, **bigint**, hoặc **tinyint**.
- ➔ Một cột có thuộc tính **IDENTITY** không cần phải có mầm và giá trị tăng dần được chỉ ra. Nếu chúng không được chỉ ra, giá trị mặc định bằng 1 sẽ được dùng cho cả hai.
- ➔ Một bảng không thể có nhiều hơn một cột với thuộc tính **IDENTITY**.
- ➔ Cột mã định danh trong một bảng phải không cho phép giá trị null và không được chứa định nghĩa hoặc đối tượng **DEFAULT**.

- ➔ Cột được định nghĩa với thuộc tính **IDENTITY** không thể có các giá trị của chúng được cập nhật.
- ➔ Những giá trị này có thể được chèn vào một cách rõ ràng vào cột nhận dạng của một bảng chỉ khi **IDENTITY _ INSERT** được đặt là **ON**. Khi **IDENTITY _ INSERT** là **ON**, các câu lệnh **INSERT** phải cung cấp một giá trị.

Ưu điểm của các cột nhận dạng là SQL Server có thể các cột tự động các giá trị khóa, như vậy sẽ làm giảm chi phí và cải thiện hiệu suất. Việc sử dụng các cột nhận dạng làm đơn giản hóa việc lập trình và giữ cho các giá trị khóa chính ngắn gọn.

Một khi thuộc tính **IDENTITY** đã được đặt, lấy giá trị của cột mã định danh có thể được thực hiện bằng cách sử dụng từ khóa **IDENTITYCOL** với tên bảng trong câu lệnh **SELECT**. Để biết xem một bảng có một cột **IDENTITY** hay không, hàm **OBJECTPROPERTY()** có thể được sử dụng. Để lấy ra tên của cột **IDENTITY** trong một bảng, hàm **COLUMNPROPERTY** được sử dụng.

Cú pháp cho thuộc tính **IDENTITY** như sau:

Cú pháp:

```
CREATE TABLE <table_name> (column_name data_type [ IDENTITY
[ (seed_value, increment_value) ] ] NOT NULL )
```

trong đó:

seed_value: là giá trị mầm từ đó để bắt đầu tạo ra giá trị nhận dạng.

increment_value: là giá trị tăng dần theo đó để tăng mỗi lần.

Đoạn mã 13 trình bày việc sử dụng thuộc tính **IDENTITY**. **HRContactPhone** được tạo ra như một bảng với hai cột trong lược đồ **Person** có sẵn trong cơ sở dữ liệu **CUST_DB**. Cột **Person_ID** là một cột nhận dạng. Giá trị hạt giống là 500, và giá trị gia tăng là 1.

Đoạn mã 13:

```
USE [CUST_DB]
GO
CREATE TABLE HRContactPhone ( Person_ID int IDENTITY(500,1) NOT NULL,
MobileNumber bigint NOT NULL )
GO
```

Trong khi chèn các hàng vào bảng, nếu `IDENTITY_INSERT` không được bật, khi đó, không thể cho các giá trị rõ ràng cho cột `IDENTITY`. Thay vào đó, có thể cho các câu lệnh tương tự như Đoạn mã 14.

Đoạn mã 14:

```
USE [CUST_DB]
INSERT INTO HRContactPhone (MobileNumber) VALUES (983452201)
INSERT INTO HRContactPhone (MobileNumber) VALUES (993026654)
GO
```

Hình 7.3 trình bày kết quả trong đó thuộc tính `IDENTITY` làm tăng các giá trị cột `Person_ID`.

	Person_ID	MobileNumber
1	500	983452201
2	501	993026654

Hình 7.3: Thuộc tính `IDENTITY` được áp dụng trên cột `Person_ID`

7.3.8 Mã định danh duy nhất toàn cầu

Ngoài thuộc tính `IDENTITY`, SQL Server còn hỗ trợ các mã định danh duy nhất toàn cầu. Thông thường, trong một môi trường được nối mạng, nhiều bảng có thể cần có một cột chứa giá trị đơn nhất tổng thể chung. Xem xét kịch bản khi dữ liệu từ nhiều hệ cơ sở dữ liệu như các cơ sở dữ liệu ngân hàng phải được hợp nhất ở chỉ một vị trí. Khi dữ liệu từ khắp thế giới được đổi chiếu ở địa điểm trung tâm về sự hợp nhất và lập báo cáo, sử dụng các giá trị đơn nhất tổng thể để tránh khách hàng ở các quốc gia khác nhau không có cùng số tài khoản ngân hàng hoặc mã ID khách hàng. Để thỏa mãn nhu cầu này, SQL Server cung cấp các cột mã nhận dạng đơn nhất tổng thể. Có thể tạo ra các cột này cho từng bảng chứa các giá trị đơn nhất trên tất cả các máy tính trong một mạng. Chỉ có thể tạo ra một cột mã nhận dạng và một cột mã nhận dạng đơn nhất tổng thể cho mỗi bảng. Để tạo và làm việc với các mã định danh duy nhất toàn cầu, một sự kết hợp của `ROWGUIDCOL`, kiểu dữ liệu `uniqueidentifier`, và hàm `NEWID` được sử dụng.

Các giá trị cho cột đơn nhất tổng thể không được tạo ra tự động. Người ta phải tạo ra một định nghĩa `DEFAULT` với hàm `NEWID()` cho cột `uniqueidentifier` để tạo ra một giá trị duy nhất toàn cầu. Hàm `NEWID()` tạo ra một số định danh duy nhất là một chuỗi nhị phân 16-byte. Cột có thể được tham chiếu trong danh sách `SELECT` bằng cách sử dụng từ khóa `ROWGUIDCOL`.

Để biết xem một bảng có cột `ROWGUIDCOL` hay không, hàm `OBJECTPROPERTY` được sử dụng. Hàm `COLUMNPROPERTY` được sử dụng để lấy tên của cột `ROWGUIDCOL`. Đoạn mã 15 trình bày cách câu lệnh `CREATE TABLE` tạo ra bảng `EMPCellularPhone`.

Cột `Person_ID` tự động tạo ra một `GUID` cho mỗi hàng mới được thêm vào bảng.

Đoạn mã 15:

```
USE [CUST_DB]
CREATE TABLE EMP_CellularPhone ( Person_ID uniqueidentifier DEFAULT NEWID() NOT NULL, PersonName varchar(60) NOT NULL)
GO
```

Đoạn mã 16 thêm một giá trị cho cột **PersonName**.

Đoạn mã 16:

```
USE [CUST_DB]
INSERT INTO EMP_CellularPhone (PersonName) VALUES ('William Smith')
SELECT * FROM EMP_CellularPhone
GO
```

Hình 7.4 trình bày kết quả trong đó mã định danh duy nhất được hiển thị với một **PersonName** cụ thể.

Results		Messages
	Person_ID	PersonName
1	362C4377-D194-4607-A466-7FF02064EAFC	William Smith

Hình 7.4: Mã định danh duy nhất

7.4 Các ràng buộc

Một trong các chức năng quan trọng của SQL Server là duy trì và tăng cường tính toàn vẹn dữ liệu. Có một số phương tiện để đạt được điều này nhưng một trong các phương pháp được dùng phổ biến và được ưa chuộng là sử dụng các ràng buộc. Ràng buộc là một thuộc tính được gán cho một cột hoặc tập hợp cột trong một bảng để ngăn một số loại giá trị dữ liệu không nhất quán không được nhập vào. Các ràng buộc được dùng để áp dụng các quy tắc logic trong kinh doanh và tăng cường tính toàn vẹn dữ liệu.

Các ràng buộc có thể được tạo ra khi bảng được tạo ra, như là một phần của định nghĩa bảng bằng cách sử dụng câu lệnh **CREATE TABLE** hoặc có thể được thêm vào ở giai đoạn sau bằng cách sử dụng câu lệnh **ALTER TABLE**. Có thể phân loại các ràng buộc thành ràng buộc cột và ràng buộc bảng. Ràng buộc cột được chỉ ra như một phần của định nghĩa cột và chỉ áp dụng cho cột đó. Ràng buộc bảng có thể áp dụng cho nhiều hơn một cột trong bảng và được khai báo độc lập khỏi định nghĩa cột. Ràng buộc bảng phải được dùng khi có nhiều hơn một cột được đưa vào trong một ràng buộc.

SQL Server hỗ trợ các loại ràng buộc sau đây:

- ➔ PRIMARY KEY
- ➔ UNIQUE

- ➔ FOREIGN KEY
- ➔ CHECK
- ➔ NOT NULL

7.4.1 PRIMARY KEY

Bảng thông thường có một khóa chính bao gồm một cột đơn hoặc tổ hợp các cột để nhận biết duy nhất từng hàng ở trong bảng. Ràng buộc PRIMARY KEY được sử dụng để tạo ra một khóa chính và thực thi tính toàn vẹn của thực thể của bảng. Chỉ có thể tạo ra một ràng buộc khóa chính cho một bảng. Hai hàng trong một bảng không thể có cùng một giá trị khóa chính và một cột là khóa chính không thể có các giá trị NULL. Do đó, khi ràng buộc khóa chính được thêm vào các cột hiện hữu của một bảng, SQL Server 2012 sẽ kiểm tra xem các quy tắc cho khóa chính có được tuân thủ hay không. Nếu dữ liệu hiện có trong các cột không tuân thủ các quy tắc cho khóa chính, khi đó ràng buộc sẽ không được thêm vào.

Cú pháp để thêm một khóa chính trong khi tạo ra một bảng như sau:

Cú pháp:

```
CREATE TABLE <table_name> (Column_name datatype PRIMARY KEY [  
column_list])
```

Đoạn mã 17 trình bày cách tạo ra bảng **EMPContactPhone** để lưu trữ các chi tiết điện thoại liên lạc của một người. Do cột **EMP_ID** phải là khóa chính để xác định mỗi hàng duy nhất, nó được tạo ra với ràng buộc khóa chính.

Đoạn mã 17:

```
USE [CUST_DB]  
  
CREATE TABLE EMPContactPhone (EMP_ID int PRIMARY KEY, MobileNumber bigint,  
ServiceProvider varchar(30), LandlineNumber bigint)  
  
GO
```

Một phương pháp khác là sử dụng từ khóa CONSTRAINT. Cú pháp như sau:

Cú pháp:

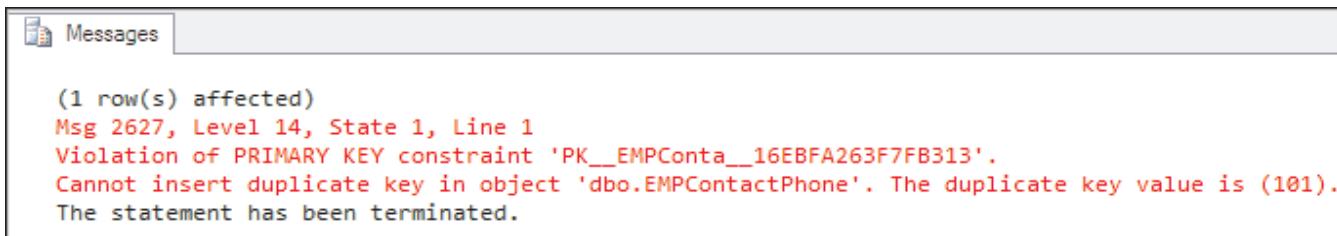
```
CREATE TABLE <table_name> (<column_name><datatype> [, column_list] CONSTRAINT  
constraint_name PRIMARY KEY)
```

Sau khi đã tạo ra một khóa chính cho **EMP_ID**, truy vấn được viết để chèn các hàng vào bảng với các câu lệnh được trình bày trong Đoạn mã 18.

Đoạn mã 18:

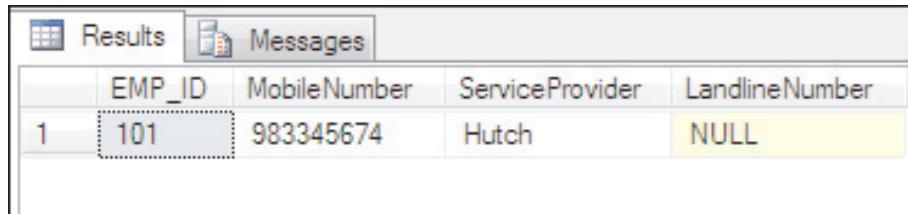
```
USE [CUST_DB]
INSERT INTO dbo.EMPContactPhone values (101, 983345674, 'Hutch', NULL)
INSERT INTO dbo.EMPContactPhone values (102, 989010002, 'Airtel', NULL)
GO
```

Câu lệnh đầu tiên được trình bày trong Đoạn mã 18 được thực hiện thành công nhưng câu lệnh `INSERT` tiếp theo sẽ thất bại vì giá trị cho `EMP_ID` là trùng lặp như được trình bày trong hình 7.5.



Hình 7.5: Thông báo lỗi cho đầu ra về EMP_ID trùng lặp

Đầu ra của Đoạn mã 18 được trình bày trong hình 7.6.



	EMP_ID	MobileNumber	ServiceProvider	LandlineNumber
1	101	983345674	Hutch	NULL

Hình 7.6: Kết quả của câu lệnh đầu tiên được thực thi

7.4.2 UNIQUE

Ràng buộc `UNIQUE` được sử dụng để đảm bảo rằng chỉ những giá trị duy nhất được nhập vào trong một cột hay tập hợp các cột. Nó cho phép người phát triển bảo đảm là không có giá trị trùng lặp được nhập vào. Các khóa chính là hoàn toàn duy nhất. Các ràng buộc khóa duy nhất bắt buộc toàn vẹn thực thể bởi vì một khi những ràng buộc này được áp dụng, không có hai hàng trong bảng có thể có cùng một giá trị cho những cột đó. Ràng buộc `UNIQUE` cho phép các giá trị null.

Bảng đơn lẻ có thể có nhiều hơn một ràng buộc `UNIQUE`.

Cú pháp để tạo ra ràng buộc `UNIQUE` như sau:

Cú pháp:

```
CREATE TABLE <table_name> ([column_list
] <column_name><data_type> UNIQUE [
column_list])
```

Đoạn mã 19 trình bày cách làm cho các cột **MobileNumber** và **LandlineNumber** là duy nhất.

Đoạn mã 19:

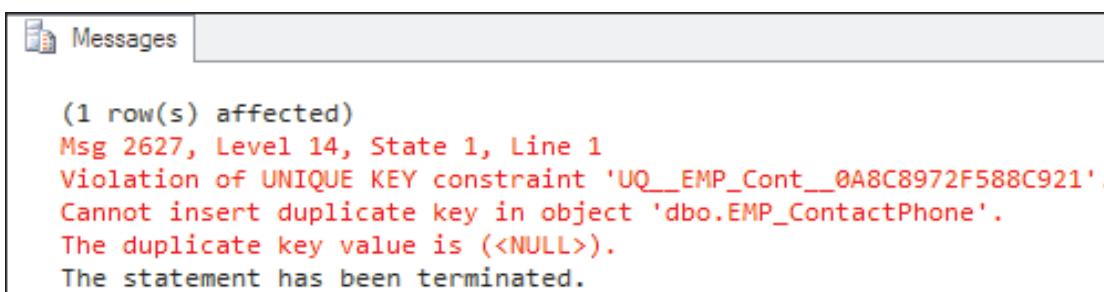
```
USE [CUST_DB]
GO
CREATE TABLE EMP_ContactPhone (Person_ID int PRIMARY KEY, MobileNumber bigint
UNIQUE, ServiceProvider varchar(30), LandlineNumber bigint UNIQUE)
```

Đoạn mã 20 trình bày cách chèn một hàng vào bảng.

Đoạn mã 20:

```
USE [CUST_DB]
INSERT INTO EMP_ContactPhone values (111, 983345674, 'Hutch', NULL)
INSERT INTO EMP_ContactPhone values (112, 983345674, 'Airtel', NULL)
GO
```

Mặc dù giá trị **NULL** đã được đưa ra cho các cột **LandlineNumber**, được định nghĩa là **UNIQUE**, lệnh sẽ thực hiện thành công bởi vì các ràng buộc **UNIQUE** chỉ kiểm tra tính duy nhất của các giá trị nhưng không ngăn chặn các mục nhập null. Câu lệnh đầu tiên được trình bày trong Đoạn mã 20 được thực hiện thành công nhưng câu lệnh **INSERT** tiếp theo sẽ thất bại thậm chí giá trị khóa chính có khác vì giá trị cho **MobileNumber** là trùng lặp như được trình bày trong hình 7.7. Điều này là do cột **MobileNumber** được định nghĩa là duy nhất và không cho phép các giá trị trùng lặp.



Hình 7.7: Thông báo lỗi đầu ra cho MobileNumber có giá trị trùng lặp

Đầu ra của Đoạn mã 20 được trình bày trong hình 7.8.

	Person_ID	MobileNumber	ServiceProvider	LandlineNumber
1	111	983345674	Hutch	NULL

Hình 7.8: Kết quả của ràng buộc **UNIQUE** được thực thi

7.4.3 FOREIGN KEY

Khóa ngoại trong bảng là một cột trỏ tới cột khóa chính trong bảng khác. Các ràng buộc khóa ngoại được dùng để tăng cường tính toàn vẹn tham chiếu. Cú pháp cho khóa ngoại như sau:

Cú pháp:

```
CREATE TABLE <table_name1>([column_list,] <column_name><datatype> FOREIGN KEY
REFERENCES <table_name> (pk_column_name) [,column_list])
```

trong đó:

<table_name>: là tên của bảng nơi tham khảo khóa chính từ đó.

<pk_column_name>: là tên của cột khóa chính.

Đoạn mã 21 trình bày cách để tạo ra ràng buộc khóa ngoại.

Đoạn mã 21:

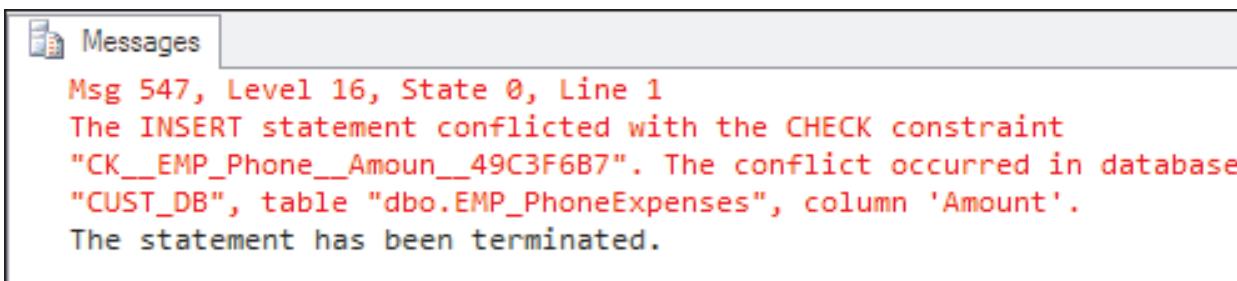
```
USE [CUST_DB]
GO
CREATE TABLE EMP_PhoneExpenses (Expense_ID int PRIMARY KEY, MobileNumber bigint
FOREIGN KEY REFERENCES EMP_ContactPhone (MobileNumber), Amount bigint)
```

Một hàng được chèn vào bảng để số điện thoại di động cũng giống như một trong những số điện thoại di động trong **EMP_ContactPhone**. Lệnh này sẽ được viết được trình bày trong Đoạn mã 22.

Đoạn mã 22:

```
INSERT INTO dbo.EMP_PhoneExpenses values(101, 993026654, 500)
SELECT * FROM dbo.EMP_PhoneExpenses
```

Thông báo lỗi của Đoạn mã 22 được trình bày trong hình 7.9.



Hình 7.9: Thông báo lỗi đầu ra của FOREIGN KEY REFERENCES

Nếu không có khóa trong bảng tham chiếu có giá trị đang được đưa vào khóa ngoại, việc chèn vào sẽ thất bại như được trình bày trong hình 7.9. Tuy nhiên, có thể thêm giá trị **NULL** vào một cột khóa ngoại.

7.4.4 CHECK

Ràng buộc CHECK hạn chế các giá trị có thể được đặt trong một cột. Các ràng buộc kiểm tra tăng cường tính toàn vẹn dữ liệu. Ví dụ, ràng buộc CHECK có thể được đưa ra để kiểm tra xem giá trị đang được nhập vào **voterAge** có lớn hơn hoặc bằng 18 hay không. Nếu dữ liệu đang được nhập cho cột không thỏa mãn điều kiện này, khi đó, việc chèn vào sẽ thất bại.

Ràng buộc CHECK hoạt động bằng cách chỉ ra một điều kiện tìm kiếm, có thể đánh giá thành TRUE, FALSE, hoặc không rõ. Các giá trị đánh giá thành FALSE bị từ chối. Nhiều ràng buộc CHECK có thể được chỉ ra cho một cột đơn lẻ. Ràng buộc CHECK đơn lẻ cũng có thể được áp dụng cho nhiều cột bằng cách tạo ra nó ở mức bảng.

Đoạn mã 23 trình bày việc tạo ra một ràng buộc CHECK để đảm bảo rằng giá trị **Amount** sẽ luôn luôn khác không. Tuy nhiên, giá trị NULL có thể được thêm vào cột **Amount** nếu giá trị của **Amount** là không biết.

Đoạn mã 23:

```
USE [CUST_DB]

CREATE TABLE EMP_PhoneExpenses (Expense_ID int PRIMARY KEY, MobileNumber bigint
FOREIGN KEY REFERENCES EMP_ContactPhone (MobileNumber) , Amount bigint CHECK
(Amount >10) )

GO
```

Một khi ràng buộc CHECK đã được định nghĩa, nếu câu lệnh INSERT được viết với dữ liệu vi phạm ràng buộc này, nó sẽ thất bại như được trình bày trong Đoạn mã 24.

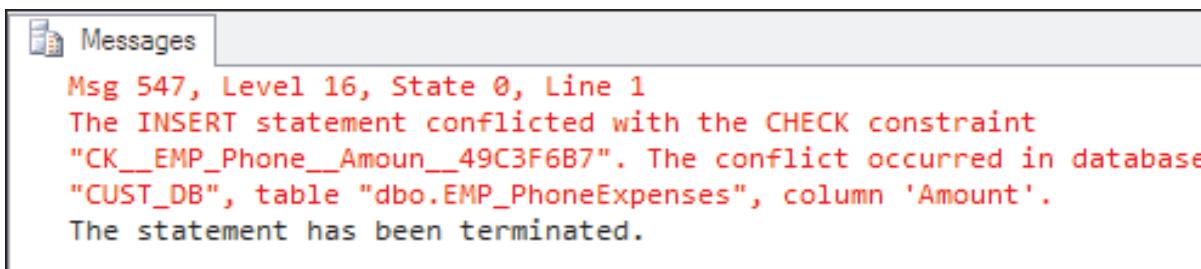
Đoạn mã 24:

```
USE [CUST_DB]

INSERT INTO dbo.EMP_PhoneExpenses values (101, 983345674, 9)

GO
```

Thông báo lỗi của Đoạn mã 24 xuất hiện khi ràng buộc **Amount** nhỏ hơn 10 được trình bày trong hình 7.10.



Hình 7.10: Thông báo lỗi đầu ra của ràng buộc CHECK

7.4.5 NOT NULL

Ràng buộc NOT NULL bắt buộc cột sẽ không chấp nhận các giá trị null. Các ràng buộc NOT NULL được sử dụng để bắt buộc tính toàn vẹn miền, tương tự như ràng buộc CHECK.

7.5 Kiểm tra tiến bộ của bạn

1. Tính năng nào sau đây của cột xác định xem các hàng trong bảng có thể chứa một giá trị null cho cột đó?

(A)	Default	(C)	GROUP BY
(B)	Multiplicity	(D)	Nullability

2. Một _____ trong một bảng là một cột trỏ tới một cột khóa chính trong bảng khác.

(A)	Khóa ngoại	(C)	Khóa lặp lại
(B)	Khóa phụ	(D)	Khóa cục bộ

3. Mã nào sau đây được sử dụng để thả một bảng từ cơ sở dữ liệu CUST _ DB?

(A)	DROP TABLE [dbo].[Table_1]	(C)	USE [CUST_DB] GO DELETE TABLE [dbo].[Table_1]
(B)	USE [CUST_DB] GO DROP TABLE [dbo].[Table_1]	(D)	USE [CUST_DB] GO SUBTRACT [dbo].[Table_1]

4. Thuộc tính nào sau đây của SQL Server được sử dụng để tạo ra các cột mã định danh có thể chứa các giá trị liên tục tự động tạo ra để xác định duy nhất mỗi hàng trong một bảng?

(A)	SELECT	(C)	INSERT
(B)	IDENTITY	(D)	DEFAULT

5. Ràng buộc _____ được sử dụng để đảm bảo rằng chỉ những giá trị duy nhất được nhập vào trong một cột hay tập hợp các cột.

(A)	UNIQUE	(C)	Khóa ngoại
(B)	DEFAULT	(D)	INSERT

7.5.1 Câu trả lời

1.	D
2.	A
3.	B
4.	B
5.	A

Tóm tắt

- Kiểu dữ liệu là một thuộc tính chỉ ra dung lượng lưu trữ của đối tượng và loại dữ liệu nó có thể giữ, như là dữ liệu số, dữ liệu ký tự, dữ liệu tiền tệ, và vân vân.
- SQL Server 2012 hỗ trợ ba loại dữ liệu:
 - Các kiểu dữ liệu hệ thống
 - Loại dữ liệu alias
 - Các loại do người dùng định nghĩa
- Hầu hết các bảng có một khóa chính, được tạo thành từ một hoặc nhiều cột của bảng chỉ ra các bản ghi là duy nhất.
- Đặc điểm tính chất không xác định của một cột xác định xem các hàng trong bảng có thể chứa giá trị null cho cột đó hay không.
- Định nghĩa DEFAULT cho một cột có thể được tạo ra vào thời điểm tạo bảng hoặc thêm ở giai đoạn sau vào bảng hiện có.
- Thuộc tính IDENTITY của SQL Server được sử dụng để tạo ra các cột mã định danh có thể chứa các giá trị liên tục tự động tạo ra để xác định duy nhất mỗi hàng trong một bảng.
- Các ràng buộc được dùng để áp dụng các quy tắc logic trong kinh doanh và tăng cường tính toàn vẹn dữ liệu.
- Ràng buộc UNIQUE được dùng để bảo đảm là chỉ các giá trị duy nhất được nhập vào một cột hoặc tập hợp cột.
- Khóa ngoại trong bảng là một cột trỏ tới cột khóa chính trong bảng khác.
- Ràng buộc CHECK hạn chế các giá trị có thể được đặt vào một cột.

Hãy thử tự làm

- Dịch vụ Saint Bảo hiểm Clara (SCI) là một công ty bảo hiểm hàng đầu có trụ sở tại New York, Mỹ. SCI Services muốn có một cách nhanh hơn, chính xác hơn, và ít tốn kém hơn để xử lý các yêu cầu bảo hiểm điều chỉnh cho các khách hàng công ty bảo hiểm của họ. Với một cơ sở khách hàng ngày càng tăng, họ quyết định tạo ra một ứng dụng dựa trên web sẽ được sử dụng không chỉ bởi các nhân viên làm việc tại hiện trường, nhưng cũng sẽ được sử dụng bởi các quản trị viên tại trụ sở chính.

SCI xử lý khoảng 650 yêu cầu bảo hiểm mỗi tháng, nhưng có thể tăng lên đến 15.000 hoặc nhiều hơn khi có một cơn bão hoặc một số thảm họa khác. Các viên chức có thể sử dụng phần mềm trên loại thiết bị mà họ lựa chọn: Máy tính bảng hoặc máy tính xách tay ngoài hiện trường, hoặc máy tính để bàn tại văn phòng của họ. Việc sử dụng Microsoft SQL Server 2005 như làm sở dữ liệu của phần mềm cho phép tiếp nhận và cập nhật tất cả các thông tin cần thiết liên quan đến khách hàng hoặc người yêu cầu bảo hiểm.

Với hàng ngàn khách hàng dự kiến mỗi tháng, tính toàn vẹn dữ liệu của dữ liệu trong cơ sở dữ liệu là rất quan trọng. Bạn cần phải thực hiện các nhiệm vụ sau:

- Tạo ra một cơ sở dữ liệu được gọi là **SaintClaraServices** để lưu trữ các chi tiết của công ty. Tạo ra bảng **CustomerHeader** với các chi tiết được đưa ra trong bảng 7.2.

Tên trường	Loại dữ liệu	Mô tả
ClientID	int	Cửa hàng mã khách hàng. Cột này là khóa chính
FirstName	char	Lưu trữ tên họ của khách hàng
LastName	char	Lưu trữ tên gọi của khách hàng
MiddleName	char	Lưu trữ tên đệm của khách hàng
Gender	char	Lưu trữ giới tính của khách hàng
DateOfBirth	datetime	Lưu trữ ngày tháng năm sinh của khách hàng
Address	varchar(max)	Lưu trữ địa chỉ của khách hàng
MaritalStatus	char	Lưu trữ tình trạng hôn nhân của khách hàng
Age	int	Lưu trữ tuổi của khách hàng
Employment	char	Lưu trữ nghề nghiệp của khách hàng
CompanyName	varchar(max)	Lưu trữ tên công ty
CompanyAddress	varchar(max)	Lưu trữ địa chỉ công ty

Bảng 7.2: Bảng CustomerHeader

Hãy thử tự làm

- b. Tạo ra bảng **CustomerDetails** với các thông số kỹ thuật được đưa ra trong bảng 7.3.

Tên trường	Loại dữ liệu	Mô tả
ClientID	int	Cửa hàng mã khách hàng. Cột này là khóa chính
FatherName	char	Lưu trữ tên cha của khách hàng
MotherName	char	Lưu trữ tên mẹ của khách hàng
Amount	money	Lưu trữ tiền vốn vay
Period	int	Lưu trữ giai đoạn bảo hiểm
Plan	char	Lưu trữ chương trình bảo hiểm
Premium	money	Lưu trữ phí bảo hiểm
NomineeName	char	Lưu trữ tên người đứng tên
Date	datetime	Lưu trữ ngày bảo hiểm được thực hiện

Bảng 7.3: Bảng CustomerDetails

- c. Thêm một khóa ngoại vào bảng **CustomerDetails**.

**Rộng bước hướng tới tương lai
nằm ở việc cho đi tất cả đến nay**

Phần - 8

Truy cập dữ liệu

Chào mừng bạn đến với phần **Truy cập dữ liệu**.

Phần này mô tả câu lệnh SELECT. Nó còn giải thích các biểu thức và các mệnh đề khác nhau được sử dụng với câu lệnh SELECT. Cuối cùng, phần này giới thiệu loại dữ liệu XML mới và mô tả cách làm việc với dữ liệu XML trong các bảng SQL Server 2012. Một cái nhìn thoáng qua về ngôn ngữ XQuery, chúng được dùng để truy vấn dữ liệu XML, cũng được thảo luận trong phần này.

Trong phần này, bạn sẽ học để:

- ➔ Mô tả câu lệnh SELECT, cú pháp và cách sử dụng câu lệnh này
- ➔ Giải thích các mệnh đề khác nhau được sử dụng với SELECT
- ➔ Nêu ra việc sử dụng mệnh đề ORDER BY
- ➔ Mô tả cách làm việc với XML định loại và không định loại
- ➔ Giải thích thủ tục để tạo ra, sử dụng, và xem lược đồ XML
- ➔ Giải thích việc sử dụng XQuery để truy cập dữ liệu XML

8.1 Giới thiệu

Câu lệnh SELECT là một lệnh lõi được sử dụng để truy cập dữ liệu trong SQL Server 2012. XML cho phép các nhà phát triển xây dựng tập hợp các thẻ riêng của họ và làm cho nó có thể cho các chương trình khác hiểu những thẻ này. XML là phương tiện ưa thích dành cho các nhà phát triển để lưu trữ, định dạng và quản lý dữ liệu trên web.

8.2 Câu lệnh SELECT

Bảng với dữ liệu của nó có thể được xem bằng cách sử dụng câu lệnh SELECT. Câu lệnh SELECT trong một truy vấn sẽ hiển thị thông tin cần thiết trong bảng.

Câu lệnh SELECT lấy các hàng và cột từ một hoặc nhiều bảng. Đầu ra của câu lệnh SELECT là một bảng khác gọi là tập kết quả. Câu lệnh SELECT còn nối hai bảng hoặc lấy một tập hợp con các cột từ một hoặc nhiều bảng. Câu lệnh SELECT định nghĩa các cột được sử dụng cho một truy vấn. Cú pháp của câu lệnh SELECT có thể bao gồm một loạt các biểu thức cách nhau bằng dấu phẩy. Mỗi biểu thức trong câu lệnh là một cột trong tập kết quả. Những cột xuất hiện theo cùng một trình tự như thứ tự của biểu thức trong câu lệnh SELECT.

Câu lệnh SELECT lấy các hàng từ cơ sở dữ liệu và cho phép lựa chọn một hoặc nhiều hàng hoặc cột từ một bảng. Sau đây là cú pháp cho câu lệnh SELECT.

Cú pháp:

```
SELECT <column_name1>...<column_nameN> FROM <table_name>
```

trong đó:

<table_name>: là bảng từ đó dữ liệu sẽ được hiển thị.

<column_name1>...<column_nameN>: là những cột sẽ được hiển thị.

Ghi chú - Tất cả các lệnh trong SQL Server 2012 không kết thúc bằng dấu chấm phẩy.

8.2.1 SELECT không có FROM

Nhiều phiên bản SQL sử dụng FROM trong truy vấn, nhưng trong tất cả các phiên bản từ SQL Server 2005, bao gồm cả SQL Server 2012, có thể sử dụng các câu lệnh SELECT mà không sử dụng mệnh đề FROM. Đoạn mã 1 cho thấy việc sử dụng câu lệnh SELECT mà không sử dụng mệnh đề FROM.

Đoạn mã 1:

```
SELECT LEFT('International', 5)
```

Đoạn mã này sẽ hiển thị chỉ 5 ký tự đầu tiên từ bên trái nhất của từ 'International'.

Kết quả được trình bày trong hình 8.1.

Results		Messages
(No column name)		
1	Inter	

Hình 8.1: 5 ký tự đầu tiên từ bên trái nhất của từ

8.2.2 Hiển thị tất cả các cột

Dấu sao (*) được sử dụng trong câu lệnh SELECT để lấy tất cả các cột từ bảng. Nó được sử dụng như một cách viết tắt để liệt kê tất cả các tên cột trong các bảng có tên trong mệnh đề FROM. Sau đây là cú pháp để chọn tất cả các cột.

Cú pháp:

```
SELECT * FROM <table_name>
```

trong đó:

*: chỉ ra tất cả các cột của bảng có tên trong mệnh đề FROM.

<table_name>: là tên của bảng mà từ đó thông tin sẽ được lấy ra. Có thể đưa vào số lượng bảng bất kỳ. Khi hai hoặc nhiều bảng được sử dụng, hàng của mỗi bảng được ánh xạ với hàng của các bảng khác. Hoạt động này mất rất nhiều thời gian nếu dữ liệu trong các bảng rất lớn. Do đó, khuyến khích nên sử dụng cú pháp này với một điều kiện.

Đoạn mã 2 cho thấy việc sử dụng '*' trong câu lệnh SELECT.

Đoạn mã 2:

```
USE AdventureWorks2012
SELECT * FROM HumanResources.Employee
GO
```

Kết quả một phần của Đoạn mã 2 với một số cột của bảng HumanResources.Employee được được trình bày trong hình 8.2.

Results		Messages
BusinessEntityID	NationalIDNumber	LoginID
1	295847284	adventure-works\ken0
2	245797967	adventure-works\timo0
3	509647174	adventure-works\roberto0
4	112457891	adventure-works\rob0
5	695256908	adventure-works\gail0
6	998320692	adventure-works\jossef0
7	134969118	adventure-works\dylan0
8	811994146	adventure-works\diane1

Hình 8.2: Hiển thị tất cả các cột

8.2.3 Hiển thị các cột đã chọn

Câu lệnh SELECT hiển thị hoặc trả lại một số cột có liên quan được người dùng lựa chọn hoặc được đề cập trong câu lệnh. Để hiển thị các cột cụ thể, kiến thức về tên cột có liên quan trong bảng là cần thiết. Sau đây là cú pháp để chọn các cột cụ thể.

Cú pháp:

```
SELECT <column_name1>..<column_nameN> FROM <table_name>
```

trong đó:

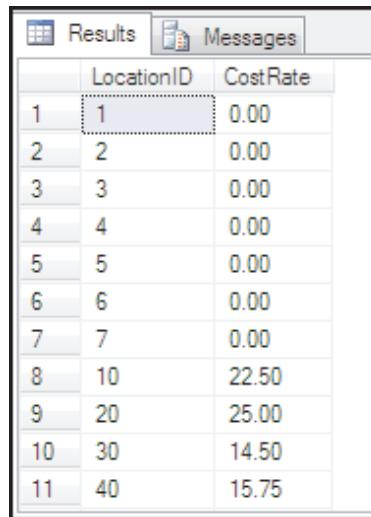
<column_name1>..<column_nameN>: là những cột sẽ được hiển thị.

Ví dụ, để hiển thị các tỷ lệ chi phí tại các địa điểm khác nhau từ bảng Production.Location trong cơ sở dữ liệu AdventureWorks2012, câu lệnh SELECT như được trình bày trong Đoạn mã 3.

Đoạn mã 3:

```
USE AdventureWorks2012
SELECT LocationID, CostRate FROM Production.Location
GO
```

Hình 8.3 trình bày các cột LocationID và CostRate từ cơ sở dữ liệu AdventureWorks2012.



	LocationID	CostRate
1	1	0.00
2	2	0.00
3	3	0.00
4	4	0.00
5	5	0.00
6	6	0.00
7	7	0.00
8	10	22.50
9	20	25.00
10	30	14.50
11	40	15.75

Hình 8.3: Các cột LocationID và CostRate

8.3 Các biểu thức khác nhau với SELECT

Câu lệnh SELECT cho phép người dùng chỉ ra các biểu thức khác nhau để xem tập kết quả theo một cách có trật tự. Những biểu thức này gán các tên gọi khác nhau cho các cột trong tập kết quả, tính toán các giá trị, và loại bỏ các giá trị trùng lặp.

8.3.1 Sử dụng các hằng số trong tập kết quả

Các hằng số dạng chuỗi ký tự được dùng khi các cột ký tự được ghép lại. Chúng giúp việc định dạng phù hợp hoặc khả năng đọc được. Những hằng số này không được chỉ định làm một cột riêng biệt trong tập kết quả.

Thường có hiệu quả hơn cho ứng dụng để dựng các giá trị hằng số vào trong các kết quả khi chúng được hiển thị, hơn là sử dụng máy chủ để kết hợp các giá trị hằng số. Ví dụ, để đưa vào ' : ' và ' → ' trong tập kết quả để hiển thị tên quốc gia, mã vùng quốc gia, và nhóm tương ứng của nó, câu lệnh SELECT được được trình bày trong Đoạn mã 4.

Đoạn mã 4:

```
USE AdventureWorks2012
SELECT [Name] + ' : ' + CountryRegionCode + ' → ' + [Group] FROM Sales.SalesTerritory
GO
```

Hình 8.4 hiển thị tên quốc gia, mã vùng quốc gia, và nhóm tương ứng từ Sales.SalesTerritory của cơ sở dữ liệu AdventureWorks2012.

	Results	Messages
(No column name)		
1	Northwest : US → North America	
2	Northeast : US → North America	
3	Central : US → North America	
4	Southwest : US → North America	
5	Southeast : US → North America	
6	Canada : CA → North America	
7	France : FR → Europe	
8	Germany : DE → Europe	
9	Australia : AU → Pacific	
10	United Kingdom : GB → Europe	

Hình 8.4: Tên quốc gia, mã vùng quốc gia, và nhóm tương ứng

8.3.2 Đổi tên các tên cột tập kết quả

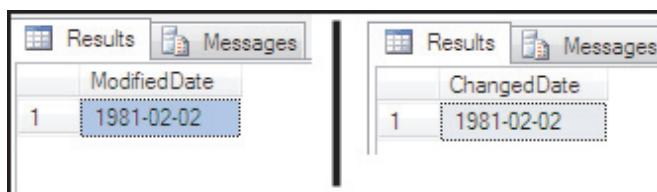
Khi các cột được hiển thị trong tập kết quả chúng đi kèm với các đầu đề tương ứng được chỉ ra trong bảng. Những đầu đề này có thể được thay đổi, đổi tên, hoặc có thể được chỉ định một tên mới bằng cách sử dụng mệnh đề AS. Do đó, bằng cách tùy biến các đầu đề, chúng trở nên dễ hiểu hơn và có ý nghĩa.

Đoạn mã 5 trình bày cách hiển thị 'ChangedDate' như đầu đề cho cột **ModifiedDate** trong bảng **dbo.Individual**, câu lệnh SELECT.

Đoạn mã 5:

```
USE CUST_DB
SELECT ModifiedDate as 'ChangedDate' FROM dbo.Individual
GO
```

Kết quả hiển thị 'ChangedDate' làm đầu đề cho cột **ModifiedDate** trong bảng **dbo.Individual**. Hình 8.5 trình bày đầu đề ban đầu và đầu đề đã thay đổi.



ModifiedDate
1981-02-02

ChangedDate
1981-02-02

Hình 8.5: Đầu đề cột được sửa đổi thành ChangedDate

8.3.3 Tính toán các giá trị trong tập kết quả

Câu lệnh SELECT có thể chứa các biểu thức toán học bằng cách áp dụng các toán tử cho một hoặc nhiều cột. Nó cho phép một tập kết quả để chứa các giá trị không tồn tại trong bảng cơ sở, nhưng được tính toán từ các giá trị được lưu trữ trong bảng cơ sở.

Ghi chú - Bảng được sử dụng trong mệnh đề FROM của một truy vấn được gọi là một bảng cơ sở.

Ví dụ, hãy xem xét bảng **Production.ProductCostHistory** từ cơ sở dữ liệu AdventureWorks2012. Xem xét ví dụ trong đó người sản xuất quyết định đưa ra 15% giảm chi phí tiêu chuẩn của tất cả các sản phẩm. Số tiền giảm giá không tồn tại nhưng có thể được tính bằng cách thực hiện câu lệnh SELECT được trình bày trong Đoạn mã 6.

Đoạn mã 6:

```
USE AdventureWorks2012
SELECT ProductID, StandardCost, StandardCost * 0.15 as Discount FROM
Production.ProductCostHistory
GO
```

Hình 8.6 trình bày kết quả khi số tiền giảm giá được tính toán sử dụng câu lệnh SELECT.

	ProductID	StandardCost	Discount
1	707	12.0278	1.804170
2	707	13.8782	2.081730
3	707	13.0863	1.962945
4	708	12.0278	1.804170
5	708	13.8782	2.081730
6	708	13.0863	1.962945
7	709	3.3963	0.509445
8	710	3.3963	0.509445
9	711	12.0278	1.804170
10	711	13.8782	2.081730
11	711	13.0863	1.962945

Hình 8.6: Số tiền giảm giá được tính toán

8.3.4 Sử dụng DISTINCT

Từ khóa DISTINCT ngăn chặn việc lấy ra các bản ghi trùng lặp. Nó giúp loại bỏ các hàng lặp lại từ tập kết quả của câu lệnh SELECT. Ví dụ, nếu cột StandardCost được chọn mà không sử dụng từ khóa DISTINCT, nó sẽ hiển thị tất cả các chi phí tiêu chuẩn hiện diện trong bảng. Khi sử dụng từ khóa DISTINCT trong truy vấn, SQL Server sẽ hiển thị mọi bản ghi của StandardCost chỉ một lần như được trình bày trong Đoạn mã 7.

Đoạn mã 7:

```
USE AdventureWorks2012
SELECT DISTINCT StandardCost FROM Production.ProductCostHistory
GO
```

8.3.5 Sử dụng TOP và PERCENT

Từ khóa TOP sẽ chỉ hiển thị tập hợp đầu tiên của các hàng như một tập kết quả. Tập hợp các hàng hoặc là hạn chế cho một số hoặc phần trăm các hàng. Biểu thức TOP cũng có thể được sử dụng với các câu lệnh khác như INSERT, UPDATE, và DELETE. Sau đây là cú pháp cho từ khóa TOP.

Cú pháp:

```
SELECT [ALL|DISTINCT] [TOP expression [PERCENT] [WITH TIES]]
```

trong đó:

expression: là số hoặc tỷ lệ phần trăm các hàng để được trả lại như là kết quả.

PERCENT: trả về số hàng bị giới hạn bởi tỷ lệ phần trăm.

WITH TIES: là số hàng bổ sung sẽ được hiển thị.

Câu lệnh SELECT có các mệnh đề khác nhau liên kết với nó. Trong phần này, từng mệnh đề sẽ được thảo luận chi tiết.

8.3.6 SELECT với INTO

Mệnh đề INTO tạo ra một bảng mới và chèn các hàng và cột được liệt kê trong câu lệnh SELECT vào nó. Mệnh đề INTO còn chèn các hàng hiện có vào bảng mới. Để thực hiện thi mệnh đề này với câu lệnh SELECT, người dùng phải có sự cho phép để CREATE TABLE trong cơ sở dữ liệu đích.

Cú pháp:

```
SELECT <column_name1>..<column_nameN> [INTO new_table] FROM table_list
```

trong đó:

new_table: là tên của bảng mới sẽ được tạo ra.

Đoạn mã 8 sử dụng mệnh đề INTO tạo ra một bảng mới Production.ProductName với các chi tiết như mã sản phẩm và tên của nó từ bảng Production.ProductModel.

Đoạn mã 8:

```
USE AdventureWorks2012
SELECT ProductModelID, Name INTO Production.ProductName FROM Production.
ProductModel
GO
```

Sau khi thực thi đoạn mã này, một thông báo nêu ra '(128 row(s) affected)' (128 hàng bị ảnh hưởng) được hiển thị.

Nếu một truy vấn được viết để hiển thị các hàng của bảng mới, đầu ra sẽ như được trình bày trong hình 8.7.

	ProductModelID	Name
1	122	All-Purpose Bike Stand
2	119	Bike Wash
3	115	Cable Lock
4	98	Chain
5	1	Classic Vest
6	2	Cycling Cap
7	121	Fender Set - Mountain
8	102	Front Brakes
9	103	Front Derailleur
10	3	Full-Finger Gloves
11	4	Half-Finger Gloves
12	109	Headlights - Dual-Beam
13	110	Headlights - Weather...
14	118	Hitch Rack - 4-Bike
15	97	HL Bottom Bracket
16	101	HL Crankset
17	106	HL Fork

Hình 8.7: Bảng mới được tạo ra

8.3.7 SELECT với WHERE

Mệnh đề WHERE với câu lệnh SELECT được sử dụng để chọn hoặc hạn chế có điều kiện các bản ghi lấy ra bằng truy vấn. Mệnh đề WHERE chỉ ra biểu thức Boolean để kiểm tra các hàng được trả lại bằng truy vấn. Hàng được trả lại nếu biểu thức là true và bị loại bỏ nếu là false.

Cú pháp:

```
SELECT <column_name1>...<column_nameN> FROM <table_name> WHERE <
search_condition>
```

trong đó:

search_condition: là điều kiện để được đáp ứng bằng các hàng.

Bảng 8.1 trình bày các toán tử khác nhau có thể được sử dụng với mệnh đề WHERE.

Toán tử	Mô tả
=	Bằng
<>	Không bằng
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng

Toán tử	Mô tả
!	Không
BETWEEN	Giữa một phạm vi
LIKE	Tìm kiếm một mẫu có thứ tự
IN	Ở trong một phạm vi

Bảng 8.1: Các toán tử

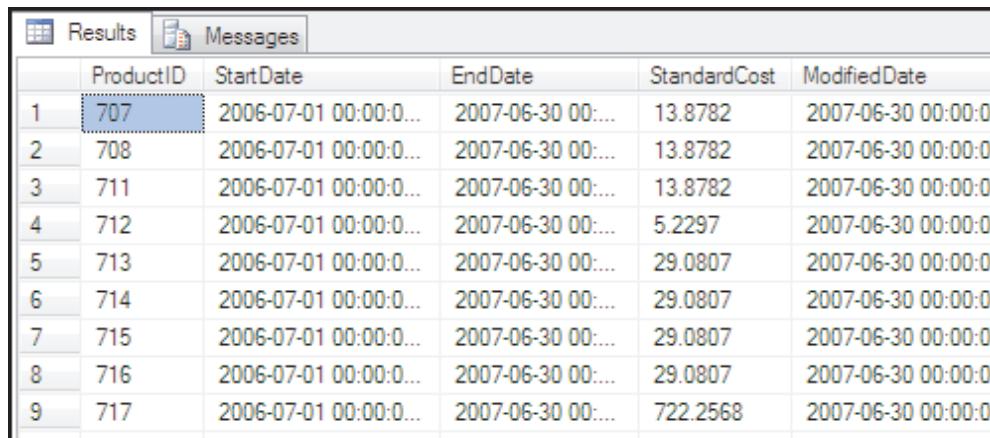
Đoạn mã 9 trình bày toán tử bằng với mệnh đề WHERE để hiển thị dữ liệu với **EndDate 6/30/2007 12:00:00 AM**.

Đoạn mã 9:

```
USE AdventureWorks2012
SELECT * FROM Production.ProductCostHistory WHERE EndDate = '6/30/2007 12:00:00
AM'
GO
```

Đoạn mã 9 sẽ trả lại tất cả các bản ghi từ bảng Production.ProductCostHistory trong đó có ngày kết thúc là '6/30/2007 12:00:00 AM'.

Kết quả SELECT với mệnh đề WHERE được trình bày trong hình 8.8.



	ProductID	StartDate	EndDate	StandardCost	ModifiedDate
1	707	2006-07-01 00:00:0...	2007-06-30 00:00:0...	13.8782	2007-06-30 00:00:00
2	708	2006-07-01 00:00:0...	2007-06-30 00:00:0...	13.8782	2007-06-30 00:00:00
3	711	2006-07-01 00:00:0...	2007-06-30 00:00:0...	13.8782	2007-06-30 00:00:00
4	712	2006-07-01 00:00:0...	2007-06-30 00:00:0...	5.2297	2007-06-30 00:00:00
5	713	2006-07-01 00:00:0...	2007-06-30 00:00:0...	29.0807	2007-06-30 00:00:00
6	714	2006-07-01 00:00:0...	2007-06-30 00:00:0...	29.0807	2007-06-30 00:00:00
7	715	2006-07-01 00:00:0...	2007-06-30 00:00:0...	29.0807	2007-06-30 00:00:00
8	716	2006-07-01 00:00:0...	2007-06-30 00:00:0...	29.0807	2007-06-30 00:00:00
9	717	2006-07-01 00:00:0...	2007-06-30 00:00:0...	722.2568	2007-06-30 00:00:00

Hình 8.8: SELECT với mệnh đề WHERE

Tất cả các truy vấn trong SQL sử dụng ngoặc đơn để bao bọc các giá trị văn bản. Ví dụ, hãy xem xét truy vấn sau đây, mà lấy tất cả các bản ghi từ bảng Person.Address có Bothell là thành phố.

Đoạn mã 10 trình bày toán tử bằng với mệnh đề WHERE để hiển thị dữ liệu với địa chỉ có thành phố là Bothell.

Đoạn mã 10:

```
USE AdventureWorks2012
SELECT DISTINCT StandardCost FROM Production.ProductCostHistory
GO
```

Kết quả của truy vấn được trình bày trong hình 8.9.

	AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode
1	5	1226 Shoe St.	NULL	Bothell	79	98011
2	11	1318 Lasalle Street	NULL	Bothell	79	98011
3	6	1399 Firestone Drive	NULL	Bothell	79	98011
4	18	1873 Lion Circle	NULL	Bothell	79	98011
5	40	1902 Santa Cruz	NULL	Bothell	79	98011
6	1	1970 Napa Ct.	NULL	Bothell	79	98011
7	10	250 Race Court	NULL	Bothell	79	98011
8	868	25111 228th St Sw	NULL	Bothell	79	98011
9	19	3148 Rose Street	NULL	Bothell	79	98011

Hình 8.9: Truy vấn với dấu ngoặc đơn

Các giá trị số không được đưa vào trong dấu ngoặc kép như được trình bày trong Đoạn mã 11.

Đoạn mã 11:

```
USE AdventureWorks2012
SELECT * FROM HumanResources.Department WHERE DepartmentID < 10
GO
```

Truy vấn trong Đoạn mã 11 hiển thị tất cả những bản ghi này trong đó giá trị trong DepartmentID ít hơn 10.

Kết quả của truy vấn được trình bày trong hình 8.10.

		Results	Messages	
	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2002-01-01
2	2	Tool Design	Research and Development	2002-01-01
3	3	Sales	Sales and Marketing	2002-01-01
4	4	Marketing	Sales and Marketing	2002-01-01
5	5	Purchasing	Inventory Management	2002-01-01
6	6	Research and Development	Research and Development	2002-01-01
7	7	Production	Manufacturing	2002-01-01
8	8	Production Control	Manufacturing	2002-01-01

Hình 8.10: Kết quả của mệnh đề WHERE với toán tử <

Mệnh đề WHERE cũng có thể được sử dụng với các ký tự đại diện như được trình bày trong bảng 8.2. Tất cả các ký tự đại diện được sử dụng cùng với từ khóa LIKE để thực hiện truy vấn chính xác và cụ thể.

Ký tự đại diện	Mô tả	Ví dụ
-	Ký hiệu này sẽ hiển thị một ký tự đơn	SELECT * FROM Person.Contact WHERE Suffix LIKE 'Jr_'
%	Ký hiệu này sẽ hiển thị một chuỗi với độ dài bất kỳ	SELECT * FROM Person.Contact WHERE LastName LIKE 'B%'
[]	Ký hiệu này sẽ hiển thị một ký tự đơn ở trong một phạm vi được bao bọc trong dấu ngoặc vuông	SELECT * FROM Sales.CurrencyRate WHERE ToCurrencyCode LIKE 'C[AN][DY]'
[^]	Nó sẽ hiển thị bất kỳ ký tự đơn lẻ nào không nằm trong phạm vi được đặt trong dấu ngoặc	SELECT * FROM Sales.CurrencyRate WHERE ToCurrencyCode LIKE 'A[^R][^S]'

Bảng 8.2: Các ký tự đại diện

Mệnh đề WHERE còn sử dụng các toán tử logic như là AND, OR, và NOT. Những toán tử này được sử dụng với các điều kiện tìm kiếm trong mệnh đề WHERE.

Toán tử AND nối hai hoặc nhiều điều kiện và trả lại TRUE chỉ khi cả hai điều kiện là TRUE. Do vậy, nó trả lại tất cả các hàng từ các bảng nơi cả hai điều kiện được liệt kê là true.

Đoạn mã 12 trình bày toán tử AND.

Đoạn mã 12:

```
USE AdventureWorks2012
SELECT * FROM Sales.CustomerAddress WHERE AddressID > 900 AND AddressTypeID = 5
GO
```

Toán tử OR trả lại TRUE và hiển thị tất cả các hàng nếu nó thỏa mãn một trong những điều kiện này.

Đoạn mã 13 trình bày toán tử OR.

Đoạn mã 13:

```
USE AdventureWorks2012
SELECT * FROM Sales.CustomerAddress WHERE AddressID < 900 OR AddressTypeID = 5
GO
```

Truy vấn trong Đoạn mã 13 sẽ hiển thị tất cả các hàng có AddressID ít hơn 900 hoặc có AddressTypeID bằng 5.

Toán tử NOT phủ định điều kiện tìm kiếm. Đoạn mã 14 trình bày toán tử NOT.

Đoạn mã 14:

```
USE AdventureWorks2012
SELECT * FROM Sales.CustomerAddress WHERE NOT AddressTypeID = 5
GO
```

Đoạn mã 14 sẽ hiển thị tất cả các bản ghi với AddressTypeID không bằng 5. Có thể sử dụng nhiều toán tử logic trong câu lệnh SELECT đơn lẻ. Khi có nhiều hơn một toán tử logic được sử dụng, NOT được đánh giá đầu tiên, sau đó AND, và cuối cùng là OR.

8.3.8 Mệnh đề GROUP BY

Mệnh đề GROUP BY phân vùng tập kết quả vào một hoặc nhiều tập hợp con. Một tập hợp con có các giá trị và biểu thức chung. Nếu một hàm tổng hợp được sử dụng trong mệnh đề GROUP BY, tập kết quả tạo ra giá trị duy nhất cho mỗi tổng hợp.

Từ khóa GROUP BY được theo sau bằng một danh sách các cột, được gọi là cột được nhóm. Mỗi cột được nhóm hạn chế số lượng hàng của tập kết quả. Đối với mỗi cột được nhóm, chỉ có một hàng. Mệnh đề GROUP BY có thể có nhiều hơn một cột được nhóm.

Sau đây là cú pháp của mệnh đề GROUP BY.

Cú pháp:

```
SELECT <column_name1>..<column_nameN> FROM <table_name> GROUP BY <column_name>
```

trong đó:

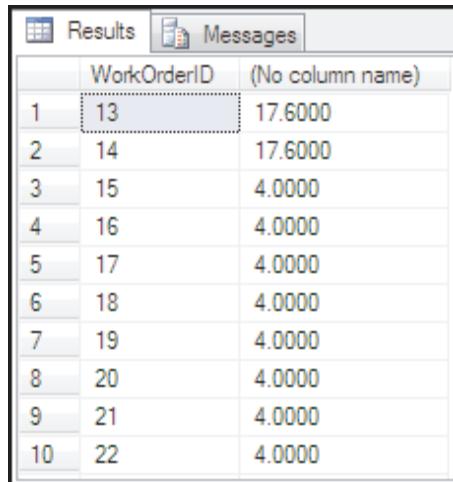
column_name1: là tên của cột theo đó tập kết quả nên được nhóm lại.

Ví dụ, hãy xem xét rằng nếu tổng số giờ tài nguyên đã được tìm thấy cho mỗi lệnh sản xuất, truy vấn trong Đoạn mã 15 sẽ lấy ra tập kết quả.

Đoạn mã 15:

```
USE AdventureWorks2012
SELECT WorkOrderID, SUM(ActualResourceHrs) FROM Production.WorkOrderRouting
GROUP BY WorkOrderID
GO
```

Kết quả được trình bày trong hình 8.11.



	WorkOrderID	(No column name)
1	13	17.6000
2	14	17.6000
3	15	4.0000
4	16	4.0000
5	17	4.0000
6	18	4.0000
7	19	4.0000
8	20	4.0000
9	21	4.0000
10	22	4.0000

Hình 8.11: Mệnh đề GROUP BY

Mệnh đề GROUP BY có thể được sử dụng với các mệnh đề khác nhau.

8.3.9 Các mệnh đề và câu lệnh

Microsoft SQL Server 2012 cung cấp các phần tử cú pháp truy vấn nâng cao để truy cập và xử lý dữ liệu mạnh mẽ hơn.

→ Common Table Expression (CTE) trong câu lệnh SELECT và INSERT

CTE là một tập kết quả tạm thời có tên dựa trên truy vấn SELECT và INSERT.

Đoạn mã 16 trình bày việc sử dụng CTE trong câu lệnh INSERT.

Đoạn mã 16:

```
USE CUST_DB
CREATE TABLE NewEmployees (EmployeeID smallint, FirstName char(10), LastName char(10), Department varchar(50), HiredDate datetime, Salary money);
INSERT INTO NewEmployees
VALUES(11, 'Kevin', 'Blaine', 'Research', '2012-07-31', 54000);
WITH EmployeeTemp (EmployeeID, FirstName, LastName, Department,
HiredDate, Salary)
AS
(
SELECT * FROM NewEmployees
)
SELECT * FROM EmployeeTemp
```

Câu lệnh trong Đoạn mã 16 chèn một hàng mới cho bảng **NewEmployees** và chuyển tập kết quả tạm thời thành **EmployeeTemp** như được trình bày trong hình 8.12.

Results						
	EmployeeID	FirstName	LastName	Department	HiredDate	Salary
1	11	Kevin	Blaine	Research	2012-07-31 00:00:00.000	54000.00

Hình 8.12: Chuyển kết quả tạm thời thành EmployeeTemp

→ Mệnh đề OUTPUT trong câu lệnh INSERT và UPDATE

Mệnh đề OUTPUT trả về thông tin về các hàng bị ảnh hưởng bởi câu lệnh INSERT và câu lệnh UPDATE.

Đoạn mã 17 trình bày cách sử dụng câu lệnh UPDATE với câu lệnh INSERT.

Đoạn mã 17:

```
USE CUST_DB;
GO
CREATE TABLE dbo.table_3
(
    id INT,
    employee VARCHAR(32)
)
go
INSERT INTO dbo.table_3 VALUES
    (1, 'Matt')
    , (2, 'Joseph')
    , (3, 'Renny')
    , (4, 'Daisy');
GO
DECLARE @updatedTable TABLE
(
    id INT, olldata_employee VARCHAR(32), newdata_employee VARCHAR(32)
);
UPDATE dbo.table_3
Set employee=UPPER(employee)
OUTPUT
    inserted.id,
    deleted.employee,
    inserted.employee
INTO @updatedTable
SELECT * FROM @updatedTable
```

Sau khi thực thi Đoạn mã 17, kết quả nơi các hàng bị ảnh hưởng bởi câu lệnh `INSERT` và câu lệnh `UPDATE` được trình bày trong hình 8.13.

	id	olddata_employee	newdata_employee
1	1	Matt	MATT
2	2	Joseph	JOSEPH
3	3	Renny	RENNY
4	4	Daisy	DAISY

Hình 8.13: Kết quả của câu lệnh UPDATE

→ Mệnh đề `.WRITE`

Mệnh đề `.WRITE` được sử dụng trong câu lệnh `UPDATE` để thay thế giá trị trong một cột có kiểu dữ liệu có giá trị lớn. Sau đây là cú pháp cho mệnh đề `.WRITE`.

Cú pháp:

```
.WRITE(expression, @offset, @Length)
```

trong đó:

`expression`: là chuỗi ký tự sẽ được đặt vào cột loại dữ liệu có giá trị lớn.

`@offset`: là giá trị bắt đầu (đơn vị) nơi thay thế sẽ được thực hiện.

`@Length`: là chiều dài của phần trong cột, bắt đầu từ `@offset` được thay thế bằng biểu thức.

Đoạn mã 18 trình bày mệnh đề `.WRITE` được sử dụng như thế nào trong câu lệnh `UPDATE`.

Đoạn mã 18:

```
USE CUST_DB;
GO
CREATE TABLE dbo.table_5
(
    Employee_role VARCHAR(max),
    Summary VARCHAR(max)
)
```

```

INSERT INTO dbo.table_5(Employee_role, Summary) VALUES ('Nghiên cứu',
'Dây là chuỗi không phải Unicode rất dài')

SELECT *FROM dbo.table_5

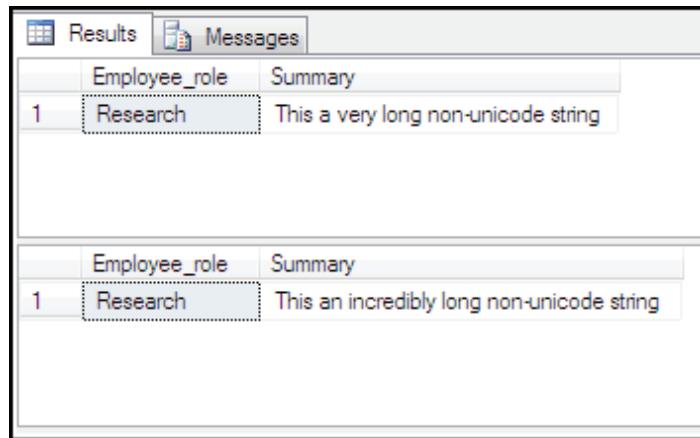
UPDATE dbo.table_5 SET Summary .WRITE('n incredibly', 6,5)

WHERE Employee_role LIKE 'Nghiên cứu'

SELECT *FROM dbo.table_5

```

Hình 8.14 hiển thị kết quả của truy vấn mệnh đề .WRITE.



	Employee_role	Summary
1	Research	This a very long non-unicode string

	Employee_role	Summary
1	Research	This an incredibly long non-unicode string

Hình 8.14: Kết quả của truy vấn mệnh đề .WRITE

8.4 Mệnh đề ORDER BY

Nó chỉ ra thứ tự các cột nên được sắp xếp trong một tập kết quả. Nó phân loại các kết quả truy vấn bằng một hoặc nhiều cột. Phân loại có thể là theo thứ tự tăng dần (ASC) hoặc giảm dần (DESC). Theo mặc định, các bản ghi được sắp xếp theo thứ tự ASC. Để chuyển sang chế độ giảm dần, hãy sử dụng từ khóa tùy chọn DESC. Khi nhiều trường được sử dụng, SQL Server xem xét trường tận cùng bên trái làm mức chính của phân loại và những cái khác làm các mức phân loại thấp hơn.

Cú pháp:

```
SELECT <column_name> FROM <table_name> ORDER BY <column_name> {ASC | DESC}
```

Câu lệnh SELECT trong Đoạn mã 19 sắp xếp các kết quả truy vấn trên cột SalesLastYear của bảng Sales.SalesTerritory.

Đoạn mã 19:

```

USE AdventureWorks2012
SELECT * FROM Sales.SalesTerritory ORDER BY SalesLastYear
GO

```

Kết quả được trình bày trong hình 8.15.

	TerritoryID	Name	Country...	Group	Sales...	SalesLastYear	Cos
1	8	Germany	DE	Europe	3805...	1307949.7917	0.0
2	10	United Kingdom	GB	Europe	5012...	1635823.3967	0.0
3	9	Australia	AU	Pacific	5977...	2278548.9776	0.0
4	7	France	FR	Europe	4772...	2396539.7601	0.0
5	3	Central	US	North America	3072...	3205014.0767	0.0
6	1	Northwest	US	North America	7887...	3298694.4938	0.0
7	2	Northeast	US	North America	2402...	3607148.9371	0.0
8	5	Southeast	US	North America	2538...	3925071.4318	0.0
9	4	Southwest	US	North America	1051...	5366575.7098	0.0

Hình 8.15: Mệnh đề ORDER BY

8.5 Làm việc với XML

Extensible Markup Language (XML) cho phép các nhà phát triển xây dựng tập hợp các thẻ riêng của họ và làm cho các chương trình khác có thể hiểu những thẻ này. XML là phương tiện ưa thích dành cho các nhà phát triển để lưu trữ, định dạng và quản lý dữ liệu trên web. Các ứng dụng của ngày nay có sự kết hợp các công nghệ như ASP, công nghệ Microsoft .NET, XML, và SQL Server 2012 làm việc song song. Trong kịch bản như vậy, tốt nhất là lưu trữ dữ liệu XML ở trong SQL Server 2012.

Các cơ sở dữ liệu XML nguyên gốc trong SQL Server 2012 có một số các ưu điểm. Một số trong số này được liệt kê như sau:

- Dễ tìm kiếm và quản lý dữ liệu - Tất cả các dữ liệu XML được lưu trữ cục bộ ở một nơi, do đó làm cho nó dễ tìm kiếm và quản lý hơn.
- Hiệu suất tốt hơn - Truy vấn từ một cơ sở dữ liệu XML được thực hiện tốt sẽ nhanh hơn so với các truy vấn trên các tài liệu được lưu trữ trong một hệ thống tập tin. Ngoài ra, cơ sở dữ liệu về bản chất phân tích từng tài liệu khi lưu trữ chúng.
- Dễ xử lý dữ liệu - Có thể xử lý các tài liệu lớn một cách dễ dàng.

SQL Server 2012 hỗ trợ lưu trữ thuần dữ liệu XML bằng cách sử dụng loại dữ liệu `xml`. Các phần sau khám phá kiểu dữ liệu `xml`, làm việc với XML được định kiểu và không định kiểu, lưu trữ chúng trong SQL Server 2012, và sử dụng XQuery để lấy dữ liệu từ các cột của kiểu dữ liệu `xml`.

8.5.1 Kiểu dữ liệu XML

Ngoài các kiểu dữ liệu thường được sử dụng thường xuyên, SQL Server 2012 cung cấp một kiểu dữ liệu hoàn mới ở dạng kiểu dữ liệu `xml`.

Kiểu dữ liệu `xml` được sử dụng để lưu trữ các tài liệu và phân đoạn XML trong một cơ sở dữ liệu SQL Server. Phân đoạn XML là một thể hiện XML với phần tử mức cao nhất thiếu từ cấu trúc của nó.

Sau đây là cú pháp để tạo ra bảng với các cột thuộc loại `xml`.

Cú pháp:

```
CREATE TABLE <table_name> ( [column_list,] <column_name> xml [, column_list])
```

Đoạn mã 20 tạo ra một bảng mới có tên là **PhoneBilling** với một trong các cột thuộc kiểu dữ liệu `xml`.

Đoạn mã 20:

```
USE AdventureWorks2012
CREATE TABLE Person.PhoneBilling (Bill_ID int PRIMARY KEY, MobileNumber bigint
UNIQUE, CallDetails xml)
GO
```

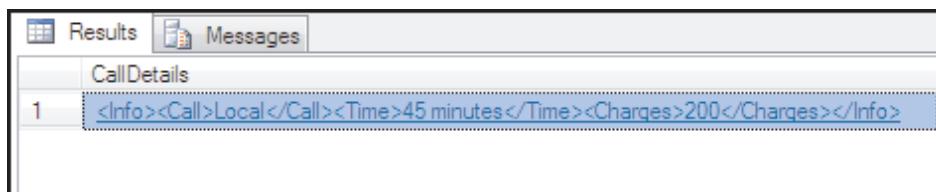
Cột thuộc loại `xml` có thể được thêm vào bảng tại thời điểm tạo ra hoặc sau khi tạo ra nó. Các cột thuộc kiểu dữ liệu `xml` hỗ trợ các giá trị `DEFAULT` cũng như là ràng buộc `NOT NULL`.

Dữ liệu có thể được chèn vào cột `xml` trong bảng `Person.PhoneBilling` như được trình bày trong Đoạn mã 21.

Đoạn mã 21:

```
USE AdventureWorks2012
INSERT INTO Person.PhoneBilling VALUES (100, 9833276605,
'<Info><Call>Local</Call><Time>45 minutes</Time><Charges>200</Charges></
Info>')
SELECT CallDetails FROM Person.PhoneBilling
GO
```

Kết quả được trình bày trong hình 8.16.



Hình 8.16: Dữ liệu XML trong các cột

Câu lệnh `DECLARE` được sử dụng để tạo ra các biến thuộc loại `xml`.

Đoạn mã 22 trình bày cách tạo ra một biến thuộc kiểu `xml`.

Đoạn mã 22:

```
DECLARE @xmlvar xml
SELECT @xmlvar='<Employee name="Joan" />'
```

Các cột thuộc kiểu dữ liệu `xml` không thể được sử dụng như một khóa chính, khóa ngoại, hoặc là một ràng buộc duy nhất.

8.5.2 XML định kiểu và không định kiểu

Có hai cách lưu trữ các tài liệu XML trong các cột thuộc kiểu dữ liệu `xml`, cụ thể là XML định kiểu và không định kiểu. Đối tượng định hình XML mà có một lược đồ được liên kết với nó được gọi là đối tượng định hình XML được phân loại. Lược đồ là một đầu đề cho một thể hiện hoặc tài liệu XML. Nó mô tả cấu trúc và giới hạn nội dung của các tài liệu XML bằng cách kết hợp các kiểu dữ liệu `xml` với các loại phần tử và thuộc tính XML. Nên kết hợp các lược đồ XML với các thể hiện hoặc tài liệu XML bởi vì dữ liệu có thể được xác thực trong khi nó đang được lưu trữ vào cột kiểu dữ liệu `xml`.

SQL Server không thực hiện bất kỳ xác thực nào đối với dữ liệu được nhập vào cột `xml`. Tuy nhiên, nó bảo đảm là dữ liệu sẽ được lưu được định hình tốt. Dữ liệu XML không định kiểu có thể được tạo ra và được lưu trữ trong các cột bảng hoặc các biến tùy theo nhu cầu và phạm vi của dữ liệu.

Bước đầu tiên trong việc sử dụng XML được phân loại là đăng ký một lược đồ. Điều này được thực hiện bằng cách sử dụng câu lệnh `CREATE XML SCHEMA COLLECTION` như được trình bày trong Đoạn mã 23.

Đoạn mã 23:

```
USE SampleDB
CREATE XML SCHEMA COLLECTION CricketSchemaCollection
AS N'<xsd:schema>
<xsd:element name="MatchDetails">
<xsd:complexType>
<xsd:complexContent>
<xsd:restriction base="xsd:anyType">
<xsd:sequence>
<xsd:element name="Team" minOccurs="0" maxOccurs="unbounded">
<xsd:complexType>
<xsd:complexContent>
<xsd:restriction base="xsd:anyType">
<xsd:sequence />
```

```

<xsd:attribute name="country" type="xsd:string" />
<xsd:attribute name="score" type="xsd:string" />
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
</xsd:element>
</xsd:schema>'
```

GO

Câu lệnh CREATE XML SCHEMA COLLECTION tạo ra một bộ sưu tập các lược đồ, bất kỳ cái nào trong đó có thể được sử dụng để xác thực dữ liệu XML định kiểu với tên của bộ sưu tập. Ví dụ này trình bày một lược đồ mới được gọi là **CricketSchemaCollection** đang được thêm vào cơ sở dữ liệu **SampleDB**. Một khi lược đồ được đăng ký, lược đồ đó có thể được sử dụng trong các thể hiện mới của kiểu dữ liệu xml.

Đoạn mã 24 tạo ra một bảng với cột loại `xml` và chỉ ra một lược đồ cho cột đó.

Đoạn mã 24:

```

USE SampleDB
CREATE TABLE CricketTeam ( TeamID int identity not null, TeamInfo
xml (CricketSchemaCollection) )
GO
```

Để tạo ra các hàng mới với dữ liệu XML định kiểu, câu lệnh `INSERT` có thể được sử dụng như được trình bày trong Đoạn mã 25.

Đoạn mã 25:

```

USE SampleDB
INSERT INTO CricketTeam (TeamInfo) VALUES ('<MatchDetails><Team
country="Australia" score="355"></Team><Team country="Zimbabwe"
score="200"></Team><Team country="England" score="475"></Team></
MatchDetails>')
GO
```

Một biến XML định kiểu cũng có thể được tạo ra bằng cách chỉ ra tên bộ sưu tập lược đồ. Ví dụ, trong Đoạn mã 26, một nhóm biến được khai báo là biến XML định kiểu với tên lược đồ là **CricketSchemaCollection**. Câu lệnh SET được sử dụng để gán biến làm một phân đoạn XML.

Đoạn mã 26:

```
USE SampleDB
DECLARE @team xml (CricketSchemaCollection)
SET @team = '<MatchDetails><Team country="Australia"></Team></MatchDetails>'
SELECT @team
GO
```

8.5.3 XQuery

Sau khi dữ liệu XML đã được lưu trữ bằng cách sử dụng kiểu dữ liệu `xml`, nó có thể được truy vấn và lấy ra sử dụng một ngôn ngữ có tên là XQuery. XML Query hay XQuery là một ngôn ngữ truy vấn mới, ngôn ngữ này kết hợp cú pháp quen thuộc với những người phát triển làm việc với cơ sở dữ liệu quan hệ, và ngôn ngữ XPath được dùng để chọn các phần riêng lẻ hoặc tập hợp các thành tố từ một tài liệu XML. XQuery có thể là truy vấn dữ liệu XML có cấu trúc hoặc bán cấu trúc. Để truy vấn một thể hiện XML được lưu trữ trong một biến hoặc cột thuộc loại `xml`, các phương thức kiểu dữ liệu `xml` được sử dụng. Ví dụ, biến thuộc loại `xml` được khai báo và truy vấn bằng cách sử dụng các phương thức của kiểu dữ liệu `xml`. Các nhà phát triển cần phải truy vấn các tài liệu XML, và điều này liên quan đến việc biến đổi các tài liệu XML ở định dạng yêu cầu. XQuery làm cho nó có thể thực hiện các truy vấn phức tạp đối với một nguồn dữ liệu XML trên web.

Một số phương thức kiểu dữ liệu `xml` được sử dụng với XQuery được mô tả như sau:

→ `exist()`

Phương thức này được sử dụng để xác định xem một hay nhiều nút quy định có mặt trong tài liệu XML hay không. Nó trả về 1 nếu biểu thức XQuery đã trả về ít nhất một nút, 0 nếu biểu thức XQuery đã đánh giá thành kết quả rỗng, và `NULL` nếu thể hiện kiểu dữ liệu `xml` dựa vào đó truy vấn đã được thực hiện là `NULL`.

Đoạn mã 27 trình bày việc sử dụng phương thức `exist()`. Giả sử là nhiều bản ghi đã được chèn vào bảng.

Đoạn mã 27:

```
USE SampleDB
SELECT TeamID FROM CricketTeam WHERE TeamInfo.exist('(/MatchDetails/
Team)') = 1
GO
```

Điều này sẽ chỉ trả lại những giá trị **TeamID** nơi phần tử **Team** đã được chỉ ra trong **TeamInfo**. Kết quả được trình bày trong hình 8.17.

Results		Messages
TeamID		
1	1	

Figure 8.17: Phương thức exist()

→ query()

Có thể sử dụng phương thức **query()** để lấy toàn bộ nội dung của một tài liệu XML hoặc một phần đã chọn của tài liệu XML. Đoạn mã 28 trình bày việc sử dụng phương thức **query()**.

Đoạn mã 28:

```
USE SampleDB
SELECT TeamInfo.query('/MatchDetails/Team') AS Info FROM CricketTeam
GO
```

Kết quả được trình bày trong hình 8.18.

Results		Messages
Info		
1	<Team country="Australia" score="355" /><Team co...	

Hình 8.18: Phương thức query()

→ value()

Có thể sử dụng phương thức **value()** để trích xuất các giá trị vô hướng từ một kiểu dữ liệu **xml**. Đoạn mã 29 trình bày phương thức này.

Đoạn mã 29:

```
USE SampleDB
SELECT TeamInfo.value('(/MatchDetails/Team/@score)[1]', 'varchar(20)')
AS Score FROM CricketTeam where TeamID=1
GO
```

Kết quả của lệnh này được trình bày trong hình 8.19.

Results		Messages
Score		
1	355	

Hình 8.19: Phương thức value()

8.6 Kiểm tra tiến bộ của bạn

1. Cái nào sau đây cho phép các nhà phát triển xây dựng tập hợp các thẻ riêng của họ và làm cho các chương trình khác có thể hiểu những thẻ này?

(A)	Xquery	(C)	DHTML
(B)	HTML	(D)	XML

2. Câu lệnh _____ lấy các hàng và cột từ một hoặc nhiều bảng.

(A)	SELECT	(C)	INSERT
(B)	DISPLAY	(D)	SHOW

3. Điều nào sau đây là định dạng chung của truy vấn mệnh đề .WRITE?

(A)	<pre>ADD INTO dbo.table_5 (Employee_role, Summary) VALUES ('Research', 'This a very long non-unicode string') SELECT * FROM dbo.table_5 UPDATE dbo.table_5 SET Summary .WRITE ('n incredibly') WHERE Employee_role LIKE 'Nghiên cứu' SELECT * FROM dbo.table_5</pre>	(C)	<pre>INSERT INTO dbo.table_5 (Employee_role, Summary) VALUES ('Nghiên cứu', 'Đây là chuỗi không phải Unicode rất dài') SELECT * FROM dbo.table_5 UPDATE dbo.table_5 SET Summary .WRITE ('n incredibly', 6,5) WHERE Employee_role LIKE 'Nghiên cứu' SELECT * FROM dbo.table_5</pre>
(B)	<pre>INSERT INTO dbo.table_5 (Employee_role, Summary) VALUES ('Nghiên cứu', 'Đây là chuỗi không phải Unicode rất dài') SELECT * FROM dbo.table_5 UPDATE dbo.table_5 SET Summary .WRITE ('n incredibly', 6,5) WHERE Employee_role LIKE 'Nghiên cứu' SELECT * FROM dbo.table_5</pre>	(D)	<pre>INSERT INTO dbo.table_5 (Employee_role, Summary) VALUES ('Nghiên cứu', 'Đây là chuỗi không phải Unicode rất dài') SELECT * FROM dbo.table_5 dbo.table_5 SET Summary ('n incredibly', 6,5) WHERE Employee_role LIKE 'Nghiên cứu' SELECT * FROM dbo.table_5</pre>

4. Mệnh đề nào sau đây với câu lệnh SELECT được sử dụng để chỉ ra các bảng hoặc lấy ra các bản ghi?

(A)	WHERE	(C)	.VALUE
(B)	FROM	(D)	.WRITE

5. _____ được sử dụng để nâng cao hiệu quả của các truy vấn trên các tài liệu XML được lưu trữ trong một cột XML.

(A)	Lập chỉ mục XML	(C)	Truy vấn XML
(B)	Nhập khẩu XML	(D)	Xuất khẩu XML

8.6.1 Câu trả lời

1.	D
2.	A
3.	B
4.	B
5.	A

Tóm tắt

- Câu lệnh SELECT lấy các hàng và cột từ các bảng.
- Câu lệnh SELECT cho phép người dùng chỉ ra các biểu thức khác nhau để xem tập kết quả theo một cách có trật tự.
- Câu lệnh SELECT có thể chứa các biểu thức toán học bằng cách áp dụng các toán tử cho một hoặc nhiều cột.
- Từ khóa DISTINCT ngăn việc gọi ra các bản ghi trùng lặp.
- XML cho phép các nhà phát triển xây dựng tập hợp các thẻ riêng của họ và làm cho nó có thể cho các chương trình khác hiểu những thẻ này.
- Thể hiện XML định kiểu là một thể hiện XML trong đó có một sơ đồ liên kết với nó.
- Dữ liệu XML có thể được truy vấn và lấy ra sử dụng ngôn ngữ XQuery.



- Transcorp United Inc là một công ty xuất nhập khẩu tại Mỹ. Cơ sở dữ liệu của công ty được tạo ra trong SQL Server 2012. Transcorp có khoảng 3.000 nhân viên trên toàn thế giới. Các chi tiết của nhân viên như là Mã nhân viên, Tên nhân viên, Phòng ban của nhân viên, Ngày gia nhập, và vân vân được lưu trữ trong bảng **EMP _ Details**.

Là quản trị viên cơ sở dữ liệu của Transcorp, bạn phải thực hiện các nhiệm vụ sau:

- Lấy dữ liệu của nhân viên đã gia nhập công ty trước năm 2012 và sau năm 2010.
- Chính sửa tên của một nhân viên nữ Julia Dreк thành Julia Dean sử dụng thuộc tính .WRITE.
- Lấy dữ liệu của tất cả các nhân viên đến từ Houston.

Phần - 9

Truy vấn và phép nối nâng cao

Chào mừng bạn đến với phần **Truy vấn và phép nối nâng cao**.

Phần này giải thích các kỹ thuật khác nhau để nhóm và tổng hợp dữ liệu và mô tả khái niệm về truy vấn con, biểu thức bảng, phép nối, và khám phá các toán tử tập hợp khác nhau. Phần này còn bao gồm xoay vòng và nhóm các phép tính tập hợp.

Trong phần này, bạn sẽ học để:

- Giải thích việc nhóm và tổng hợp số liệu
- Mô tả các truy vấn con
- Mô tả các biểu thức bảng
- Giải thích các phép nối
- Mô tả các loại phép nối khác nhau
- Giải thích việc sử dụng các toán tử tập hợp khác nhau để kết hợp dữ liệu
- Mô tả xoay vòng và nhóm các phép tính tập hợp

9.1 Giới thiệu

SQL Server 2012 đưa vào một số tính năng truy vấn mạnh mẽ giúp bạn lấy dữ liệu một cách hiệu quả và nhanh chóng. Dữ liệu có thể được nhóm lại và/hoặc tập hợp lại với nhau để trình bày thông tin tóm tắt. Sử dụng khái niệm các truy vấn con, một tập kết quả của `SELECT` có thể được sử dụng làm tiêu chí cho câu lệnh `SELECT` hoặc truy vấn khác. Phép nối giúp bạn kết hợp dữ liệu cột từ hai cột hoặc nhiều bảng dựa trên mối quan hệ logic giữa những bảng này. Mặt khác, các toán tử tập hợp như `UNION` và `INTERSECT` giúp bạn kết hợp dữ liệu hàng từ hai hoặc nhiều bảng. Các toán tử `PIVOT` và `UNPIVOT` được sử dụng để chuyển đổi hướng của dữ liệu từ hướng cột sang hướng hàng và ngược lại. Mệnh đề con `GROUPING SET` của mệnh đề `GROUP BY` giúp chỉ ra nhiều nhóm trong một truy vấn đơn lẻ.

9.2 Phân nhóm dữ liệu

Mệnh đề `GROUP BY` phân vùng tập kết quả vào một hoặc nhiều tập hợp con. Một tập hợp con có các giá trị và biểu thức chung. Từ khóa `GROUP BY` được theo sau bằng một danh sách các cột, được gọi là cột được nhóm. Mỗi cột được nhóm hạn chế số lượng hàng của tập kết quả. Đối với mỗi cột được nhóm, chỉ có một hàng. Mệnh đề `GROUP BY` có thể có nhiều hơn một cột được nhóm. Sau đây là cú pháp của mệnh đề `GROUP BY`.

Cú pháp:

```
CREATE TYPE [ schema_name. ] type_name { FROM base_type [ ( precision [ , scale ] ) ] [ NULL | NOT NULL ] } [ ; ]
```

trong đó:

`column_name`: là tên của cột theo đó tập kết quả nên được nhóm lại.

Xem xét bảng `WorkOrderRouting` trong cơ sở dữ liệu `AdventureWorks2012`. Tổng số giờ tài nguyên cho mỗi lệnh sản xuất cần phải được tính toán. Để đạt được điều này, các bản ghi cần phải được nhóm lại theo số thứ tự công việc, đó là `WorkOrderID`.

Đoạn mã 1 lấy và hiển thị tổng số giờ tài nguyên mỗi đơn hàng làm việc cùng với số thứ tự công việc. Trong truy vấn này, hàm dựng sẵn có tên là `SUM()` được sử dụng để tính toán tổng số. `SUM()` là một hàm tổng hợp. Các hàm tổng hợp sẽ được trình bày chi tiết trong phần sau.

Đoạn mã 1:

```
SELECT WorkOrderID, SUM(ActualResourceHrs) AS TotalHoursPerWorkOrder FROM Production.WorkOrderRouting GROUP BY WorkOrderID
```

Thực hiện truy vấn này sẽ trả lại tất cả những số lệnh công tác cùng với tổng số giờ tài nguyên cho mỗi lệnh công tác.

Một phần của kết quả được trình bày trong hình 9.1.

	WorkOrderID	TotalHoursPerWorkOrder
1	13	17.6000
2	14	17.6000
3	15	4.0000
4	16	4.0000
5	17	4.0000
6	18	4.0000
7	19	4.0000
8	20	4.0000
9	21	4.0000
10	22	4.0000

Hình 9.1: Sử dụng mệnh đề GROUP BY

Mệnh đề GROUP BY cũng có thể được sử dụng kết hợp với các mệnh đề khác. Những mệnh đề này như sau:

→ GROUP BY với WHERE

Mệnh đề WHERE cũng có thể được sử dụng với mệnh đề GROUP BY để hạn chế các hàng để gộp nhóm. Các hàng đáp ứng điều kiện tìm kiếm được xem xét để tạo nhóm. Các hàng không đáp ứng các điều kiện trong mệnh đề WHERE được loại bỏ trước khi bất kỳ gộp nhóm nào được thực hiện.

Đoạn mã 2 trình bày một truy vấn tương tự như Đoạn mã 1 nhưng hạn chế các hàng được hiển thị, bằng cách xem xét chỉ các bản ghi với WorkOrderID nhỏ hơn 50.

Đoạn mã 2:

```
SELECT WorkOrderID, SUM(ActualResourceHrs) AS TotalHoursPerWorkOrder
FROM Production.WorkOrderRouting WHERE WorkOrderID < 50 GROUP BY WorkOrderID
```

Bởi số bản ghi trả lại lớn hơn 25, một phần của kết quả được trình bày trong hình 9.2.

	WorkOrderID	TotalHoursPerWorkOrder
1	13	17.6000
2	14	17.6000
3	15	4.0000
4	16	4.0000
5	17	4.0000
6	18	4.0000
7	19	4.0000
8	20	4.0000

Hình 9.2: GROUP BY với Where

→ GROUP BY với NULL

Nếu cột gộp nhóm chứa giá trị NULL, dòng đó sẽ trở thành một nhóm riêng biệt trong tập kết quả. Nếu cột gộp nhóm có nhiều hơn một giá trị NULL, giá trị NULL được đưa vào một hàng đơn lẻ. Xem xét bảng Production.Product. Có một số hàng trong đó có các giá trị NULL trong cột Class.

Sử dụng GROUP BY trên một truy vấn cho bảng này cũng sẽ được xem xét các giá trị NULL. Ví dụ, Đoạn mã 3 lấy và hiển thị giá trị trung bình của giá niêm yết cho mỗi Class.

Đoạn mã 3:

```
SELECT Class, AVG (ListPrice) AS 'AverageListPrice' FROM
Production.Product GROUP BY Class
```

Như được trình bày trong hình 9.3, các giá trị NULL được nhóm lại thành một hàng đơn lẻ trong kết quả.

	Class	AverageListPrice
1	NULL	16.314
2	H	1679.4964
3	L	370.6887
4	M	635.5816

Hình 9.3: GROUP BY với NULL

→ GROUP BY với ALL

Cũng có thể được sử dụng từ khóa ALL với mệnh đề GROUP BY. Nó chỉ có ý nghĩa khi SELECT có mệnh đề WHERE. Khi ALL được sử dụng, nó đưa vào tất cả các nhóm mà mệnh đề GROUP BY tạo ra. Nó thậm chí bao gồm cả các nhóm mà không đáp ứng các điều kiện tìm kiếm. Sau đây là cú pháp của việc sử dụng GROUP BY với ALL.

Cú pháp:

```
SELECT <column_name> FROM <table_name> WHERE <condition> GROUP BY ALL
<column_name>
```

Xem xét bảng Sales.SalesTerritory. Bảng này có một cột tên là Group chỉ ra khu vực địa lý khi lãnh thổ bán hàng thuộc vào nơi đó. Đoạn mã 4 tính toán và hiển thị tổng doanh số cho mỗi nhóm. Kết quả cần để hiển thị tất cả các nhóm cho dù họ có bất kỳ doanh số nào hay không. Để đạt được điều này, đoạn mã sử dụng GROUP BY với ALL.

Đoạn mã 4:

```
SELECT [Group], SUM(SalesYTD) AS 'TotalSales' FROM Sales.SalesTerritory
WHERE [Group] LIKE 'N%' OR [Group] LIKE 'E%' GROUP BY ALL [Group]
```

Ngoài các hàng được hiển thị trong Đoạn mã 4, nó cũng sẽ hiển thị nhóm 'Pacific' với các giá trị null như được trình bày trong hình 9.4. Điều này là do khu vực Thái Bình Dương không có bất kỳ doanh số nào.

	Group	TotalSales
1	Europe	13590506.0212
2	North America	33182889.0168
3	Pacific	NULL

Hình 9.4: GROUP BY với ALL

→ GROUP BY với HAVING

Mệnh đề HAVING chỉ được sử dụng với câu lệnh SELECT để chỉ ra một điều kiện tìm kiếm cho một nhóm. Mệnh đề HAVING hoạt động như mệnh đề WHERE ở những nơi mệnh đề WHERE không thể được sử dụng đối với các hàm tổng hợp như là SUM(). Một khi bạn đã tạo ra các nhóm với mệnh đề GROUP BY, bạn có thể muốn lọc các kết quả hơn nữa. Mệnh đề HAVING hoạt động như một bộ lọc trên các nhóm, tương tự như cách mệnh đề WHERE hoạt động như một bộ lọc trên các hàng trả về từ mệnh đề FROM. Sau đây là cú pháp của GROUP BY với HAVING.

Cú pháp:

```
SELECT <column_name> FROM <table_name> GROUP BY <column_name> HAVING
<search_condition>
```

Đoạn mã 5 hiển thị hàng với nhóm 'Pacific' vì nó có tổng doanh thu ít hơn 6000000.

Đoạn mã 5:

```
SELECT [Group], SUM(SalesYTD) AS 'TotalSales' FROM Sales.SalesTerritory
WHERE [Group] LIKE 'N%' OR [Group] LIKE 'E%' GROUP BY ALL [Group]
```

Đầu ra của mã này là hàng duy nhất, với tên Group là Pacific và tổng doanh thu là 5.977.814,9154.

9.3 Tổng kết dữ liệu

Mệnh đề GROUP BY còn sử dụng các toán tử như là CUBE và ROLLUP để trả lại dữ liệu tóm tắt. Số lượng các cột trong mệnh đề GROUP BY xác định số hàng tóm tắt trong tập kết quả. Các toán tử được mô tả như sau:

- **CUBE:** CUBE là một toán tử tổng hợp tạo ra một hàng siêu tổng hợp. Ngoài các hàng thông thường được cung cấp bằng GROUP BY, nó còn cung cấp bản tóm tắt các hàng mà mệnh đề GROUP BY tạo ra. Hàng tóm tắt được hiển thị cho tất cả các tổ hợp có thể có của các nhóm trong tập kết quả. Hàng tóm tắt hiển thị NULL trong tập kết quả nhưng đồng thời trả về tất cả những giá trị cho những cái này. Sau đây là cú pháp của CUBE.

Cú pháp:

```
SELECT <column_name> FROM <table_name> GROUP BY <column_name> WITH CUBE
```

Đoạn mã 6 trình bày việc sử dụng CUBE.

Đoạn mã 6:

```
SELECT Name, CountryRegionCode, SUM(SalesYTD) AS TotalSales FROM Sales.
SalesTerritory WHERE Name <> 'Australia' AND Name <> 'Canada' GROUP BY
Name, CountryRegionCode WITH CUBE
```

Đoạn mã 6 lấy và hiển thị tổng doanh thu của mỗi quốc gia và ngoài ra, tổng doanh thu của tất cả các vùng của các nước.

Kết quả được trình bày trong hình 9.5.

	Name	CountryRegionCode	TotalSales
1	Germany	DE	3805202.3478
2	NULL	DE	3805202.3478
3	France	FR	4772398.3078
4	NULL	FR	4772398.3078
5	United Kingdom	GB	5012905.3656
6	NULL	GB	5012905.3656
7	Central	US	3072175.118
8	Northeast	US	2402176.8476
9	Northwest	US	7887186.7882
10	Southeast	US	2538667.2515

Hình 9.5: Sử dụng Group By với CUBE

- **ROLLUP:** Ngoài những hàng thông thường được tạo ra bằng GROUP BY, nó còn đưa các hàng tóm tắt vào tập kết quả. Nó tương tự như toán tử CUBE, nhưng tạo ra một tập kết quả cho thấy các nhóm được sắp xếp theo một trật tự thứ bậc. Nó sắp xếp các nhóm từ thấp lên cao. Hệ thống phân cấp nhóm trong kết quả phụ thuộc vào thứ tự trong đó các cột được nhóm lại được chỉ ra.

Sau đây là cú pháp của ROLLUP.

Cú pháp:

```
SELECT <column_name> FROM <table_name> GROUP BY <column_name> WITH ROLLUP
```

Đoạn mã 7 trình bày việc sử dụng ROLLUP. Nó lấy và hiển thị tổng doanh thu của mỗi quốc gia và ngoài ra, tổng doanh thu của tất cả các vùng của các nước và sắp xếp chúng theo thứ tự.

Đoạn mã 7:

```
SELECT Name, CountryRegionCode, SUM(SalesYTD) AS TotalSales
FROM Sales.SalesTerritory
WHERE Name <> 'Australia' AND Name <> 'Canada'
GROUP BY Name, CountryRegionCode
WITH ROLLUP
```

Kết quả được trình bày trong hình 9.6.

	Name	TotalSales
1	Australia	5977814.9154
2	Canada	6771829.1376
3	Central	3072175.118
4	France	4772398.3078
5	Germany	3805202.3478
6	Northeast	2402176.8476
7	Northwest	7887186.7882
8	Southeast	2538667.2515
9	Southwest	10510853.8...
10	United Kingdom	5012905.3656
11	NULL	52751209.9...

Hình 9.6: Sử dụng Group By với ROLLUP

9.4 Các hàm tổng hợp

Thỉnh thoảng, các nhà phát triển cũng có thể cần phải thực hiện phân tích trên các hàng, chẳng hạn như đếm số hàng đáp ứng các tiêu chí cụ thể hoặc tóm tắt tổng doanh số cho tất cả các đơn đặt hàng. Các hàm tổng hợp cho phép thực hiện điều này.

Do các hàm tổng hợp trả lại một giá trị duy nhất, chúng có thể được sử dụng trong các câu lệnh SELECT nơi biểu thức đơn lẻ được sử dụng, như là các mệnh đề SELECT, HAVING, và ORDER BY. Các hàm tổng hợp bỏ qua NULL, ngoại trừ khi sử dụng COUNT(*) .

Các hàm tổng hợp trong danh sách SELECT không tạo ra bí danh cột. Bạn có thể muốn sử dụng mệnh đề AS để cung cấp một cái.

Các hàm tổng hợp trong mệnh đề SELECT hoạt động trên tất cả các hàng được truyền vào giai đoạn SELECT. Nếu không có mệnh đề GROUP BY, tất cả các hàng sẽ được tóm tắt lại.

SQL Server cung cấp nhiều hàm tổng hợp dựng sẵn. Các hàm thường sử dụng được đưa vào trong bảng 9.1:

Tên hàm	Cú pháp	Mô tả
AVG	AVG(<biểu thức>)	Tính giá trị trung bình của tất cả các giá trị số không phải là NULL trong một cột.
COUNT or COUNT_BIG	COUNT(*) hoặc COUNT(<biểu thức>)	Khi (*) được sử dụng, hàm này đếm tất cả các hàng, bao gồm cả những hàng có NULL. Hàm trả về số hàng không phải là NULL cho cột đó khi một cột được quy định là <biểu thức>. Giá trị trả về của hàm COUNT là int. Giá trị trả về của COUNT_BIG là một big_int.

Tên hàm	Cú pháp	Mô tả
MAX	MAX(<biểu thức>)	Trả về số lớn nhất, ngày/giờ mới nhất, hoặc chuỗi xảy ra cuối cùng.
MIN	MIN (<biểu thức>)	Trả về số nhỏ nhất, ngày/giờ mới nhất, hoặc chuỗi xảy ra đầu tiên.
SUM	SUM (<biểu thức>)	Tính tổng của tất cả các giá trị số không phải là NULL trong một cột.

Bảng 9.1: Các hàm tổng hợp thường được sử dụng

Để sử dụng tổng hợp dựng sẵn trong mệnh đề SELECT, xem xét truy vấn sau đây trong Đoạn mã 8.

Đoạn mã 8:

```
SELECT AVG([UnitPrice]) AS AvgUnitPrice,
MIN([OrderQty]) AS MinQty,
MAX([UnitPriceDiscount]) AS MaxDiscount
FROM Sales.SalesOrderDetail;
```

Do truy vấn không sử dụng mệnh đề GROUP BY, tất cả các hàng trong bảng sẽ được tóm tắt bằng các công thức tổng hợp trong mệnh đề SELECT.

Kết quả được trình bày trong hình 9.7.

	AvgUnitPrice	MinQty	MaxDiscount
1	465.0934	1	0.40

Hình 9.7: Sử dụng các hàm tổng hợp

Khi sử dụng các giá trị tổng hợp trong mệnh đề SELECT, tất cả các cột được tham chiếu trong danh sách SELECT phải được sử dụng làm đầu vào cho hàm tổng hợp hoặc phải được tham chiếu trong mệnh đề GROUP BY. Không làm này, sẽ có một lỗi. Ví dụ, truy vấn trong Đoạn mã 9 sẽ trả về một lỗi.

Đoạn mã 9:

```
SELECT SalesOrderID, AVG(UnitPrice) AS AvgPrice
FROM Sales.SalesOrderDetail;
```

Cái này trả về một lỗi chỉ ra rằng cột Sales.SalesOrderDetail.SalesOrderID không hợp lệ trong danh sách SELECT bởi vì nó không được chứa trong hàm tổng hợp hoặc mệnh đề GROUP BY. Bởi truy vấn không sử dụng mệnh đề GROUP BY, tất cả các hàng sẽ được đối xử như một nhóm duy nhất. Do đó, tất cả các cột phải được sử dụng làm đầu vào cho các hàm tổng hợp. Để sửa chữa hoặc ngăn chặn lỗi, cần phải loại bỏ SalesOrderID khỏi truy vấn.

Ngoài việc sử dụng dữ liệu số, biểu thức tổng hợp cũng có thể bao gồm ngày, giờ, và dữ liệu ký tự để tóm tắt.

Đoạn mã 10 trả về ngày đặt hàng sớm nhất và mới nhất, sử dụng MIN và MAX.

Đoạn mã 10:

```
SELECT MIN(OrderDate) AS Earliest,
       MAX(OrderDate) AS Latest
  FROM Sales.SalesOrderHeader;
```

Hình 9.8 trình bày kết quả.

	Earliest	Latest
1	2005-07-01 00:00:00.000	2008-07-31 00:00:00.000

Hình 9.8: Sử dụng các hàm tổng hợp với dữ liệu không phải là số

Các hàm khác cũng có thể được sử dụng kết hợp với các hàm tổng hợp.

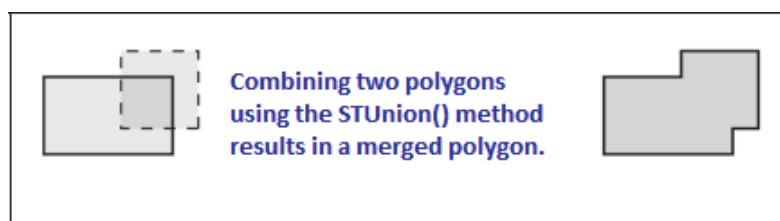
9.5 Các tập hợp không gian

SQL Server cung cấp một số phương pháp có thể giúp tổng hợp hai mục riêng lẻ của dữ liệu hình học hoặc địa lý. Những phương thức này được liệt kê trong hình 9.2.

P h ư ơ n g thức	Mô tả
STUnion	Trả về một đối tượng đại diện cho phép hợp của một thể hiện hình học/địa lý với một thể hiện hình học/địa lý khác.
STIntersection	Trả về một đối tượng đại diện cho các điểm nơi một thể hiện hình học/địa lý giao cắt với một thể hiện hình học/địa lý khác.
STConvexHull	Trả về một đối tượng trình bày vỏ lồi của một thể hiện hình học/địa lý. Tập hợp các điểm được gọi là lồi nếu đối với bất kỳ hai điểm nào, toàn bộ đoạn được chứa trong tập hợp. Vỏ lồi của tập hợp điểm là tập lồi nhỏ nhất chứa tập hợp này. Đối với bất kỳ tập hợp các điểm nào, chỉ có một vỏ lồi.

Bảng 9.2: Phương pháp tổng hợp không gian

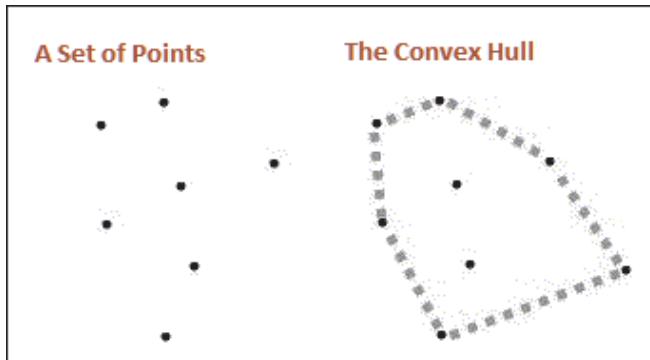
Hình 9.9, 9.10, và 9.11 mô tả trực quan ví dụ về những phương pháp này.



Hình 9.9: STUnion()



Hình 9.10: `STIntersection()`



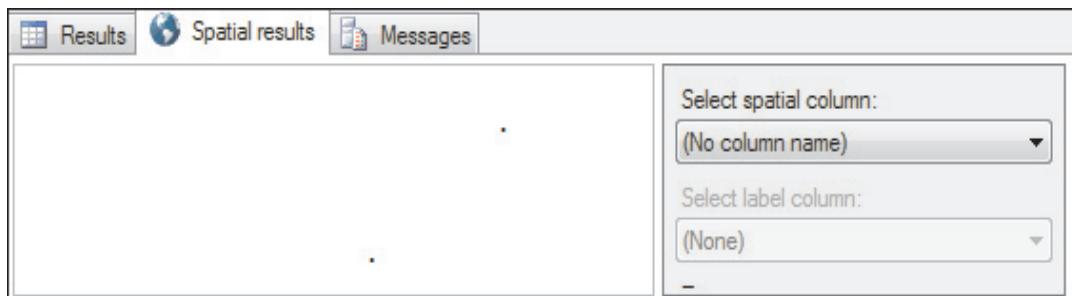
Hình 9.11: `STConvexHull()`

Đoạn mã 11 trình bày việc sử dụng `STUnion()`.

Đoạn mã 11:

```
SELECT geometry::Point(251, 1, 4326).STUnion(geometry::Point(252, 2, 4326));
```

Kết quả được trình bày trong hình 9.12. Nó trình bày hai điểm.



Hình 9.12: Sử dụng `STUnion()` với một loại hình học

Một ví dụ khác được đưa ra trong Đoạn mã 12.

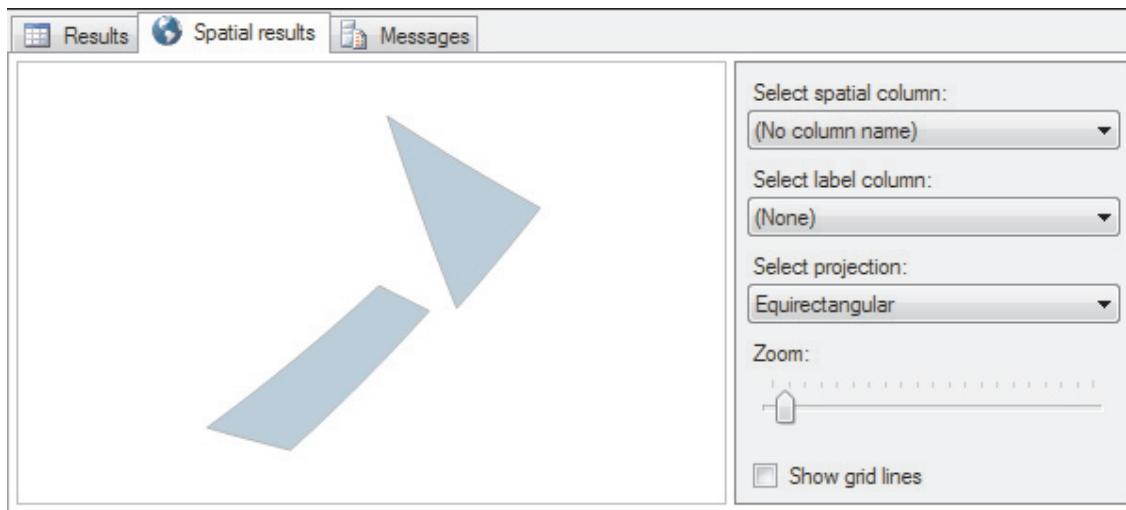
Đoạn mã 12:

```
DECLARE @City1 geography
SET @City1 = geography::STPolyFromText(
    'POLYGON((175.3 -41.5, 178.3 -37.9, 172.8 -34.6, 175.3 -41.5))',
    4326)
DECLARE @City2 geography
```

```
SET @City2=geography::STPolyFromText(
'POLYGON((169.3-46.6, 174.3-41.6, 172.5-40.7, 166.3-45.8, 169.3-46.6) ', 
4326)
DECLARE @CombinedCity geography=@City1.STUnion(@City2)
SELECT @CombinedCity
```

Ở đây, hai biến được khai báo thuộc loại `geography` và giá trị thích hợp được gán cho chúng. Sau đó, chúng được kết hợp thành một biến thứ ba thuộc loại `geography` bằng cách sử dụng phương pháp `STUnion()`.

Kết quả của đoạn mã này được trình bày trong hình 9.13.



Hình 9.13: Sử dụng `STUnion()` với một loại địa lý

9.5.1 Các tổng hợp không gian mới

Thật dễ dàng để tính toán phép hợp của dữ liệu số thông thường bằng cách sử dụng các toán tử cơ bản với các truy vấn như là `SELECT x + y` hoặc bằng cách sử dụng toán tử `UNION`. Bạn có thể tính toán phép hợp của hai hình học riêng lẻ hoặc hai khu vực địa lý bằng cách sử dụng toán tử `STUnion()`. Điều gì xảy ra nếu có nhu cầu phải tính toán phép hợp của một tập hợp các đối tượng hình học/địa lý hoặc tất cả các phần tử trong một cột không gian? Điều gì xảy ra nếu có nhu cầu phải tìm ra giá trị trung bình của một tập hợp các phần tử Point? Sẽ không thể sử dụng hàm `AVG()` ở đây. Trong trường hợp này, bạn sẽ sử dụng các hàm tổng hợp không gian mới trong SQL Server 2012.

SQL Server 2012 đã giới thiệu bốn tổng hợp mới cho bộ toán tử không gian trong SQL Server:

- ➔ Tổng hợp phép hợp
- ➔ Tổng hợp phong bì
- ➔ Tổng hợp bộ sưu tập

➔ **Tổng hợp vỏ lồi**

Những tổng hợp này được thực hiện như là các phương pháp tĩnh, làm việc cho một trong kiểu dữ liệu geography hoặc geometry. Mặc dù các tổng hợp được áp dụng cho tất cả các lớp dữ liệu không gian, chúng có thể được mô tả tốt nhất với đa giác.

➔ **Tổng hợp phép hợp**

Nó thực hiện phép hợp trên một tập hợp các đối tượng geometry. Nó kết hợp nhiều đối tượng không gian vào một đối tượng không gian, loại bỏ các ranh giới bên trong, nếu có thể áp dụng. Sau đây là cú pháp của UnionAggregate.

Cú pháp:

```
UnionAggregate ( geometry_operand hoặc geography_operand)
```

trong đó:

geometry_operand: là cột bảng loại geometry bao gồm tập hợp các đối tượng geometry trên đó có một phép hợp sẽ được thực hiện.

geography_operand: là cột bảng loại geography bao gồm tập hợp các đối tượng geography trên đó có một phép hợp sẽ được thực hiện.

Đoạn mã 13 trình bày ví dụ đơn giản của việc sử dụng tổng hợp Union. Nó sử dụng bảng Person.Address trong cơ sở dữ liệu AdventureWorks2012.

Đoạn mã 13:

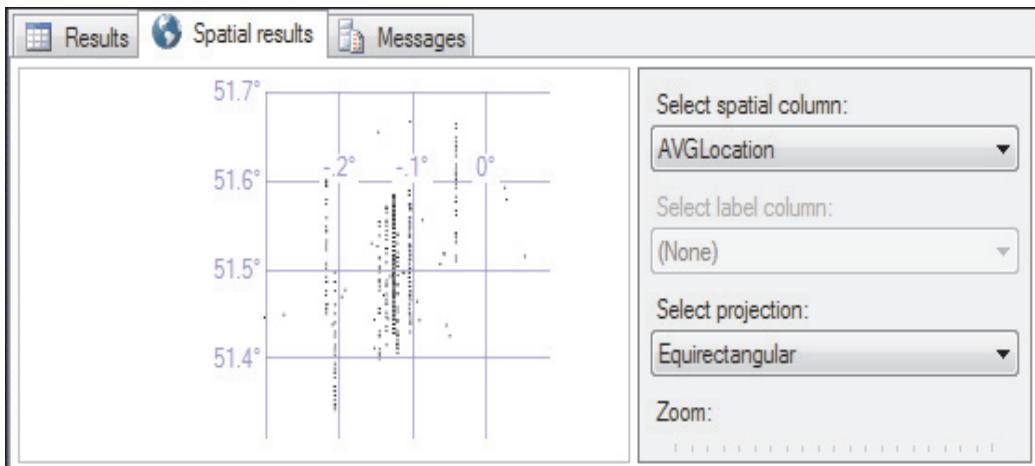
```
SELECT Geography::UnionAggregate(SpatialLocation)
AS AVGLocation
FROM Person.Address
WHERE City = 'London';
```

Kết quả của mã này được trình bày trong hình 9.14.

AVGLocation	
1	0xE61000000104A00100003DA82EC605C249407D37109CAD...

Hình 9.14: Sử dụng các tập hợp không gian

Để xem một phần trình bày trực quan của dữ liệu không gian, bạn có thể bấm vào tab **Spatial results** trong cửa sổ đầu ra. Điều này sẽ hiển thị đầu ra như được trình bày trong hình 9.15.



Hình 9.15: Xem các kết quả không gian

➔ Tổng hợp phong bì

Nó trả về một khu vực giới hạn cho một tập hợp đã cho của các đối tượng geometry hoặc geography.

Tổng hợp phong bì thể hiện các hành vi khác nhau cho các loại geography và geometry. Dựa trên loại đối tượng được áp dụng, nó sẽ trả về các kết quả khác nhau. Đối với loại geometry, kết quả là một đa giác hình chữ nhật “truyền thống”, gắn chặt chẽ các đối tượng đầu vào đã chọn. Đối với loại geography, kết quả là một đối tượng hình tròn, gắn lỏng lẻo các đối tượng đầu vào đã chọn. Hơn nữa, đối tượng hình tròn được định nghĩa sử dụng tính năng `CurvePolygon` mới. Sau đây là cú pháp của `EnvelopeAggregate`.

Cú pháp:

```
EnvelopeAggregate (geometry_operand hoặc geography_operand)
```

trong đó:

`geometry_operand`: là cột bảng loại geometry bao gồm tập hợp các đối tượng geometry.

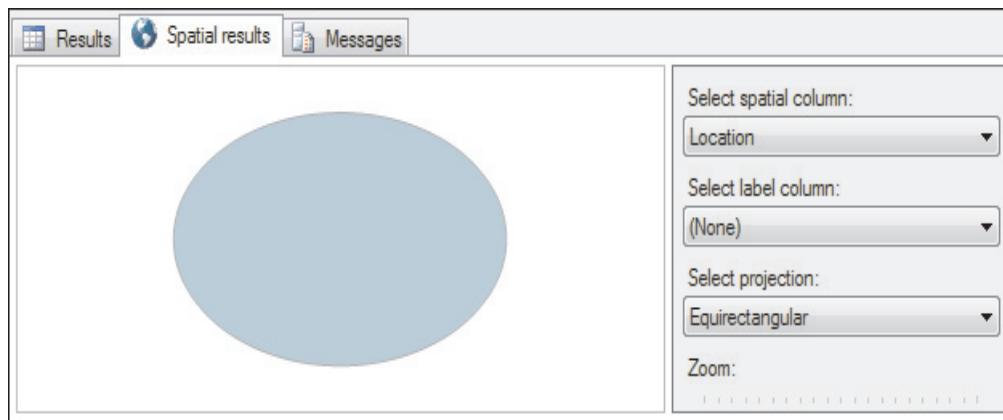
`geography_operand`: là cột bảng loại geography bao gồm tập hợp các đối tượng geography.

Đoạn mã 14 trả về một khung giới hạn cho tập hợp các đối tượng trong một cột biến bảng.

Đoạn mã 14:

```
SELECT Geography::EnvelopeAggregate(SpatialLocation)
AS Location
FROM Person.Address
WHERE City = 'London'
```

Trình bày trực quan của kết quả này được trình bày trong hình 9.16.



Hình 9.16: EnvelopeAggregate

→ Tổng hợp bộ sưu tập

Nó trả về thể hiện GeometryCollection/GeographyCollection với một phần geometry/geography cho từng đối tượng không gian trong tập lựa chọn. Sau đây là cú pháp của CollectionAggregate.

Cú pháp:

```
CollectionAggregate (geometry_operand hoặc geography_operand)
```

trong đó:

`geometry_operand`: là cột bảng loại geometry bao gồm tập hợp các đối tượng geometry.

`geography_operand`: là cột bảng loại geography bao gồm tập hợp các đối tượng geography.

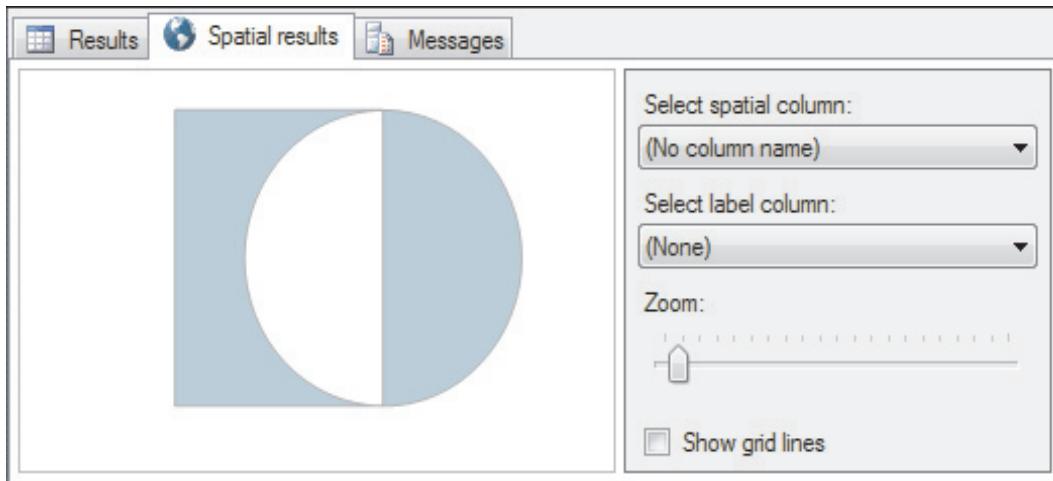
Đoạn mã 15 trả về thể hiện GeometryCollection có chứa CurvePolygon và Polygon.

Đoạn mã 15:

```
DECLARE @CollectionDemo TABLE
(
shape geometry,
shapeType nvarchar(50)
)
INSERT INTO @CollectionDemo(shape, shapeType) VALUES ('CURVEPOLYGON(CIRC
ULARSTRING(2 3, 4 1, 6 3, 4 5, 2 3))', 'Circle'),
('POLYGON((1 1, 4 1, 4 5, 1 5, 1 1))', 'Rectangle');

SELECT geometry::CollectionAggregate(shape)
FROM @CollectionDemo;
```

Kết quả của đoạn mã này được trình bày trong hình 9.17.



Hình 9.17: Sử dụng CollectionAggregate

→ Tổng hợp vỏ lồi

Nó trả về một đa giác vỏ lồi, bao quanh một hoặc nhiều đối tượng không gian cho một tập hợp đã cho của các đối tượng geometry/geography. Sau đây là cú pháp của ConvexHullAggregate.

Cú pháp:

```
ConvexHullAggregate (geometry_operand hoặc geography_operand)
```

trong đó:

geometry_operand: là cột bảng loại geometry bao gồm tập hợp các đối tượng geometry.

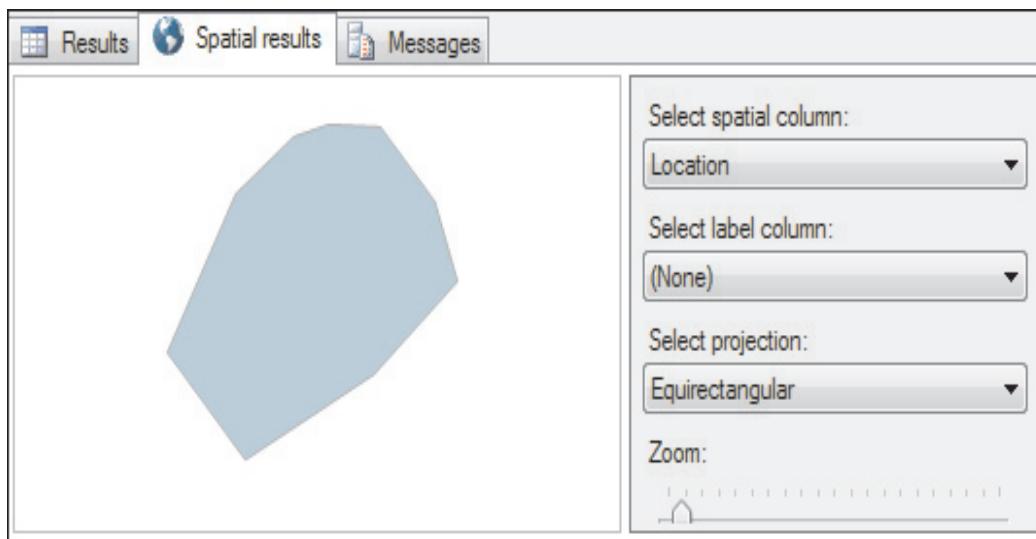
geography_operand: là cột bảng loại geography bao gồm tập hợp các đối tượng geography.

Đoạn mã 16 trình bày việc sử dụng ConvexHullAggregate.

Đoạn mã 16:

```
SELECT Geography::ConvexHullAggregate(SpatialLocation)
AS Location
FROM Person.Address
WHERE City = 'London'
```

Kết quả được trình bày trong hình 9.18.



Hình 9.18: Sử dụng ConvexHullAggregate

9.6 Các truy vấn con

Bạn có thể sử dụng câu lệnh SELECT hoặc một truy vấn để trả lại các bản ghi sẽ được sử dụng làm tiêu chí cho câu lệnh SELECT hoặc truy vấn khác. Truy vấn bên ngoài được gọi là truy vấn cha và truy vấn bên trong được gọi là truy vấn con. Mục đích của một truy vấn con là để trả về các kết quả cho truy vấn bên ngoài. Nói cách khác, câu lệnh truy vấn bên trong nên trả về cột này hoặc các cột được sử dụng trong tiêu chí của câu lệnh truy vấn bên ngoài.

Hình thức đơn giản nhất của một truy vấn con là cái trả về chỉ một cột. Truy vấn cha có thể sử dụng các kết quả của truy vấn con này dùng dấu =.

Cú pháp cho dạng cơ bản nhất của truy vấn con dùng chỉ một cột với dấu = được hiển thị.

Cú pháp:

```
SELECT <ColumnName> FROM <table>
WHERE <ColumnName> = ( SELECT <ColumnName> FROM <Table> WHERE <ColumnName> =
<Condition> )
```

Trong một truy vấn con, câu lệnh SELECT trong cùng được thực hiện đầu tiên và kết quả của nó được chuyển làm tiêu chí cho câu lệnh SELECT bên ngoài.

Xem xét một kịch bản trong đó cần phải xác định ngày đến hạn và ngày chuyển hàng cho các đơn đặt hàng gần đây nhất.

Đoạn mã 17 trình bày đoạn mã này để đạt được điều này.

Đoạn mã 17:

```
SELECT DueDate, ShipDate
FROM Sales.SalesOrderHeader
WHERE Sales.SalesOrderHeader.OrderDate =
    (SELECT MAX(OrderDate)
FROM Sales.SalesOrderHeader)
```

Ở đây, truy vấn con đã được sử dụng để đạt được kết quả mong muốn. Truy vấn hoặc truy vấn con bên trong lấy ngày đặt hàng gần đây nhất. Sau đó cái này được truyền cho truy vấn bên ngoài, hiển thị ngày đến hạn và ngày chuyển hàng cho tất cả các đơn đặt hàng đã được đặt vào ngày cụ thể đó.

Một phần của kết quả của đoạn mã này được trình bày trong hình 9.19.

	DueDate	ShipDate
1	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
2	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
4	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
5	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
6	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
7	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
8	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
9	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
10	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000

Hình 9.19: Sử dụng một truy vấn con đơn giản

Dựa trên kết quả trả về của truy vấn bên trong, một truy vấn con có thể được phân loại là một truy vấn con vô hướng hoặc một truy vấn con có nhiều giá trị. Những công cụ này được mô tả như sau:

- ➔ Truy vấn con vô hướng trả lại một giá trị duy nhất. Ở đây, truy vấn bên ngoài cần phải được viết để xử lý một kết quả duy nhất.
- ➔ Truy vấn con có nhiều giá trị trả về kết quả tương tự như một bảng có một cột. Ở đây, truy vấn bên ngoài cần phải được viết để xử lý nhiều kết quả khả năng.

9.6.1 Làm việc với các truy vấn nhiều giá trị

Nếu toán tử = được sử dụng với truy vấn con, truy vấn con phải trả lại một giá trị vô hướng duy nhất. Nếu có nhiều hơn một giá trị được trả về, sẽ có lỗi và truy vấn sẽ không được xử lý. Trong kịch bản như vậy, những từ khóa ANY, ALL, IN, và EXISTS có thể được sử dụng với mệnh đề WHERE của câu lệnh SELECT khi truy vấn trả về một cột nhưng có một hoặc nhiều hàng.

Những từ khóa này, còn được gọi là các xác nhận, được sử dụng với các truy vấn có nhiều giá trị.

Ví dụ, hãy xem xét tất cả các tên họ và tên gọi của người lao động có chức danh công việc là 'Giám đốc Nghiên cứu và Phát triển' cần được hiển thị. Ở đây, truy vấn bên trong có thể trả về nhiều hơn một dòng bởi có thể có nhiều hơn một nhân viên với chức danh công việc đó. Để đảm bảo rằng truy vấn bên ngoài có thể sử dụng các kết quả của truy vấn bên trong này, từ khóa IN sẽ phải được sử dụng.

Đoạn mã 18 trình bày phương thức này.

Đoạn mã 18:

```
SELECT FirstName, LastName FROM Person.Person
WHERE Person.Person.BusinessEntityID IN (SELECT BusinessEntityID
FROM HumanResources.Employee WHERE JobTitle = 'Research and Development
Manager');
```

Ở đây, truy vấn bên trong lấy BusinessEntityID từ bảng HumanResources.Employee cho những bản ghi có chức danh công việc là 'Giám đốc Nghiên cứu và Phát triển'. Những kết quả này sau đó được truyền đến cho truy vấn bên ngoài, so khớp BusinessEntityID với cái trong bảng Person.Person. Cuối cùng, từ các bản ghi đang so khớp, tên họ và tên gọi được trích xuất và hiển thị.

Kết quả được hiển thị trong hình 9.20.

	FirstName	LastName
1	Dylan	Miller
2	Michael	Raheem

Hình 9.20: Kết quả của truy vấn phụ với từ khóa IN

Những từ khóa SOME hoặc ANY đánh giá thành true nếu kết quả là một truy vấn bên trong có chứa ít nhất một hàng đáp ứng sự so sánh này. Chúng so sánh một giá trị vô hướng với một cột các giá trị. SOME và ANY là tương đương, cả hai trả về cùng một kết quả. Chúng ít khi được sử dụng.

Có một số hướng dẫn cần phải được tuân thủ khi làm việc với các truy vấn con. Bạn nên nhớ các điểm sau đây khi sử dụng các truy vấn con:

- ➔ Không thể sử dụng kiểu dữ liệu ntext, text, và image trong danh sách các truy vấn con SELECT.
- ➔ Danh sách các truy vấn con SELECT được giới thiệu với toán tử so sánh có thể chỉ có một biểu thức hoặc tên cột.
- ➔ Truy vấn con được giới thiệu bằng toán tử so sánh không theo sau bằng từ khóa ANY hoặc ALL không thể đưa vào mệnh đề GROUP BY và HAVING.
- ➔ Bạn không thể sử dụng từ khóa DISTINCT với các truy vấn con bao gồm GROUP BY.

- Bạn có thể chỉ ra ORDER BY chỉ khi TOP cũng được chỉ ra.

Bên cạnh các truy vấn con vô hướng và có nhiều giá trị, bạn cũng có thể lựa chọn giữa các truy vấn con khép kín và truy vấn con tương quan. Điều này được định nghĩa như sau:

- Truy vấn con khép kín được viết làm truy vấn độc lập, không có bất kỳ phụ thuộc nào vào truy vấn bên ngoài. Truy vấn con khép kín được xử lý một lần khi truy vấn bên ngoài chạy và chuyển các kết quả của nó cho truy vấn bên ngoài.
- Truy vấn con tương quan tham khảo một hoặc nhiều cột từ truy vấn bên ngoài và do đó, phụ thuộc vào truy vấn bên ngoài. Truy vấn con tương quan không thể chạy riêng rẽ với truy vấn bên ngoài.

Từ khóa EXISTS được sử dụng với một truy vấn con để kiểm tra sự tồn tại của các hàng được trả về từ truy vấn con đó. Truy vấn con không thực sự trả về bất kỳ dữ liệu nào, nó trả về giá trị TRUE hoặc FALSE.

Sau đây là cú pháp của một truy vấn con có chứa từ EXISTS.

Cú pháp:

```
SELECT <ColumnName> FROM <table>
WHERE [NOT] EXISTS
(
    <Subquery_Statement>
)
```

trong đó:

Subquery_Statement: chỉ ra truy vấn con này.

Mã trong Đoạn mã 18 có thể được viết lại như được trình bày trong Đoạn mã 19 sử dụng từ khóa EXISTS để mang lại cùng một kết quả.

Đoạn mã 19:

```
SELECT FirstName, LastName FROM Person.Person AS A
WHERE EXISTS (SELECT *
FROM HumanResources.Employee AS B WHERE JobTitle = 'Research and Development
Manager' AND A.BusinessEntityID=B.BusinessEntityID);
```

Ở đây, truy vấn con bên trong lấy tất cả các bản ghi so khớp chức danh công việc là 'Giám đốc Nghiên cứu và Phát triển' và BusinessEntityId khớp với cái trong bảng Person. Nếu không có các bản ghi phù hợp với cả hai điều kiện này, truy vấn con bên trong sẽ không trả về bất kỳ hàng nào. Như vậy, trong trường hợp đó, EXISTS sẽ trả về false và truy vấn bên ngoài cũng sẽ không trả về bất kỳ hàng nào. Tuy nhiên, mã trong Đoạn mã 19 sẽ trả về hai hàng vì những điều kiện đã cho được thỏa mãn. Kết quả sẽ giống như hình 9.20.

Tương tự như vậy, người ta có thể sử dụng từ khóa NOT EXISTS. Mệnh đề WHERE trong đó nó được sử dụng được thỏa mãn nếu không có hàng được trả về từ truy vấn con này.

9.6.2 Các truy vấn con xếp lồng

Truy vấn con mà được được định nghĩa bên trong truy vấn con khác được gọi là truy vấn con xếp lồng.

Xem xét rằng bạn muốn lấy ra và hiển thị tên của những người từ Canada. Không có cách trực tiếp nào để lấy thông tin này từ bảng Sales.SalesTerritory không liên quan đến bảng Person.Person. Do đó, truy vấn con lồng nhau được sử dụng ở đây như được trình bày trong Đoạn mã 20.

Đoạn mã 20:

```
SELECT LastName, FirstName
FROM Person.Person
WHERE BusinessEntityID IN
    (SELECT BusinessEntityID
    FROM Sales.SalesPerson
    WHERE TerritoryID IN
        (SELECT TerritoryID
        FROM Sales.SalesTerritory
        WHERE Name='Canada')
    )
```

Kết quả được trình bày trong hình 9.21.

	Last Name	First Name
1	Vargas	Gamett
2	Saraiwa	José

Hình 9.21: Đầu ra của các truy vấn con lồng nhau

9.6.3 Các truy vấn tương quan

Trong nhiều truy vấn có chứa các truy vấn con, truy vấn con cần được cho giá trị chỉ một lần để cung cấp các giá trị mà truy vấn cha cần. Điều này là do trong hầu hết những truy vấn, truy vấn con làm cho không có tham chiếu đến truy vấn cha mẹ, vì vậy giá trị trong truy vấn con vẫn không đổi.

Tuy nhiên, nếu truy vấn con tham chiếu tới một truy vấn cha, truy vấn con cần được đánh giá lại cho mỗi bước lặp trong truy vấn cha. Điều này là do tiêu chuẩn tìm kiếm trong truy vấn con phụ thuộc vào giá trị của một bản ghi cụ thể trong truy vấn cha.

Khi truy vấn con lấy các tham số từ truy vấn cha của nó, cái này được biết như là truy vấn con tương quan. Xem xét rằng bạn muốn lấy tất cả các id thực thể kinh doanh của những người có thông tin liên lạc được sửa đổi lần cuối cùng không sớm hơn năm 2012. Để làm điều này, bạn có thể sử dụng truy vấn con tương quan như được trình bày trong Đoạn mã 21.

Đoạn mã 21:

```
SELECT e.BusinessEntityID
FROM Person.BusinessEntityContact e
WHERE e.ContactTypeID IN
(
    SELECT c.ContactTypeID
    FROM Person.ContactType c
    WHERE YEAR(e.ModifiedDate) >= 2012
)
```

Trong Đoạn mã 21, truy vấn bên trong lấy các mã loại liên hệ cho tất cả những người có thông tin liên lạc đã được sửa đổi vào hoặc trước năm 2012. Những kết quả này sau đó được truyền cho truy vấn bên ngoài, so khớp những mã loại liên hệ với những người trong bảng Person.BusinessEntityContact và hiển thị các mã thực thể kinh doanh của những bản ghi đó. Hình 9.22 trình bày một phần của kết quả.

	BusinessEntityID
1	292
2	294
3	296
4	298
5	300
6	302
7	304

Hình 9.22: Kết quả của các truy vấn tương quan

9.7 Các phép nối

Các phép nối được dùng để gọi ra dữ liệu từ hai hoặc nhiều bảng dữ liệu mối quan hệ logic giữa các bảng. Phép nối thông thường chỉ ra mối quan hệ khóa ngoại giữa các bảng. Nó định nghĩa cách thức trong đó hai bảng được liên kết vào một truy vấn bằng cách:

- Chỉ ra cột từ mỗi bảng để được dùng cho phép nối. Phép nối điển hình chỉ ra khóa ngoại từ một bảng và khóa được liên kết của nó trong bảng kia.
- Chỉ ra một toán tử logic như là =, <> để được sử dụng trong việc so sánh các giá trị từ những cột này.

Phép nối có thể được quy định trong mệnh đề FROM hoặc WHERE.

Sau đây là cú pháp của câu lệnh JOIN.

Cú pháp:

```
SELECT <ColumnName1>, <ColumnName2>...<ColumnNameN>
FROM Table_A AS Table_Alias_A
JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn>=Table_Alias_B.<CommonColumn>
```

trong đó:

<ColumnName1>, <ColumnName2>: Là một danh sách các cột cần được hiển thị.

Table_A: Là tên của bảng bên trái của từ khóa JOIN.

Table_B: Là tên của bảng bên phải của từ khóa JOIN.

AS Table_Alias: Là cách gán tên bí danh cho bảng. Có thể sử dụng bí danh được định nghĩa cho bảng để biểu thị bảng để không cần phải dùng tên đầy đủ của bảng.

<CommonColumn>: Là cột chung cho cả hai bảng. Trong trường hợp này, việc nối chỉ thành công khi các cột có các giá trị so khớp.

Xem xét rằng bạn muốn liệt kê tên họ, tên gọi của nhân viên và các chức danh công việc của họ từ HumanResources.Employee và Person.Person. Để trích xuất thông tin này từ hai bảng, bạn cần phải nối chúng lại dựa trên BusinessEntityID như được trình bày trong Đoạn mã 22.

Đoạn mã 22:

```
SELECT A.FirstName, A.LastName, B.JobTitle
FROM Person.Person A
JOIN
HumanResources.Employee B
ON
A.BusinessEntityID=B.BusinessEntityID;
```

Ở đây, các bảng HumanResources.Employee và Person.Person được cho bí danh A và B. Chúng được nối với nhau trên cơ sở các id thực thể kinh doanh của chúng. Câu lệnh SELECT sau đó lấy các cột mong muốn thông qua các bí danh.

Hình 9.23 trình bày kết quả.

	FirstName	LastName	Job Title
1	Ken	Sánchez	Chief Executive Officer
2	Temi	Duffy	Vice President of Engineering
3	Roberto	Tamburello	Engineering Manager
4	Rob	Walters	Senior Tool Designer
5	Gail	Erickson	Design Engineer
6	Jossef	Goldberg	Design Engineer
7	Dylan	Miller	Research and Development Manager

Hình 9.23: Kết quả của phép nối

Có ba loại phép nối như sau:

- ➔ Phép nối trong
- ➔ Phép nối ngoài
- ➔ Phép tự nối

9.7.1 Phép nối trong

Phép nối trong được hình thành khi các bản ghi từ hai bảng được kết hợp chỉ khi các hàng từ cả hai bảng được so khớp dựa trên một cột chung. Sau đây là cú pháp của phép nối bên trong.

Cú pháp:

```
SELECT <ColumnName1>, <ColumnName2>...<ColumnNameN> FROM
Table_A AS Table_Alias_A
INNER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn>=Table_Alias_B.<CommonColumn>
```

Đoạn mã 23 trình bày việc sử dụng phép nối bên trong. Kịch bản cho tính năng này tương tự như Đoạn mã 22.

Đoạn mã 23:

```
SELECT A.FirstName, A.LastName, B.JobTitle
FROM Person.Person A
INNER JOIN HumanResources.Employee B
ON
A.BusinessEntityID=B.BusinessEntityID;
```

Trong Đoạn mã 23, phép nối bên trong được xây dựng giữa Person.Person và HumanResources.Employee dựa trên các id thực thể kinh doanh thông thường. Ở đây một lần nữa, hai bảng được cho các bí danh tương ứng là A và B. Kết quả giống như được trình bày trong hình 9.23.

9.7.2 Phép nối ngoài

Phép nối ngoài là các câu lệnh phép nối trả về tất cả các hàng từ ít nhất một trong các bảng đã chỉ định trong mệnh đề `FROM`, miễn là những hàng này đáp ứng bất kỳ điều kiện `WHERE` hoặc `HAVING` nào của câu lệnh `SELECT`.

Hai loại phép nối bên ngoài thường được sử dụng như sau:

- ➔ Phép nối ngoài bên trái
- ➔ Phép nối ngoài bên phải

Mỗi loại phép nối này sẽ được giải thích bây giờ.

➔ Phép nối ngoài bên trái

Nối ngoài bên trái trả lại tất cả các bản ghi từ bảng bên trái và chỉ so khớp các bản ghi từ bảng bên phải. Sau đây là cú pháp của phép nối bên ngoài.

Cú pháp:

```
SELECT <ColumnList> FROM
Table_A AS Table_Alias_A
LEFT OUTER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn>=Table_Alias_B.<CommonColumn>
```

Xem xét rằng bạn muốn lấy tất cả các id khách hàng từ bảng Sales.Customers và thông tin đơn đặt hàng như là ngày gửi hàng và ngày đến hạn, ngay cả khi khách hàng đã không đặt bất kỳ đơn đặt hàng nào. Do số bản ghi sẽ rất lớn, nó sẽ được giới hạn chỉ những đơn đặt hàng được đặt trước năm 2012. Để đạt được điều này, bạn thực hiện phép nối bên trái như được trình bày trong Đoạn mã 24.

Đoan mă 24:

```
SELECT A.CustomerID, B.DueDate, B.ShipDate  
FROM Sales.Customer A LEFT OUTER JOIN  
Sales.SalesOrderHeader B  
ON  
A.CustomerID = B.CustomerID AND YEAR(B.DueDate) < 2012;
```

Trong Đoạn mã 24, phép nối bên ngoài bên trái được xây dựng giữa các bảng Sales.Customer và Sales.SalesOrderHeader. Những bảng này được nối trên cơ sở các id khách hàng. Trong trường hợp này, tất cả các bảng đều từ bảng bên trái Sales.Customer và chỉ các bảng ghi sổ khớp từ bảng bên phải Sales.SalesOrderHeader, được trả về. Hình 9.24 trình bày kết quả.

	CustomerID	DueDate	ShipDate
3...	18178	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	13671	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	11981	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	18749	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	15251	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	15868	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	18759	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	215	NULL	NULL
3...	46	NULL	NULL
3...	169	NULL	NULL
3...	507	NULL	NULL
3...	630	NULL	NULL

Hình 9.24: Kết quả của phép nối bên ngoài bên trái

Như được trình bày trong kết quả, một số bản ghi trình bày các ngày đến hạn và ngày gửi hàng là NULL. Điều này là do cho một số khách hàng, không có đơn đặt hàng nào được đặt, do đó, các bản ghi của họ sẽ hiển thị những ngày này là NULL.

→ Phép nối ngoài bên phải

Nối ngoài bên phải gọi ra tất cả các bản ghi từ bảng thứ hai trong phần nối bất kể là có dữ liệu khớp trong bảng thứ nhất hay không. Sau đây là cú pháp của phép nối bên ngoài bên phải.

Cú pháp:

```
SELECT <ColumnList>
FROM Left_Table_Name
AS
Table_A AS Table_Alias_A
RIGHT OUTER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn>=Table_Alias_B.<CommonColumn>
```

Xem xét rằng bạn muốn lấy tất cả các tên sản phẩm từ bảng Product và tất cả các id đơn đặt hàng tương ứng từ bảng SalesOrderDetail ngay cả khi không có bản ghi phù hợp cho những sản phẩm trong bảng SalesOrderDetail. Để làm điều này, bạn sẽ sử dụng phép nối bên ngoài bên phải như được trình bày trong Đoạn mã 25.

Đoạn mã 25:

```
SELECT P.Name, S.SalesOrderID
FROM Sales.SalesOrderDetails S
RIGHT OUTER JOIN
Production.Product P
ON P.ProductID=S.ProductID;
```

Trong đoạn mã này, tất cả các bản ghi từ bảng Product được hiển thị bất kể chúng đã được bán hay chưa.

9.7.3 Phép tự nối

Phép tự nối được dùng để tìm ra các bản ghi trong một bảng và có liên quan đến các bản ghi khác trong cùng một bảng. Bảng được nối vào bản thân nó trong một phép tự nối.

Hãy xem xét bảng **Employee** trong cơ sở dữ liệu có tên là **Sterling** có cột tên là **mgr_id** để biểu thị thông tin cho các nhà quản lý nơi nhân viên phải báo cáo cho họ. Giả sử rằng bảng có các bản ghi thích hợp được đưa vào nó.

Người quản lý cũng là một nhân viên. Điều này có nghĩa là **mgr_id** trong bảng là **emp_id** của một nhân viên.

Ví dụ, Anabela với emp_id là ARD36773F là một nhân viên nhưng Anabela cũng là một người quản lý cho Victoria, Palle, Karla và các nhân viên khác như được trình bày trong hình 9.25.

	emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date	mgr_id
1	PMA42628M	Paolo	M	Accorti	13	35	0877	1992-08-27 00:00:00.000	POK93028M
2	PSA89086M	Pedro	S	Afonso	14	89	1389	1990-12-24 00:00:00.000	POK93028M
3	VPA30890F	Victoria	P	Ashworth	6	140	0877	1990-09-13 00:00:00.000	ARD36773F
4	H-B39728F	Helen		Bennett	12	35	0877	1989-09-21 00:00:00.000	POK93028M
5	L-B31947F	Lesley		Brown	7	120	0877	1991-02-13 00:00:00.000	ARD36773F
6	F-C16315M	Francisco		Chang	4	227	9952	1990-11-03 00:00:00.000	MAS70474F
7	PTC11962M	Philip	T	Cramer	2	215	9952	1989-11-11 00:00:00.000	MAS70474F
8	A-C71970F	Aria		Cruz	10	87	1389	1991-10-26 00:00:00.000	POK93028M
9	AMD15433F	Ann	M	Devon	3	200	9952	1991-07-16 00:00:00.000	MAS70474F
10	ARD36773F	Anabela	R	Doming...	8	100	0877	1993-01-27 00:00:00.000	NULL
11	PHF38899M	Peter	H	Franken	10	75	0877	1992-05-17 00:00:00.000	POK93028M
12	PXH22250M	Paul	X	Henriot	5	159	0877	1993-08-19 00:00:00.000	MAS70474F

Hình 9.25: Bảng Employee

Để có được danh sách các tên của người quản lý cùng với các chi tiết khác, bạn có thể sử dụng phép tự nối để nối bảng nhân viên với chính nó và sau đó, trích xuất các bản ghi mong muốn. Đoạn mã 26 trình bày cách sử dụng phép tự nối.

Đoạn mã 26:

```
SELECT TOP 7 A.fname + ' ' + A.lname AS 'Employee Name', B.fname + ' ' + B.lname AS
'Manager'
FROM
Employee AS A
INNER JOIN
Employee AS B
ON A.mgr_id = B.emp_id
```

Trong Đoạn mã 26, bảng Employee được nối vào chính nó dựa trên các cột mgr_id và emp_id.

Kết quả của đoạn mã này được trình bày trong hình 9.26.

	Employee Name	Manager
1	Paolo Accorti	Pirkko Koskitalo
2	Pedro Afonso	Pirkko Koskitalo
3	Victoria Ashworth	Anabela Domingues
4	Helen Bennett	Pirkko Koskitalo
5	Lesley Brown	Anabela Domingues
6	Francisco Chang	Margaret Smith
7	Philip Cramer	Margaret Smith

Hình 9.26: Sử dụng phép tự nối

9.7.4 Câu lệnh MERGE

Câu lệnh **MERGE** cho phép bạn duy trì một bảng đích dựa trên một số điều kiện phép nối trên bảng nguồn sử dụng một câu lệnh duy nhất. Nay giờ bạn có thể thực hiện những hành động sau trong một câu lệnh **MERGE**:

- ➔ Đưa vào một hàng mới từ nguồn nếu hàng bị thiếu trong bảng đích
- ➔ Cập nhật một hàng đích nếu bản ghi đã tồn tại trong bảng nguồn
- ➔ Xóa một hàng đích nếu hàng bị thiếu trong bảng nguồn

Ví dụ, giả sử bạn có bảng **Products** duy trì các bản ghi của tất cả các sản phẩm. Bảng **NewProducts** duy trì các bản ghi các sản phẩm mới. Bạn muốn cập nhật bảng **Products** với các bản ghi từ bảng **NewProducts**. Ở đây, bảng **NewProducts** là bảng nguồn và **Products** là bảng đích. Bảng **Products** chứa các bản ghi của các sản phẩm hiện có với các dữ liệu đã cập nhật và các sản phẩm mới. Hình 9.27 trình bày hai bảng này.

Products				
	ProductID	Name	Type	PurchaseDate
1	101	Rivets	Hardware	2012-12-01
2	102	Nuts	Hardware	2012-12-01
3	103	Washers	Hardware	2011-01-01
4	104	Rings	Hardware	2013-01-15
5	105	Paper Clips	Stationery	2013-01-01

NewProducts				
	ProductID	Name	Type	PurchaseDate
1	102	Nuts	Hardware	2012-12-01
2	103	Washers	Hardware	2011-01-01
3	107	Rings	Hardware	2013-01-15
4	108	Paper Clips	Stationery	2013-01-01

Hình 9.27: Bảng Products và NewProducts

Xem xét việc bạn muốn:

- So sánh tên họ và tên gọi của khách hàng từ cả bảng nguồn và bảng đích
- Cập nhật thông tin khách hàng trong bảng đích nếu tên họ và tên gọi khớp nhau
- Đưa các bản ghi mới vào bảng đích nếu tên họ và tên gọi trong bảng nguồn không tồn tại trong bảng đích
- Xóa các bản ghi hiện hữu trong bảng đích nếu tên họ và tên gọi không khớp với những cái của bảng nguồn

Câu lệnh `MERGE` hoàn thành những tác vụ trong một câu lệnh duy nhất. `MERGE` còn cho phép bạn hiển thị một cách tùy chọn những bản ghi đã được đưa vào, cập nhật hay xóa bằng cách sử dụng mệnh đề `OUTPUT`. Sau đây là cú pháp của câu lệnh `MERGE`.

Cú pháp:

```

MERGE target_table
USING source_table
ON match_condition
WHEN MATCHED THEN UPDATE SET Col1=val1 [, Col2=val2...]
WHEN [TARGET] NOT MATCHED THEN INSERT (Col1 [,Col2...]) VALUES (Val1 [,,
Val2...])
WHEN NOT MATCHED BY SOURCE THEN DELETE
[OUTPUT $action, Inserted.Col1, Deleted.Col1,...];

```

trong đó:

target_table: là bảng WHERE những thay đổi đang được thực hiện.

source_table: là bảng mà từ đó các hàng sẽ được chèn, cập nhật hay xóa vào bảng đích.

match_conditions: là những điều kiện JOIN và bất kỳ toán tử so sánh nào khác.

MATCHED: true nếu một hàng trong target_table và source_table phù hợp với match_condition.

NOT MATCHED: true nếu một hàng từ source_table không tồn tại trong target_table.

SOURCE NOT MATCHED: true nếu một hàng tồn tại trong target_table nhưng không phải trong source_table.

OUTPUT: Mệnh đề tùy chọn cho phép xem những bản ghi đã được chèn/xóa/cập nhật trong target_table.

Các câu lệnh MERGE được kết thúc bằng dấu chấm phẩy (;).

Đoạn mã 27 trình bày cách sử dụng câu lệnh MERGE. Nó sử dụng cơ sở dữ liệu **Sterling**.

Đoạn mã 27:

```

MERGE INTO Products AS P1
USING
NewProducts AS P2
ON P1.ProductId = P2.ProductId
WHEN MATCHED THEN
    UPDATE SET
        P1.Name = P2.Name,
        P1.Type = P2.Type,
        P1.PurchaseDate = P2.PurchaseDate
WHEN NOT MATCHED THEN
    INSERT (ProductId, Name, Type, PurchaseDate)
    VALUES (P2.ProductId, P2.Name, P2.Type, P2.PurchaseDate)
WHEN NOT MATCHED BY SOURCE THEN
    DELETE

OUTPUT $action, Inserted.ProductId, Inserted.Name, Inserted.Type, Inserted.
PurchaseDate, Deleted.ProductId, Deleted.Name, Deleted.Type, Deleted.
PurchaseDate;

```

Hình 9.28 trình bày kết quả.

	\$action	ProductId	Name	Type	PurchaseDate	ProductId	Name	Type	PurchaseDate
1	INSERT	107	Rings	Hardware	2013-01-15	NULL	NULL	NULL	NULL
2	INSERT	108	Paper Clips	Stationery	2013-01-01	NULL	NULL	NULL	NULL
3	DELETE	NULL	NULL	NULL	NULL	101	Rivets	Hardware	2012-12-01
4	UPDATE	102	Nuts	Hardware	2012-12-01	102	Nuts	Hardware	2012-12-01
5	UPDATE	103	Washers	Hardware	2011-01-01	103	Washers	Hardware	2011-01-01
6	DELETE	NULL	NULL	NULL	NULL	104	Rings	Hardware	2013-01-15
7	DELETE	NULL	NULL	NULL	NULL	105	Paper Clips	Stationery	2013-01-01

Hình 9.28: Sử dụng MERGE

Bảng **NewProducts** là bảng nguồn và bảng **Products** là bảng đích. Điều kiện so khớp là cột **ProductId** của cả hai bảng. Nếu điều kiện so khớp đánh giá thành false (NOT MATCHED), khi đó các bản ghi mới được chèn vào bảng đích.

Nếu điều kiện so khớp đánh giá thành true (MATCHED), khi đó các bản ghi được cập nhật vào bảng đích từ bảng nguồn.

Nếu các bản ghi hiện diện trong bảng đích không khớp với những cái trong bảng nguồn (NOT MATCHED BY SOURCE), khi đó chúng bị xóa khỏi bảng đích. Câu lệnh cuối cùng sẽ hiển thị một báo cáo bao gồm các hàng đã được chèn vào/cập nhật/xóa như được trình bày trong kết quả.

9.8 Các biểu thức bảng chung (CTE)

Biểu thức bảng chung (CTE) tương tự như tập kết quả tạm thời được định nghĩa trong phạm vi thực hiện của câu lệnh SELECT, INSERT, UPDATE, DELETE, hoặc CREATE VIEW đơn lẻ. CTE là một biểu thức có tên được định nghĩa trong một truy vấn. CTE được định nghĩa ở đoạn đầu truy vấn và có thể được tham chiếu nhiều lần trong truy vấn bên ngoài. CTE bao gồm các tham chiếu tới chính nó được gọi là CTE đệ quy.

Ghi chú - Biểu thức bảng chung (CTE) lần đầu tiên được giới thiệu trong SQL Server 2005.

Lợi thế chính của CTE là cải thiện khả năng đọc và dễ dàng trong việc bảo trì các truy vấn phức tạp.

Sau đây là cú pháp để tạo ra CTE.

Cú pháp:

```
WITH <CTE_name>
AS (<CTE_definition> )
```

Ví dụ, để lấy và hiển thị số lượng khách hàng theo năm cho các đơn hàng hiện diện trong bảng Sales.SalesOrderHeader, đoạn mã này sẽ như được ra trong Đoạn mã 28.

Đoạn mã 28:

```
WITH CTE_OrderYear
AS
(
SELECT YEAR(OrderDate) AS OrderYear, CustomerID
FROM Sales.SalesOrderHeader
)
SELECT OrderYear, COUNT(DISTINCT CustomerID) AS CustomerCount
FROM CTE_OrderYear
GROUP BY OrderYear;
```

Ở đây, **CTE_OrderYear** được chỉ ra làm tên CTE. Các từ khóa **WITH...AS** bắt đầu định nghĩa CTE. Sau đó, CTE được sử dụng trong câu lệnh **SELECT** để lấy và hiển thị các kết quả mong muốn.

Hình 9.29 trình bày kết quả.

	OrderYear	CustomerCount
1	2007	9864
2	2008	11844
3	2005	1216
4	2006	3094

Hình 9.29: Kết quả của CTE

Các hướng dẫn sau đây cần phải được ghi nhớ trong khi định nghĩa các CTE:

- ➔ CTE được giới hạn trong phạm vi đến việc thực hiện truy vấn bên ngoài. Do đó, khi truy vấn bên ngoài kết thúc, tuổi thọ của CTE sẽ kết thúc.
- ➔ Bạn cần phải định nghĩa tên cho CTE và ngoài ra, định nghĩa tên duy nhất cho mỗi cột đã tham chiếu trong mệnh đề **SELECT** của CTE.
- ➔ Có thể sử dụng các bí danh nội tuyến hoặc bên ngoài cho các cột trong CTE.
- ➔ Một CTE đơn lẻ có thể được tham chiếu nhiều lần trong cùng một truy vấn với một định nghĩa.

- Nhiều CTE cũng có thể được định nghĩa trong cùng một mệnh đề WITH. Ví dụ, hãy xem xét Đoạn mã 29. Nó định nghĩa hai CTE sử dụng mệnh đề WITH duy nhất. Đoạn mã này giả định rằng ba bảng có tên là Student, City, và Status được tạo ra.

Đoạn mã 29:

```
WITH CTE_Students
AS
(
    Select StudentCode, S.Name, C.CityName, St.Status
    FROM Student S
    INNER JOIN City C
        ON S.CityCode = C.CityCode
    INNER JOIN Status St
        ON S.StatusId = St.StatusId)
,
StatusRecord -- Đây là CTE thứ hai đang được định nghĩa
AS
(
    SELECT Status, COUNT(Name) AS CountofStudents
    FROM CTE_Students
    GROUP BY Status
)
SELECT * FROM StatusRecord
```

Giả sử một số bản ghi được chèn vào trong tất cả ba bảng, kết quả có thể như được trình bày trong hình 9.30.

	Status	CountofStudents
1	Failed	2
2	Passed	2

Hình 9.30: Sử dụng nhiều CTE với một WITH

9.9 Kết hợp dữ liệu sử dụng các toán tử SET

SQL Server 2012 cung cấp một số từ khóa nhất định, còn được gọi là các toán tử, để kết hợp dữ liệu từ nhiều bảng. Những toán tử này như sau:

- UNION
- INTERSECT
- EXCEPT

9.9.1 Toán tử UNION

Kết quả từ hai câu lệnh truy vấn khác nhau có thể được kết hợp thành một tập kết quả duy nhất sử dụng toán tử UNION. Các câu lệnh truy vấn phải có các loại cột tương thích và số các cột bằng nhau. Tên cột có thể khác nhau trong từng câu lệnh nhưng các loại dữ liệu phải là loại tương thích. Bằng các loại dữ liệu tương thích, nó có nghĩa là sẽ có thể chuyển đổi nội dung của một trong các cột vào trong cột khác. Ví dụ, nếu một trong những câu lệnh truy vấn có kiểu dữ liệu int và câu lệnh truy vấn kia có kiểu dữ liệu tiền, chúng tương thích và phép hợp có thể xảy ra giữa chúng, vì dữ liệu int có thể được chuyển đổi thành dữ liệu tiền.

Sau đây là cú pháp của toán tử UNION.

Cú pháp:

```
Query_Statement1
UNION [ALL]
Query_Statement2
```

trong đó:

Query_Statement1 và Query_Statement2 là các câu lệnh SELECT.

Đoạn mã 30 trình bày toán tử UNION.

Đoạn mã 30:

```
SELECT Product.ProductId FROM Production.Product
UNION
SELECT ProductId FROM Sales.SalesOrderDetail
```

Điều này sẽ liệt kê tất cả các id sản phẩm của cả hai bảng khớp với nhau. Nếu bạn đưa vào mệnh đề ALL, tất cả các hàng được đưa vào trong tập kết quả bao gồm cả các bản ghi trùng lặp.

Đoạn mã 31 trình bày toán tử UNION ALL.

Đoạn mã 31:

```
SELECT Product.ProductId FROM Production.Product
UNION ALL
SELECT ProductId FROM Sales.SalesOrderDetail
```

Theo mặc định, toán tử UNION loại bỏ các bản ghi trùng lặp từ tập kết quả. Tuy nhiên, nếu bạn sử dụng mệnh đề ALL với toán tử UNION, sau đó tất cả các hàng được trả về. Ngoài UNION, những toán tử khác được sử dụng để kết hợp dữ liệu từ nhiều bảng là INTERSECT và EXCEPT.

9.9.2 Toán tử INTERSECT

Xem xét lại hai bảng Product và SalesOrderDetail hiện diện trong AdventureWorks2012.

Giả sử bạn muốn chỉ hiển thị những hàng là chung cho cả hai bảng. Để làm điều này, bạn sẽ cần phải sử dụng toán tử có tên là INTERSECT. Toán tử INTERSECT được sử dụng với hai câu lệnh truy vấn để trả về một tập riêng biệt các hàng chung cho cả hai câu lệnh truy vấn. Sau đây là cú pháp của toán tử INTERSECT.

Cú pháp:

```
Query_statement1
INTERSECT
Query_statement2
```

trong đó:

Query_Statement1 và Query_Statement2 là các câu lệnh SELECT.

Đoạn mã 32 trình bày toán tử INTERSECT.

Đoạn mã 32:

```
SELECT Product.ProductId FROM Production.Product
INTERSECT
SELECT ProductId FROM Sales.SalesOrderDetail
```

Những quy tắc cơ bản cho việc sử dụng INTERSECT như sau:

- ➔ Số các cột và thứ tự theo đó chúng được đưa ra phải giống nhau trong cả hai truy vấn.
- ➔ Các loại dữ liệu của các cột được dùng phải tương thích.

Kết quả của giao điểm của các bảng Production.Product and Sales.SalesOrderDetail sẽ chỉ có những id sản phẩm có các bản ghi so khớp trong bảng Production.Product. Ở các doanh nghiệp lớn, có số lượng lớn dữ liệu được lưu trữ trong các cơ sở dữ liệu. Thay bằng lưu toàn bộ dữ liệu trong một bảng đơn, nó có thể được trải ra trên một số bảng mà có liên quan lẫn nhau. Khi dữ liệu được lưu trong các bảng như vậy, phải có một số phương tiện để kết hợp và gọi ra dữ liệu từ các bảng này. Việc sử dụng SQL Server, có một số cách để kết hợp dữ liệu từ nhiều bảng. Các phần sau đây khám phá các cách này một cách chi tiết.

9.9.3 Toán tử EXCEPT

Toán tử EXCEPT trả về tất cả các hàng riêng biệt từ truy vấn đã cho ở bên trái của toán tử EXCEPT và loại bỏ tất cả các hàng khỏi tập kết quả so khớp các hàng ở bên phải của toán tử EXCEPT.

Sau đây là cú pháp của toán tử EXCEPT.

Cú pháp:

```
Query_statement1
EXCEPT
Query_statement2
```

trong đó:

Query_Statement1 và Query_Statement2 là các câu lệnh SELECT.

Hai quy tắc áp dụng cho toán tử INTERSECT còn được áp dụng cho toán tử EXCEPT.

Đoạn mã 33 trình bày toán tử EXCEPT.

Đoạn mã 33:

```
SELECT Product.ProductId FROM Production.Product
EXCEPT
SELECT ProductId FROM Sales.SalesOrderDetail
```

Nếu thứ tự của hai bảng trong ví dụ này được hoán đổi, chỉ có những hàng được trả về từ bảng Production.Product mà không khớp với các hàng hiện diện trong Sales.SalesOrderDetail.

Như vậy, trong các điều kiện đơn giản, toán tử EXCEPT chọn tất cả các bản ghi từ bảng đầu tiên ngoại trừ những bản ghi khớp với bảng thứ hai. Do đó, khi bạn đang sử dụng toán tử EXCEPT, thứ tự của hai bảng trong những truy vấn này rất quan trọng. Trong khi đó, với toán tử INTERSECT, không quan trọng bảng nào được chỉ ra đầu tiên.

9.10 Xoay vòng và nhóm các phép tính tập hợp

Xem xét kịch bản khi dữ liệu cần phải được hiển thị theo một hướng khác so với nó được lưu trữ, theo quan điểm bố trí hàng và cột. Quá trình chuyển đổi dữ liệu từ một hướng dựa trên hàng thành một hướng dựa trên cột được gọi là xoay vòng. Các toán tử PIVOT và UNPIVOT của SQL Server giúp thay đổi hướng của dữ liệu từ hướng cột sang hướng hàng và ngược lại. Điều này được thực hiện bằng cách hợp nhất các giá trị hiện diện trong một cột vào danh sách các giá trị riêng biệt và sau đó trình chiếu danh sách đó ở dạng đầu đề cột.

9.10.1 Toán tử PIVOT

Sau đây là cú pháp ngắn gọn cho PIVOT.

Cú pháp:

```

SELECT <cột không được xoay vòng>,
    [cột được xoay vòng đầu tiên] AS <tên cột>,
    [cột được xoay vòng thứ hai] AS <tên cột>,
    ...
    [cột được xoay vòng cuối cùng] AS <tên cột>

FROM
    (<SELECT truy vấn tạo ra dữ liệu>)
    AS <bí danh cho truy vấn nguồn>

PIVOT
(
    <hàm tổng hợp> (<cột đang được tổng hợp>)

FOR
    [<cột có chứa những giá trị sẽ trở thành các đầu đề cột>]
    IN ( [cột được xoay vòng đầu tiên], [cột được xoay vòng thứ hai],
        ... [cột được xoay vòng cuối cùng] )
) AS <bí danh cho bảng xoay vòng>
<mệnh đề ORDER BY tùy chọn>;

```

trong đó:

`table_source`: là một bảng hoặc biểu thức bảng.

`aggregate_function`: là một hàm tổng hợp do người dùng định nghĩa hoặc dựng sẵn nhận một hoặc nhiều giá trị đầu vào.

`value_column`: là cột giá trị của toán tử PIVOT.

`pivot_column`: là cột xoay vòng của toán tử PIVOT. Cột này phải thuộc một loại có thể ngụ ý hay rõ ràng được chuyển đổi sang `nvarchar()`.

`IN (column_list)`: là các giá trị trong `pivot_column` sẽ trở thành tên cột của bảng đầu ra. Danh sách không phải bao gồm bất kỳ tên cột nào đã tồn tại trong `table_source` đầu vào đang được xoay vòng.

`table_alias`: là tên bí danh của bảng đầu ra.

Đầu ra của phần này sẽ là một bảng chứa tất cả các cột của `table_source` ngoại trừ `pivot_column` và `value_column`. Những cột này của `table_source`, ngoại trừ `pivot_column` và `value_column`, được gọi là các cột gộp nhóm của toán tử xoay vòng.

Nói một cách đơn giản, để sử dụng toán tử PIVOT, bạn cần phải cung cấp ba phần tử cho toán tử này:

- **Gộp nhóm:** Trong mệnh đề `FROM`, những cột đầu vào phải được cung cấp. Toán tử PIVOT sử dụng những cột đó để xác định cột nào để sử dụng cho gộp nhóm dữ liệu cho tập hợp.
- **Lan rộng:** Ở đây, danh sách tách biệt bằng dấu phẩy của các giá trị xảy ra trong dữ liệu nguồn được cung cấp sẽ được sử dụng làm các đầu đề cột cho dữ liệu xoay vòng.
- **Tổng hợp:** Hàm tổng hợp, như là `SUM`, để được thực hiện trên những hàng đã phân nhóm.

Xem xét một ví dụ để hiểu toán tử PIVOT. Đoạn mã 34 được trình bày không có toán tử PIVOT và chứng minh sự tổng hợp GROUP BY đơn giản. Bởi số lượng bản ghi sẽ là rất lớn, tập kết quả được giới hạn tới 5 bằng cách chỉ ra `TOP 5`.

Đoạn mã 34:

```
SELECT TOP 5 SUM(SalesYTD) AS TotalSalesYTD, Name
FROM Sales.SalesTerritory
GROUP BY Name
```

Hình 9.31 trình bày kết quả.

	TotalSalesYTD	Name
1	7887186.7882	Northwest
2	2402176.8476	Northeast
3	3072175.118	Central
4	10510853.8739	Southwest
5	2538667.2515	Southeast

Hình 9.31: Gộp nhóm không có PIVOT

Top 5 doanh số năm cho đến nay cùng với tên lãnh thổ nhóm theo tên lãnh thổ được hiển thị.

Bây giờ, cùng một truy vấn được viết lại trong Đoạn mã 35 sử dụng PIVOT để dữ liệu được chuyển từ một hướng dựa trên hàng cho hướng dựa trên cột.

Đoạn mã 35:

```
-- Bảng xoay vòng với một hàng và sáu cột
SELECT TOP 5 'TotalSalesYTD' AS GrandTotal,
[Northwest], [Northeast], [Central], [Southwest], [Southeast]
```

```

FROM
(SELECT TOP 5 Name, SalesYTD
FROM Sales.SalesTerritory
) AS SourceTable
PIVOT
(
SUM(SalesYTD)
FOR Name IN ([Northwest], [Northeast], [Central], [Southwest], [Southeast])
) AS PivotTable;

```

Hình 9.32 trình bày kết quả.

	GrandTotal	Northwest	Northeast	Central	Southwest	Southeast
1	TotalSalesYTD	7887186.7882	2402176.8476	3072175.1118	10510853.8739	2538667.2515

Hình 9.32: Gộp nhóm có PIVOT

Như được trình bày trong hình 9.32, dữ liệu được chuyển đổi và tên lảnh thổ bây giờ được xem là các cột thay vì các hàng. Điều này cải thiện khả năng đọc. Một thách thức lớn trong việc viết các truy vấn sử dụng PIVOT là sự cần thiết để cung cấp một danh sách cố định của các phần tử lan rộng cho toán tử PIVOT, chẳng hạn như tên lảnh thổ cụ thể được đưa ra trong Đoạn mã 35. Nó sẽ không có tính khả thi hoặc thực tế để thực hiện điều này cho số lượng lớn các phần tử lây lan. Để khắc phục điều này, các nhà phát triển có thể sử dụng SQL động. SQL động cung cấp phương tiện để xây dựng một chuỗi ký tự được truyền cho SQL Server, được phiên dịch là một lệnh, và sau đó, thực hiện.

9.10.2 Toán tử UNPIVOT

UNPIVOT thực hiện hầu hết hoạt động ngược lại của PIVOT, bằng cách xoay cột vào hàng. Bỏ xoay vòng không khôi phục lại dữ liệu gốc. Dữ liệu mức chi tiết đã bị mất trong quá trình xử lý tổng hợp trong xoay vòng gốc. UNPIVOT không có khả năng phân bổ các giá trị để trả về các giá trị chi tiết ban đầu. Thay vì chuyển các hàng thành cột, bỏ xoay vòng dẫn đến các cột được chuyển đổi thành các hàng. SQL Server cung cấp toán tử bảng UNPIVOT để trả về hiển thị dạng bảng hướng hàng từ dữ liệu đã xoay vòng.

Khi bỏ xoay vòng dữ liệu, một hoặc nhiều cột được định nghĩa là nguồn để được chuyển đổi thành hàng. Dữ liệu trong những cột đó được kéo dài ra, hoặc phân chia, thành một hoặc nhiều hàng mới, tùy thuộc vào bao nhiêu cột đang được bỏ xoay vòng.

Để sử dụng toán tử UNPIVOT, bạn cần phải cung cấp ba yếu tố như sau:

- ➔ Các cột nguồn để được bỏ xoay vòng
- ➔ Tên cho cột mới sẽ hiển thị những giá trị đã bỏ xoay vòng
- ➔ Tên cho cột sẽ hiển thị tên của các những trị đã bỏ xoay vòng

Xem xét kịch bản trước đây. Đoạn mã 36 trình bày đoạn mã để chuyển đổi bảng xoay vòng tạm thời thành bảng lâu dài để cùng một bảng có thể được sử dụng để trình bày các phép tính UNPIVOT.

Đoạn mã 36:

```
-- Bảng xoay vòng với một hàng và sáu cột
SELECT TOP 5 'TotalSalesYTD' AS GrandTotal,
[Northwest], [Northeast], [Central], [Southwest], [Southeast]
FROM
(SELECT TOP 5 Name, SalesYTD
FROM Sales.SalesTerritory
) AS SourceTable
PIVOT
(
SUM(SalesYTD)
FOR Name IN ([Northwest], [Northeast], Central], [Southwest], [Southeast])
) AS PivotTable;
```

Đoạn mã 37 trình bày toán tử UNPIVOT.

Đoạn mã 37:

```
SELECT Name, SalesYTD FROM
(SELECT GrandTotal, Northwest, Northeast, Central, Southwest, Southeast FROM
TotalTable) P
UNPIVOT
(SalesYTD FOR Name IN
(Northwest, Northeast, Central, Southwest, Southeast))
) AS unpvt;
```

Kết quả sẽ giống như trong hình 9.32.

9.10.3 GROUPING SETS

Toán tử GROUPING SETS hỗ trợ tổng hợp của nhiều gộp nhóm cột và tổng cộng tùy chọn. Xem xét trường hợp bạn cần một báo cáo có nhóm nhiều cột của một bảng. Hơn nữa, bạn muốn tập hợp các cột lại. Trong các phiên bản trước đây của SQL Server, bạn phải viết một số mệnh đề GROUP BY riêng biệt tiếp theo là các mệnh đề UNION để đạt được điều này. Đầu tiên được giới thiệu trong SQL Server 2008, toán tử GROUPING SETS cho phép bạn nhóm lại với nhau nhiều gộp nhóm các cột tiếp theo là hàng tổng số tùy chọn, ký hiệu bằng dấu ngoặc đơn (). Sẽ hiệu quả hơn khi sử dụng các toán tử GROUPING SETS thay vì nhiều GROUP BY với các mệnh đề UNION bởi vì cái sau thêm các chi phí xử lý lên máy chủ cơ sở dữ liệu.

Sau đây là cú pháp của toán tử GROUPING SETS.

Cú pháp:

```
GROUP BY
GROUPING SETS ( <danh sách tập hợp gộp nhóm> )
```

trong đó:

danh sách tập hợp gộp nhóm: bao gồm một hoặc nhiều cột, cách nhau bằng dấu phẩy.

Cặp dấu ngoặc đơn (), không có bất kỳ tên cột nào biểu thị tổng cộng.

Đoạn mã 38 trình bày toán tử GROUPING SETS. Giả định rằng một bảng **Students** được tạo ra với các trường có tên tương ứng là **Id**, **Name**, và **Marks**.

Đoạn mã 38:

```
SELECT Id, Name, AVG(Marks) Marks
FROM Students
GROUP BY
GROUPING SETS
(
    (Id, Name, Marks),
    (Id),
    ()
)
```

Hình 9.33 trình bày kết quả của đoạn mã.

	Id	Name	Marks
1	91	Sasha Goldsmith	78
2	91	NULL	78
3	92	Karen Hues	55
4	92	NULL	55
5	93	William Pinter	67
6	93	NULL	67
7	94	Yuri Gogol	89
8	94	NULL	89
9	NULL	NULL	72

Hình 9.33: GROUPING SETS

Ở đây, đoạn mã này sử dụng GROUPING SETS để hiển thị điểm trung bình cho mỗi học sinh. Các giá trị NULL trong **Name** chỉ ra điểm trung bình cho mỗi học sinh. Giá trị NULL trong cả cột **Id** và **Name** chỉ ra tổng cộng.

9.11 Kiểm tra tiến bộ của bạn

1. Câu lệnh nào sau đây có thể được sử dụng với các truy vấn con để trả về một cột và nhiều hàng?

(A)	ANY	(C)	IN
(B)	ALL	(D)	=

2. Toán tử _____ được sử dụng để chỉ hiển thị những hàng là chung cho cả hai bảng.

(A)	INTERSECT	(C)	UNION
(B)	EXCEPT	(D)	UNION WITH ALL

3. Truy vấn con bao gồm truy vấn con khác bên trong nó được gọi là _____.

(A)	join	(C)	truy vấn con tương quan
(B)	truy vấn con xếp lồng	(D)	truy vấn con cha mẹ

4. _____ được hình thành khi các bản ghi từ hai bảng được kết hợp chỉ khi các hàng của cả hai bảng được so khớp dựa trên một cột chung.

(A)	Phép nối bên trong	(C)	Phép tự nối
(B)	Phép nối bên ngoài bên trái	(D)	Phép nối bên ngoài bên phải

5. _____ trả về tất cả các hàng từ ít nhất một trong các bảng trong mệnh đề FROM của câu lệnh SELECT, miễn là những hàng đó đáp ứng bất kỳ điều kiện WHERE hoặc HAVING nào của câu lệnh SELECT.

(A)	Phép nối bên trong	(C)	Phép tự nối
(B)	Phép nối bên ngoài	(D)	Các truy vấn con

6. Xem xét bạn có hai bảng là **Products** và **Orders** đã được tập kết. Dựa trên các đơn đặt hàng mới, bạn muốn cập nhật **Quantity** trong bảng **Products**. Bạn viết đoạn mã sau đây:

```
MERGE Products AS T
USING Orders AS S
ON S.ProductID = T.ProductID
```

Đoạn mã nào sau đây, khi đưa vào khoảng trống, sẽ cho phép bạn đạt được điều này?

(A)	WHEN MATCHED THEN UPDATE SET T.Quantity = S.Quantity	(C)	WHEN NOT MATCHED THEN UPDATE SET T.Quantity = S.Quantity
(B)	WHEN MATCHED THEN UPDATE SET Quantity = Quantity	(D)	WHEN MATCHED THEN UPDATE SET S.Quantity = T.Quantity

7. Điều nào sau đây sẽ là kết quả của đoạn mã đã cho?

```
SELECT ProdId, Year,
SUM(Purchase) AS TotalPurchase
FROM Products
GROUP BY GROUPING SETS ((ProdId), ())
```

(A)	Lỗi cú pháp	(C)	Sẽ hiển thị tổng dữ liệu đặt mua theo năm ngoại trừ số tổng cộng
(B)	Sẽ thực thi như sẽ không tạo ra kết quả nào cả	(D)	Sẽ hiển thị tổng dữ liệu đặt mua theo năm với số tổng cộng

9.11.1 Câu trả lời

1.	C
2.	B
3.	B
4.	B
5.	B
6.	A
7.	A

Tóm tắt

- Mệnh đề GROUP BY và các hàm tổng hợp đã cho phép để gộp nhóm và/hoặc tổng hợp dữ liệu với nhau để trình bày thông tin tóm tắt.
- Các hàm tổng hợp không gian mới được giới thiệu trong SQL Server 2012.
- Truy vấn con cho phép tập kết quả của một câu lệnh SELECT được sử dụng làm tiêu chí cho câu lệnh SELECT khác.
- Phép nối giúp bạn kết hợp dữ liệu cột từ hai cột hoặc nhiều bảng dựa trên mỗi quan hệ logic giữa những bảng này.
- Các toán tử tập hợp như UNION và INTERSECT giúp bạn kết hợp dữ liệu hàng từ hai hoặc nhiều bảng.
- Các toán tử PIVOT và UNPIVOT giúp thay đổi hướng của dữ liệu từ hướng cột sang hướng hàng và ngược lại.
- Mệnh đề con GROUPING SET của mệnh đề GROUP BY giúp chỉ ra nhiều nhóm trong một truy vấn đơn lẻ.

Hãy thử tự làm

1. Viết một truy vấn để hiển thị tên nhân viên và các phòng ban của họ từ cơ sở dữ liệu AdventureWorks2012.
2. Sử dụng các bảng Sales.SalesPerson and Sales.SalesTerritory, lấy các ID của tất cả những người bán hàng hoạt động ở Canada.
3. Sử dụng các bảng Sales.SalesPerson và Sales.SalesTerritory, lấy các ID của tất cả những người bán hàng hoạt động ở Northwest hoặc Northeast.
4. So sánh các giá trị tiền thưởng của nhân viên bán hàng trong bảng Sales.SalesPerson để tìm ra những người bán hàng kiếm tiền thưởng nhiều hơn. Hiển thị SalesPersonID và các giá trị tiền thưởng theo thứ tự giảm dần. (Gợi ý: Sử dụng phép tự nối và ORDER BY ...DESC).
5. Lấy tất cả các giá trị của SalesPersonID từ bảng Sales.SalesPerson, nhưng bỏ qua những giá trị đó, có hiện diện trong bảng Sales.Store. (Gợi ý: Sử dụng toán tử EXCEPT).
6. Kết hợp tất cả các SalesPersonIDs của các bảng Sales.SalesPerson và Sales.Store.
7. Lấy tất cả các ID người bán hàng và ID lãnh thổ từ bảng Sales.SalesPerson bất kể là họ có các bản ghi khớp với các bản ghi trong bảng Sales.SalesTerritory hay không. (Gợi ý: Sử dụng phép nối ngoài bên trái).
8. Lấy một tập riêng biệt các ID lãnh thổ hiện diện trong cả hai bảng Sales.SalesPerson và Sales.SalesTerritory. (Gợi ý: Sử dụng toán tử INTERSECT).

Phần - 10

Sử dụng khung nhìn, thủ tục đã lưu trữ và truy vấn siêu dữ liệu

Chào mừng bạn đến phần **Sử dụng khung nhìn, thủ tục đã lưu trữ và truy vấn siêu dữ liệu**.

Phần này giải thích về khung nhìn và mô tả việc tạo ra, thay đổi, và thả bỏ các khung nhìn. Phần này còn mô tả các thủ tục lưu trữ một cách chi tiết. Phần này kết thúc với phần giải thích về các kỹ thuật để truy vấn siêu dữ liệu.

Trong phần này, bạn sẽ học để:

- ➔ Định nghĩa các khung nhìn
- ➔ Mô tả kỹ thuật này để tạo ra, thay đổi, và thả bỏ các khung nhìn
- ➔ Định nghĩa các thủ tục lưu trữ
- ➔ Giải thích các loại thủ tục lưu trữ
- ➔ Mô tả thủ tục để tạo ra, thay đổi, và thực thi các thủ tục lưu trữ
- ➔ Mô tả các thủ tục lưu trữ lồng nhau
- ➔ Mô tả truy vấn siêu dữ liệu SQL Server
 - Các khung nhìn và hàm Danh mục Hệ thống
 - Truy vấn các đối tượng quản lý động

10.1 Giới thiệu

Cơ sở dữ liệu SQL Server có hai thể loại đối tượng chính: những đối tượng lưu trữ dữ liệu và những đối tượng truy cập, thao tác, hoặc cung cấp quyền truy cập vào dữ liệu. Các khung nhìn và thủ tục lưu trữ thuộc về thể loại sau này.

10.2 Các khung nhìn

Khung nhìn là một bảng ảo được tạo thành từ các cột đã chọn từ một hoặc nhiều bảng. Những bảng từ đó khung nhìn được tạo ra được gọi là bảng cơ sở. Những bảng cơ sở này có thể là từ các cơ sở dữ liệu khác nhau. Khung nhìn cũng có thể bao gồm các cột từ các khung nhìn khác được tạo ra trong cùng một cơ sở dữ liệu hoặc một cơ sở dữ liệu khác. Khung nhìn có thể có tối đa là 1024 cột.

Các dữ liệu bên trong khung nhìn từ các bảng cơ sở được tham chiếu trong định nghĩa khung nhìn. Những hàng và cột của khung nhìn được tạo ra dạng động khi khung nhìn được tham chiếu.

10.2.1 Tạo ra các khung nhìn

Một người dùng có thể tạo ra một khung nhìn sử dụng các cột từ các bảng hoặc khung nhìn khác chỉ khi người dùng có quyền truy cập những bảng và khung nhìn này. Khung nhìn được tạo ra sử dụng câu lệnh CREATE VIEW và nó có thể được tạo ra chỉ trong cơ sở dữ liệu hiện tại.

SQL Server xác minh sự tồn tại của các đối tượng được tham chiếu trong định nghĩa khung nhìn.

Ghi chú - Trong khi tạo ra một khung nhìn, thử nghiệm câu lệnh SELECT định nghĩa khung nhìn nhằm đảm bảo rằng SQL Server trả về kết quả mong đợi. Bạn tạo ra khung nhìn chỉ sau khi câu lệnh SELECT được thử nghiệm và tập kết quả đã được xác minh.

Cú pháp sau đây được sử dụng để tạo ra một khung nhìn.

Cú pháp:

```
CREATE VIEW <view_name>
AS <select_statement>
```

trong đó:

index_name: chỉ ra tên của khung nhìn.

select_statement: chỉ ra câu lệnh SELECT định nghĩa khung nhìn.

Đoạn mã 1 tạo ra một khung nhìn từ bảng Product để chỉ hiển thị id sản phẩm, số sản phẩm, tên, và mức độ lưu kho an toàn của các sản phẩm.

Đoạn mã 1:

```
CREATE VIEW vwProductInfo AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel
FROM Production.Product;
GO
```

Ghi chú - Những từ vw được đặt đầu tiên trước tên khung nhìn theo các công ước mã hóa đề xuất.

Mã sau đây trong Đoạn mã 2 được sử dụng để hiển thị các chi tiết của khung nhìn **vwProductInfo**.

Đoạn mã 2:

```
SELECT * FROM vwProductInfo
```

Kết quả sẽ trình bày những cột cụ thể của tất cả các sản phẩm từ bảng Product. Một phần của kết quả được trình bày trong hình 10.1.

	ProductID	ProductNumber	Name	SafetyStockLevel
1	1	AR-5381	Adjustable Race	1000
2	2	BA-8327	Bearing Ball	1000
3	3	BE-2349	BB Ball Bearing	800
4	4	BE-2908	Headset Ball Bearings	800
5	316	BL-2036	Blade	800
6	317	CA-5965	LL Crankarm	500
7	318	CA-6738	ML Crankarm	500
8	319	CA-7457	HL Crankarm	500
9	320	CB-2903	Chainring Bolts	1000
10	321	CN-6137	Chainring Nut	1000

Hình 10.1: Lựa chọn các bản ghi từ một khung nhìn

10.2.2 Tạo ra các khung nhìn sử dụng từ khóa JOIN

Từ khóa JOIN cũng có thể được sử dụng để tạo ra các khung nhìn. Câu lệnh CREATE VIEW được sử dụng cùng với từ khóa JOIN để tạo ra một khung nhìn sử dụng các cột từ nhiều bảng.

Cú pháp sau đây được sử dụng để tạo ra một khung nhìn với từ khóa JOIN.

Cú pháp:

```
CREATE VIEW <view_name>
AS
SELECT * FROM table_name1
JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

trong đó:

index_name: chỉ ra tên của khung nhìn.

table_name1: chỉ ra tên của bảng đầu tiên.

JOIN: chỉ ra rằng hai bảng được nối sử dụng từ khóa JOIN.

table_name: chỉ ra tên của bảng thứ hai.

Đoạn mã 3 tạo ra một khung nhìn có tên là **vwPersonDetails** với các cột cụ thể từ bảng Person và Employee. Từ khóa JOIN và ON nối hai bảng dựa trên cột BusinessEntityID.

Đoạn mã 3:

```
CREATE VIEW vwPersonDetails
AS
SELECT
    p.Title
    ,p.[FirstName]
    ,p.[MiddleName]
    ,p.[LastName]
    ,e.[JobTitle]
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Person] p
        ON p.BusinessEntityID = e.BusinessEntityID
GO
```

Khung nhìn này sẽ chứa các cột Title, FirstName, MiddleName, và LastName từ bảng Person và JobTitle từ bảng Employee. Sau khi khung nhìn được tạo ra, bạn có thể lấy các bản ghi từ đó, và còn thao tác và chỉnh sửa các bản ghi.

Đoạn mã 4 trình bày thực hiện một truy vấn trên khung nhìn này.

Đoạn mã 4:

```
SELECT * FROM vwPersonDetails
```

Kết quả sẽ như được trình bày trong hình 10.2.

	Title	FirstName	MiddleName	LastName	Job Title
1	NULL	Ken	J	Sánchez	Chief Executive Officer
2	NULL	Temi	Lee	Duffy	Vice President of Engineering
3	NULL	Roberto	NULL	Tamburello	Engineering Manager
4	NULL	Rob	NULL	Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7	NULL	Dylan	A	Miller	Research and Development Manager
8	NULL	Diane	L	Margheim	Research and Development Engineer
9	NULL	Gigi	N	Matthew	Research and Development Engineer
10	NULL	Michael	NULL	Raheem	Research and Development Manager

Hình 10.2: Tạo ra các khung nhìn với phép nối

Như được trình bày trong hình 10.2, tất cả các hàng có thể không có các giá trị cho các cột Title hoặc MiddleName - một số có thể có NULL trong đó. Một người nhìn thấy kết quả này có thể không có khả năng hiểu được ý nghĩa của các giá trị NULL. Do đó, để thay thế tất cả các giá trị NULL ở đầu ra với một chuỗi null, hàm COALESCE () có thể được sử dụng như được trình bày trong Đoạn mã 5.

Đoạn mã 5:

```
CREATE VIEW vwPersonDetails
AS
SELECT
    COALESCE(p.Title, '') AS Title
    ,p.[FirstName]
    ,COALESCE(p.MiddleName, '') AS MiddleName
    ,p.[LastName]
    ,e.[JobTitle]
FROM [HumanResources].[Employee] e
    INNER JOIN [Person].[Person] p
        ON p.BusinessEntityID = e.BusinessEntityID
GO
```

Sử dụng khung nhìn, thủ tục đã lưu trữ và truy vấn siêu dữ liệu

Khi khung nhìn này được truy vấn với câu lệnh SELECT, kết quả sẽ được trình bày trong hình 10.3.

	Title	FirstName	MiddleName	LastName	Job Title
1		Ken	J	Sánchez	Chief Executive Officer
2		Temi	Lee	Duffy	Vice President of Engineering
3		Roberto		Tamburello	Engineering Manager
4		Rob		Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7		Dylan	A	Miller	Research and Development Manager
8		Diane	L	Margheim	Research and Development Engineer
9		Gigi	N	Matthew	Research and Development Engineer
10		Michael		Raheem	Research and Development Manager

Hình 10.3: Sử dụng hàm COALESCE trong một khung nhìn

10.2.3 Hướng dẫn và hạn chế về khung nhìn

Khung nhìn có thể được tạo ra sử dụng lệnh CREATE VIEW. Trước khi tạo ra một khung nhìn, hướng dẫn và hạn chế sau đây cần được xem xét:

- Khung nhìn được tạo ra chỉ trong cơ sở dữ liệu hiện tại. Các bảng cơ sở và khung nhìn từ đó khung nhìn được tạo ra có thể là từ các cơ sở dữ liệu hoặc máy chủ khác.
- Các tên khung nhìn phải là duy nhất và không thể được giống như các tên bảng trong lược đồ.
- Khung nhìn không thể được tạo ra trên các bảng tạm thời.
- Khung nhìn không thể có một chỉ mục toàn văn.
- Khung nhìn không thể chứa định nghĩa DEFAULT.
- Câu lệnh CREATE VIEW có thể bao gồm mệnh đề ORDER BY chỉ khi từ khóa TOP được sử dụng.
- Khung nhìn không thể tham chiếu hơn 1024 cột.
- Câu lệnh CREATE VIEW không thể bao gồm từ khóa INTO.
- Câu lệnh CREATE VIEW không thể được kết hợp với các câu lệnh Transact-SQL khác trong một khối lệnh đơn lẻ.

Nếu khung nhìn chứa các cột có giá trị bắt nguồn từ một biểu thức, các cột như vậy phải được đặt tên bí danh. Ngoài ra, nếu khung nhìn chứa các cột có tên tương tự từ các bảng khác nhau, để phân biệt những cột này, tên bí danh phải được chỉ ra.

Đoạn mã 6 tái sử dụng mã được cho trong Đoạn mã 5 với mệnh đề ORDER BY. Từ khóa TOP hiển thị tên của mười nhân viên đầu tiên với các tên họ của họ theo thứ tự tăng dần.

Đoạn mã 6:

```
CREATE VIEW vwSortedPersonDetails
AS
SELECT TOP 10
    COALESCE(p.Title, '') AS Title
    ,p.[FirstName]
    ,COALESCE(p.MiddleName, '') AS MiddleName
    ,p.[LastName]
    ,e.[JobTitle]

    FROM [HumanResources].[Employee] e
        INNER JOIN [Person].[Person] p
            ON p.BusinessEntityID = e.BusinessEntityID
        ORDER BY p.FirstName

GO

--Lấy các bản ghi từ khung nhìn
SELECT * FROM vwSortedPersonDetails
```

10.3 Sửa đổi dữ liệu thông qua các khung nhìn

Khung nhìn có thể được sử dụng để sửa đổi dữ liệu trong các bảng cơ sở dữ liệu. Dữ liệu có thể được chèn, sửa đổi hoặc xóa thông qua một khung nhìn sử dụng các câu lệnh sau:

- ➔ INSERT
- ➔ UPDATE
- ➔ DELETE

10.3.1 INSERT với các khung nhìn

Câu lệnh INSERT được sử dụng để thêm một hàng mới vào một bảng hoặc khung nhìn. Trong khi thực hiện câu lệnh, nếu giá trị cho một cột không được cung cấp, SQL Server Database Engine phải cung cấp một giá trị dựa trên định nghĩa của cột. Nếu Database Engine không thể cung cấp giá trị này, khi đó hàng mới sẽ không được thêm vào.

Giá trị cho cột này được cung cấp tự động nếu:

- Cột có thuộc tính IDENTITY.
- Cột có giá trị mặc định đã chỉ ra.
- Cột có kiểu dữ liệu timestamp.
- Cột có các giá trị null.
- Cột là một cột tính.

Trong khi sử dụng câu lệnh INSERT trên khung nhìn, nếu có bất kỳ quy tắc nào bị vi phạm, bản ghi sẽ không được chèn vào.

Trong ví dụ sau đây, khi dữ liệu được đưa vào thông qua khung nhìn, việc chèn không diễn ra bởi khung nhìn được tạo ra từ hai bảng cơ sở.

Đầu tiên, tạo một bảng **Employee_Personal_Details** như được trình bày trong Đoạn mã 7.

Đoạn mã 7:

```
CREATE TABLE Employee_Personal_Details
(
    EmpID int NOT NULL,
    FirstName varchar(30) NOT NULL,
    LastName varchar(30) NOT NULL,
    Address varchar(30)
)
```

Sau đó, tạo một bảng **Employee_Personal_Details** như được trình bày trong Đoạn mã 8.

Đoạn mã 8:

```
CREATE TABLE Employee_Salary_Details
(
    EmpID
    int NOT NULL,
    Designation varchar(30),
    Salary int NOT NULL
)
```

Đoạn mã 9 tạo ra một khung nhìn `vwEmployee_Details` sử dụng các cột từ bảng `Employee_Personal_Details` và `Employee_Salary_Details` bằng cách nối hai bảng trên cột `EmpID`.

Đoạn mã 9:

```
CREATE VIEW vwEmployee_Details
AS
SELECT e1.EmpID, FirstName, LastName, Designation, Salary
FROM Employee_Personal_Details e1
JOIN Employee_Salary_Details e2
ON e1.EmpID = e2.EmpID
```

Đoạn mã 10 sử dụng câu lệnh `INSERT` để chèn dữ liệu thông qua khung nhìn `vwEmployee_Details`. Tuy nhiên, dữ liệu không được chèn vào bởi khung nhìn được tạo ra từ hai bảng cơ sở.

Đoạn mã 10:

```
INSERT INTO vwEmployee_Details VALUES (2, 'Jack', 'Wilson', 'Software
Developer', 16000)
```

Thông báo lỗi sau được hiển thị khi câu lệnh `INSERT` được thực thi.

'Msg 4405, Level 16, State 1, Line 1

Khung nhìn hoặc hàm 'vwEmployee_Details' là không thể cập nhật vì sửa đổi ảnh hưởng đến nhiều bảng cơ sở.'

Giá trị có thể được chèn vào các cột thuộc kiểu dữ liệu do người dùng định nghĩa bằng cách:

- Chỉ ra giá trị của loại do người dùng định nghĩa.
- Việc gọi một hàm do người dùng định nghĩa trả về giá trị thuộc loại do người dùng định nghĩa.

Các quy tắc và hướng dẫn sau đây phải được tuân thủ khi sử dụng câu lệnh `INSERT`:

- Câu lệnh `INSERT` phải chỉ ra các giá trị cho tất cả các cột trong một khung nhìn trong bảng bên dưới không cho phép các giá trị null và không có định nghĩa `DEFAULT`.
- Khi có một phép tự nối với cùng một khung nhìn hoặc bảng cơ sở, câu lệnh `INSERT` không làm việc.

Đoạn mã 11 tạo ra một khung nhìn **vwEmpDetails** sử dụng bảng **Employee_Personal_Details**.
Bảng **Employee_Personal_Details** chứa một cột có tên là **LastName** không cho phép các giá trị null
được chèn vào.

Đoạn mã 11:

```
CREATE VIEW vwEmpDetails
AS
SELECT FirstName, Address
FROM Employee_Personal_Details
GO
```

Đoạn mã 12 cố gắng chèn các giá trị vào khung nhìn **vwEmpDetails**.

Đoạn mã 12:

```
INSERT INTO vwEmpDetails VALUES ('Jack', 'NYC')
```

Việc chèn này không được phép bởi khung nhìn không chứa cột **LastName** từ bảng cơ sở và cột đó không
cho phép các giá trị null.

10.3.2 UPDATE với các khung nhìn

Có thể sử dụng câu lệnh **UPDATE** để thay đổi dữ liệu trong một khung nhìn. Việc cập nhật một khung nhìn
còn cập nhật bảng bên dưới.

Xem xét một ví dụ. Đoạn mã 13 tạo ra một bảng có tên là **Product_Details**.

Đoạn mã 13:

```
CREATE TABLE Product_Details
(
    ProductID int,
    ProductName varchar(30),
    Rate money
)
```

Giả sử một số bản ghi được thêm vào bảng như được trình bày trong hình 10.4.

	ProductID	ProductName	Rate
1	5	DVD Writer	2250.00
2	4	DVD Writer	1250.00
3	6	DVD Writer	1250.00
4	2	External Hard Drive	4250.00
5	3	External Hard Drive	4250.00

Hình 10.4: Các bản ghi trong Product_Details

Đoạn mã 14 tạo ra một khung nhìn dựa trên bảng Product_Details.

Đoạn mã 14:

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName, Rate FROM Product_Details
```

Đoạn mã 15 cập nhật khung nhìn nhằm thay đổi tất cả giá của các máy ghi DVD thành 3000.

Đoạn mã 15:

```
UPDATE vwProduct_Details
SET Rate=3000
WHERE ProductName='DVD Writer'
```

Kết quả của mã này ảnh hưởng đến không chỉ khung nhìn vwProduct_Details, mà còn cả bảng bên dưới nơi khung nhìn đã được tạo ra.

Hình 10.5 trình bày bảng đã cập nhật được tự động cập nhật vì khung nhìn này.

	ProductID	ProductName	Rate
1	5	DVD Writer	3000.00
2	4	DVD Writer	3000.00
3	6	DVD Writer	3000.00
4	2	External Hard Drive	4250.00
5	3	External Hard Drive	4250.00

Hình 10.5: Bảng đã cập nhật

Các kiểu dữ liệu giá trị lớn bao gồm varchar(max), nvarchar(max), và varbinary(max). Để cập nhật dữ liệu có các kiểu dữ liệu giá trị lớn, mệnh đề .WRITE được sử dụng. Mệnh đề .WRITE chỉ ra rằng một phần của giá trị trong cột sẽ được sửa đổi. Mệnh đề .WRITE không thể được sử dụng để cập nhật giá trị NULL trong một cột. Ngoài ra, không thể sử dụng mệnh đề này để đặt giá trị cột thành NULL.

Cú pháp:

```
column_name .WRITE (expression, @Offset, @Length)
```

trong đó:

column_name: chỉ ra tên của cột thuộc kiểu dữ liệu có giá trị lớn.

expression: chỉ ra tên được sao chép vào cột.

@Offset: chỉ ra điểm khởi đầu trong giá trị của cột, tại đó biểu thức được ghi.

@Length: chỉ ra độ dài của phần trong cột.

@Offset và @Length được chỉ ra theo byte cho kiểu dữ liệu varbinary và varchar và theo các ký tự cho kiểu dữ liệu nvarchar.

Giả sử bảng **Product_Details** được sửa đổi để đưa vào cột **Description** có kiểu dữ liệu nvarchar(max).

Khung nhìn được tạo ra dựa trên bảng này, có các cột **ProductName**, **Description**, và **Rate** như được trình bày trong Đoạn mã 16.

Đoạn mã 16:

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName,
Description,
Rate FROM Product_Details
```

Đoạn mã 17 sử dụng câu lệnh UPDATE trên khung nhìn **vwProduct_Details**. Mệnh đề .WRITE được sử dụng để thay đổi giá trị của Internal trong cột Description thành External.

Đoạn mã 17:

```
UPDATE vwProduct_Details
SET Description .WRITE (N'Ex', 0, 2)
WHERE ProductName= 'Ổ cứng di động'
```

Là kết quả của đoạn mã này, tất cả các hàng trong khung nhìn có tên sản phẩm là 'Ổ cứng di động' sẽ được cập nhật với External thay vì Internal trong cột **Description**.

Hình 10.6 trình bày một mẫu đầu ra của khung nhìn sau khi cập nhật.

	ProductName	Description	Rate
1	Hard Disk Drive	Internal 120 GB	3570.00
2	Portable Hard Drive	External Drive 500 GB	5580.00
3	Portable Hard Drive	External Drive 500 GB	5580.00
4	Hard Disk Drive	Internal 120 GB	3570.00
5	Portable Hard Drive	External Drive 500 GB	5580.00

Hình 10.6: Cập nhật khung nhìn

Những quy tắc và hướng dẫn sau đây phải được tuân thủ khi sử dụng câu lệnh UPDATE:

- ➔ Giá trị của cột có thuộc tính IDENTITY không thể được cập nhật.
- ➔ Không thể cập nhật các bản ghi nếu bảng cơ sở có chứa cột TIMESTAMP.
- ➔ Trong khi cập nhật một hàng, nếu ràng buộc hoặc quy tắc bị vi phạm, câu lệnh bị chấm dứt, lỗi được trả về, và không có bản ghi nào được cập nhật.
- ➔ Khi có một phép tự nối với cùng một khung nhìn hoặc bảng cơ sở, câu lệnh UPDATE không làm việc.

10.3.3 DELETE với các khung nhìn

SQL Server cho phép bạn xóa các hàng từ khung nhìn. Các hàng có thể được xóa từ khung nhìn sử dụng câu lệnh DELETE. Khi các hàng được xóa khỏi khung nhìn, các hàng tương ứng sẽ bị xóa khỏi bảng cơ sở.

Ví dụ, hãy xem xét khung nhìn `vwCustDetails` liệt kê thông tin tài khoản của các khách hàng khác nhau. Khi khách hàng đóng tài khoản, những chi tiết của khách hàng này cần phải được xóa. Điều này được thực hiện bằng cách sử dụng câu lệnh DELETE.

Cú pháp sau đây được sử dụng để xóa dữ liệu khỏi khung nhìn.

Cú pháp:

```
DELETE FROM <view_name>
WHERE <search_condition>
```

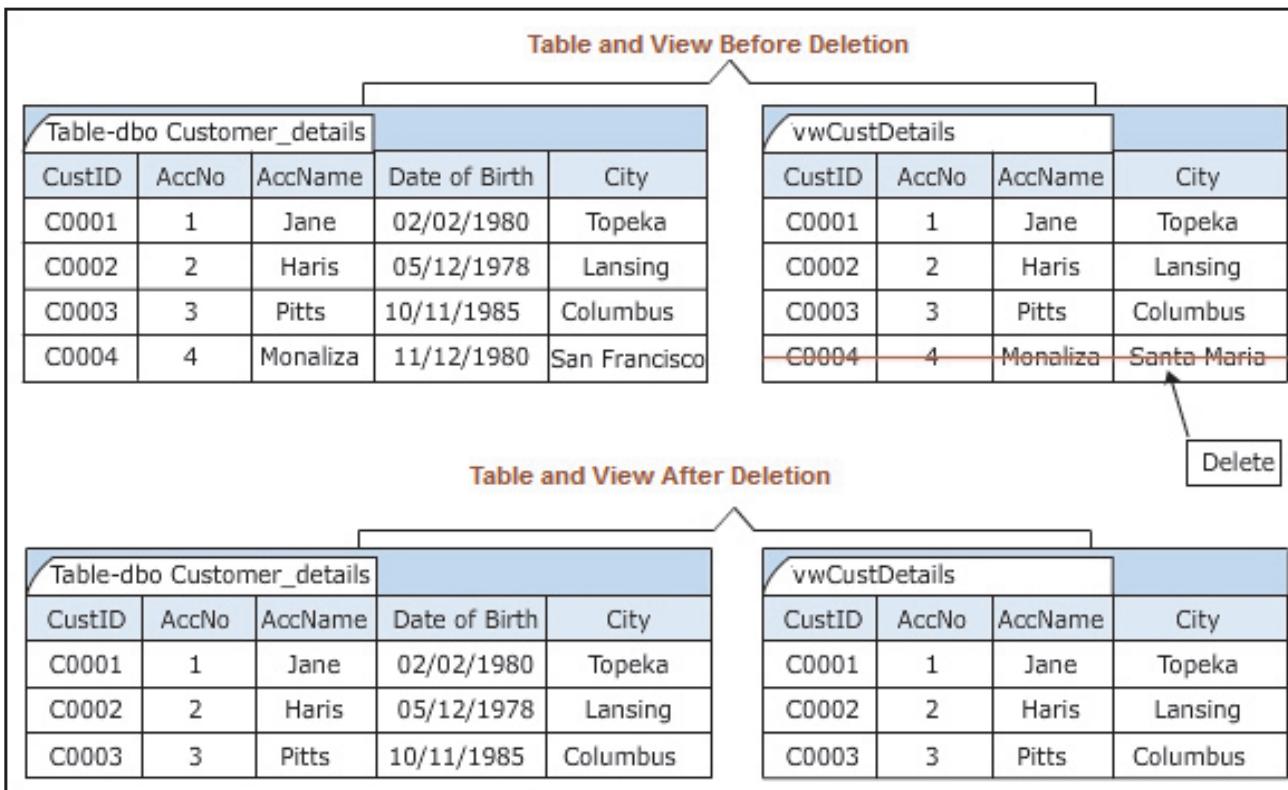
Giả sử rằng bảng có tên là `Customer_Details` và khung nhìn `vwCustDetails` dựa trên bảng này được tạo ra.

Đoạn mã 18 được sử dụng để xóa bản ghi từ khung nhìn `vwCustDetails` có `CustID` C0004.

Đoạn mã 18:

```
DELETE FROM vwCustDetails WHERE CustID='C0004'
```

Hình 10.7 mô tả logic việc xóa khỏi các khung nhìn.



Hình 10.7: Xóa khỏi các khung nhìn

10.4 Thay đổi các khung nhìn

Bên cạnh việc sửa đổi dữ liệu trong khung nhìn, người dùng cũng có thể thay đổi định nghĩa của khung nhìn. Có thể sửa đổi hoặc thay đổi khung nhìn bằng cách thả bỏ và tái tạo nó hoặc thực thi câu lệnh ALTER VIEW. Câu lệnh ALTER VIEW sửa đổi khung nhìn hiện có mà không cần phải tổ chức lại các cấp quyền của nó và các thuộc tính khác. ALTER VIEW có thể được áp dụng cho các khung nhìn có chỉ mục, tuy nhiên, nó thả bỏ vô điều kiện tất cả các chỉ mục trên khung nhìn đó.

Các khung nhìn thường được thay đổi khi người dùng yêu cầu thêm thông tin hoặc thay đổi trong định nghĩa bảng bên dưới.

Ghi chú - Sau khi khung nhìn được tạo ra, nếu cấu trúc của các bảng bên dưới của nó được thay đổi bằng cách thêm các cột, những cột mới này không xuất hiện trong khung nhìn. Điều này là do danh sách cột được giải thích chỉ khi bạn lần đầu tiên tạo ra khung nhìn. Để xem các cột mới trong khung nhìn, bạn phải thay đổi khung nhìn.

Cú pháp sau đây được sử dụng để thay đổi khung nhìn.

Cú pháp:

```
ALTER VIEW <view_name>
AS <select_statement>
```

Đoạn mã 19 thay đổi khung nhìn vwProductInfo để đưa vào cột ReOrderPoint.

Đoạn mã 19:

```
ALTER VIEW vwProductInfo AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel, ReOrderPoint
FROM Production.Product;
GO
```

10.5 Thả bỏ các khung nhìn

Có thể loại bỏ khung nhìn khỏi cơ sở dữ liệu nếu nó không còn cần thiết. Điều này được thực hiện bằng cách sử dụng câu lệnh `DROP VIEW`. Khi khung nhìn được thả bỏ, dữ liệu trong các bảng cơ sở vẫn không bị ảnh hưởng. Định nghĩa của khung nhìn và thông tin khác liên quan đến khung nhìn sẽ bị xóa khỏi danh mục hệ thống. Tất cả các sự cho phép cho khung nhìn đó cũng sẽ bị xóa. Nếu người dùng truy vấn bất kỳ khung nhìn nào tham chiếu tới khung nhìn đã thả bỏ, người dùng sẽ nhận được thông báo lỗi.

Ghi chú- Chủ sở hữu khung nhìn có sự cho phép để thả bỏ khung nhìn và sự cho phép này là không thể chuyển nhượng. Tuy nhiên, quản trị viên hệ thống hoặc chủ sở hữu cơ sở dữ liệu có thể thả bỏ bất kỳ đối tượng nào bằng cách xác định tên chủ sở hữu trong câu lệnh `DROP VIEW`.

Cú pháp sau đây được sử dụng để thả bỏ khung nhìn.

Cú pháp:

```
DROP VIEW <view_name>
```

Đoạn mã 20 xóa khung nhìn vwProductInfo.

Đoạn mã 20:

```
DROP VIEW vwProductInfo
```

10.6 Định nghĩa của khung nhìn

Định nghĩa của khung nhìn giúp hiểu dữ liệu của nó được lấy từ các bảng nguồn như thế nào. Có một số thủ tục lưu trữ `sp_helptext` hiển thị thông tin liên quan đến khung nhìn khi tên của khung nhìn được cho là tham số của nó. Có thể thu được thông tin về định nghĩa của khung nhìn nếu thông tin đó không bị mã hóa.

Cú pháp sau đây được sử dụng để xem thông tin định nghĩa của khung nhìn.

Cú pháp:

```
sp_helptext <view_name>
```

Sử dụng khung nhìn, thủ tục đã lưu trữ và truy vấn siêu dữ liệu

Đoạn mã 21 hiển thị thông tin về khung nhìn `vwProductPrice`.

Đoạn mã 21:

```
EXEC sp_helptext vwProductPrice
```

Việc thực thi đoạn mã này sẽ hiển thị định nghĩa về khung nhìn như được trình bày trong hình 10.8.

Text	
1	CREATE VIEW vwProductPrice AS
2	SELECT ProductID, ProductNumber, Name, SafetySto...
3	FROM Production.Product;

Hình 10.8: Sử dụng `sp_helptext` để hiển thị các định nghĩa khung nhìn

10.7 Tạo một khung nhìn sử dụng các hàm dựng sẵn

Có thể tạo ra các khung nhìn bằng cách sử dụng các hàm dựng sẵn của SQL Server. Khi các chức năng được sử dụng, cột xuất phát phải bao gồm tên cột trong câu lệnh `CREATE VIEW`.

Hãy xem xét khung nhìn đã được tạo ra trong Đoạn mã 16. Nó đã được tái tạo trong Đoạn mã 22 để sử dụng hàm `AVG()`.

Đoạn mã 22:

```
CREATE VIEW vwProduct_Details
AS
SELECT
ProductName,
AVG(Rate) AS AverageRate
FROM Product_Details
GROUP BY ProductName
```

Ở đây, hàm `AVG()` tính toán tỷ lệ trung bình của các sản phẩm tương tự bằng cách sử dụng mệnh đề `GROUP BY`.

Hình 10.9 trình bày kết quả khi khung nhìn được truy vấn.

	ProductName	AverageRate
1	Hard Disk Drive	3570.00
2	Portable Hard Drive	5580.00

Hình 10.9: Sử dụng các hàm tích hợp với các khung nhìn

10.8 CHECK OPTION

CHECK OPTION là một tùy chọn liên quan đến câu lệnh CREATE VIEW. Nó được sử dụng để đảm bảo rằng tất cả các bản cập nhật trong khung nhìn đáp ứng những điều kiện đã nêu trong định nghĩa khung nhìn. Nếu những điều kiện không được thỏa mãn, công cụ cơ sở dữ liệu trả về lỗi. Như vậy, CHECK OPTION được sử dụng để thực thi tính toàn vẹn miền, nó sẽ kiểm tra định nghĩa của khung nhìn để thấy rằng các điều kiện WHERE trong câu lệnh SELECT không bị vi phạm.

Mệnh đề WITH CHECK OPTION bắt buộc tất cả các câu lệnh sửa đổi đã thực thi đối với khung nhìn này để thực hiện theo điều kiện được đặt trong câu lệnh SELECT. Khi hàng được sửa đổi, WITH CHECK OPTION đảm bảo rằng dữ liệu vẫn có thể nhìn thấy thông qua khung nhìn.

Cú pháp sau đây tạo ra một khung nhìn sử dụng CHECK OPTION.

Cú pháp:

```
CREATE VIEW <view_name>
AS select_statement [ WITH CHECK OPTION ]
```

trong đó:

WITH CHECK OPTION: chỉ ra rằng dữ liệu đã thay đổi trong khung nhìn tiếp tục đáp ứng định nghĩa khung nhìn.

Đoạn mã 23 tạo ra lại khung nhìn **vwProductInfo** có SafetyStockLevel nhỏ hơn hoặc bằng 1000.

Đoạn mã 23:

```
CREATE VIEW vwProductInfo AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel,
ReOrderPoint
FROM Production.Product
WHERE SafetyStockLevel <=1000
WITH CHECK OPTION;
GO
```

Trong Đoạn mã 24, câu lệnh UPDATE được sử dụng để sửa đổi khung nhìn **vwProductInfo** bằng cách thay đổi giá trị của cột SafetyStockLevel cho sản phẩm có id 321 đến 2500.

Đoạn mã 24:

```
UPDATE vwProductInfo SET SafetyStockLevel=2500
WHERE ProductID=321
```

Câu lệnh UPDATE không thực thi được vì nó vi phạm định nghĩa khung nhìn, chỉ ra rằng SafetyStockLevel phải nhỏ hơn hoặc bằng 1000. Do đó, không có hàng bị ảnh hưởng trong khung nhìn **vwProductInfo**.

Ghi chú - Bất kỳ bản cập nhật nào được thực hiện trên các bảng cơ sở không được xác nhận đối với khung nhìn này, ngay cả khi CHECK OPTION được chỉ ra.

10.9 Tùy chọn SCHEMABINDING

Có thể liên kết khung nhìn với cấu trúc của bảng cơ sở sử dụng tùy chọn SCHEMABINDING. Có thể sử dụng tùy chọn này với câu lệnh CREATE VIEW hoặc ALTER VIEW. Khi tùy chọn SCHEMABINDING được chỉ ra, khung nhìn sửa đổi bảng cơ sở sẽ ảnh hưởng đến các định nghĩa khung nhìn. Định nghĩa khung nhìn trước tiên phải được sửa đổi hoặc xóa để loại bỏ sự phụ thuộc vào bảng sẽ được sửa đổi.

Trong khi sử dụng tùy chọn SCHEMABINDING trong khung nhìn, bạn phải chỉ ra tên lược đồ cùng với tên đối tượng trong câu lệnh SELECT.

Cú pháp sau đây được sử dụng để tạo ra một khung nhìn với tùy chọn SCHEMABINDING.

Cú pháp:

```
CREATE VIEW <view_name> WITH SCHEMABINDING
AS <select_statement>
```

trong đó:

index_name: chỉ ra tên của khung nhìn.

WITH SCHEMABINDING: chỉ ra rằng khung nhìn phải được ràng buộc với một lược đồ.

select_statement: Chỉ ra câu lệnh SELECT định nghĩa khung nhìn đó.

Đoạn mã 25 tạo ra khung nhìn **vwNewProductInfo** với tùy chọn SCHEMABINDING để ràng buộc khung nhìn với lược đồ Production, là lược đồ của bảng Product.

Đoạn mã 25:

```
CREATE VIEW vwNewProductInfo
WITH SCHEMABINDING AS
SELECT ProductID, ProductNumber, Name, SafetyStockLevel
FROM Production.Product;
GO
```

10.10 Sử dụng sp_refreshview

Trong khi tạo một khung nhìn, tùy chọn SCHEMABINDING được sử dụng để ràng buộc khung nhìn với lược đồ của những bảng được đưa vào trong khung nhìn. Tuy nhiên, cũng có thể tạo ra khung nhìn mà không chọn tùy chọn SCHEMABINDING. Trong trường hợp này, nếu thay đổi được thực hiện cho các đối tượng nằm dưới (bảng hoặc khung nhìn) để khung nhìn phụ thuộc vào đó, thủ tục lưu trữ sp_refreshview phải được thực thi. Thủ tục lưu trữ sp_refreshview cập nhật siêu dữ liệu cho khung nhìn. Nếu thủ tục sp_refreshview không được thực hiện, siêu dữ liệu của khung nhìn không được cập nhật để phản ánh những thay đổi trong những bảng cơ sở này. Điều này dẫn đến việc tạo ra các kết quả bất ngờ khi khung nhìn được truy vấn.

Thủ tục lưu trữ sp_refreshview trả về giá trị mã số 0 nếu việc thực thi thành công hoặc trả về một số khác số 0 trong trường hợp việc thực thi đã thất bại.

Cú pháp sau đây được sử dụng để chạy thủ tục lưu trữ sp_refreshview.

Cú pháp:

```
sp_refreshview '<view_name>'
```

Đoạn mã 26 tạo ra bảng **Customers** với các cột **CustID**, **CustName**, và **Address**.

Đoạn mã 26:

```
CREATE TABLE Customers
(
    CustID int,
    CustName varchar(50),
    Address varchar(60)
)
```

Đoạn mã 27 tạo ra khung nhìn **vwCustomers** dựa trên bảng **Customers**.

Đoạn mã 27:

```
CREATE VIEW vwCustomers
AS
SELECT * FROM Customers
```

Đoạn mã 28 thực thi truy vấn SELECT trên khung nhìn.

Đoạn mã 28:

```
SELECT * FROM vwCustomers
```

Đầu ra của Đoạn mã 28 trình bày ba cột **CustID**, **CustName**, và **Address**.

Sử dụng khung nhìn, thủ tục đã lưu trữ và truy vấn siêu dữ liệu

Đoạn mã 29 sử dụng câu lệnh ALTER TABLE để thêm một cột **Age** vào bảng **Customers**.

Đoạn mã 29:

```
ALTER TABLE Customers ADD Age int
```

Đoạn mã 30 thực thi truy vấn SELECT trên khung nhìn.

Đoạn mã 30:

```
SELECT * FROM vwCustomers
```

Cột đã cập nhật **Age** không được nhìn thấy trong khung nhìn.

Để giải quyết điều này, thủ tục lưu trữ **sp_refreshview** phải được thực thi trên khung nhìn **vwCustomers** như được trình bày trong Đoạn mã 31.

Đoạn mã 31:

```
EXEC sp_refreshview 'vwCustomers'
```

Khi truy vấn SELECT được chạy một lần nữa trên khung nhìn, cột **Age** được nhìn thấy trong kết quả. Điều này là do thủ tục **sp_refreshview** làm mới siêu dữ liệu cho khung nhìn **vwCustomers**.

Không thể thả bỏ các bảng được gắn với lược đồ vào khung nhìn trừ khi khung nhìn được thả bỏ hoặc thay đổi làm sao nó không có ràng buộc lược đồ nào. Nếu khung nhìn không được thả bỏ hoặc thay đổi và bạn cố gắng thả bỏ bảng, Công cụ cơ sở dữ liệu trả về một thông báo lỗi.

Ngoài ra, khi câu lệnh ALTER TABLE ảnh hưởng đến định nghĩa khung nhìn của một khung nhìn có gắn lược đồ, câu lệnh ALTER TABLE thất bại.

Hãy xem xét khung nhìn có gắn lược đồ đã được tạo ra trong Đoạn mã 25. Nó phụ thuộc vào bảng Production.Product.

Đoạn mã 32 cố gắng sửa đổi kiểu dữ liệu của cột **ProductID** trong bảng Production.Product từ int thành varchar(7).

Đoạn mã 32:

```
ALTER TABLE Production.Product ALTER COLUMN ProductID varchar(7)
```

Công cụ cơ sở dữ liệu trả về thông báo lỗi như bảng được ràng buộc lược đồ với khung nhìn **vwNewProductInfo** và do đó, không thể thay đổi được bởi nó vi phạm định nghĩa khung nhìn của khung nhìn.

10.11 Các thủ tục lưu trữ

Thủ tục lưu trữ là một nhóm các câu lệnh Transact-SQL hành động như một khối mã duy nhất thực hiện một nhiệm vụ cụ thể. Khối mã này được xác định bằng một tên được gán và được lưu trữ trong cơ sở dữ liệu ở dạng đã biên soạn. Thủ tục lưu trữ cũng có thể là tham chiếu đến một phương pháp .NET Framework Common Language Runtime (CLR).

Thủ tục lưu trữ rất hữu ích các nhiệm vụ lặp lại phải được thực hiện. Điều này giúp loại bỏ sự cần thiết phải gõ lặp lại nhiều câu lệnh Transact-SQL và sau đó biên dịch lặp lại chúng.

Các thủ tục lưu trữ có thể nhận các giá trị ở dạng các tham số đầu vào và trả về các giá trị đầu ra như được định nghĩa bằng các tham số đầu ra.

Việc sử dụng các thủ tục lưu trữ đem lại nhiều lợi thế hơn so với việc sử dụng câu lệnh Transact-SQL. Chúng bao gồm:

→ **Cải thiện bảo mật**

Người quản trị cơ sở dữ liệu có thể cải thiện bảo mật bằng cách kết hợp các đặc quyền cơ sở dữ liệu với các thủ tục được lưu trữ. Người dùng có thể được cho phép để thực hiện một thủ tục lưu trữ ngay cả khi người dùng không có sự cho phép để truy cập những bảng hoặc khung nhìn này.

→ **Thực thi biên dịch sẵn**

Các thủ tục lưu trữ được biên dịch trong quá trình thực thi đầu tiên. Đối với mỗi lần thực thi tiếp theo, SQL Server tái sử dụng phiên bản biên dịch sẵn này. Điều này làm giảm thời gian và nguồn lực cần thiết để biên dịch.

→ **Lưu lượng máy khách/máy chủ giảm**

Các thủ tục lưu trữ giúp làm giảm lưu lượng mạng. Khi các câu lệnh Transact-SQL được thực thi một cách riêng biệt, có mức sử dụng mạng riêng để thực thi mỗi câu lệnh. Khi một thủ tục lưu trữ được thực thi, các câu lệnh Transact-SQL được thực thi cùng nhau như một đơn vị đơn lẻ. Đường dẫn mạng không được sử dụng một cách riêng biệt cho việc thực thi từng câu lệnh riêng lẻ. Điều này làm giảm lưu lượng mạng.

→ **Tái sử dụng mã**

Các thủ tục lưu trữ có thể được sử dụng nhiều lần. Điều này giúp loại bỏ sự cần thiết phải gõ lặp lại hàng trăm câu lệnh Transact-SQL mỗi khi một nhiệm vụ tương tự sẽ được thực hiện.

10.11.1 Các loại thủ tục lưu trữ

SQL Server hỗ trợ nhiều loại thủ tục lưu trữ khác nhau. Những công cụ này được mô tả như sau:

→ **Các thủ tục lưu trữ do người dùng định nghĩa**

Các thủ tục lưu trữ do người dùng định nghĩa còn được gọi là các thủ tục lưu trữ tùy chỉnh. Những thủ tục này được sử dụng để tái sử dụng các câu lệnh Transact-SQL để thực hiện các nhiệm vụ lặp lại. Có hai loại thủ tục lưu trữ do người dùng định nghĩa, những thủ tục lưu trữ Transact-SQL và thủ tục lưu trữ Common Language Runtime (CLR).

Thủ tục lưu trữ Transact-SQL bao gồm các câu lệnh Transact-SQL trong khi các thủ tục lưu trữ CLR được dựa trên những phương pháp CLR của khuôn khổ .NET. Cả hai thủ tục lưu trữ có thể lấy và trả về các tham số do người dùng định nghĩa.

→ Các thủ tục lưu trữ mở rộng

Thủ tục lưu trữ mở rộng giúp SQL Server trong việc tương tác với hệ điều hành. Thủ tục lưu trữ mở rộng không phải là đối tượng cư trú của SQL Server. Chúng là các thủ tục được thực hiện như các thư viện liên kết động (DLL) được thực thi bên ngoài môi trường SQL Server. Ứng dụng tương tác với SQL Server gọi DLL tại thời gian chạy. DLL được tải tự động và chạy bằng SQL Server. SQL Server phân bổ không gian để chạy các thủ tục lưu trữ mở rộng. Các thủ tục lưu trữ mở rộng sử dụng tiền tố 'xp'. Các nhiệm vụ phức tạp hoặc không thể được thực thi sử dụng các câu lệnh Transact-SQL được thực hiện bằng cách sử dụng thủ tục lưu trữ mở rộng.

→ Các thủ tục lưu trữ hệ thống

Thủ tục lưu trữ hệ thống thường được sử dụng để tương tác với các bảng hệ thống và thực hiện các tác vụ hành chính như cập nhật các bảng hệ thống. Thủ tục lưu trữ hệ thống được bắt đầu với 'sp_'. Những thủ tục này nằm trong cơ sở dữ liệu Resource. Những thủ tục này có thể được nhìn thấy trong lược đồ sys của mỗi hệ thống và cơ sở dữ liệu do người dùng định nghĩa. Thủ tục lưu trữ hệ thống cho phép các quyền GRANT, DENY, và REVOKE.

Thủ tục lưu trữ hệ thống là một tập hợp các câu lệnh Transact-SQL biên dịch sẵn được thực thi như một đơn vị đơn lẻ. Thủ tục hệ thống được sử dụng trong các hoạt động hành chính và thông tin cơ sở dữ liệu. Những thủ tục này đem lại khả năng dễ dàng truy cập vào thông tin siêu dữ liệu về các đối tượng cơ sở dữ liệu như các bảng hệ thống, bảng do người dùng định nghĩa, khung nhìn, và chỉ mục.

Thủ tục lưu trữ hệ thống xuất hiện một cách logic trong lược đồ sys của hệ thống và các cơ sở dữ liệu do người dùng định nghĩa. Khi tham chiếu một lưu trữ thủ tục hệ thống, mã định danh lược đồ sys được sử dụng. Thủ tục lưu trữ hệ thống được lưu trữ vật lý trong cơ sở dữ liệu Resource ẩn và có tiền tố sp_. Thủ tục lưu trữ hệ thống được sở hữu bởi người quản trị cơ sở dữ liệu.

Ghi chú - Bảng hệ thống được tạo ra theo mặc định tại thời điểm tạo ra một cơ sở dữ liệu mới. Những bảng này lưu trữ thông tin siêu dữ liệu về các đối tượng do người dùng định nghĩa như các bảng và khung nhìn. Người dùng không thể truy cập hoặc cập nhật các bảng hệ thống sử dụng các thủ tục lưu trữ hệ thống ngoại trừ thông qua các quyền được cấp bởi người quản trị cơ sở dữ liệu.

10.11.2 Phân loại các thủ tục lưu trữ hệ thống

Các thủ tục lưu trữ hệ thống có thể được phân loại thành các thể loại khác nhau tùy thuộc vào các tác vụ mà chúng thực hiện. Một số thể loại quan trọng như sau:

→ Các thủ tục lưu trữ danh mục

Tất cả thông tin về các bảng trong cơ sở dữ liệu người dùng được lưu trữ trong một tập hợp các bảng đã gọi danh mục hệ thống đó. Thông tin từ danh mục hệ thống có thể được truy cập bằng cách sử dụng các thủ tục danh mục. Ví dụ, thủ tục lưu trữ danh mục sp_tables hiển thị danh sách của tất cả các bảng trong cơ sở dữ liệu hiện tại.

→ Các thủ tục lưu trữ bảo mật

Thủ tục lưu trữ bảo mật được sử dụng để quản lý tính bảo mật của cơ sở dữ liệu. Ví dụ, thủ tục lưu trữ bảo mật

`sp_changedbowner` được sử dụng để thay đổi chủ sở hữu của cơ sở dữ liệu hiện tại.

→ Các thủ tục lưu trữ con trỏ

Thủ tục con trỏ được sử dụng để thực hiện chức năng của một con trỏ. Ví dụ, thủ tục lưu trữ con trỏ `sp_cursor_list` liệt kê tất cả các con trỏ được mở bằng kết nối này và mô tả các thuộc tính của chúng.

→ Các thủ tục lưu trữ truy vấn phân phối

Thủ tục lưu trữ phân phối được sử dụng trong việc quản lý các truy vấn phân phối. Ví dụ, thủ tục lưu trữ truy vấn phân phối `sp_indexes` trả về thông tin chỉ mục cho bảng từ xa đã chỉ định.

→ Các thủ tục lưu trữ Database Mail và SQL Mail

Các thủ tục lưu trữ Database Mail và SQL Mail được sử dụng để thực hiện các hoạt động e-mail từ bên trong SQL Server. Ví dụ, thủ tục lưu trữ thư cơ sở dữ liệu `sp_send_dbmail` sẽ gửi các thông báo e-mail đến người nhận đã chỉ định. Thông báo này có thể bao gồm một tập kết quả truy vấn hoặc tập tin đính kèm hoặc cả hai.

10.11.3 Các thủ tục lưu trữ tạm thời

Thủ tục lưu trữ được tạo ra để sử dụng tạm thời trong một phiên được gọi là thủ tục lưu trữ tạm thời. Những thủ tục này được lưu trữ trong cơ sở dữ liệu `tempdb`. Cơ sở dữ liệu hệ thống `tempdb` là một nguồn lực toàn cầu có sẵn cho tất cả người dùng được kết nối với một thể hiện của SQL Server. Nó chứa tất cả các bảng tạm thời và các thủ tục lưu trữ tạm thời.

SQL Server hỗ trợ hai loại thủ tục lưu trữ tạm thời cụ thể là cục bộ và toàn cầu. Sự khác biệt giữa hai loại này được đưa ra trong bảng 10.1.

Thủ tục tạm thời cục bộ	Thủ tục tạm thời toàn cầu
Chỉ thấy được cho người dùng đã tạo ra nó	Thấy được tất cả người dùng
Thả bỏ vào cuối phiên hiện tại	Thả bỏ vào cuối phiên cuối cùng
Thủ tục tạm thời cục bộ	Thủ tục tạm thời toàn cầu
Chỉ có thể được sử dụng bởi chủ nhân của nó	Có thể được sử dụng bởi bất kỳ người dùng nào
Sử dụng tiền tố # trước tên thủ tục	Sử dụng tiền tố ## trước tên thủ tục

Bảng 10.1: Sự khác nhau giữa thủ tục tạm thời cục bộ và toàn cầu

Ghi chú - Phiên được thiết lập khi người dùng kết nối với cơ sở dữ liệu và kết thúc khi dùng ngắt kết nối.

Tên đầy đủ của một thủ tục lưu trữ tạm thời toàn cầu bao gồm cả tiền tố ## không thể vượt quá 128 ký tự. Tên đầy đủ của một thủ tục lưu trữ tạm thời cục bộ bao gồm cả tiền tố # không thể vượt quá 116 ký tự.

10.11.4 Các thủ tục lưu trữ từ xa

Thủ tục lưu trữ chạy trên SQL Server từ xa được gọi là thủ tục lưu trữ từ xa. Thủ tục lưu trữ từ xa có thể được sử dụng chỉ khi máy chủ từ xa cho phép truy cập từ xa.

Khi một thủ tục lưu trữ từ xa được thực thi từ một thể hiện cục bộ của SQL Server đến máy tính khách hàng, có thể gặp phải lỗi hủy câu lệnh. Khi một lỗi như vậy xảy ra, câu lệnh đã gây ra lỗi bị chấm dứt nhưng thủ tục từ xa tiếp tục được thực hiện.

10.11.5 Các thủ tục lưu trữ mở rộng

Thủ tục lưu trữ mở rộng được sử dụng để thực hiện các tác vụ mà không thể được thực hiện sử dụng các câu lệnh Transact-SQL tiêu chuẩn. Các thủ tục lưu trữ mở rộng sử dụng tiền tố 'xp_'. Những thủ tục lưu trữ này được chứa trong lược đồ dbo của cơ sở dữ liệu tổng thể.

Cú pháp sau đây được sử dụng để thực thi thủ tục lưu trữ mở rộng.

Cú pháp:

```
EXECUTE <procedure_name>
```

Đoạn mã 33 thực thi thủ tục lưu trữ mở rộng xp_fileexist để kiểm tra xem tập tin MyTest.txt có tồn tại hay không.

Đoạn mã 33:

```
EXECUTE xp_fileexist 'c:\MyTest.txt'
```

Ghi chú - Khi bạn thực thi một thủ tục lưu trữ mở rộng, hoặc trong một khối lệnh hoặc trong một môđun, xác định tên thủ tục lưu trữ với master.dbo.

10.12 Các thủ tục lưu trữ tùy chỉnh hoặc do người dùng định nghĩa

Trong SQL Server, người dùng được phép tạo ra các thủ tục lưu trữ tùy chỉnh để thực hiện các tác vụ khác nhau. Các thủ tục lưu trữ như vậy được gọi là thủ tục lưu trữ do người dùng định nghĩa hoặc tùy chỉnh.

Ví dụ, hãy xem xét bảng **Customer_Details** lưu trữ các chi tiết về tất cả các khách hàng. Bạn sẽ cần phải gõ các câu lệnh Transact-SQL mỗi khi bạn muốn xem các chi tiết về khách hàng. Thay vào đó, bạn có thể tạo ra một thủ tục lưu trữ tùy chỉnh sẽ hiển thị những chi tiết này bất cứ khi nào thủ tục đó được thực thi.

Tạo ra một thủ tục lưu trữ tùy chỉnh yêu cầu quyền CREATE PROCEDURE trong cơ sở dữ liệu và quyền ALTER trên lược đồ trong đó thủ tục đang được tạo ra.

Cú pháp sau đây được sử dụng để tạo ra thủ tục lưu trữ tùy chỉnh.

Cú pháp:

```
CREATE { PROC | PROCEDURE } procedure_name
[ { @parameter data_type } ]
AS <sql_statement>
```

trong đó:

`procedure_name`: chỉ ra tên của thủ tục.

`@parameter`: chỉ ra các tham số đầu vào/đầu ra trong thủ tục.

`data_type`: chỉ ra kiểu dữ liệu của các tham số.

`sql_statement`: chỉ ra một hoặc nhiều câu lệnh Transact-SQL được đưa vào trong thủ tục.

Đoạn mã 34 tạo ra và sau đó thực thi một thủ tục lưu trữ tùy chỉnh, `uspGetCustTerritory`, sẽ hiển thị các chi tiết của khách hàng như mã khách hàng, mã lãnh thổ, và tên lãnh thổ.

Đoạn mã 34:

```
CREATE PROCEDURE uspGetCustTerritory
AS
SELECT TOP 10 CustomerID, Customer.TerritoryID, Sales.SalesTerritory.Name
FROM Sales.Customer JOIN Sales.SalesTerritory ON Sales.Customer.TerritoryID =
Sales.SalesTerritory.TerritoryID
```

Để thực thi thủ tục lưu trữ này, lệnh `EXEC` được sử dụng như được trình bày trong Đoạn mã 35.

Đoạn mã 35:

```
EXEC uspGetCustTerritory
```

Kết quả được trình bày trong hình 10.10.

	CustomerID	TerritoryID	Name
1	15	9	Australia
2	33	9	Australia
3	51	9	Australia
4	69	9	Australia
5	87	9	Australia
6	105	9	Australia
7	123	9	Australia
8	141	9	Australia
9	159	9	Australia
10	177	9	Australia

Hình 10.10: Đầu ra của một thủ tục lưu trữ đơn giản

10.12.1 Sử dụng các tham số

Lợi thế thực sự của một thủ tục lưu trữ đi vào hình ảnh chỉ khi một hoặc nhiều tham số được sử dụng với nó. Dữ liệu được truyền giữa thủ tục lưu trữ và chương trình gọi khi việc gọi được thực hiện cho một thủ tục lưu trữ. Truyền dữ liệu này được thực hiện bằng cách sử dụng các tham số. Tham số thuộc hai loại như sau:

➔ Các tham số đầu vào

Tham số đầu vào cho phép chương trình gọi truyền các giá trị cho một thủ tục lưu trữ. Những giá trị này được nhận vào các biến đã định nghĩa trong thủ tục lưu trữ.

➔ Các tham số đầu ra

Các tham số đầu ra cho phép thủ tục lưu trữ truyền các giá trị trả lại chương trình gọi. Những giá trị này được nhận vào các biến bằng chương trình gọi.

Những cái này bây giờ được mô tả chi tiết.

➔ Các tham số đầu vào

Các giá trị được truyền từ chương trình gọi đến thủ tục lưu trữ và những giá trị này được chấp nhận vào các tham số đầu vào của thủ tục lưu trữ đó. Các tham số đầu vào được định nghĩa vào thời điểm tạo ra thủ tục lưu trữ. Những giá trị này đã truyền vào các tham số đầu vào có thể là hằng số hoặc biến. Những giá trị này được truyền cho thủ tục tại thời điểm gọi thủ tục. Thủ tục lưu trữ thực hiện các tác vụ đã chỉ định sử dụng những giá trị này.

Cú pháp sau đây được sử dụng để tạo ra thủ tục lưu trữ.

Cú pháp:

```
CREATE PROCEDURE <procedure_name>
@parameter <data_type>
AS <sql_statement>
```

trong đó:

`data_type`: chỉ ra kiểu dữ liệu do hệ thống định nghĩa.

Cú pháp sau đây được sử dụng để thực thi một thủ tục lưu trữ và truyền giá làm các tham số đầu vào.

Cú pháp:

```
EXECUTE <procedure_name> <parameters>
```

Đoạn mã 36 tạo ra thủ tục lưu trữ **uspGetSales** với tham số **territory** chấp nhận tên của lãnh thổ và hiển thị các chi tiết bán hàng và mã nhân viên bán hàng cho lãnh thổ đó. Sau đó, đoạn mã thực thi thủ tục lưu trữ với Northwest được truyền làm tham số đầu vào.

Đoạn mã 36:

```
CREATE PROCEDURE uspGetSales
@territory varchar(40)
AS
SELECT BusinessEntityID, B.SalesYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID=B.TerritoryID
WHERE B.Name = @territory;

--Thực thi thủ tục lưu trữ
EXEC uspGetSales 'Northwest'
```

Kết quả được trình bày trong hình 10.11.

	BusinessEntityID	SalesYTD	SalesLastYear
1	280	7887186.7882	3298694.4938
2	283	7887186.7882	3298694.4938
3	284	7887186.7882	3298694.4938

Hình 10.11: Sử dụng thủ tục lưu trữ với các tham số

→ Các tham số đầu ra

Thủ tục lưu trữ đôi khi cần phải trả kết quả ngược lại cho chương trình gọi. Việc truyền dữ liệu này từ thủ tục lưu trữ cho chương trình gọi được thực hiện bằng cách sử dụng các tham số đầu ra.

Các tham số đầu ra được định nghĩa vào thời điểm tạo ra thủ tục. Để chỉ ra một tham số đầu ra, từ khóa **OUTPUT** được sử dụng trong khi khai báo tham số. Ngoài ra, câu lệnh gọi phải có một biến đã chỉ định với từ khóa **OUTPUT** để chấp nhận đầu ra từ thủ tục được gọi.

Cú pháp sau đây được sử dụng để truyền các tham số đầu ra trong một thủ tục lưu trữ và sau đó, thực thi thủ tục lưu trữ với tham số **OUTPUT** được chỉ định.

Cú pháp:

```
EXECUTE <procedure_name> <parameters>
```

Đoạn mã 37 tạo ra thủ tục lưu trữ **uspGetTotalSales** với tham số đầu vào **@territory** nhận tên của lãnh thổ và tham số đầu ra **@sum** để hiển thị tổng doanh thu năm đến nay trên lãnh thổ đó.

Đoạn mã 37:

```
CREATE PROCEDURE uspGetTotalSales
@territory varchar(40), @sum int OUTPUT
AS
SELECT @sum= SUM(B.SalesYTD)
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID=B.TerritoryID
WHERE B.Name = @territory
```

Đoạn mã 38 khai báo biến **sumsales** để nhận kết quả của thủ tục **uspGetTotalSales**.

Đoạn mã 38:

```
DECLARE @sumsales money;
EXEC uspGetTotalSales 'Northwest', @sum=@sum OUTPUT;
PRINT 'The year-to-date sales figure for this territory is ' +
convert(varchar(100),@sumsales);
GO
```

Đoạn mã này truyền **Northwest** làm đầu vào cho thủ tục lưu trữ **uspGetTotalSales** và nhận kết quả vào biến **sumsales**. Kết quả được in bằng cách sử dụng lệnh **PRINT**.

Các tham số OUTPUT có các đặc điểm sau:

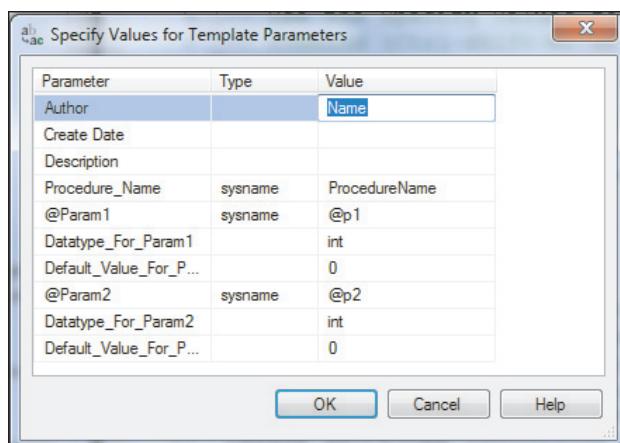
- Tham số không thể thuộc kiểu dữ liệu văn bản và hình ảnh.
- Câu lệnh gọi phải có một biến để nhận giá trị trả về.
- Có thể sử dụng biến trong các câu lệnh Transact-SQL tiếp theo trong khối lệnh hoặc thủ tục gọi.
- Các tham số đầu ra có thể là chỗ dành sẵn con trỏ.

Mệnh đề OUTPUT trả về thông tin từ mỗi hàng trên đó các câu lệnh INSERT, UPDATE, và DELETE đã được thực thi. Mệnh đề này rất hữu ích để lấy giá trị của một nhận dạng hoặc cột tính sau hoạt động INSERT hoặc UPDATE.

10.12.2 Sử dụng SSMS để tạo ra các thủ tục lưu trữ

Bạn cũng có thể tạo ra một thủ tục lưu trữ do người dùng định nghĩa bằng cách sử dụng SSMS. Những bước để thực hiện điều này như sau:

1. Khởi chạy Object Explorer.
2. Trong Object Explorer, kết nối với một thể hiện của Công cụ cơ sở dữ liệu.
3. Sau khi kết nối thành công với thể hiện, mở rộng thể hiện đó.
4. Expand Databases và sau đó, mở rộng cơ sở dữ liệu AdventureWorks2012.
5. Expand Programmability, bấm chuột phải Stored Procedures, và sau đó bấm vào New Stored Procedure.
6. Trên menu Query, bấm vào Specify Values for Template Parameters. Hộp thoại Specify Values for Template Parameters được hiển thị như được trình bày trong hình 10.12.



Hình 10.12: Hộp thoại Specify Values for Template Parameters

7. Trong hộp thoại **Specify Values for Template Parameters**, nhập các giá trị sau cho các tham số như được trình bày trong bảng 10.2.

Tham số	Giá trị
Tác giả	Tên của bạn
Ngày tạo ra	Ngày hôm nay
Mô tả	Trả về số liệu năm bán hàng cho một vùng lãnh thổ
Procedure_Name	uspGetTotals
@Param1	@territory
@Datatype_For_Param1	varchar(50)
Default_Value_For_Param1	NULL
@Param2	
@Datatype_For_Param2	
Default_Value_For_Param2	

Bảng 10.2: Các giá trị tham số

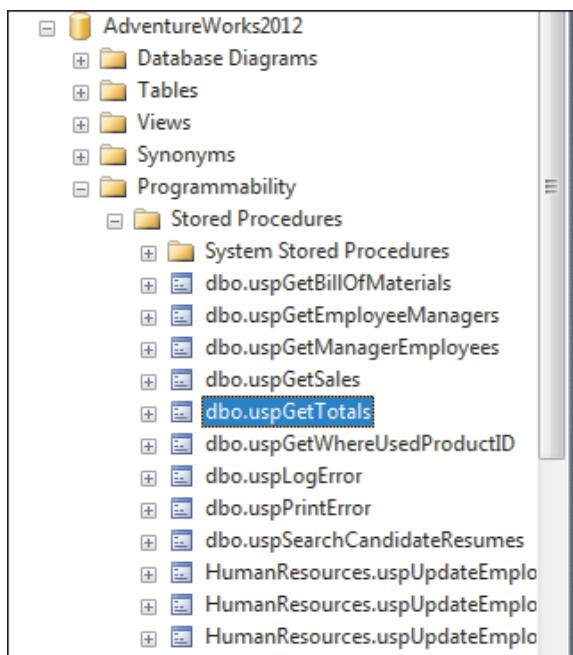
8. Sau khi nhập những chi tiết này, bấm vào **OK**.

9. Trong Query Editor, thay câu lệnh **SELECT** bằng câu lệnh sau:

```
SELECT BusinessEntityID, B.SalesYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory;
```

10. Để kiểm tra cú pháp, trên menu **Query**, bấm vào **Parse**. Nếu thông báo lỗi được trả về, so sánh các câu lệnh với thông tin này và chỉnh sửa khi cần thiết.
11. Để tạo ra thủ tục này, từ menu **Query**, bấm vào **Execute**. Thủ tục được tạo ra như một đối tượng trong cơ sở dữ liệu.
12. Để xem thủ tục được liệt kê trong **Object Explorer**, bấm chuột phải vào **Stored Procedures** và chọn **Refresh**.

Tên thủ tục sẽ được hiển thị trong cây **Object Explorer** như được trình bày trong hình 10.13.



Hình 10.13: Thủ tục lưu trữ được thấy trong Object Explorer

13. Để chạy thủ tục này, trong **Object Explorer**, bấm chuột phải vào tên thủ tục lưu trữ **uspGetTotals** và chọn **Execute Stored Procedure**.
14. Trong cửa sổ **Execute Procedure**, nhập **Northwest** làm giá trị cho tham số **@territory**.

Ghi chú- Trong SQL Server 2012, thủ tục lưu trữ có thể có dung lượng lên đến 250 MB. Nói cách khác, những byte trong văn bản nguồn của một thủ tục lưu trữ không thể vượt quá 250 MB.

10.13 Xem các định nghĩa thủ tục lưu trữ

Định nghĩa của một thủ tục lưu trữ có thể được xem bằng cách sử dụng thủ tục lưu trữ hệ thống `sp_helptext`. Để xem định nghĩa, bạn phải chỉ ra tên của thủ tục lưu trữ làm tham số khi thực thi `sp_helptext`. Định nghĩa này là ở dạng các câu lệnh Transact-SQL.

Các câu lệnh Transact-SQL của định nghĩa thủ tục bao gồm câu lệnh `CREATE PROCEDURE` cũng như các câu lệnh SQL định nghĩa phần thân của thủ tục.

Cú pháp sau đây được sử dụng để xem định nghĩa của thủ tục lưu trữ.

Cú pháp:

```
sp_helptext '<procedure_name>'
```

Sử dụng khung nhìn, thủ tục đã lưu trữ và truy vấn siêu dữ liệu

Đoạn mã 39 hiển thị định nghĩa của thủ tục lưu trữ có tên là `uspGetTotals`.

Đoạn mã 39:

```
EXEC sp_helptext uspGetTotals
```

10.14 Sửa chữa và thả bỏ các thủ tục lưu trữ

Những cấp quyền liên quan đến thủ tục lưu trữ bị mất khi thủ tục lưu trữ được tái tạo. Tuy nhiên, khi một thủ tục lưu trữ được thay đổi, những cấp quyền đã định nghĩa cho thủ tục lưu trữ vẫn như cũ mặc dù định nghĩa thủ tục được thay đổi.

Có thể thay đổi thủ tục bằng cách sử dụng câu lệnh `ALTER PROCEDURE`.

Cú pháp sau đây được sử dụng để sửa đổi thủ tục lưu trữ.

Cú pháp:

```
ALTER PROCEDURE <procedure_name>
@parameter <data_type> [ OUTPUT ]
[ WITH { ENCRYPTION | RECOMPILE } ]
AS <sql_statement>
```

trong đó:

`ENCRYPTION`: mã hóa định nghĩa thủ tục lưu trữ.

`RECOMPILE`: chỉ ra rằng thủ tục được biên dịch vào thời gian chạy.

`sql_statement`: chỉ ra các câu lệnh Transact-SQL được đưa vào trong phần thân của thủ tục.

Đoạn mã 40 sửa đổi định nghĩa của thủ tục lưu trữ có tên là `uspGetTotals` để thêm cột mới `CostYTD` được lấy từ `Sales.SalesTerritory`.

Đoạn mã 40:

```
ALTER PROCEDURE [dbo]. [uspGetTotals]
@territory varchar = 40
AS
    SELECT BusinessEntityID, B.SalesYTD, B.CostYTD, B.SalesLastYear
    FROM Sales.SalesPerson A
    JOIN Sales.SalesTerritory B
    ON A.TerritoryID = B.TerritoryID
    WHERE B.Name = @territory;
GO
```

Ghi chú- Khi bạn thay đổi định nghĩa của một thủ tục lưu trữ, những đối tượng phụ thuộc có thể thất bại khi được thực thi. Điều này xảy ra nếu các đối tượng phụ thuộc không được cập nhật để phản ánh những thay đổi được thực hiện cho thủ tục lưu trữ.

→ Hướng dẫn sử dụng câu lệnh ALTER PROCEDURE

Các thủ tục lưu trữ được thay đổi bằng cách sử dụng câu lệnh ALTER PROCEDURE. Các sự kiện sau đây phải được xem xét trong khi sử dụng câu lệnh ALTER PROCEDURE:

- Khi thủ tục lưu trữ được tạo ra bằng cách sử dụng các tùy chọn như là tùy chọn WITH ENCRYPTION, những tùy chọn này cũng nên được đưa vào trong câu lệnh ALTER PROCEDURE.
- Câu lệnh ALTER PROCEDURE thay đổi một thủ tục đơn lẻ. Khi thủ tục lưu trữ gọi các thủ tục lưu trữ khác, những thủ tục lưu trữ lồng nhau không bị ảnh hưởng bằng cách thay đổi thủ tục gọi.
- Những bộ tạo lập thủ tục lưu trữ, các thành viên của vai trò máy chủ sysadmin và các thành viên của vài trò cơ sở dữ liệu cố định db_owner và db_ddladmin có quyền thực thi câu lệnh ALTER PROCEDURE.
- Đề nghị là bạn không nên sửa đổi các thủ tục lưu trữ hệ thống. Nếu bạn cần thay đổi chức năng của một thủ tục lưu trữ hệ thống, khi đó sẽ tạo ra một thủ tục lưu trữ hệ thống do người dùng định nghĩa bằng cách sao chép các câu lệnh từ một thủ tục lưu trữ hiện có và sửa đổi thủ tục do người dùng định nghĩa này.

→ Thả bỏ các thủ tục lưu trữ

Thủ tục lưu trữ có thể được thả bỏ nếu chúng không còn cần thiết. Nếu thủ tục lưu trữ khác gọi một thủ tục đã xóa, thông báo lỗi được hiển thị.

Nếu thủ tục mới được tạo ra sử dụng cùng tên cũng như cùng các tham số như thủ tục đã thả bỏ, tất cả các lần gọi đến thủ tục đã thả bỏ sẽ được thực thi thành công. Điều này là do bây giờ chúng sẽ tham khảo thủ tục mới, trong đó có cùng tên và các tham số như thủ tục đã xóa.

Trước khi thả bỏ một thủ tục lưu trữ, hãy thực thi thủ tục lưu trữ hệ thống sp_depends để xác định đối tượng nào phụ thuộc vào thủ tục này.

Thủ tục được thả bỏ sử dụng câu lệnh DROP PROCEDURE.

Cú pháp sau đây được sử dụng để thả bỏ thủ tục lưu trữ.

Cú pháp:

```
DROP PROCEDURE <procedure_name>
```

Đoạn mã 41 thả bỏ thủ tục lưu trữ uspGetTotals.

Đoạn mã 41:

```
DROP PROCEDURE uspGetTotals
```

10.15 Các thủ tục lưu trữ lồng nhau

SQL Server 2012 cho phép các thủ tục lưu trữ được gọi bên trong các thủ tục lưu trữ khác. Những thủ tục được gọi đến lượt mình có thể gọi các thủ tục khác. Kiến trúc gọi một thủ tục từ một thủ tục khác này được gọi là kiến trúc thủ tục lưu trữ lồng nhau.

Khi thủ tục lưu trữ gọi một thủ tục lưu trữ khác, mức độ lồng nhau được cho là tăng lên một. Tương tự như vậy, khi thủ tục gọi hoàn thành việc thực thi của nó và chuyển điều khiển trở lại cho thủ tục gọi, mức độ lồng nhau được cho là giảm đi một. Mức lồng nhau tối đa được SQL Server 2012 hỗ trợ là 32. Nếu mức lồng nhau vượt quá 32, quá trình gọi thất bại. Ngoài ra, lưu ý rằng nếu thủ tục lưu trữ cố gắng truy cập hơn 64 cơ sở dữ liệu, hoặc nhiều hơn hai cơ sở dữ liệu trong kiến trúc lồng nhau, sẽ có lỗi.

Đoạn mã 42 được sử dụng để tạo ra thủ tục lưu trữ **NestedProcedure** gọi hai thủ tục lưu trữ khác đã được tạo ra trước đó thông qua Đoạn mã 34 và 36.

Đoạn mã 42:

```
CREATE PROCEDURE NestedProcedure
AS
BEGIN
EXEC uspGetCustTerritory
EXEC uspGetSales 'France'
END
```

Khi thủ tục **NestedProcedure** được thực thi, thủ tục này đến lượt mình gọi các thủ tục lưu trữ **uspGetCustTerritory** và **uspGetSales** và truyền giá trị France làm tham số đầu vào cho thủ tục lưu trữ **uspGetSales**.

Ghi chú- Mặc dù có thể có tối đa là 32 mức lồng nhau, không có giới hạn về số lượng thủ tục lưu trữ có thể được gọi từ một thủ tục lưu trữ đã cho.

10.15.1 Hàm @@NESTLEVEL

Mức lồng nhau của thủ tục hiện tại có thể được xác định bằng cách sử dụng hàm **@@NESTLEVEL**. Khi hàm **@@NESTLEVEL** được thực thi trong chuỗi Transact-SQL, giá trị trả về là mức lồng nhau hiện tại + 1.

Nếu bạn sử dụng **sp_executesql** để thực thi hàm **@@NESTLEVEL**, giá trị trả về là mức lồng nhau hiện tại + 2 (như là một thủ tục lưu trữ có tên là **sp_executesql**, được thêm vào chuỗi lồng nhau).

Cú pháp:

```
@@NESTLEVEL
```

trong đó:

`@@NESTLEVEL`: Là hàm trả về một giá trị số nguyên chỉ ra mức lồng nhau.

Đoạn mã 43 tạo ra và thực hiện thủ tục `Nest_Procedure` sẽ thực thi hàm `@@NESTLEVEL` để xác định mức lồng nhau trong ba kịch bản khác nhau.

Đoạn mã 43:

```
CREATE PROCEDURE Nest_Procedure
AS
SELECT @@NESTLEVEL AS NestLevel;
EXECUTE ('SELECT @@NESTLEVEL AS [NestLevel With Execute]');
EXECUTE sp_executesql N'SELECT @@NESTLEVEL AS [NestLevel With sp_executesql]';
```

Đoạn mã 44 thực thi thủ tục lưu trữ `Nest_Procedure`.

Đoạn mã 44:

```
EXECUTE Nest_Procedure
```

Ba kết quả được hiển thị trong hình 10.14 cho ba phương pháp khác nhau được sử dụng để gọi hàm `@@NESTLEVEL`.

NestLevel	
1	1

NestLevel With Execute	
1	2

NestLevel With sp_executesql	
1	3

Hình 10.14: Sử dụng `@@NESTLEVEL`

Ghi chú - Thủ tục lưu trữ `sp_executesql` còn có thể được sử dụng để thực thi các câu lệnh Transact-SQL.

10.16 Truy vấn siêu dữ liệu hệ thống

Những thuộc tính của một đối tượng như bảng hoặc khung nhìn được lưu trữ trong các bảng hệ thống đặc biệt. Những thuộc tính này được gọi là siêu dữ liệu. Tất cả các đối tượng SQL tạo ra siêu dữ liệu. Siêu dữ liệu này có thể được xem bằng cách sử dụng các khung nhìn hệ thống, là các khung nhìn được xác định trước của SQL Server.

Có hơn 230 khung nhìn hệ thống khác nhau và chúng được tự động đưa vào cơ sở dữ liệu do người dùng tạo ra. Những khung nhìn này được nhóm lại thành một số lược đồ khác nhau.

→ Các khung nhìn danh mục hệ thống

Những cái này chứa thông tin về danh mục trong hệ thống SQL Server. Danh mục tương tự như một hàng tồn kho các đối tượng. Những khung nhìn này có chứa một loạt các siêu dữ liệu. Trong các phiên bản trước đây của SQL Server, người dùng phải truy vấn một số lượng lớn các bảng hệ thống, khung nhìn hệ thống, và các hàm hệ thống. Trong SQL Server 2012, tất cả các siêu dữ liệu danh mục người dùng có thể truy cập có thể dễ dàng được tìm thấy bằng cách truy vấn chỉ các khung nhìn danh mục.

Đoạn mã 45 lấy một danh sách các bảng và các thuộc tính người dùng từ khung nhìn danh mục hệ thống sys.tables.

Đoạn mã 45:

```
SELECT name, object_id, type, type_desc
FROM sys.tables;
```

→ Các khung nhìn lược đồ thông tin

Người dùng có thể truy vấn các khung nhìn lược đồ thông tin để trả về siêu dữ liệu hệ thống. Những khung nhìn này sẽ hữu ích cho các công cụ của bên thứ ba có thể không phải là cụ thể cho SQL Server. Các khung nhìn lược đồ thông tin cung cấp một khung nhìn nội bộ, hệ thống độc lập với bảng của siêu dữ liệu SQL Server. Các khung nhìn lược đồ thông tin cho phép các ứng dụng làm việc một cách chính xác mặc dù các thay đổi đáng kể đã được thực hiện cho những bảng hệ thống nằm dưới.

Những điểm sau đây trong bảng 10.3 sẽ giúp quyết định xem nên truy vấn các khung nhìn hệ thống cụ thể cho SQL Server hay các khung nhìn lược đồ thông tin:

Các khung nhìn lược đồ thông tin	Các khung nhìn hệ thống SQL Server
Chúng được lưu trữ trong lược đồ riêng INFORMATION_SCHEMA.	Chúng xuất hiện trong lược đồ sys.
Chúng sử dụng thuật ngữ tiêu chuẩn thay vì thuật ngữ SQL Server. Ví dụ, chúng sử dụng danh mục thay vì cơ sở dữ liệu và tên miền thay vì kiểu dữ liệu do người dùng định nghĩa.	Họ tuân thủ bảng thuật ngữ SQL Server.
Chúng có thể không tiếp xúc với tất cả các siêu dữ liệu có sẵn cho các khung nhìn danh mục của chính SQL Server. Ví dụ, sys.columns bao gồm các tính chất cho thuộc tính nhận dạng và thuộc tính cột tính, trong khi information_schema.columns lại không.	Chúng có thể tiếp xúc với tất cả các siêu dữ liệu có sẵn cho các khung nhìn danh mục của SQL Server.

Bảng 10.3: Các khung nhìn lược đồ thông tin và các khung nhìn hệ thống cụ thể cho SQL Server

Đoạn mã 46 lấy dữ liệu từ khung nhìn INFORMATION_SCHEMA.TABLES trong cơ sở dữ liệu AdventureWorks2012:

Đoạn mã 46:

```
SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES;
```

→ **Các hàm siêu dữ liệu hệ thống**

Ngoài các khung nhìn, SQL Server cung cấp một số các hàm dựng sẵn trả về siêu dữ liệu cho truy vấn. Chúng bao gồm các hàm vô hướng và hàm có giá trị bảng, có thể trả về thông tin về các thiết lập hệ thống, tùy chọn phiên, và một loạt các đối tượng.

Các hàm siêu dữ liệu SQL Server đến ở một loạt các định dạng. Một số xuất hiện tương tự như các hàm vô hướng tiêu chuẩn, như là ERROR_NUMBER(). Những cái khác sử dụng tiền tố đặc biệt, chẳng hạn như @@VERSION hoặc \$PARTITION. Bảng 10.4 trình bày một số hàm siêu dữ liệu hệ thống thông thường.

Tên hàm	Mô tả	Ví dụ
OBJECT_ID(<object_name>)	Trả về mã đối tượng của một đối tượng cơ sở dữ liệu.	OBJECT_ID('Sales.Customer')
OBJECT_NAME(<object_id>)	Trả về tên tương ứng với một mã đối tượng.	OBJECT_NAME(197575742)
@@ERROR	Trả về 0 nếu câu lệnh cuối cùng đã thành công, nếu không trả về số lỗi.	@@ERROR
SERVERPROPERTY(<property >)	Trả về giá trị thuộc tính máy chủ đã chỉ định.	SERVERPROPERTY('Collation')

Bảng 10.4: Các hàm siêu dữ liệu hệ thống thông thường

Đoạn mã 47 sử dụng câu lệnh SELECT để truy vấn hàm siêu dữ liệu hệ thống.

Đoạn mã 47:

```
SELECT SERVERPROPERTY ('EDITION') AS EditionName;
```

10.17 Truy vấn các đối tượng quản lý động

Đầu tiên được giới thiệu trong SQL Server 2005, Khung nhìn quản lý động (DMV) và Hàm quản lý động (DMF) là các đối tượng quản lý động trả về thông tin trạng thái máy chủ và cơ sở dữ liệu. DMV và DMF được gọi chung là đối tượng quản lý động. Chúng cung cấp cái nhìn sâu sắc hữu ích vào hoạt động của phần mềm và có thể được sử dụng để kiểm tra trạng thái của thể hiện SQL Server, xử lý sự cố, và điều chỉnh hiệu năng.

Cả DMV và DMF trả về dữ liệu dưới dạng bảng nhưng sự khác biệt là trong khi DMF bình thường nhận ít nhất một tham số, DMV không nhận các tham số. SQL Server 2012 cung cấp gần 200 đối tượng quản lý động. Để truy vấn các DMV, cần thiết phải có quyền VIEW SERVER STATE hoặc VIEW DATABASE STATE, tùy thuộc vào phạm vi của DMV.

10.17.1 Phân loại và truy vấn các DMV

Bảng 10.5 liệt kê quy ước đặt tên giúp tổ chức DMV theo hàm.

Mẫu đặt tên	Mô tả
db	cơ sở dữ liệu có liên quan
io	số liệu thống kê nhập/xuất
Os	Thông tin hệ điều hành SQL Server
"tran"	giao tác có liên quan
"exec"	truy vấn siêu dữ liệu liên quan đến thực thi

Bảng 10.5: Tổ chức DMVs theo hàm

Để truy vấn một đối tượng quản lý động, bạn sử dụng câu lệnh SELECT như khi bạn làm với bất kỳ khung nhìn do người dùng định nghĩa hoặc hàm có giá trị bảng. Ví dụ, Đoạn mã 48 trả về một danh sách các kết nối người dùng hiện tại từ khung nhìn sys.dm_exec_sessions.

sys.dm_exec_sessions là một DMV thuộc phạm vi máy chủ hiển thị thông tin về tất cả các kết nối người dùng hoạt động và các tác vụ nội bộ. Thông tin này bao gồm người dùng đăng nhập, thiết lập phiên hiện tại, phiên bản máy khách, tên chương trình máy khách, thời gian máy khách đăng nhập, và vân vân. sys.dm_exec_sessions có thể được sử dụng để xác định một phiên cụ thể và tìm kiếm thông tin về nó.

Đoạn mã 48:

```
SELECT session_id, login_time, program_name
FROM sys.dm_exec_sessions
WHERE login_name = 'sa' and is_user_process=1;
```

Ở đây, is_user_process là một cột trong khung nhìn xác định xem phiên có phải là một phiên hệ thống hay không. Giá trị 1 chỉ ra rằng nó không phải là một phiên hệ thống mà là một phiên người dùng. Cột program_name xác định tên của chương trình máy khách đã khởi tạo phiên này. Cột login_time thiết lập thời gian khi phiên bắt đầu. Đầu ra của Đoạn mã 48 được trình bày trong hình 10.15.

	session_id	login_time	program_name
1	51	2013-01-29 12:26:08.443	Microsoft SQL Server Management Studio
2	53	2013-01-29 12:26:20.247	Microsoft SQL Server Management Studio - Query

Hình 10.15: Truy vấn DMV sys.dm_exec_sessions

10.18 Kiểm tra tiến bộ của bạn

1. Câu nào sau đây về khung nhìn là đúng?

a.	Khung nhìn cho phép bạn xem và thao tác các phần đã chọn của bảng.		
b.	Chỉ có thể chọn các cột từ một bảng cho một khung nhìn, các hàng thì không thể.		
c.	Các khung nhìn hệ thống hiển thị thông tin về hệ thống hoặc máy.		
d.	Các khung nhìn có tối đa 1024 cột.		
e.	Khi dữ liệu trong khung nhìn được thay đổi, nó không được phản ánh trong bảng nằm dưới.		

(A)	a, c, d, e	(C)	a, b, d
(B)	a, c	(D)	Tất cả các câu trên trên

2. Bạn đang tạo ra khung nhìn **Supplier _ View** với các cột **FirstName**, **LastName**, và **City** từ bảng **Supplier _ Details**. Đoạn mã nào sau đây vi phạm định nghĩa về khung nhìn?

(A)	CREATE VIEW Supplier_View AS SELECT FirstName, LastName, City FROM Supplier_Details WHERE City IN('New York', 'Boston', 'Orlando')	(C)	CREATE VIEW Supplier_View AS SELECT FirstName, LastName, City FROM Supplier_Details ORDER BY FirstName
(B)	CREATE VIEW Supplier_View AS SELECT TOP 100 FirstName, LastName, City FROM Supplier_Details WHERE FirstName LIKE 'A%' ORDER BY FirstName	(D)	CREATE VIEW Supplier_View AS SELECT TOP 100 FirstName, LastName, City FROM Supplier_Details

3. Câu nào sau đây về tùy chọn CHECK OPTION và SCHEMABINDING là đúng?

a.	CHECK OPTION đảm bảo tính toàn vẹn thực thể.
b.	Tùy chọn SCHEMABINDING liên kết khung nhìn với lược đồ của bảng cơ sở.
c.	Khi hàng được sửa đổi, WITH CHECK OPTION đảm bảo rằng dữ liệu vẫn có thể nhìn thấy thông qua khung nhìn.
d.	Tùy chọn SCHEMABINDING đảm bảo bảng cơ sở không thể được sửa đổi theo cách sẽ ảnh hưởng đến định nghĩa khung nhìn.
e.	Tùy chọn SCHEMABINDING không thể được sử dụng với các câu lệnh ALTER VIEW.

(A)	a, b, c	(C)	b, c, d
(B)	b, c	(D)	c, d, e

4. Bạn muốn tạo ra khung nhìn **Account _ Details** với tùy chọn SCHEMABINDING. Đoạn mã nào sau đây sẽ đạt được mục tiêu này?

(A)	CREATE VIEW Account_Details AS SELECT AccNo, City FROM dbo.Customer_Details WITH SCHEMABINDING	(C)	CREATE VIEW Account_Details WITH SCHEMABINDING AS SELECT AccNo, City FROM dbo.Customer_Details
(B)	CREATE VIEW Account_Details SCHEMABINDING AS SELECT AccNo, City FROM Customer_Details	(D)	CREATE VIEW Account_Details WITH SCHEMABINDING AS SELECT AccNo, City FROM Customer_Details

5. Bảng **Item _ Details** được tạo ra với các cột **ItemCode**, **ItemName**, **Price**, và **Quantity**. Cột **ItemCode** được định nghĩa là PRIMARY KEY, **ItemName** được định nghĩa với các ràng buộc UNIQUE và NOT NULL, **Price** được định nghĩa với ràng buộc NOT NULL, và **Quantity** được định nghĩa với ràng buộc NOT NULL và có giá trị mặc định đã chỉ định. Khung nhìn nào sau đây được tạo ra bằng cách sử dụng các cột từ bảng **Item _ Details** có thể được sử dụng để chèn các bản ghi vào bảng này?

(A)	<pre>CREATE VIEW ItemDetails AS SELECT ItemCode, ItemName, Price FROM Item_Details</pre>	(C)	<pre>CREATE VIEW ItemDetails AS SELECT ItemName, Price, Quantity FROM Item_Details</pre>
(B)	<pre>CREATE VIEW ItemDetails AS SELECT ItemCode, Price, Quantity FROM Item_Details</pre>	(D)	<pre>CREATE VIEW ItemDetails AS SELECT ItemCode, ItemName, Quantity FROM Item_Details</pre>

6. Câu nào sau đây về các thủ tục lưu trữ là đúng?

a.	Thủ tục lưu trữ là một nhóm các câu lệnh Transact-SQL hành động như một khối mã được sử dụng để thực hiện một tác vụ đặc thù.
b.	Tất cả các thủ tục lưu trữ hệ thống được xác định bằng tiền tố 'xp_ '.
c.	Thủ tục lưu trữ phân phối được sử dụng trong việc quản lý các truy vấn phân phối.
d.	Các thủ tục Database Mail và SQL Mail được sử dụng để thực hiện các hoạt động e-mail trong SQL Server.
e.	Các thủ tục lưu trữ do người dùng định nghĩa còn được gọi là các thủ tục lưu trữ tùy chỉnh.

(A)	a, d	(C)	a, c, d, e
(B)	b, c, e	(D)	d

10.18.1 Câu trả lời

1.	B
2.	C
3.	C
4.	B
5.	A
6.	C

Tóm tắt

- Khung nhìn là một bảng ảo được tạo thành từ các cột đã chọn từ một hoặc nhiều bảng và được tạo ra bằng cách sử dụng lệnh CREATE VIEW trong SQL Server.
- Người dùng có thể thao tác dữ liệu trong các khung nhìn, như là chèn vào các khung nhìn, sửa đổi dữ liệu trong các khung nhìn, và xóa khỏi các khung nhìn.
- Thủ tục lưu trữ là một nhóm các câu lệnh Transact-SQL hành động như một khối mã duy nhất thực hiện một nhiệm vụ cụ thể.
- SQL Server hỗ trợ nhiều loại thủ tục lưu trữ khác nhau, chẳng hạn như Thủ tục lưu trữ do người dùng định nghĩa, Thủ tục lưu trữ mở rộng, và Thủ tục lưu trữ hệ thống.
- Thủ tục lưu trữ hệ thống có thể được phân loại thành các loại khác nhau như là Thủ tục lưu trữ danh mục, Thủ tục lưu trữ bảo mật, và Thủ tục lưu trữ con trỏ.
- Các tham số đầu vào và đầu ra có thể được sử dụng với các thủ tục lưu trữ để truyền và nhận dữ liệu từ các thủ tục lưu trữ.
- Những thuộc tính của đối tượng như là bảng hoặc khung nhìn được lưu trữ trong các bảng hệ thống đặc biệt và được gọi là siêu dữ liệu.
- DMV và DMF là đối tượng quản lý động trả về thông tin trạng thái máy chủ và cơ sở dữ liệu. DMV và DMF được gọi chung là đối tượng quản lý động.

Hãy thử tự làm

- Trong SQL Server 2012 Management Studio, xác định vị trí các thủ tục lưu trữ mở rộng được định nghĩa dưới cơ sở dữ liệu chính và thực thi những thủ tục sau đây trong cửa sổ truy vấn:


```
sys.xp_readerrorlog
sys.xp_getnetname
sys.xp_fixeddrives
```
- ShoezUnlimited là một cửa hàng giày hợp thời trang có trụ sở tại Miami. Họ lưu kho các loại giày dép khác nhau ở cửa hàng của mình và bán chúng để có lợi nhuận. ShoezUnlimited duy trì các chi tiết của tất cả các sản phẩm trong một cơ sở dữ liệu SQL Server 2012. Ban quản lý muốn nhà phát triển của họ sử dụng các thủ tục lưu trữ cho các nhiệm vụ thường được thực hiện. Giả sử bạn là nhà phát triển đó, hãy thực hiện các nhiệm vụ sau đây:

Tạo bảng Shoes có cấu trúc như được trình bày trong bảng 10.6 trong cơ sở dữ liệu ShoezUnlimited.

Tên trường	Loại dữ liệu	Trường khóa	Mô tả
ProductCode	varchar(5)	Primary Key	Mã sản phẩm xác định duy nhất mỗi chiếc giày
BrandName	varchar(30)		Tên thương hiệu của giày
Category	varchar(30)		Loại giày, ví dụ như là giày thể thao, mặc giản dị, mặc hội hè, và vân vân
UnitPrice	money		Giá giày tính bằng đô la
QtyOnHand	int		Số lượng có sẵn

Bảng 10.6: Bảng Shoes

- Thêm ít nhất 5 bản ghi cho bảng. Đảm bảo rằng giá trị của cột QtyOnHand nhiều hơn 20 cho mỗi đôi giày.
- Viết các câu lệnh để tạo ra thủ tục lưu trữ có tên là **PriceIncrease** sẽ tăng unitprice của tất cả các đôi giày lên đến 10 đô la.
- Viết các câu lệnh để tạo ra thủ tục lưu trữ **QtyOnHand** sẽ giảm số lượng có sẵn của các thương hiệu đã chỉ định xuống còn 25. Tên thương hiệu cần được cung cấp làm đầu vào.
- Thực thi các thủ tục lưu trữ **PriceIncrease** và **QtyOnHand**.

Phần - 11

Chỉ mục

Chào mừng bạn đến với phần **Chỉ mục**.

Phần này giải thích chỉ mục và hiệu suất của chúng. Nó còn giải thích các loại chỉ mục khác nhau. Cuối cùng, phần này xác định và mô tả thủ tục để truy vấn dữ liệu hiệu suất sử dụng các chỉ mục.

Trong phần này, bạn sẽ học để:

- ➔ Định nghĩa và giải thích các chỉ mục
- ➔ Mô tả hiệu suất của các chỉ mục
- ➔ Giải thích các chỉ mục bó cụm
- ➔ Giải thích các chỉ mục không bó cụm
- ➔ Giải thích phân vùng dữ liệu
- ➔ Giải thích các bước để hiển thị dữ liệu hiệu suất truy vấn sử dụng các chỉ mục

11.1 Giới thiệu

SQL Server 2012 sử dụng các chỉ mục để tìm dữ liệu khi truy vấn được xử lý. Công cụ SQL Server sử dụng chỉ mục theo cách tương tự như học viên sử dụng chỉ mục cuốn sách. Ví dụ, xem xét rằng bạn cần phải tìm tất cả các tham khảo đến các câu lệnh `INSERT` trong một cuốn sách SQL. Cách tiếp cận ngay lập tức đã sử dụng sẽ quét từng trang của cuốn sách bắt đầu từ trang bắt đầu. Bạn đánh dấu mỗi lần từ `INSERT` được tìm thấy, cho đến tới cuối cuốn sách. Cách tiếp cận này rất tốn thời gian và công sức. Cách thứ hai là sử dụng chỉ mục ở phía sau của cuốn sách để tìm số trang cho mỗi lần xuất hiện của các câu lệnh `INSERT`. Cách thứ hai tạo ra các kết quả tương tự như cách đầu tiên, nhưng rất tiết kiệm thời gian.

Khi SQL Server đã không xác định bất kỳ chỉ mục để tìm kiếm, khi đó quá trình này tương tự như cách đầu tiên trong ví dụ, công cụ SQL cần phải ghé thăm mỗi hàng trong bảng. Trong thuật ngữ cơ sở dữ liệu, hành vi này được gọi là quét bảng, hoặc chỉ là quét.

Quét bảng không phải luôn phiền hà, nhưng đôi khi không thể tránh khỏi. Tuy nhiên, khi bảng lớn lên đến hàng ngàn và hàng triệu hàng và hơn thế nữa, quét trở nên chậm hơn và tốn kém hơn. Trong các trường hợp này, đặc biệt nên sử dụng chỉ mục.

Việc tạo hoặc loại bỏ các chỉ mục khỏi lược đồ cơ sở dữ liệu sẽ không ảnh hưởng đến mã của một ứng dụng. Chỉ mục hoạt động ở phía sau với sự hỗ trợ của công cụ cơ sở dữ liệu. Hơn nữa, việc tạo ra một chỉ mục thích hợp có thể làm tăng đáng kể hiệu suất của ứng dụng.

11.1.1 Tổng quan về lưu trữ dữ liệu

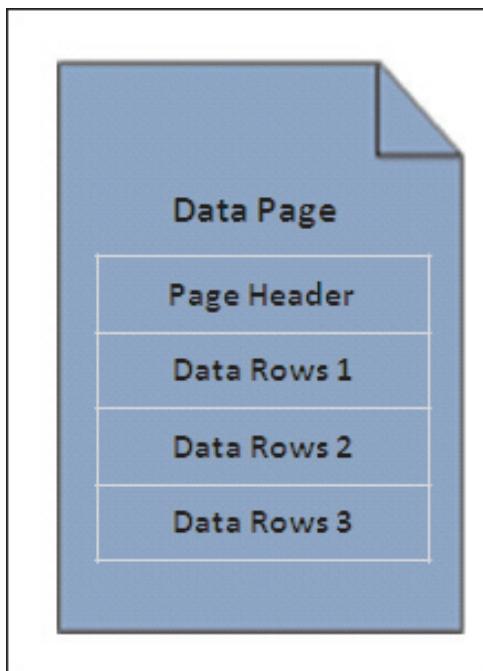
Một cuốn sách có các trang, trong đó có chứa các đoạn văn được tạo ra từ các câu. Tương tự như vậy, SQL Server 2012 lưu trữ dữ liệu trong các đơn vị lưu trữ được gọi là các trang dữ liệu. Những trang này chứa dữ liệu dưới dạng các hàng.

Trong SQL Server 2012, dung lượng của mỗi trang dữ liệu là 8 Kilo Byte (KB). Như vậy, cơ sở dữ liệu SQL Server có 128 trang dữ liệu cho mỗi Mega Byte (MB) không gian lưu trữ.

Trang bắt đầu với phần đầu đầ 96 byte, lưu trữ thông tin hệ thống về trang đó. Những thông tin này bao gồm:

- ➔ Trang số
- ➔ Loại trang
- ➔ Số lượng không gian còn trống trên trang
- ➔ ID đơn vị phân bổ của đối tượng để phân bổ cho trang đó

Hình 11.1 trình bày cấu trúc lưu trữ dữ liệu của một trang dữ liệu.



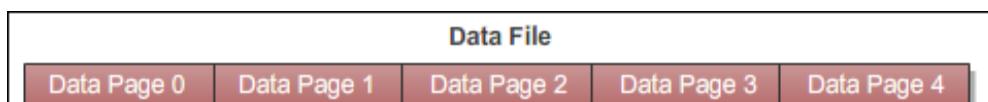
Hình 11.1: Lưu trữ dữ liệu

Ghi chú - Một trang dữ liệu là đơn vị lưu trữ dữ liệu nhỏ nhất. Đơn vị phân bổ là một tập hợp các trang dữ liệu được nhóm lại với nhau dựa trên loại trang. Việc nhóm lại này được thực hiện để quản lý hiệu quả dữ liệu.

11.1.2 Các tập tin dữ liệu

Tất cả các phép tính đầu vào và đầu ra trong cơ sở dữ liệu được thực hiện ở mức trang. Điều này có nghĩa rằng công cụ cơ sở dữ liệu đọc hoặc viết các trang dữ liệu. Một bộ tám trang dữ liệu tiếp giáp được gọi là một mức độ.

SQL Server 2012 lưu trữ dữ liệu trong các tập tin được gọi là các tập tin dữ liệu. Không gian được phân bổ cho một tập tin dữ liệu được chia thành các trang dữ liệu được đánh số thứ tự. Đánh số bắt đầu từ số 0 như được trình bày trong hình 11.2.



Hình 11.2: Các tập tin dữ liệu

Có ba loại tập tin dữ liệu trong SQL Server 2012. Chúng bao gồm:

→ **Tập tin dữ liệu chính**

Tập tin dữ liệu chính được tạo ra tự động vào thời điểm tạo ra cơ sở dữ liệu. Tập tin này có các tham khảo tới tất cả các tập tin khác trong cơ sở dữ liệu. Phần mở rộng tập tin được đề nghị cho các tập tin dữ liệu chính là .mdf.

→ **Tập tin dữ liệu phụ**

Tập tin dữ liệu phụ là tùy chọn trong một cơ sở dữ liệu và có thể được tạo ra để phân biệt các đối tượng cơ sở dữ liệu như là các bảng, khung nhìn, và thủ tục. Phần mở rộng tập tin được đề nghị cho các tập tin dữ liệu phụ là .ndf.

→ **Tập tin nhật ký**

Tập tin nhật ký chứa thông tin về các thay đổi được thực hiện trong cơ sở dữ liệu. Thông tin này rất hữu ích trong phục hồi dữ liệu trong các sự cố bất ngờ như mất điện đột ngột hoặc cần phải chuyển cơ sở dữ liệu đến một máy chủ khác. Có ít nhất một tập tin nhật ký cho mỗi cơ sở dữ liệu. Phần mở rộng tập tin được đề nghị cho các tập tin nhật ký là .ldf.

11.1.3 Yêu cầu đối với các chỉ mục

Để tạo điều kiện lấy dữ liệu nhanh chóng từ một cơ sở dữ liệu, SQL Server 2012 cung cấp tính năng lập chỉ mục. Tương tự như chỉ mục trong một cuốn sách, chỉ mục trong cơ sở dữ liệu SQL Server 2012 chứa thông tin cho phép bạn tìm kiếm dữ liệu cụ thể mà không quét qua toàn bộ bảng như được trình bày trong hình 11.3.

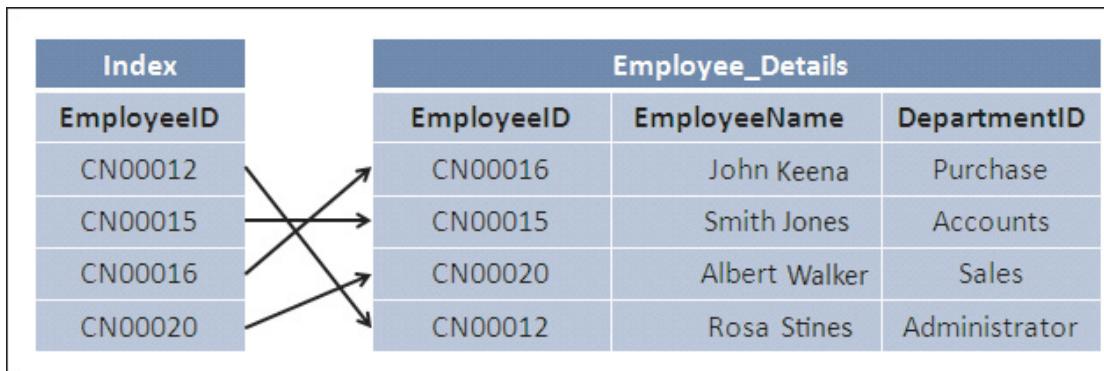
Index			
A		C	
Adapter	1	Border	19
Aggregate	10	Bullet	58
Analysis	13		
Average	23	Consistency	20
B		Connect	22
Board	17	Communication	24
Brilliant	18	Character	30

Hình 11.3: Yêu cầu đối với các chỉ mục

11.1.4 Chỉ mục

Trong bảng, các bản ghi được lưu trữ theo thứ tự chúng được nhập vào. Lưu trữ của chúng trong cơ sở dữ liệu không được phân loại. Khi dữ liệu được lấy từ các bảng như vậy, toàn bộ bảng cần phải được quét. Điều này làm chậm quá trình truy xuất truy vấn. Để tăng tốc truy xuất truy vấn, cần phải tạo ra các chỉ mục.

Khi chỉ mục được tạo ra trên một bảng, chỉ mục tạo ra một trật tự cho các hàng dữ liệu hoặc bản ghi trong bảng như được trình bày trong hình 11.4. Điều này hỗ trợ ở vị trí nhanh hơn và truy xuất dữ liệu trong quá trình tìm kiếm.



Hình 11.4: Chỉ mục

Ghi chú - Nhiều chỉ mục có thể được tạo ra trên một bảng.

Chỉ mục sẽ được tự động tạo ra khi các ràng buộc PRIMARY KEY và UNIQUE được định nghĩa trên bảng. Chỉ mục giảm các hoạt động nhập/xuất của đĩa và tiêu thụ các tài nguyên hệ thống ít hơn.

Câu lệnh CREATE INDEX được sử dụng để tạo ra một chỉ mục. Sau đây là cú pháp cho câu lệnh này.

Cú pháp:

```
CREATE INDEX <index_name> ON <table_name> (<column_name>)
```

trong đó:

index_name: chỉ ra tên của chỉ mục.

table_name: chỉ ra tên của bảng.

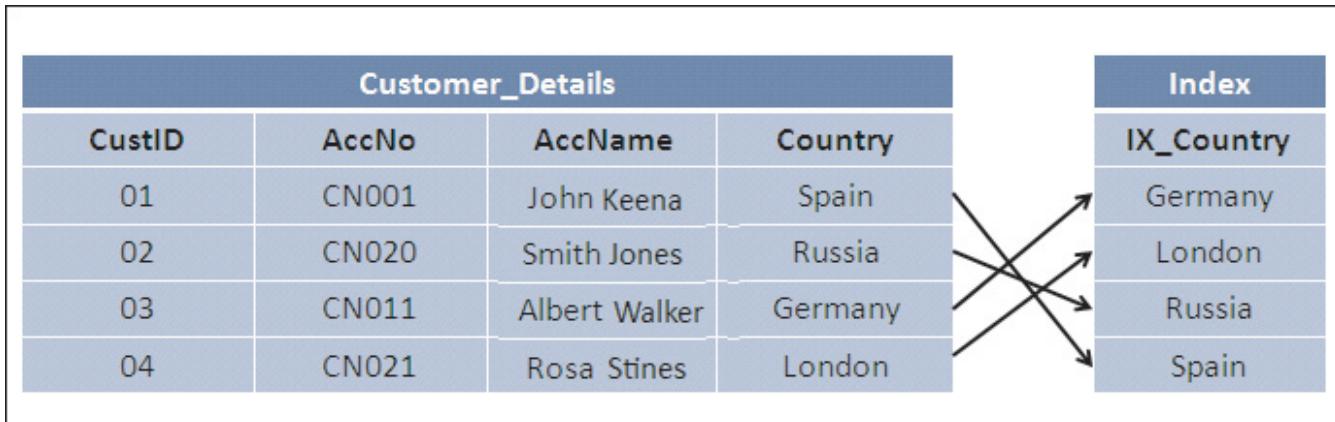
column_name: chỉ ra tên của cột.

Đoạn mã 1 tạo ra chỉ mục **IX_Country** trên cột **Country** trong bảng **Customer_Details**.

Đoạn mã 1:

```
USE CUST_DB
CREATE INDEX IX_Country ON Customer_Details(Country);
GO
```

Hình 11.5 trình bày bảng có chỉ mục của **Customer_Details**.



The diagram illustrates a relationship between a table and an index. On the left is a table named "Customer_Details" with four columns: CustID, AccNo, AccName, and Country. The rows contain data for customers 01 through 04. On the right is an index named "IX_Country" with four entries: Germany, London, Russia, and Spain. Arrows point from the "Country" column of the table to the corresponding entries in the index. Specifically, row 01 points to "Germany", row 02 to "Russia", row 03 to "Germany", and row 04 to "London".

Customer_Details				Index
CustID	AccNo	AccName	Country	IX_Country
01	CN001	John Keena	Spain	Germany
02	CN020	Smith Jones	Russia	London
03	CN011	Albert Walker	Germany	Russia
04	CN021	Rosa Stines	London	Spain

Hình 11.5: Bảng có chỉ mục của Customer_Details

Chỉ mục trả đến vị trí của một hàng trên một trang dữ liệu thay vì tìm kiếm suốt cả bảng.

Xem xét các sự kiện và hướng dẫn sau đây về chỉ mục:

- Chỉ mục làm tăng tốc độ các truy vấn nối các bảng hoặc thực hiện các hoạt động phân loại.
- Chỉ mục thực hiện sự duy nhất của các hàng nếu được định nghĩa khi bạn tạo chỉ mục.
- Chỉ mục được tạo ra và duy trì theo thứ tự tăng dần hoặc giảm dần.

11.1.5 Kịch bản

Trong danh bạ điện thoại, nơi một số lượng lớn dữ liệu được lưu trữ và thường xuyên được truy cập, lưu trữ dữ liệu được thực hiện theo thứ tự bảng chữ cái. Nếu dữ liệu đó đã không được phân loại, nó sẽ gần như không thể tìm kiếm một số điện thoại cụ thể.

Tương tự như vậy, trong một bảng cơ sở dữ liệu có một số lượng lớn các bản ghi được truy cập thường xuyên, dữ liệu sẽ được sắp xếp để truy xuất nhanh chóng. Khi chỉ mục được tạo ra trên bảng, chỉ mục phân loại một cách thực tế hoặc logic các bản ghi. Vì vậy, tìm kiếm một bản ghi cụ thể trở nên nhanh hơn và có ít căng thẳng lên tài nguyên hệ thống.

11.1.6 Truy cập theo nhóm dữ liệu

Chỉ mục rất hữu ích khi dữ liệu cần phải được truy cập theo nhóm. Ví dụ, bạn muốn thực hiện các sửa đổi cho trợ cấp đi lại cho tất cả các nhân viên dựa trên bộ phận nơi họ làm việc. Ở đây, bạn muốn thực hiện những thay đổi cho tất cả các nhân viên trong một bộ phận trước khi chuyển sang các nhân viên trong bộ phận khác. Trong trường hợp này, chỉ mục có thể được tạo ra như được trình bày trong hình 11.6 trên cột **Department** trước khi truy cập các bản ghi.

Chỉ mục này sẽ tạo ra các khối logic của các hàng dữ liệu dựa trên bộ phận. Điều này một lần nữa sẽ hạn chế số lượng dữ liệu thực sự đã quét trong thời gian truy xuất truy vấn.

Do đó, truy xuất sẽ nhanh hơn và sẽ có ít căng thẳng hơn lên tài nguyên hệ thống.

Department Name	Employee Name
Marketing	Jenny Woods
Marketing	Merry Thomas
Marketing	John Updeeke
Marketing	Robert Williamson
Sales	Smith Gordon
Sales	Albert Wang

Hình 11.6: Truy cập theo nhóm dữ liệu

11.2 Kiến trúc chỉ mục

Trong SQL Server 2012, có thể lưu trữ dữ liệu trong cơ sở dữ liệu theo cách có sắp xếp hoặc ngẫu nhiên. Nếu dữ liệu được lưu trữ theo một cách có sắp xếp, dữ liệu được cho là có mặt trong một cấu trúc ghép cụm. Nếu được lưu trữ một cách ngẫu nhiên, nó được cho là có mặt trong một cấu trúc heap.

Hình 11.7 trình bày ví dụ minh họa kiến trúc chỉ mục.

Employee_Details		
EmpID	EmpName	DeptID
CN00020	Rosa Stevens	BN0001
CN00018	John Updeeke	BN0020
CN00019	Smith Gordon	BN0021
CN00012	Robert Tyson	BN0011

Heap Structure

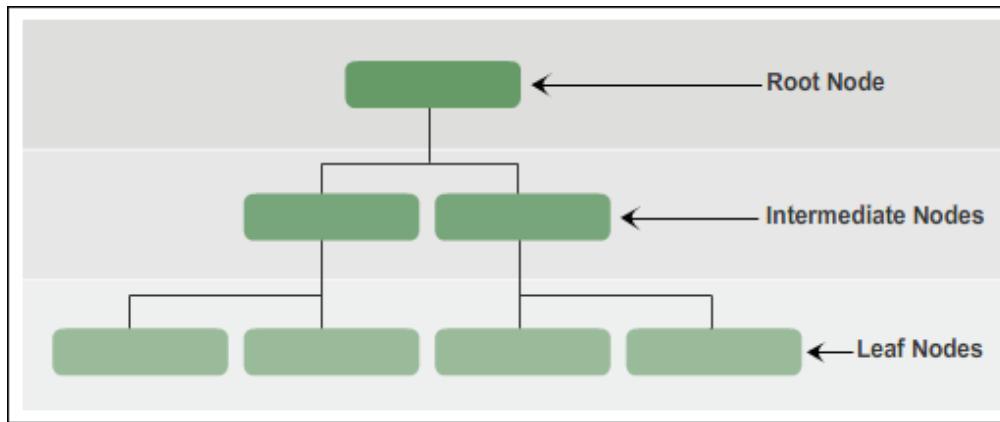
Employee_Details		
EmpID	EmpName	DeptID
CN00012	Robert Tyson	BN0011
CN00018	John Updeeke	BN0020
CN00019	Smith Gordon	BN0021
CN00020	Rosa Stevens	BN0001

Clustered Structure

Hình 11.7: Kiến trúc chỉ mục

11.2.1 B-Tree

Trong SQL Server, tất cả các chỉ mục được cấu trúc ở dạng B-Tree. Cấu trúc B-Tree có thể được hình dung như một cây ngược với gốc ngay ở trên cùng, chia thành các nhánh và sau đó, vào lá ngay ở dưới cùng như trong hình 11.8.



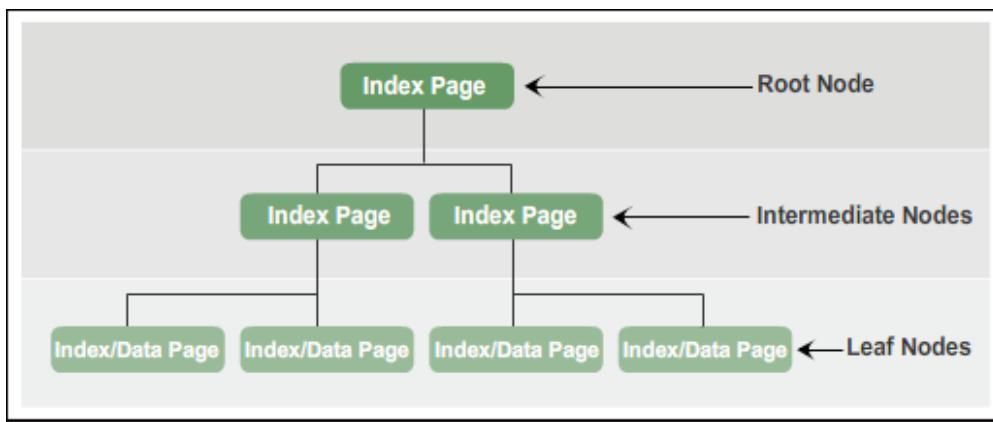
Hình 11.8: B-Tree

Trong cấu trúc B-Tree, có một nút gốc duy nhất ở trên cùng. Sau đó nút này phân nhánh vào mức tiếp theo, được gọi là mức ở giữa đầu tiên. Những nút ở mức ở giữa đầu tiên có thể phân nhánh ra thêm. Việc phân nhánh này có thể tiếp tục thành nhiều mức ở giữa và sau đó, cuối cùng là mức lá. Những nút ở mức lá được gọi là các nút lá.

Ghi chú - Chỉ mục B-Tree đi qua từ trên xuống dưới bằng cách sử dụng các con trỏ.

11.2.2 Cấu trúc B-Tree của chỉ mục

Trong cấu trúc B-Tree của một chỉ mục, nút gốc bao gồm một trang chỉ mục. Trang chỉ mục này có chứa các con trỏ trả về các trang chỉ mục có trong mức ở giữa đầu tiên. Những trang chỉ mục này đến lượt mình trả về các trang chỉ mục có trong mức ở giữa tiếp theo. Có thể có nhiều mức ở giữa trong một B-Tree của chỉ mục. Các nút lá của B-Tree của chỉ mục có các trang dữ liệu chứa các hàng dữ liệu hoặc các trang chỉ mục có chứa các hàng chỉ mục trả đến các hàng dữ liệu như được trình bày trong hình 11.9.



Hình 11.9: Cấu trúc B-Tree của chỉ mục

Các loại nút khác nhau như sau:

- **Nút gốc** - Chứa một trang chỉ mục với các con trỏ trỏ đến các trang chỉ mục ở mức ở giữa đầu tiên.
- **Các nút trung gian** - Chứa các trang chỉ mục với các con trỏ trỏ các trang chỉ mục ở mức trung gian tiếp theo hoặc tới các trang chỉ mục hoặc dữ liệu ở mức lá.
- **Các nút lá** - Chứa các trang dữ liệu hoặc các trang chỉ mục trỏ đến các trang dữ liệu.

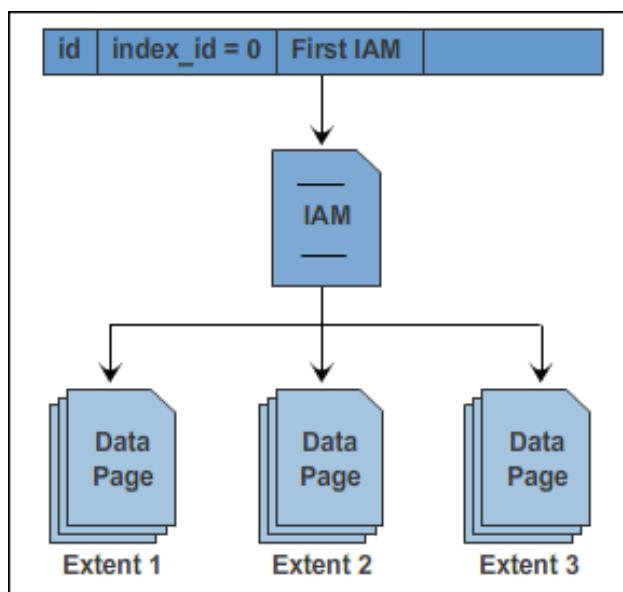
Ghi chú - Trang dữ liệu có chứa các mục nhập chỉ mục được gọi là trang chỉ mục.

11.2.3 Các cấu trúc heap

Trong cấu trúc heap, các trang dữ liệu và bản ghi không được sắp xếp theo thứ tự được sắp xếp. Kết nối duy nhất giữa các trang dữ liệu là thông tin được ghi trong các trang Bản đồ phân bổ chỉ mục (IAM).

Trong SQL Server 2012, các trang IAM được sử dụng để quét qua một cấu trúc heap. Các trang IAM ánh xạ các mức độ được sử dụng bởi một đơn vị phân bổ trong một phần của tập tin cơ sở dữ liệu.

Heap có thể được đọc bằng cách quét các trang IAM để tìm những mức độ có chứa những trang cho heap đó như được trình bày trong hình 11.10.



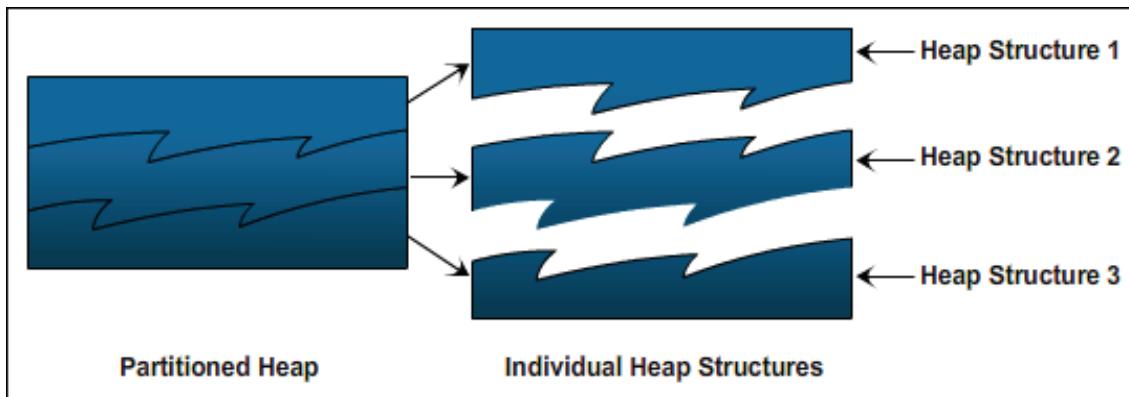
Hình 11.10: Cấu trúc heap

Ghi chú - Nếu một đơn vị phân bổ có chứa các mức độ từ nhiều hơn một tập tin, sẽ có nhiều trang IAM được liên kết với nhau trong một chuỗi IAM để ánh xạ những mức độ này.

11.2.4 Phân vùng các cấu trúc heap

Bảng có thể được chia một cách logic thành các nhóm hàng nhỏ hơn. Sự phân chia này được gọi là phân vùng. Bảng được phân vùng để thực hiện các hoạt động bảo trì hiệu quả hơn. Theo mặc định, bảng có một phân vùng duy nhất.

Khi các phân vùng được tạo ra trong bảng có cấu trúc heap, mỗi phân vùng sẽ chứa dữ liệu trong cấu trúc heap riêng lẻ. Ví dụ, nếu heap có ba phân vùng, khi đó có ba cấu trúc heap hiện diện, một trong mỗi phân vùng như được trình bày trong hình 11.11.

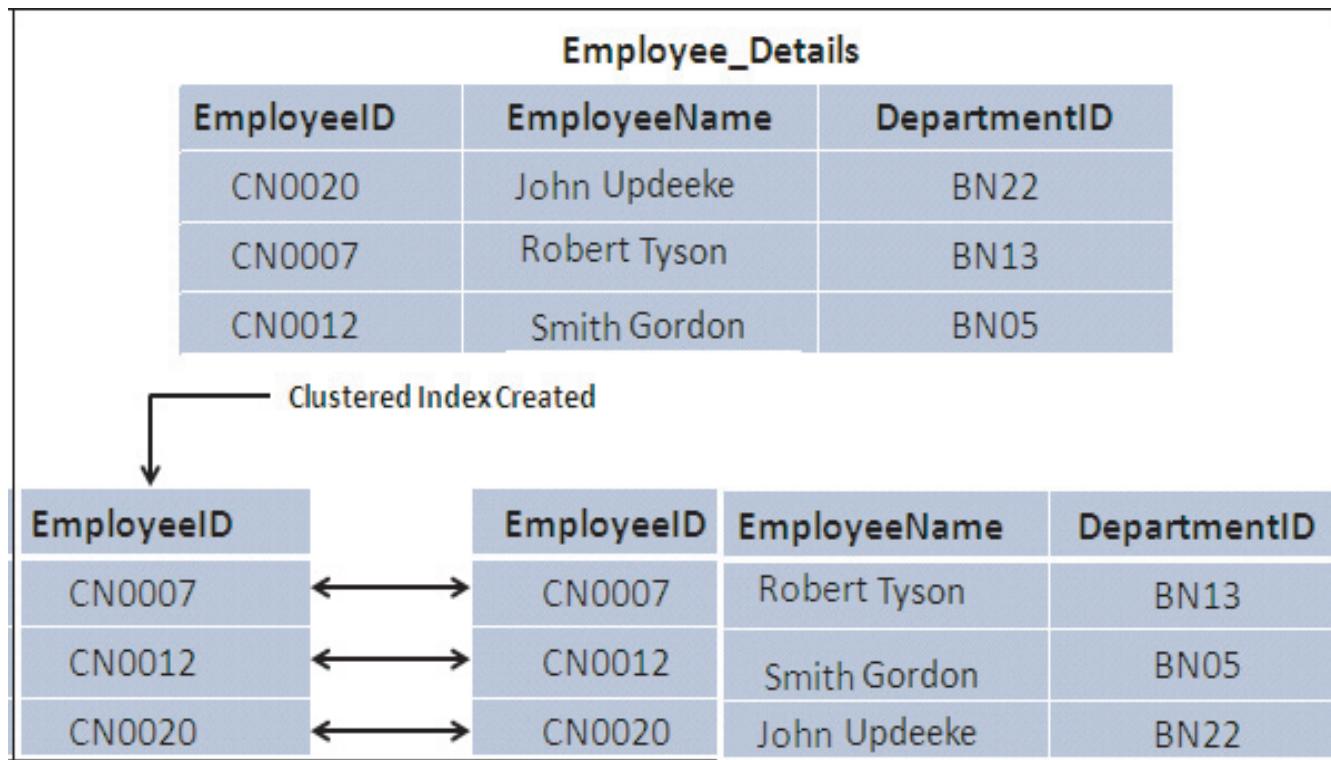


Hình 11.11: Phân vùng của cấu trúc heap

11.2.5 Các cấu trúc chỉ mục ghép cụm

Chỉ mục ghép cụm làm cho các bản ghi được lưu trữ vật lý theo thứ tự sắp xếp hoặc tuần tự. Chỉ mục ghép cụm xác định thứ tự thực tế theo đó dữ liệu được lưu trữ trong cơ sở dữ liệu. Do đó, bạn có thể tạo ra chỉ một chỉ mục ghép cụm trong một bảng.

Tính duy nhất của một giá trị trong một chỉ mục ghép cụm được duy trì một cách rõ ràng sử dụng từ khóa UNIQUE hoặc một cách ngầm hiểu sử dụng một mã định danh duy nhất nội bộ như được trình bày trong hình 11.12.



Hình 11.12: Các chỉ mục ghép cụm

11.2.6 Tạo chỉ mục ghép cụm

Chỉ mục ghép cụm làm cho các bản ghi được lưu trữ vật lý theo thứ tự sắp xếp hoặc tuần tự. Do đó, chỉ mục ghép cụm xác định thứ tự thực tế theo đó dữ liệu được lưu trữ trong cơ sở dữ liệu. Do đó, bạn có thể tạo ra chỉ một chỉ mục ghép cụm trong một bảng.

Chỉ mục ghép cụm được tạo ra bằng cách sử dụng câu lệnh CREATE INDEX với từ khóa CLUSTERED. Cú pháp sau đây tạo ra một chỉ mục ghép cụm trên một bảng đã chỉ định.

Cú pháp:

```
CREATE CLUSTERED INDEX index_name ON <table_name> (column_name)
```

trong đó:

CLUSTERED: Chỉ ra rằng một chỉ mục ghép cụm được tạo ra.

Đoạn mã 2 tạo ra chỉ mục ghép cụm **IX_CustID** trên cột **CustID** trong bảng **Customer_Details**.

Đoạn mã 2:

```
USE CUST_DB
CREATE CLUSTERED INDEX IX_CustID ON Customer_Details (CustID)
GO
```

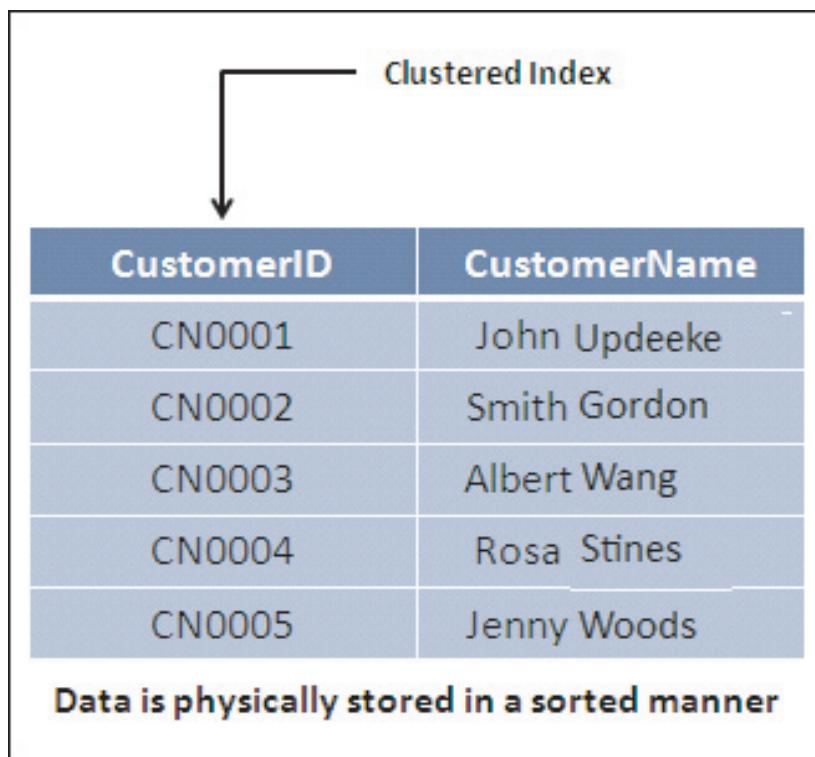
Ghi chú - Trước khi bạn tạo ra một chỉ mục ghép cụm, bạn cần phải chắc chắn rằng không gian trống trong hệ thống của bạn có ít nhất 1,2 lần số lượng dữ liệu trong bảng.

11.2.7 Truy cập dữ liệu với một chỉ mục ghép cụm

Chỉ mục ghép cụm có thể được tạo ra trên một bảng sử dụng một cột không có các giá trị trùng lặp. Chỉ mục này sẽ tổ chức lại các bản ghi theo thứ tự tuần tự của những giá trị trong cột chỉ mục.

Chỉ mục ghép cụm được sử dụng để định vị một hàng đơn lẻ hoặc một loạt các hàng. Bắt đầu từ trang đầu tiên của chỉ mục, giá trị tìm kiếm được kiểm tra đối với mỗi giá trị khóa trên trang này. Khi giá trị khóa so khớp được tìm thấy, công cụ cơ sở dữ liệu chuyển đến trang được chỉ ra bằng giá trị đó như được trình bày trong hình 11.13. Hàng mong muốn hoặc một loạt các hàng sau đó được truy cập.

Chỉ mục ghép cụm rất hữu ích cho các cột được tìm kiếm thường xuyên cho các giá trị khóa hoặc được truy cập theo thứ tự sắp xếp.



Hình 11.13: Truy cập dữ liệu với chỉ mục ghép cụm

Chỉ mục ghép cụm được tự động tạo ra trên một bảng khi khóa chính được định nghĩa trên bảng đó.

Trong bảng không có cột khóa chính, chỉ mục ghép cụm lý tưởng nên được định nghĩa trên:

- Cột khóa được tìm kiếm trên diện rộng.
- Cột được sử dụng trong các truy vấn trả lại các tập kết quả lớn.
- Cột có dữ liệu duy nhất.
- Cột được sử dụng trong phép nối bảng.

Ghi chú - Hai hoặc nhiều bảng có thể được nối theo logic qua các cột là phổ biến cho những bảng này. Sau đó dữ liệu có thể được lấy từ những bảng này như thể chúng là một bảng đơn lẻ.

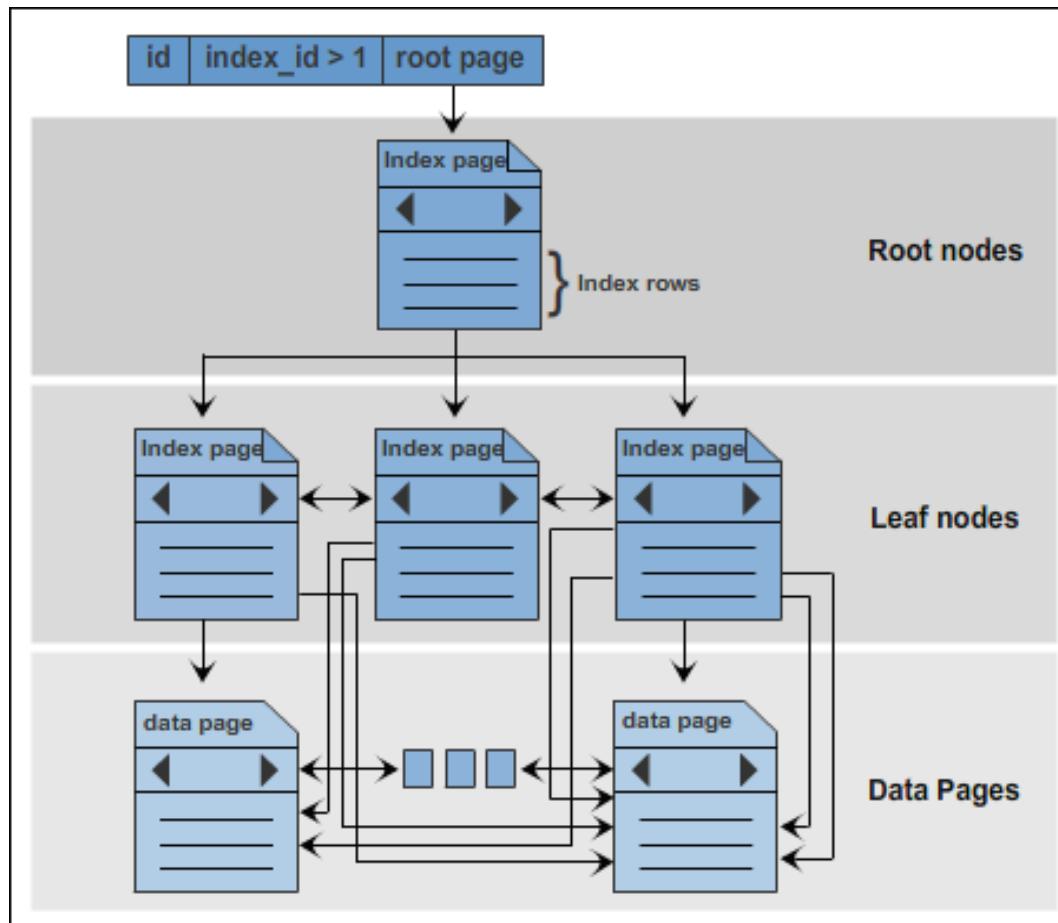
11.2.8 Các cấu trúc chỉ mục không ghép cụm

Chỉ mục không ghép cụm được định nghĩa trên bảng có dữ liệu trong cấu trúc hoặc heap ghép cụm. Chỉ mục không ghép cụm sẽ là loại mặc định nếu chỉ mục không được định nghĩa trên bảng. Mỗi hàng chỉ mục trong chỉ mục không ghép cụm chứa một giá trị khóa không ghép cụm và bộ định vị hàng. Bộ định vị hàng này trỏ tới hàng dữ liệu tương ứng với giá trị khóa trong bảng.

Các chỉ mục không ghép cụm có một cấu trúc B-Tree tương tự như các chỉ mục ghép cụm nhưng với những khác biệt sau:

- Những hàng dữ liệu của bảng không được lưu trữ vật lý theo thứ tự được định nghĩa bằng các khóa không ghép cụm của chúng.
- Trong cấu trúc chỉ mục không ghép cụm, mức lá có chứa các hàng chỉ mục.

Hình 11.14 trình bày cấu trúc chỉ mục không ghép cụm.



Hình 11.14: Cấu trúc chỉ mục không ghép cụm

Các chỉ mục không ghép cụm rất hữu ích khi bạn cần nhiều cách để tìm kiếm dữ liệu. Một số thực tế và hướng dẫn phải được xem xét trước khi tạo ra một chỉ mục không ghép cụm như sau:

- ➔ Khi một chỉ mục ghép cụm được tái tạo hoặc tùy chọn `DROP _ EXISTING` được sử dụng, SQL Server xây dựng lại các chỉ mục không ghép cụm hiện có.
- ➔ Một bảng có thể có đến 999 chỉ mục không ghép cụm.
- ➔ Tạo chỉ mục ghép cụm trước khi tạo ra một chỉ mục không ghép cụm.

Cú pháp sau đây tạo ra một chỉ mục không ghép cụm.

Cú pháp:

```
CREATE NONCLUSTERED INDEX <index_name> ON <table_name> (column_name)
```

trong đó:

NONCLUSTERED: chỉ ra rằng một chỉ mục không ghép cụm được tạo ra.

Đoạn mã 3 tạo ra chỉ mục không ghép cụm **IX_State** trên cột **State** trong bảng **Customer_Details**.

Đoạn mã 3:

```
USE CUST_DB
CREATE NONCLUSTERED INDEX IX_State ON Customer_Details (State)
GO
```

11.2.9 Chỉ mục lưu trữ cột

Chỉ mục lưu trữ cột là một tính năng mới trong SQL Server 2012. Nó giúp tăng cường hiệu suất của các truy vấn kho dữ liệu một cách bao quát. Những chỉ mục thông thường hoặc các heap của các máy chủ SQL cũ lưu trữ dữ liệu trong cấu trúc B-Tree theo hàng, nhưng chỉ mục lưu trữ cột trong SQL Server 2012 lưu trữ dữ liệu theo cột. Do tốc độ truyền dữ liệu chậm trong các máy chủ cơ sở dữ liệu, do đó chỉ mục lưu trữ cột sử dụng phép nén mạnh mẽ để giảm nhập/xuất đĩa cần thiết để phục vụ yêu cầu truy vấn.

B-Tree và heap lưu trữ dữ liệu theo hàng, có nghĩa là dữ liệu từ tất cả các cột của một hàng được lưu trữ cùng nhau liên tục kế nhau trên cùng một trang.

Ví dụ, nếu có một bảng với mươi cột (C1 đến C10), dữ liệu của tất cả mươi cột từ mỗi hàng được lưu trữ cùng nhau liên tục kế nhau trên cùng một trang như được trình bày trong hình 11.15.

Row store for B-Tree or Heap										
Row 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Page 1										
Row 6	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 7	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
.....	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row n	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Page 2										

Hình 11.15: Chỉ mục B-Tree

Khi chỉ mục lưu trữ cột được tạo ra, dữ liệu được lưu trữ theo cột, có nghĩa là dữ liệu của mỗi cột riêng lẻ từ mỗi hàng được lưu trữ với nhau trên cùng một trang.

Ví dụ, dữ liệu của cột C1 của tất cả các hàng được lưu trữ cùng nhau trên một trang và dữ liệu cho cột C2 của tất cả các hàng được lưu trữ trên một trang khác và vân vân như được trình bày trong hình 11.16.

Column Store Index										
Row 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 6	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 7	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
.....	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row n	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
	Page 1	Page 2	Page 3	Page 4	Page 5	Page 6	Page 7	Page 8	Page 9	Page 10

Hình 11.16: Chỉ mục lưu trữ cột

Sau đây là cú pháp để tạo ra một chỉ mục lưu trữ cột:

Cú pháp:

```
CREATE [ NONCLUSTERED ] COLUMNSTORE INDEX index_name ON <object> ( column
[ ,...n ] )
[ WITH ( <column_index_option> [ ,...n ] ) ]
ON
```

Giả sử rằng một bảng có tên là **ResellerSalesPtnd** đã được tạo ra trong cơ sở dữ liệu AdventureWorks2012. Đoạn mã 4 trình bày cách tạo ra một chỉ mục lưu trữ cột trên bảng này.

Đoạn mã 4:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX [csindx_ResellerSalesPtnd]
ON [ResellerSalesPtnd]
```

```
(  
    [ProductKey],  
    [OrderDateKey],  
    [DueDateKey],  
    [ShipDateKey],  
    [CustomerKey],  
    [EmployeeKey],  
    [PromotionKey],  
    [CurrencyKey],  
    [SalesTerritoryKey],  
    [SalesOrderNumber],  
    [SalesOrderLineNumber],  
    [RevisionNumber],  
    [OrderQuantity],  
    [UnitPrice],  
    [ExtendedAmount],  
    [UnitPriceDiscountPct],  
    [DiscountAmount],  
    [ProductStandardCost],  
    [TotalProductCost],  
    [SalesAmount],  
    [TaxAmt],  
    [Freight],  
    [CarrierTrackingNumber],  
    [CustomerPONumber],  
    [OrderDate],  
    [DueDate],  
    [ShipDate]  
);
```

Ghi chú - COLUMNSTORE INDEX chỉ hoạt động trên phiên bản doanh nghiệp của SQL Server 2012.

11.2.10 Thả bỏ một chỉ mục

Khi thả bỏ một chỉ mục ghép cụm, những hàng ở cấp độ lá của chỉ mục ghép cụm được sao chép vào heap. Tất cả các chỉ mục không ghép cụm trên bảng này sau đó sẽ trở đến heap trong đó dữ liệu được lưu trữ. Điều này được thực hiện bằng cách xây dựng lại các chỉ mục không ghép cụm khi chỉ mục ghép cụm được thả bỏ. Do đó, việc thả bỏ chỉ mục ghép cụm là một quá trình tốn nhiều thời gian. Vì vậy, trong khi thả bỏ tất cả các chỉ mục trên bảng, trước tiên bạn phải thả bỏ các chỉ mục không ghép cụm đầu tiên và sau đó, các chỉ mục ghép cụm.

SQL Server 2012 có thể thả bỏ chỉ mục ghép cụm và di chuyển heap (bảng không có thứ tự) vào một nhóm tập tin khác hoặc một lược đồ phân vùng sử dụng tùy chọn MOVE TO.

- ➔ Tùy chọn này là không hợp lệ cho các chỉ mục không ghép cụm.
- ➔ Lược đồ phân vùng hoặc nhóm tập tin được chỉ ra trong mệnh đề MOVE TO phải tồn tại.
- ➔ Bảng sẽ được đặt trong cùng một lược đồ phân vùng hoặc nhóm tập tin của chỉ mục ghép cụm đã thả bỏ.

Sau đây là cú pháp để thả bỏ một chỉ mục ghép cụm.

Cú pháp:

```
DROP INDEX <index_name> ON <table_name>
[ WITH ( MOVE TO { <partition_scheme_name> ( <column_name> )
| <filegroup_name>
| 'default'
} )
]
```

trong đó:

index_name: chỉ ra tên của chỉ mục.

partition_scheme_name: chỉ ra tên của lược đồ phân vùng.

filegroup_name: chỉ ra tên của nhóm tập tin để lưu trữ các phân vùng.

default: chỉ ra vị trí mặc định để lưu trữ bảng kết quả.

Đoạn mã 5 thả bỏ chỉ mục **IX_SuppID** được tạo ra trên cột **SuppID** của bảng **Supplier_Details**.

Đoạn mã 5:

```
DROP INDEX IX_SuppID ON Supplier_Details
WITH (MOVE TO 'default')
```

Dữ liệu trong bảng **Supplier_Details** kết quả được chuyển đến vị trí mặc định.

Đoạn mã 6 thả bỏ chỉ mục **IX_SuppID** được tạo ra trên cột **SuppID** của bảng **Supplier_Details**.

Đoạn mã 6:

```
DROP INDEX IX_SuppID ON Supplier_Details
WITH (MOVE TO FGCountry)
```

Dữ liệu trong bảng **Supplier_Details** kết quả được chuyển đến nhóm tập tin **FGCountry**.

11.2.11 Sự khác nhau giữa chỉ mục ghép cụm và không ghép cụm

Các chỉ mục ghép cụm và không ghép cụm là khác nhau về kiến trúc của chúng và tính hữu dụng của chúng trong các lần thực thi truy vấn. Bảng 11.1 liệt kê ra những khác biệt giữa các chỉ mục ghép cụm và không ghép cụm.

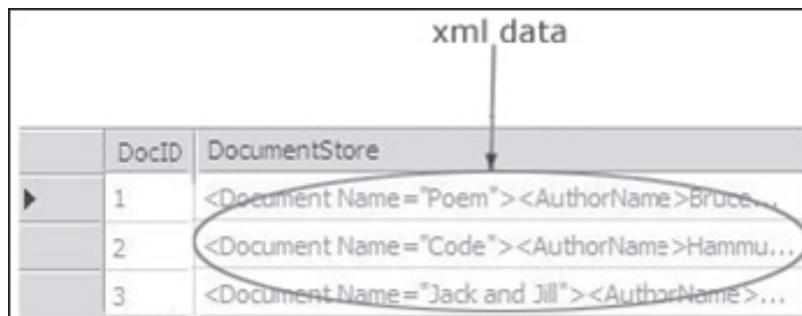
Các chỉ mục ghép cụm	Các chỉ mục không ghép cụm
Được sử dụng trong các truy vấn trả lại các tập kết quả lớn	Được sử dụng trong các truy vấn không trả lại các tập kết quả lớn
Chỉ một chỉ mục ghép cụm có thể được tạo ra trên một bảng	Nhiều chỉ mục không ghép cụm có thể được tạo ra trên một bảng
Dữ liệu được lưu trữ theo một cách có sắp xếp trên khóa ghép cụm	Dữ liệu không được lưu trữ theo một cách có sắp xếp trên khóa không ghép cụm
Các nút lá của một chỉ mục ghép cụm chứa các trang dữ liệu	Những nút lá của một chỉ mục không ghép cụm chứa các trang chỉ mục

Bảng 11.1: Các sự khác nhau giữa chỉ mục ghép cụm và không ghép cụm

11.2.12 Các chỉ mục XML

Kiểu dữ liệu **xml** được sử dụng để lưu trữ các tài liệu và phân đoạn XML như được trình bày trong hình 11.17. Một đoạn XML là một thể hiện XML có một phần tử mức cao nhất bị mất tích.

Do kích thước lớn của các cột XML, các truy vấn tìm kiếm trong những cột này có thể sẽ chậm. Bạn có thể tăng tốc những truy vấn này bằng cách tạo ra một chỉ mục XML trên mỗi cột. Chỉ mục XML có thể là một chỉ mục ghép cụm hoặc không ghép cụm. Mỗi bảng có thể có đến 249 chỉ mục XML.



Hình 11.17: Kiểu dữ liệu XML

Ghi chú - XML là một văn bản đơn giản, ngôn ngữ dựa trên Unicode cung cấp cơ chế để mô tả cấu trúc tài liệu sử dụng các thẻ đánh dấu. Ví dụ, hãy xem xét một tổ chức có các chi tiết của nhân viên được lưu trữ trong tài liệu XML. Thông tin cho mỗi nhân viên được lưu trữ ở định dạng sau đây:

```
<Employees>
<Name>John</Name>
<Age>34</Age>
<Salary>500000</Salary>
</Employees>
```

Ở đây, `<Employees>` là nút gốc và `<Name>`, `<Age>`, và `<Salary>` là những nút con.

11.2.13 Các loại chỉ mục XML

Các chỉ mục XML có thể được tạo ra trên một bảng chỉ khi có một chỉ mục ghép cụm dựa trên khóa chính của bảng. Khóa chính này không thể vượt quá 15 cột.

Những loại chỉ mục XML khác nhau như sau:

- ➔ **Chỉ mục XML chính** - Quá trình thực hiện các truy vấn trong một cột XML đôi khi có thể rất chậm. Chỉ mục XML chính được tạo ra trên mỗi cột XML để tăng tốc những truy vấn này. Nó là một chỉ mục đặc biệt cắt nhỏ dữ liệu XML để lưu trữ thông tin. Sau đây là cú pháp để tạo ra một chỉ mục XML chính:

Cú pháp:

```
CREATE PRIMARY XML INDEX index_name ON <table_name> (column_name)
```

Đoạn mã 7 tạo ra một chỉ mục XML chính trên cột CatalogDescription trong bảng Production.ProductModel.

Đoạn mã 7:

```
USE AdventureWorks2012;
CREATE PRIMARY XML INDEX PXML_ProductModel_CatalogDescription
    ON Production.ProductModel (CatalogDescription);
GO
```

- ➔ **Chỉ mục XML phụ** - Chỉ mục XML phụ là chỉ mục XML chuyên biệt trợ giúp với các truy vấn XML cụ thể. Những tính năng của các chỉ mục XML phụ như sau:

- Tìm kiếm các giá trị bất cứ nơi nào trong tài liệu XML.
- Lấy các thuộc tính đối tượng đặc biệt từ bên trong một tài liệu XML.

Các chỉ mục XML phụ chỉ có thể được tạo ra trên các cột đã có một chỉ mục XML chính.

Đoạn mã 8 trình bày cách tạo ra một chỉ mục XML phụ trên cột CatalogDescription trong bảng Production.ProductModel.

Đoạn mã 8:

```
USE AdventureWorks2012;
CREATE XML INDEX IX_ProductModel_CatalogDescription_Path
    ON Production.ProductModel (CatalogDescription)
    USING XML INDEX PXML_ProductModel_CatalogDescription FOR PATH ;
GO
```

- **Chỉ mục XML có lựa chọn (SXI)** – Đây là một loại chỉ mục XML mới đã được giới thiệu trong SQL Server 2012. Những tính năng của chỉ mục mới này là để cải thiện hiệu suất truy vấn trên dữ liệu được lưu trữ là XML trong SQL Server, cho phép lập chỉ mục nhanh hơn của khối lượng công việc dữ liệu XML lớn, và cải thiện khả năng mở rộng bằng cách giảm chi phí lưu trữ của chỉ mục. Sau đây là cú pháp để tạo ra một chỉ mục XML có lựa chọn:

Cú pháp:

```
CREATE SELECTIVE XML INDEX index_name ON <table_name> (column_name)
```

Sau đây là một tài liệu XML trong một bảng gồm khoảng 500.000 hàng.

```
<book>
    <created>2004-03-01</created>
    <authors>Khác nhau</authors>
    <subjects>
        <subject>Trí tuệ và hài hước tiếng Anh -- Tạp chí</subject>
        <subject>AP</subject>
    </subjects>
    <title>Punch, or the London Charivari, Tập 156, ngày 2/4/1919</title>
    <id>etext11617</id>
</book>
```

Đoạn mã 9 trình bày cách tạo ra một chỉ mục XML có lựa chọn trên cột **SnnbBookDetailsSnnb** trong bảng **BooksBilling**.

Đoạn mã 9:

```
USE CUST_DB
CREATE SELECTIVE XML INDEX SXI_index
ON BooksBilling (BookDetails)
FOR
(
    pathTitle = '/book/title/text()' AS XQUERY 'xs:string',
    pathAuthors = '/book/authors' AS XQUERY 'node()',
    pathId = '/book/id' AS SQL NVARCHAR(100)
)
GO
```

Ghi chú - SELECTIVE XML INDEX sẽ chỉ làm việc trong phiên bản doanh nghiệp của SQL Server 2012.

11.2.14 Sửa đổi chỉ mục XML

Chỉ mục XML, chính hay phụ, có thể được sửa đổi bằng cách sử dụng câu lệnh ALTER INDEX.

Cú pháp:

```
ALTER INDEX <xml_index_name> ON <table_name> REBUILD
```

trong đó:

`xml_index_name`: chỉ ra tên của chỉ mục XML.

Đoạn mã 10 xây dựng lại chỉ mục XML chính **PXML_DocumentStore** được tạo ra trên bảng **XMLDocument**.

Đoạn mã 10:

```
ALTER INDEX PXML_DocumentStore ON XMLDocument REBUILD
```

11.2.15 Loại bỏ chỉ mục XML

Sau đây là cú pháp để loại bỏ chỉ mục XML sử dụng câu lệnh DROP INDEX.

Cú pháp:

```
DROP INDEX <xml_index_name> ON <table_name>
```

Đoạn mã 11 loại bỏ chỉ mục XML chính `PXML_DocumentStore` được tạo ra trên bảng `XMLDocument`.

Đoạn mã 11:

```
DROP INDEX PXML_DocumentStore ON XMLDocument
```

11.3 Các đơn vị phân bổ

Heap hoặc cấu trúc chỉ mục ghép cụm có chứa các trang dữ liệu trong một hoặc nhiều đơn vị phân bổ. Đơn vị phân bổ là một tập hợp các trang và được sử dụng để quản lý dữ liệu dựa trên loại trang của chúng. Những loại đơn vị phân bổ được sử dụng để quản lý dữ liệu trong các bảng và chỉ mục như sau:

→ IN_ROW_DATA

Nó được sử dụng để quản lý các hàng dữ liệu hoặc chỉ mục có chứa tất cả các loại dữ liệu ngoại trừ dữ liệu đối tượng lớn (LOB).

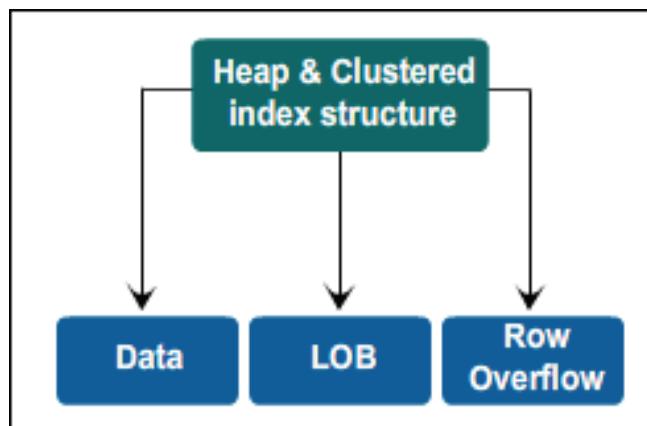
→ LOB_DATA

Nó được sử dụng để quản lý dữ liệu đối tượng lớn, được lưu trữ trong một hoặc nhiều kiểu dữ liệu sau: `varbinary(max)`, `varchar(max)`, và `xml`.

→ ROW_OVERFLOW_DATA

Nó được sử dụng để quản lý dữ liệu có độ dài biến đổi, được lưu trữ trong các cột `varchar`, `nvarchar`, `varbinary`, hoặc `sql_variant`.

Hình 11.18 trình bày các đơn vị phân bổ.



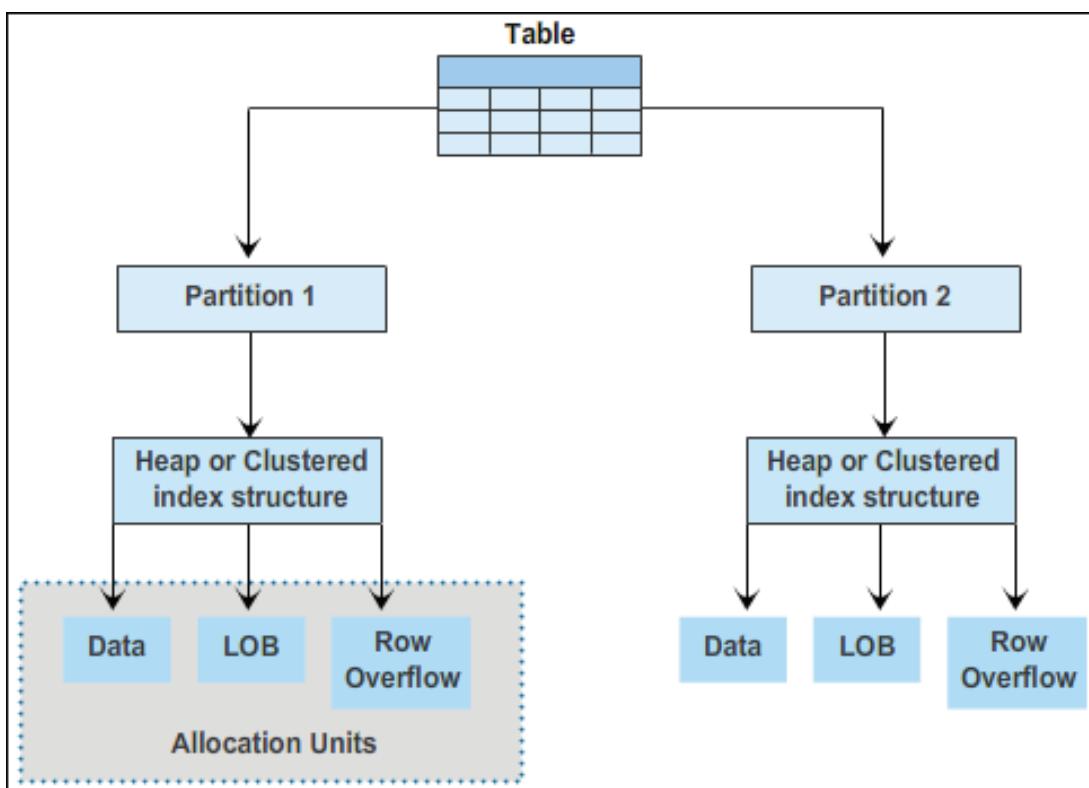
Hình 11.18: Các đơn vị phân bổ

Ghi chú - Heap có thể chỉ có một đơn vị phân bổ thuộc từng loại trong một phân vùng cụ thể của một bảng.

11.4 Phân vùng

Phân vùng chia dữ liệu thành các tập con. Điều này làm cho các bảng lớn hoặc chỉ mục dễ quản lý hơn. Phân vùng cho phép bạn truy cập dữ liệu một cách nhanh chóng và hiệu quả. Các hoạt động bảo trì trên một phần nhỏ của dữ liệu được thực hiện hiệu quả hơn bởi vì chúng chỉ nhắm tới phần nhỏ của dữ liệu yêu cầu thay vì toàn bộ bảng.

Theo mặc định, bảng hoặc chỉ mục chỉ có một phân vùng chứa tất cả các dữ liệu hoặc các trang chỉ mục. Khi một bảng hoặc chỉ mục sử dụng nhiều phân vùng, dữ liệu được phân chia theo chiều ngang thành các nhóm hàng như được trình bày trong hình 11.19.



Hình 11.19: Phân vùng

11.4.1 Khung nhìn sys.partitions

Khung nhìn sys.partitions là một khung nhìn hệ thống có chứa thông tin đầy đủ về những phân vùng khác nhau của tất cả các bảng và chỉ mục trong cơ sở dữ liệu.

Bảng 11.2 trình bày các cột khác nhau của khung nhìn sys.partitions cùng với các kiểu dữ liệu và mô tả của chúng:

Tên cột	Loại dữ liệu	Mô tả
partition_id	bigint	Chứa id của phân vùng và là duy nhất trong một cơ sở dữ liệu.
object_id	int	Chứa id của đối tượng có chứa phân vùng đó.

Tên cột	Loại dữ liệu	Mô tả
index_id	int	Chứa id của chỉ mục nơi phân vùng thuộc đó.
partition_number	int	Chứa số phân vùng trong chỉ mục hoặc heap.
hobt_id	bigint	Chứa id của heap dữ liệu hoặc B-Tree có chứa các hàng cho phân vùng đó.
rows	bigint	Chỉ rõ số lượng gần đúng của các hàng trong phân vùng.

Bảng 11.2: Các cột của các kiểu dữ liệu và khung nhìn sys.partitions

11.4.2 Cột index_id

Cột `index_id` chứa id của chỉ mục nơi phân vùng thuộc vào đó. Khung nhìn danh mục `sys.partitions` trả về một hàng cho mỗi phân vùng trong một bảng hoặc chỉ mục. Các giá trị của cột `index_id` là duy nhất trong bảng, trong đó phân vùng được tạo ra. Kiểu dữ liệu của cột `index_id` là `int`.

Sau đây là các giá trị khác nhau của cột `index_id`:

- ➔ Giá trị `index_id` cho một heap là 0.
- ➔ Giá trị `index_id` cho một chỉ mục ghép cụm là 1.
- ➔ Giá trị `index_id` cho một chỉ mục không ghép cụm lớn hơn 1.
- ➔ Giá trị `index_id` cho các đối tượng lớn sẽ lớn hơn 250.

Hình 11.20 trình bày cột `index_id` trong khung nhìn `sys.partitions`.

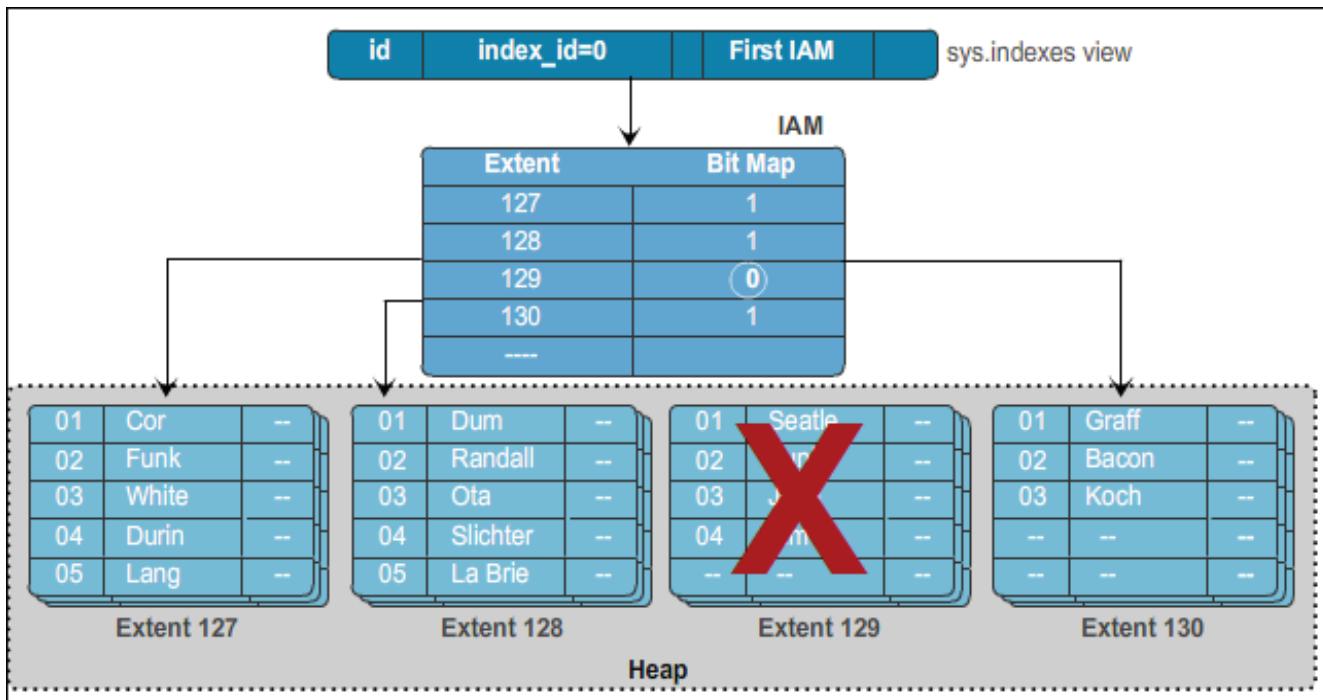
	partition_id	object_id	index_id	partition_number	hobt_id	rows
1	196608	3	1	1	196608	1779
2	327680	5	1	1	327680	332
3	458752	7	1	1	458752	379
4	524288	8	0	1	524288	2
5	281474977103872	6	1	1	281474977103872	0
6	281474977300480	9	1	1	281474977300480	0
7	281474977824768	17	1	1	281474977824768	0
8	281474977890304	18	1	1	281474977890304	0
9	281474977955840	19	1	1	281474977955840	0
10	281474978021376	20	1	1	281474978021376	2

Hình 11.20: Cột Index_id

11.5 Tìm các hàng

SQL Server sử dụng các khung nhìn danh mục để tìm các hàng khi chỉ mục không được tạo ra trên một bảng. Nó sử dụng khung nhìn sys.indexes để tìm trang IAM. Trang IAM này chứa một danh sách tất cả các trang của một bảng cụ thể thông qua đó SQL Server có thể đọc tất cả các trang dữ liệu.

Khi khung nhìn sys.indexes được sử dụng, trình tối ưu truy vấn kiểm tra tất cả các hàng trong một bảng và trích xuất chỉ những hàng được tham chiếu trong truy vấn như được trình bày trong hình 11.21. Việc quét này tạo ra nhiều hoạt động nhập/xuất và tận dụng nhiều tài nguyên.



Hình 11.21: Tìm các hàng không có chỉ mục

Đoạn mã 12 trình bày cách tạo ra bảng Employee_Details không có chỉ mục.

Đoạn mã 12:

```
USE CUST_DB
CREATE TABLE Employee_Details
(
    EmpID int not null,
    FirstName varchar(20) not null,
    LastName varchar(20) not null,
    DateofBirth datetime not null,
    Gender varchar(6) not null,
```

```
City varchar (30) not null,
)
GO
```

Giả sử rằng nhiều bản ghi được chèn vào trong bảng **Employee_Details**. Câu lệnh SELECT được sử dụng để tìm kiếm các bản ghi có **FirstName** là John.

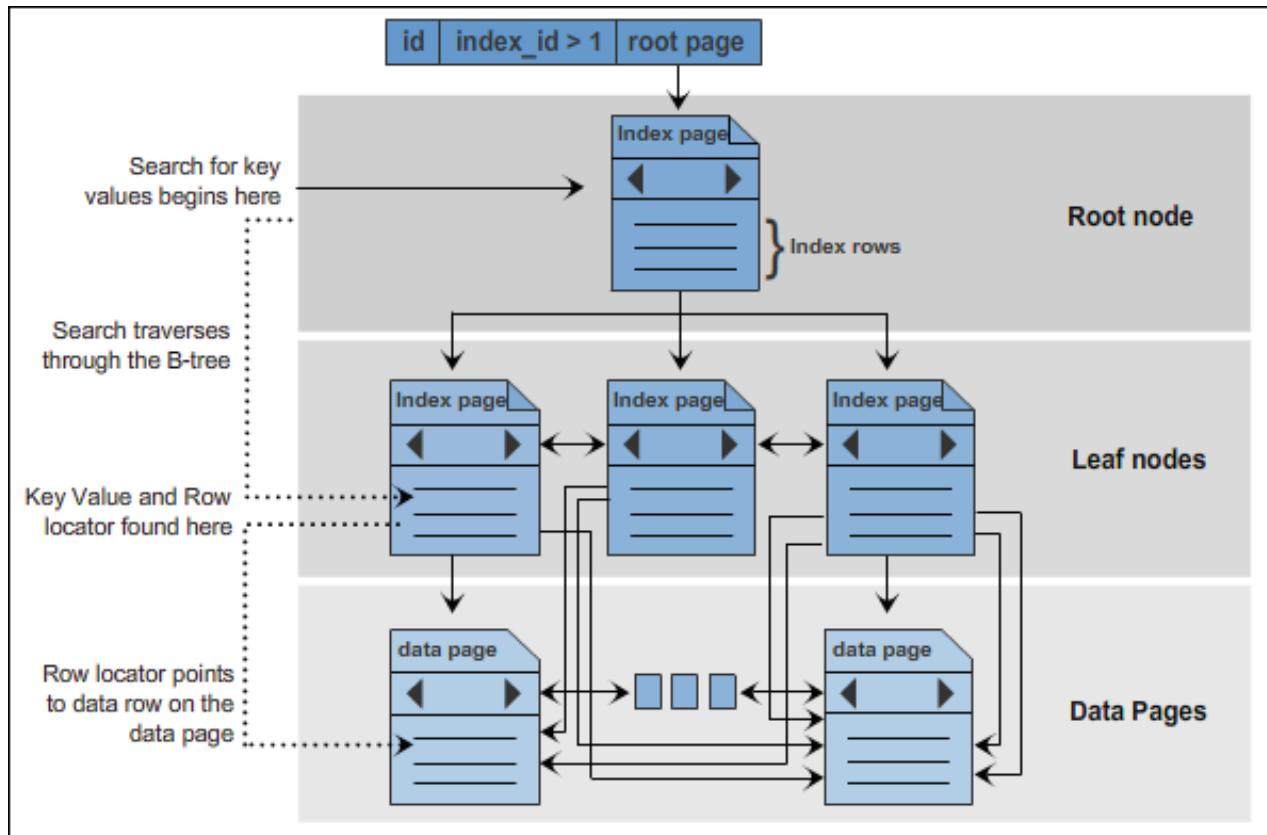
Vì không có chỉ mục liên quan đến cột **FirstName**, SQL Server sẽ thực hiện một lần quét bảng đầy đủ.

11.5.1 Tìm các dòng có chỉ mục không ghép cụm

Chỉ mục không ghép cụm tương tự như chỉ mục cuốn sách, dữ liệu và chỉ mục được lưu trữ ở những nơi khác nhau. Những con trỏ ở cấp độ lá của chỉ mục trỏ đến vị trí lưu trữ của dữ liệu trong bảng bên dưới. Chỉ mục không ghép cụm được sử dụng để tìm kiếm các truy vấn khớp chính xác. Điều này là do chỉ mục chứa các mục nhập mô tả vị trí chính xác của dữ liệu trong bảng.

Để tìm kiếm các hàng sử dụng các chỉ mục không ghép cụm, câu lệnh SELECT được sử dụng với cột chỉ mục không ghép cụm đã chỉ định trong mệnh đề WHERE.

Hình 11.22 trình bày quá trình tìm các hàng với chỉ mục không ghép cụm.



Hình 11.22: Tìm các dòng có chỉ mục không ghép cụm

Đoạn mã 13 trình bày cách tạo ra một chỉ mục không ghép cụm **IX_EmployeeCity** trên cột **City** của bảng **Employee_Details**.

Đoạn mã 13:

```
USE CUST_DB
CREATE NONCLUSTERED INDEX IX_EmployeeCity ON Employee_Details(City);
GO
```

Giả sử rằng nhiều bản ghi được chèn vào trong bảng **Employee_Details**. Câu lệnh **SELECT** được sử dụng để tìm kiếm các hồ sơ của người lao động từ thành phố Boston như được trình bày trong Đoạn mã 14.

Đoạn mã 14:

```
USE CUST_DB
SELECT EmpID, FirstName, LastName, City FROM Employee_Details WHERE City='Boston'
GO
```

Do có một chỉ mục không ghép cụm liên quan đến cột **City**, SQL Server sẽ sử dụng chỉ mục **IX_EmployeeCity** để trích xuất các bản ghi như được trình bày trong hình 11.23.

	EmpID	FirstName	LastName	City
1	101	Andrew	Waller	Boston
2	103	Sophia	Broderich	Boston

Hình 11.23: Chỉ mục IX_EmployeeCity

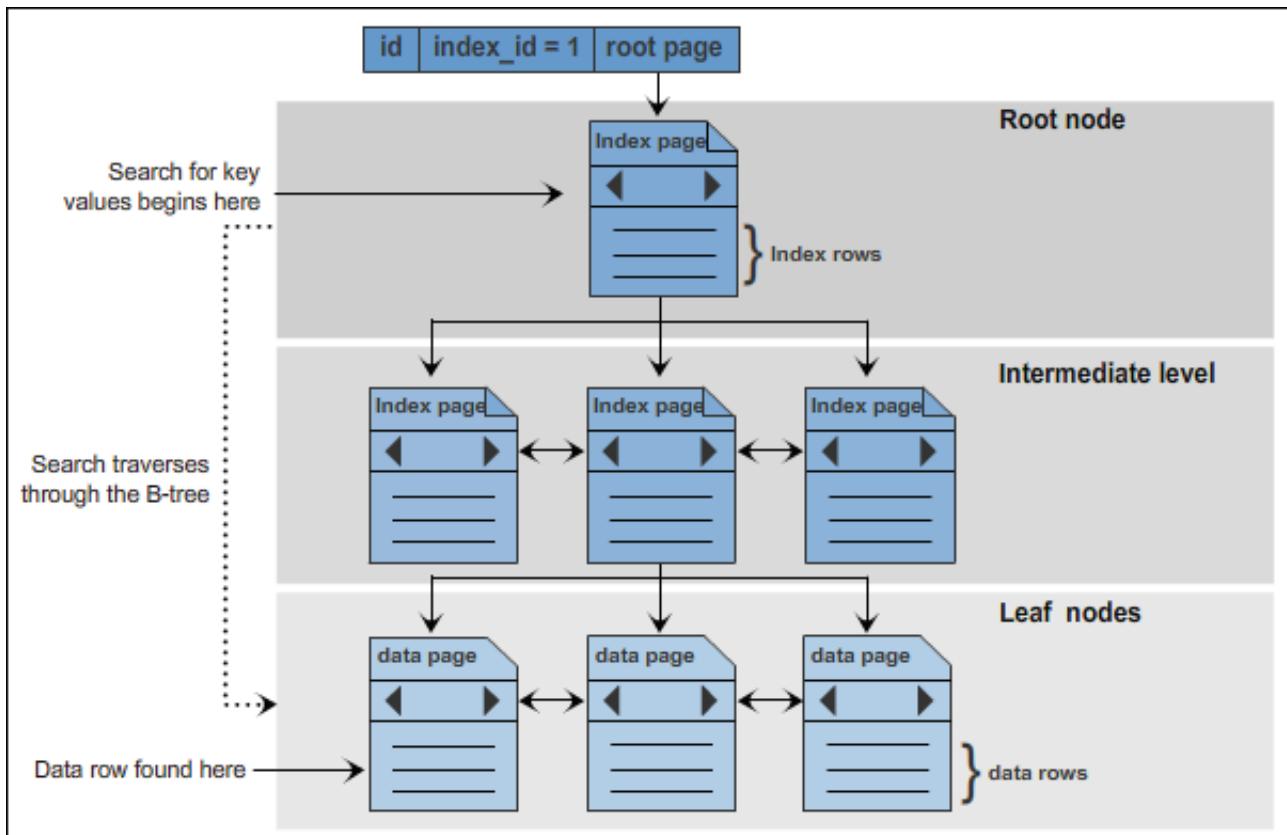
Ghi chú - Trong bảng có một chỉ mục không ghép cụm, không có thứ tự lưu trữ cụ thể của dữ liệu. Dữ liệu được lấy trực tiếp từ vị trí vật lý của nó.

11.5.2 Tìm các hàng trong một chỉ mục ghép cụm

Các chỉ mục ghép cụm lưu trữ các hàng dữ liệu trong bảng dựa trên các giá trị khóa của chúng. Dữ liệu này được lưu trữ theo một cách có sắp xếp. Nếu giá trị khóa ghép cụm nhỏ, có thể đặt thêm số lượng các hàng chỉ mục trên trang chỉ mục. Điều này làm giảm số lượng các mức trong B-Tree chỉ mục phải đi qua để đạt tới các hàng dữ liệu tạo ra các kết quả truy vấn nhanh hơn. Điều này làm giảm thiểu tổng lượng nhập/xuất.

Để tìm kiếm các hàng sử dụng các chỉ mục ghép cụm, câu lệnh **SELECT** được sử dụng với cột chỉ mục ghép cụm đã chỉ ra trong mệnh đề **WHERE**.

Hình 11.24 trình bày quá trình tìm các hàng với chỉ mục ghép cụm.



Hình 11.24: Tìm các hàng có chỉ mục ghép cụm

Đoạn mã 15 trình bày cách tạo ra một chỉ mục ghép cụm **IX_EmployeeID** trên cột **EmpID** của bảng **Employee_Details**.

Đoạn mã 15:

```
USE CUST_DB
CREATE UNIQUE CLUSTERED INDEX IX_EmployeeID ON Employee_Details (EmpID);
GO
```

Giả sử rằng nhiều bản ghi được chèn vào trong bảng **Employee_Details**. Câu lệnh **SELECT** được sử dụng để tìm kiếm các hồ sơ của người lao động có **EmpID** từ 102 đến 105 như được trình bày trong Đoạn mã 16.

Đoạn mã 16:

```
USE CUST_DB
SELECT EmpID, FirstName, LastName, City FROM Employee_Details WHERE EmpID >= 102
AND EmpID <= 105;
GO
```

Do có một chỉ mục ghép cụm liên quan đến cột `EmpID`, SQL Server sẽ sử dụng chỉ mục `IX_EmployeeID` để trích xuất các bản ghi như được trình bày trong hình 11.25.

	EmpID	FirstName	LastName	City
1	102	AJ	Stiles	Liverpool
2	103	Sophia	broderich	Boston
3	104	Shawn	roderichs	Texas

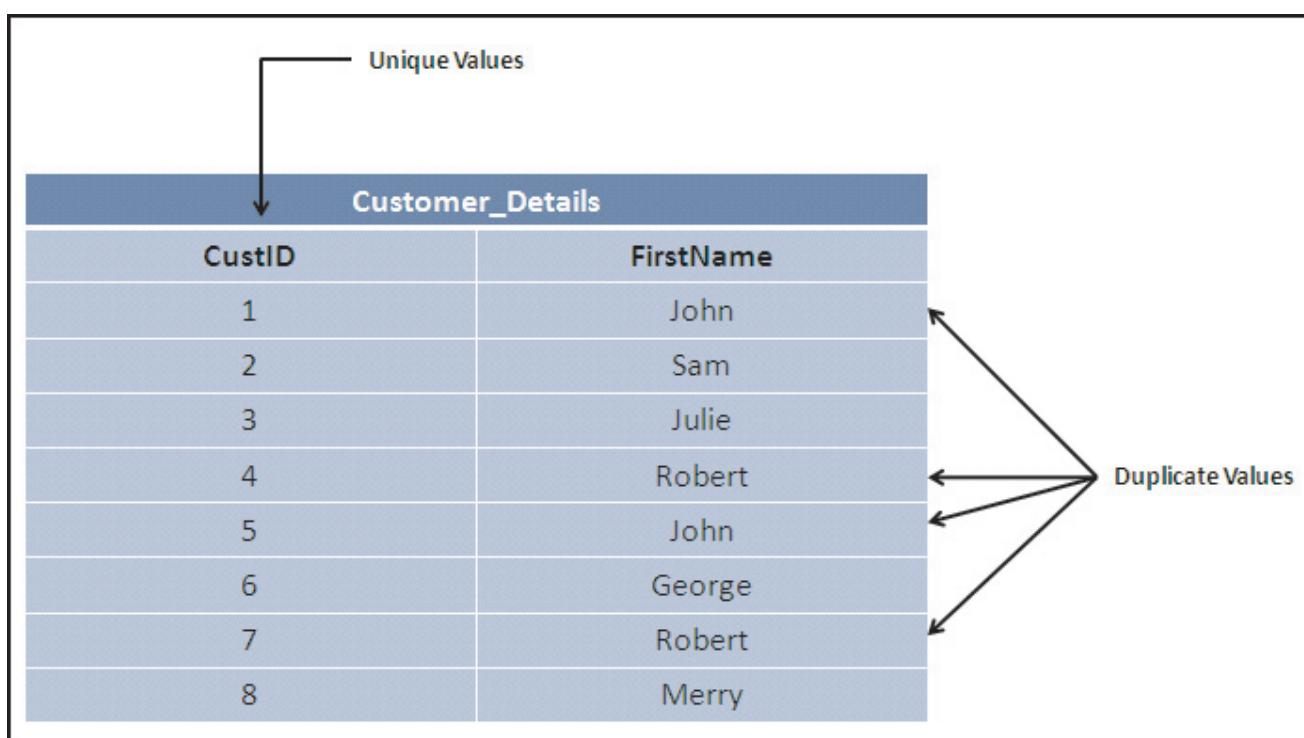
Hình 11.25: Chỉ mục `IX_EmployeeID`

11.5.3 Tạo các chỉ mục duy nhất

Chỉ mục duy nhất có thể được tạo ra trên một cột không có bất kỳ giá trị trùng lặp nào. Ngoài ra, một khi chỉ mục duy nhất được tạo ra, giá trị trùng lặp sẽ không được nhận trong cột này. Do đó, các chỉ mục duy nhất nên được tạo ra chỉ trên các cột nơi tính duy nhất của các giá trị là một đặc điểm quan trọng. Chỉ mục duy nhất đảm bảo tính toàn vẹn thực thể trong một bảng.

Nếu định nghĩa bảng có cột `PRIMARY KEY` hoặc một cột với ràng buộc `UNIQUE`, SQL Server tự động tạo ra một chỉ mục duy nhất khi bạn thực thi câu lệnh `CREATE TABLE` như được trình bày trong hình 11.26.

Chỉ mục duy nhất có thể được tạo ra bằng cách sử dụng câu lệnh `CREATE UNIQUE INDEX` hoặc sử dụng SSMS.



Hình 11.26: Tạo các chỉ mục duy nhất

Cú pháp sau đây được sử dụng để tạo ra chỉ mục duy nhất.

Cú pháp:

```
CREATE UNIQUE INDEX <index_name> ON <table_name> (<column_name>)
```

trong đó:

`column_name`: chỉ ra tên của cột trên đó chỉ mục sẽ được tạo ra.

`UNIQUE`: chỉ ra rằng không có các giá trị trùng lặp nào được cho phép trong cột này.

Đoạn mã 17 tạo ra chỉ mục duy nhất trên cột `CustID` trong bảng `Customer_Details`.

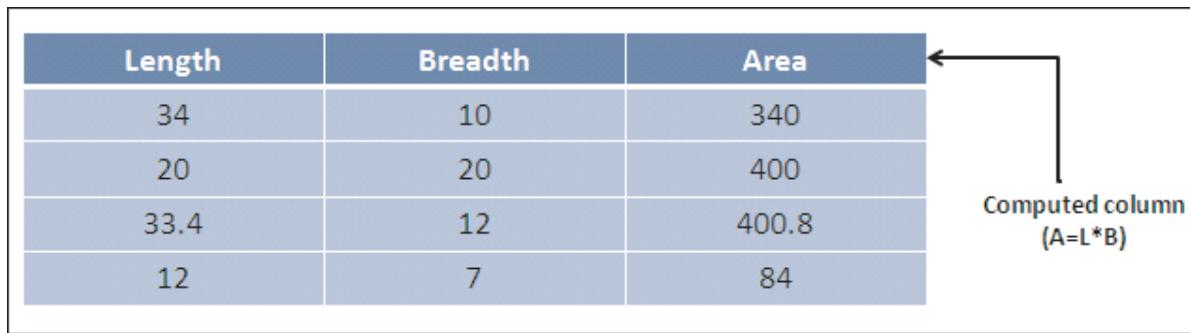
Đoạn mã 17:

```
CREATE UNIQUE INDEX IX_CustID ON Customer_Details (CustID)
```

11.5.4 Tạo các cột tính

Cột tính là một cột ảo trong một bảng có giá trị được tính vào thời gian chạy. Những giá trị trong cột này không được lưu trữ trong bảng nhưng được tính toán dựa trên biểu thức định nghĩa cột đó. Biểu thức này bao gồm tên cột không tính toán kết hợp với hằng số, hàm, hoặc biến sử dụng các toán tử số học hoặc logic.

Một cột tính có thể được tạo ra sử dụng các câu lệnh `CREATE TABLE` hoặc `ALTER TABLE` như được trình bày trong hình 11.27.



Length	Breadth	Area
34	10	340
20	20	400
33.4	12	400.8
12	7	84

Computed column
(A=L*B)

Hình 11.27: Tạo ra các cột tính

Cú pháp sau đây được sử dụng để tạo ra một cột tính.

Cú pháp:

```
CREATE TABLE <table_name> ([<column_name> AS <computed_column_expression>])
```

trong đó:

`table_name`: Chỉ ra tên của bảng.

`column_name AS computed_column_expression`: Chỉ ra tên của cột tính cũng như biểu thức định nghĩa những giá trị trong cột này.

Đoạn mã 18 tạo ra cột tính **Area** với giá trị của nó có được tính toán từ các giá trị nhập vào các trường **Length** và **Breadth**.

Đoạn mã 18:

```
USE SampleDB
CREATE TABLE Calc_Area (Length int, Breadth int, Area AS Length*Breadth)
GO
```

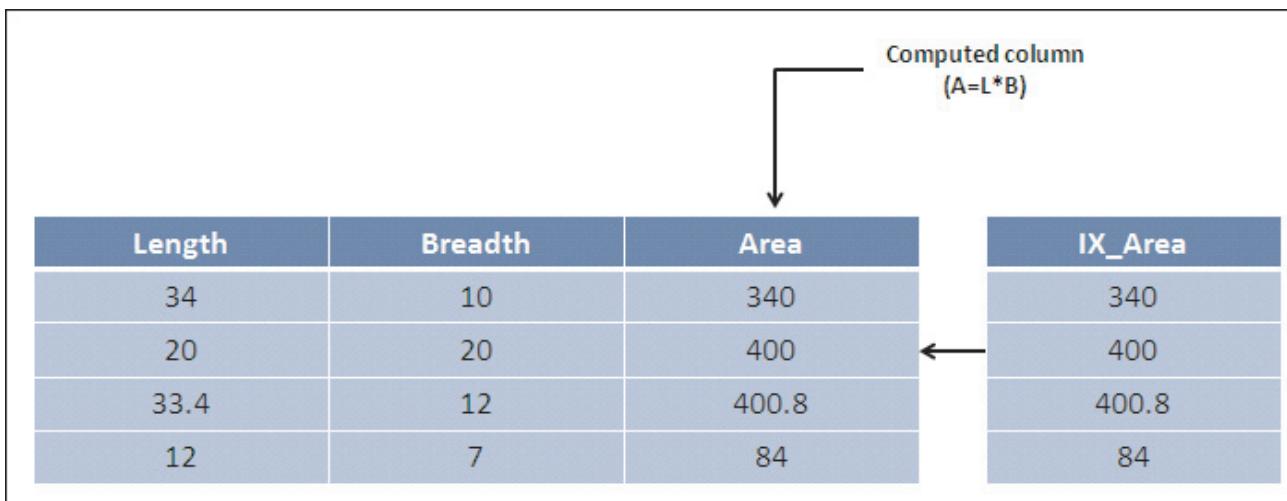
Ghi chú - Không thể sử dụng một cột tính làm một phần của bất kỳ định nghĩa ràng buộc PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, hay CHECK.

11.5.5 Tạo chỉ mục trên các cột tính

Có thể tạo ra chỉ mục trên một cột tính nếu cột đó được đánh dấu PERSISTED. Điều này đảm bảo rằng Công cụ cơ sở dữ liệu lưu trữ các giá trị tính toán trong bảng. Những giá trị này được cập nhật khi có bất kỳ cột nào khác nơi cột tính phụ thuộc vào đó sẽ được cập nhật.

Công cụ cơ sở dữ liệu sử dụng giá trị tiếp tục này khi nó tạo ra một chỉ mục trên cột.

Chỉ mục được tạo ra trên cột tính bằng cách sử dụng câu lệnh CREATE INDEX như được trình bày trong hình 11.28.



Hình 11.28: 11.5.5 Tạo chỉ mục trên các cột tính

Cú pháp sau đây tạo ra một chỉ mục trên cột tính.

Cú pháp:

```
CREATE INDEX <index_name> ON <table_name> (<computed_column_name>)
```

trong đó:

`computed_column_name` chỉ ra tên của cột tính.

Đoạn mã 19 tạo ra một chỉ mục **IX_Area** trên cột tính **Area**.

Đoạn mã 19:

```
USE SampleDB
CREATE INDEX IX_Area ON Calc_Area (Area) ;
GO
```

11.6 Các con trỏ

Một đối tượng cơ sở dữ liệu được sử dụng để lấy dữ liệu mỗi lần một hàng, từ một tập kết quả được gọi là con trỏ. Con trỏ được sử dụng thay vì các lệnh Transact-SQL hoạt động trên tất cả các hàng cùng một lúc trong tập kết quả. Con trỏ được sử dụng khi các bản ghi trong bảng cơ sở dữ liệu cần phải được cập nhật một hàng mỗi lần.

Các loại con trỏ

- ➔ **Con trỏ tĩnh** – Những con trỏ này giúp chuyển đến tập kết quả khi con trỏ được tạo ra và kết quả truy vấn được lưu trữ. Này là cái chậm nhất trong tất cả các con trỏ. Con trỏ này có thể di chuyển/cuộn theo cả hai hướng tiến lùi. Ngoài ra, con trỏ tĩnh không thể được cập nhật hoặc xóa.
- ➔ **Con trỏ động** – Những con trỏ này cho phép bạn xem các thủ tục chèn, cập nhật và xóa khi con trỏ được mở ra. Đây là một trong những con trỏ nhạy cảm nhất và có thể cuộn.
- ➔ **Con trỏ chỉ chuyển tiếp** – Những con trỏ này còn hỗ trợ cập nhật và xóa dữ liệu. Đây là con trỏ nhanh nhất mặc dù nó không hỗ trợ cuộn về phía sau. The three types of forward cursors are FORWARD _ ONLY KEYSET, FORWARD _ ONLY STATIC, và FAST _ FORWARD.
- ➔ **Con trỏ hướng tập khóa** – Những con trỏ này tạo ra một tập hợp các mã định danh duy nhất làm các khóa trong một tập khóa được sử dụng để điều khiển con trỏ. Đây còn là một con trỏ nhạy cảm có thể cập nhật và xóa dữ liệu. Tập khóa phụ thuộc vào tất cả các hàng xác định câu lệnh SELECT tại thời điểm mở con trỏ.

Sau đây là cú pháp để khai báo con trỏ.

Cú pháp:

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR select_statement
```

```
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[ ; ]
```

trong đó:

cursor_name: là tên của con trỏ đã định nghĩa, **cursor_name** phải tuân thủ những quy tắc cho mã định danh.

LOCAL: chỉ ra rằng con trỏ có thể được sử dụng cục bộ cho khối lệnh, thủ tục lưu trữ, hoặc trigger trong đó con trỏ được tạo ra.

GLOBAL: chỉ ra rằng con trỏ có thể được sử dụng trên toàn cầu cho kết nối đó.

FORWARD_ONLY: chỉ ra rằng con trỏ chỉ có thể được cuộn từ hàng đầu tiên đến hàng cuối cùng.

STATIC: định nghĩa một con trỏ tạo ra một bản sao tạm thời của dữ liệu để con trỏ sử dụng.

KEYSET: chỉ ra rằng tư cách thành viên và thứ tự của các hàng trong con trỏ là cố định khi con trỏ được mở ra.

DYNAMIC: định nghĩa một con trỏ phản ánh tất cả thay đổi dữ liệu được thực hiện cho những hàng trong tập kết quả của nó khi bạn cuộn xung quanh con trỏ.

FAST_FORWARD: chỉ ra **FORWARD_ONLY**, con trỏ **READ_ONLY** với tối ưu hóa hiệu suất đã cho phép.

READ_ONLY: ngăn chặn các lần cập nhật được thực hiện thông qua con trỏ này.

SCROLL_LOCKS: chỉ ra rằng các lần cập nhật hoặc xóa đã định vị đã thực hiện thông qua con trỏ được đảm bảo sẽ thành công.

Đoạn mã 20 tạo ra bảng **Employee** trong cơ sở dữ liệu **SampleDB**.

Đoạn mã 20:

```
USE SampleDB
CREATE TABLE Employee
(
    EmpID int PRIMARY KEY,
    EmpName varchar (50) NOT NULL,
    Salary int NOT NULL,
    Address varchar (200) NOT NULL,
)
GO
```

```

INSERT INTO Employee (EmpID, EmpName, Salary, Address) VALUES (1, 'Derek', 12000, 'Houston')

INSERT INTO Employee (EmpID, EmpName, Salary, Address)
VALUES (2, 'David', 25000, 'Texas')

INSERT INTO Employee (EmpID, EmpName, Salary, Address) VALUES (3, 'Alan', 22000, 'New York')

INSERT INTO Employee (EmpID, EmpName, Salary, Address)
VALUES (4, 'Mathew', 22000, 'Las Vegas')

INSERT INTO Employee (EmpID, EmpName, Salary, Address) VALUES (5, 'Joseph', 28000, 'Chicago')

GO

SELECT * FROM Employee

```

Hình 11.29 trình bày kết quả của đoạn mã.

	EmpId	FirstName	Salary	City
1	1	Derek	12000	Houston
2	2	David	25000	Texas
3	3	Alan	22000	New York
4	4	Mathew	22000	Las Vegas
5	5	Joseph	28000	Chicago

Hình 11.29: Bảng Employee được tạo ra trong cơ sở dữ liệu SampleDB

Đoạn mã 21 trình bày cách khai báo con trỏ trên bảng **Employee**.

Đoạn mã 21:

```

USE SampleDB

SET NOCOUNT ON

DECLARE @Id int
DECLARE @name varchar(50)
DECLARE @salary int
/* Con trỏ được khai báo bằng cách định nghĩa câu lệnh SQL trả về một tập kết quả.*/
DECLARE cur_emp CURSOR

```

STATIC FOR

```

SELECT EmpID, EmpName, Salary from Employee

/* Con trỏ được mở ra và đặt bằng cách thực hiện câu lệnh SQL được định nghĩa bằng con
trỏ này.*/

OPEN cur_emp

IF @@CURSOR_ROWS > 0

BEGIN

/* Hàng được lấy từ con trỏ từng cái một hoặc trong một khối để làm thao tác dữ liệu*/

FETCH NEXT FROM cur_emp INTO @Id, @name, @salary

WHILE @@Fetch_status = 0

BEGIN

PRINT 'ID : '+convert(varchar(20),@Id) +', Name : '+@name+' , Salary :
'+convert(varchar(20),@salary)

FETCH NEXT FROM cur_emp INTO @Id, @name, @salary

END

END

-- đóng con trỏ một cách rõ ràng

CLOSE cur_emp

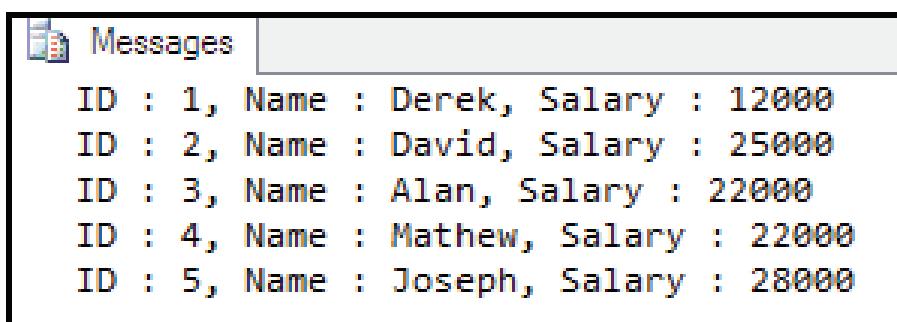
/* Xóa định nghĩa con trỏ và giải phóng tất cả các tài nguyên hệ thống liên quan đến con
trỏ này*/

DEALLOCATE cur_emp

SET NOCOUNT OFF

```

Hình 11.30 trình bày kết quả của đoạn mã.



ID	Name	Salary
1	Derek	12000
2	David	25000
3	Alan	22000
4	Mathew	22000
5	Joseph	28000

Hình 11.30: Con trỏ đã tạo ra trên bảng Employee

Trong đoạn mã này, những chi tiết được lấy mỗi lần một hàng. Thủ tục này sẽ giúp lấy các cơ sở dữ liệu lớn theo tuần tự.

Đầu tiên, con trỏ được khai báo bằng cách định nghĩa câu lệnh SQL trả về một tập kết quả. Sau đó, nó được mở ra và đặt bằng cách thực hiện câu lệnh SQL được định nghĩa bằng con trỏ này. Sau đó các hàng được lấy từ con trỏ từng cái một hoặc trong một khối để thực hiện thao tác dữ liệu.

Con trỏ sau đó được đóng và cuối cùng, định nghĩa con trỏ sẽ bị xóa và tất cả các tài nguyên hệ thống liên quan đến con trỏ này được giải phóng.

11.7 Kiểm tra tiến bộ của bạn

1. Điều nào sau đây là mã đúng để định nghĩa một chỉ mục?

(A)	USE CUST_DB CREATE INDEX IX_CountryCustomer_Details(Country); GO	(C)	USE CUST_DB CREATE INDEX IX_CountryWithCustomer_Details(Country); GO
(B)	USE CUST_DB CREATE INDEX IX_Country FROM Customer_Details(Country); GO	(D)	USE CUST_DB CREATE INDEX IX_Country ON Customer_Details(Country); GO

2. SQL Server 2012 lưu trữ dữ liệu trong các đơn vị lưu trữ được gọi là _____.

(A)	các trang dữ liệu	(C)	các bản ghi
(B)	các biểu mẫu	(D)	các cột

3. Đoạn mã nào sau đây di chuyển dữ liệu trong bảng **Supplier_Details** kết quả tới vị trí mặc định?

(A)	DROP INDEX IX_SuppID ON Supplier_Details	(C)	DROP INDEX IX_SuppID ON Supplier_Details WITH (MOVE TO 'default')
(B)	MOVE INDEX IX_SuppID ON Supplier_Details WITH ('default')	(D)	DELETE INDEX IX_SuppID ON Supplier_Details WITH ('default')

4. Mã nào sau đây được sử dụng để tạo ra INDEX ghép cụm trên **CustID** trong bảng **Customer_Details**?

(A)	USE CUST_DB CREATE CLUSTERED INDEX Customer_Details(CustID) GO	(C)	USE CUST_DB CREATE INDEX Customer_Details ON IX_CustID GO
(B)	USE CUST_DB CREATE INDEX IX_CustID ON Customer_Details GO	(D)	USE CUST_DB CREATE CLUSTERED INDEX IX_CustID ON Customer_Details(CustID) GO

5. Chỉ mục ghép cụm có thể được tạo ra trên một bảng sử dụng một cột không có các giá trị _____.

(A)	tương tự	(C)	trong suốt
(B)	trùng lặp	(D)	bất kỳ

11.7.1 Câu trả lời

1.	D
2.	A
3.	C
4.	D
5.	B



Tóm tắt

- ➔ Các chỉ mục tăng tốc độ của quá trình truy vấn bằng cách cung cấp khả năng truy cập nhanh tới các hàng hoặc cột trong một bảng dữ liệu.
- ➔ SQL Server 2012 lưu trữ dữ liệu trong các đơn vị lưu trữ được gọi là các trang dữ liệu.
- ➔ Tất cả các phép tính đầu vào và đầu ra trong cơ sở dữ liệu này được thực hiện ở mức trang.
- ➔ SQL Server sử dụng các khung nhìn danh mục để tìm các hàng khi chỉ mục không được tạo ra trên một bảng.
- ➔ Chỉ mục ghép cụm làm cho các bản ghi được lưu trữ vật lý theo thứ tự sắp xếp hoặc tuần tự.
- ➔ Chỉ mục không ghép cụm được định nghĩa trên bảng có dữ liệu trong cấu trúc ghép cụm hoặc heap.
- ➔ Các chỉ mục XML có thể tăng tốc các truy vấn trên các bảng có dữ liệu XML.
- ➔ Chỉ mục lưu trữ cột giúp tăng cường hiệu suất của các truy vấn kho dữ liệu một cách bao quát.

Hãy thử tự làm

1. Thư viện Tiểu bang Houston là một trong những thư viện nổi tiếng tại Houston, Texas. Thư viện có một lượng lưu kho khoảng 1000000 cuốn sách thuộc các thể loại khác nhau. Thư viện phát hành sách cho học viên của các trường đại học gần đó. Với luồng vào của các học viên đến thư viện ngày càng tăng theo cấp số nhân, Thư viện Tiểu bang Houston đã quyết định để tự động hóa toàn bộ quá trình phát hành sách cho học viên. Thư viện đã tăng số lượng từng đầu sách đến 10 bản, tùy thuộc vào nhu cầu từ các học viên.

- Hãy tạo ra một cơ sở dữ liệu có tên là **HoustonStateLibrary** để lưu trữ các chi tiết về sách của Thư viện.
- Tạo ra một bảng có tên là **BooksMaster** để lưu trữ các chi tiết về sách trong thư viện như được trình bày trong bảng 11.3.

Tên trường	Loại dữ liệu	Trường khóa	Mô tả
BookCode	varchar(50)	Primary Key	Lưu trữ mã sách của cuốn sách
Title	varchar(max)		Lưu trữ tiêu đề cuốn sách
ISBN	varchar(50)		Lưu trữ ISBN của cuốn sách
Author	char(30)		Lưu trữ tên tác giả của cuốn sách
Price	money		Lưu trữ giá của cuốn sách
Publisher	char(30)		Lưu trữ tên nhà xuất bản của cuốn sách
NumPages	numeric(10,0)		Cửa hàng số lượng trang trong cuốn sách

Bảng 11.3: Bảng BooksMaster

- Tạo ra một chỉ mục ghép cụm có tên là **IX_Title** trên cột **Title** trong bảng **BooksMaster**.
- Tạo bảng **BooksMaster1** có các tên trường **BookCode**, **Title**, và **Book Details**. Chỉ ra kiểu dữ liệu cho **BookDetails** là **xml**. Tạo ra một tài liệu XML với các chi tiết về **ISBN**, **Author**, **Price**, **Publisher**, và **NumPages**.
- Thư viện muốn lấy tên nhà xuất bản của công ty in một cuốn sách của tác giả cụ thể. Tạo ra chỉ mục XML chính **PXML_Books** trên cột **BookCode** của bảng **BooksMaster**.

“

**Người khôn học từ sai lầm của người khác,
kẻ ngốc học từ của chính mình**

”

Phần - 12

Các trigger

Chào mừng bạn đến với phần **Các trigger**.

Phần này giải thích những trigger và các loại trigger khác nhau. Phần này còn mô tả thủ tục để tạo ra và thay đổi các trigger DML, trigger lồng nhau, các hàm cập nhật, xử lý nhiều hàng trong một phiên, và hàm ý hiệu suất của trigger.

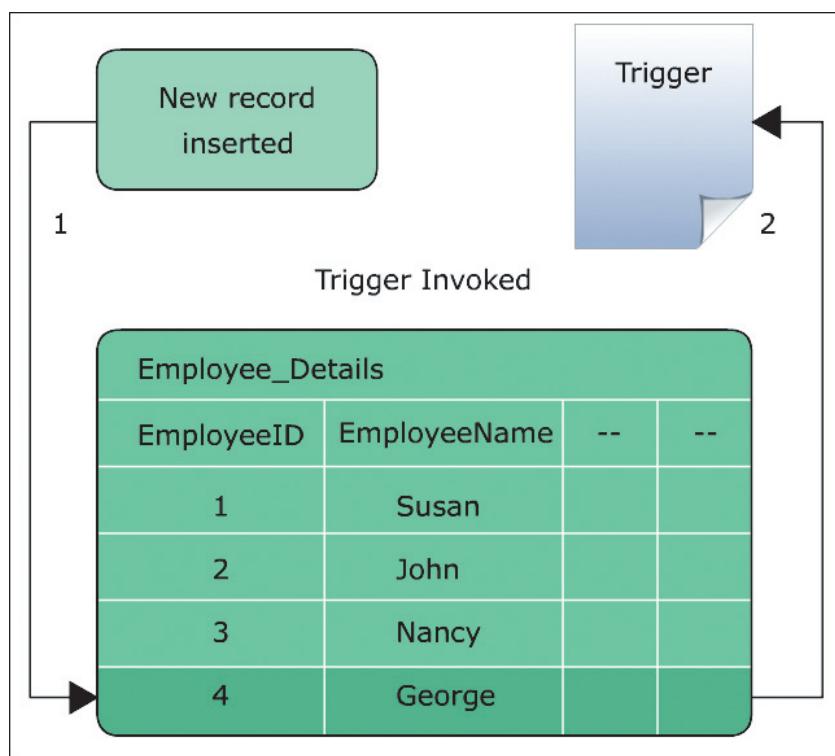
Trong phần này, bạn sẽ học để:

- ➔ Giải thích về trigger
- ➔ Giải thích các loại trigger khác nhau
- ➔ Giải thích thủ tục để tạo ra trigger DML
- ➔ Giải thích thủ tục để thay đổi trigger DML
- ➔ Mô tả các trigger lồng nhau
- ➔ Mô tả các hàm cập nhật
- ➔ Giải thích việc xử lý nhiều hàng trong một phiên
- ➔ Giải thích hàm ý hiệu suất của trigger

12.1 Giới thiệu

Trigger là một thủ tục lưu trữ được thực hiện khi có một nỗ lực để sửa đổi dữ liệu trong bảng được bảo vệ bằng trigger. Không giống như các thủ tục lưu trữ hệ thống tiêu chuẩn, không thể thực thi các trigger trực tiếp, chúng cũng không truyền hoặc nhận các tham số. Gây nên được định nghĩa trên các bảng cụ thể và những bảng này được gọi là bảng trigger.

Nếu trigger được định nghĩa trên hành động INSERT, UPDATE, hoặc DELETE trên một bảng, nó tự hủy khi những hành động này được thử. Sự thực thi tự động này của trigger không thể bị phá vỡ. Trong SQL Server 2012, trigger được tạo ra bằng cách sử dụng câu lệnh CREATE TRIGGER. Hình 12.1 hiển thị ví dụ về trigger.



Hình 12.1: Các trigger

12.2 Cách dùng các trigger

Trigger có thể chứa logic xử lý phức tạp và thường được sử dụng để duy trì tính toàn vẹn dữ liệu ở mức độ thấp. Cách dùng chính của trigger có thể được phân loại như sau:

→ Phân tầng thay đổi thông qua các bảng liên quan

Người dùng có thể sử dụng một trigger cho các thay đổi tầng thông qua các bảng liên quan. Ví dụ, hãy xem xét bảng **Salary_Details** có **FOREIGN KEY**, **EmpID** tham chiếu tới **PRIMARY KEY**, **EmpID** của bảng **Employee_Details**. Nếu sự kiện cập nhật hoặc xóa xảy ra trong bảng **Employee_Details**, trigger cập nhật hoặc xóa có thể được định nghĩa để xếp tầng những thay đổi đối với bảng **Salary_Details**.

→ **Bắt buộc tính toàn vẹn dữ liệu phức tạp hơn các ràng buộc CHECK**

Không giống như các hạn chế CHECK, trigger có thể tham khảo các cột trong các bảng khác. Tính năng này có thể được sử dụng để áp dụng các kiểm tra tính toàn vẹn dữ liệu phức tạp. Tính toàn vẹn dữ liệu có thể được thực thi bằng cách:

- Kiểm tra các ràng buộc trước khi xếp tầng các cập nhật hoặc xóa.
- Tạo ra các trigger nhiều hàng cho các hành động được thực hiện trên nhiều hàng.
- Thực thi tính toàn vẹn tham chiếu giữa các cơ sở dữ liệu.

→ **Định nghĩa các thông báo lỗi tùy chỉnh**

Các thông báo lỗi tùy chỉnh được sử dụng để cung cấp lời giải thích phù hợp hoặc chi tiết hơn trong các tình huống lỗi nhất định. Có thể sử dụng các trigger để gọi các thông báo lỗi tùy chỉnh được xác định trước như vậy khi các điều kiện lỗi có liên quan xảy ra.

→ **Duy trì dữ liệu khử tiêu chuẩn hóa**

Tính toàn vẹn dữ liệu mức thấp có thể được duy trì trong các môi trường cơ sở dữ liệu khử tiêu chuẩn hóa sử dụng các trigger. Dữ liệu khử tiêu chuẩn hóa thường đề cập đến dữ liệu dư thừa hoặc bắt nguồn. Ở đây, các trigger được sử dụng cho các kiểm tra không yêu cầu các so khớp chính xác. Ví dụ, nếu giá trị của năm sẽ được kiểm tra đối với các ngày hoàn tất, trigger có thể được sử dụng để thực hiện kiểm tra.

→ **So sánh trước và sau các trạng thái của dữ liệu đang được sửa đổi**

Trigger cung cấp tùy chọn này để tham khảo các thay đổi được thực hiện cho dữ liệu bằng các câu lệnh INSERT, UPDATE, và DELETE. Điều này cho phép người dùng tham khảo các hàng bị ảnh hưởng khi các sửa đổi được thực hiện thông qua các trigger.

12.3 Các loại trigger

Trigger có thể được đặt để thực thi tự động một hành động khi một sự kiện ngôn ngữ xảy ra trong một bảng hoặc khung nhìn. Các sự kiện ngôn ngữ có thể được phân loại là các sự kiện DML và các sự kiện DDL. Trigger liên quan đến các sự kiện DML được gọi là trigger DML, trong khi trigger kết hợp với các sự kiện DDL được gọi là trigger DDL.

Trigger trong SQL Server 2012 có thể được phân thành ba loại cơ bản:

→ **Các trigger DML**

Các trigger DML thực thi khi dữ liệu được chèn, sửa đổi hoặc xóa trong một bảng hoặc khung nhìn sử dụng câu lệnh INSERT, UPDATE, hoặc DELETE.

→ **Các trigger DDL**

Các trigger DDL thực thi khi một bảng hoặc khung nhìn được tạo ra, chỉnh sửa hoặc xóa bằng cách sử dụng câu lệnh CREATE, ALTER, hoặc DROP.

→ Các trigger đăng nhập

Các trigger đăng nhập thực thi các thủ tục lưu trữ khi một phiên được thiết lập với một sự kiện LOGON. Những trigger này được gọi sau khi chứng thực đăng nhập được hoàn thành và trước khi phiên thực tế được thiết lập. Các trigger đăng nhập kiểm soát các phiên máy chủ bằng cách hạn chế các thông tin đăng nhập không hợp lệ hoặc hạn chế số lượng phiên.

12.4 Các trigger DDL so với các trigger DML

Các trigger DDL và DML có cách sử dụng khác nhau và được thực thi với các sự kiện cơ sở dữ liệu khác nhau. Bảng 12.1 liệt kê ra những khác biệt giữa các trigger DDL và DML.

Các trigger DDL	Các trigger DML
Các trigger DDL thực thi các thủ tục lưu trữ trên câu lệnh CREATE, ALTER, và DROP.	Các trigger DML thực thi trên các câu lệnh INSERT, UPDATE, và DELETE.
Các trigger DDL được sử dụng để kiểm tra và kiểm soát các hoạt động của cơ sở dữ liệu.	Các trigger DML được sử dụng để thực thi các quy tắc kinh doanh khi dữ liệu được sửa đổi trong các bảng hoặc khung nhìn.
Các trigger DDL chỉ hoạt động sau khi bảng hoặc khung nhìn được sửa đổi.	Các trigger DML thực thi trong khi sửa đổi dữ liệu hoặc sau khi dữ liệu được sửa đổi.
Các trigger DDL được định nghĩa ở mức cơ sở dữ liệu hoặc máy chủ.	Các trigger DML được định nghĩa ở mức cơ sở dữ liệu.

Bảng 12.1: Sự khác biệt giữa các trigger DDL và DML

12.5 Tạo các trigger DML

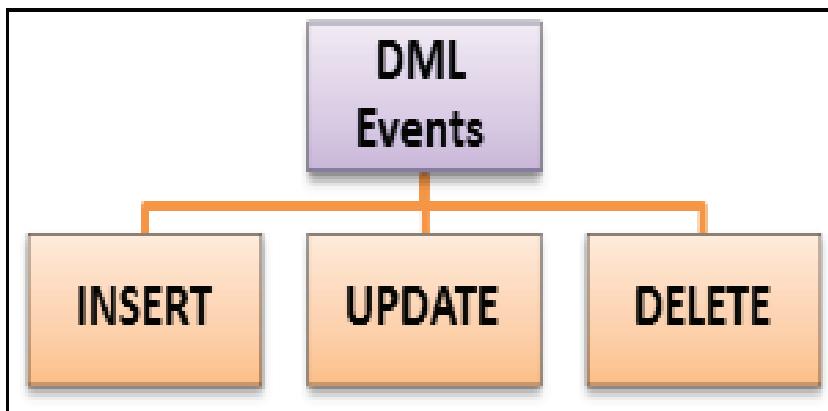
Các trigger DML được thực thi khi sự kiện DML xảy ra trong các bảng hoặc khung nhìn. Những sự kiện DML này bao gồm các câu lệnh INSERT, UPDATE, và DELETE. Các trigger DML có thể thực thi khi hoàn thành các sự kiện DML hoặc ở vị trí của các sự kiện DML.

Các trigger DML bắt buộc tính toàn vẹn tham chiếu bằng cách xếp tầng các thay đổi tới các bảng có liên quan khi một hàng được sửa đổi. Các trigger DML có thể thực hiện nhiều hành động cho mỗi câu lệnh sửa đổi.

Các trigger DML thuộc ba loại chính:

- Trigger INSERT
- Trigger UPDATE
- Trigger DELETE

Hình 12.2 hiển thị các loại trigger DML.



Hình 12.2: Các trigger DML

12.5.1 Giới thiệu về các bảng đã chèn và xóa

Những câu lệnh SQL trong các trigger DML sử dụng hai loại bảng đặc biệt để sửa đổi dữ liệu trong cơ sở dữ liệu. Khi dữ liệu được chèn, cập nhật, hoặc xóa, SQL Server 2012 tạo ra và quản lý những bảng này một cách tự động. Các bảng tạm thời lưu trữ dữ liệu ban đầu cũng như dữ liệu đã sửa đổi. Những bảng này như sau:

→ **Bảng đã chèn**

Bảng `Inserted` chứa bản sao các bản ghi được sửa đổi với hoạt động `INSERT` và `UPDATE` trên bảng trigger. Bảng trigger là bảng trên đó trigger được định nghĩa. Hoạt động `INSERT` và `UPDATE` chèn các bản ghi mới vào bảng `Inserted` và Trigger.

→ **Bảng đã xóa**

Bảng `Deleted` chứa bản sao các bản ghi được sửa đổi với hoạt động `DELETE` và `UPDATE` trên bảng trigger. Bảng trigger là bảng trên đó trigger được định nghĩa. Những hoạt động này xóa các bản ghi từ bảng trigger và chèn chúng vào bảng `Deleted`.

Ghi chú - Bảng `Inserted` và `Deleted` về khía cạnh vật lý không thể giữ hiện diện trong cơ sở dữ liệu. Chúng được tạo ra và thả bỏ bất cứ khi nào bất kỳ sự kiện kích hoạt nào xảy ra.

12.5.2 Chèn các trigger

Trigger `INSERT` được thực thi khi một bản ghi mới được chèn vào bảng. Trigger `INSERT` đảm bảo rằng giá trị đang được nhập vào phù hợp với các ràng buộc được định nghĩa trên bảng đó.

Khi người dùng chèn một bản ghi vào bảng đó, trigger `INSERT` lưu một bản sao của bản ghi đó vào bảng `Inserted`. Sau đó nó sẽ kiểm tra xem giá trị mới trong bảng `Inserted` phù hợp với các ràng buộc đã chỉ định.

Nếu bản ghi hợp lệ, trigger `INSERT` chèn hàng này vào bảng trigger, nếu không nó sẽ hiển thị thông báo lỗi.

Trigger `INSERT` được tạo ra sử dụng từ khóa `INSERT` trong câu lệnh `CREATE TRIGGER` và `ALTER TRIGGER`.

Sau đây là cú pháp cho việc tạo ra trigger `INSERT`.

Cú pháp:

```
CREATE TRIGGER [schema_name.] trigger_name
ON [schema_name.] table_name
[WITH ENCRYPTION]
{ FOR INSERT }
AS
[ IF UPDATE (column_name) ... ]
[ { AND | OR } UPDATE (column_name) ... ]
<sql_statements>
```

trong đó:

`schema_name`: chỉ ra tên của lược đồ có chứa bảng/trigger đó.

`trigger_name`: chỉ ra tên của trigger.

`table_name`: chỉ ra bảng mà trên đó trigger DML được tạo ra.

`WITH ENCRYPTION`: mã hóa văn bản của câu lệnh `CREATE TRIGGER`.

`FOR`: chỉ ra rằng trigger DML thực thi sau khi các hoạt động sửa đổi được hoàn tất.

`INSERT`: chỉ ra rằng trigger DML này sẽ được gọi bởi các hoạt động chèn.

`UPDATE`: Trả về giá trị Boolean cho biết xem lần thử `INSERT` hoặc `UPDATE` đã được thực hiện hay chưa trên một cột cụ thể.

`column_name`: Là tên của cột để kiểm tra về hành động `UPDATE`.

`AND`: Kết hợp hai biểu thức Boolean và trả về `TRUE` khi cả hai biểu thức là `TRUE`.

`OR`: Kết hợp hai biểu thức Boolean và trả về `TRUE` nếu ít nhất một biểu thức là `TRUE`.

`sql_statement`: chỉ ra các câu lệnh SQL được thực thi trong trigger DML.

Đoạn mã 1 tạo ra trigger `INSERT` trên bảng có tên là `Account_Transactions`. Khi một bản ghi mới được đưa vào, và nếu số tiền rút vượt quá `80000`, trigger chèn sẽ hiển thị thông báo lỗi và quay lui giao tác sử dụng câu lệnh `ROLLBACK TRANSACTION`.

Giao tác là tập hợp một hoặc nhiều câu lệnh được xử lý như một đơn vị công việc. Giao tác có thể thành công hoặc thất bại thành một khối, do đó tất cả các câu lệnh trong đó cùng thành công hay thất bại. Câu lệnh `ROLLBACK TRANSACTION` hủy bỏ hoặc quay lui một giao tác.

Đoạn mã 1:

```
CREATE TRIGGER CheckWithdrawal_Amount
ON Account_Transactions
FOR INSERT
AS
IF (SELECT Withdrawal From inserted) > 80000
BEGIN
PRINT 'Số tiền rút không thể vượt quá 80000'
ROLLBACK TRANSACTION
END
```

Đoạn mã 2 chèn một bản ghi nơi `Số tiền rút` vượt quá `80000`. Điều này làm cho trigger `INSERT` hiển thị thông báo lỗi và quay lui giao tác đó.

Đoạn mã 2:

```
INSERT INTO Account_Transactions
(TransactionID, EmployeeID, CustomerID, TransactionTypeID, TransactionDate,
TransactionNumber, Deposit, Withdrawal)
VALUES
(1008, 'E08', 'C08', 'T08', '05/02/12', 'TN08', 300000, 90000)
```

Thông báo lỗi sau được hiển thị như được chỉ ra bằng câu lệnh `PRINT`:

Số tiền rút không thể vượt quá 80000.

12.5.3 Cập nhật các trigger

Trigger `UPDATE` sao chép bản ghi gốc vào bảng `Deleted` và bản ghi mới vào bảng `Inserted` khi bản ghi được cập nhật. Sau đó nó đánh giá bản ghi mới để xác định xem các giá trị này có phù hợp với các ràng buộc đã chỉ định trong bảng trigger hay không.

Nếu các giá trị mới là hợp lệ, bản ghi từ bảng `Inserted` được sao chép vào bảng trigger. Tuy nhiên, nếu các giá trị mới là không hợp lệ, thông báo lỗi được hiển thị. Ngoài ra, bản ghi gốc được sao chép từ bảng `Deleted` trở lại vào bảng trigger.

Trigger UPDATE được tạo ra sử dụng từ khóa UPDATE trong câu lệnh CREATE TRIGGER và ALTER TRIGGER . Sau đây là cú pháp cho việc tạo ra trigger UPDATE ở mức bảng.

Cú pháp:

```
CREATE TRIGGER [schema_name.] trigger_name
ON [schema_name.] table_name
[WITH ENCRYPTION]
{ FOR UPDATE }
AS
[ IF UPDATE (column_name) ... ]
[ {AND | OR} UPDATE (column_name) ... ]
<sql_statements>
```

trong đó:

FOR UPDATE: chỉ ra rằng trigger DML này sẽ được gọi sau các hoạt động cập nhật.

Đoạn mã 3 tạo ra trigger UPDATE ở mức bảng trên bảng EmployeeDetails. Khi một bản ghi được sửa đổi, trigger UPDATE được kích hoạt. Nó sẽ kiểm tra xem ngày tháng năm sinh có lớn hơn ngày hôm nay không. Nếu có, nó sẽ hiển thị thông báo lỗi cho các giá trị không hợp lệ và quay lui hoạt động sửa đổi bằng cách sử dụng câu lệnh ROLLBACK TRANSACTION.

Đoạn mã 3:

```
CREATE TRIGGER CheckBirthDate
ON EmployeeDetails
FOR UPDATE
AS
IF (SELECT BirthDate From inserted) > getDate()
BEGIN
PRINT 'Ngày tháng năm sinh không thể lớn hơn ngày hôm nay'
ROLLBACK
END
```

Đoạn mã 4 cập nhật một bản ghi trong đó ngày tháng năm sinh không hợp lệ được chỉ định. Điều này làm cho trigger cập nhật hiển thị thông báo lỗi và quay lui giao tác đó.

Đoạn mã 4:

```
UPDATE EmployeeDetails
SET BirthDate='2015/06/02'
WHERE EmployeeID='E06'
```

Thông báo lỗi sau được hiển thị như được chỉ ra bằng câu lệnh PRINT:

Ngày tháng năm sinh không thể lớn hơn ngày hôm nay.

→ Tạo các trigger cập nhật

Những trigger UPDATE được tạo ra ở mức cột hoặc ở mức bảng. Những trigger ở mức cột thực thi khi các lần cập nhật được thực hiện trong cột chỉ định. Những trigger ở mức bảng thực thi khi các lần cập nhật được thực hiện bất cứ đâu trong toàn bộ bảng.

Để tạo ra một trigger UPDATE ở mức cột, hàm UPDATE () được sử dụng để chỉ ra cột này.

Đoạn mã 5 tạo ra trigger UPDATE ở mức cột trên cột **EmployeeID** của bảng **EmployeeDetails**. Khi **EmployeeID** được sửa đổi, trigger UPDATE được kích hoạt và thông báo lỗi được hiển thị. Hoạt động sửa đổi được quay lui bằng cách sử dụng câu lệnh ROLLBACK TRANSACTION.

Đoạn mã 5:

```
CREATE TRIGGER Check_EmployeeID
ON EmployeeDetails
FOR UPDATE
AS
IF UPDATE (EmployeeID)
BEGIN
PRINT 'Bạn không thể thay đổi ID của nhân viên'
ROLLBACK TRANSACTION
END
```

Đoạn mã 6 cập nhật một bản ghi trong đó giá trị trong cột EmployeeID đang được sửa đổi. Điều này làm cho trigger cập nhật hoạt động. Trigger cập nhật hiển thị thông báo lỗi và quay lui giao tác đó.

Đoạn mã 6:

```
UPDATE EmployeeDetails
SET EmployeeID='E12'
WHERE EmployeeID='E04'
```

12.5.4 Xóa các trigger

Có thể tạo ra trigger DELETE để hạn chế người dùng không xoá một bản ghi cụ thể trong bảng. Điều sau đây sẽ xảy ra nếu người dùng cố gắng xóa bản ghi:

- ➔ Bản ghi bị xóa khỏi bảng trigger và chèn vào bảng Deleted.
- ➔ Nó được kiểm tra về các ràng buộc đối với việc xóa.
- ➔ Nếu có ràng buộc trên bản ghi để ngăn chặn việc xóa, trigger DELETE sẽ hiển thị thông báo lỗi.
- ➔ Bản ghi đã xóa được lưu trữ trong bảng Deleted được sao chép ngược lại bảng trigger.

Trigger DELETE được tạo ra bằng cách sử dụng từ khóa DELETE trong câu lệnh CREATE TRIGGER. Sau đây là cú pháp cho việc tạo ra một trigger DELETE.

Cú pháp:

```
CREATE TRIGGER <trigger_name>
ON <table_name>
[WITH ENCRYPTION]
FOR DELETE
AS <sql_statement>
```

trong đó:

DELETE: chỉ ra rằng trigger DML này sẽ được gọi bởi các hoạt động xóa.

Đoạn mã 7 tạo ra một trigger DELETE trên bảng **Account_Transactions**. Nếu bản ghi của một **TransactionID** bị xóa, trigger DELETE được kích hoạt và thông báo lỗi được hiển thị. Hoạt động xóa được quay lui sử dụng câu lệnh ROLLBACK TRANSACTION.

Đoạn mã 7:

```
CREATE TRIGGER CheckTransactions
ON Account_Transactions
FOR DELETE
AS
IF 'T01' IN (SELECT TransactionID FROM deleted)
BEGIN
PRINT 'Người dùng không thể xóa những giao tác này.'
ROLLBACK TRANSACTION
END
```

Đoạn mã 8 thử xóa các bản ghi từ bảng **Account_Transactions** trong đó **Deposit** là 50000.

Đoạn mã 8:

```
DELETE FROM Account_Transactions
WHERE Deposit= 50000
```

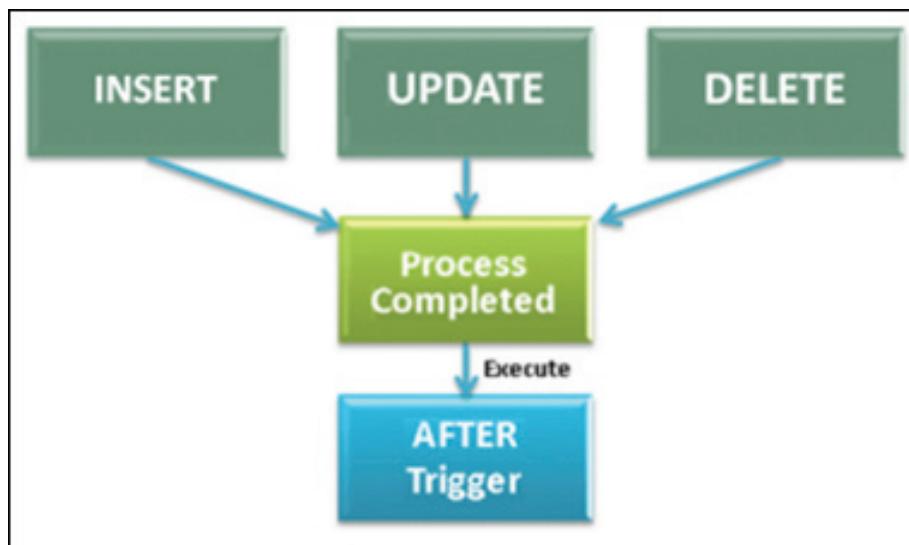
Thông báo lỗi được hiển thị như được chỉ ra bằng câu lệnh PRINT.

Người dùng không thể xóa những giao tác này.

12.6 Các trigger AFTER

Trigger AFTER được thực thi khi hoàn thành các hoạt động INSERT, UPDATE, hoặc DELETE. Trigger AFTER có thể được tạo ra chỉ trên các bảng. Bảng có thể có nhiều trigger AFTER được định nghĩa cho mỗi hoạt động INSERT, UPDATE, và DELETE. Nếu nhiều trigger AFTER được tạo ra trên cùng một bảng, người dùng phải định nghĩa thứ tự theo đó những trigger này phải được thực thi. Trigger AFTER được thực thi khi kiểm tra ràng buộc trong bảng đã hoàn tất. Ngoài ra, trigger được thực thi sau khi các bảng Inserted và Deleted được tạo ra.

Hình 12.3 hiển thị các loại trigger AFTER.



Hình 12.3: Các trigger AFTER

Sau đây là cú pháp cho việc tạo ra trigger AFTER.

Cú pháp:

```

CREATE TRIGGER <trigger_name>
ON <table_name>
[WITH ENCRYPTION]
{ FOR | AFTER }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS <sql_statement>
  
```

trong đó:

FOR | AFTER: chỉ ra rằng trigger DML thực thi sau khi các hoạt động sửa đổi được hoàn tất.

{ [INSERT] [,] [UPDATE] [,] [DELETE] }: chỉ ra những hoạt động gọi trigger DML.

Đoạn mã 9 tạo ra một trigger AFTER DELETE trên bảng **EmployeeDetails**. Nếu có bất kỳ bản ghi nhân viên nào bị xóa khỏi bảng, trigger AFTER DELETE được kích hoạt. Trigger sẽ hiển thị số lượng bản ghi nhân viên đã xóa khỏi bảng này.

Đoạn mã 9:

```
CREATE TRIGGER Employee_Deletion
ON EmployeeDetails
AFTER DELETE
AS
BEGIN
DECLARE @num nchar;
SELECT @num = COUNT(*) FROM deleted
PRINT 'No. of employees deleted= ' + @num
END
```

Đoạn mã 10 xóa một bản ghi khỏi bảng **EmployeeDetails**.

Đoạn mã 10:

```
DELETE FROM EmployeeDetails WHERE EmployeeID='E07'
```

Thông báo lỗi sau được hiển thị:

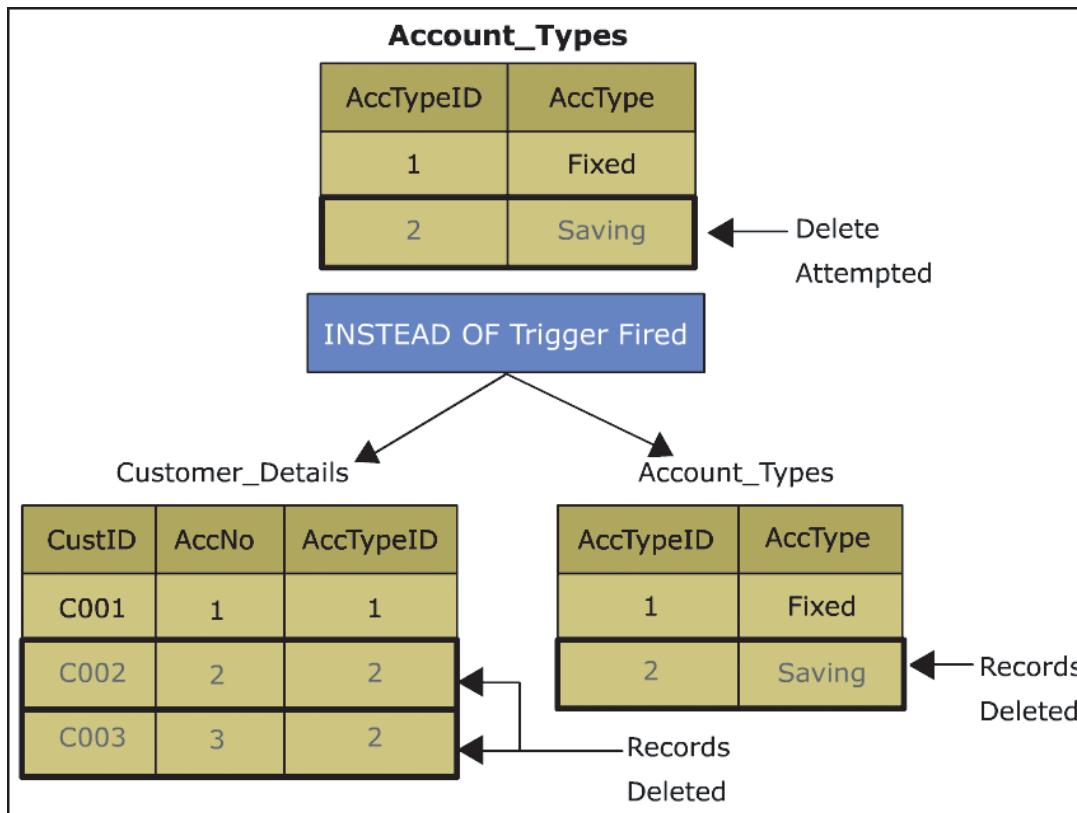
Số lượng nhân viên đã xóa = 0

12.7 Các trigger INSTEAD OF

Trigger **INSTEAD OF** được thực thi thay cho các hoạt động **INSERT**, **UPDATE**, hoặc **DELETE**. Các trigger **INSTEAD OF** có thể được tạo ra trên các bảng cũng như khung nhìn. Bảng hoặc khung nhìn có thể chỉ có một trigger **INSTEAD OF** được định nghĩa cho mỗi hoạt động **INSERT**, **UPDATE**, và **DELETE**.

Các trigger **INSTEAD OF** được thực thi trước khi các lần kiểm tra ràng buộc được thực hiện trên bảng này. Những trigger này được thực thi sau khi việc tạo ra các bảng **Inserted** và **Deleted**. Các trigger **INSTEAD OF** tăng sự đa dạng của các loại cập nhật mà người dùng có thể thực hiện đối với khung nhìn.

Hình 12.4 hiển thị ví dụ về các trigger INSTEAD OF.



Hình 12.4: Các trigger INSTEAD OF

Trong ví dụ được trình bày trong hình 12.4, trigger INSTEAD OF DELETE trên bảng **Account_Types** được tạo ra. Nếu có bất kỳ bản ghi nào trong bảng **Account_Types** bị xóa, những bản ghi tương ứng trong **Customer_Details** cũng sẽ được gỡ bỏ. Do đó, thay vì làm việc chỉ trên một bảng, ở đây, trigger đảm bảo rằng hoạt động xóa được thực hiện trên cả hai bảng.

Ghi chú - Người dùng không thể tạo các trigger INSTEAD OF cho các hoạt động xóa hoặc cập nhật trên các bảng có các tùy chọn tầng ON DELETE và tầng ON UPDATE đã chọn.

Sau đây là cú pháp cho việc tạo ra trigger INSTEAD OF.

Cú pháp:

```
CREATE TRIGGER <trigger_name>
ON { <table_name> | <view_name> }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS <sql_statement>
```

trong đó:

view_name: chỉ ra khung nhìn mà trên đó trigger DML được tạo ra.

INSTEAD OF: chỉ ra rằng trigger DML thực thi thay cho các hoạt động sửa đổi. Những trigger này không được định nghĩa trên các khung nhìn có thể cập nhật sử dụng WITH CHECK OPTION.

Đoạn mã 11 tạo ra trigger INSTEAD OF DELETE trên bảng **Account_Transactions**. Nếu có bất kỳ bản ghi nào trong bảng **Account_Transactions** bị xóa, những bản ghi tương ứng trong bảng **EmployeeDetails** sẽ được gỡ bỏ.

Đoạn mã 11:

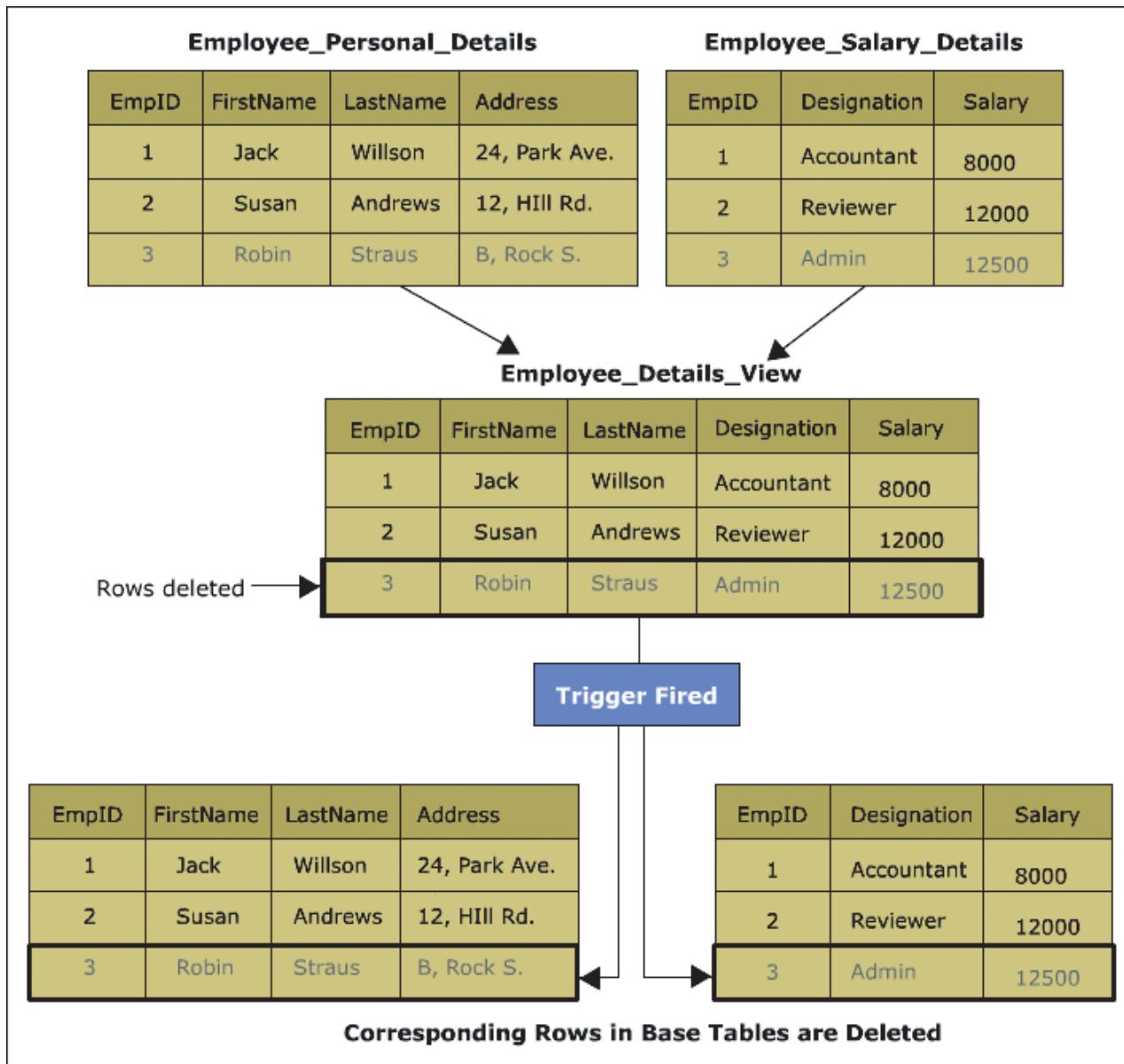
```
CREATE TRIGGER Delete_AccType
ON Account_Transactions
INSTEAD OF DELETE
AS
BEGIN
DELETE FROM EmployeeDetails WHERE EmployeeID IN
(SELECT TransactionTypeID FROM deleted)
DELETE FROM Account_Transactions WHERE TransactionTypeID IN
(SELECT TransactionTypeID FROM deleted)
END
```

12.7.1 Sử dụng các trigger INSTEAD OF với các khung nhìn

Các trigger INSTEAD OF có thể được chỉ ra trên các bảng cũng như khung nhìn. Trigger này thực thi thay vì hành động kích hoạt ban đầu. Các trigger INSTEAD OF cung cấp phạm vi rộng hơn và các loại cập nhật mà người dùng có thể thực hiện đối với khung nhìn. Mỗi bảng hoặc khung nhìn được giới hạn chỉ cho một trigger INSTEAD OF cho mỗi hành động kích hoạt (INSERT, UPDATE, hoặc DELETE).

Người dùng không thể tạo ra một trigger INSTEAD OF trên các khung nhìn có mệnh đề WITH CHECK OPTION đã định nghĩa.

Hình 12.5 hiển thị ví dụ về việc sử dụng các trigger INSTEAD OF với các khung nhìn.



Hình 12.5: Sử dụng các trigger INSTEAD OF với các khung nhìn

Đoạn mã 12 tạo ra một bảng có tên là **Employee_Personal_Details**.

Đoạn mã 12:

```
CREATE TABLE Employee_Personal_Details
(
    EmpID int NOT NULL,
    FirstName varchar(30) NOT NULL,
    LastName varchar(30) NOT NULL,
    Address varchar(30)
)
```

Đoạn mã 13 tạo ra một bảng có tên là **Employee_Salary_Details**.

Đoạn mã 13:

```
CREATE TABLE Employee_Salary_Details
(
    EmpID int NOT NULL,
    Designation varchar(30),
    Salary int NOT NULL
)
```

Đoạn mã 14 tạo ra một khung nhìn **Employee_Details_View** sử dụng các cột từ bảng **Employee_Personal_Details** và **Employee_Salary_Details** bằng cách nối hai bảng trên cột **EmpID**.

Đoạn mã 14:

```
CREATE VIEW Employee_Details_View
AS
SELECT e1.EmpID, FirstName, LastName, Designation, Salary
FROM Employee_Personal_Details e1
JOIN Employee_Salary_Details e2
ON e1.EmpID = e2.EmpID
```

Đoạn mã 15 tạo ra trigger INSTEAD OF DELETE **Delete_Employees** trên khung nhìn **Employee_Details_View**. Khi một hàng được xóa khỏi khung nhìn, trigger được kích hoạt. Nó xóa những bản ghi tương ứng khỏi những bảng cơ sở của khung nhìn có tên là **Employee_Personal_Details** và **Employee_Salary_Details**.

Đoạn mã 15:

```
CREATE TRIGGER Delete_Employees
ON Employee_Details_View
INSTEAD OF DELETE
AS
BEGIN
DELETE FROM Employee_Salary_Details WHERE EmpID IN
(SELECT EmpID FROM deleted)
DELETE FROM Employee_Personal_Details WHERE EmpID IN
(SELECT EmpID FROM deleted)
```

Đoạn mã 16 xóa một hàng khỏi khung nhìn **Employee_Details_View** trong đó **EmpID='2'**.

Đoạn mã 16:

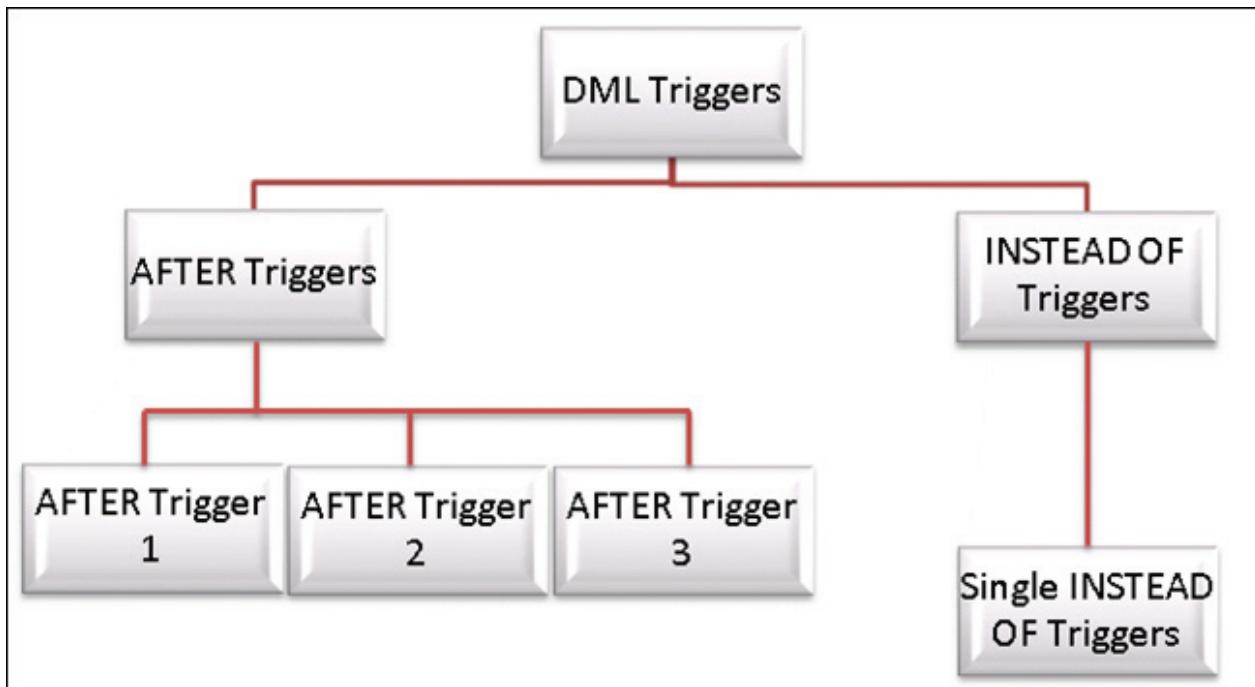
```
DELETE FROM Employee_Details_View WHERE EmpID='3'
```

12.8 Làm việc với các trigger DML

SQL Server 2012 cho phép người dùng tạo ra nhiều trigger AFTER cho mỗi hành động kích hoạt (như là UPDATE, INSERT, và DELETE) trên một bảng. Tuy nhiên, người dùng có thể tạo ra chỉ một trigger INSTEAD OF cho mỗi hành động kích hoạt trên một bảng.

Khi người dùng có nhiều trigger AFTER trên một hành động kích hoạt, tất cả những trigger này phải có một tên khác. Trigger AFTER có thể bao gồm một số câu lệnh SQL thực hiện các hàm khác nhau.

Hình 12.6 hiển thị những loại trigger DML.



Hình 12.6: Các loại trigger DML

Ghi chú - Một trigger AFTER đơn lẻ có thể được gọi bởi nhiều hơn một hành động kích hoạt.

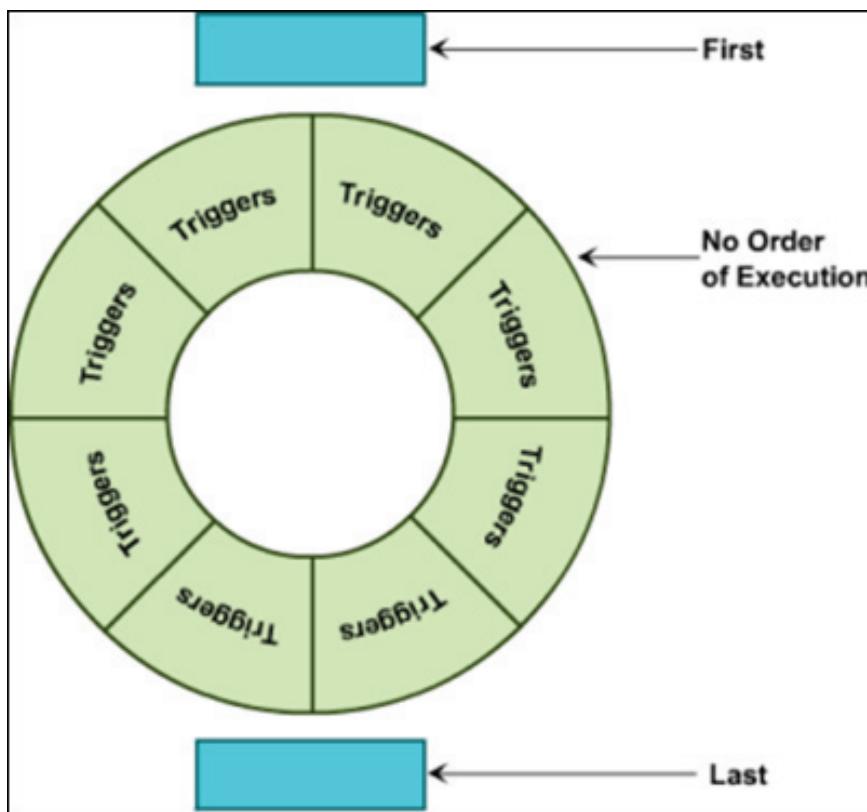
→ Thứ tự thực thi các trigger DML

SQL Server 2012 cho phép người dùng chỉ ra trigger AFTER nào sẽ được thực thi đầu tiên và cái nào sẽ được thực thi cuối cùng. Tất cả các trigger AFTER đã gọi giữa những trigger đầu tiên và cuối cùng không có thứ tự thực thi nhất định.

Tất cả các hành động gây ra có trigger đầu tiên và cuối cùng được định nghĩa cho chúng. Tuy nhiên, không có hai hành động kích hoạt trên một bảng có thể có cùng các trigger đầu tiên và cuối cùng.

Người dùng có thể sử dụng thủ tục lưu trữ `sp_settriggerorder` để định nghĩa thứ tự của các trigger DML AFTER.

Hình 12.7 hiển thị thứ tự thực thi của các trigger DML.



Hình 12.7: Thứ tự thực thi các trigger DML

Sau đây là cú pháp để chỉ ra thứ tự thực thi nhiều trigger AFTER DML.

Cú pháp:

```
sp_settriggerorder [ @triggername = ] '[ triggerschema.] triggername'
, [ @order = ] 'value'
, [ @stmttype = ] 'statement_type'
```

trong đó:

[triggerschema.] triggername: là tên của trigger DML hoặc DDL và lược đồ thuộc về nó và thứ tự của nó cần phải được chỉ ra.

value: chỉ ra thứ tự thực hiện của trigger như FIRST, LAST, hoặc NONE. Nếu FIRST được chỉ ra, sau đó trigger được khai hỏa đầu tiên. Nếu LAST được chỉ ra, trigger được khai hỏa cuối cùng. Nếu NONE được chỉ ra, thứ tự khai hỏa của trigger là không xác định.

statement_type: chỉ ra loại câu lệnh SQL (INSERT, UPDATE, hoặc DELETE) gọi trigger DML.

Đoạn mã 17 đầu tiên thực thi trigger **Employee_Deletion** đã định nghĩa trên bảng này khi hoạt động DELETE được thực hiện trên cột **withdrawal** của bảng này.

Đoạn mã 17:

```
EXEC sp_settriggerorder @triggername = 'Employee_Deletion', @order =  
'FIRST', @stmttype = 'DELETE'
```

12.8.1 Xem định nghĩa về các trigger DML

Định nghĩa trigger bao gồm tên trigger, bảng trên đó trigger được tạo ra, các hành động kích hoạt, và các câu lệnh SQL được thực thi. SQL Server 2012 cung cấp thủ tục lưu trữ **sp_helptrigger** để lấy các định nghĩa trigger.

Tên trigger DML phải được chỉ ra như là tham số khi thực thi **sp_helptrigger**. Hình 12.8 hiển thị định nghĩa các trigger DML.

Ghi chú - Không thể xem định nghĩa trigger nếu định nghĩa được mã hóa.

Sau đây là cú pháp cho việc xem trigger DML.

Cú pháp:

```
sp_helptrigger '<DML_trigger_name>'
```

trong đó:

DML_trigger_name: chỉ ra tên của trigger DML định nghĩa của nó sẽ được hiển thị.

Đoạn mã 18 hiển thị những định nghĩa của trigger **Employee_Deletion**, đã tạo ra trên bảng này.

Đoạn mã 18:

```
sp_helptrigger 'Employee_Deletion'
```

12.8.2 Sửa đổi định nghĩa về các trigger DML

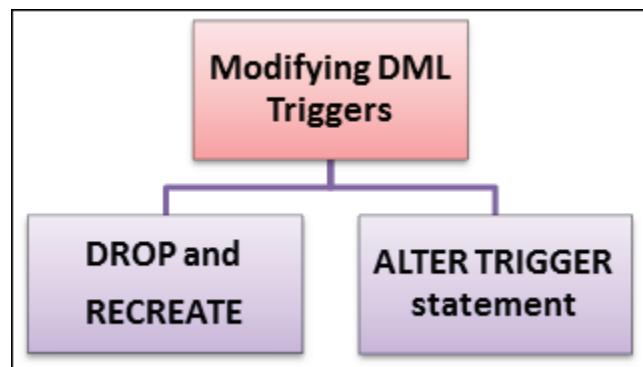
Các tham số trigger được định nghĩa vào thời điểm tạo ra trigger. Những tham số này bao gồm loại hành động kích hoạt để gọi trigger và các câu lệnh SQL được thực thi.

Nếu người dùng muốn sửa đổi bất kỳ tham số nào cho trigger DML, người dùng có thể làm như vậy trong bất kỳ một trong hai cách:

- ➔ Thả bỏ và tái tạo trigger với những tham số mới.
- ➔ Thay đổi những tham số này bằng cách sử dụng câu lệnh ALTER TRIGGER.

Nếu đối tượng tham khảo trigger DML được đổi tên, trigger phải được sửa đổi để phản ánh sự thay đổi trong tên đối tượng.

Ghi chú - Trigger DML có thể được mã hóa để ẩn định nghĩa của nó.



Hình 12.8: Sửa đổi các trigger DML

Sau đây là cú pháp cho việc sửa đổi trigger DML.

Cú pháp:

```

ALTER TRIGGER <trigger_name>
ON { <table_name> | <view_name> }
[WITH ENCRYPTION]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS <sql_statement>
  
```

trong đó:

WITH ENCRYPTION: chỉ ra rằng các định nghĩa trigger DML không được hiển thị.

FOR | AFTER: chỉ ra rằng trigger DML thực thi sau khi các hoạt động sửa đổi được hoàn tất.

INSTEAD OF: chỉ ra rằng trigger DML thực thi thay cho các hoạt động sửa đổi.

Đoạn mã 19 thay đổi trigger **CheckEmployeeID** đã tạo ra trên bảng **EmployeeDetails** bằng cách sử dụng tùy chọn WITH ENCRYPTION.

Đoạn mã 19:

```
ALTER TRIGGER CheckEmployeeID
ON EmployeeDetails
WITH ENCRYPTION
FOR INSERT
AS
IF 'E01' IN (SELECT EmployeeID FROM inserted)
BEGIN
PRINT 'Người dùng không thể chèn vào các khách hàng của Áo'
ROLLBACK TRANSACTION
END
```

Bây giờ, nếu người dùng cố gắng xem định nghĩa của trigger **CheckEmployeeID** bằng cách sử dụng thủ tục lưu trữ **sp_helptext**, thông báo lỗi sau đây được hiển thị:

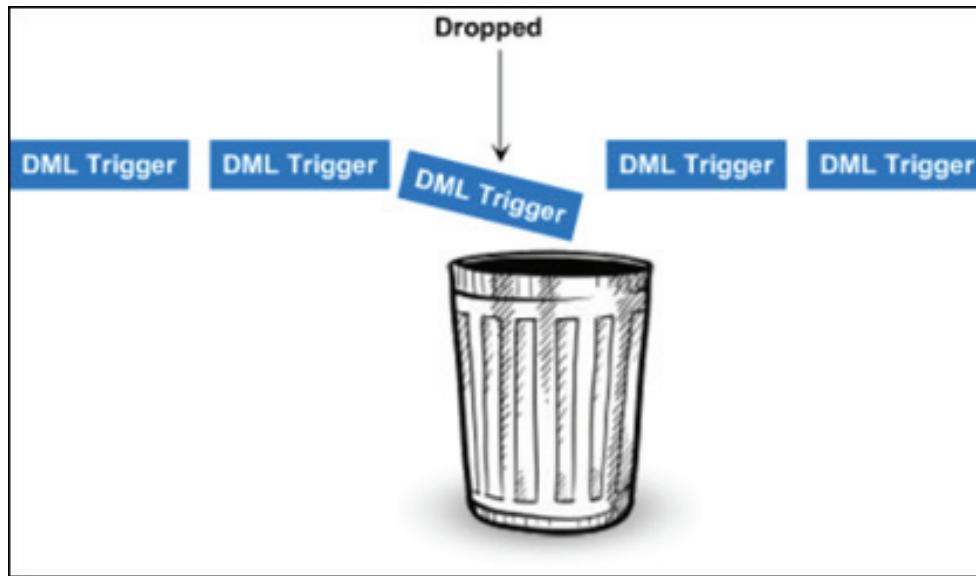
Văn bản cho đối tượng **CheckEmployeeID** được mã hóa.

12.8.3 Thả bỏ các trigger DML

SQL Server 2012 cung cấp tùy chọn thả bỏ một trigger DML đã tạo ra trên bảng nếu trigger không còn cần thiết. Có thể thả bỏ trigger bằng cách sử dụng câu lệnh **DROP TRIGGER**. Cũng có thể thả bỏ nhiều trigger bằng cách sử dụng một câu lệnh trigger thả bỏ đơn lẻ.

Khi một bảng bị thả bỏ, tất cả trigger đã định nghĩa trên bảng đó cũng được thả bỏ.

Hình 12.9 minh họa khái niệm về các trigger DML đã thả bỏ.



Hình 12.9: Thả bỏ các trigger DML

Ghi chú - Khi trigger DML bị xóa khỏi bảng này, thông tin về trigger cũng bị loại bỏ khỏi các khung nhìn danh mục.

Sau đây là cú pháp cho việc thả bỏ các trigger DML.

Cú pháp:

```
DROP TRIGGER <DML_trigger_name> [ , . . . n ]
```

trong đó:

`DML_trigger_name`: chỉ ra tên của trigger DML sẽ được thả bỏ.

`[, . . . n]`: chỉ ra rằng nhiều trigger DML có thể được thả bỏ.

Đoạn mã 20 thả bỏ trigger `CheckEmployeeID` đã tạo ra trên bảng `EmployeeDetails`.

Đoạn mã 20:

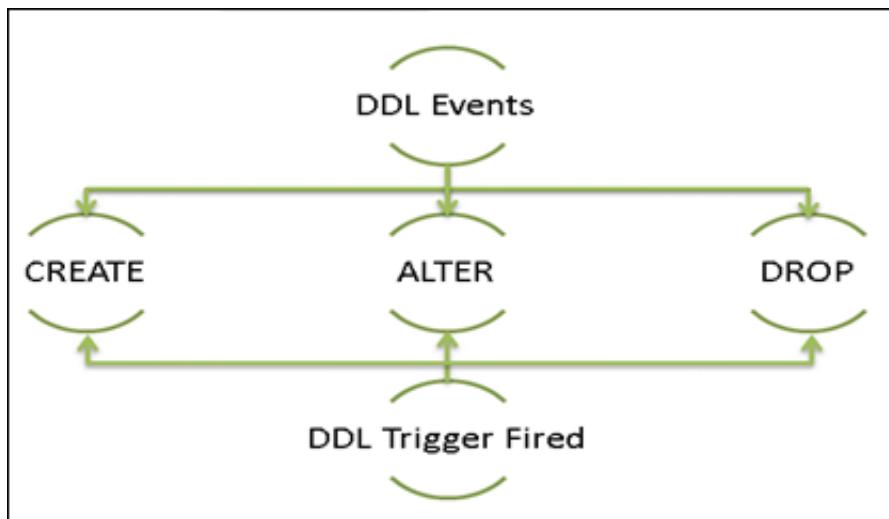
```
DROP TRIGGER CheckEmployeeID
```

12.9 Các trigger DDL

Trigger Ngôn ngữ định nghĩa dữ liệu (DDL) thực thi thủ tục lưu trữ khi các sự kiện DDL như là các câu lệnh CREATE, ALTER, và DROP xảy ra trong cơ sở dữ liệu hoặc máy chủ này. Các trigger DDL có thể chỉ hoạt động khi hoàn thành các sự kiện DDL.

Có thể sử dụng các trigger DDL để ngăn chặn những thay đổi trong lược đồ cơ sở dữ liệu. Lược đồ là một tập hợp các đối tượng như bảng, dạng xem, và vân vân trong một cơ sở dữ liệu.

Các trigger DDL có thể gọi một sự kiện hoặc hiển thị một thông báo dựa trên những sửa đổi đã thử trên lược đồ này. Các trigger DDL được định nghĩa ở mức cơ sở dữ liệu hoặc ở mức máy chủ. Hình 12.10 hiển thị các loại trigger DDL.



Hình 12.10: Các trigger DDL

Sau đây là cú pháp cho việc thả bỏ các trigger DDL.

Cú pháp:

```

CREATE TRIGGER <trigger_name>
ON { ALL SERVER | DATABASE }
[WITH ENCRYPTION]
{ FOR | AFTER } { <event_type> }
AS <sql_statement>
  
```

trong đó:

ALL SERVER: chỉ ra rằng trigger DDL thực thi khi các sự kiện DDL xảy ra trong máy chủ hiện tại.

DATABASE: chỉ ra rằng trigger DDL thực thi khi các sự kiện DDL xảy ra trong cơ sở dữ liệu hiện tại.

event_type: chỉ ra tên của sự kiện DDL để gọi trigger DDL.

Đoạn mã 21 tạo ra một trigger DDL để thả bỏ và thay đổi một bảng.

Đoạn mã 21:

```
CREATE TRIGGER Secure
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
PRINT 'Bạn phải vô hiệu hóa trigger "Secure" để thả bỏ hoặc sửa các bảng ! '
ROLLBACK;
```

Trong đoạn mã này, trigger DDL được tạo ra cho câu lệnh `DROP TABLE` và `ALTER TABLE`.

12.9.1 Phạm vi của các trigger DDL

Các trigger DDL được gọi bằng các câu lệnh SQL đã thực thi trong cơ sở dữ liệu hiện tại hoặc trên máy chủ hiện tại. Ví dụ, trigger DDL đã tạo ra cho câu lệnh `CREATE TABLE` thực thi trên sự kiện `CREATE TABLE` trong cơ sở dữ liệu. Trigger DDL đã tạo ra cho câu lệnh `CREATE LOGIN` thực thi trên sự kiện `CREATE LOGIN` trong máy chủ.

Phạm vi của trigger DDL phụ thuộc vào việc trigger thực thi cho các sự kiện cơ sở dữ liệu hay các sự kiện máy chủ. Theo đó, các trigger DDL được phân thành hai loại như sau:

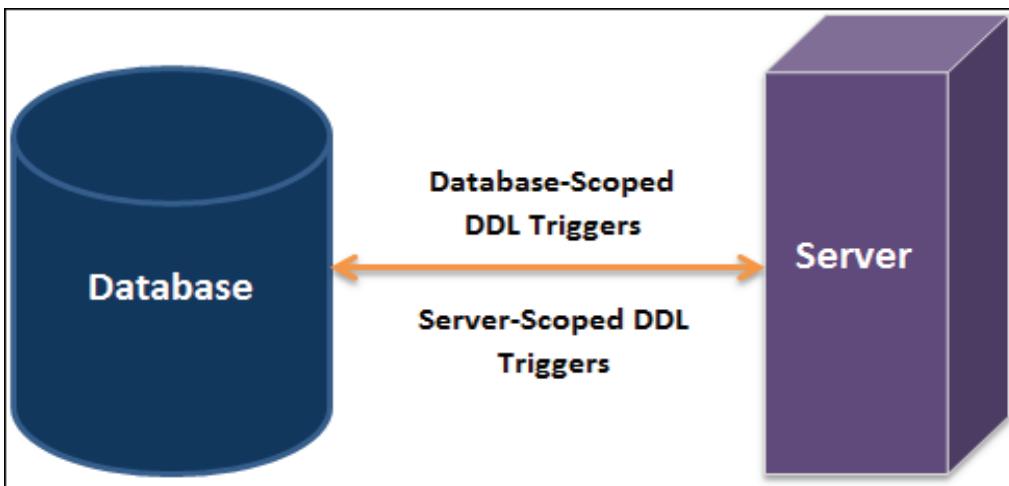
→ Các trigger DDL có phạm vi cơ sở dữ liệu

Các trigger DDL có phạm vi cơ sở dữ liệu được gọi bằng những sự kiện thay đổi lược đồ cơ sở dữ liệu. Những trigger này được lưu trữ trong cơ sở dữ liệu và thực thi trên các sự kiện DDL, ngoại trừ những cái liên đến các bảng tạm thời.

→ Các trigger DDL có phạm vi máy chủ

Các trigger DDL có phạm vi máy chủ được gọi bằng các sự kiện DDL ở mức máy chủ. Những trigger này được lưu trữ trong cơ sở dữ liệu `master`.

Hình 12.11 hiển thị phạm vi của các trigger DDL.



Hình 12.11: Phạm vi của các trigger DDL

12.10 Các trigger lồng nhau

Cả trigger DDL và DML được lồng nhau khi trigger thực hiện một hành động khởi xướng trigger khác. Có thể lồng các trigger DDL và DML lên đến 32 mức. Giả sử nếu một trigger sửa đổi một bảng trên đó có một trigger khác, trigger thứ hai được khởi xướng, sau đó gọi một trigger thứ ba, và vân vân.

Nếu các trigger lồng nhau được cho phép, khi đó những trigger này theo thứ tự bắt đầu một vòng lặp vô hạn. Điều này sẽ vượt quá mức lồng nhau và trigger sẽ chấm dứt.

Các trigger lồng nhau có thể được sử dụng để thực hiện các hàm như lưu trữ bản sao lưu của những hàng bị ảnh hưởng bằng những hành động trước đó.

Trigger Transact-SQL thực thi mã có quản lý thông qua tham khảo một thường trình CLR, tổng hợp, hoặc loại, tham chiếu số đếm làm một mức đối với giới hạn xếp lồng 32 cấp. Các phương pháp được gọi từ bên trong mã có quản lý không được đếm đối với giới hạn này.

Người dùng có thể vô hiệu hóa các trigger lồng nhau, bằng cách đặt tùy chọn trigger lồng nhau của `sp_configure` thành 0 hoặc off. Cấu hình mặc định được cho phép cho các trigger lồng nhau. Nếu tùy chọn trigger lồng nhau là off, khi đó trigger đệ quy bị vô hiệu hóa, không phân biệt thiết lập trigger đệ quy được đặt bằng cách sử dụng `ALTER DATABASE`.

Đoạn mã 22 tạo ra một trigger AFTER DELETE có tên là **Employee_Deletion** trên bảng **Employee_Personal_Details**.

Đoạn mã 22:

```
CREATE TRIGGER Employee_Deletion
ON Employee_Personal_Details
AFTER DELETE
AS
BEGIN
PRINT 'Việc xóa sẽ ảnh hưởng đến bảng Employee_Salary_Details'
DELETE FROM Employee_Salary_Details WHERE EmpID IN
(SELECT EmpID FROM deleted)
END
```

Khi một bản ghi được xóa khỏi bảng **Employee_Personal_Details**, trigger **Employee_Deletion** được kích hoạt và tin nhắn được hiển thị. Ngoài ra, bản ghi của nhân viên được xóa khỏi bảng **Employee_Salary_Details**.

Đoạn mã 23 tạo ra một trigger AFTER DELETE **Deletion_Confirmation** trên bảng **Employee_Salary_Details**.

Đoạn mã 23:

```
CREATE TRIGGER Deletion_Confirmation
ON Employee_Salary_Details
AFTER DELETE
AS
BEGIN
PRINT 'Chi tiết nhân viên đã xóa thành công khỏi bảng Employee_Salary_Details'
END
DELETE FROM Employee_Personal_Details WHERE EmpID=1
```

Khi bản ghi được xóa khỏi bảng **Employee_Salary_Details**, trigger **Deletion_Confirmation** được kích hoạt. Trigger này in thông báo xác nhận về bản ghi đang bị xóa.

Như vậy, các trigger **Employee_Deletion** và **Deletion_Confirmation** được thấy sẽ được lồng nhau.

12.11 UPDATE()

Hàm UPDATE () trả về một giá trị Boolean chỉ ra liệu hành động UPDATE hoặc INSERT đã được thực hiện trên một khung nhìn hoặc cột cụ thể của một bảng.

Hàm UPDATE () có thể được sử dụng bất cứ nơi nào bên trong phần thân của một trigger UPDATE hoặc INSERT Transact-SQL để kiểm tra xem trigger nên thực thi một số hành động hay không.

Sau đây là cú pháp cho UPDATE () .

Cú pháp:

```
UPDATE ( column )
```

trong đó:

column: là tên của cột để kiểm tra về hành động INSERT hoặc UPDATE.

Đoạn mã 24 tạo ra trigger **Accounting** trên bảng **Account_Transactions** để cập nhật các cột **TransactionID** hoặc **EmployeeID**.

Đoạn mã 24:

```
CREATE TRIGGER Accounting
ON Account_Transactions
AFTER UPDATE
AS
IF ( UPDATE (TransactionID) OR UPDATE (EmployeeID) )
BEGIN
RAISERROR (50009, 16, 10)
END;
GO
```

12.12 Xử lý nhiều hàng trong một phiên

Khi người dùng viết mã cho một trigger DML, khi đó câu lệnh này làm cho trigger kích hoạt sẽ là câu lệnh đơn lẻ. Câu lệnh đơn lẻ này sẽ ảnh hưởng đến nhiều hàng dữ liệu, thay vì một hàng đơn lẻ. Đây là một hành vi phổ biến cho các trigger DELETE và UPDATE bởi những câu lệnh này thường ảnh hưởng đến nhiều hàng. Hành vi cho các trigger INSERT ít phổ biến hơn bởi câu lệnh INSERT cơ bản chỉ thêm một hàng.

Khi chức năng của một trigger DML tự động kéo theo việc tính toán lại các giá trị tóm tắt của một bảng và lưu trữ kết quả vào bảng khác, sau đó cân nhắc nhiều hàng là rất quan trọng.

Đoạn mã 25 lưu trữ tổng đang chạy cho một lần chèn đơn hàng.

Đoạn mã 25:

```
USE AdventureWorks2012;
GO
CREATE TRIGGER PODetails
ON Purchasing.PurchaseOrderDetail
AFTER INSERT AS
    UPDATE PurchaseOrderHeader
    SET SubTotal = SubTotal + LineTotal
    FROM inserted
    WHERE PurchaseOrderHeader.PurchaseOrderID = inserted.PurchaseOrderID;
```

Trong đoạn mã này, tổng phụ được tính toán và lưu trữ cho một hoạt động chèn đơn hàng.

Đoạn mã 26 lưu trữ tổng đang chạy cho một lần chèn đa hàng hoặc đơn hàng.

Đoạn mã 26:

```
USE AdventureWorks2012;
GO
CREATE TRIGGER PODetailsMultiple
ON Purchasing.PurchaseOrderDetail
AFTER INSERT AS
    UPDATE Purchasing.PurchaseOrderHeader
    SET SubTotal = SubTotal +
        (SELECT SUM(LineTotal)
        FROM inserted
        WHERE PurchaseOrderHeader.PurchaseOrderID
        = inserted.PurchaseOrderID)
    WHERE PurchaseOrderHeader.PurchaseOrderID IN
        (SELECT PurchaseOrderID FROM inserted);
```

Trong đoạn mã này, tổng phụ được tính toán và lưu trữ cho hoạt động chèn đa hàng hoặc đơn hàng.

12.13 Ảnh hưởng hiệu suất của các trigger

Trong thực tế, trigger không mang chi phí, mà là chúng khá nhạt. Tuy nhiên, nhiều vấn đề hiệu suất có thể xảy ra do logic hiện tại bên trong trigger. Giả sử một trigger tạo ra một con trỏ và các vòng qua nhiều hàng, sau đó sẽ có một sự suy giảm trong quá trình này.

Tương tự như vậy, hãy xem xét rằng trigger thực thi các câu lệnh SQL khác nhau đối với các bảng khác tách biệt khỏi các bảng Inserted và Deleted. Điều này một lần nữa sẽ dẫn đến sự suy giảm tốc độ của các câu lệnh SQL nằm trong trigger này.

Một nguyên tắc tốt sẽ giữ cho logic đơn giản trong những trigger và tránh sử dụng các con trỏ trong khi thực thi các câu lệnh đối với bảng khác và các tác vụ khác làm cho hiệu suất chậm lại.

12.14 Kiểm tra kiến thức của bạn

1. Câu nào sau đây về các trigger trong SQL Server 2012 là đúng?

a.	Trigger lấy thông tin từ các bảng của cùng cơ sở dữ liệu cũng như cơ sở dữ liệu khác.	c.	Các trigger DML thực thi trên các câu lệnh INSERT, UPDATE, và DELETE.
b.	Các trigger DDL chỉ hoạt động sau khi một bảng hoặc khung nhìn được sửa đổi.	d.	Các trigger DDL thực thi trong khi sửa đổi dữ liệu hoặc sau khi dữ liệu được sửa đổi.
(A)	a, b, c	(C)	a, b, d
(B)	b, c, d	(D)	a, c, d

2. So khớp các loại trigger DML trong SQL Server 2012 đối với các mô tả tương ứng của chúng.

	Mô tả		Trigger DML
a.	Thực thi khi người dùng thay thế một bản ghi hiện có bằng một giá trị mới.	1.	INSERT
b.	Thực thi khi hoàn thành các hoạt động sửa đổi.	2.	UPDATE
c.	Thực thi thay cho các hoạt động sửa đổi.	3.	DELETE
d.	Thực hiện khi người dùng thêm một bản ghi lên một bảng.	4.	AFTER
e.	Thực hiện khi người dùng loại bỏ một bản ghi khỏi một bảng.	5.	INSTEAD OF

(A)	a-1, b-4, c-2, d-3, e-5	(C)	a-2, b-4, c-3, d-5, e-1
(B)	a-2, b-4, c-5, d-1, e-3	(D)	a-1, b-2, c-3, d-4, e-5

3. Câu nào sau đây về các trigger trong SQL Server 2012 là đúng?

a.	Các trigger DML có thể thực hiện nhiều hành động cho mỗi câu lệnh sửa đổi.	c.	Các trigger UPDATE không sử dụng bảng Deleted để cập nhật các bản ghi trong một bảng.
b.	Các bảng Inserted và Deleted được tạo ra bởi SQL Server 2012 khi một bảng mới được tạo ra trong cơ sở dữ liệu.	d.	Các trigger đã xóa không sử dụng bảng Inserted để xóa các bản ghi trong một bảng.

(A)	a, b	(C)	a, d
(B)	c, d	(D)	b, d

4. Câu nào sau đây về làm việc với các trigger DML của SQL Server 2012 là đúng?

a.	Mỗi hành động kích hoạt không có thể có nhiều trigger AFTER.	c.	Định nghĩa trigger DML có thể được sửa đổi bằng cách thả bỏ và tái tạo trigger đó.
b.	Hai hành động kích hoạt trên một bảng có thể có cùng các trigger đầu tiên và cuối cùng.	d.	Định nghĩa trigger DML có thể được xem bằng cách sử dụng thủ tục lưu trữ sp_helptrigger.

(A)	a, c	(C)	b, d
(B)	b, c	(D)	c, d

5. Các trigger _____ có thể được sử dụng để thực hiện các hàm như lưu trữ bản sao lưu của những hàng bị ảnh hưởng bằng những hành động trước đó.

(A)	Lồng nhau	(C)	DDL
(B)	DML	(D)	INSTEAD OF

12.14.1 Câu trả lời

1.	A
2.	B
3.	C
4.	D
5.	A

Tóm tắt

- Trigger là một thủ tục lưu trữ được thực hiện khi có một nỗ lực để sửa đổi dữ liệu trong bảng được bảo vệ bằng trigger.
- Các trigger đăng nhập thực thi các thủ tục lưu trữ khi một phiên được thiết lập với một sự kiện LOGON.
- Các trigger DML được thực thi khi sự kiện DML xảy ra trong các bảng hoặc khung nhìn.
- Trigger INSERT được thực thi khi một bản ghi mới được chèn vào bảng.
- Trigger UPDATE sao chép bản ghi gốc vào bảng Deleted và bản ghi mới vào bảng Inserted khi bản ghi được cập nhật.
- Có thể tạo ra trigger DELETE để hạn chế người dùng không xoá một bản ghi cụ thể trong bảng.
- Trigger AFTER được thực thi khi hoàn thành các hoạt động INSERT, UPDATE, hoặc DELETE.



1. **Galaxy Airlines** là một dịch vụ hàng hàng không vừa được tung ra, hoạt động các chuyến bay đến và đi từ các thành phố khác nhau trên toàn châu Âu. Công ty duy trì dữ liệu liên quan đến các giao tác hàng ngày liên quan đến các dịch vụ bay trong cơ sở dữ liệu SQL Server 2012. Để cho phép hoạt động hiệu quả và nhanh hơn, **Galaxy Airlines** đã quyết định kết hợp sử dụng các trigger trong các ứng dụng cơ sở dữ liệu của họ. Danh sách chi tiết các hoạt động phải được thực hiện được liệt kê như sau:
- b. Tạo ra những bảng sau đây trong cơ sở dữ liệu **GalaxyAirlines**. Bảng 12.2 liệt kê bảng **Flight**.

Tên trường	Loại dữ liệu	Trường khóa	Mô tả
Aircraft_code	varchar(10)	Khóa chính	Lưu trữ mã máy bay
Loại	varchar(6)		Mô tả loại máy bay
Source	varchar(20)		Lưu trữ tên của thành phố là nơi máy bay sẽ khởi hành
Destination	varchar(20)		Lưu trữ tên của thành phố là nơi máy bay sẽ đến
Dep_time	varchar(10)		Lưu trữ thời gian khởi hành
Journey_hrs	int		Lưu trữ số giờ hành trình

Bảng 12.2: Bảng Flight

Bảng 12.3 liệt kê bảng **Flight_Details**.

Tên trường	Loại dữ liệu	Trường khóa	Mô tả
Class_Code	varchar(10)	Khóa chính	Lưu trữ hạng, gồm ưu tiên, thương gia hoặc phổ thông
Aircraft_code	varchar(10)	Khóa ngoại	Lưu trữ mã máy bay
Tiền vé	money		Lưu trữ số tiền vé
Số ghế	int		Lưu trữ tổng số chỗ ngồi trên chuyến bay

Bảng 12.3: Bảng Flight_Details

- b. Viết các câu lệnh để tạo ra trigger **trgCheckSeats** sẽ kích hoạt bất cứ khi nào một hàng mới được đưa vào bảng **Flight_Details**. Giới hạn tối đa số ghế một chuyến bay có thể chứa là

Hãy thử tự làm

- c. Chèn ít nhất là năm bản ghi vào mỗi bảng.
- d. Viết các câu lệnh để tạo ra trigger `UpdateValid` sẽ kích hoạt mỗi lần hàng được cập nhật vào bảng `Flight_Details`. Trigger nên xác định xem cột Seats có hiện diện trong danh sách các cột đang được cập nhật hay không. Nếu có, hoạt động UPDATE sẽ không thành công vì cột Seats được định nghĩa là một hằng số và không thể thay đổi được.
- e. Viết các câu lệnh để tạo ra một trigger DDL `ProhibitDelete` sẽ kích hoạt bất cứ khi nào người dùng đang cố gắng xóa một bảng khỏi cơ sở dữ liệu `Galaxy Airlines`. Trigger không phải cho phép người dùng thực hiện ca lần xóa và phải hiển thị thông báo "You are not allowed to delete tables in this database" (Bạn không được phép xóa các bảng trong cơ sở dữ liệu này).

“ Democritus nói, lời nói là
cái bóng tối của hành động

Phần - 13

Lập trình Transact-SQL

Chào mừng bạn đến với phần **Lập trình Transact-SQL**.

Phần này giới thiệu lập trình với Transact-SQL và mô tả các phần tử lập trình Transact-SQL khác nhau. Phần này còn mô tả các câu lệnh luồng chương trình, các hàm Transact-SQL, và vân vân. Phần này giải thích thêm thủ tục để tạo ra và thay đổi các hàm do người dùng định nghĩa, và tạo ra các cửa sổ sử dụng hàm OVER và cửa sổ.

Trong phần này, bạn sẽ học để:

- ➔ Mô tả tổng quan về lập trình Transact-SQL
- ➔ Mô tả các phần tử lập trình Transact-SQL
- ➔ Mô tả các câu lệnh luồng chương trình
- ➔ Mô tả các hàm Transact-SQL khác nhau
- ➔ Giải thích thủ tục để tạo ra và thay đổi các hàm do người dùng định nghĩa (UDF)
- ➔ Giải thích việc tạo ra các cửa sổ với OVER
- ➔ Mô tả các hàm cửa sổ

13.1 Giới thiệu

Lập trình Transact-SQL là một phần mở rộng ngôn ngữ thủ tục cho SQL. Lập trình Transact-SQL được mở rộng bằng cách thêm các thường trình con và các cấu trúc lập trình tương tự như các ngôn ngữ cấp cao. Như các ngôn ngữ cấp cao, lập trình Transact-SQL cũng có những quy tắc và cú pháp kiểm soát và cho phép các câu lệnh lập trình để làm việc cùng nhau. Người dùng có thể kiểm soát dòng chảy của các chương trình bằng cách sử dụng các câu lệnh điều kiện như `IF` và các vòng lặp như `WHILE`.

13.2 Các phần tử lập trình Transact-SQL

Các phần tử lập trình Transact-SQL cho phép thực hiện các phép tính khác nhau mà không thể được thực hiện trong một câu lệnh duy nhất. Người dùng có thể nhóm một số câu lệnh Transact-SQL cùng nhau bằng cách sử dụng một trong những cách sau:

→ Khối lệnh

Khối lệnh là tập hợp một hoặc nhiều câu lệnh Transact-SQL được gửi như một đơn vị từ một ứng dụng đến máy chủ.

→ Thủ tục lưu trữ

Thủ tục lưu trữ là một tập hợp các câu lệnh Transact-SQL được biên dịch sẵn và định nghĩa trước trên máy chủ.

→ Trigger

Trigger là một loại thủ tục lưu trữ đặc biệt được thực hiện khi người dùng thực hiện một sự kiện như là phép tính `INSERT`, `DELETE`, or `UPDATE` trên bảng.

→ Tập lệnh

Tập lệnh là một chuỗi các câu lệnh Transact-SQL được lưu trữ trong một tập tin được sử dụng làm đầu vào cho trình biên tập mã SSMS hoặc tiện ích `sqlcmd`.

Những tính năng sau đây cho phép người dùng làm việc với các câu lệnh Transact-SQL:

→ Biến

Biến cho phép người dùng lưu trữ dữ liệu có thể được sử dụng làm đầu vào trong một câu lệnh Transact-SQL.

→ Điều khiển luồng

Điều khiển luồng được sử dụng để đưa vào các cấu trúc điều kiện trong Transact-SQL.

→ Xử lý lỗi

Xử lý lỗi là một cơ chế được sử dụng để xử lý các lỗi và cung cấp thông tin cho người dùng về lỗi đã xảy ra.

13.2.1 Các khối lệnh Transact-SQL

Khối lệnh Transact-SQL là nhóm một hoặc nhiều câu lệnh Transact-SQL được gửi đến máy chủ như một đơn vị từ một ứng dụng. SQL Server biên dịch khối lệnh SQL vào một đơn vị thực thi duy nhất, còn được gọi là một kế hoạch thực hiện. Trong kế hoạch thực hiện, những câu lệnh SQL được thực hiện từng cái một. Khối lệnh Transact-SQL nên được kết thúc bằng dấu chấm phẩy. Điều kiện này là không bắt buộc, nhưng tiện ích này kết thúc câu lệnh mà không có một dấu chấm phẩy bị phản đối và có thể bị loại bỏ trong các phiên bản mới của SQL Server trong tương lai. Do đó, nên sử dụng các dấu chấm phẩy để kết thúc các khối lệnh.

Lỗi biên dịch như là lỗi cú pháp hạn chế việc biên dịch kế hoạch thực hiện. Vì vậy, nếu lỗi thời gian biên dịch xảy ra, không có câu lệnh trong khối lệnh được thực hiện.

Lỗi thời gian chạy như là vi phạm ràng buộc hoặc tràn số học có một trong những tác động sau:

- ➔ Hầu hết các lỗi thời gian chạy dừng câu lệnh hiện tại và các câu lệnh tiếp theo trong khối lệnh.
- ➔ Lỗi thời gian chạy cụ thể như là vi phạm ràng buộc dừng chỉ câu lệnh hiện có và các câu lệnh còn lại trong khối lệnh được thực hiện.

Các câu lệnh SQL thực hiện trước khi gặp phải lỗi thời gian chạy sẽ không bị ảnh hưởng. Ngoại lệ duy nhất là khi khối lệnh ở trong một giao tác và lỗi dẫn đến giao tác được phục hồi.

Ví dụ, giả sử có 10 câu lệnh trong một khối lệnh và câu lệnh thứ sáu có lỗi cú pháp, khi đó những câu lệnh còn lại trong khối lệnh sẽ không thực hiện. Nếu khối lệnh được biên dịch và câu lệnh thứ ba không chạy, khi đó, kết quả của hai câu lệnh đầu tiên vẫn không bị ảnh hưởng vì nó đã được thực hiện.

Những quy tắc sau đây được áp dụng để sử dụng các khối lệnh:

1. Câu lệnh CREATE FUNCTION, CREATE DEFAULT, CREATE RULE, CREATE TRIGGER, CREATE PROCEDURE, CREATE VIEW, và CREATE SCHEMA không thể được sử dụng cùng với các câu lệnh khác trong một khối lệnh. Câu lệnh CREATE SQL bắt đầu khối lệnh và tất cả các câu lệnh khác có bên trong khối lệnh sẽ được coi như là một phần của định nghĩa câu lệnh CREATE.
2. Không có thay đổi được thực hiện trong bảng và các cột mới tham khảo cùng một khối lệnh.
3. Nếu câu lệnh đầu tiên trong một khối lệnh có câu lệnh EXECUTE, khi đó, từ khóa EXECUTE là không cần thiết. Nó chỉ cần thiết khi câu lệnh EXECUTE không tồn tại trong câu lệnh đầu tiên trong khối lệnh.

Đoạn mã 1 tạo ra một khung nhìn trong khối lệnh.

Đoạn mã 1:

```
USE AdventureWorks2012;
GO
CREATE VIEW dbo.vProduct
```

```
AS
SELECT ProductNumber, Name
FROM Product;
GO
SELECT *
FROM dbo.vProduct;
GO
```

Trong đoạn mã này, một khung nhìn được tạo ra trong khối lệnh. CREATE VIEW là câu lệnh duy nhất trong khối lệnh, những lệnh GO là rất cần thiết để tách câu lệnh CREATE VIEW ra khỏi các câu lệnh SELECT và USE. Đây là một ví dụ đơn giản để chứng minh việc sử dụng một khối lệnh. Trong thực tế, một số lượng lớn các câu lệnh có thể được sử dụng trong một khối lệnh đơn lẻ. Nó cũng có thể kết hợp hai hay nhiều khối lệnh trong một giao tác.

Đoạn mã 2 trình bày ví dụ về điều này.

Đoạn mã 2:

```
BEGIN TRANSACTION
GO
USE AdventureWorks2012;
GO
CREATE TABLE Company
(
Id_Num int IDENTITY(100, 5),
Company_Name nvarchar(100)
)
GO
INSERT Company (Company_Name)
VALUES (N'A Bike Store')
INSERT Company (Company_Name)
VALUES (N'Progressive Sports')
INSERT Company (Company_Name)
VALUES (N'Modular Cycle Systems')
INSERT Company (Company_Name)
VALUES (N'Advanced Bike Components')
```

```

INSERT Company (Company_Name)
VALUES (N'Metropolitan Sports Supply')

INSERT Company (Company_Name)
VALUES (N'Aerobic Exercise Company')

INSERT Company (Company_Name)
VALUES (N'Associated Bikes')

INSERT Company (Company_Name)
VALUES (N'Exemplary Cycles')

GO

SELECT Id_Num, Company_Name
FROM dbo.Company
ORDER BY Company_Name ASC;

GO

COMMIT;

GO
  
```

Trong đoạn mã này, một số khối lệnh được kết hợp thành một giao tác. Các câu lệnh BEGIN TRANSACTION và COMMIT kèm theo các câu lệnh giao tác. Các câu lệnh CREATE TABLE, BEGIN TRANSACTION, SELECT, COMMIT, và USE là trong các khối lệnh một câu lệnh. Các câu lệnh INSERT tất cả được đưa vào trong một khối lệnh.

13.2.2 Các biến Transact-SQL

Biến cho phép người dùng lưu trữ dữ liệu để sử dụng như là đầu vào trong một câu lệnh Transact-SQL. Ví dụ, người dùng có thể tạo ra một truy vấn yêu cầu các loại giá trị dữ liệu khác nhau đã chỉ định trong mệnh đề WHERE mỗi lần truy vấn được thực thi. Ở đây, người dùng có thể sử dụng các biến trong mệnh đề WHERE, và viết logic để lưu trữ các biến với dữ liệu thích hợp.

SQL Server cung cấp các câu lệnh sau đây để đặt và khai báo các biến cục bộ.

→ DECLARE

Biến được khai báo với câu lệnh DECLARE trong phần thân của khối lệnh. Những biến này được gán các giá trị bằng cách sử dụng câu lệnh SELECT hoặc SET. Những biến này được khởi tạo với các giá trị NULL nếu người dùng đã không cung cấp một giá trị tại thời điểm khai báo.

Sau đây là cú pháp cơ bản để khai báo một biến cục bộ.

Cú pháp:

```
DECLARE { { @local_variable [AS] data_type } | [=value] }
```

trong đó:

`@local_variable`: chỉ ra tên của các biến và bắt đầu với ký hiệu `@`.

`data_type`: chỉ ra kiểu dữ liệu. Biến không thể thuộc kiểu dữ liệu `image`, `text`, hoặc `ntext`.

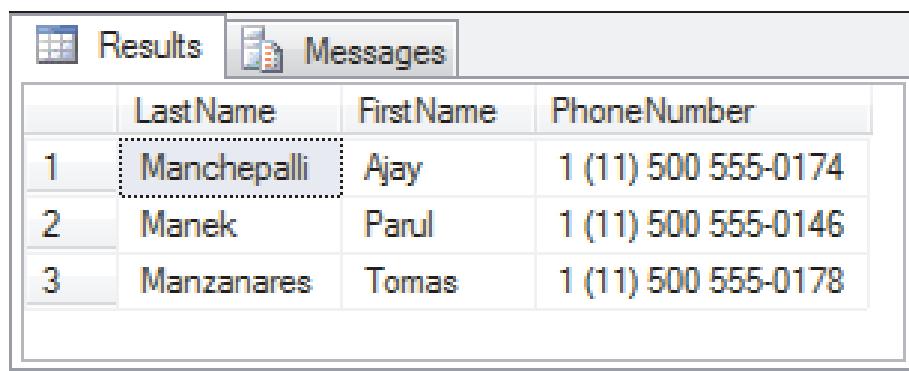
`=value`: Gán một giá trị nội tuyến cho một biến. Giá trị có thể là một biểu thức hoặc một giá trị hằng số. Giá trị phải phù hợp với loại khai báo biến hay nó nên được mặc nhiên chuyển đổi sang loại đó.

Đoạn mã 3 sử dụng một biến cục bộ để lấy thông tin liên hệ cho các tên gọi bắt đầu bằng Man.

Đoạn mã 3:

```
USE AdventureWorks2012;
GO
DECLARE @find varchar(30) = 'Man%';
SELECT p.LastName, p.FirstName, ph.PhoneNumber
FROM Person.Person AS p
JOIN Person.PersonPhone AS ph ON p.BusinessEntityID=ph.BusinessEntityID
WHERE LastName LIKE @find;
```

Trong đoạn mã này, một biến cục bộ có tên là `@find` được sử dụng để lưu trữ các tiêu chí tìm kiếm, mà sau đó sẽ được sử dụng để lấy thông tin liên hệ. Ở đây, những tiêu chí bao gồm tất cả các tên gọi bắt đầu bằng Man. Hình 13.1 hiển thị đầu ra.



	Last Name	First Name	Phone Number
1	Manchepalli	Ajay	1 (11) 500 555-0174
2	Manek	Parul	1 (11) 500 555-0146
3	Manzanares	Tomas	1 (11) 500 555-0178

Hình 13.1: Thông tin liên hệ

→ SET

Câu lệnh `SET` thiết lập biến cục bộ được tạo ra bằng câu lệnh `DECLARE` với giá trị chỉ định.

Sau đây là cú pháp cơ bản để đặt một biến cục bộ.

Cú pháp:

```
SET
{ @local_variable = {biểu thức}
}
|
{
@local_variable
{ += | -= | *= | /= | %= | &= | ^= | |= } biểu thức
}
```

trong đó:

`@local_variable`: chỉ ra tên của biến và bắt đầu với ký hiệu `@`.

`=`: Gán giá trị ở bên phải cho biến ở phía bên trái.

`{ = | += | -= | *= | /= | %= | &= | ^= | |= }`: chỉ ra các toán tử gán hỗn hợp.

Đó là:

- `+=` Cộng và sau đó, gán
- `-=` Trừ và sau đó, gán
- `*=` Nhân và sau đó, gán
- `/=` Chia và sau đó, gán
- `%=` Modulo và sau đó, gán
- `&=` Bitwise AND và sau đó, gán
- `^=` Bitwise XOR và sau đó, gán
- `|=` Bitwise OR và sau đó, gán

`biểu thức`: chỉ ra bất kỳ biểu thức hợp lệ nào thậm chí có thể bao gồm một truy vấn con vô hướng.

Đoạn mã 4 trình bày việc sử dụng `SET` để gán giá trị chuỗi cho một biến.

Đoạn mã 4:

```
DECLARE @myvar char(20);
SET @myvar = 'Đây là thử nghiệm';
```

Trong đoạn mã này, biến `@myvar` được gán một giá trị chuỗi.

→ SELECT

Câu lệnh `SELECT` chỉ ra rằng biến cục bộ chỉ định đã được tạo ra sử dụng `DECLARE` nên được đặt thành biến đã cho.

Sau đây là cú pháp của câu lệnh SELECT.

Cú pháp:

```
SELECT { @local_variable { = | += | -= | *= | /= | %= | &= | ^= | |= } biểu thức } [ ,...n ] [ ; ]
```

trong đó:

`@local_variable`: chỉ ra các biến cục bộ để một giá trị sẽ được gán cho nó.

`=`: Gán giá trị ở bên phải cho biến ở phía bên trái.

`{ = | += | -= | *= | /= | %= | &= | ^= | |= }`: chỉ ra các toán tử gán hỗn hợp.

`biểu thức`: chỉ ra bất kỳ biểu thức hợp lệ nào thậm chí có thể bao gồm một truy vấn con vô hướng.

Đoạn mã 5 trình bày cách sử dụng SELECT để trả về một giá duy nhất.

Đoạn mã 5:

```
USE AdventureWorks2012 ;
GO
DECLARE @var1 nvarchar(30);
SELECT @var1 = 'Công ty vô danh';

SELECT @var1 = Name
FROM Sales.Store
WHERE BusinessEntityID = 10;

SELECT @var1 AS 'Tên công ty';
```

Trong đoạn mã này, biến `@var1` được gán Công ty vô danh làm giá trị của nó.

Truy vấn đối với bảng Store sẽ trả lại không hàng là giá trị được chỉ ra cho `BusinessEntityID` không tồn tại trong bảng này. Sau đó biến sẽ giữ lại giá trị Công ty vô danh và sẽ được hiển thị với đầu đề Tên công ty. Hình 13.2 hiển thị đầu ra.

	Company Name
1	Unnamed Company

Hình 13.2: Tên chung

Mặc dù cả hai câu lệnh SET và SELECT trông tương tự nhau, nhưng không phải vậy. Dưới đây là một vài sự khác biệt giữa hai câu lệnh này:

- Có thể gán mỗi lần chỉ một biến sử dụng SET. Tuy nhiên, sử dụng SELECT bạn có thể làm nhiều phép gán cùng một lúc.
- SET có thể chỉ gán một giá trị vô hướng khi gán từ một truy vấn. Nó gây nên một lỗi và không làm việc nếu truy vấn trả về nhiều giá trị/hàng. Tuy nhiên, SELECT gán một trong những giá trị trả lại cho biến và người dùng thậm chí sẽ không biết rằng nhiều giá trị đã được trả lại.

Ghi chú - Để gán các biến, đề nghị sử dụng SET @local_variable thay vì SELECT @local_variable.

13.3 Các từ đồng nghĩa

Từ đồng nghĩa là các đối tượng cơ sở dữ liệu phục vụ cho các mục đích sau:

- Chúng cung cấp một tên khác cho một đối tượng cơ sở dữ liệu khác, còn được gọi là đối tượng cơ bản, có thể tồn tại trên một máy chủ từ xa hoặc cục bộ.
- Chúng thể hiện một lớp trừu tượng bảo vệ ứng dụng máy khách từ các sửa đổi được thực hiện cho vị trí và tên của đối tượng cơ sở.

Ví dụ, hãy xem xét bảng Department của AdventureWorks2012 nằm trên máy chủ đầu tiên có tên là **Server1**. Để tham khảo bảng này từ máy chủ thứ hai **Server2**, ứng dụng máy chủ phải sử dụng tên bốn phần Server1.AdventureWorks2012.Person.Department. Nếu vị trí của bảng đã được sửa đổi, ví dụ, đến một máy chủ khác, ứng dụng máy khách sẽ phải được sửa chữa để phản ánh sự thay đổi đó. Để giải quyết cả hai vấn đề này, người dùng có thể tạo ra một từ đồng nghĩa **DeptEmpTable**, trên **Server2** cho bảng Department trên **Server1**. Böyle giờ, ứng dụng máy khách chỉ phải sử dụng tên duy nhất **DeptEmpTable**, để tham khảo tới bảng Employee .

Tương tự như vậy, nếu vị trí của bảng Department thay đổi, người dùng phải sửa đổi từ đồng nghĩa **DeptEmpTable**, để trả đến vị trí mới của bảng Department. Do không có câu lệnh ALTER SYNONYM, đầu tiên bạn phải thả bỏ từ đồng nghĩa **DeptEmpTable**, và sau đó, tạo ra lại từ đồng nghĩa có cùng tên, nhưng trả từ đồng nghĩa tới vị trí mới của Department.

Ghi chú - Từ đồng nghĩa là một phần của lược đồ, và tương tự như các đối tượng lược đồ khác, tên đồng nghĩa phải là duy nhất.

Bảng 13.1 liệt kê các đối tượng cơ sở dữ liệu mà người dùng có thể tạo ra các từ đồng nghĩa.

Các đối tượng của cơ sở dữ liệu
Thủ tục lưu trữ mở rộng
Hàm có giá trị bảng SQL
Thủ tục lưu trữ SQL

Các đối tượng của cơ sở dữ liệu
Bảng (do người dùng định nghĩa)
Nhân bản-lọc-thủ tục
Hàm vô hướng SQL
Hàm có giá trị bảng trong dòng SQL
Dạng xem

Bảng 13.1: Các đối tượng của cơ sở dữ liệu

→ Từ đồng nghĩa và lược đồ

Giả sử người dùng muốn tạo ra một từ đồng nghĩa và có một lược đồ mặc định không thuộc sở hữu của họ. Trong trường hợp này, họ có thể hội đủ điều kiện tên đồng nghĩa với tên lược đồ mà họ thực sự sở hữu. Xem xét ví dụ mà người dùng sở hữu một lược đồ **Resources**, nhưng **Materials** là lược đồ mặc định của người dùng. Nếu người dùng này muốn tạo ra một từ đồng nghĩa, họ phải thêm tiền tố cho tên của từ đồng nghĩa với lược đồ **Resources**.

→ Cấp quyền trên các từ đồng nghĩa

Chỉ các thành viên của vai trò `db_owner` hoặc `db_ddladmin` hoặc chủ sở hữu đồng nghĩa được phép cấp quyền trên một từ đồng nghĩa. Người dùng có thể từ chối, cấp, hoặc thu hồi tất cả hoặc bất kỳ quyền nào trên một từ đồng nghĩa. Bảng 13.2 hiển thị danh sách các quyền được áp dụng trên một từ đồng nghĩa.

Các quyền
DELETE
INSERT
TAKE OWNERSHIP
VIEW DEFINITION
CONTROL
EXECUTE
SELECT
UPDATE

Bảng 13.2: Các quyền

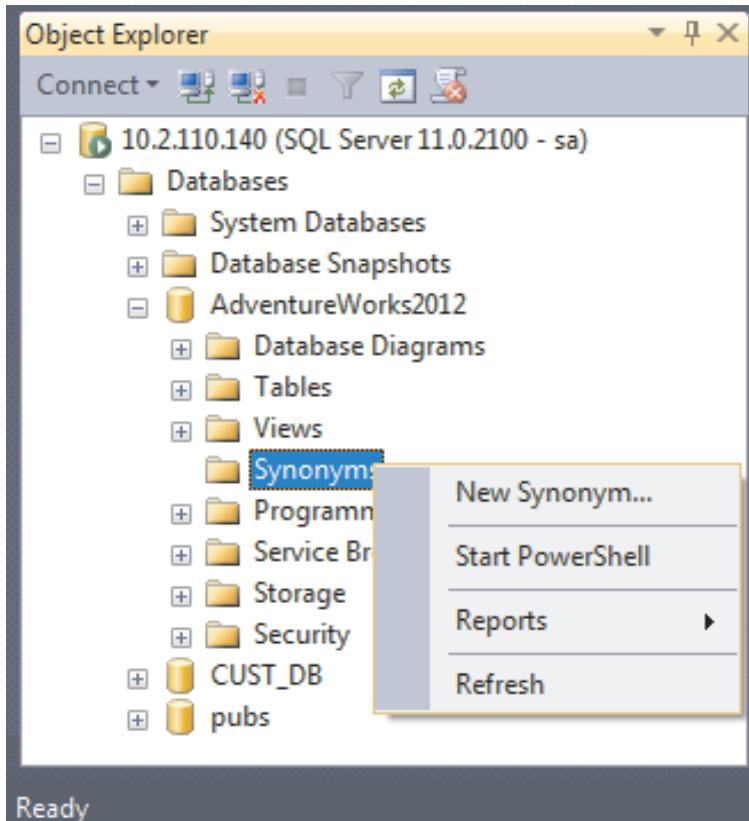
→ Làm việc với các đồng nghĩa

Người dùng có thể làm việc với các từ đồng nghĩa trong SQL Server 2012 sử dụng Transact-SQL hoặc SSMS.

Để tạo ra một từ đồng nghĩa sử dụng SSMS, thực hiện các bước sau:

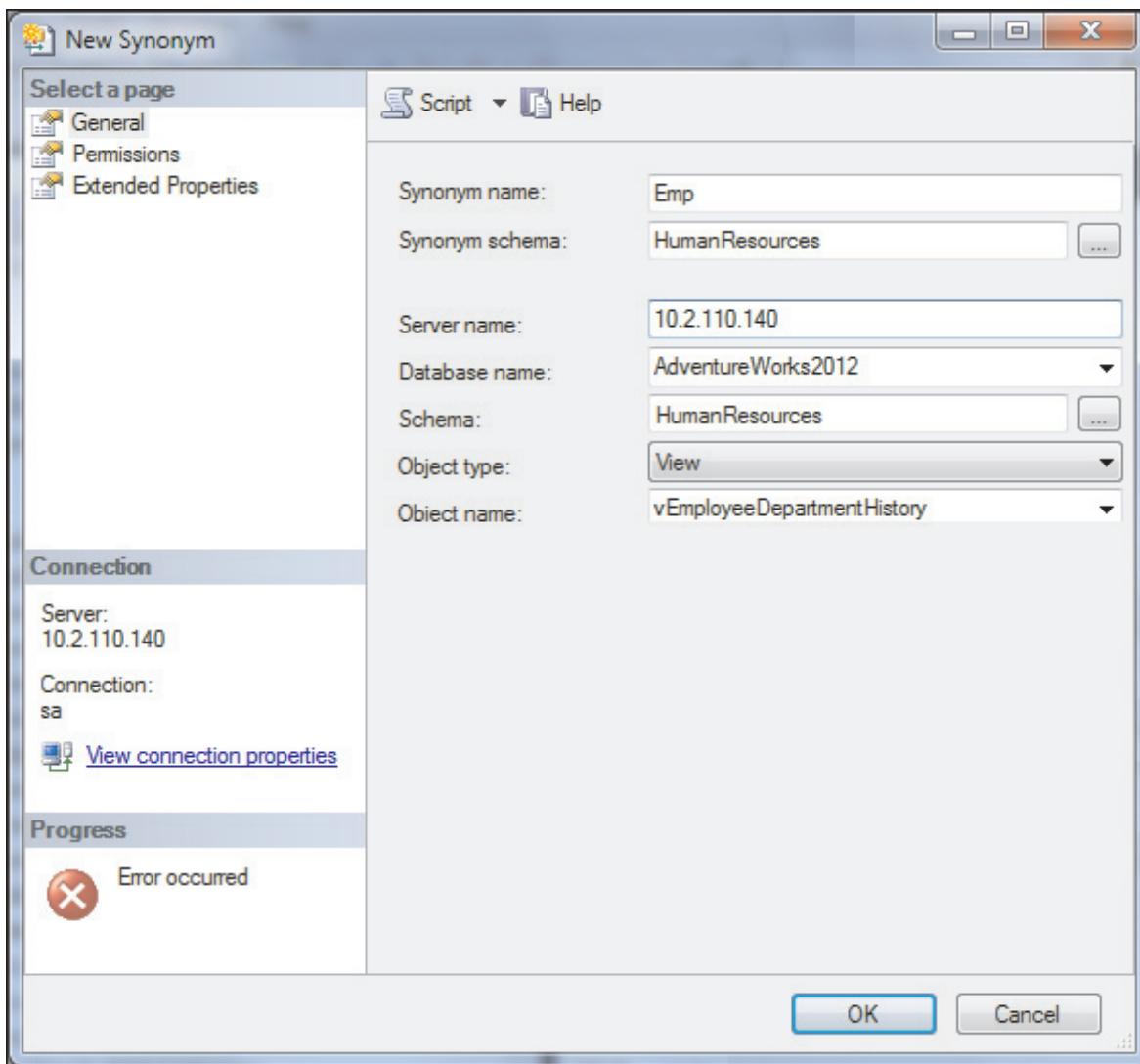
- Trong **Object Explorer**, mở rộng cơ sở dữ liệu nơi bạn muốn tạo ra một từ đồng nghĩa mới.

2. Chọn thư mục **Synonyms**, kích chuột phải vào nó và sau đó, bấm vào **New Synonym...** như được trình bày trong hình 13.3.



Hình 13.3: Tạo ra một đồng nghĩa mới

3. Trong hộp thoại **New Synonym**, cung cấp thông tin như được trình bày trong hình 13.4.



Hình 13.4: Thêm chi tiết vào hộp thoại New Synonym

trong đó:

Synonym name: là tên mới cho đối tượng. Ở đây, **Emp** là tên.

Synonym schema: là tên mới cho đối tượng lược đồ. Ở đây, cùng một tên lược đồ **HumanResources** được sử dụng cho từ đồng nghĩa và loại đối tượng.

Server name: là tên của máy chủ sẽ được kết nối. Ở đây, tên máy chủ được chỉ ra là **10.2.110.140**.

Database name: là tên cơ sở dữ liệu để kết nối đối tượng. Ở đây, **AdventureWorks2012** là tên cơ sở dữ liệu.

Schema: là lược đồ sở hữu đối tượng.

Object type và **Object name**: tương ứng là loại đối tượng và tên. Ở đây, loại đối tượng được lựa chọn là view và tên đối tượng tham khảo từ đồng nghĩa là vEmployeeDepartmentHistory.

Để tạo ra một từ đồng nghĩa sử dụng Transact-SQL, thực hiện các bước sau:

1. Kết nối với Database Engine.
2. Bấm vào **New Query** trong thanh tiêu chuẩn.
3. Viết truy vấn để tạo ra từ đồng nghĩa trong cửa sổ truy vấn.

Sau đây là cú pháp để tạo ra từ đồng nghĩa.

Cú pháp:

```
CREATE SYNONYM [ schema_name_1. ] synonym_name FOR <object>

<object> ::=

{
    [ server_name. [ database_name ] . [ schema_name_2 ] . ] database_name . [
    schema_name_2 ] . [ schema_name_2 ] object_name
}
```

trong đó:

schema_name_1: chỉ ra rằng lược đồ trong đó đồng nghĩa được tạo ra.

synonym_name: chỉ ra tên từ đồng nghĩa mới.

server_name: chỉ ra tên máy chủ, nơi đối tượng cơ sở được đặt.

database_name: chỉ ra tên cơ sở dữ liệu nơi đối tượng cơ sở được đặt.

schema_name_2: chỉ ra tên giản đồ của đối tượng cơ sở.

object_name: chỉ ra tên đối tượng cơ sở, được tham chiếu bởi từ đồng nghĩa.

Đoạn mã 6 tạo ra một từ đồng nghĩa từ bảng hiện có.

Đoạn mã 6:

```
USE tempdb;
GO
CREATE SYNONYM MyAddressType
FOR AdventureWorks2012.Person.AddressType;
GO
```

Trong đoạn mã này, một từ đồng nghĩa được tạo ra từ một bảng hiện có hiện diện trong cơ sở dữ liệu AdventureWorks2012.

4. Bấm vào **Execute** trên thanh công cụ để hoàn thành việc tạo ra từ đồng nghĩa.

13.4 Các câu lệnh luồng chương trình

Có nhiều loại chương trình khác nhau của các hàm và câu lệnh luồng chương trình được hỗ trợ bởi Transact-SQL. Một số trong số này như sau:

→ Ngôn ngữ Điều khiển luồng Transact-SQL

Ngôn ngữ Điều khiển luồng xác định luồng thực thi các câu lệnh Transact-SQL, khối câu lệnh, hàm do người dùng định nghĩa, và các thủ tục lưu trữ.

Theo mặc định, các câu lệnh Transact-SQL được thực hiện tuần tự, theo thứ tự chúng xảy ra. Các phần tử ngôn ngữ điều khiển luồng cho phép các câu lệnh được thực hiện theo một thứ tự cụ thể, có liên quan đến nhau, và làm cho phụ thuộc lẫn nhau sử dụng các cấu trúc tương tự như các ngôn ngữ lập trình.

Bảng 13.3 liệt kê một số từ khóa ngôn ngữ điều khiển luồng Transact-SQL.

Ác từ khóa ngôn ngữ điều khiển luồng
RETURN
THROW
TRY....CATCH
WAITFOR
WHILE
BEGIN....END
BREAK
CONTINUE
GOTO label
IF...ELSE

Bảng 13.3: Từ khóa

→ BEGIN....END

Những câu lệnh BEGIN...END bao quanh một loạt các câu lệnh Transact-SQL để nhóm các câu lệnh Transact-SQL được thực thi.

Sau đây là cú pháp cơ bản cho câu lệnh BEGIN và END.

Cú pháp:

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

trong đó:

{ sql_statement | statement_block }: Là bất kỳ câu lệnh Transact-SQL hợp lệ nào được định nghĩa sử dụng một khối câu lệnh.

Đoạn mã 7 trình bày việc sử dụng các câu lệnh BEGIN và END.

Đoạn mã 7:

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;
GO
IF @@TRANCOUNT = 0
BEGIN
    SELECT FirstName, MiddleName
    FROM Person.Person WHERE LastName = 'Andy';
    ROLLBACK TRANSACTION;
    PRINT N'Việc quay lui giao tác hai lần sẽ tạo ra lỗi.';
END;
ROLLBACK TRANSACTION;
PRINT N'Dã quay lui giao tác.';
GO
```

Trong đoạn mã này, các câu lệnh BEGIN và END mô tả một chuỗi các câu lệnh Transact-SQL được thực thi với nhau. Giả sử BEGIN và END không được đưa vào, khi đó, các câu lệnh ROLLBACK TRANSACTION sẽ thực thi và cả hai thông báo PRINT sẽ được hiển thị.

→ IF...ELSE

Câu lệnh IF...ELSE bắt buộc một điều kiện về việc thực thi một câu lệnh Transact-SQL. Câu lệnh Transact-SQL được tuân thủ với từ khóa IF và điều kiện này thực hiện chỉ khi điều kiện được thỏa mãn và trả về TRUE. Từ khóa ELSE là một câu lệnh Transact-SQL tùy chọn thực hiện chỉ khi điều kiện IF không được thỏa mãn và trả về FALSE.

Sau đây là cú pháp cho câu lệnh IF...ELSE.

Cú pháp:

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

trong đó:

Boolean_expression: chỉ ra biểu thức trả về giá trị TRUE hoặc FALSE.

{ sql_statement| statement_block }: Là bất kỳ câu lệnh hoặc gộp nhóm câu lệnh Transact-SQL nào được định nghĩa sử dụng khối câu lệnh. Nếu khối câu lệnh không được sử dụng, điều kiện IF hoặc ELSE có thể ảnh hưởng đến hiệu suất của chỉ có một câu lệnh Transact-SQL. Để định nghĩa khối câu lệnh, từ khóa BEGIN và END được sử dụng.

Đoạn mã 8 trình bày việc sử dụng các câu lệnh IF...ELSE.

Đoạn mã 8:

```
USE AdventureWorks2012
GO
DECLARE @ListPrice money;
SET @ListPrice = (SELECT MAX(p.ListPrice)
                  FROM Production.Product AS p
                  JOIN Production.ProductSubcategory AS s
                    ON p.ProductSubcategoryID = s.ProductSubcategoryID
                  WHERE s.[Name] = 'Mountain Bikes');
PRINT @ListPrice
IF @ListPrice < 3000
    PRINT 'Tất cả các sản phẩm trong thẻ loại này có thể được mua với số tiền ít hơn 3000'
ELSE
    PRINT 'Giá cho một số sản phẩm thuộc thẻ loại này vượt quá 3000'
```

Trong đoạn mã này, câu lệnh IF...ELSE được sử dụng để tạo thành một câu lệnh điều kiện. Đầu tiên, biến @ListPrice được định nghĩa và truy vấn được tạo ra để trả về giá niêm yết tối đa của loại sản phẩm Xe đạp leo núi. Sau đó, mức giá này được so sánh với giá trị là 3000 để xác định xem sản phẩm có thể được mua cho một số tiền ít hơn 3000 hay không. Nếu có, một thông báo thích hợp được in sử dụng câu lệnh PRINT đầu tiên. Nếu không, khi đó câu lệnh PRINT thứ hai thực hiện.

→ WHILE

Câu lệnh WHILE chỉ rõ một điều kiện cho thực thi lặp lại của khối câu lệnh. Các câu lệnh được thực thi lặp lại miễn là điều kiện đã chỉ định là đúng. Việc thực thi các câu lệnh trong vòng lặp WHILE có thể được kiểm soát bằng cách sử dụng các từ khóa BREAK và CONTINUE.

Sau đây là cú pháp cho câu lệnh WHILE.

Cú pháp:

```
WHILE Boolean_expression
    { sql_statement | statement_block | BREAK | CONTINUE }
```

trong đó:

Boolean_expression: chỉ ra biểu thức trả về giá trị TRUE hoặc FALSE.

{sql_statement | statement_block}: Là bất kỳ câu lệnh Transact-SQL nào định nghĩa khối câu lệnh.

BREAK: Kết quả trong một đầu ra từ vòng lặp WHILE phía trước nhất. Mỗi câu lệnh xuất hiện sau từ khóa END, đánh dấu sự kết thúc vòng lặp, được thực thi.

CONTINUE: Dẫn đến vòng lặp WHILE được khởi động lại. Các câu lệnh sau từ khóa CONTINUE trong phần thân của vòng lặp không được thực thi.

Đoạn mã 9 trình bày việc sử dụng câu lệnh WHILE.

Đoạn mã 9:

```
DECLARE @flag int
SET @flag = 10
WHILE (@flag <= 95)
BEGIN
    IF @flag % 2 = 0
        PRINT @flag
    SET @flag = @flag + 1
    CONTINUE;
END
GO
```

Sử dụng đoạn mã này, tất cả các số chẵn bắt đầu từ 10 đến 95 được hiển thị. Điều này đạt được bằng cách sử dụng vòng lặp WHILE cùng với câu lệnh IF.

Tương tự như vậy, vòng lặp IF cũng có thể được sử dụng với các truy vấn và câu lệnh Transact-SQL khác.

13.5 Các hàm Transact-SQL

Các hàm Transact-SQL thường được sử dụng như sau:

→ Các hàm xác định và không xác định

Các hàm do người dùng định nghĩa có các thuộc tính định nghĩa khả năng của Công cụ cơ sở dữ liệu SQL Server. Công cụ sở dữ liệu được sử dụng để tạo chỉ mục kết quả của hàm thông qua các cột tính để hàm này gọi hoặc các dạng xem có chỉ mục tham khảo những hàm này. Một thuộc tính như vậy là tính xác định của một hàm.

Hàm xác định trả về cùng một kết quả mỗi khi chúng được gọi với một tập hợp xác định các giá trị đầu vào và chỉ ra cùng một trạng thái của cơ sở dữ liệu. Các hàm không xác định trả về các kết quả khác nhau mỗi lần chúng được gọi với tập hợp đã chỉ định các giá trị đầu vào mặc dù cơ sở dữ liệu được truy cập vẫn giữ nguyên.

Ví dụ, nếu một người dùng gọi hàm DAY() trên một cột cụ thể, nó luôn luôn trả về ngày dạng số cho tham số ngày được truyền vào. Tuy nhiên, nếu người dùng gọi hàm DATENAME(), kết quả không thể dự đoán được vì nó có thể khác nhau mỗi lần, tùy thuộc vào phần nào của ngày được truyền làm đầu vào. Như vậy, ở đây, DAY() là một hàm xác định, trong khi DATENAME() là một hàm không xác định. Tương tự như vậy, RAND (không có bất kỳ giá trị mầm nào), @@TIMETICKS, @@CONNECTIONS, và GETDATE() là không xác định.

Người dùng không thể ảnh hưởng đến tính xác định của các hàm dựng sẵn. Mỗi hàm dựng sẵn là xác định hoặc không xác định tùy thuộc vào cách hàm đó được thực hiện bằng SQL Server.

Bảng 13.4 liệt kê một số các hàm dựng sẵn xác định và không xác định.

Các hàm dựng sẵn xác định	Các hàm dựng sẵn không xác định
POWER	@@TOTAL_WRITE
ROUND	CURRENT_TIMESTAMP
RADIANS	GETDATE
EXP	GETUTCDATE
FLOOR	GET_TRANSMISSION_STATUS
SQUARE	NEWID
SQRT	NEWSEQUENTIALID
LOG	@@CONNECTIONS
YEAR	@@CPU_BUSY
ABS	@@DBTS
ASIN	@@IDLE
ACOS	@@IOBUSY
SIGN	@@PACK_RECEIVED
SIN	@@PACK_SENT

Bảng 13.4: Các hàm dựng sẵn xác định và không xác định

Ngoài ra còn có một số hàm không phải là luôn luôn xác định nhưng bạn có thể sử dụng chúng trong các dạng xem có chỉ mục nếu chúng được đưa ra một cách xác định. Bảng 13.5 liệt kê một số những hàm này.

Hàm	Mô tả
CONVERT	Là xác định chỉ khi một trong những điều kiện này tồn tại: <ul style="list-style-type: none"> ➔ Có một loại nguồn sql_variant. ➔ Có một loại đích sql_variant và loại nguồn là không xác định. ➔ Có loại nguồn hoặc đích của nó là smalldatetime hoặc datetime, có loại nguồn hoặc đích là một chuỗi ký tự, và có một kiểu không xác định đã chỉ ra. Tham số kiểu phải là một hằng số sẽ là xác định.
CAST	Là xác định chỉ khi nó được sử dụng với smalldatetime, sql_variant, hoặc datetime.
ISDATE	Là xác định trừ khi được sử dụng với hàm CONVERT, tham số kiểu CONVERT được chỉ ra, và kiểu không bằng 0, 100, 9, hoặc 109.
CHECKSUM	Là xác định, với ngoại lệ của CHECKSUM(*).

Bảng 13.5: Các hàm xác định

➔ Gọi thủ tục lưu trữ mở rộng từ các hàm

Hàm gọi các thủ tục lưu trữ mở rộng là không xác định vì các thủ tục lưu trữ mở rộng có thể dẫn đến các tác dụng phụ lên cơ sở dữ liệu. Các thay đổi đối với trạng thái toàn cầu của một cơ sở dữ liệu như là một sự thay đổi đến một nguồn lực bên ngoài, hoặc các cập nhật cho bảng, tập tin, hoặc mạng được gọi là tác dụng phụ. Ví dụ, gửi e-mail, hoặc xóa một tập tin có thể gây ra tác dụng phụ. Trong khi thực thi một thủ tục lưu trữ mở rộng từ một hàm do người dùng định nghĩa, người dùng không thể đảm bảo rằng nó sẽ trả về một tập kết quả phù hợp.

Do đó, các hàm do người dùng định nghĩa tạo ra các tác dụng phụ trên cơ sở dữ liệu không được khuyến khích.

→ Các hàm có giá trị vô hướng

Hàm có giá trị vô hướng (SVF) luôn luôn trả về một giá trị int, bit, hoặc string. Kiểu dữ liệu được trả về từ và các tham số đầu vào của SVF có thể là bất kì kiểu dữ liệu nào ngoại trừ text, ntext, image, cursor, và timestamp.

Hàm vô hướng trong dòng có một câu lệnh duy nhất và không có phần thân hàm. Hàm vô hướng đa câu lệnh bao quanh phần thân hàm trong khối BEGIN...END.

→ Các hàm có giá trị bảng

Hàm có giá trị bảng là hàm do người dùng định nghĩa trả về một bảng. Tương tự như một hàm vô hướng trong dòng, hàm có giá trị bảng trong dòng có một câu lệnh duy nhất và không có phần thân hàm.

Đoạn mã 10 trình bày việc sử dụng hàm có giá trị bảng.

Đoạn mã 10:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID (N'Sales.ufn_CustDates', N'IF') IS NOT NULL
    DROP FUNCTION Sales.ufn_ufn_CustDates;
GO
CREATE FUNCTION Sales.ufn_CustDates ()
RETURNS TABLE
AS
RETURN
(
    SELECT A.CustomerID, B.DueDate, B.ShipDate
    FROM Sales.Customer A
    LEFT OUTER JOIN
        Sales.SalesOrderHeader B
    ON
        A.CustomerID = B.CustomerID AND YEAR(B.DueDate) < 2012
);
```

Ở đây, hàm có giá trị bảng trong dòng định nghĩa phép nối bên ngoài bên trái giữa các bảng Sales.Customer và Sales.SalesOrderHeader.

Những bảng này được kết nối dựa trên các id khách hàng. Trong trường hợp này, tất cả các bản ghi từ bảng bên trái và chỉ các bản ghi so khớp từ bảng bên phải được trả về. Sau đó bảng kết quả được trả về từ hàm có giá trị bảng.

Hàm này được gọi ra như được trình bày trong Đoạn mã 11.

Đoạn mã 11:

```
SELECT * FROM Sales.ufn_CustDates();
```

Kết quả sẽ là đầu ra của phép nối được trình bày theo một định dạng bảng.

13.6 Thay đổi các hàm do người dùng định nghĩa

Người dùng có thể sửa đổi các hàm do người dùng định nghĩa trong SQL Server 2012 bằng cách sử dụng Transact-SQL hoặc SSMS. Việc thay đổi các hàm do người dùng định nghĩa không sửa đổi các cấp phép của các hàm, nó cũng sẽ không ảnh hưởng đến bất kỳ thủ tục lưu trữ, trigger, hoặc hàm.

→ **Giới hạn và hạn chế**

ALTER FUNCTION không cho phép người dùng thực hiện những hành động sau:

- Sửa đổi hàm có giá trị vô hướng thành một hàm có giá trị bảng.
- Sửa đổi hàm trong dòng thành một hàm đa câu lệnh.
- Sửa đổi hàm Transact-SQL thành hàm CLR.

→ **Các quyền**

Cho phép ALTER là bắt buộc trên lược đồ hoặc hàm. Nếu hàm chỉ ra loại do người dùng định nghĩa, khi đó cần phải có cho phép EXECUTE trên loại đó.

→ **Sửa đổi một hàm do người dùng định nghĩa sử dụng SSMS**

Người dùng cũng có thể thay đổi các hàm do người dùng định nghĩa sử dụng SSMS.

Để sửa đổi hàm do người dùng định nghĩa sử dụng SSMS, thực hiện các bước sau:

1. Bấm vào dấu cộng (+) bên cạnh cơ sở dữ liệu có chứa hàm sẽ được sửa đổi.
2. Bấm vào dấu cộng (+) bên cạnh thư mục Programmability.
3. Bấm vào dấu cộng (+) bên cạnh thư mục có chứa hàm sẽ được sửa đổi. Có ba loại thư mục như sau:
 - Các hàm có giá trị bảng
 - Các hàm có giá trị vô hướng
 - Các hàm tổng hợp
 - Các hàm hệ thống

4. Bấm chuột phải vào hàm sẽ được sửa đổi và sau đó, chọn **Modify**. Đoạn mã cho hàm này xuất hiện trong cửa sổ trình soạn thảo truy vấn.
5. Trong cửa sổ trình soạn thảo truy vấn, thực hiện các thay đổi cần thiết cho phần thân câu lệnh ALTER FUNCTION.
6. Bấm vào **Execute** trên thanh công cụ để thực hiện câu lệnh ALTER FUNCTION.

➔ **Sửa đổi hàm do người dùng định nghĩa sử dụng Transact-SQL**

Để sửa đổi hàm do người dùng định nghĩa sử dụng Transact-SQL, thực hiện các bước sau:

- Trong **Object Explorer**, kết nối với thể hiện Công cụ cơ sở dữ liệu.
- Trên **thanh Standard**, bấm vào **New Query**.
- Gõ mã ALTER FUNCTION vào **Query Editor**.
- Bấm vào **Execute** trên thanh công cụ để thực hiện câu lệnh ALTER FUNCTION.

Đoạn mã 12 trình bày việc sửa đổi một hàm có giá trị bảng.

Đoạn mã 12:

```
USE [AdventureWorks2012]
GO
ALTER FUNCTION [dbo].[ufnGetAccountingEndDate] ()
RETURNS [datetime]
AS
BEGIN
    RETURN DATEADD(millisecond, -2, CONVERT(datetime, '20040701', 112));
END;
```

13.7 Sư tạo ra các cửa sổ với OVER

Hàm cửa sổ là một hàm áp dụng cho một bộ sưu tập các hàng. Từ 'cửa sổ' được sử dụng để tham khảo tới bộ sưu tập các hàng nơi hàm hoạt động trên đó.

Trong Transact-SQL, mệnh đề OVER được sử dụng để định nghĩa một cửa sổ trong một tập kết quả truy vấn. Việc sử dụng các cửa sổ và mệnh đề OVER với các hàm cung cấp một số lợi thế. Ví dụ, chúng giúp tính toán các giá trị tổng hợp. Chúng còn cho phép số dòng trong một tập kết quả được tạo ra dễ dàng.

13.7.1 Tạo cửa sổ các thành phần

Ba thành phần cốt lõi của việc tạo ra các cửa sổ với mệnh đề OVER như sau:

→ Phân vùng

Phân vùng là một tính năng giới hạn cửa sổ của tính toán gần đây để chỉ những hàng từ tập kết quả chứa cùng một giá trị trong các cột phân vùng như ở hàng hiện có. Nó sử dụng mệnh đề PARTITION BY.

Đoạn mã 13 trình bày việc sử dụng mệnh đề PARTITION BY và OVER với các hàm tổng hợp. Ở đây, việc sử dụng mệnh đề OVER chứng minh là có hiệu quả tốt hơn là sử dụng các truy vấn phụ để tính toán các giá trị tổng hợp.

Đoạn mã 13:

```
USE AdventureWorks2012;
GO
SELECT SalesOrderID, ProductID, OrderQty
    , SUM(OrderQty) OVER (PARTITION BY SalesOrderID) AS Total
    , MAX(OrderQty) OVER (PARTITION BY SalesOrderID) AS MaxOrderQty
FROM Sales.SalesOrderDetail
WHERE ProductId IN(776, 773);
GO
```

Kết quả của mã này được trình bày trong hình 13.5.

	SalesOrderID	ProductID	OrderQty	Total	MaxOrderQty
1	43659	776	1	3	2
2	43659	773	2	3	2
3	43661	776	4	6	4
4	43661	773	2	6	4
5	43664	773	1	1	1
6	43665	773	1	2	1
7	43665	776	1	2	1
8	43667	773	1	1	1
9	43670	773	2	3	2
10	43670	776	1	3	2
11	43672	776	2	2	2

Hình 13.5: Phân vùng

→ Sắp thứ tự

Phần tử sắp thứ tự định nghĩa sắp thứ tự cho tính toán trong phân vùng đó. Trong một phần tử sắp thứ tự SQL tiêu chuẩn tất cả các hàm được hỗ trợ. Trước đây, SQL Server không hỗ trợ cho những phần tử sắp thứ tự với các hàm tổng hợp vì nó chỉ hỗ trợ phân vùng. Trong SQL Server 2012, có sự hỗ trợ cho các phần tử sắp thứ tự với các hàm tổng hợp. Phần tử sắp thứ tự có ý nghĩa khác nhau ở một số mức độ cho các thể loại hàm khác nhau. Với các hàm xếp hạng, sắp thứ tự là tự phát.

Đoạn mã 14 trình bày ví dụ về phần tử sắp thứ tự.

Đoạn mã 14:

```
SELECT CustomerID, StoreID,
RANK() OVER(ORDER BY StoreID DESC) AS Rnk_All,
RANK() OVER(PARTITION BY PersonID
ORDER BY CustomerID DESC) AS Rnk_Cust
FROM Sales.Customer;
```

Đoạn mã này sử dụng hàm RANK () trả về thứ hạng của mỗi hàng trong phân vùng của một tập kết quả. Thứ hạng của một hàng được xác định bằng cách thêm 1 vào số các thứ hạng đến trước hàng được chỉ định. Ví dụ, trong khi sử dụng sắp thứ tự giảm dần, hàm RANK () trả về một cái nhiều hơn số lượng hàng trong phân vùng tương ứng có một giá trị sắp thứ tự lớn hơn cái đã chỉ định.

Hình 13.6 hiển thị kết quả của Đoạn mã 14.

	CustomerID	StoreID	Rnk_All	Rnk_Cust
1	701	844	813	1
2	700	1030	633	2
3	699	842	815	3
4	698	640	1009	4
5	697	1032	631	5
6	696	840	817	6
7	695	638	1011	7
8	694	1034	629	8
9	693	838	819	9
10	692	802	855	10
11	691	1036	627	11

Hình 13.6: Sắp thứ tự

Đoạn mã 15 hiển thị một truy vấn với hai tính toán RANK và mệnh đề ORDER BY.

Đoạn mã 15:

```
SELECT TerritoryID, Name, SalesYTD,
```

```
RANK () OVER (ORDER BY SalesYTD DESC) AS Rnk_One,
RANK () OVER (PARTITION BY TerritoryID
               ORDER BY SalesYTD DESC) AS Rnk_Two
FROM Sales.SalesTerritory;
```

Đoạn mã này sử dụng hàm RANK () trả về thứ hạng của mỗi hàng trong phân vùng của một tập kết quả. Nói chung, thứ hạng của một hàng được xác định bằng cách thêm 1 vào số lượng thứ hạng đến trước hàng đã chỉ định. Ở đây trong đoạn mã này, hàm RANK () đầu tiên tạo ra thuộc tính Rnk_One phụ thuộc vào phân vùng mặc định, và hàm RANK thứ hai tạo ra Rnk_Two sử dụng phân vùng rõ ràng bằng TerritoryID.

Hình 13.7 hiển thị các phân vùng được định nghĩa cho một mẫu ba kết quả tính toán trong truy vấn: một giá trị Rnk_One và hai giá trị Rnk_Two.

	TerritoryID	Name	SalesYTD	Rnk_One	Rnk_Two
1	1	Northwest	7887186.7882	2	1
2	2	Northeast	2402176.8476	10	1
3	3	Central	3072175.118	8	1
4	4	Southwest	10510853.8739	1	1
5	5	Southeast	2538667.2515	9	1
6	6	Canada	6771829.1376	3	1
7	7	France	4772398.3078	6	1
8	8	Germany	3805202.3478	7	1
9	9	Australia	5977814.9154	4	1
10	10	United Kingdom	5012905.3656	5	1

Hình 13.7: Phân vùng và xếp hạng

→ Tạo khung

Tạo khung là một tính năng cho phép bạn chỉ ra việc chia nhỏ thêm các hàng trong một phân vùng cửa sổ. Điều này được thực hiện bằng cách chỉ ra ranh giới trên và dưới cho khung cửa sổ thể hiện các hàng cho hàm cửa sổ. Nói một cách đơn giản, khung tương tự như di chuyển cửa sổ trên dữ liệu bắt đầu và kết thúc tại các vị trí quy định. Khung cửa sổ có thể được định nghĩa bằng cách sử dụng các mệnh đề phụ ROW hoặc RANGE và cung cấp ranh giới bắt đầu và kết thúc.

Đoạn mã 16 hiển thị một truy vấn đối với ProductInventory, tính toán tổng lượng chạy cho mỗi sản phẩm và vị trí.

Đoạn mã 16:

```
SELECT ProductID, Shelf, Quantity,
       SUM(Quantity) OVER (PARTITION BY ProductID
```

```

ORDER BY LocationID
ROWS BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW) AS RunQty
FROM Production.ProductInventory;

```

Trong đoạn mã này, hàm cửa sổ áp dụng tổng hợp SUM cho thuộc tính Quantity, phân vùng cửa sổ theo ProductID, sắp thứ tự các hàng phân vùng theo LocationID, và tạo khung các hàng phân vùng phụ thuộc vào thứ tự đã cho giữa cách quãng không bị chặn (không có điểm ranh giới dưới) và hàng hiện tại. Nói cách khác, kết quả sẽ là tổng của tất cả các hàng trước trong khung, bao gồm cả dòng hiện tại. Hình 13.8 hiển thị kết quả của Đoạn mã 16.

	ProductID	Shelf	Quantity	RunQty
1	1	A	408	408
2	1	B	324	732
3	1	A	353	1085
4	2	A	427	427
5	2	B	318	745
6	2	A	364	1109
7	3	A	585	585
8	3	B	443	1028
9	3	A	324	1352
10	4	A	512	512
11	4	B	422	934

Hình 13.8: Tạo khung

13.8 Các hàm cửa sổ

Một số loại hàm cửa sổ khác nhau như sau:

➔ **Các hàm xếp hạng**

Những hàm này trả về giá trị thứ hạng cho mỗi hàng trong một phân vùng. Dựa trên hàm được sử dụng, nhiều hàng sẽ trả về cùng một giá trị như những hàng khác. Các hàm xếp hạng là không xác định.

Bảng 13.6 liệt kê các hàm xếp hạng khác nhau.

Các hàm xếp hạng	Mô tả
NTILE	Trả các hàng trong một phân vùng có thứ tự vào một số nhóm đã cho, bắt đầu từ 1. Đối với mỗi hàng, hàm trả về số nhóm có hàng đó.
ROW NUMBER	Lấy số thứ tự của một hàng trong một phân vùng của một tập kết quả, bắt đầu từ 1 cho hàng đầu tiên trong mỗi phân vùng.
DENSE RANK	Trả về thứ hạng của các hàng trong phân vùng của một tập kết quả, mà không có bất kỳ khoảng trống trong xếp hạng. Thứ hạng của một hàng là một điểm cộng số lượng thứ hạng riêng biệt đến trước hàng được nói đến.

Bảng 13.6: Các hàm xếp hạng

Đoạn mã 17 trình bày việc sử dụng các hàm xếp hạng.

Đoạn mã 17:

```
USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName
    , ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS 'Row Number'
    , NTILE(4) OVER (ORDER BY a.PostalCode) AS 'NTILE'
    , s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson AS s
    INNER JOIN Person.Person AS p
        ON s.BusinessEntityID=p.BusinessEntityID
    INNER JOIN Person.Address AS a
        ON a.AddressID=p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
    AND SalesYTD <> 0;
```

Hàm NTILE () phá vỡ một bộ sưu tập đầu vào đã cho vào N nhóm logic có kích thước bằng nhau. Để xác định bao nhiêu hàng thuộc vào mỗi nhóm, SQL Server phải xác định tổng số hàng trong bộ sưu tập đầu vào. Mệnh đề OVER quyết định thứ tự của các hàng khi họ đã được chia thành các nhóm. Có thể thực hiện gộp nhóm theo một thứ tự và trả về tập kết quả theo thứ tự khác.

Hình 13.9 hiển thị kết quả của Đoạn mã 17.

	FirstName	LastName	Row Number	NTILE	SalesYTD	PostalCode
1	Michael	Blythe	1	1	3763178.1787	98027
2	Linda	Mitchell	2	1	4251368.5497	98027
3	Jillian	Carson	3	1	3189418.3662	98027
4	Garrett	Vargas	4	1	1453719.4653	98027
5	Tsvi	Reiter	5	2	2315185.611	98027
6	Pamela	Anzman-Wolfe	6	2	1352577.1325	98027
7	Shu	Ito	7	2	2458535.6169	98055
8	José	Saraiva	8	2	2604540.7172	98055
9	David	Campbell	9	3	1573012.9383	98055
10	Tete	Mensa-Annan	10	3	1576562.1966	98055
11	Lynn	Tsoflias	11	3	1421810.9242	98055

Hình 13.9: Các hàm xếp hạng

→ Các hàm OFFSET

Những loại hàm bù trừ khác nhau như sau:

- **SWITCHOFFSET**

Hàm này trả về giá trị DATETIMEOFFSET được sửa đổi từ bù trừ múi giờ đã lưu trữ cho bù trừ múi giờ mới cụ thể.

Sau đây là cú pháp cho hàm SWITCHOFFSET.

Cú pháp:

```
SWITCHOFFSET ( DATETIMEOFFSET, time_zone )
```

trong đó:

DATETIMEOFFSET: là một biểu thức được giải thành giá trị datetimeoffset(n).
 time_zone: chỉ ra chuỗi ký tự ở định dạng [+|-]TZH:TZM hoặc một số nguyên có dấu (phút) trình bày độ lệch múi giờ, và được giả định là nhận thức và điều chỉnh giờ làm việc theo mùa hè.

Đoạn mã 18 trình bày việc sử dụng hàm SWITCHOFFSET.

Đoạn mã 18:

```
CREATE TABLE Test
(
```

```

ColDatetimeoffset datetimeoffset
)
GO
INSERT INTO Test
VALUES ('1998-09-20 7:45:50.71345 -5:00');
GO
SELECT SWITCHOFFSET (ColDatetimeoffset, '-08:00')
FROM Test;
GO
--Trả về: 20/09/1998 4:45:50 SA. 7134500 -08:00
SELECT ColDatetimeoffset
FROM Test;
  
```

Hình 13.10 hiển thị đầu ra của Đoạn mã 18.

Results		Messages	
(No column name)			
1	1998-09-20 04:45:50.7134500 -08:00		
ColDatetimeoffset			
1	1998-09-20 07:45:50.7134500 -05:00		

Hình 13.10: Sử dụng hàm SWITCHOFFSET

➔ DATETIMEOFFSETFROMPARTS

Hàm này trả về giá trị datetimeoffset cho ngày và thời gian cụ thể với độ chính xác và độ lệch đã chỉ định.

Sau đây là cú pháp cho DATETIMEOFFSETFROMPARTS.

Cú pháp:

```
DATETIMEOFFSETFROMPARTS ( year, month, day, hour,
                           minute, seconds, fractions, hour_offset,
                           minute_offset, precision )
```

trong đó:

- year: chỉ ra biểu thức số nguyên cho năm.
- month: chỉ ra biểu thức số nguyên cho tháng.
- day: chỉ ra biểu thức số nguyên cho ngày.
- hour: chỉ ra biểu thức số nguyên cho giờ.
- minute: chỉ ra biểu thức số nguyên cho phút.
- seconds: chỉ ra biểu thức số nguyên cho giây.
- fractions: chỉ ra biểu thức số nguyên cho các phân số
- hour_offset: chỉ ra biểu thức số nguyên cho phần giờ của độ lệch múi giờ.
- minute_offset: chỉ ra biểu thức số nguyên cho phần phút của độ lệch múi giờ.
- precision: chỉ ra độ chính xác theo số nguyên của giá trị datetimeoffset để được trả về.

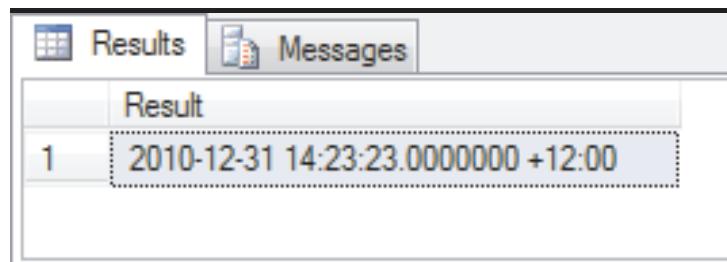
Đoạn mã 19 trình bày việc sử dụng hàm DATETIMEOFFSETFROMPARTS.

Đoạn mã 19:

```
SELECT DATETIMEOFFSETFROMPARTS (2010, 12, 31, 14, 23, 23, 0, 12, 0,
                                7)
AS Result;
```

Đoạn mã này hiển thị giá trị datetimeoffset cho ngày và thời gian đã cho với độ chính xác và độ lệch đã chỉ định.

Hình 13.11 hiển thị đầu ra của Đoạn mã 19.



Hình 13.11: Sử dụng hàm DATETIMEOFFSETFROMPARTS

→ SYSDATETIMEOFFSET

Những hàm trả về giá trị datetimetzoffset (7), trong đó có ngày và thời gian của máy tính trên đó thể hiện của SQL Server đang chạy.

Sau đây là cú pháp cho SYSDATETIMEOFFSET.

Cú pháp:

```
SYSDATETIMEOFFSET ()
```

Đoạn mã 20 hiển thị các định dạng khác nhau được sử dụng bởi các hàm ngày và thời gian.

Đoạn mã 20:

```
SELECT SYSDATETIME () AS SYSDATETIME
      , SYSDATETIMEOFFSET () AS SYSDATETIMEOFFSET
      , SYSUTCDATETIME () AS SYSUTCDATETIME
```

Hình 13.12 trình bày việc sử dụng hàm SYSDATETIMEOFFSET.

	SYSDATETIME	SYSDATETIMEOFFSET	SYSUTCDATETIME
1	2013-02-08 16:08:16.6565247	2013-02-08 16:08:16.6565247 +05:30	2013-02-08 10:38:16.6565247

Hình 13.12: Sử dụng hàm SYSDATETIMEOFFSET

→ Các hàm phân tích

SQL Server 2012 hỗ trợ một số hàm phân tích. Những hàm này tính toán tổng giá trị dựa trên một nhóm các hàng. Các hàm phân tích tính toán tổng số chạy, trung bình di chuyển, hoặc N kết quả đầu trong một nhóm.

Bảng 13.7 liệt kê một số những hàm này.

Hàm	Mô tả
LEAD	Cung cấp quyền truy cập vào dữ liệu từ hàng tiếp theo trong cùng một tập kết quả mà không sử dụng phép tự nối.
LAST_VALUE	Lấy giá trị cuối cùng trong một tập giá trị có thứ tự.
LAG	Cung cấp quyền truy cập vào dữ liệu từ hàng trước trong cùng một tập kết quả mà không sử dụng phép tự nối.
FIRST_VALUE	Lấy giá trị đầu tiên trong một tập giá trị có thứ tự.
CUME_DIST	Tính toán phân phối lũy của một giá trị trong một nhóm các giá trị.
PERCENTILE_CONT	Tính phần trăm dựa trên phân phối liên tục của giá trị cột trong SQL.
PERCENTILE_DISC	Tính toán phần trăm cụ thể cho các giá trị đã sắp xếp trong toàn bộ tập hàng hoặc trong các phân vùng riêng biệt của một tập hàng.

Bảng 13.7: Các hàm phân tích

Đoạn mã 21 trình bày việc sử dụng hàm LEAD().

Đoạn mã 21:

```
USE AdventureWorks2012;
GO
SELECT BusinessEntityID, YEAR(QuotaDate) AS QuotaYear, SalesQuota AS
NewQuota,
    LEAD(SalesQuota, 1, 0) OVER (ORDER BY YEAR(QuotaDate)) AS FutureQuota
FROM Sales.SalesPersonQuotaHistory
WHERE BusinessEntityID = 275 and YEAR(QuotaDate) IN ('2007', '2008');
```

Trong đoạn mã này, hàm LEAD() được sử dụng để trả lại sự khác biệt trong các hạn ngạch bán hàng cho một nhân viên cục thể trong những năm tiếp theo.

Đoạn mã 22 trình bày việc sử dụng hàm FIRST_VALUE().

Đoạn mã 22:

```
USE AdventureWorks2012;
GO
SELECT Name, ListPrice,
    FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LessExpensive
FROM Production.Product
WHERE ProductSubcategoryID = 37
```

Trong đoạn mã này, hàm FIRST_VALUE() so sánh các sản phẩm trong thể loại sản phẩm 37 và trả về tên sản phẩm rẻ hơn.

13.9 Kiểm tra tiến bộ của bạn

1. Câu nào sau đây không phải là tính năng kiểm soát việc sử dụng nhiều câu lệnh Transact-SQL cùng một lúc?

(A)	Tập lệnh	(C)	Điều khiển luồng
(B)	Xử lý lỗi	(D)	Biến

2. Điều nào sau đây được sử dụng để đặt và khai báo các biến cục bộ được cung cấp bởi SQL Server?

a.	DECLARE
b.	SET
c.	DELETE
d.	INSERT

(A)	a, d	(C)	a, b
(B)	b, c	(D)	c, d

3. Điều nào sau đây không phải là cấp phép được áp dụng trên một từ đồng nghĩa?

(A)	GRANT	(C)	DELETE
(B)	CONTROL	(D)	UPDATE

4. Đoạn mã nào sau đây sử dụng biến cục bộ để lấy thông tin liên hệ cho các tên gọi bắt đầu bằng Per?

(A)	<pre>USE AdventureWorks2012; GO DECLARE @find varchar(30); DECLARE @find varchar(30) = 'Per%'; SET @find = 'Per%'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID=ph.BusinessEntityID WHERE LastName LIKE @find;</pre>
(B)	<pre>USE AdventureWorks2012; GO DECLARE find varchar(30); DECLARE find varchar(30) = 'Per%'; SET find = 'Per%'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID=ph.BusinessEntityID WHERE LastName LIKE find;</pre>
(C)	<pre>USE AdventureWorks2012; GO @find varchar(30); @find varchar(30) = 'Per%'; SET @find = 'Per%'; SELECT p.LastName, p.FirstName, ph.PhoneNumber FROM Person.Customer AS p JOIN Person.Phone AS ph ON p.BusinessEntityID=ph.BusinessEntityID WHERE LastName LIKE @find;</pre>

(D)

```

USE AdventureWorks2012;
GO
SET @find varchar(30);
SET @find varchar(30) = 'Per%';
SET @find = 'Per';
SELECT p.LastName, p.FirstName, ph.PhoneNumber
FROM Person.Customer AS p
JOIN Person.Phone AS ph ON p.BusinessEntityID=ph.BusinessEntityID
WHERE LastName LIKE @find;
  
```

5. Hàm nào sau đây không phải là hàm không xác định?

(A)	@@PACK_SENT	(C)	@@DBTS
(B)	@@IOBUSY	(D)	IDLE

13.9.1 Câu trả lời

1.	A
2.	C
3.	A
4.	A
5.	D

Tóm tắt

- Transact-SQL cung cấp các phần tử lập trình cơ bản như là các biến, phần tử điều khiển luồng, cấu trúc điều kiện, và vòng lặp.
- Khối lệnh là tập hợp một hoặc nhiều câu lệnh Transact-SQL được gửi như một đơn vị từ một ứng dụng đến máy chủ.
- Biến cho phép người dùng lưu trữ dữ liệu để sử dụng làm đầu vào trong các câu lệnh Transact-SQL khác.
- Các đồng nghĩa đem lại cách để có bí danh cho đối tượng cơ sở dữ liệu có thể tồn tại trên một máy chủ từ xa hoặc cục bộ.
- Các hàm xác định mỗi lần trả về cùng một kết quả mỗi khi chúng được gọi với một tập hợp xác định các giá trị đầu vào và chỉ ra cùng một trạng thái của cơ sở dữ liệu.
- Các hàm không xác định trả về các kết quả khác nhau mỗi lần chúng được gọi với tập hợp đã chỉ định các giá trị đầu vào mặc dù cơ sở dữ liệu được truy cập vẫn giữ nguyên.
- Hàm cửa sổ là một hàm áp dụng cho một bộ sưu tập các hàng.



- Zen Technologies là một công ty hàng đầu trong lĩnh vực dệt nambre nằm ở California. Ban quản lý của công ty muốn đưa ra giải thưởng trung thành với tất cả nhân viên hoàn thành nhiệm kỳ 5 năm trong tổ chức. Sử dụng cùng một bảng **Employee**, tạo ra một khối lệnh Transact-SQL để trả về **EmployeeID**, **FirstName**, **Department**, và **HireDate** của tất cả các nhân viên như vậy. Giả sử rằng ban quản lý cho tăng lương hai mươi lăm phần trăm cho tất cả mọi người hoàn thành một năm trong tổ chức. Chủ tịch hội đồng quản trị của tổ chức muốn tham gia vào cuộc khảo sát lương toàn quốc gia.

Hãy viết khối lệnh để xác định tổng số tiền lương trả cho một bộ phận trong đó sử dụng nhiều hơn một nhân viên. Ban quản lý muốn tìm ra bộ phận đã thuê số lượng người lao động lớn nhất trong năm qua.

Gợi ý:

- Sử dụng hàm DATETIMEOFFSETFROMPARTS.

Phần - 14

Giao tác

Chào mừng bạn đến với phần **Giao tác**.

Phần này giải thích các loại giao tác và thủ tục để thực hiện những giao tác này. Phần này còn mô tả quy trình để kiểm soát và đánh dấu một giao tác, và liệt kê các sự khác biệt giữa giao tác ẩn và rõ ràng. Nó còn giải thích thêm về mức độ cô lập, phạm vi, các loại khóa, và quản lý giao tác.

Trong phần này, bạn sẽ học để:

- ➔ Xác định và mô tả các giao tác
- ➔ Giải thích thủ tục để thực hiện các giao tác
- ➔ Giải thích quy trình kiểm soát các giao tác
- ➔ Giải thích các bước để đánh dấu một giao tác
- ➔ Phân biệt giữa giao tác ẩn và rõ ràng
- ➔ Giải thích các mức độ cô lập
- ➔ Giải thích phạm vi và các loại khóa khác nhau
- ➔ Giải thích quản lý giao tác

14.1 Giới thiệu

Giao tác là một đơn vị công việc. Giao tác thành công chỉ khi tất cả sửa đổi dữ liệu được thực hiện trong một giao tác được gửi và được lưu trong cơ sở dữ liệu vĩnh viễn. Nếu giao tác được quay lui hoặc hủy bỏ, khi đó nó có nghĩa là giao tác đã gặp các lỗi và không thực hiện bất kỳ thay đổi nào trong những nội dung của cơ sở dữ liệu. Do đó, giao tác có thể được triển khai hoặc quay lui.

14.2 Nhu cầu về giao tác

Có rất nhiều trường hợp người dùng cần phải thực hiện nhiều thay đổi dữ liệu trong nhiều bảng cơ sở dữ liệu. Trong nhiều trường hợp, dữ liệu sẽ không phù hợp để thực thi các lệnh riêng lẻ. Giả sử nếu câu lệnh đầu tiên thực hiện chính xác nhưng các câu lệnh khác thất bại thì dữ liệu vẫn còn ở trạng thái không chính xác.

Ví dụ, một kịch bản tốt sẽ là hoạt động chuyển tiền trong hệ thống ngân hàng. Việc chuyển tiền sẽ cần một câu lệnh `INSERT` và hai câu lệnh `UPDATE`. Đầu tiên, người dùng phải tăng số dư tài khoản đích và sau đó, giảm số dư của tài khoản nguồn. Người dùng phải xem xét xem các giao tác có được triển khai không và có cùng những thay đổi được thực hiện cho tài khoản nguồn và tài khoản đích.

→ Định nghĩa các giao tác

Một đơn vị công việc hợp lý phải thể hiện bốn thuộc tính, được gọi là các thuộc tính nguyên tử, nhất quán, cô lập, và độ bền (ACID), để đủ điều kiện là một giao tác.

- **Tính nguyên tử:** Nếu giao tác có nhiều phép tính thì tất cả phải được thực hiện. Nếu bất kỳ phép tính nào trong nhóm thất bại thì nó nên được quay lui.
- **Tính nhất quán:** Chuỗi các phép tính phải nhất quán.
- **Cô lập:** Những phép tính được thực hiện phải được phân lập từ những phép tính khác trên cùng một máy chủ hoặc trên cùng một cơ sở dữ liệu.
- **Độ bền:** Những phép tính được thực hiện trên cơ sở dữ liệu phải được lưu lại và lưu trữ trong cơ sở dữ liệu vĩnh viễn.

→ Thực hiện các giao tác

SQL Server hỗ trợ các giao tác trong một số chế độ. Một số chế độ này như sau:

- **Tự động thực hiện giao tác:** Mỗi câu lệnh một dòng được thực hiện tự động ngay sau khi nó hoàn thành. Ở chế độ này, người ta không cần phải viết bất kỳ câu lệnh cụ thể nào để bắt đầu và kết thúc các giao tác. Đây là chế độ mặc định cho SQL Server Database Engine.
- **Các giao tác rõ ràng:** Mọi giao tác bắt đầu một cách rõ ràng với câu lệnh `BEGIN TRANSACTION` và kết thúc với giao tác `ROLLBACK` hoặc `COMMIT`.
- **Các giao tác ẩn:** Một giao tác mới sẽ tự động bắt đầu khi giao tác trước đó hoàn thành và mọi giao tác được hoàn thành một cách rõ ràng bằng cách sử dụng câu lệnh `ROLLBACK` hoặc `COMMIT`.

- Các giao tác có phạm vi theo khối lệnh:** Những giao tác này có liên quan đến Nhiều Tập hợp Kết quả Hoạt động (MARS - Multiple Active Result Sets). Bất kỳ giao tác ẩn hay rõ ràng bắt đầu trong một phiên MARS là một giao tác có phạm vi theo khối lệnh. Giao tác có phạm vi theo khối lệnh được quay lui khi khối lệnh hoàn thành tự động được quay lui từ SQL Server.

→ Các giao tác mở rộng khối lệnh

Những câu lệnh giao tác xác định khối mã sẽ thất bại hoặc thành công và cung cấp cơ sở nơi công cụ cơ sở dữ liệu có thể hoàn tác hoặc quay lui những phép tính đó. Những lỗi gặp phải trong quá trình thực thi khối lệnh đơn giản có khả năng thành công một phần, mà không phải là kết quả mong muốn. Điều này còn dẫn đến sự thiếu nhất quán trong các bảng và cơ sở dữ liệu. Để khắc phục điều này, người dùng có thể thêm mã để xác định khối lệnh như một giao tác và đặt khối lệnh giữa BEGIN TRANSACTION và COMMIT TRANSACTION. Người dùng có thể thêm mã xử lý lỗi để quay lui giao tác trong trường hợp có lỗi. Mã xử lý lỗi sẽ hoàn tác những thay đổi một phần đã được thực hiện trước khi lỗi đã xảy ra. Bằng cách này, sự thiếu nhất quán trong các bảng và cơ sở dữ liệu có thể được ngăn chặn.

14.3 Kiểm soát các giao tác

Các giao tác có thể được kiểm soát thông qua các ứng dụng bằng cách chỉ ra điểm bắt đầu và kết thúc của một giao tác. Điều này được thực hiện bằng cách sử dụng các hàm API cơ sở dữ liệu hoặc các câu lệnh Transact-SQL.

Giao tác được quản lý ở mức kết nối, theo mặc định. Khi một giao tác được bắt đầu trên một kết nối, tất cả các câu lệnh Transact-SQL được thực hiện trên cùng một kết nối và là một phần của kết nối đó cho đến khi giao tác kết thúc.

14.3.1 Bắt đầu và kết thúc giao tác sử dụng Transact-SQL

Một trong những cách người dùng có thể bắt đầu và kết thúc giao tác là bằng cách sử dụng các câu lệnh Transact-SQL. Người dùng có thể bắt đầu một giao tác trong SQL Server ở chế độ ẩn hay rõ ràng. Chế độ giao tác rõ ràng bắt đầu một giao tác bằng cách sử dụng câu lệnh BEGIN TRANSACTION. Người dùng có thể kết thúc một giao tác bằng cách sử dụng các câu lệnh ROLLBACK hoặc COMMIT.

Sau đây là một số loại câu lệnh giao tác:

→ BEGIN TRANSACTION

Câu lệnh BEGIN TRANSACTION đánh dấu điểm bắt đầu của một giao tác rõ ràng hoặc cục bộ.

Sau đây là cú pháp cho câu lệnh BEGIN TRANSACTION.

Cú pháp:

```
BEGIN { TRAN | TRANSACTION }
  [ { transaction_name | @tran_name_variable }
    [ WITH MARK [ 'description' ] ]
  ]
[ ; ]
```

trong đó:

transaction_name: chỉ ra tên được gán cho giao tác. Cần thực hiện theo những quy tắc cho các mã định danh và giới hạn những mã định danh này dài 32 ký tự.

@tran_name_variable: chỉ ra tên của một biến do người dùng định nghĩa có chứa tên giao tác hợp lệ.

WITH MARK['description']: chỉ ra giao tác được đánh dấu trong nhật ký. Chuỗi mô tả định nghĩa dấu này.

Đoạn mã 1 trình bày cách tạo ra và bắt đầu một giao tác.

Đoạn mã 1:

```
USE AdventureWorks2012;
GO
DECLARE @TranName VARCHAR(30);
SELECT @TranName = 'FirstTransaction';
BEGIN TRANSACTION @TranName;
DELETE FROM HumanResources.JobCandidate
WHERE JobCandidateID = 13;
```

Trong đoạn mã này, tên giao tác được khai báo sử dụng một biến với giá trị **FirstTransaction**. Một giao tác mới với tên này sau đó được tạo ra có một câu lệnh DELETE. Khi giao tác bao gồm câu lệnh một dòng, nó được thực hiện hoàn toàn.

→ COMMIT TRANSACTION

Câu lệnh COMMIT TRANSACTION này đánh dấu sự kết thúc của một giao tác ẩn hay rõ ràng thành công. Nếu @@TRANCOUNT là 1, khi đó, COMMIT TRANSACTION thực hiện tất cả thay đổi dữ liệu được thực hiện trên cơ sở dữ liệu và trở thành một phần vĩnh viễn của cơ sở dữ liệu. Hơn nữa, nó giải phóng các nguồn lực do giao tác nắm giữ và giảm xuống @@TRANCOUNT đến 0. Nếu @@TRANCOUNT lớn hơn 1, khi đó COMMIT TRANSACTION làm giảm @@TRANCOUNT đến 1 và giữ giao tác ở trạng thái hoạt động.

Sau đây là cú pháp cho câu lệnh COMMIT TRANSACTION.

Cú pháp:

```
COMMIT { TRAN | TRANSACTION } [ transaction_name | @tran_name_variable
] ]
[ ; ]
```

trong đó:

`transaction_name`: chỉ ra tên được gán bằng câu lệnh BEGIN TRANSACTION trước. Cần thực hiện theo những quy tắc cho các mã định danh và không cho phép các mã định danh dài 32 ký tự.

`@tran_name_variable`: chỉ ra tên của một biến do người dùng định nghĩa có chứa tên giao tác hợp lệ. Biến có thể được khai báo là kiểu dữ liệu `char`, `varchar`, `nchar`, hoặc `nvarchar`. Nếu có nhiều hơn 32 ký tự được truyền vào biến, khi đó chỉ có 32 ký tự được sử dụng và các ký tự còn lại sẽ bị cắt bớt.

Đoạn mã 2 cho thấy cách thực hiện một giao tác trong bảng `HumanResources.JobCandidate` của cơ sở dữ liệu AdventureWorks2012.

Đoạn mã 2:

```
BEGIN TRANSACTION;
GO
DELETE FROM HumanResources.JobCandidate
WHERE JobCandidateID = 11;
GO
COMMIT TRANSACTION;
GO
```

Đoạn mã này định nghĩa một giao tác sẽ xóa một hồ sơ ứng viên có `JobCandidateID` là 11.

→ COMMIT WORK

Câu lệnh COMMIT WORK đánh dấu sự kết thúc của giao tác.

Sau đây là cú pháp cho câu lệnh COMMIT WORK.

Cú pháp:

```
COMMIT [ WORK ]
[ ; ]
```

COMMIT TRANSACTION và COMMIT WORK là giống hệt nhau ngoại trừ một thực tế là COMMIT TRANSACTION chấp nhận tên giao tác do người dùng định nghĩa.

→ Đánh dấu một giao tác

Đoạn mã 3 trình bày cách đánh dấu một giao tác trong bảng HumanResources.JobCandidate của cơ sở dữ liệu AdventureWorks2012.

Đoạn mã 3:

```
BEGIN TRANSACTION DeleteCandidate
    WITH MARK N'Deleting a Job Candidate';
GO
DELETE FROM HumanResources.JobCandidate
    WHERE JobCandidateID = 11;
GO
COMMIT TRANSACTION DeleteCandidate;
```

Trong đoạn mã này, giao tác có tên là DeleteCandidate được tạo ra và đánh dấu trong nhật ký.

→ ROLLBACK TRANSACTION

Giao tác này quay lui hoặc hủy bỏ một giao tác ẩn hay rõ ràng tới điểm bắt đầu của giao tác, hoặc tới một điểm đã lưu trong giao tác. Điểm đã lưu là một cơ chế để quay lui một số phần của các giao tác. ROLLBACK TRANSACTION được sử dụng để xóa tất cả các sửa đổi dữ liệu được thực hiện từ khi bắt đầu giao tác hoặc tới điểm đã lưu. Nó còn giải phóng các nguồn lực do giao tác nắm giữ.

Sau đây là cú pháp cho câu lệnh ROLLBACK TRANSACTION.

Cú pháp:

```
ROLLBACK { TRAN | TRANSACTION }
    [ transaction_name | @tran_name_variable
    | savepoint_name | @savepoint_variable ]
[ ; ]
```

trong đó:

transaction_name: chỉ ra tên được gán cho câu lệnh BEGIN TRANSACTION. Cần xác nhận những quy tắc cho các mã định danh và không cho phép các mã định danh dài 32 ký tự.

@tran_name_variable: chỉ ra tên của một biến do người dùng định nghĩa có chứa tên giao tác hợp lệ. Biến có thể được khai báo là kiểu dữ liệu char, varchar, nchar, hoặc nvarchar.

savepoint_name: chỉ ra savepoint_name từ câu lệnh SAVE TRANSACTION. Sử dụng savepoint_name chỉ khi một lần quay lui có điều kiện ảnh hưởng đến một phần của giao tác.

@savepoint_variable: chỉ ra tên của một biến điểm đã lưu có chứa tên điểm đã lưu hợp lệ. Biến có thể được khai báo là kiểu dữ liệu char, varchar, nchar, hoặc nvarchar.

Hãy xem xét ví dụ trình bày việc sử dụng ROLLBACK. Giả định rằng một cơ sở dữ liệu có tên **Sterling** đã được tạo ra. Bảng có tên là **ValueTable** được tạo ra trong cơ sở dữ liệu này như được trình bày trong Đoạn mã 4.

Đoạn mã 4:

```
USE Sterling;
GO
CREATE TABLE ValueTable ([value] char)
GO
```

Đoạn mã 5 tạo ra một giao tác chèn 2 bản ghi vào **ValueTable**. Sau đó, nó quay lui giao tác và một lần nữa chèn một bản ghi vào **ValueTable**. Khi câu lệnh **SELECT** được sử dụng để truy vấn bảng, bạn sẽ thấy rằng chỉ có một bản ghi duy nhất có giá trị **C** được hiển thị. Điều này là do các phép tính **INSERT** trước đó đã được quay lui hoặc hủy bỏ.

Đoạn mã 5:

```
BEGIN TRANSACTION
    INSERT INTO ValueTable VALUES('A');
    INSERT INTO ValueTable VALUES('B');
GO
ROLLBACK TRANSACTION
INSERT INTO ValueTable VALUES('C');
SELECT [value] FROM ValueTable;
```

→ ROLLBACK WORK

Câu lệnh này quay lui một giao tác do người dùng quy định tới điểm bắt đầu giao tác.

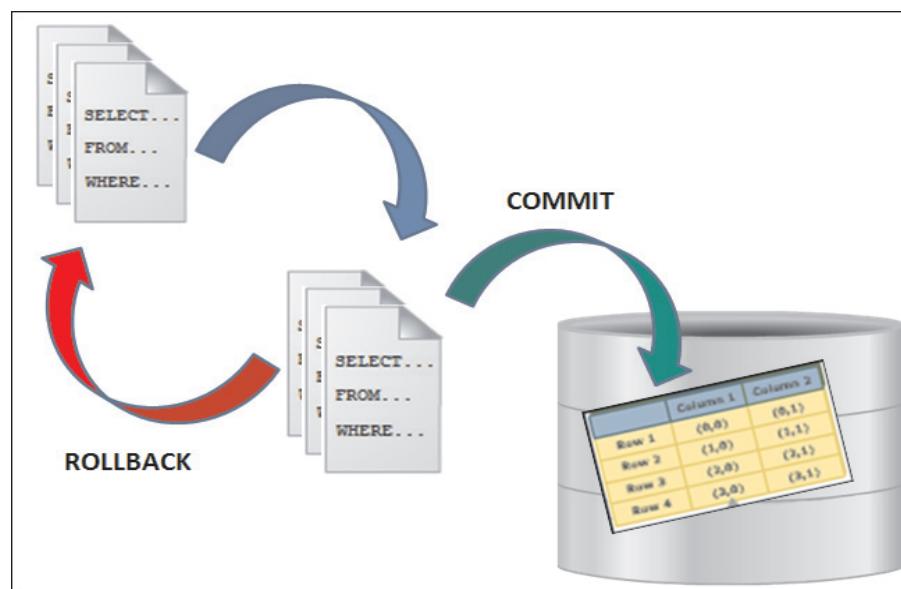
Sau đây là cú pháp cho câu lệnh **ROLLBACK WORK**.

Cú pháp:

```
ROLLBACK [ WORK ]
[ ; ]
```

Từ khóa **WORK** là tùy chọn và ít được sử dụng.

Hình 14.1 hiển thị hoạt động của giao tác.



Hình 14.1: Hoạt động của giao tác

→ SAVE TRANSACTION

Câu lệnh **SAVE TRANSACTION** thiết lập một điểm đã lưu trong một giao tác. Sau đây là cú pháp cho câu lệnh **SAVE TRANSACTION**.

Cú pháp:

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }
[ ; ]
```

trong đó:

savepoint_name: chỉ ra `savepoint_name` đã gán. Những tên này phù hợp với những quy tắc của mã định danh và được giới hạn đến 32 ký tự.

@savepoint_variable: chỉ ra tên của một biến do người dùng định nghĩa có chứa tên điểm đã lưu hợp lệ. Biến có thể được khai báo là kiểu dữ liệu `char`, `varchar`, `nchar`, hoặc `nvarchar`. Hơn 32 ký tự được phép để truyền cho các biến nhưng chỉ có 32 ký tự đầu tiên được sử dụng.

Đoạn mã 6 trình bày cách sử dụng giao tác điểm đã lưu.

Đoạn mã 6:

```

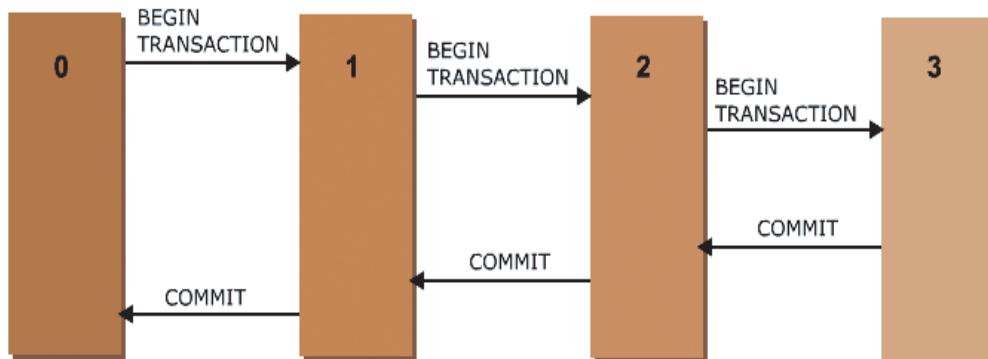
CREATE PROCEDURE SaveTranExample
    @InputCandidateID INT
AS
    DECLARE @TranCounter INT;
    SET @TranCounter = @@TRANCOUNT;
    IF @TranCounter > 0
        SAVE TRANSACTION ProcedureSave;
    ELSE
        BEGIN TRANSACTION;
        DELETE HumanResources.JobCandidate
            WHERE JobCandidateID = @InputCandidateID;
        IF @TranCounter = 0
            COMMIT TRANSACTION;
        IF @TranCounter = 1
            ROLLBACK TRANSACTION ProcedureSave;
GO

```

Trong đoạn mã này, một giao tác điểm đã lưu được tạo ra trong thủ tục lưu trữ. Sau đó cái này sẽ được sử dụng để quay lui chỉ những thay đổi được thực hiện bởi thủ tục lưu trữ nếu một giao tác hoạt động đã bắt đầu trước khi thực hiện thủ tục lưu trữ.

14.4 @@TRANCOUNT

Hàm hệ thống @@TRANCOUNT trả về một số câu lệnh BEGIN TRANSACTION xảy ra trong kết nối hiện tại. Hình 14.2 hiển thị một ví dụ của việc sử dụng @@TRANCOUNT.



Hình 14.2: @@TRANCOUNT

Sau đây là cú pháp cho câu lệnh @@TRANCOUNT.

Cú pháp:

```
@@TRANCOUNT
```

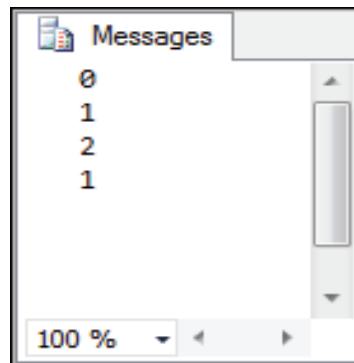
Đoạn mã 7 cho thấy hiệu quả các câu lệnh BEGIN và COMMIT lồng nhau có trên biến @@TRANCOUNT.

Đoạn mã 7:

```
PRINT @@TRANCOUNT
BEGIN TRAN
    PRINT @@TRANCOUNT
    BEGIN TRAN
        PRINT @@TRANCOUNT
        COMMIT
        PRINT @@TRANCOUNT
        COMMIT
    PRINT @@TRANCOUNT
```

Đoạn mã này hiển thị số lần câu lệnh BEGIN TRAN và COMMIT thực hiện trong kết nối hiện tại.

Hình 14.3 hiển thị kết quả của Đoạn mã 7.



Hình 14.3: Kết quả của Đoạn mã 7

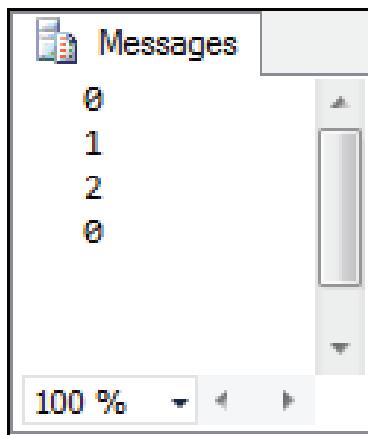
Đoạn mã 8 trình bày hiệu quả các câu lệnh BEGIN và COMMIT lồng nhau có trên biến @@TRANCOUNT .

Đoạn mã 8:

```
PRINT @@TRANCOUNT
BEGIN TRAN
    PRINT @@TRANCOUNT
    BEGIN TRAN
        PRINT @@TRANCOUNT
ROLLBACK
PRINT @@TRANCOUNT
```

Trong trường hợp này, đoạn mã này hiển thị số lần câu lệnh BEGIN TRAN và COMMIT thực hiện trong kết nối hiện tại.

Hình 14.4 hiển thị kết quả của Đoạn mã 8.



Hình 14.4: Kết quả của Đoạn mã 8

14.5 Đánh dấu một giao tác

Người dùng có thể sử dụng các dấu giao tác để phục hồi các bản cập nhật liên quan đã thực hiện tới hai hoặc nhiều cơ sở dữ liệu. Mặc dù sự phục hồi này mất mọi giao tác được thực hiện sau khi đánh dấu được sử dụng làm điểm khôi phục. Đánh dấu một giao tác chỉ có ích khi người dùng sẵn sàng để mất các giao tác đã thực hiện gần đây hoặc đang thử nghiệm các cơ sở dữ liệu có liên quan. Việc đánh dấu các giao tác có liên quan trên cơ sở thường xuyên trong mỗi cơ sở dữ liệu đơn lẻ có liên quan sẽ tạo ra một chuỗi các điểm phục hồi chung trong một cơ sở dữ liệu. Các dấu hiệu giao tác được kết hợp trong các sao lưu nhật ký và cũng được ghi lại trong nhật ký giao tác. Trong trường hợp có thảm họa, người dùng có thể khôi phục mỗi cơ sở dữ liệu đến cùng một đánh dấu giao tác để phục hồi chúng đến điểm phù hợp.

→ Mối quan tâm cho việc sử dụng các giao tác đã đánh dấu

Xem xét các tình huống sau đây trước khi chèn các dấu có tên trong giao tác:

- Bởi dấu giao tác tiêu thụ không gian nhật ký, sử dụng chúng chỉ cho các giao tác đóng một vai trò quan trọng trong chiến lược phục hồi cơ sở dữ liệu.
- Khi giao tác đã đánh dấu được thực hiện, khi đó hàng được chèn vào bảng logmarkhistory trong msdb.
- Nếu một giao tác đã đánh dấu trải dài trên nhiều cơ sở dữ liệu trên các máy chủ khác nhau hoặc trên cùng một máy chủ cơ sở dữ liệu, các dấu phải được ghi lại trong các bản ghi của tất cả các cơ sở dữ liệu bị ảnh hưởng.

14.5.1 Tạo ra các giao tác đã đánh dấu

Để tạo ra một giao tác đã đánh dấu, người dùng có thể sử dụng câu lệnh BEGIN TRANSACTION và mệnh đề WITH MARK [mô tả]. Mô tả tùy chọn là một mô tả bằng văn bản về đánh dấu này. Một tên đánh dấu cho giao tác được tái sử dụng và là bắt buộc. Nhật ký giao tác ghi lại mô tả, tên, người dùng, cơ sở dữ liệu, thông tin ngày giờ, và Số thứ tự nhật ký (LSN) của đánh dấu. Thông tin ngày tháng được sử dụng với tên đánh dấu cho nhận dạng duy nhất của đánh dấu đó.

Để tạo ra một giao tác được đánh dấu trong một tập hợp các cơ sở dữ liệu, những bước sau đây là bắt buộc:

1. Tên giao tác trong câu lệnh BEGIN TRAN và sử dụng mệnh đề WITH MARK.

2. Thực hiện một bản cập nhật chối lại tất cả các cơ sở dữ liệu trong tập hợp này.

Đoạn mã 9 trình bày cách cập nhật ListPrice trong bảng Product của cơ sở dữ liệu AdventureWorks2012.

Đoạn mã 9:

```
USE AdventureWorks2012
GO
BEGIN TRANSACTION ListPriceUpdate
    WITH MARK 'UPDATE Product list prices';
GO

UPDATE Production.Product
    SET ListPrice = ListPrice * 1.20
    WHERE ProductNumber LIKE 'BK-%';
GO

COMMIT TRANSACTION ListPriceUpdate;
GO
```

Đoạn mã 10 trình bày cách khôi phục nhật ký giao tác. Giả định rằng sao lưu có tên AdventureWorksBackups đã được tạo ra.

Đoạn mã 10:

```
USE AdventureWorks2012
GO
BEGIN TRANSACTION ListPriceUpdate
    WITH MARK 'UPDATE Product list prices';
GO

UPDATE Production.Product
    SET ListPrice = ListPrice * 1.20
    WHERE ProductNumber LIKE 'BK-%';
GO

COMMIT TRANSACTION ListPriceUpdate;
GO
```

14.6 Khác biệt giữa giao tác ẩn và rõ ràng

Bảng 14.1 liệt kê những khác biệt giữa giao tác ẩn và rõ ràng.

Ẩn	Rõ ràng
Những giao tác này được duy trì bằng SQL Server cho mỗi và mọi câu lệnh DML và DDL	Những giao tác này được định nghĩa từ các lập trình viên
Những câu lệnh DML và DDL này thực hiện dưới các giao tác ẩn	Các câu lệnh DML được đưa vào để thực hiện như một đơn vị
SQL Server sẽ quay lui toàn bộ câu lệnh	Câu lệnh SELECT không được đưa vào khi họ không sửa đổi dữ liệu

Bảng 14.1 Các khác biệt giữa giao tác ẩn và rõ ràng

14.7 Các mức độ cô lập

Giao tác xác định các mức độ cô lập để định nghĩa mức độ mà một giao tác phải được cô lập khỏi các sửa đổi dữ liệu hoặc tài nguyên được thực hiện bằng các giao tác khác. Mức độ cô lập được định nghĩa theo quan điểm những tác dụng phụ đồng thời nào như số lần đọc bẩn lần được phép. Khi một giao tác thay đổi giá trị và một giao tác thứ hai đọc cùng một giá trị trước khi thay đổi ban đầu đã được thực hiện hoặc quay lui, nó được gọi là đọc bẩn.

Các mức cô lập giao tác kiểm soát như sau:

- ➔ Khi dữ liệu được đọc, có bất kỳ khóa nào được lấy không và loại khóa nào được yêu cầu?
- ➔ Các khóa đọc giữ bao nhiêu thời gian?
- ➔ Nếu hoạt động đọc đang tham khảo một hàng được sửa đổi bằng số số giao tác khác là:
 - Chặn cho đến khi khóa độc quyền trên hàng này là tự do
 - Lấy phiên bản đã thực hiện của hàng tồn tại vào thời điểm khi giao tác hoặc câu lệnh bắt đầu.
 - Đọc sự sửa đổi dữ liệu không được thực hiện.

Khi lựa chọn một mức cô lập giao tác, những khóa ngăn chặn sửa đổi dữ liệu sẽ không bị ảnh hưởng. Giao tác thu được một khóa độc quyền mỗi lần trên mỗi dữ liệu nó sửa đổi. Sau đó, nó giữ khóa đó cho đến khi giao tác được hoàn thành, không phân biệt mức cô lập được thiết lập cho giao tác đó.

Mức cô lập giao tác chủ yếu là mô tả các mức bảo vệ khỏi những tác động đặc biệt của những thay đổi được thực hiện bởi các giao tác khác cho các hoạt động đọc. Mức cô lập thấp hơn làm tăng khả năng một số người dùng truy cập dữ liệu cùng một lúc. Tuy nhiên, nó làm tăng số lượng hiệu ứng đồng thời như là đọc bẩn hoặc bị mất thông tin cập nhật mà người dùng có thể đi qua. Mặt khác, mức cô lập cao hơn làm giảm các loại hiệu ứng đồng thời người dùng có thể gặp phải. Điều này đòi hỏi thêm các tài nguyên hệ thống và làm tăng cơ hội một giao tác chặn giao tác khác.

Việc chọn một mức cô lập phù hợp là dựa trên các yêu cầu toàn vẹn dữ liệu của ứng dụng so với tổng phí của từng mức cô lập. Mức cô lập cao hơn, tuần tự, đảm bảo rằng giao tác sẽ phục hồi cùng một dữ liệu mỗi lần nó lặp lại hoạt động đọc. Sau đó, giao tác thực hiện điều này bằng cách thực hiện một mức khóa được dự kiến sẽ ảnh hưởng đến những người dùng khác trong một hệ thống đa người dùng. Mức cô lập thấp hơn, đọc không ràng buộc, lấy dữ liệu được sửa đổi, và không được thực hiện bằng các giao tác khác. Tất cả các hiệu ứng phụ đồng thời xảy ra trong đọc không ràng buộc, tuy nhiên, không có xác định phiên bản hoặc khóa đọc, do đó, chi phí được giảm thiểu.

Bảng 14.2 liệt kê những hiệu ứng đồng thời được cho phép bằng các mức cô lập khác nhau.

Mức cô lập	Đọc bẩn	Đọc không lặp lại được
Đọc đã thực hiện	Không	Có
Đọc không ràng buộc	Có	Không
Ảnh	Không	Không
Đọc lặp lại	Không	Không
Có thể xếp theo thứ tự	Không	Không

Bảng 14.2: Các mức cô lập

Các giao tác cần phải thực thi ở mức cô lập của ít nhất là đọc lặp lại, ngăn chặn các lần cập nhật bị mất xảy ra khi hai giao tác mỗi cái lấy cùng một hàng, và sau đó cập nhật hàng đó, phụ thuộc vào các hàng ban đầu được lấy.

14.8 Phạm vi và các loại khóa khác nhau

Công cụ cơ sở dữ liệu SQL Server khóa những tài nguyên sử dụng các chế độ khóa khác nhau, trong đó xác định những tài nguyên có thể truy cập đến các giao tác đồng thời.

Bảng 14.3 liệt kê các chế độ khóa tài nguyên được sử dụng bởi Công cụ cơ sở dữ liệu.

Chế độ khóa	Mô tả
Update	Được sử dụng trên các tài nguyên sẽ được cập nhật.
Shared	Được sử dụng cho các hoạt động đọc mà không thay đổi dữ liệu như câu lệnh SELECT.
Intent	Được sử dụng để thiết lập một hệ thống phân cấp của các khóa.
Exclusive	Được sử dụng cho hoạt động sửa đổi dữ liệu INSERT, UPDATE, hoặc DELETE.
BULK UPDATE	Được sử dụng trong khi sao chép dữ liệu với số lượng lớn vào bảng.
Schema	Được sử dụng khi hoạt động phụ thuộc vào lược đồ bảng.

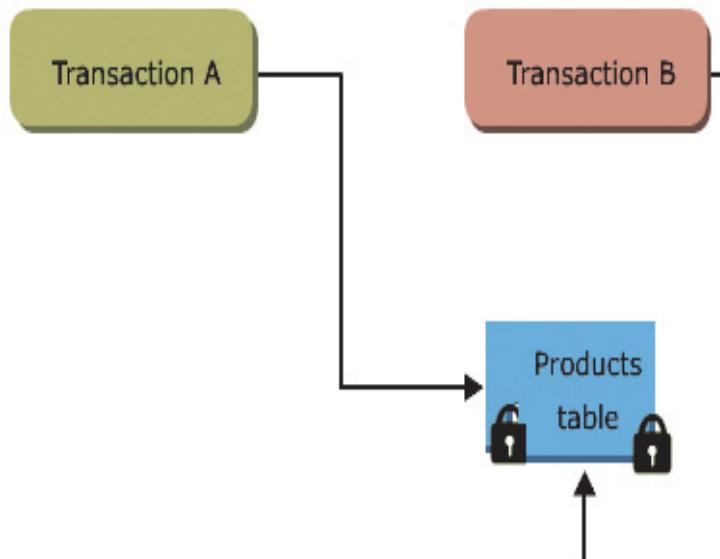
Bảng 14.3: Các chế độ khóa

Những loại khóa khác nhau như sau:

→ Khóa cập nhật

Những khóa này tránh các dạng bế tắc phổ biến. Trong một giao tác tuần tự, giao tác này sẽ đọc dữ liệu, có được một khóa chia sẻ trên hàng đó hoặc một trang, và sửa đổi dữ liệu đòi hỏi chuyển đổi khóa thành một khóa độc quyền. Khi hai giao tác có được một khóa chia sẻ trên một tài nguyên và thử cập nhật dữ liệu cùng một lúc, cùng một giao tác cố gắng chuyển đổi khóa thành một khóa độc quyền. Chế độ chia sẻ để chuyển đổi khóa độc quyền nên chờ khi khóa độc quyền cho một giao tác không tương thích với khóa chế độ chia sẻ của giao tác khác chờ xảy ra. Tương tự như vậy, giao tác thứ hai sẽ cố gắng để có được một khóa độc quyền để cập nhật. Khi cả hai giao tác được chuyển sang khóa độc quyền và mỗi cái chờ cho giao tác khác nhả chế độ khóa chia sẻ của mình, bế tắc xảy ra.

Hình 14.5 minh họa khái niệm về bế tắc như vậy.



Hình 14.5: Bế tắc

Để tránh bế tắc này, cập nhật các khóa được sử dụng. Chỉ có một giao tác mỗi lần có thể có được một khóa cập nhật cho một tài nguyên. Khi giao tác sửa đổi tài nguyên, khi đó khóa cập nhật được chuyển thành khóa độc quyền.

→ Khóa chia sẻ

Những khóa này cho phép các giao tác song song để đọc tài nguyên dưới sự kiểm soát đồng thời bị quan. Các giao tác có thể thay đổi dữ liệu trong khi các khóa chia sẻ tồn tại trên tài nguyên này. Khóa chia sẻ được nhả ra trên một tài nguyên một khi hoạt động đọc hoàn tất, ngoại trừ mức cô lập được đặt thành đọc lặp lại, hoặc cao hơn.

→ Khóa độc quyền

Những khóa này ngăn chặn truy cập đến các tài nguyên của các giao tác đồng thời. Bằng cách sử dụng một khóa độc quyền, không có giao tác khác có thể thay đổi dữ liệu và các hoạt động đọc diễn ra chỉ thông qua mức cô lập đọc không ràng buộc hoặc gọi ý NOLOCK. Các câu lệnh DML như là INSERT, DELETE, và UPDATE kết hợp các hoạt động sửa đổi và đọc. Những câu lệnh này đầu tiên thực hiện một hoạt động đọc để có dữ liệu trước khi sửa đổi những câu lệnh này. Các câu lệnh DML thường yêu cầu các khóa độc quyền và chia sẻ. Ví dụ, nếu người dùng muốn sử dụng câu lệnh UPDATE rằng sửa đổi hàng trong một bảng phụ thuộc vào phép nối với bảng khác. Vì vậy, những câu lệnh cập nhật yêu cầu khóa chia sẻ trên các hàng đọc từ bảng nối và yêu cầu các khóa độc quyền trên những hàng đã sửa đổi.

→ Khóa ý định

Công cụ cơ sở dữ liệu sử dụng các khóa ý định để bảo vệ và đặt một khóa độc quyền hoặc chia sẻ trên tài nguyên ở mức thấp hơn trong hệ thống phân cấp khóa. Tên khóa ý định được đưa ra bởi vì chúng có được trước khóa ở mức thấp và do đó, chỉ ra ý định để đặt các khóa ở mức thấp. Khóa ý định là hữu ích cho hai mục đích:

- Để ngăn chặn các giao tác khác không thay đổi tài nguyên mức cao hơn theo một cách sẽ làm mất hiệu lực khóa này ở mức thấp hơn.
- Để nâng cao hiệu quả của Công cụ cơ sở dữ liệu để xác định các xung đột khóa đó là ở mức cao hơn của độ chi tiết.

Ví dụ, khóa ý định chia sẻ được yêu cầu ở mức bảng trước khi yêu cầu các khóa chia sẻ trên các hàng hoặc trang bên trong bảng này. Thiết lập khóa ý định ở mức bảng bảo vệ giao tác khác sau đó không có được khóa độc quyền trên bảng có chứa các trang. Các khóa ý định còn có Độc quyền ý định (IX), Chia sẻ ý định (IS), và Chia sẻ với độc quyền ý định (SIX). Bảng 14.4 liệt kê các chế độ khóa trong Khóa ý định.

Chế độ khóa	Mô tả
Chia sẻ ý định (IS)	Bảo vệ khóa chia sẻ đã yêu cầu trên một số tài nguyên đang thấp hơn trong hệ thống phân cấp.
Độc quyền ý định (IX)	Bảo vệ khóa độc quyền đã yêu cầu trên một số tài nguyên thấp hơn trong hệ thống phân cấp. IX là một siêu tập hợp của IS, bảo vệ các khóa chia sẻ yêu cầu trên các tài nguyên mức thấp hơn.
Chia sẻ với độc quyền ý định (SIX)	Bảo vệ khóa chia sẻ đã yêu cầu trên tất cả các tài nguyên thấp hơn trong hệ thống phân cấp và khóa độc quyền ý định trên một số tài nguyên mức thấp hơn. Khóa IS đồng thời được cho phép ở tài nguyên mức cao nhất.
Cập nhật ý định (IU)	Bảo vệ khóa cập nhật đã yêu cầu trên tất cả tài nguyên thấp hơn trong hệ thống phân cấp. Khóa IU chỉ được sử dụng trên các tài nguyên trang. Khóa IU được chuyển đổi sang khóa IX nếu một hoạt động cập nhật diễn ra.
Cập nhật ý định chia sẻ (SIU)	Cung cấp sự kết hợp của khóa S và IU, là kết quả của việc có được những khóa này một cách riêng biệt và đồng thời giữ cả hai khóa.
Độc quyền ý định cập nhật (UIX)	Cung cấp sự kết hợp của khóa U và IX, là kết quả của việc có được những khóa này một cách riêng biệt và đồng thời giữ cả hai khóa.

Bảng 14.4: Các chế độ khóa trong Khóa ý định

→ Khóa cập nhật số lượng lớn

Khóa cập nhật số lượng lớn được công cụ cơ sở dữ liệu sử dụng. Những khóa này được sử dụng khi một số lượng lớn các dữ liệu được sao chép vào một bảng (các hoạt động sao chép số lượng lớn) và tùy chọn table lock on bulk load được đặt hoặc gợi ý TABLOCK được chỉ ra sử dụng sp_tableoption.

Những khóa này cho phép nhiều phân luồng tải dữ liệu số lượng lớn liên tục trong cùng một bảng, tuy nhiên, ngăn chặn các quá trình khác mà không phải là dữ liệu tải số lượng lớn không truy cập vào bảng này.

→ Khóa lược đồ

Khóa sửa đổi lược đồ được sử dụng bởi Công cụ cơ sở dữ liệu trong khi thực hiện một hoạt động DDL bảng như thả bỏ một bảng hoặc cột. Khóa lược đồ ngăn chặn truy cập đồng thời vào bảng, có nghĩa là khóa lược đồ chặn tất cả các hoạt động bên ngoài cho đến khi khóa nhả ra.

Một số hoạt động DML như cắt bớt một bảng sử dụng Khóa lược đồ để ngăn chặn truy cập vào các bảng bị ảnh hưởng của các hoạt động đồng thời.

Khóa ổn định giản đồ được sử dụng bởi công cụ cơ sở dữ liệu trong khi biên dịch và thực thi những truy vấn này. Những khóa ổn định này không chặn bất kỳ khóa giao tác nào bao gồm cả những khóa độc quyền. Do đó, các giao tác bao gồm khóa X trên bảng này tiếp tục thực thi trong khi biên dịch truy vấn. Mặc dù vậy, những hoạt động DML và DDL đồng thời có được khóa sửa đổi lược đồ không thực hiện trên những bảng này.

→ Khóa dải khóa

Những loại khóa này bảo vệ một bộ sưu tập các hàng đang ngầm hiện diện trong một tập bản ghi đang được đọc bằng một câu lệnh Transact-SQL trong khi sử dụng mức cô lập giao tác tuần tự. Khóa dải khóa ngăn chặn các lần đọc không có thực. Bằng cách bảo vệ phạm vi của các khóa giữa các hàng, chúng còn ngăn chặn việc xóa bỏ hoặc chèn không có thực trong tập bản ghi truy cập một giao tác.

14.9 Quản lý giao tác

Mỗi câu lệnh đơn lẻ được thực thi, theo mặc định, là dạng giao tác trong SQL Server. Nếu một câu lệnh SQL đơn lẻ được phát hành, khi đó, một giao tác ngầm được bắt đầu. Nó có nghĩa là câu lệnh sẽ bắt đầu và hoàn thành ngầm. Khi người dùng sử dụng các lệnh BEGIN TRAN/COMMIT TRAN rõ ràng, họ có thể nhóm chúng lại với nhau như là một giao tác rõ ràng. Những câu lệnh này sẽ thành công hoặc thất bại. SQL Server thực hiện một số mức cô lập giao tác đảm bảo những thuộc tính ACID của những giao tác này. Trong thực tế, nó có nghĩa là nó sử dụng các khóa để tạo điều kiện cho truy cập giao tác tới các tài nguyên cơ sở dữ liệu được chia sẻ và còn ngăn ngừa sự giao thoa giữa những giao tác này.

14.10 Nhật ký giao tác

Mỗi cơ sở dữ liệu SQL Server có một nhật ký giao tác, trong đó ghi tất cả các giao tác và sửa đổi cơ sở dữ liệu được thực hiện bởi mỗi giao tác. Nhật ký giao tác sẽ được cắt bớt thường xuyên để giữ cho nó không đầy lên. Việc giám sát dung lượng nhật ký là quan trọng vì có thể có một số yếu tố trì hoãn việc cắt bớt nhật ký.

Nhật ký giao tác là một thành phần hết sức quan trọng của cơ sở dữ liệu và nếu một lỗi hệ thống xảy ra, nhật ký giao tác sẽ được yêu cầu để mang cơ sở dữ liệu tới dữ liệu phù hợp. Không nên di chuyển hoặc xóa nhật ký giao tác cho đến khi người dùng hiểu được hậu quả của việc làm đó. Các hoạt động được hỗ trợ bởi nhật ký giao tác như sau:

- Phục hồi các giao tác riêng lẻ
- Phục hồi các giao tác không đầy đủ khi SQL Server bắt đầu
- Hỗ trợ trùng lắp giao tác
- Các giải pháp khắc phục thảm họa và hỗ trợ tính sẵn sàng cao
- Quay lui tập tin, cơ sở dữ liệu đã phục hồi, nhóm tập tin, hoặc trang phía trước điểm thất bại

→ Cắt bỏ nhật ký giao tác

Việc cắt bớt nhật ký giải phóng không gian trong tập tin nhật ký cho việc tái sử dụng nhật ký giao tác. Sự cắt bớt các nhật ký bắt đầu tự động sau các sự kiện sau đây:

- Trong một mô hình phục hồi đơn giản sau điểm kiểm soát.
- Trong một mô hình phục hồi được ghi số lượng lớn hoặc mô hình phục hồi đầy đủ, nếu điểm kiểm soát được xảy ra kể từ lần sao lưu cuối cùng, sự cắt bớt xảy ra sau lần sao lưu nhật ký.

Có các yếu tố trì hoãn sự cắt bớt nhật ký. Khi các bản ghi nhật ký vẫn hoạt động trong một thời gian dài, sự cắt bớt nhật ký giao tác sẽ trễ và nhật ký giao tác đây lên. Sự cắt bớt nhật ký bị trì hoãn vì nhiều lý do. Người dùng cũng có thể khám phá nếu bất cứ điều gì ngăn cản sự cắt bớt nhật ký bằng cách truy vấn cột `log_reuse_desc` và `log_reuse_wait` của khung nhìn danh mục `sys.databases`.

Bảng 14.5 liệt kê những giá trị của một số những cột này.

Chế độ giá trị <code>log_reuse_wait</code>	Giá trị <code>log_reuse_wait_desc</code>	Mô tả
0	NOTHING	Chỉ ra rằng hiện nay, có nhiều hơn một tập tin nhật ký ảo có thể tái sử dụng.
1	CHECKPOINT	Chỉ ra rằng không có điểm kiểm soát đã xảy ra kể từ lần cắt bớt nhật ký cuối cùng, hoặc phần đầu của nhật ký đã không vượt quá một tập tin nhật ký ảo.
2	LOG_BACKUP	Chỉ ra một bản sao lưu nhật ký được yêu cầu trước khi cắt bớt nhật ký giao tác.

Chế độ giá trị log_reuse_wait	Giá trị log_reuse_wait_desc	Mô tả
3	ACTIVE_BACKUP_OR_RESTORE	Chỉ ra rằng sao lưu dữ liệu hoặc khôi phục đang được tiến hành.
4	ACTIVE_TRANSACTION	Chỉ ra rằng một giao tác đang hoạt động.
5	DATABASE_MIRRORING	Chỉ ra rằng phản chiếu cơ sở dữ liệu bị tạm dừng, hoặc theo chế độ hiệu suất cao, cơ sở dữ liệu phản chiếu là phía sau cơ sở dữ liệu chính đáng kể.

Bảng 14.5: Giá trị của cột log_reuse_wait_desc và log_reuse_wait

14.11 Kiểm tra tiến bộ của bạn

1. Loại giao tác nào sau đây có liên quan đến Nhiều tập kết quả hoạt động?

(A)	Tự xác nhận	(C)	Ẩn
(B)	Rõ ràng	(D)	Có phạm vi theo khối lệnh

2. _____ đánh dấu điểm bắt đầu của một giao tác rõ ràng hoặc cục bộ.

(A)	ROLLBACK TRANSACTION	(C)	COMMIT WORK
(B)	BEGIN TRANSACTION	(D)	COMMIT TRANSACTION

3. Nhận dạng hàm trả về số lượng câu lệnh BEGIN TRANSACTION xảy ra trong kết nối hiện tại.

(A)	@@TRANCOUNTER	(C)	@@TRANCOUNT
(B)	@@ERRORMESSAGE	(D)	@@ERROR

4. Điều nào sau đây không phải là tác dụng đồng thời được cho phép bằng các mức cô lập khác nhau?

(A)	Đọc đã thực hiện	(C)	Đọc lặp lại
(B)	Ảnh	(D)	COMMIT

5. Phù hợp với các loại chế độ khóa tài nguyên trong SQL Server 2012 đối với các mô tả tương ứng của chúng.

	Mô tả		Các chế độ khóa
a.	Được sử dụng trên các tài nguyên sẽ được cập nhật.	1.	Schema
b.	Được sử dụng cho các hoạt động đọc mà không thay đổi dữ liệu như câu lệnh SELECT.	2.	Exclusive
c.	Được sử dụng để thiết lập một hệ thống phân cấp của các khóa.	3.	Intent
d.	Được sử dụng cho các hoạt động sửa đổi dữ liệu INSERT, UPDATE, hoặc DELETE.	4.	Shared
e.	Được sử dụng khi hoạt động phụ thuộc vào lược đồ bảng.	5.	Update

(A)	a-1, b-4, c-2, d-3, e-5	(C)	a-2, b-4, c-3, d-5, e-1
(B)	a-5, b-4, c-3, d-2, e-1	(D)	a-1, b-2, c-3, d-4, e-5

14.11.1 Câu trả lời

1.	D
2.	B
3.	C
4.	D
5.	B

Tóm tắt

- Giao tác là một chuỗi các hoạt động làm việc như một đơn vị đơn lẻ.
- Có thể kiểm soát các giao tác bằng một ứng dụng bằng cách chỉ ra điểm bắt đầu và kết thúc.
- BEGIN TRANSACTION đánh dấu điểm bắt đầu của một giao tác rõ ràng hoặc cục bộ.
- COMMIT TRANSACTION đánh dấu điểm kết thúc của một giao tác ẩn hay rõ ràng thành công.
- ROLLBACK với từ khóa tùy chọn WORK quay lui giao tác do người dùng quy định tới điểm bắt đầu của giao tác.
- @@TRANCOUNT là một hàm hệ thống trả về số lượng câu lệnh BEGIN TRANSACTION xảy ra trong kết nối hiện tại.
- Các mức cô lập được cung cấp bằng giao tác này để mô tả mức độ mà tới đó một giao tác đơn lẻ cần phải được cô lập khỏi những thay đổi được thực hiện bởi các giao tác khác.
- Công cụ cơ sở dữ liệu SQL Server khóa những tài nguyên sử dụng các chế độ khóa khác nhau, trong đó xác định những tài nguyên có thể truy cập đến các giao tác đồng thời.

Hãy thử tự làm

1. Công ty Zamora Electronics có hơn 500 công nhân trong các đơn vị của mình. Một số trong số này là ở cấp cơ sở, trong khi một số là ở cấp cao phụ thuộc vào chuyên môn và số năm kinh nghiệm của họ. Mỗi nhân viên được cho nghỉ hàng năm dựa trên chức vụ. Ban quản trị tại Công ty Zamora Electronics đang có kế hoạch tin học hóa bộ phận nguồn nhân lực của họ và tất cả các dữ liệu liên quan đến người lao động bây giờ sẽ được lưu trữ trong cơ sở dữ liệu SQL Server 2012. Công ty đã thực hiện một số thay đổi trong chính sách nghỉ phép của người lao động và muốn cập nhật cùng như vậy trong các bảng của họ. Giả sử rằng bạn là người quản trị cơ sở dữ liệu và bạn được giao những nhiệm vụ sau đây:

- Tạo ra một giao tác để cập nhật những bản ghi trong bảng này theo chính sách nghỉ phép mới.
- Kiểm tra xem các giao tác có được cập nhật trong bảng thích hợp hay không.
- Kiểm tra xem các giao tác có không được cập nhật. Sau đó, đảm bảo rằng chúng sẽ được quay lui với các thông báo lỗi thích hợp.

Bảng 14.6 liệt kê bảng **EmployeeDetails**.

Tên trường	Loại dữ liệu	Trường khóa	Mô tả
Employee_Id	varchar(5)	Primary Key	Lưu trữ số nhận dạng nhân viên
FirstName	varchar(30)		Lưu trữ tên họ của nhân viên
LastName	varchar(30)		Lưu trữ tên gọi của nhân viên
Address	varchar(60)		Lưu trữ địa chỉ của nhân viên
PhoneNumber	varchar(20)		Số điện thoại của nhân viên, nó có thể là điện thoại cố định hoặc điện thoại di động
Department_Id	varchar(4)		Lưu trữ mã bộ phận của phòng ban nơi nhân viên đó thuộc vào.
Designation	varchar(30)		Lưu trữ chức vụ hoặc vai trò công việc của nhân viên
Salary	money		Lưu trữ lương của nhân viên
Join_date	datetime		Lưu trữ ngày tham gia cho nhân viên
Performance_Rating	int		Lưu trữ xếp hạng của nhân viên

Bảng 14.6: Bảng EmployeeDetails

Phần - 15

Xử lý lỗi

Chào mừng bạn đến với phần **Xử lý lỗi**.

Phần này giới thiệu các kỹ thuật xử lý lỗi trong SQL Server và mô tả việc sử dụng các khối TRY-CATCH. Các hàm và câu lệnh khác nhau có thể giúp thông tin hiển thị thông tin lỗi cũng được đề cập trong phần này.

Đến cuối của phần này, bạn sẽ có thể:

- ➔ Giải thích các loại lỗi khác nhau
- ➔ Giải thích việc xử lý lỗi và thực hiện
- ➔ Mô tả khối TRY-CATCH
- ➔ Giải thích thủ tục để hiển thị thông tin lỗi
- ➔ Mô tả câu lệnh @@ERROR và RAISERROR
- ➔ Giải thích việc sử dụng ERROR_STATE, ERROR_SEVERITY, và ERROR_PROCEDURE
- ➔ Giải thích việc sử dụng ERROR_NUMBER, ERROR_MESSAGE, và ERROR_LINE
- ➔ Mô tả câu lệnh THROW

15.1 Giới thiệu

Xử lý lỗi trong SQL Server đã trở nên dễ dàng thông qua một số kỹ thuật khác nhau. SQL Server đã giới thiệu các tùy chọn có thể giúp bạn xử lý lỗi hiệu quả. Thông thường, không thể nắm bắt các lỗi xảy ra ở đầu cuối của người dùng. SQL Server cung cấp câu lệnh TRY...CATCH giúp xử lý các lỗi có hiệu quả ở nền sau. Ngoài ra còn có một số hàm hệ thống in thông tin liên quan đến lỗi, có thể giúp sửa chữa các lỗi dễ dàng.

15.2 Các loại lỗi

Là người lập trình Transact-SQL, cần phải nhận thức được các loại lỗi khác nhau có thể xảy ra trong khi làm việc với các câu lệnh SQL Server. Bước đầu tiên người ta có thể thực hiện là xác định loại lỗi và sau đó xác định làm thế nào để xử lý hoặc vượt qua nó.

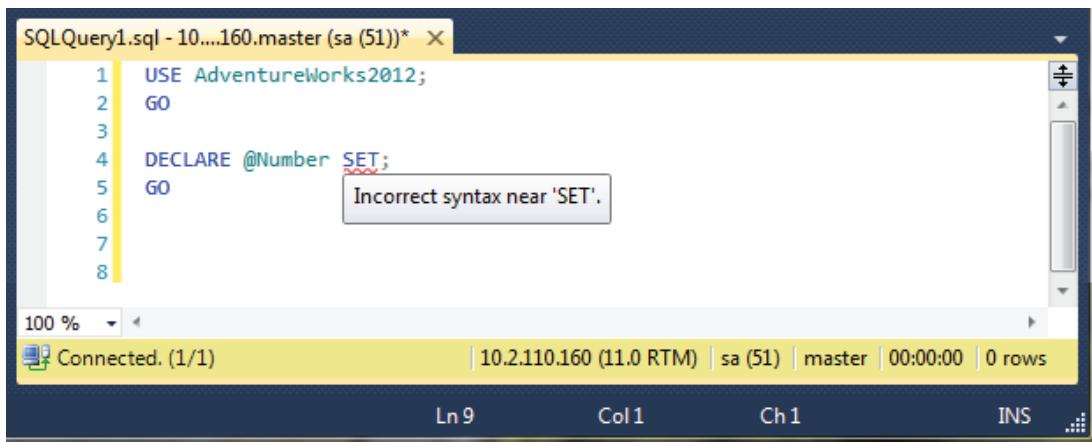
Một số loại lỗi như sau:

→ Lỗi cú pháp

Lỗi cú pháp là lỗi xảy ra khi mã không thể phân tích được bằng SQL Server. Lỗi như vậy được phát hiện bởi SQL Server trước khi bắt đầu quá trình thực hiện khối Transact-SQL hoặc thủ tục lưu trữ.

Một số kịch bản, nơi xảy ra các lỗi cú pháp như sau:

- Nếu người dùng đang gõ một toán tử hoặc một từ khóa được sử dụng một cách sai lầm, trình biên tập mã sẽ hiển thị chú giải công cụ trình bày lỗi này. Hình 15.1 hiển thị ví dụ về lỗi cú pháp.



The screenshot shows a SQL query window titled "SQLQuery1.sql - 10....160.master (sa (51))*". The code entered is:

```

1 USE AdventureWorks2012;
2 GO
3
4 DECLARE @Number SET;
5 GO
6
7
8

```

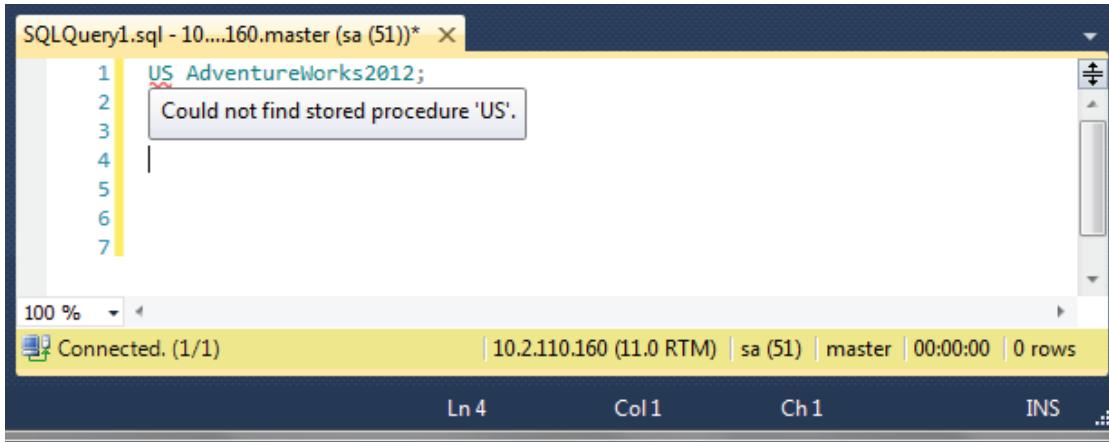
An error message box is displayed at the bottom of the code area, stating "Incorrect syntax near 'SET'." Below the code, the status bar shows "Connected. (1/1)" and "10.2.110.160 (11.0 RTM) | sa (51) | master | 00:00:00 | 0 rows".

Hình 15.1: Lỗi cú pháp

Trong hình 15.1, toán tử SET được sử dụng sai trong câu lệnh Transact-SQL, do đó lỗi cú pháp sẽ nảy sinh.

- Nếu người dùng gõ một từ khóa hoặc một toán tử sai bởi vì người dùng không nhớ việc sử dụng hợp lệ, trình biên tập mã sẽ chỉ ra một cách thích hợp.

Hình 15.2 hiển thị ví dụ.



The screenshot shows a SQL query window titled "SQLQuery1.sql - 10....160.master (sa (51))*". The query is:

```

1 US AdventureWorks2012;
2 Could not find stored procedure 'US'.
3
4
5
6
7

```

The second line contains a syntax error, and a tooltip "Could not find stored procedure 'US'." is displayed over it. Below the query window, the status bar shows "Connected. (1/1)" and connection details: "10.2.110.160 (11.0 RTM) | sa (51) | master | 00:00:00 | 0 rows". The bottom navigation bar includes "Ln 4", "Col 1", "Ch 1", "INS", and other icons.

Hình 15.2: Lỗi sai từ khóa sai

- Nếu người dùng quên gõ một cái gì đó là bắt buộc, trình biên tập mã sẽ hiển thị lỗi khi người dùng thực hiện câu lệnh đó.

Lỗi cú pháp có thể dễ dàng được xác định khi trình biên tập mã chỉ ra chúng. Do đó, những lỗi này có thể dễ dàng khắc phục. Tuy nhiên, nếu người dùng sử dụng một ứng dụng dựa trên lệnh như `sqlcmd`, lỗi chỉ được hiển thị sau khi mã được thực thi.

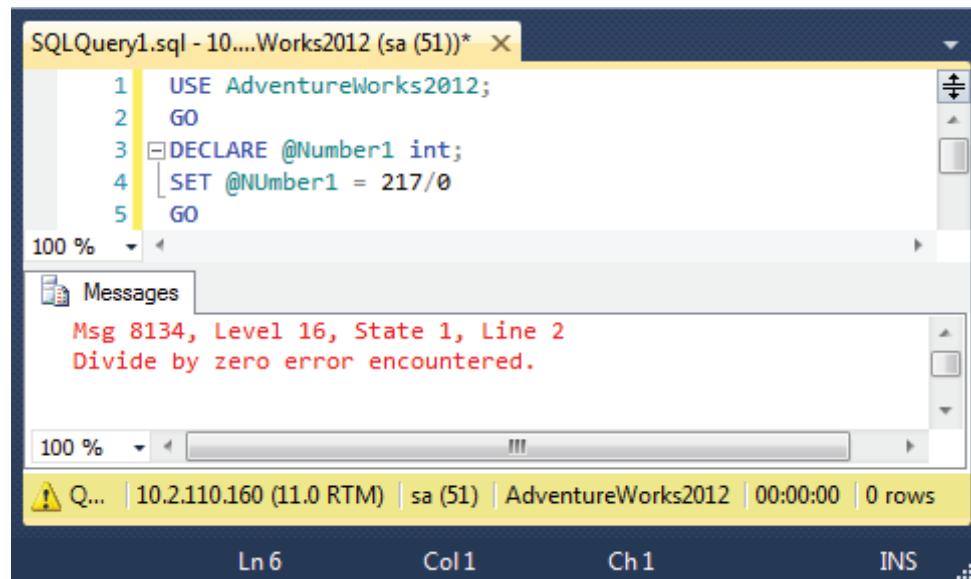
→ **Lỗi thời gian chạy**

Lỗi thời gian chạy là lỗi xảy ra khi ứng dụng cố gắng thực hiện một hành động không được hỗ trợ bởi SQL Server cũng không phải bởi hệ điều hành. Lỗi thời gian chạy đôi khi rất khó để khắc phục bởi chúng không được xác định rõ ràng hoặc bên ngoài đối với cơ sở dữ liệu.

Một số trường hợp các lỗi thời gian chạy có thể xảy ra như sau:

- Thực hiện một phép tính như chia cho 0
- Thử thực thi mã không được định nghĩa rõ ràng

Hình 15.3 trình bày lỗi chia cho số 0.



The screenshot shows a SQL query window titled "SQLQuery1.sql - 10....Works2012 (sa (51))*". The query is:

```

1 USE AdventureWorks2012;
2 GO
3 DECLARE @Number1 int;
4 SET @Number1 = 217/0
5 GO

```

In the "Messages" pane, an error message is displayed in red:

```

Msg 8134, Level 16, State 1, Line 2
Divide by zero error encountered.

```

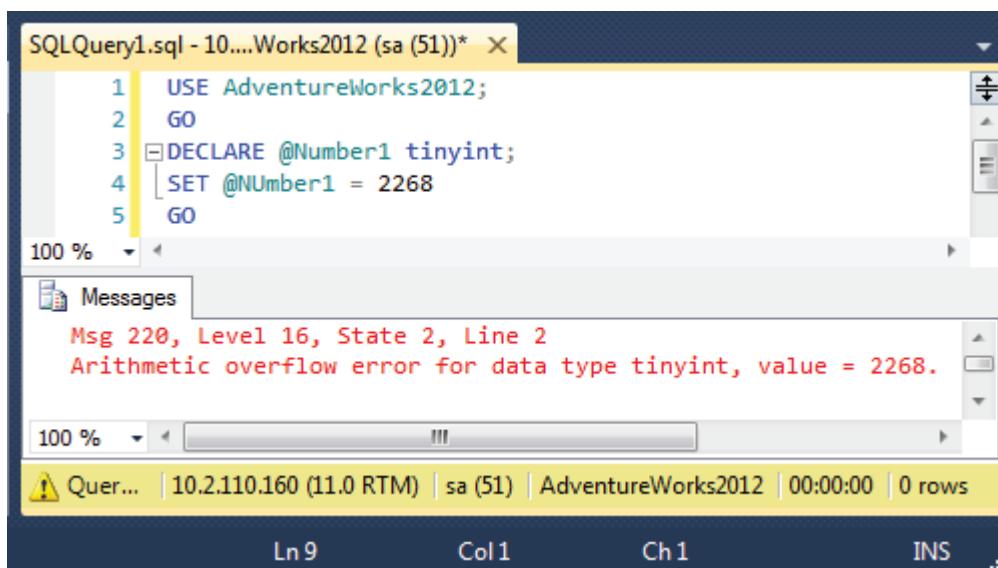
The status bar at the bottom shows: Q... | 10.2.110.160 (11.0 RTM) | sa (51) | AdventureWorks2012 | 00:00:00 | 0 rows

Hình 15.3: Lỗi chia cho số 0

Ở đây, trình biên tập mã sẽ không hiển thị bất kỳ lỗi nào trước khi thực hiện vì không có lỗi cú pháp.

- Sử dụng một thủ tục lưu trữ, hoặc một hàm, hoặc một trigger không có sẵn
- Cố gắng thực hiện một hành động mà một đối tượng hoặc một biến không thể xử lý
- Cố gắng truy cập hoặc sử dụng bộ nhớ máy tính không đủ

Hình 15.4 hiển thị ví dụ về việc cố gắng lưu trữ một giá trị vào biến không đáp ứng phạm vi chỉ định. Trong trường hợp này, lỗi tràn số học xảy ra.



The screenshot shows a SQL query window titled "SQLQuery1.sql - 10....Works2012 (sa (51))*". The query is:

```

1 USE AdventureWorks2012;
2 GO
3 DECLARE @Number1 tinyint;
4 SET @Number1 = 2268
5 GO

```

In the "Messages" pane, an error message is displayed in red:

```

Msg 220, Level 16, State 2, Line 2
Arithmetic overflow error for data type tinyint, value = 2268.

```

The status bar at the bottom shows: Q... | 10.2.110.160 (11.0 RTM) | sa (51) | AdventureWorks2012 | 00:00:00 | 0 rows

Hình 15.4: Lỗi tràn số học

- Cố gắng thực hiện một hành động trên các loại không tương thích
- Sử dụng các câu lệnh điều kiện sai

15.3 Thực hiện xử lý lỗi

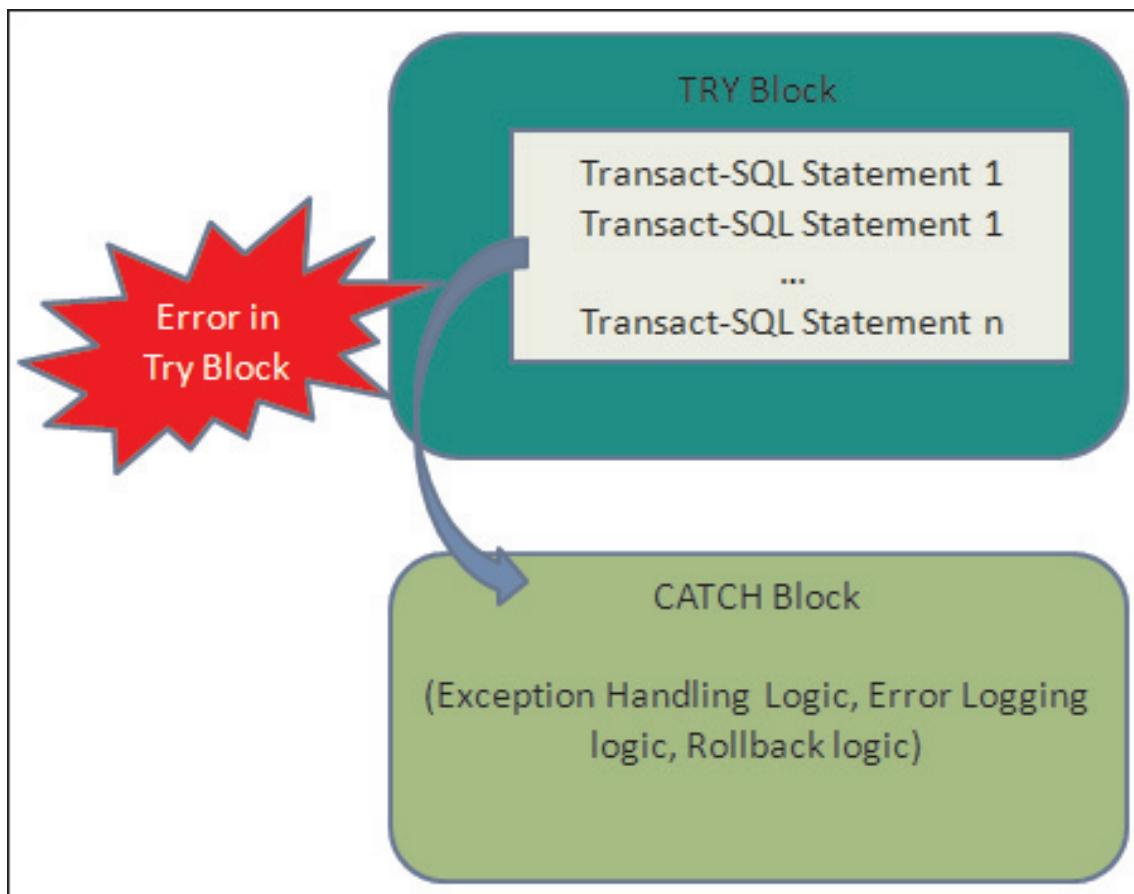
Trong khi phát triển bất kỳ ứng dụng nào, một trong những điều quan trọng nhất mà người dùng cần phải quan tâm là xử lý lỗi. Trong cùng một cách, người dùng cũng phải quan tâm đến việc xử lý ngoại lệ và các lỗi trong khi thiết kế cơ sở dữ liệu. Có thể sử dụng các cơ chế xử lý lỗi khác nhau. Một số trong số này như sau:

- ➔ Khi thực thi một số câu lệnh DML như là INSERT, DELETE, và UPDATE, người dùng có thể xử lý các lỗi để đảm bảo đầu ra chính xác.
- ➔ Khi giao tác không thành công và người dùng cần quay lui giao tác, có thể hiển thị thông báo lỗi thích hợp.
- ➔ Khi làm việc với các con trỏ trong SQL Server, người dùng có thể xử lý các lỗi để đảm bảo có các kết quả chính xác.

15.4 TRY...CATCH

Các câu lệnh TRY...CATCH được sử dụng để thực hiện xử lý ngoại lệ trong Transact-SQL. Một hoặc nhiều câu lệnh Transact-SQL có thể được đưa vào trong khối TRY. Nếu lỗi xảy ra trong khối TRY, kiểm soát được truyền cho khối CATCH có thể chứa một hoặc nhiều câu lệnh.

Hình 15.5 minh họa logic TRY...CATCH.



Hình 15.5: Logic TRY...CATCH

Sau đây là cú pháp cho các câu lệnh TRY...CATCH.

Cú pháp:

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

trong đó:

`sql_statement`: chỉ ra bất kỳ câu lệnh Transact-SQL nào.

`statement_block`: chỉ ra nhóm các câu lệnh Transact-SQL trong khối BEGIN...END.

Cấu trúc TRY...CATCH sẽ bắt tất cả các lỗi thời gian chạy có mức độ nghiêm trọng cao hơn 10 và không đóng kết nối cơ sở dữ liệu. Khối TRY sau là khối CATCH liên quan. Khối TRY...CATCH không thể kéo dài nhiều đợt hoặc nhiều khối câu lệnh Transact-SQL.

Nếu không có lỗi trong khối TRY, sau khi câu lệnh cuối cùng trong khối TRY đã thực hiện, kiểm soát được truyền tới câu lệnh tiếp theo sau câu lệnh END CATCH. Nếu có một lỗi trong khối TRY, kiểm soát được truyền tới câu lệnh đầu tiên bên trong khối CATCH. Nếu END CATCH là câu lệnh cuối cùng trong một trigger hoặc một thủ tục lưu trữ, kiểm soát được truyền trở lại vào khối gọi.

Đoạn mã 1 trình bày một ví dụ đơn giản về các câu lệnh TRY...CATCH.

Đoạn mã 1:

```
BEGIN TRY
    DECLARE @num int;
    SELECT @num=217/0;
END TRY
BEGIN CATCH
    PRINT 'Lỗi xảy ra, không thể chia cho 0'
END CATCH;
```

Trong mã này, nỗ lực được thực hiện để chia một số cho số 0. Điều này sẽ gây ra lỗi, do đó, câu lệnh TRY...CATCH được sử dụng ở đây để xử lý lỗi.

Cả hai khối TRY và CATCH có thể chứa cấu trúc TRY...CATCH lồng nhau. Ví dụ, khối CATCH có thể có một cấu trúc TRY...CATCH nhúng để xử lý các lỗi phải đối mặt với mã CATCH. Lỗi gặp phải trong khối CATCH được đối xử giống như các lỗi được tạo ra ở nơi khác. Nếu khối CATCH bao quanh một cấu trúc TRY...CATCH, bất kỳ lỗi nào trong khối TRY lồng nhau truyền kiểm soát tới khối CATCH lồng nhau. Nếu không có cấu trúc TRY...CATCH lồng nhau, lỗi được truyền lại cho hàm gọi.

Cấu trúc TRY...CATCH cũng có thể bắt các lỗi không được xử lý từ các trigger hoặc thủ tục lưu trữ sẽ thực hiện thông qua mã trong khối TRY. Tuy nhiên, là một cách tiếp cận khác, các trigger hoặc thủ tục lưu trữ cũng có thể bao bọc các cấu trúc TRY...CATCH của chính mình để xử lý các lỗi được tạo ra thông qua mã của chúng.

Câu lệnh GOTO có thể được sử dụng để nhảy đến một nhãn bên trong cùng một khối TRY...CATCH hoặc để rời khỏi khối TRY...CATCH. Cấu trúc TRY...CATCH không nên được sử dụng trong một hàm do người dùng định nghĩa.

15.5 Thông tin lỗi

Thực hành tốt là hiển thị thông tin lỗi cùng với lỗi, để nó có thể giúp giải quyết lỗi một cách nhanh chóng và hiệu quả.

Để đạt được điều này, các hàm hệ thống cần phải được sử dụng trong khối CATCH để tìm thông tin về lỗi đã khởi xướng khối CATCH thực thi.

Những hàm hệ thống như sau:

- **ERROR _ NUMBER()**: trả về số của lỗi.
- **ERROR _ SEVERITY()**: trả về mức độ nghiêm trọng.
- **ERROR _ STATE()**: trả về số trạng thái của lỗi.
- **ERROR _ PROCEDURE()**: trả về tên của trigger hoặc thủ tục lưu trữ nơi đã xảy ra lỗi.
- **ERROR _ LINE()**: trả về số của dòng đã gây ra lỗi.
- **ERROR _ MESSAGE()**: trả về văn bản đầy đủ của lỗi. Văn bản có giá trị được cung cấp cho các tham số như là tên đối tượng, chiều dài, hoặc thời gian.

Những hàm này trả về NULL khi chúng được gọi ra bên ngoài phạm vi của khối CATCH.

→ **Sử dụng TRY..CATCH với thông tin lỗi**

Đoạn mã 2 trình bày một ví dụ đơn giản hiển thị thông tin lỗi.

Đoạn mã 2:

```
USE AdventureWorks2012;
GO
BEGIN TRY
    SELECT 217/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Trong mã này, câu lệnh SELECT sẽ gây ra một lỗi chia cho số không được xử lý sử dụng câu lệnh TRY...CATCH. Lỗi gây ra việc thực thi để nhảy đến khối CATCH trong đó thông tin lỗi sẽ được hiển thị.

Hình 15.6 trình bày kết quả của thông tin lỗi. Tập kết quả đầu tiên là trống vì câu lệnh không thành công.

(No column name)				
	ErrorNumber	ErrorSeverity	ErrorLine	ErrorMessage
1	8134	16	2	Divide by zero error encountered.

Hình 15.6: Thông tin lỗi

Đoạn mã 3 trình bày một thủ tục lưu trữ có chứa các hàm xử lý lỗi.

Đoạn mã 3:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ('sp_ErrorInfo', 'P') IS NOT NULL
DROP PROCEDURE sp_ErrorInfo;
GO
CREATE PROCEDURE sp_ErrorInfo
AS
SELECT
    ERROR_NUMBER() AS ErrorNumber,
    ERROR_SEVERITY() AS ErrorSeverity,
    ERROR_STATE() AS ErrorState,
    ERROR_PROCEDURE() AS ErrorProcedure,
    ERROR_LINE() AS ErrorLine,
    ERROR_MESSAGE() AS ErrorMessage;
GO
```

```
BEGIN TRY
SELECT 217/0;
END TRY
BEGIN CATCH
EXECUTE sp_ErrorInfo;
END CATCH;
```

Trong đoạn mã này, khi lỗi xảy ra, khối CATCH của cấu trúc TRY...CATCH được gọi ra và thông tin lỗi được trả lại.

→ Sử dụng TRY...CATCH với giao tác

Đoạn mã 4 thể hiện khối TRY...CATCH làm việc bên trong một giao tác.

Đoạn mã 4:

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;
BEGIN TRY
    DELETE FROM Production.Product
    WHERE ProductID = 980;
END TRY
BEGIN CATCH
    SELECT
        ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_NUMBER() AS ErrorNumber
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_STATE() AS ErrorState
        ,ERROR_MESSAGE() AS ErrorMessage
        ,ERROR_LINE() AS ErrorLine;
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
END CATCH;
```

```
IF @@TRANCOUNT > 0
    COMMIT TRANSACTION;
GO
```

Trong mã này, khối TRY...CATCH làm việc trong giao tác. Câu lệnh bên trong khối TRY tạo ra một lỗi vi phạm ràng buộc như sau:

The DELETE statement is conflicted with the REFERENCE constraint "FK_BillOfMaterials_Product_ProductAssemblyID". The conflict occurred in database "AdventureWorks2012", table "Production.BillOfMaterials", column 'ProductAssemblyID'.

→ Các giao tác không thể thực hiện được

Nếu lỗi được tạo ra trong khối TRY, nó làm cho trạng thái của giao tác hiện tại thành không hợp lệ và giao tác được coi là một giao tác không được thực hiện. Giao tác không thể thực hiện được chỉ thực hiện các hoạt động ROLLBACK TRANSACTION hoặc đọc. Giao tác không thực thi bất kỳ câu lệnh Transact-SQL nào mà thực hiện hoạt động COMMIT TRANSACTION hoặc viết.

Hàm XACT_STATE trả về -1 nếu giao tác đã được phân loại như là một giao tác không thể thực hiện được. Khi khối lệnh đã hoàn thành, Database Engine quay lui bất kỳ giao tác không thể thực hiện được nào. Nếu không có thông báo lỗi nào được gửi đi khi các giao tác đi vào trạng thái không thể thực hiện được khi hoàn thành khối lệnh, khi đó các thông báo lỗi được gửi cho máy khách. Điều này chỉ ra rằng một giao tác không thể thực hiện được được phát hiện và quay lui.

15.6 @@ERROR

Hàm @@ERROR trả về số của lỗi cho câu lệnh Transact-SQL cuối cùng đã thực hiện.

Sau đây là cú pháp cho hàm @@ERROR.

Cú pháp:

```
@@ERROR
```

Hàm hệ thống @@ERROR trả về giá trị thuộc loại số nguyên. Hàm này trả về 0, nếu câu lệnh Transact-SQL trước không gặp phải lỗi nào. Nó còn trả về số của lỗi chỉ khi các câu lệnh trước đó gặp phải lỗi. Nếu lỗi là một trong những danh sách các lỗi trong dạng xem danh mục sys.messages bao gồm giá trị từ cột sys.messages.messages_id cho lỗi đó. Người dùng có thể xem văn bản có liên quan đến số của lỗi @@ERROR trong dạng xem danh mục sys.messages.

Đoạn mã 5 trình bày cách sử dụng @@ERROR để kiểm tra các vi phạm ràng buộc.

Đoạn mã 5:

```
USE AdventureWorks2012;
GO
BEGIN TRY
    UPDATE HumanResources.EmployeePayHistory
        SET PayFrequency = 4
        WHERE BusinessEntityID = 1;
END TRY
BEGIN CATCH
    IF @@ERROR = 547
        PRINT N'Kiểm tra vi phạm ràng buộc đã xảy ra.';
END CATCH
```

Trong đoạn mã này, @@ERROR được sử dụng để kiểm tra vi phạm ràng buộc kiểm tra (trong đó có số của lỗi 547) trong câu lệnh UPDATE.

Nó sẽ hiển thị thông báo lỗi sau:

```
Check constraint violation has occurred.
```

15.7 RAISERROR

Câu lệnh RAISERROR bắt đầu quá trình xử lý lỗi cho một phiên và hiển thị thông báo lỗi. RAISERROR có thể tham khảo một thông báo do người dùng định nghĩa được lưu trữ trong dạng xem danh mục sys.messages hoặc xây dựng các thông báo lỗi động vào thời gian chạy. Thông báo này được trả về như một thông báo lỗi máy chủ cho ứng dụng gọi hoặc cho khối CATCH liên quan của cấu trúc TRY...CATCH.

Sau đây là cú pháp cho câu lệnh RAISERROR.

Cú pháp:

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

trong đó:

msg_id: chỉ ra số của thông báo lỗi do người dùng định nghĩa được lưu trữ trong dạng xem danh mục sys.messages sử dụng sp_addmessage.

`msg_str`: Chỉ ra các thông báo do người dùng định nghĩa với định dạng. `msg_str` là một chuỗi ký tự với thông số kỹ thuật chuyển đổi nhúng tùy chọn. Thông số kỹ thuật chuyển đổi có định dạng sau:

```
% [[flag] [width] [. precision] [{h | l}]] type
```

Các tham số có thể được sử dụng trong `msg_str` như sau:

{h | l} type: Chỉ ra việc sử dụng các loại ký tự `d`, `i`, `o`, `s`, `x`, `X`, hoặc `u`, và tạo ra `shortint`(`h`) hoặc `longint`(`l`).

Sau đây là một số thông số kỹ thuật loại:

`d` hoặc `i`: Chỉ ra số nguyên có dấu

`o`: Chỉ ra bát phân không dấu

`x` hoặc `X`: Chỉ ra thập lục phân không dấu

flag: Chỉ ra mã xác định giãn cách và sự cân chỉnh giá trị thay thế. Cái này có thể bao gồm các biểu tượng như `-` (trừ) và `+(cộng)` để chỉ ra căn chỉnh trái hoặc để chỉ ra giá trị tương ứng là loại có dấu.

precision: Chỉ ra số lượng ký tự tối đa được lấy từ giá trị đối số cho các giá trị chuỗi. Ví dụ, nếu một chuỗi có năm ký tự và độ chính xác là 2, chỉ có hai ký tự đầu tiên của giá trị chuỗi được sử dụng.

width: Chỉ ra một số nguyên định nghĩa chiều rộng tối thiểu cho trường, trong đó giá trị đối số được đặt.

`@local_variable`: Chỉ ra một biến thuộc bất kỳ kiểu dữ liệu ký tự hợp lệ nào có chứa chuỗi được định dạng trong cùng một cách như `msg_str`.

severity: Mức độ nghiêm trọng từ 0 đến 18 được chỉ ra bởi bất kỳ người dùng nào. Mức độ nghiêm trọng từ 19 đến 25 được quy định bởi các thành viên của vai trò máy chủ cố định `sysadmin` hoặc người dùng với cấp phép `ALTER TRACE`. Mức độ nghiêm trọng từ 19 đến 25 sử dụng tùy chọn `WITH LOG` là bắt buộc.

option: Chỉ ra tùy chọn tùy chỉnh cho lỗi này.

Bảng 15.1 liệt kê những giá trị cho các tùy chọn tùy chỉnh.

Giá trị	Mô tả
<code>LOG</code>	Ghi lại lỗi trong nhật ký lỗi và nhật ký ứng dụng cho thể hiện của Công cụ cơ sở dữ liệu Microsoft SQL Server.
<code>NOWAIT</code>	Gửi thông báo trực tiếp cho máy khách
<code>SETERROR</code>	Đặt các giá trị <code>ERROR_NUMBER</code> và <code>@@ERROR</code> thành <code>msg_id</code> hoặc 5000 không phân biệt mức độ nghiêm trọng.

Bảng 15.1: Các giá trị thông số kỹ thuật loại

Khi `RAISERROR` thực thi với mức độ nghiêm trọng là 11 hoặc cao hơn trong khối `TRY`, nó sẽ chuyển điều khiển đến khối `CATCH` liên quan.

Những lỗi sau đây được trả về lại cho hàm gọi nếu RAISERROR thực hiện:

- Ra khỏi phạm vi của bất kỳ khối TRY nào
- Có mức độ nghiêm trọng là 10 hoặc thấp hơn trong khối TRY
- Có mức độ nghiêm trọng là 20 hoặc cao hơn sẽ chấm dứt kết nối cơ sở dữ liệu

Khối CATCH có thể sử dụng câu lệnh RAISERROR để ném lại lỗi đã gọi ra khối CATCH. Đối với điều này, sẽ cần phải biết thông tin lỗi ban đầu có thể thu được thông qua các hàm hệ thống ERROR_NUMBER và ERROR_MESSAGE.

Theo mặc định, @@ERROR được đặt là 0 cho các thông báo có mức độ nghiêm trọng từ 1 đến 10.

Đoạn mã 6 trình bày cách để xây dựng câu lệnh RAISERROR để hiển thị câu lệnh lỗi tùy chỉnh.

Đoạn mã 6:

```
RAISERROR (N'Dây là thông báo lỗi %s %d.',  
          10, 1, N'số seri', 23);  
  
GO
```

Trong đoạn mã này, câu lệnh RAISERROR có tham số đầu tiên của N'số seri' thay đổi thông số kỹ thuật chuyển đổi đầu tiên là %s, và tham số thứ hai là 23 thay đổi chuyển đổi thứ hai là %d. Đoạn mã này hiển thị 'Đây là thông báo lỗi số seri 23'. Đoạn mã 7 trình bày cách sử dụng câu lệnh RAISERROR để trả về cùng một chuỗi.

Đoạn mã 7:

```
RAISERROR (N'%*.*s', 10, 1, 7, 3, N'Hello world');  
  
GO  
  
RAISERROR (N'%7.3s', 10, 1, N'Hello world');  
  
GO
```

Trong đoạn mã này, câu lệnh RAISERROR trả về cùng một chuỗi **He1**. Câu lệnh đầu tiên chỉ ra chiều rộng và các giá trị độ chính xác và câu lệnh thứ hai chỉ ra thông số kỹ thuật chuyển đổi.

Người dùng cũng có thể trả về thông tin lỗi từ khối CATCH.

Đoạn mã 8 trình bày cách sử dụng câu lệnh RAISERROR bên trong khối TRY.

Đoạn mã 8:

```
BEGIN TRY
    RAISERROR ('Raises Error in the TRY block.', 16, 1);
END TRY
BEGIN CATCH
    DECLARE @ErrorMessage NVARCHAR(4000);
    DECLARE @ErrorSeverity INT;
    DECLARE @ErrorState INT;
    SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE();
    RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
END CATCH;
```

Trong đoạn mã này, câu lệnh RAISERROR được sử dụng bên trong khối TRY có mức độ nghiêm trọng 16, từ đó dẫn đến việc thực thi để chuyển đến khối CATCH liên kết.

RAISERROR sau đó được sử dụng bên trong khối CATCH để trả về thông tin lỗi về lỗi ban đầu.

15.8 ERROR_STATE

Hàm hệ thống ERROR_STATE trả về số lỗi trạng thái làm cho khối CATCH của cấu trúc TRY...CATCH thực hiện. Sau đây là cú pháp cho hàm hệ thống ERROR_STATE.

Cú pháp:

```
ERROR_STATE ( )
```

Khi được gọi trong khối CATCH, nó trả về số trạng thái của thông báo lỗi đã làm cho khối CATCH được chạy. Hàm này trả về NULL khi chúng được gọi ra bên ngoài phạm vi của khối CATCH.

Có các thông báo lỗi cụ thể được nêu ra tại các điểm khác nhau trong đoạn mã cho Công cụ cơ sở dữ liệu SQL Server. Ví dụ, lỗi 1105 được sinh ra cho một số điều kiện khác nhau. Mỗi điều kiện cụ thể gây nên lỗi sẽ gán mã trạng thái duy nhất này.

ERROR_STATE được gọi từ bất cứ nơi nào trong phạm vi khối CATCH. ERROR_STATE trả về trạng thái lỗi bất kể bao nhiêu lần nó được thực thi hoặc cho dù nó được thực thi trong phạm vi của khối CATCH. Đây là trong so sánh với những hàm như @@ERROR chỉ trả về số lỗi trong câu lệnh ngay sau cái gây ra lỗi, hoặc trong câu lệnh đầu tiên của khối CATCH.

Người dùng có thể sử dụng `ERROR_STATE` trong khối CATCH. Đoạn mã 9 trình bày cách sử dụng câu lệnh `ERROR_STATE` bên trong khối TRY.

Đoạn mã 9:

```
BEGIN TRY
    SELECT 217/0;
END TRY
BEGIN CATCH
    SELECT ERROR_STATE() AS ErrorState;
END CATCH;
GO
```

Trong đoạn mã này, câu lệnh `SELECT` tạo ra lỗi chia cho số 0. Câu lệnh `CATCH` sau đó sẽ trả về trạng thái của lỗi. The `ERROR_STATE` được hiển thị là 1.

15.9 `ERROR_SEVERITY`

Hàm `ERROR_SEVERITY` trả về mức độ nghiêm trọng của lỗi gây ra khối CATCH của cấu trúc `TRY...CATCH` sẽ được thực thi.

Sau đây là cú pháp cho `ERROR_SEVERITY`.

Cú pháp:

```
ERROR_SEVERITY ( )
```

Nó trả về giá trị `NULL` nếu được gọi ra bên ngoài phạm vi của khối CATCH. `ERROR_STATE` có thể được gọi bất cứ nơi nào trong phạm vi khối CATCH. Trong các khối CATCH lồng nhau, `ERROR_SEVERITY` sẽ trả về mức độ nghiêm trọng của lỗi cụ thể cho phạm vi của khối CATCH nơi nó được tham chiếu. Người dùng có thể sử dụng hàm `ERROR_SEVERITY` trong khối CATCH.

Đoạn mã 10 trình bày cách hiển thị mức độ nghiêm trọng của lỗi.

Đoạn mã 10:

```
BEGIN TRY
    SELECT 217/0;
BEGIN CATCH
    SELECT ERROR_SEVERITY() AS ErrorSeverity;
END CATCH;
```

GO

END TRY

Trong đoạn mã này, một nỗ lực để chia cho số 0 tạo ra lỗi và làm cho khối CATCH hiển thị lỗi nghiêm trọng là 16.

15.10 ERROR PROCEDURE

Hàm ERROR PROCEDURE trả về trigger hoặc tên thủ tục lưu trữ nơi đã xảy ra lỗi đã làm cho khối CATCH của một cấu trúc TRY...CATCH được thực thi. Sau đây là cú pháp của ERROR PROCEDURE.

Cú pháp:

```
ERROR PROCEDURE ( )
```

Nó trả về kiểu dữ liệu nvarchar. Khi hàm được gọi ra trong khối CATCH, nó sẽ trả về tên của thủ tục lưu trữ nơi đã xảy ra lỗi. Hàm này trả về giá trị NULL nếu lỗi đã không xảy ra trong trigger hoặc một thủ tục lưu trữ. ERROR PROCEDURE có thể được gọi từ bất cứ nơi nào trong phạm vi khối CATCH. Hàm này còn trả về NULL nếu hàm này được gọi ra bên ngoài phạm vi của khối CATCH.

Trong các khối CATCH lồng nhau, ERROR PROCEDURE trả về trigger hoặc tên thủ tục lưu trữ cụ thể cho phạm vi của khối CATCH, nơi nó được tham chiếu.

Đoạn mã 11 trình bày việc sử dụng hàm ERROR PROCEDURE.

Đoạn mã 11:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ( 'usp_Example', 'P' ) IS NOT NULL
DROP PROCEDURE usp_Example;
GO
CREATE PROCEDURE usp_Example
AS
SELECT 217/0;
GO
BEGIN TRY
EXECUTE usp_Example;
```

```

END TRY
BEGIN CATCH
    SELECT ERROR_PROCEDURE () AS ErrorProcedure;
END CATCH;
GO

```

Trong đoạn mã này, thủ tục lưu trữ **usp_Example** tạo ra lỗi chia cho số 0. Vì vậy hàm **ERROR_PROCEDURE** trả về tên của thủ tục lưu trữ này, nơi đã xảy ra lỗi.

Đoạn mã 12 trình bày việc sử dụng hàm **ERROR_PROCEDURE** cùng với các hàm khác.

Đoạn mã 12:

```

USE AdventureWorks2012;
GO
IF OBJECT_ID ( 'usp_Example', 'P' ) IS NOT NULL
DROP PROCEDURE usp_Example;
GO
CREATE PROCEDURE usp_Example
AS
SELECT 217/0;
GO
BEGIN TRY
EXECUTE usp_Example;
END TRY
BEGIN CATCH
SELECT
    ERROR_NUMBER () AS ErrorNumber,
    ERROR_SEVERITY () AS ErrorSeverity,
    ERROR_STATE () AS ErrorState,
    ERROR_PROCEDURE () AS ErrorProcedure,
    ERROR_MESSAGE () AS ErrorMessage,
    ERROR_LINE () AS ErrorLine;
END CATCH;
GO

```

Đoạn mã này sử dụng một số hàm hệ thống xử lý lỗi có thể giúp phát hiện và khắc phục lỗi dễ dàng.

15.11 ERROR_NUMBER

Hàm hệ thống `ERROR_NUMBER` khi được gọi trong khối `CATCH` trả về số lỗi của lỗi làm cho khối `CATCH` của cấu trúc `TRY...CATCH` được thực thi. Sau đây là cú pháp của `ERROR_NUMBER`.

Cú pháp:

```
ERROR_NUMBER ( )
```

Hàm này có thể được gọi từ bất cứ nơi nào trong phạm vi khối `CATCH`. Hàm này sẽ trả về `NULL` khi nó được gọi ra bên ngoài phạm vi của khối `CATCH`.

`ERROR_NUMBER` trả về số lỗi bất kể bao nhiêu lần nó thực thi hoặc cho dù nó được thực thi trong phạm vi của khối `CATCH`. Cái này khác so với `@@ERROR` chỉ trả về số lỗi trong câu lệnh này ngay sau cái gây ra lỗi, hoặc câu lệnh đầu tiên của khối `CATCH`.

Đoạn mã 13 trình bày việc sử dụng `ERROR_NUMBER` trong `CATCH`.

Đoạn mã 13:

```
BEGIN TRY
    SELECT 217/0;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber;
END CATCH;
GO
```

Là kết quả của đoạn mã này, số lỗi được hiển thị khi cố gắng chia cho số 0 xảy ra.

15.12 ERROR_MESSAGE

Hàm `ERROR_MESSAGE` trả về thông báo lỗi của lỗi làm cho khối `CATCH` của cấu trúc `TRY...CATCH` thực thi.

Sau đây là cú pháp của `ERROR_MESSAGE`.

Cú pháp:

```
ERROR_MESSAGE ( )
```

Khi hàm `ERROR_MESSAGE` được gọi trong khối CATCH, nó sẽ trả về toàn bộ nội dung của thông báo lỗi làm cho khối CATCH thực thi. Văn bản bao gồm những giá trị được cung cấp cho bất kỳ tham số nào có thể thay thế được như tên đối tượng, thời gian, hoặc độ dài. Nó còn trả về NULL nếu hàm này được gọi ra bên ngoài phạm vi của khối CATCH.

Đoạn mã 14 trình bày việc sử dụng `ERROR_MESSAGE` trong khối CATCH.

Đoạn mã 14:

```
BEGIN TRY
    SELECT 217/0;
END TRY
BEGIN CATCH
    SELECT ERROR_MESSAGE () AS ErrorMessage;
END CATCH;
GO
```

Trong đoạn mã này, tương tự như các ví dụ khác, câu lệnh `SELECT` tạo ra lỗi chia cho số 0. Khối CATCH hiển thị thông báo lỗi.

15.13 `ERROR_LINE`

Hàm `ERROR_LINE` trả về số dòng mà tại đó xảy ra lỗi trong khối TRY...CATCH.

Sau đây là cú pháp của `ERROR_LINE`.

Cú pháp:

```
ERROR_LINE ( )
```

Khi hàm này được gọi trong khối CATCH, nó trả về số dòng nơi đã xảy ra lỗi. Nếu lỗi đã xảy ra trong trigger hoặc thủ tục lưu trữ, nó trả về số dòng trong trigger hoặc thủ tục lưu trữ đó. Tương tự như các hàm khác, hàm này trả về NULL khi chúng được gọi ra bên ngoài phạm vi của khối CATCH.

Đoạn mã 15 trình bày việc sử dụng `ERROR_LINE` trong khối CATCH.

Đoạn mã 15:

```
BEGIN TRY
    SELECT 217/0;
END TRY
```

```
BEGIN CATCH
SELECT ERROR_LINE() AS ErrorLine;
END CATCH;
GO
```

Là kết quả của đoạn mã này, số dòng tại đó lỗi đã xảy ra sẽ được hiển thị.

15.14 Lỗi bị ảnh hưởng bởi cấu trúc TRY...CATCH

Cấu trúc TRY...CATCH không bấy các điều kiện sau:

- ➔ Các thông báo cung cấp thông tin hoặc cảnh báo có mức độ nghiêm trọng là 10 hoặc thấp hơn
- ➔ Lỗi có mức độ nghiêm trọng là 20 hoặc cao hơn sẽ dừng lại việc xử lý tác vụ Công cụ cơ sở dữ liệu SQL Server cho phiên đó. Nếu xảy ra lỗi có mức độ nghiêm trọng là 20 hoặc cao hơn và kết nối cơ sở dữ liệu không bị gián đoạn, TRY...CATCH sẽ xử lý lỗi
- ➔ Sự quan tâm như là kết nối máy khách bị hỏng hoặc yêu cầu máy khách bị gián đoạn
- ➔ Khi phiên kết thúc vì câu lệnh KILL được sử dụng bởi quản trị viên hệ thống

Những loại lỗi sau đây không được xử lý bằng khối CATCH xảy ra ở cùng một cấp độ thực thi như của cấu trúc TRY...CATCH:

- ➔ Biên dịch lỗi như lỗi cú pháp để hạn chế khối lệnh không chạy
- ➔ Lỗi phát sinh trong biên dịch lại ở mức câu lệnh như là lỗi phân giải tên đối tượng xảy ra sau khi biên dịch do độ phân giải tên bị trì hoãn.

Đoạn mã 16 trình bày cách một lỗi phân giải tên đối tượng được tạo ra do câu lệnh SELECT.

Đoạn mã 16:

```
USE AdventureWorks2012;
GO
BEGIN TRY
    SELECT * FROM Nonexistent;
END TRY
```

```
BEGIN CATCH
SELECT
    ERROR_NUMBER() AS ErrorNumber,
    ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

Đoạn mã này sẽ gây ra lỗi phân giải tên đối tượng trong câu lệnh SELECT. Nó sẽ không bắt được bằng cấu trúc TRY...CATCH.

Việc chạy câu lệnh SELECT bên trong một thủ tục lưu trữ gây ra lỗi xảy ra ở một mức độ thấp hơn khối TRY. Lỗi này được xử lý bởi cấu trúc TRY...CATCH.

Đoạn mã 17 trình bày thông báo lỗi được hiển thị như thế nào trong trường hợp như vậy.

Đoạn mã 17:

```
IF OBJECT_ID ( N'sp_Example', N'P' ) IS NOT NULL
DROP PROCEDURE sp_Example;
GO
CREATE PROCEDURE sp_Example
AS
SELECT * FROM Nonexistent;
GO
BEGIN TRY
EXECUTE sp_Example;
END TRY
BEGIN CATCH
SELECT
    ERROR_NUMBER() AS ErrorNumber,
    ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

15.15 THROW

Câu lệnh THROW phát sinh ngoại lệ chuyển điều khiển thực thi cho khối CATCH của cấu trúc TRY...CATCH.

Sau đây là cú pháp của câu lệnh THROW.

Cú pháp:

```
THROW [ { error_number | @local_variable },
{ message | @local_variable },
{ state | @local_variable }
] [ ; ]
```

trong đó:

`error_number`: chỉ ra một hằng số hoặc biến trình bày `error_number` là int.

`message`: chỉ ra một biến hoặc chuỗi định nghĩa thông báo ngoại lệ là nvarchar(2048).

`State`: chỉ ra một biến hoặc hằng số từ 0 đến 255 chỉ ra trạng thái để liên kết với trạng thái của thông báo là tinyint.

Đoạn mã 18 trình bày việc sử dụng câu lệnh THROW để gây ra lại một ngoại lệ.

Đoạn mã 18:

```
USE tempdb;
GO
CREATE TABLE dbo.TestRethrow
(
    ID INT PRIMARY KEY
);
BEGIN TRY
    INSERT dbo.TestRethrow(ID) VALUES(1);
    INSERT dbo.TestRethrow(ID) VALUES(1);
END TRY
BEGIN CATCH
    PRINT 'In catch block.';
    THROW;
END CATCH;
```

Trong đoạn mã này, câu lệnh THROW được sử dụng để gây ra lại một lần nữa ngoại lệ đã xảy ra lần cuối.

Kết quả của đoạn mã này sẽ như sau:

(1 row(s) affected)

(0 row(s) affected)

In catch block.

Msg 2627, Level 14, State 1, Line 6

Violation of PRIMARY KEY constraint 'PK__TestReth__3214EC27AAB15FEE'.
Cannot insert duplicate key in object 'dbo.TestRethrow'. The duplicate
key value is (1).

15.16 Kiểm tra tiến bộ của bạn

1. _____ chỉ xảy ra nếu người dùng viết đoạn mã mà không thể được phân tích bằng SQL Server, chẳng hạn như từ khóa sai hoặc câu lệnh không đầy đủ.

(A)	Lỗi cú pháp	(C)	Lỗi logic
(B)	Lỗi thời gian chạy	(D)	Nhật ký lỗi

2. Cấu trúc nào sau đây có thể bắt các lỗi chưa xử lý từ các trigger hoặc thủ tục lưu trữ?

(A)	IF-ELSE	(C)	RAISERROR
(B)	TRY...CATCH	(D)	@@ERROR

3. Hàm nào sau đây trả về số lỗi cho câu lệnh Transact-SQL cuối cùng đã thực thi?

(A)	ERROR_LINE	(C)	@@ERROR
(B)	RAISERROR	(D)	@@ERROR_NUMBER

4. Hàm nào sau đây trả về mức độ nghiêm trọng của lỗi làm cho khối CATCH của cấu trúc TRY...CATCH được thực thi?

(A)	ERROR_LINE	(C)	ERROR_PROCEDURE
(B)	ERROR_NUMBER	(D)	ERROR_SEVERITY

5. Câu lệnh _____ sinh ra ngoại lệ và truyền thực thi cho khối CATCH của cấu trúc TRY...CATCH trong SQL Server 2012.

(A)	BEGIN	(C)	THROW
(B)	END	(D)	ERROR

15.16.1 Câu trả lời

1.	(A)
2.	(B)
3.	(C)
4.	(D)
5.	(C)

Tóm tắt

- Lỗi cú pháp là lỗi xảy ra khi mã không thể phân tích được bằng SQL Server.
- Lỗi thời gian chạy xảy ra khi ứng dụng cố gắng thực hiện một hành động không được hỗ trợ bởi Microsoft SQL Server cũng không phải bởi hệ điều hành.
- Các câu lệnh TRY...CATCH được sử dụng để xử lý các ngoại lệ trong Transact-SQL.
- Cấu trúc TRY...CATCH cũng có thể bắt các lỗi không được xử lý từ các trigger hoặc thủ tục lưu trữ sẽ thực hiện thông qua mã trong khối TRY.
- Các câu lệnh GOTO có thể được sử dụng để nhảy đến một nhãn bên trong cùng một khối TRY...CATCH hoặc để rời khỏi khối TRY...CATCH.
- Các hàm hệ thống khác nhau có sẵn trong Transact-SQL để in thông tin lỗi về lỗi đã xảy ra.
- Câu lệnh RAISERROR được dùng để bắt đầu quá trình xử lý lỗi cho một phiên và hiển thị thông báo lỗi.



1. Đối với các giao tác được tạo ra trong Hãy thử tự làm của Phần 10 và 15, hãy thêm các câu lệnh xử lý lỗi để chú ý đến các lỗi.
2. Acme Technologies Private Limited là một công ty phần mềm hàng đầu tại New York. Công ty đã đạt được nhiều giải thưởng như đại lý tốt nhất trong phát triển các công nghệ phần mềm. Công ty đã nhận được nhiều dự án mới về phát triển di động và web. Hiện nay, họ đang làm việc trên một dự án cơ sở dữ liệu cho Hệ thống quản lý tiền lương trong SQL Server 2012.

Họ đã tạo ra cơ sở dữ liệu trên hệ thống quản lý tiền lương cho người lao động. Trong khi tạo ra các bảng, họ nhận được các loại lỗi khác nhau. Giả sử rằng bạn là quản trị viên cơ sở dữ liệu của Acme Technologies và tổ trưởng kỹ thuật đã giao cho bạn nhiệm vụ chỉnh sửa những lỗi đó. Thực hiện những bước sau:

- a. Viết các câu lệnh xử lý lỗi sử dụng cấu trúc TRY...CATCH cho cả hai câu lệnh bình thường cũng như thủ tục lưu trữ.
- b. Hiển thị thông tin lỗi sử dụng như sau:
 - ◆ ERROR_NUMBER
 - ◆ ERROR_MESSAGE
 - ◆ ERROR_LINE