|  |
| --- |
| 3. NGÔN NGỮ TRUY VẤN QUAN HỆ |
| Một ngôn ngữ truy vấn là một ngôn ngữ trong đó một người sử dụng cơ sở dữ liệu có thể yêu cầu thông tin từ cơ sở dữ liệu. Hầu hết các ngôn ngữ truy vấn đều ở một mức cao hơn các ngôn ngữ lập trình chuẩn như C hay Java. Các ngôn ngữ truy vấn thuộc nhóm ngôn ngữ thế hệ thứ tư. Các ngôn ngữ truy vấn có thể được phân loại chung thành hai nhóm: ngôn ngữ có thủ tục và ngôn ngữ phi thủ tục. Một ngôn ngữ truy vấn thủ tục đòi hỏi người sử dụng xác định một chuỗi các phép toán trên cơ sở dữ liệu để tính kết  quả mong muốn (Người sử dụng sẽ xác định cụ thể cần cái gì và thực hiện như thế nào). Một ngôn ngữ phi thủ tục đòi hỏi người sử dụng mô tả kết quả mong muốn mà không cần xác định chuỗi các thao tác cần thiết để đạt được kết quả đó (Người sử dụng chỉ xác định cụ thể cần cái gì). Hầu hết các hệ thống cơ sở dữ liệu quan hệ có sẵn trên thị trường cung cấp một ngôn ngữ truy vấn thuộc loại trung gian lai ghép giữa hai loại trình bày trên. Ngôn ngữ truy vấn lai bao gồm các thành phần của cả hai cách tiếp cận có thủ tục và phi thủ tục tới ngôn ngữ truy vấn. Chúng ta chỉ xem xét các ngôn ngữ truy vấn quan hệ thuần túy. Những ngôn ngữ này là chính thống và thiếu nhiều cú pháp mở rộng có sẵn trong các ngôn ngữ trên thị trường hiện nay nhưng nó mô tả những kỹ thuật nền tảng được sử dụng bởi tất cả các ngôn ngữ truy vấn để lấy dữ liệu ra từ cơ sở dữ liệu. Khi chúng ta xem xét các ngôn ngữ thuần túy này, mặc dù các ngôn ngữ chuẩn này không chứa các đặc tính mở rộng, một ngôn ngữ truy vấn hoàn chỉnh cung cấp các cách thức để thêm và xóa các bộ từ các quan hệ cũng như cho phép thay đổi các bộ đang có. Ngôn ngữ có thủ tục được xem xét đến là ngôn ngữ đại số quan hệ. Ngôn ngữ phi thủ tục có thể kể đến trong phạm vi môn học này là tính toán bộ và tính toán vị từ. |
| 3.1. Ngôn ngữ đại số quan hệ |
| Là một ngôn ngữ truy vấn thủ tục. Nó bao gồm các phép toán tập hợp hoặc là loại hai ngôi hoặc là một ngôi, có nghĩa là các toán hạng của các phép toán tập hợp này sẽ là một quan hệ hoặc hai quan hệ. Mỗi một phép toán tập hợp này sẽ sinh ra một quan hệ là kết quả đầu ra. Năm phép toán cơ bản của đại số quan hệ và một số phép toán mở rộng bổ sung dựa trên năm phép toán cơ bản này sẽ được chúng ta xem xét trong phạm vi bài giảng này. Một phép toán đặt lại tên *rename* đôi khi cũng được coi là phép toán cơ bản. Năm phép toán cơ bản bao gồm: phép chọn, phép hợp, phép chiếu, phép trừ, và tích Đề các. Chúng ta sẽ xem xét mỗi phép toán riêng rẽ trước khi kết hợp các phép toán đó vào một biểu thức phức tạp hơn. Một số các phép toán bổ sung khác cũng được định nghĩa trong đại số quan hệ bao gồm: phép giao, kết nối tự nhiên, phép chia, phép nửa kết nối và kết nối ngoài. Chúng ta sẽ lần lượt xem xét từng phép toán một trước khi dùng chúng trong một biểu thức đại số phức tạp. |
| 3.1.1. Các phép toán cơ bản trong đại số quan hệ |
| 3.1.1.1. Phép chọn |
| Là một phép toán một ngôi được ký hiệu bởi một chữ cái Hy lạp là sigma \fn_cm \sigma Dạng chung của phép chọn này như sau: \fn_cm \sigma(mệnh đề)(thể hiện một quan hệ) Lược đồ của quan hệ kết quả: giống hết như quan hệ toán hạng. Kích cỡ của quan hệ kết quả (số bộ) \fn_cm \leq |quan hệ toán hạng|  *Ví dụ:* \fn_cm \sigma(major = “CS”)(students) \fn_cm \sigma(major = “CS” and hair-color = “brown”)(students) \fn_cm \sigma(hours-attempted > hours-earned)(students) Phép chọn này sẽ lựa chọn ra những bộ từ thể hiện của một quan hệ mà thỏa mãn một mệnh đề cụ thể nào đó. Nhìn chung, một mệnh đề có thể chứa bất kể một toán tử so sánh logic nào trong số \fn_cm =, \fn_cm \leq, \fn_cm <, \fn_cm \geq, \fn_cm >, \fn_cm \neq hơn nữa một số các mệnh đề có thể kết hợp chúng với các phép toán liên kết như *and* (\fn_cm \wedge), *or* (\fn_cm \vee), và *not* (\fn_cm \rightharpoondown). Phép toán lựa chọn có thể được hình dung như một lát cắt ngang của quan hệ toán hạng. Ví dụ về toán tử lựa chọn được thể hiện dưới đây với một quan hệ R   |  | | --- | |  | |  | |
| 3.1.1.2. Phép chiếu |
| Là một loại phép toán một ngôi được kí hiệu bởi một chữ cái Hy Lạp  pi: \fn_cm \pi  Dạng chung của phép toán này là \fn_cm \pi(danh sách các thuộc tính)(thể hiện của quan hệ) Lược đồ của quan hệ kết: được xác định bởi <danh sách các thuộc tính > Kích cỡ của quan hệ kết quả (số bộ) \fn_cm \leq |quan hệ toán hạng|  *Ví dụ:*  \fn_cm \pi(student-id, name, major)(students) \fn_cm \pi(name, advisor)(students) \fn_cm \pi(name, gpa, hours-attempted)(students)  Phép chiếu này có thể được hình dung như việc sinh ra một lát cắt theo chiều dọc của quan hệ toán hạng. Nếu phép toán sinh ra các bộ giống hệt nhau, những bộ này sẽ thường được loại bỏ khỏi quan hệ kết quả trong việc giữ chức năng set-like. Ví dụ về phép toán này với một thể hiện của quan hệ R như sau:   |  | | --- | |  | |  | |
| 3.1.1.3. Phép hợp |
| Là một phép toán hai ngôi được ký hiệu bằng biểu tượng hợp  \fn_cm \cup. Dạng chung của phép toán này là:   r \fn_cm \cups, với *r* và s là hai quan hệ khả hợp. Lược đồ của quan hệ kết quả chính là lược đồ của các quan hệ toán hạng. Kích cỡ của quan hệ kết quả (số bộ): \fn_cm \leq max\left \{ \left | r \right |,\left | s \right | \right \} *Ví dụ:*  r \fn_cm \cups                \fn_cm \pi(a, b)(r) \fn_cm \cup\fn_cm \pi(a, b)(s)  Phép hợp này cung cấp một phương tiện để trích lọc thông tin nằm trên hai quan hệ toán hạng, hai quan hệ này phải khả hợp với nhau. Khả hợp là một đặc tính mà hai điều kiện sau đây phải được thỏa mãn:   1. Hai quan hệ *r(R)* và *s(S)* trong biểu thức *r* \fn_cm \cup*s* phải cùng cấp, có nghĩa là chúng phải có cùng số lượng thuộc tính. 2. Miền giá trị của thuộc tính thứ (i) của r và thuộc tính thứ (i) của s phải giống nhau, cho mọi giá trị của i.   Ví dụ về toán tử hợp (union) với thể hiện quan hệ R, T, S, X được thể hiện dưới đây.   |  | | --- | |  | |  | |
| 3.1.1.4. Phép trừ |
| Là một phép toán hai ngôi được ký hiệu với biểu tượng: - . Dạng chung của phép toán này là:   r - s, với *r* và *s* là phải khả hợp. Lược đồ của quan hệ kết quả: là lược đồ của quan hệ toán hạng. Kích cỡ của quan hệ kết quả (số bộ): \fn_cm \leq |quan hệ r| (lực lượng của quan hệ r) *Ví dụ:*  r - s  Phép toán trừ này cho phép trích lọc các thông tin được chứa trong một quan hệ mà không chứa trong quan hệ thứ hai. Giống như phép toán hợp, phép trừ đòi hỏi rằng hai quan hệ toán hạng là khả hợp. Ví dụ về phép toán này được thể hiện như dưới đây.   |  | | --- | |  | |  | |
| 3.1.1.5. Phép tích đề các |
| Là phép toán hai ngôi với ký hiệu là: \fn_cm \times . Dạng chung của phép toán này là:   r \fn_cm \timess (không có hạn chế trên *r* và *s*) Lược đồ của quan hệ kết quả: lược đồ của *r* \fn_cm \timeslược đồ *s* với việc đặt lại tên một số thuộc tính. Kích cỡ của quan hệ kết quả (số bộ): > |quan hệ  *r|* và  >|quan hệ *s*| *Ví dụ:*  r \fn_cm \timess  Phép tích đề các cho phép kết hợp hai quan hệ bất kỳ thành một quan hệ đơn. Gợi nhớ rằng một quan hệ được định nghĩa là một tập con của tích đề các tập miền giá trị vì vậy điều này khiến bạn có một số ý tưởng về cách hành xử của phép toán tích đề các. Ví dụ về phép toán này với hai quan hệ T và X được thể hiện trong hình vẽ dưới đây.   |  | | --- | |  | |  | |  | | |  | | |
| 3.1.2. Biểu thức đại số quan hệ |
| Trong khi mỗi một toán tử đại số quan hệ cơ bản trong số năm toán tử trình bày ở trên có thể được sử dụng riêng rẽ để hình thành một truy vấn, sức mạnh thể hiện của chúng thực sự được cải thiện đáng kể khi chúng được kết hợp với nhau để hình thành các biểu thức truy vấn. Trước khi giới thiệu những phép toán bổ sung trong đại số quan hệ chúng ta sẽ xem xét việc tạo ra các tổ hợp phức tạp hơn của năm phép toán cơ bản, việc này cũng sẽ khiến cho bạn đánh giá cao hơn các phép toán bổ sung này.   |  | | --- | |  | |  |   Sử dụng các kỹ thuật để chuyển đổi một lược đồ E-R sang thành một tập các lược đồ quan hệ chúng ta có các lược đồ kết quả như sau:  S = STUDENTS(s#, name, age, major, gpa, hours\_completed) C = COURSES(c#, term, name, dept, enrollment) P = PROFESSORS(p#, name, dept, yrs\_teaching, area) TA = TAKES(s#, c#, term, grade) TE = TEACH(p#, c#, term)  Khi bạn bắt đầu viết một câu truy vấn trong một ngôn ngữ truy vấn mới, việc trực quan thông qua thể hiện dữ liệu của các quan hệ toán hạng  cần thiết cho một phép toán mà bạn đang thực hiện đôi khi rất có ích. Bạn nên tự trải nghiệm điều này qua  thể hiện của các quan hệ trên sao cho bạn có thể hình dung trực quan được tuy nhiên, đây cũng là thứ mà bạn sẽ cần loại bỏ khi bạn trở nên có kinh nghiệm hơn trong việc tạo các câu truy vấn bởi vì bạn không muốn làm ảnh hưởng đến bản thiết kế các truy vấn bởi sự trực quan một thể hiện quan hệ mà nó có thể không chứa tất cả các bộ có thể có cho câu truy vấn trong quá trình thực hiện. ***Ví dụ truy vấn 1:*** Tìm tên của tất cả các sinh viên học chuyên ngành công nghệ thông tin (Computer Science majors). Cách tiếp cận truy vấn này như sau:  -         Đầu tiên chọn tất cả  các sinh viên mà học chuyên ngành CS  r = \fn_cm \sigma(major = “Computer Science”)(S)  -          Tiếp đến chiếu kết quả trên lên thuộc tính *name:*  result = \fn_cm \pi(name)(r)  Biểu thức truy vấn hoàn chỉnh là: result =\fn_cm \pi(name)(\fn_cm \sigma(major = “Computer Science”)(S))  ***Ví dụ truy vấn 2:***  Tìm số hiệu sinh viên s# và tên của tất cả các sinh viên đã hoàn thành hơn 90 giờ học (theo hệ thống học tín chỉ). Cách tiếp cận truy vấn này như sau:  -         Đầu tiên chọn tất cả các sinh viên đã hoàn thành hơn 90 giờ học  r = \fn_cm \sigma(hours\_completed > 90)(S)  -         Tiếp đến chiếu kết quả trên lên thuộc tính mã hiệu sinh viên và tên sinh viên  result = \fn_cm \pi(s#, name)(r)  Biểu thức truy vấn hoàn chỉnh là  result = \fn_cm \pi(s#, name)(\fn_cm \sigma(hours\_completed > 90)(S))  ***Ví dụ truy vấn 3:*** Tìm tên của tất cả các sinh viên dưới 20 tuổi mà đã hoàn thành nhiều hơn 80 giờ học. Cách tiếp cận truy vấn này như sau:  -         Đầu tiên chọn tất cả các sinh viên đã hoàn thành hơn 80 giờ học và ít hơn 20 tuổi  r = \fn_cm \sigma((hours\_completed > 80) AND (age < 20))(S)  -         Tiếp đến chiếu kết quả trên lên thuộc tính tên sinh viên  result = \fn_cm \pi(name)(r)  Biểu thức truy vấn hoàn chỉnh là:  result = \fn_cm \pi(name)(\fn_cm \sigma((hours\_completed > 80) AND (age < 20))(S))  ***Ví dụ truy vấn 4:*** Tìm tên của tất cả các lớp học do khoa công nghệ thông tin hoặc khoa vật lý tổ chức. Cách tiếp cận truy vấn này như sau:  -         Đầu tiên lựa chọn tất cả các lớp học được khoa công nghệ thông tin tổ chức  r = \fn_cm \sigma((dept = Computer Science) or (dept = Physics))(C))  -         Tiếp đó chiếu kết quả trên lên thuộc tính tên  result = \fn_cm \pi(name)(r)  Biểu thức truy vấn hoàn chỉnh là:                                      result = \fn_cm \pi(name)(\fn_cm \sigma((dept = Computer Science)  \fn_cm \vee(dept = Physics))(C))  ***Ví dụ truy vấn 5:*** Tìm tên của tất cả các giáo sư đã dạy một lớp trong kỳ mùa thu 2002 (Fall 2002). Cách tiếp cận truy vấn như sau:  -         Đầu tiên đưa toàn bộ thông tin về giáo sư vào cùng với thông tin của khóa học.  -         Tiếp đến, lựa chọn chỉ những giáo sư và những lớp học liên quan tới nhau và thỏa mãn điều kiện.  -         Cuối cùng, chiếu kết quả trên lên tên sinh viên.  Biểu thức truy vấn hoàn chỉnh là: result = \fn_cm \pi(P.name)(\fn_cm \sigma((TE.term = Fall 2002)  \fn_cm \wedge(P.p# = TE.p#))(P \fn_cm \timesTE))  ***Ví dụ truy vấn 6:*** Tìm tên của tất cả các sinh viên đã tham gia một lớp học trong kỳ mùa thu 2003 mà được giảng bởi một giáo sư có hơn 20 năm kinh nghiệm giảng dạy. Cách tiếp cận truy vấn này như sau:  -         Đầu tiên ghép thông tin về giáo sư và lớp học vào với nhau  -         Tiếp đến, lựa chọn chỉ những sinh viên, giáo sư và khóa học liên quan và thỏa mãn điều kiện từ kết quả trên  -         Cuối cùng,  chiếu kết quả trên lên tên sinh viên.  Biểu thức truy vấn hoàn chỉnh là:  result = \fn_cm \pi(S.name)(\fn_cm \sigma((TA.term=Fall 2003)\fn_cm \wedge(P.yrs\_teaching>20)\fn_cm \wedge(S.s# = TA.s#)\fn_cm \wedge(P.p#=TE.p#)\fn_cm \wedge(TA.c# = TE.c#) \fn_cm \wedge(TA.term=TE.term))(S\fn_cm \timesP\fn_cm \timesTA\fn_cm \timesTE))  ***Ví dụ truy vấn 7:*** Tìm tên của tất cả các giáo sư hoặc thuộc khoa công nghệ thông tin hoặc có kinh nghiệm giảng dạy trên 20 năm. Biểu thức truy vấn hoàn chỉnh là:  result = [\fn_cm \pi(name)(\fn_cm \sigma(dept = Computer Science)(P))] \fn_cm \cup [\fn_cm \pi(name)(\fn_cm \sigma(yrs\_teaching > 20)(P))]  hoặc result = \fn_cm \pi(name)(\fn_cm \sigma((dept = Computer Science) \fn_cm \vee(yrs\_teaching > 20))(P))  ***Ví dụ truy vấn 8:*** Tìm mã hiệu của các sinh viên chỉ đăng ký học vào kỳ mùa xuân năm 2003. Biểu thức truy vấn hoàn chỉnh như sau  result = [\fn_cm \pi(TA.s#)(\fn_cm \sigma(TA.term = Spring 2003)(TA))] - [\fn_cm \pi(TA.s#)(\fn_cm \sigma(TA.term ¹ Spring 2003))(TA))]  Lưu ý biểu thức truy vấn sau đây không đúng cho câu truy vấn này  result =\fn_cm \pi(TA.s#)(\fn_cm \sigma(TA.term = Spring 2003))(TA))  bởi vì nó chỉ liệt kê những sinh viên có tham gia đăng ký học  vào kỳ mùa xuân 2003 và cũng có thể các sinh viên đó có tham gia vào các kỳ học khác nữa. |
| 3.1.3. Các phép toán mở rộng trong đại số quan hệ |
| Có thể chứng minh được rằng (mặc dù chúng ta sẽ không đề cập tới việc chứng minh trong bài giảng này) năm phép toán cơ bản được trình bày ở bài trước là đủ để thể hiện câu truy vấn trong đại số quan hệ. Tuy nhiên quá trình chứng minh không phát biểu rằng một số câu truy vấn phức tạp sẽ đòi hỏi các biểu thức truy vấn khó và dài dòng. Đó chính là nguyên nhân một số các phép toán khác ra đời trong đại số quan hệ để cung cấp sức mạnh thể hiện biểu thức truy vấn và cung cấp một sự đơn giản hóa trong việc thể hiện những câu truy vấn phức tạp. Chúng ta sẽ xem xét một số các phép toán quan trọng và chung  nhất và sẽ chỉ ra cách định nghĩa chúng theo cách thể hiện qua các phép toán cơ bản. Chúng ta lại xem xét lược đồ E-R cho một cơ sở dữ liệu để làm ví dụ cho các phép toán mở rộng này (như hình vẽ dưới đây)   |  | | --- | |  | |  |   Lược đồ cơ sở dữ liệu tương ứng với lược đồ E-R trên được thể hiện trong hình vẽ dưới đây   |  | | --- | |  | |  | |
| 3.1.3.1. Phép giao |
| Là một phép toán hai ngôi, được ký hiệu bởi biểu tượng  \fn_cm \cap. Dạng chung của phép toán này là: r \fn_cm \caps với *r* và *s* là hai quan hệ khả hợp  Lược đồ của quan hệ kết quả: là lược đồ của quan hệ toán hạng  Kích cỡ của quan hệ kết quả (số bộ):  \fn_cm \leq |r|  Định nghĩa:  r \fn_cm \caps = r - (r - s)  Ví dụ:   (\fn_cm \pi(p#)(SPJ)) \fn_cm \cap(\fn_cm \pi(p#)(P))  Phép giao sinh ra một tập các bộ mà chúng xuất hiện ở cả hai quan hệ toán hạng. Ví dụ của toán tử giao với thể hiện của R và S, R và T được thể hiện như dưới đây   |  | | --- | |  | |  | |
| 3.1.3.2. Phép kết nối |
| Như chúng ta đã thấy trong các biểu thức truy vấn trước đó có liên quan tới tích Đề các, chúng ta phải cung cấp thêm các phép toán lựa chọn để loại bỏ những tổ hợp các bộ không liên quan tới nhau trong kết quả của tích Đề các (chúng không có ý nghĩa giống như kiểu một vận chuyển của một linh kiện cụ thể nào đó được kết hợp với thông tin của linh kiện nhưng những thông tin này không phải của linh kiện cần được vận chuyển). Điều này thường xuyên xảy ra đến mức mà việc kết hợp tích Đề các và các phép toán lựa chọn được phát triển thành một phép toán kết nối (join operation). Có nhiều loại phép toán kết nối khác nhau bao gồm kết nối theta, kết nối bằng, kết nối tự nhiên, kết nối ngoài và nửa kết nối. Chúng ta sẽ xem xét lần lượt mỗi phép kết nối này và các điều kiện được chúng sử dụng.  ***Phép kết nối Theta và kết nối bằng*** Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là  \fn_cm \triangleright \triangleleft (predicate), trong đó predicate biểu diễn một mệnh đề nào đó. Lược đồ của quan hệ kết quả là ghép nối các quan hệ toán hạng. Định nghĩa                r \fn_cm \triangleright \triangleleft(predicate) s \fn_cm \equiv\fn_cm \sigma(predicate)(r \fn_cm \times s) Ví dụ: phép kết nối bằng r \fn_cm \triangleright \triangleleft(color='blue' \fn_cm \wedgesize=3) s và phép kết nối theta r \fn_cm \triangleright \triangleleft(color='blue' \fn_cm \wedgesize>3) s Phép kết nối theta là một cách viết ngắn gọn cho một tích đề các và một phép chọn được thực hiện sau đó. Phép kết nối bằng là một trường hợp đặc biệt của phép kết nối theta trong đó tất cả các điều kiện trong mệnh đề đều sử dụng điều kiện bằng. Cả phép toán kết nối bằng và kết nối theta đều không loại bỏ các bộ dư thừa vì vậy việc lọai bỏ các bộ dư thừa phải được quản lý một cách tường minh thông qua các mệnh đề điều kiện. Ví dụ về toán tử kết nối theta được thể hiện như dưới đây với các thể  hiện của quan hệ R và S.   |  | | --- | |  | |  | |
| 3.1.3.2.1. Phép kết nối tự nhiên |
| Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là r \* s Lược đồ của quan hệ kết quả là ghép nối các quan hệ toán hạng trong đó các thuộc tính với tên chung chỉ được xuất hiện một lần trong lược đồ.  Định nghĩa: r \* s \fn_cm \equiv r \fn_cm \triangleright \triangleleft (r.commonAttribute=s.commonAttribute) s  Ví dụ: phép kết nối tự nhiên s \* spj \* p Phép kết nối tự nhiên thực hiện một phép kết nối bằng trên tất cả các thuộc tính có chung tên trùng nhau của hai quan hệ toán hạng. Cấp của quan hệ kết quả của một kết nối bằng là tổng của cấp quan hệ các toán hạng trừ đi số lượng các thuộc tính chung của hai quan hệ toán hạn. Nói một cách khác, một thể hiện của mỗi thuộc tính chung được lọa bỏ khỏi quan hệ kết quả. Phép kết nối tự nhiên có lẽ là phép toán phổ biến nhất trong số các dạng phép kết nối. Nó thực sự rất có ích trong việc loại bỏ các bộ dư thừa. Những thuộc tính có chung tên giữa các quan hệ toán hạng thường được gọi với cái tên là các ***thuộc tính kết nối***. Ví dụ về toán tử kết nối tự nhiên được thể  hiện như dưới đây với thể hiện của quan hệ R và S   |  | | --- | |  | |  | |
| 3.1.3.2.2. Phép kết nối ngoài |
| Là loại phép toán hai ngôi sử dụng ký hiệu và dạng chung với ba loại kết nối ngoài như sau:    Kết nối ngoài bên trái: r \fn_cm \supset\triangleleft  s Kết nối ngoài bên phải: r \fn_cm \triangleright\subset s Kết nối ngoài đầy đủ: r \fn_cm \supset\triangleleft \fn_cm \triangleright\subset s  Lược đồ quan hệ kết quả là sự ghép nối các quan hệ toán hạng.  Định nghĩa:  r \fn_cm \supset\triangleleft  s \fn_cm \equivkết nối tự nhiện của r và s với các bộ của r không tương ứng với s vẫn được giữ lại trong kết quả. Tất cả các giá trị bị khuyết ở thuộc tính của s đều được gán cho giá trị rỗng. r \fn_cm \triangleright\subset s \fn_cm \equivkết nối tự nhiện của r và s với các bộ của s không tương ứng với r vẫn được giữ lại trong kết quả. Tất cả các giá trị bị khuyết ở thuộc tính của r đều được gán cho giá trị rỗng. r \fn_cm \supset\triangleleft \fn_cm \triangleright\subset s \fn_cm \equivkết nối tự nhiện của r và s với các bộ từ r và s không tương ứng với nhau vẫn được giữ lại trong kết quả. Tất cả các giá trị bị khuyết ở thuộc tính của r hoặc s đều được gán cho giá trị rỗng. Ví dụ:  Cho r(A,B) = {(a, b), (c, d), (b,c)} và cho s(A,C) = {(a, d), (s, t), (b, d)} thì r \fn_cm \supset\triangleleft  s = (A,B,C) = {(a,b,d), (b,c,d), (c,d,null)}, r \fn_cm \triangleright\subset s = (A,B,C) = {(a,b,d), (b,c,d), (s,null,t)}, và  r \fn_cm \supset\triangleleft \fn_cm \triangleright\subset s = (A,B,C) = {(a,b,d), (b,c,d), (s,null,t), (c,d,null)},  Ví dụ về phép kết nối ngoài thông quan các thể hiện của quan hệ R và S được thể hiện như dưới đây   |  | | --- | |  | |  | |
| 3.1.3.2.3. Phép nửa kết nối |
| Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là r \fn_cm \triangleright(*predicate*) s, trong đó *predicate* là một mệnh đề. Lược đồ của quan hệ kết quả là lược đồ của r. Định nghĩa: r \fn_cm \triangleright(*predicate*) s \fn_cm \equiv\fn_cm \pi(các thuộc tính của r) (r \fn_cm \triangleright\triangleleft(*predicate*) s)  Phép toán nửa kết nối thực hiện một phép kết nối với hai quan hệ toán hạng và chiếu lên các thuộc tính của quan hệ toán hạng bên trái. Ưu điểm chính của phép toán nửa kết nối là nó sẽ giảm số lượng các bộ cần phải quản lý để hình thành phép kết nối. Điều này thực sự rất có ích trong môi trường phân tán. Trong dạng chung của phép kết nối này, phép nửa kết nối là dạng nửa kết nối theta, nửa kết nối bằng và nửa kết nối tự nhiên có thể được định nghĩa theo một cách thức tương tự. *Ví dụ về toán tử nửa kết nối được thể hiện như sau*   |  | | --- | |  | |  | |
| 3.1.3.2.4. Phép chia |
| Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là  r \fn_cm \div s với r({A}) và s({B}). Lược đồ quan hệ kết quả: C với C = A - B  Định nghĩa: r \fn_cm \divs \fn_cm \equiv \pi(A-B)(r) - (\fn_cm \pi(A-B)((\fn_cm \pi(A-B)(r) \fn_cm \times s) - r))  Ví dụ:  Cho r(A,B,C) = {(a,b,c), (a,d,d), (a,b,d), (a,c,c), (a,d,d)} và s(C) = {(c), (d)} thì  r \fn_cm \divs = t(A,B) = {(a,b)}  Yêu cầu cho phép toán chia   1. Quan hệ r được xác định trên tập các thuộc tính A và quan hệ s được xác định trên tập thuộc tính B sao cho *B* \fn_cm \subseteq *A*. 2. Cho C là tập các thuộc tính trong A-B   Với các ràng buộc này phép toán chia được định nghĩa là:  một bộ *t*  là của *r* \fn_cm \div*s* nếu cho mỗi bộ *ts* trong *s* có một bộ *tr* trong *r* thỏa mãn cả hai điều kiện sau đây:             tr [C] = ts [C]  và tr [A-B] = t[A-B] Ví dụ về toán tử chia được thể hiện như dưới đây   |  | | --- | |  | |  | |
| 3.1.3.2.5. Tính hữu ích của các toán tử mở rộng |
| Các toán tử mở rộng của đại số quan hệ được coi là dư thừa bởi vì tất cả chúng đều được định nghĩa dựa trên năm phép toán cơ bản. Tuy nhiên tính hữu ích của chúng được mô tả tốt nhất qua phép toán chia. Xem xét câu truy vấn dựa trên cơ sở dữ liệu linh kiện-công việc-vận chuyển được cho ở đầu bài để thấy được sự hữu ích này. Cây truy vấn: tìm số hiệu của các nhà cung cấp mà chuyển mọi linh kiện. Một giải pháp cho câu truy vấn này chỉ dùng năm toán tử cơ bản là:  •         T1 = \fn_cm \pi (s#, p#)(spj)   // tất cả các cặp (s#,p#) cho các vận chuyển thực tế  •         T2 = \fn_cm \pi(p#)(p)  //tất cả (p#)  {tất cả các linh kiện tồn tại cho dù được chuyển hoặc không}  •         T3 = \fn_cm \pi(s#)(T1) \fn_cm \times T2 // tất cả s# trong T1 cặp với mỗi bộ trong T2  {spj.s#, p.p#}  •         T4 = T3 – T1 // tất cả các bộ trong T3 mà không trong T1  •         T5 = T1 – T4 // tất cả các bộ trong T1 mà không trong T4.  •         T6 = \fn_cm \pi(s#)(T5)  // giải pháp  Giải pháp hoàn chỉnh là:            \fn_cm \pi(s#)(spj) - (\fn_cm \pi(s#)((\fn_cm \pi(s#)(spj) \fn_cm \times p) - spj))  ***Giải pháp sử dụng các phép toán mở rộng (dư thừa)***               (\fn_cm \pi(s#, p#)(spj)) ¸ (\fn_cm \pi(p#)(p)) |
| 3.1.4. Một số các truy vấn sử dụng chỉ năm phép toán cơ bản |
| Cho S, P, SPJ 1. Tìm tất cả số hiệu của các nhà cung cấp nằm tại Milan hoặc chuyển hàng tới bất kể công việc nào với số lượng lớn hơn 40  [\fn_cm \pi(s#)(\fn_cm \sigma(city = Milan)(S))] \fn_cm \cup [\fn_cm \pi(s#)(\fn_cm \sigma(qty > 40)(SPJ))]  2. Tìm tất cả mã hiệu của các nhà cung cấp mà chỉ vận chuyển các linh kiện màu đỏ [\fn_cm \pi(S.name)(\fn_cm \sigma((SPJ.s#=S.S#)\fn_cm \wedge(SPJ.p#=P.p#)\fn_cm \wedge(color=red))(SPJ\fn_cm \timesS\fn_cm \timesP))] -[\fn_cm \pi(S.name)(\fn_cm \sigma((SPJ.s#=S.S#)\fn_cm \wedge(SPJ.p#=P.p#)\fn_cm \wedge(color ¹ red))(SPJ\fn_cm \timesS\fn_cm \timesP))] 3. Tìm tên của các nhà cung cấp cùng thành phố với công việc mà họ vận chuyển các linh kiện đến cho. T1 = (S \fn_cm \timesSPJ \fn_cm \timesJ) T2 = \fn_cm \sigma(S.s# = SPJ.s#)(T1)   // lựa chọn các bộ có cùng mã nhà cung cấp s# T3 = \fn_cm \sigma(J.j# = SPJ.j#)(T2)   // lựa chọn các bộ có cùng mã công việc j# T4 = \fn_cm \sigma(J.city = S.city)(T3)  // lựa chọn các bộ có cùng thành phố T5 = \fn_cm \pi(S.name)(T4)  // chiếu lên tập thuộc tính cuối cùng 4. Tìm số hiệu của tất cả các linh kiện được vận chuyển bởi cả hai nhà cung cấp “S1” và “S2”. Lưu ý biểu thức sau đây không đúng bởi điều kiện lựa chọn của phép chọn luôn bằng false:  \fn_cm \pi(p#)(\fn_cm \sigma((s# = S1) \fn_cm \wedge(s# = S2))(SPJ)) Biểu thức đúng cho truy vấn này như sau [\fn_cm \pi(p#)(\fn_cm \sigma(s#=S1)(SPJ)] – ([\fn_cm \pi(p#)(\fn_cm \sigma(s#=S1)(SPJ)] - [\fn_cm \pi(p#)(\fn_cm \sigma(s#=S2)(SPJ)]) 5.  Tìm số hiệu các  nhà cung cấp mà cung cấp một linh kiện màu xanh và một linh kiện màu đỏ. Lưu ý biểu thức sau đây là không đúng vì điều kiện lựa chọn nếu viết như vậy không thể hiện được yêu cầu của câu truy vấn và đồng thời giá trị trả về của biểu thức điều kiện luôn là false nên kết quả của truy vấn sẽ là rỗng. Cách đúng để thực hiện truy vấn đó trong đại số quan hệ như sau T1 = \fn_cm \pi(s#)(\fn_cm \sigma((color = blue) \fn_cm \wedge(SPJ.p# = P.p#)) (P \fn_cm \timesSPJ)) T2 = \fn_cm \pi(s#)(\fn_cm \sigma((color = red) \fn_cm \wedge(SPJ.p# = P.p#)) (P \fn_cm \timesSPJ))             T3 = T2 – T1             T4 = T2 – T3 6. Tìm tất cả các cặp (s#, j#) của những nhà cung cấp và công việc có cùng thành phố nhưng nhà cung cấp đó không vận chuyển linh kiện nào tới cho công việc này. T1 = \fn_cm \pi(s#, j#)(\fn_cm \sigma(S.city = J.city)(S \fn_cm \timesJ))  // tất cả cặp (s#,j#) ở cùng một thành phố T2 = \fn_cm \pi(s#, j#)(\fn_cm \sigma((S.city = J.city) \fn_cm \wedge(SPJ.j# = J.j#) \fn_cm \wedge(SPJ.s# = S.s#))(S \fn_cm \timesSPJ \fn_cm \times  J))   //T2 chứa tất cả những cặp (s#,j#) thể hiện việc vận chuyển linh kiện của các nhà cung cấp tới công việc tương ứng trong cùng thành phố.             T3 = T1 – T2 |
| 3.1.5. Toán tử đặt lại tên |
| Không giống như các quan hệ cơ bản trong cơ sở dữ liệu, các quan hệ trung gian được sản sinh ra từ các kết quả đánh giá của một truy vấn, không có tên để tham chiếu tới. Trừ khi các quan hệ trung gian này được lưu trữ một cách tường minh, nó sẽ không tồn tại sau khi truy vấn này thực thi xong. Tuy nhiên trong nhiều trường hợp, việc lưu trữ các quan hệ trung gian này lại rất có ích vì nó có thể chứa một tập các bộ là kết quả của các truy vấn liên quan hoặc nó sẽ chứa đựng một tập các bộ có thể được sử dụng để đánh giá một truy vấn khác. Lưu trữ các quan hệ trung gian đôi khi có nghĩa là một công việc tương tự không cần thiết phải thực hiện lại. Phép toán đặt tên được biểu diễn bởi một chữ cái Hy lạp thường rho (\fn_cm \rho) và nó được sử dụng để đặt tên cho cả các quan hệ và các thuộc tính. Dạng chung đầu tiên của phép toán đặt tên cho các quan hệ là \fn_cm \rhotên mới cho quan hệ (quan hệ). Ví dụ \fn_cm \rhox(r) đặt lại tên cho quan hệ *r* thành *x.* Dạng thứ hai cho phép toán đặt lại tên cho cả quan hệ và thuộc tính giả sử rằng quan hệ toán hạng có cấp n    \fn_cm \rhotên mới của quan hệ (A1, A2, …, AN)(quan hệ). Ví dụ \fn_cm \rho*x*(*one, two, …, last*)(*r*) sẽ đổi tên quan hệ *r* thành quan hệ *x* và *n* thuộc tính của quan hệ *x* thành *một, hai, …, cuối cùng*. **Các truy vấn để thực hành việc sử dụng tất cả các toán tử của đại số quan hệ** 1.      Liệt kê tất cả các cặp số hiệu nhà cung cấp ở cùng một thành phố:  \fn_cm \pi(s.s#, x.s#)(s \fn_cm \triangleright \triangleleft\fn_cm \rhox(s)) 2.      Liệt kê mọi vận chuyển liên quan tới một linh kiện màu xanh:  spj \fn_cm \triangleright \triangleleft(\fn_cm \sigma(color='green') p)  3.      Liệt kê số hiệu của tất cả các nhà cung cấp mà vận chuyển một linh kiện được sản xuất ở cùng một thành phố  với nhà cung cấp:  \fn_cm \pi(s.s#)(s \fn_cm \triangleright \triangleleftp \fn_cm \triangleright \triangleleftj)  4.      Liệt kê tên của các nhà cung cấp mà vân chuyển tất cả các linh kiện màu xanh dương:  \fn_cm \pi(s.name)(s \fn_cm \triangleright \triangleleft  (\fn_cm \pi(s.s#,p.p#)spj) \fn_cm \div\fn_cm \pi(p.p#)(\fn_cm \sigma(color='blue') p)))  5.      Liệt kê số hiệu của các nhà cung cấp mà chỉ vận chuyển các linh kiện màu xanh dương: [\fn_cm \pi(s.#)(spj \fn_cm \triangleright \triangleleft  (\fn_cm \sigma(color='blue') p))] -[\fn_cm \pi(s.#)(spj \fn_cm \triangleright \triangleleft  (\fn_cm \sigma(color\fn_cm \neq'blue') p)))] Hãy nhận xét kết quả biểu thức trên với biểu thức \fn_cm \pi(s.#)(s \fn_cm \triangleright \triangleleft  (\fn_cm \pi(s.s#,p.p#)spj) \fn_cm \div\fn_cm \pi(p.p#)(\fn_cm \sigma(color='blue') p))) |
| 3.2. Ngôn ngữ truy vấn có cấu trúc SQL |
| 3.2.1. Giới thiệu chung về ngôn ngữ SQL |
| SQL là viết tắt của từ tiếng Anh structural query language đã trở thành một ngôn ngữ tuân thủ chuẩn de facto cho việc tạo ra và truy vấn các cơ sở dữ liệu quan hệ. Nó được chấp nhận bởi viện tiêu chuẩn quốc gia Hoa Kỳ ANSI và tổ chức tiêu chuẩn quốc tế ISO cũng như thoả mãn các tiêu chuẩn xử lý thông tin của liên bang FIPS. Giữa những năm 1974 và 1979, các nhân viên làm việc trong phòng nghiên cứu thí nghiệm của công ty IBM tại San Jose, bang California, Hoa Kỳ đã tiến hành phát triển hệ thống có tên là R, ngay sau khi bài báo truyền thống định nghĩa cơ sở dữ liệu quan hệ được công bố. Mục tiêu của hệ thống R là chứng minh tính khả thi việc cài đặt mô hình quan hệ trong một hệ quản trị cơ sở dữ liệu. Họ sử dụng một ngôn ngữ có tên là SEQUEL (Structured English Query Language), là một ngôn ngữ nối tiếp của SQUARE (Specifying Queries as Relational Expressions), cả hai đều được phát triển tại IBM ở San Jose. Sau đó SEQUEL được đổi tên thành SQL trong dự án xây dựng hệ thống R này. Bản thân hệ thống R chưa bao giờ được đưa ra thương mại hoá nhưng nó trực tiếp đưa đến sự phát triển của SQL/DS (bản SQL chạy trên hệ điều hành DOS năm 1981, và trên VE OS một phiên bản máy ảo vào năm 1982) là hệ quản trị cơ sở dữ liệu quan hệ được thương mại hoá đầu tiên của IBM. Tuy nhiên IBM không phải là công ty đưa ra phiên bản cài đặt thương mại đầu tiên cho một hệ quản trị cơ sở dữ liệu quan hệ mà vinh dự đó được chuyển cho Oracle (cho ra đời phần mềm quan hệ) năm 1979. Ngày này hệ thống quản trị cơ sở dữ liệu quan hệ của tất cả các công ty sản xuất đều dựa trên SQL. Mỗi nhà sản xuất cung cấp tất cả các đặc tính chuẩn của SQL nhưng hầu hết mỗi nhà sản xuất đều cung cấp thêm những đặc tính riêng biệt của mình, được gọi là mở rộng của SQL chuẩn. Những mở rộng này dẫn đến các vấn đề về khả năng dùng lẫn khi đem các ứng dụng dựa trên SQL sang các hệ quản trị cơ sở dữ liệu khác. Các nhà sản xuất cố gằng phân biệt các phiên bản SQL của họ thông qua những mở rộng này. Phiên bản hiện thời của chuẩn ANSO cho SQL là SQL-99 (hay còn được biết đến như SQL3). Chuẩn này cũng được chấp nhận bởi ISO. Mặc dù rất nhiều bản mở rộng của SQL tồn tại, chúng ta sẽ xem xét bản chất bên trong của SQL, là nội dung sẽ tìm thấy trong mọi hệ quản trị cơ sở dữ liệu. Cho dù bạn sử dụng Oracle, Microsoft SQL Server, IBM’s DB2, Microsoft Acess hay MySQL hoặc bất kể hệ quản trị cơ sở dữ liệu đuợc xây dựng tốt nào, bạn đều có thể tìm hiểu hệ thống đó một cách nhanh chóng với những thông tin sẽ được đề cập tới trong bài giảng này. SQL là một ngôn ngữ cơ sở dữ liệu quan hệ đầy đủ vì nó bao gồm cả hai loại ngôn ngữ: ngôn ngữ định nghĩa dữ liệu DDL và ngôn ngữ thao tác dữ liệu DML. Chúng ta sẽ tìm hiểu các thành phần của cả hai loại ngôn ngữ này của SQL. Nếu bạn dùng Access, ví dụ như vậy, bạn sẽ cần biết ít về DDL hơn khi bạn dùng Oracle9i hoặc MySQL. Bảng dưới đây sẽ tóm tắt các câu lệnh trong phân DDL của SQL. Mỗi mục vào của bảng không thể hiện trật tự bạn sử dụng các câu lệnh này, đơn giản chỉ đưa ra cho bạn một danh sách tóm tắt các lệnh có sẵn của DDL nhưng cũng không phải là một danh sách đầy đủ.   |  |  | | --- | --- | | Câu lệnh hoặc lựa chọn | Mô tả | | CREATE SCHEMA AUTHORIZATION | Tạo ra một lược đồ cơ sở dữ liệu | | CREATE TABLE | Tạo ra một bảng mới trong lược đồ cơ sở dữ liệu của người dùng | | NOT NULL | Ràng buộc đảm bảo một cột sẽ không có giá trị rỗng | | UNIQUE | Ràng buộc đảm bảo một cột sẽ không có giá trị trùng lặp | | PRIMARY KEY | Định nghĩa một khoá chính cho một bảng | | FOREIGN KEY | Định nghĩa một khoá ngoại cho một bảng | | DEFAULT | Định nghĩa một giá trị ngầm định cho một cột (khi không đưa một giá trị nào đó vào) | | CHECK | Ràng buộc được sử dụng để kiểm tra tính đúng đắn của dữ liệu trong một cột | | CREATE INDEX | Tạo chỉ mục cho một bảng | | CREATE VIEW | Tạo một tập con động cần thiết các hàng/cột từ một hoặc nhiều bảng | | ALTER TABLE | Thay đổi định nghĩa của một bảng: thêm vào/xoá bỏ/cập nhật các thuộc tính hoặc ràng buộc | | DROP TABLE | Xoá vĩnh viễn một bảng (cùng với dữ liệu của nó) từ một lược đồ cơ sở dữ liệu | | DROP INDEX | Xoá vĩnh viễn một tập chỉ mục | | DROP VIEW | Xoá vĩnh viễn một khung nhìn |   Trước khi bạn có thể sử dụng một hệ quản trị cơ sở dữ liệu, hai công việc cần được thực hiện đó là: (1) tạo một cấu trúc cơ sở dữ liệu và (2) tạo các bảng để lưu trữ dữ liệu của người dùng cuối. Hoàn thành công việc thứ nhất liên quan tới việc kiến tạo ra các tệp vật lý để lưu cơ sở dữ liệu. Hệ quản trị cơ sở dữ liệu sẽ tự động tạo ra các bảng từ điển dữ liệu và tạo ra một người quản trị cơ sở dữ liệu ngầm định (DBA). Việc tạo ra các tệp vật lý đòi hỏi sự tương tác giữa hệ điều hành và hệ quản trị cơ sở dữ liệu. Vì vậy, tạo ra cấu trúc cơ sở dữ liệu là một đặc tính có sự khác nhau từ một hệ quản trị cơ sở dữ liệu này sang hệ khác. Với một ngoại lệ là có thể tạo ra cơ sở dữ liệu, hầu hết các nhà cung cấp hệ thống quản trị cơ sở dữ liệu sử dụng bản SQL có khác một chút với bản SQL chuẩn của ANSI. Mặc dù vậy nhưng cũng chỉ thỉnh thoảng bạn mói gặp những sự khác nhau nhỏ trong cú pháp của các câu lệnh SQL. Ví dụ, hầu hết để yêu cầu mọi câu lệnh SQL được kết thúc bởi dấu chấm phẩy (;) tuy nhiên một số bản cài đặt SQL không sử dụng dấu (;). Hầu hết những sự khác nhau chung về cú pháp sẽ được chỉ ra trong bài giảng này hoặc ít nhất cũng liệt kê những sự khác nhau do người viết nhận thức được. |
| 3.2.2. Ngôn ngữ định nghĩa dữ liệu DDL trong SQL |
| Chúng ta sẽ dùng cơ sở dữ liệu dưới đây để mô tả cho cách dử dụng của các lệnh DDL của SQL. Cơ sở dữ liệu này liên quan một chút tới cơ sở dữ liệu về nhà cung cấp-linh kiện-công việc-vận chuyển. Các qui định về nghiệp vụ của hệ thống này như sau:   1. Một khách hàng có thể có nhiều yêu cầu trả tiền. mỗi bản yêu cầu trả tiền (invoice) chỉ được tạo ra bởi một khách hàng 2. Một invoice chứa một hoặc nhiều dòng. mỗi dòng của invoice liên quan tới một invoice 3. Mỗi dòng invoice là cho một mặt hàng. Một mặt hàng có thể tìm thấy ở nhiều dòng khác nhau. Một nhà cung cấp có thể cung cấp nhiều mặt hàng. Một vài nhà cung cấp có thể không cung cấp mặt hàng nào cả. 4. Nếu một mặt hàng được cung cấp bởi một nhà cung cấp, thì mặt hàng đó chỉ được cung cấp bởi duy nhất nhà cung cấp đó. 5. Một số mặt hàng không được cung cấp bởi nhà cung cấp nào cả mà chúng được công ty tự sản xuất (in-house) hoặc đuợc cung cấp qua những cách khác.   Cơ sở dữ liệu của hệ thống ví dụ trên được thiết kế nhanh trong Access và được thể hiện trong hình vẽ dưới đây trong đó Customer để chỉ thực thể khách hàng; Invoice để thể hiện các yêu cầu trả tiền của khách hàng; Line là bảng để lưu thông tin các dòng của các invoice; Vendor để lưu thông tin của các nhà cung cấp; Product để lưu thông tin của các mặt hàng.   |  | | --- | |  | |  |   Các qui định về ký pháp cho câu lệnh SQL được trình bày trong bài giảng được thể hiện trong bảng sau đây   |  |  | | --- | --- | | **Ký pháp** | **Mô tả** | | Viết hoa | từ khoá cần thiết cho câu lệnh SQL | | *viết nghiêng* | Một tham số do người dùng cung cấp- thường là cần thiết | | {a | b | ... } | Một tham số bắt buộc, sử dụng một trong số danh sách lựa chọn | | [...] | Một tham số lựa chọn-mọi thứ trong ngoặc vuông đều là lựa chọn | | *tablename* | Tên của bảng | | *column* | Tên của một thuộc tính trong một bảng | | *data type* | Một định nghĩa kiểu dữ liệu đúng | | *constraint* | Một định nghĩa  ràng buộc  đúng cách | | *condition* | Một biểu thức điều kiện đúng cách-trả về giá trị đúng hoặc sai | | *columnlist* | Một hoặc nhiều tên cột hoặc biểu thức được phân cách nhau bởi dấu phảy | | *tablelist* | Một hoặc nhiều tên bảng được phân cách nhau bởi dấu phảy | | *conditionlist* | Một hoặc nhiều biểu thức điều kiện được phân cách nhau bởi dấu phảy | | *expression* | Một giá trị đơn (ví dụ 76 or ‘married’) hoặc một công thức (ví dụ., price-10) |  |  |  | | --- | --- | | **Ký pháp** | **Mô tả** | | Viết hoa | từ khoá cần thiết cho câu lệnh SQL | | *viết nghiêng* | Một tham số do người dùng cung cấp- thường là cần thiết | | {a | b | ... } | Một tham số bắt buộc, sử dụng một trong số danh sách lựa chọn | | [...] | Một tham số lựa chọn-mọi thứ trong ngoặc vuông đều là lựa chọn | | *tablename* | Tên của bảng | | *column* | Tên của một thuộc tính trong một bảng | | *data type* | Một định nghĩa kiểu dữ liệu đúng | | *constraint* | Một định nghĩa  ràng buộc  đúng cách | | *condition* | Một biểu thức điều kiện đúng cách-trả về giá trị đúng hoặc sai | | *columnlist* | Một hoặc nhiều tên cột hoặc biểu thức được phân cách nhau bởi dấu phảy | | *tablelist* | Một hoặc nhiều tên bảng được phân cách nhau bởi dấu phảy | | *conditionlist* | Một hoặc nhiều biểu thức điều kiện được phân cách nhau bởi dấu phảy | | *expression* | Một giá trị đơn (ví dụ 76 or ‘married’) hoặc một công thức (ví dụ., price-10) | |
| 3.2.2.1. Câu lệnh tạo cấu trúc bảng trong SQL |
| Sử dụng câu lệnh CREATE TABLE để tạo cấu trúc bảng,  cú pháp tùy thuộc vào hệ quản trị cơ sở dũ liệu nhưng thông thường có dạng sau:   |  | | --- | |  | |  |   Ví dụ tạo một bảng VENDOR của cơ sở dữ liệu ví dụ được mô tả ở trên   |  | | --- | |  | |  |   trong câu lệnh trên ta đã dùng ký pháp để tạo khoá chính V\_CODE là PRIMARY KEY, sau đó ta sẽ tạo tiếp đến bảng PRODUCT như sau   |  | | --- | |  | |  |   Trong ví dụ trên khoá ngoại V-CODE được tạo ra bằng cách dùng từ khoá FOREIGN KEY, nó được tham chiếu tới bảng VENDOR bằng cách dùng từ khoá REFERENCE với ràng buộc ON UPDATE CASCADE (sẽ được mô tả sau). Ví dụ tiếp theo là tạo ra bảng CUSTOMER như sau   |  | | --- | |  | |  |   trong ví dụ này ta dùng thêm các cách để tạo ràng buộc cho mỗi cột riêng rẽ, tạo giá trị ngầm định bằng từ khoá DEFAULT, tạo ràng buộc kiểm tra tính đúng đắn của dữ liệu nhập vào bằng việc dùng hàm có từ khoá CHECK (kiểm tra các giá trị nhập vào của CUS\_AREACODE có nằm trong tập 3 giá trị (‘615’,’713’,’931’). Tạo ràng buộc cho toàn bảng dùng từ khoá CONSTRAINT. Bây giờ là tạo bảng INVOICE với câu lệnh như sau   |  | | --- | |  | |  |   trong ví dụ này ràng buộc CHECK được sử dụng để kiểm tra tính hợp lệ của ngày invoice có lớn hơn 1/1/2002 không. Hàm TO\_DATE cần hai tham số bao gồm ngày cụ thể và định dạng ngày được sử dụng, cuối cùng chúng ta tạo bảng LINE như sau   |  | | --- | |  | |  |   trong ví dụ này một ràng buộc trên toàn bảng được tạo ra để ngăn chặn không có hai dòng trong một invoice giống nhau. Ngoài ra việc sử dụng ON DELETE CASCADE được khuyến cáo nên dùng cho các thực thể yếu để đảm bảo rằng việc xoá một dòng (tương ứng với một mã invoice) trong thực thể chính (khoẻ) sẽ gây ra việc xoá tự động các dòng tương ứng (các dòng thể hiện các mặt hàng của mã invoice bị xoá) trong thực thể yếu phụ thuộc vào thực thể chính đó.  **Một số các lưu ý đối với việc tạo bảng**   * Đối với cơ sở dữ liệu ví dụ trên, bảng PRODUCT chứa một khoá ngoại tham chiếu tới bảng VENDOR. Vì vậy, bảng VENDOR phải được tạo trước. Nói chung, các bảng nằm bên phía lực lượng 1 của một quan hệ 1-nhiều phải được tạo trước khi bảng bên phía lực lượng nhiều có thể được tạo ra. * Với hệ quản trị cơ sở dữ liệu Oracle nếu sử dụng cách định nghĩa khoá chính bằng từ khoá PRIMARY KEY thì không cần đưa yêu cầu NOT NULL và UNIQUE vào câu lệnh tạo bảng nữa. Thực tế, bạn sẽ nhận được một thông báo lỗi nếu bạn làm như vậy. * Ràng buộc ON UPDATE CASCADE là một phần của chuẩn ANSI nhưng nhiều hệ quản trị cơ sở dữ liệu không hỗ trợ nps. Oracle là một trong số những hệ thống quản trị không hỗ trợ tính năng này. * Nếu khoá chính là một khoá ghép, tất cả các thuộc tính của khoá được chứa trong một dấu ngoặc đơn và được phân tách nhau bởi dấu phảy. Ví dụ, bảng LINE có khoá chính được định nghĩa như sau                     PRIMARY KEY (inv\_number, line\_number)   * Hỗ trợ ràng buộc tham chiếu rất đa dạng, thay đổi từ hệ quản trị này sang hệ quản trị khác o       MS Access, SQL Server và Oracle hỗ trợ ON DELETE CASCADE (để tránh trường hợp dị thường khi xoá bộ) o       MS Access, SQL Server hỗ trợ ON UPDATE CASCADE (để tránh dị thường khi cập nhật) o       Oracle không hỗ trợ ON UPDATE CASCADE o       MS Access, SQL Server không hỗ trợ SET NULL o       Oracle hỗ trợ SET NUL   MS Access không hỗ trợ ON DELETE CASCADE hoặc ON UPDATE CASCADE tại mức câu lệnh SQL tuy nhiên nó lại hỗ trợ thông quan giao diện cửa số quan hệ. Chúng ta tạm dừng xem xét cách sử dụng các câu DDL của SQL ở đây để chuyển sang giới thiệu về phần ngôn ngữ thao tác dữ liệu DML của SQL trước, sau đó sẽ quay lại DDL với những câu lệnh phức tạp hơn. |
| 3.2.3. Ngôn ngữ thao tác dữ liệu DML của SQL |
| Ngôn ngữ thao tác dữ liệu của SQL có thể được chia ra làm hai phần tách riêng nhưng vẫn chung nhau ở một số phạm vi nào đó. Hai phần này là các câu lệnh DML không truy vấn và các câu lệnh truy vấn dữ liệu. Ngôn ngữ thao tác không truy vấn cho phép bạn thêm dữ liệu vào bảng (INSERT), sửa đổi dữ liệu (UPDATE), xoá dữ liệu từ các bảng (DELETE) và thực hiện những thay đổi vĩnh viễn  (COMMIT) và huỷ những thay đổi (tới một mức độ nào đó với ROLLBACK). Các câu lệnh truy vấn DML chắc chắn bao gồm cấu lệnh đơn SELECT với rất nhiều các mệnh đề lựa chọn khác nhau. Chúng ta sẽ xem xét các câu lệnh không truy vấn của DML trước. Tóm tắt các câu lệnh DML của SQL được mô tả trong bảng dưới đây   |  |  | | --- | --- | | **Câu lệnh hoặc lựa chọn** | **Mô tả** | | INSERT | Chèn thêm một (các) hàng vào trong một bảng | | SELECT | Lựa chọn các thuộc tính từ các hàng trong một hoặc nhiều bảng hoặc khung nhìn | | WHERE | Hạn chế việc lựa chọn các hàng dựa trên một biểu thức điều kiện | | GROUP BY | Gộp nhóm các hàng đã được chọn ra dựa trên một hoặc nhiều thuộc tính | | HAVING | Hạn chế sự lựa chọn các hàng để gộp nhóm dựa trên một điều kiện | | ORDER BY | Xếp thứ tự các hàng được chọn | | UPDATE | Sửa đổi giá trị thuộc tính của một hoặc nhiều hàng của một bảng | | DELETE | Xoá một hoặc nhiều hàng từ một bảng | | COMMIT | Lưu trữ vĩnh viễn những thay đổi về dữ liệu | | ROLLBACK | Phục hồi dữ liệu về những giá trị ban đầu của chúng | | *Các phép toán so sánh* |  | | =, <, >, <=, >=, <> | Được sử dụng trong các biểu thức điều kiện | | *Các phép toán logic* |  | | AND, OR, NOT | Được sử dụng trong các biểu thức điều kiện | | Câu lệnh hoặc lựa chọn | Mô tả | | *Các phép toán đặc biệt* | *được sử dụng trong các biểu thức điều kiện* | | BETWEEN | Kiểm tra xem các giá trị của một thuộc tính có nằm trong một khoảng | | IS NULL | Kiểm tra xem giá trị của một thuộc tính có là trống không | | LIKE | Kiểm tra xem giá trị của một thuộc tính có giống với một kiểu chuỗi ký tự cho truớck | | IN | Kiểm tra xem giá trị của một thuộc tính có  nằm trong một danh sách các giá trị | | EXISTS | Kiểm tra xem một truy vấn con có trả về hàng dữ liệu nào không | | DISTINCT | Hạn chế các giá trị tới những giá trị duy nhất, hay loại bỏ những giá trị trùng lặp | | *Hàm tổng hợp (thống kê)* | *được sử dụng với SELECT để trả về những giá trị tổng hợp trên các cột* | | COUNT | Trả  về số lượng các hàng với các giá trị không rỗng cho một cột nào đó | | MIN | Trả về giá trị  nhỏ nhất của một thuộc tính được tìm thấy trong một cột nào đó | | MAX | Trả về giá trị  lớn nhất của một thuộc tính được tìm thấy trong một cột nào đó | | SUM | Trả về tổng của tất cả các giá trị của một cột nào đó | | AVG | Trả về giá trị trung bình của tất cả c ác giá trị của một cột nào đó | |  |  | |
| 3.2.3.1. Câu lệnh INSERT |
| Câu lệnh thêm bản ghi mới vào bảng: SQL dùng câu lệnh INSERT để thêm dữ liệu mới vào một bảng. Cú pháp của câu lệnh này như sau:   |  | | --- | |  | |  |   Ví dụ: Thêm hai bản ghi mới vào bảng VENDOR chúng ta cần thực hiện hai câu lệnh SQL dưới đây INSERT INTO VENDOR      VALUES (21225, ‘Bryson, Inc.’, ‘Smithson’, ‘615’, ‘223-3234’, ‘TN’, ‘Y’); INSERT INTO VENDOR      VALUES (21226, ‘SuperLoo, Inc.’, ‘Flushing’, ‘904’, ‘215-8995’, ‘FL’, ‘N’); Nếu một thuộc tính của một bản ghi không có giá trị (hay có giá trị là null) bạn sẽ sử dụng cú pháp sau đây để thêm một hàng vào bảng INSERT INTO PRODUCT      VALUES (‘23114-AA’,  ‘Sledge hammer, 12 lb.’, ’02-Jan-02’, 8, 5, 14.40, 0.05, NULL); Trong một số trường hợp, nhiều hơn một thuộc tính có thể nhận giá trị null hoặc không. Thay vì khai báo các thuộc tính là NULL trong lệnh INSERT, bạn có thể chỉ cần mô tả các thuộc tính cần có giá trị nhập, không cần quan tâm tới các thuộc tính nhận giá trị NULL. Việc này được thực hiện bằng cách liệt kê tên của các thuộc tính mà giá trị của chúng được đưa vào bên trong dấu ngoặc đơn ngay sau tên của bảng. Xét ví dụ dưới đây, giả sử rằng chỉ P\_CODE và P\_DESCRIPT cần nhập giá trị vào trong bảng PRODUCT. Hai cách nhập sau đây đều đúng: INSERT INTO PRODUCT VALUES (‘23114-AA’,  ‘Sledge hammer, 12 lb.’, NULL, NULL, NULL, NULL, NULL, NULL); hoặc INSERT INTO PRODUCT(P\_CODE, P\_DESCRIPT)      VALUES(‘23114-AA’, ‘Sledge hammer, 12 lb.’); |
| 3.2.3.2. Câu lệnh DELETE |
| Dễ dàng sử dụng SQL để xoá một hàng từ một bảng, thông qua câu lệnh DELETE. Cú pháp của câu lệnh này như sau:  DELETE FROM *tablename*         [WHERE *conditionlist* ];  Để xoá một hàng từ một bảng dựa trên giá trị của khoá chính, bạn có thể sử dụng câu lệnh như sau:  DELETE FROM PRODUCT WHERE P\_CODE = ‘23114-AA’;  Lệnh DELETE cũng có thể xoá bỏ nhiều hàng từ một bảng. Ví dụ bạn muốn xoá các sản phẩm từ bảng PRODUCT mà giá trị P\_MIN =5. Để thực hiện điều này bạn có thể dùng câu lệnh:  DELETE FROM PRODUCT WHERE P\_MIN = 5;  ***Lưu ý: Câu lệnh DELETE là một câu lệnh hướng tập hơp, nghĩa là điều kiện ở câu lệnh WHERE là có thể có hoặc không, nếu điều kiện đó không được chỉ rõ thì tất cả các hàng của bảng sẽ bị xoá!*** |
| 3.2.3.3. Câu lệnh UPDATE |
| Để thay đổi dữ liệu trong một bảng, câu lệnh UPDATE được sử dụng. Cú pháp của câu lệnh này như sau:  UPDATE  *tablename*           SET *columnname = expression* [, *columnname = expression* ] [ WHERE *conditionlist* ];  ***Lưu ý rằng điều kiện WHERE là có thể có hoặc không trong câu lệnh UPDATE. Nếu điều kiện không được xác định, thì câu lệnh UPDATE sẽ thực hiện trên tất cả các hàng của bảng đó.***  Ví dụ bạn muốn thay đổi P\_INDATE từ 13/12/2003 thành 18/1/2004 trong hàng thứ hai của bảng PRODUCT. Bạn cần sử dụng giá trị của khoá chính 13-Q2/P2 để xác định đúng hàng trong bảng cần thay đổi, câu lệnh tương ứng như sau  UPDATE PRODUCT         SET P\_INDATE = ’18-Jan-2004’  WHERE P\_CODE = ’13-Q2/P2’;  Nếu nhiều hơn một thuộc tính cần được thay đổi giá trị trong một hàng, các câu lệnh UPDATE sẽ được phân cách nhau bởi dấu phảy  UPDATE PRODUCT         SET P\_INDATE = ’18-JAN-2004’, P\_PRICE = 16.99, P\_MIN = 10 WHERE P\_CODE = ’13-Q2/P2’; |
| 3.2.3.4. Khẳng định, hủy bỏ những thay đổi dữ liệu trong bảng |
| Mọi thay đổi được thực hiện tới nội dung của bảng sẽ không được lưu trữ một cách vật lý trong một bảng vật lý (là một tệp trong hệ thống) cho tới khi một câu lệnh COMMIT được thực thi. Phụ thuộc vào sự phức tạp của hệ thống bạn đang làm việc, nếu năng lượng điện bị ngắt trong quá trình cập nhật một bảng (hoặc một cơ sở dữ liệu nói chung), trước khi câu lệnh COMMIT được thực thi, tất cả các thay đổi bạn thực hiện trước đó sẽ bị mất. Những hệ thống phức tạp hơn sẽ có khả năng phục hồi dữ liệu sau những sự cố như vật nhưng các hệ thống máy tính cá nhân thì tốt nhất bạn nên sử dụng bộ lưu điện UPS để hạn chế sự cố này. Cú pháp của câu lệnh COMMIT như sau  COMMIT [ *tablename* ]; // lưu tất cả các thay đổi cho một bảng hoặc COMMIT;// lưu tất cả những thay đổi được thực hiện trong tất cả các bảng  Nếu bạn chưa sử dụng câu lệnh COMMOT để lưu trữ vĩnh viễn những thay đổi trong cơ sở dữ liệu, bạn có thể phục hồi cơ sở dữ liệu về trạng thái trước đó (kết quả của lần cuối thực hiện câu lệnh COMMIT) bằng việc sử dụng câu lệnh ROLLBACK. Câu lệnh ROLLBACK phục hồi lại những thay đổi được thực hiện và trả lại dữ liệu những giá trị cũ của nó trước khi những thay đổi này được thực hiện. Cú pháp của câu lệnh như sau:  ROLLBACK;  Lưu ý là MS Access không hỗ trợ câu lệnh ROLLBACK. Một số hệ quản trị cơ sở dữ liệu như Oracle tự động COMMIT những thay đổi dữ liệu khi thực hiện các câu lệnh DDL vì vậy ROLLBACK sẽ không làm được gì đối với những hệ thống loại này. ROLLBACK sẽ phục hồi tất cả những thay đổi kể từ lần hệ thống COMMIT cuối cùng, điều này có nghĩa là thậm chí những thay đổi mà bạn không muốn phục hồi cũng sẽ được khôi phục lại với giá trị cũ nếu không một câu lệnh COMMIT nào được thực thi |
| 3.2.3.5. Tóm tắt các câu lệnh DML không truy vấn của SQL |
| Nhìn hình vẽ một biểu mẫu nhập dữ liệu dưới đây, mỗi mục vào trong SQL cũng tương đối phức tạp. Các ứng dụng của người dùng cuối được tạo ra tốt nhất với các tiện ích mà tạo ra các màn hình nhập liệu dễ dàng và hấp dẫn. Bạn sẽ thấy MS Access xử lý việc nhập liệu tốt hơn nhiều so với SQL thuần tuý.   |  | | --- | |  | |  | |
| 3.2.3.6. Câu lệnh truy vấn của DML của SQL |
| Câu lệnh truy vấn của DML bao gồm duy nhất một câu lệnh đơn được gọi là lệnh SELECT. Cú pháp của câu lệnh như sau:  SELECT [ ALL | DISTINCT] *columnlist*     FROM *tablelist* [ WHERE *condition* ] [GROUP BY  *columnlist* ] [HAVING *condition* ] [ORDER BY *columnlist* ];  Chúng ta sẽ tìm hiều hầu hết các đặc tính của câu lệnh SELECT, bắt đầu từ các câu truy vấn đơn giản tới những câu truy vấn phức tạp hơn, vẫn sử dụng cùng một cơ sở dữ liệu mà chúng ta đang xây dựng để minh hoạ. Câu lệnh lựa chọn đơn giản trong SQL: đây có lẽ là câu lệnh đơn giản nhất có thể, lấy ra các hàng từ một bảng cụ thể nào đó. Ví dụ, bạn muốn liệt kê tất cả các thuộc tính của mọi bản ghi của bảng PRODUCT. Nói cách khác, để xem bảng này thì dùng câu lệnh sau:  SELECT        P\_CODE, P\_DESCRIPT, P\_INDATE, P\_ONHAND, P\_MIN, P\_PRICE, P\_DISCOUNT, V\_CODE    FROM PRODUCT;  Hoặc  SELECT \*          FROM PRODUCT;  Ở đây dùng kí tự thay thế \* để thể hiện tất cả các thuộc tính của một bảng  ***Sử dụng một câu truy vấn con SELECT để thêm các hàng vào một bảng***  Mặc dù câu lệnh INSERT là một thao tác dữ liệu không truy vấn, nhưng nó cũng có thể bao gồm một câu truy vấn. Ví dụ sau sẽ thể hiện điều đó trước khi chúng ta chuyển sang các biểu thức truy vấn phức tạp hơn. SQL cho phép bạn có thể thêm các hàng của một bảng từ dữ liệu lấy từ một bảng khác. Cú pháp thực hiện loại câu lệnh INSERT này như sau:  INSERT INTO *tablename*           SELECT *columnlist*              FROM  *tablename*;  Ở đây câu lệnh SELECT là câu lệnh truy vấn con  (hoặc truy vấn lồng nhau). Với loại câu lệnh này thì các câu truy vấn bên trong được thực hiện trước bởi hệ quản trị cơ sở dữ liệu, kết quả của nó sẽ được dùng như đầu vào cho câu lệnh bên ngoài sát với câu lệnh đó (ở loại câu lệnh đang xét thì câu lệnh SELECT là loại bên trong, và câu lệnh INSERT là câu lệnh ngoài). Các giá trị được trả về bởi câu lệnh bên trong phải phù hợp với các thuộc tính và kiểu dữ liệu của bảng trong câu lệnh INSERT.  ***Câu lệnh truy vấn lựa chọn với sự hạn chế của điều kiện***  Bạn có thể chọn một phần nội dung của bảng bằng cách thiết lập giới hạn trên các bản ghi kết quả. Điều này được tiến hành với việc dùng mệnh đề WHERE cú pháp như sau  SELECT *columnlist*    FROM *tablelist* WHERE *conditionlist* ;  Lệnh SELECT sẽ lấy tất cả các hàng thoả mãn điều kiện được thể hiện ở mệnh đề WHERE. Ví dụ:  SELECT P\_DESCRIPT, P\_INDATE, P\_PRICE, V\_CODE     FROM PRODUCT WHERE V\_CODE = 21344;  Cấu trúc của câu lệnh SQL cung cấp hầu hết tính linh hoạt trong truy vấn không hạn chế. Các giới hạn điều kiện số có thể được dùng cho nội dung các bảng được lựa chọn. Trừ việc kiểm tra giá trị các thuộc tính là NULL, SQL không trả về những hàng mà giá trị thuộc tính được lựa chọn là NULL trong kết quả. Xem ví dụ trong truy vấn dưới đây  SELECT  P\_DESCRIPT, P\_INDATE, P\_PRICE, V\_CODE     FROM PRODUCT  WHERE V\_CODE <> 21344;  Bảng PRODUCT được thể hiện trong hình vẽ dưới đây chứa kết quả của truy vấn trên, để ý là hàng thứ 10 và 13 trong bảng PRODUCT không xuất hiện trong các kết quả của truy vấn này   |  | | --- | |  | |  |   Kết quả của truy vấn thể hiện trong hình vẽ dưới đây   |  | | --- | |  | |  |   Các thủ tục về ngày tháng thường cụ thể cho từng phần mềm hơn các thủ tục SQL khác. Ví dụ, truy vấn để liệt kê tất cả các hàng trong đó ngày lưu kho từ 20/1/2004, sẽ như sau:  SELECT  P\_DESCRIPT, P\_ONHAND, P\_MIN, P\_PRICE, P\_INDATE FROM PRODUCT WHERE P\_INDATE >= “20-Jan-2004’;  Lưu ý trong Access ký tự phân biệt dùng cho ngày tháng là # vì vậy câu truy vấn trên trong Access sẽ như sau  SELECT  P\_DESCRIPT, P\_ONHAND, P\_MIN, P\_PRICE, P\_INDATE     FROM PRODUCT  WHERE P\_INDATE >= #20-Jan-2004#;  ***Sử dụng các cột tính toán và đổi tên các cột***  Giả sử rằng câu truy vấn của bạn cần xác định một giá trị mà không được lưu trữ vật lý trong cơ sở dữ liệu nhưng lại được tính toán từ dữ liệu trong cơ sở dữ liệu. Ví dụ bạn muốn tính tổng giá trị của các mặt hàng hiện tại có trong khi. Về logic, tổng số tiền này bằng tích của số lượng mỗi mặt hàng hiện đang có nhân với giá bán hiện tại của mặt hàng đó. Câu truy vấn SQL cho yêu cầu này được thể hiện như sau  SELECT  P\_DESCRIPT, P\_ONHAND,  P\_PRICE, P\_ONHAND \* P\_PRICE   AS TOTVALUE     FROM PRODUCT  SQL sẽ chấp nhận mọi biểu thức hợp lệ (như P\_ONHAND \* P\_PRICE ở mệnh đề SELECT trong câu truy vấn trên) trong các cột tính toán, sử dụng các thuộc tính của các bảng được liệt kê trong mệnh đề FROM. Chú ý Acces sẽ tự động thêm một nhãn Expr tới tất cả các cột tính toán. Oracle sử dụng biểu thức thực tế làm nhãn của các cột tính toán này. SQL chuẩn cho phép sử dụng biệt danh (alias-một tên khác) cho bất kể các cột nào trong mệnh đề SELECT. Biệt danh cho mỗi cột được bắt đầu bởi từ khoá AS. Kết quả của câu lệnh SELECT trên được thể hiện trong hình vẽ dưới đây trong đó cột tính toán được biểu diễn bởi biệt danh của nó.   |  | | --- | |  | |  |   Giả sử chúng ta muốn liệt kê một danh sách các sản phẩm không còn bảo hành nữa. Trong trường hợp này, giả thiết rằng những sản phẩm không còn bảo hành là những sản phẩm được nhập kho quá 90 ngày. Vì thế, P\_INDATE phải nhỏ hơn ngày hiện tại ít nhất là 90 ngày. Phiên bản Access của câu truy vấn này được thể hiện dưới đây, tiếp đó là phiên bản của Oracle với kết quả được thể hiện ở màn hình sau đó.  Access Version  SELECT P\_CODE, P\_INDATE, DATE() – 90 AS CUTDATE   FROM PRODUCT  WHERE P\_INDATE <= DATE() – 90;  Oracle Version  SELECT  P\_CODE, P\_INDATE, SYSDATE – 90 AS CUTDATE    FROM PRODUCT  WHERE P\_INDATE <= SYSDATE – 90;   |  | | --- | |  | |  |   Trong thực tế, việc tìm kiếm dữ liệu thường liên quan tới nhiều điều kiện. SQL cho phép bạn có thể thể hiện nhiều điều kiện trong một câu truy vấn đơn bằng cách dùng các phép toán logic. Các phép toán logic mà SQL hỗ trợ là AND, OR và NOT.  Ví dụ: giả sử bạn muốn liệt kê một danh sách các sản phẩm từ bảng PRODUCT mà V\_CODE=21344 hoặc =24288. Cây truy vấn như sau  SELECT  P\_DESCRIPT, P\_INDATE, P\_PRICE, V\_CODE   FROM PRODUCT  WHERE  V\_CODE = 21344  OR  V\_CODE = 24288; |
| 3.2.3.7. Các toán tử đặc biệt trong SQL |
| SQL theo chuẩn ANSI cho phép sử dụng các toán tử đặc biệt trong biểu thức nằm trong mệnh đề WHERE. Các toán tử này bao gồm: BETWEEN- được dùng để kiểm tra xem giá trị một thuộc tính có nằm trong một khoảng nào đó không IS NULL- được dùng để xác định liệu một thuộc tính có nhận giá trị NULL không, lưu ý biểu thức X=NULL luôn có giá trị bằng  **false**. LIKE- được dùng để gắn giá trị một thuộc tính với một kiểu chuỗi ký tự. Nhiều ký tự thay thế được sẵn có để sử dụng với toán tử này. IN- được sử dụng để xác định liệu giá trị một thuộc tính có nằm trong một danh sách giá trị không EXISTS- được dùng để xác định liệu một truy vấn con có trả về một tập rỗng hay không |
| 3.2.3.8. Các bảng ảo - tạo các khung nhìn |
| Nhắc lại rằng kết quả của một phép toán quan hệ (kiểu như SELECT trong SQL) là các quan hệ khác (hoặc bảng khác). Trong bài này chúng ta tiếp tục sử dụng cơ sở dữ liệu xuyên suốt bài giảng từ trước để thể hiện các ví dụ minh hoạ cho ngôn ngữ SQL. Giả sử rằng cuối ngày làm việc, chúng ta muốn có danh sách của tất cả các mặt hàng được đặt hàng lại, tức là tất cả các mặt hàng có số lượng sẵn có trong kho nhỏ hơn một giá trị ngưỡng nào đó (số lượng nhỏ nhất để phân phối mặt hàng này). Thay vì cứ gõ cùng một câu truy vấn hàng ngày, chúng ta nên lưu trữ câu truy vấn này dài hạn trong cơ sở dữ liệu để dùng đi dùng lại nhiều lần.  Khung nhìn quan hệ là một chức năng để có thể làm được việc này. Trong SQL, một khung nhìn là một bảng dựa trên một câu truy vấn SELECT. Câu truy vấn này có thể chứa các cột, các cột là kết quả tính toán, các cột biệt danh và các hàm tổng hợp từ một hoặc nhiều bảng. Các bảng mà dựa trên đó một khung nhìn được thiết lập được gọi là bảng cơ sở (base table). Các khung nhìn được tạo ra trong SQL sử dụng lệnh CREATE VIEW với cú pháp như sau:  CREATE VIEW  *viewname* AS SELECT *query*  Câu lệnh CREATE VIEW là một lệnh DDL nhưng chứa một câu truy vấn con được sử dụng để tạo ra một bảng ảo trong từ điển dữ liệu. Ví dụ:  CREATE VIEW PRODUCT\_3 AS      SELECT P\_DESCRIPT, P\_ONHAND, P\_PRICE         FROM PRODUCT       WHERE P\_PRICE > 50.00;  Lưu ý: Câu lệnh CREATE VIEW không được hỗ trợ trưc tiếp trong Access. Để tạo một khung nhìn trong Access, bạn cần tạo một truy vấn SQL và sau đó lưu trữ nó. Một khung nhìn quan hệ có một số các đặc tính đặc biệt sau:   1. Bạn có thể sử dụng tên của một khung nhìn ở bất kể vị trí nào mà một tên bảng có thể được đặt vào trong câu lệnh SQL 2. Các khung nhìn được cập nhật động. Có nghĩa là khung nhìn được tạo lại mỗi khi cần nó liên quan tới công việc đang thực hiện 3. Các khung nhìn cung cấp một mức độ bảo mật trong cơ sở dữ liệu bởi vì khung nhìn có thể hạn chế người dùng chỉ truy nhập tới các cột cụ thể và các hàng cụ thể trong một bảng 4. Các khung nhìn cũng có thể được sử dụng làm nền tảng cho các báo cáo. Khung nhìn được tạo ra sau đây thể hiện một bảng tóm tắt về tổng cho phí các sản phẩm và thống kê số lượng sản phẩm có trong kho được gộp nhóm theo từng nhà cung cấp   CREATE VIEW SUMPRDXVEN AS     SELECT V\_CODE, SUM(P\_ONHAND\*P\_PRICE) AS TOTCOST,            MAX(P\_ONHAND) AS MAXQTY, MIN(P\_OHAND) AS MINQTY,  AVG(P\_ONHAND) AS AVGQTY       FROM PRODUCT      GROUP BY V\_CODE; |
| 3.2.3.9. Kết nối các bảng trong cơ sở dữ liệu |
| Khả năng kết nối các bảng thông qua các thuộc tính chung có lẽ là một đặc tính nổi bật quan trọng nhất của cơ sở dữ liệu quan hệ so với các loại cơ sở dữ liệu khác. Trong SQL, một phép kết nối được thực hiện khi dữ liệu được lấy ra từ nhiều hơn một bảng tại một thời điểm. Để kết nối các bảng, đơn giản bạn chỉ cần liệt kê các bảng trong mệnh đề FROM của câu lệnh SELECT. Hệ quản trị cơ sở dữ liệu sẽ tạo ra tích Đề các của mọi bảng được liệt kê trong mệnh đề FROM. Để thực hiện phép kết nối tự nhiên, bạn phải chỉ ra cụ thể sự liên kết trên các thuộc tính chung trong mệnh đề WHERE. Sự liên kết này được gọi là **điều kiện kết nối.**  Điều kiện kết nối thường được cấu thành bởi một phép so sánh bằng giữa khoá ngoài và khoá chính của các bảng liên quan. Giả sử rằng chúng ta muốn kết nối hai bản VENDOR và PRODUCT. V\_CODE là khoá ngoài trong bảng PRODUCT và là khoá chính trong bảng VENDOR, điều kiên kết nối sẽ được thể hiện qua thuộc tính này trong câu lệnh sau  SELECT  PRODUCT.P\_DESCRIPT, PRODUCT.P\_PRICE, VENDOR.V\_NAME, VENDOR.V\_CONTACT, VENDOR.V\_AREACODE, VENDOR.V\_PHONE    FROM  PRODUCT, VENDOR   WHERE  PRODUCT.V\_CODE = VENDOR.V\_CODE;  Nếu bạn không xác định cụ thể một điều kiện kết nói tại mệnh đề WHERE thì kết quả của câu lệnh sẽ là một tích đề các. Sử dụng cùng một cơ sở dữ liệu, bảng PRODUCT bao gồm 16 bộ (hàng) và bảng VENDOR bao gồm 11 bộ, sẽ cho kết quả một tích đề các gồm 16 x 11 = 176 bộ. Hầu hết các bộ này (xem kết quả trong hình vẽ dưới đây) đều không sử dụng đến (không phục vụ cho mục đích câu truy vấn).  Khi kết nối ba hoặc nhiều bảng, bạn cần xác định một điều kiện kết nối cho mỗi cặp bảng. Số lượng các điều kiện kết nối sẽ luôn là N-1 trong đó N là số lượng các bảng được liệt kê trong mệnh đề FROM. Nên cẩn thận để tránh tạo các điều kiện kết nối vòn tròn. Ví dụ nếu bảng A kết nối với bảng B, bảng B kết nối với bảng C, và bảng C cũng có liên kết với bảng A, lúc đó chỉ tạo hai điều kiện kết nối: A với B và B với C. Không tạo kết nối C vớiA !  ***Kết nối đệ quy: là kết nối một bảng với chính bảng đó.***  Một biệt danh có thể được sử dụng để xác định bảng nguồn từ đó dữ liệu được lấy ra cho câu truy vấn. Ví dụ:  SELECT  P\_DESCRIPT, P\_PRICE, V\_NAME, V\_CONTACT, V\_AREACODE, V\_PHONE   FROM  PRODUCT P, VENDOR V   WHERE P.V\_CODE = V.V\_CODE   ORDER BY P\_PRICE;  Trong câu truy vấn trên một biệt danh được sử dụng cho bảng PRODUCT và bảng VENDOR. Lưu ý trong Access cần phải thêm từ khoá AS trước biệt danh. Một biệt danh đặc biệt rất hữu ích khi mọt bảng phải kết nối với chính nó, cái được gọi là kết nối đệ quy. Ví dụ, sử dụng bảng EMPLOYEE chúng ta muốn tạo ra danh sách tất cả các nhân viên cùng với tên của người quản lý của họ. Nếu không sử dụng một biệt danh cho bảng thì truy vấn này là không thể, bởi vì tên của các thuộc tính cần thiết là không duy nhất để lấy ra, sẽ gây nhầm lẫn cho hệ quản trị cơ sở dữ liệu. Sử dụng biệt danh, câu truy vấn tương ứng trong Access được thể hiện ở hình vẽ dưới đây và kết quả của truy vấn được thể hiện trong hình vẽ sau:   |  | | --- | |  | |  | |  | |  | |
| 3.2.3.10. Kết nối ngoài |
| Kết quả câu truy vấn cho kết nối tự nhiên trình bày ở trên giữa hai bảng PRODUCT và VENDOR có 14 dòng được liệt kê. Nếu bạn so sánh kết quả này với bản thân các bản ghi của bảng PRODUCT sẽ thấy thiếu mất hai sản phẩm. Lý do là hai sản phẩm bị thiếu đó có các giá trị rỗng ở thuộc tính V\_CODE trong bảng PRODUCT. Bởi vì không có giá trị nào trong bảng VENDOR có thể kết nối với giá trị NULL trong PRODUCT nên chúng không xuất hiện trong kết quả cuối cùng của phép kết nối. Để đưa hai bản ghi thiếu trong trường hợp trên vào kết quả cuối cùng của phép kết nối, chúng ta cần sử dụng một phép kết nối ngoài. Nhắc lại rằng có ba loại kết nối ngoài, kết nối bên trái, bên phải và kết nối đầy đủ cả hai bên. Cho trước hai bảng A và B, A kết nối ngoài bên trái với B cho kết quả là tất cả những bản ghi thoả mãn điều kiện kết nối cộng thêm tất cả các bản ghi không thoả mãn của A. A kết nối ngoài bên phải với B cho kết quả là tất cả các bản ghi thoả mãn điều kiện kết nối cộng thêm tất cả những bản ghi không thoả mãn trong B. Chúng ta sẽ xem xét phép kết nối ngoài đầy đủ sau. Để thêm bộ có giá trị NULL của V\_CODE từ bảng PRODUCT vào kết quả cuối cùng, chúng ta sẽ sử dụng câu truy vấn sau đây   |  | | --- | |  | |  |   Lưu ý là từ kết nối ngoài không xuất hiện trong câu truy vấn, nó được ngầm hiểu khi thể hiện từ khoá LEFT là kết nối ngoài bên trái, hay RIGHT là kết nối ngoài bên phải. Kết quả của câu lệnh được thể hiện trong hình vẽ dưới đây: đưa ra tất cả các hàng trong bảng VENDOR thoả mãn điều kiện kết nối trái với bảng PRODUCT.   |  | | --- | |  | |  |   Bảng VENDOR có một số bản ghi mà V-CODE không phù hợp với bất kể bản ghi nào xuất hiện trong bảng PRODUCT. Nếu thực hiện kết nối ngoài bên phải thể hiện trong hình sau   |  | | --- | |  | |  |   Kết quả của truy vấn được thể hiện trong hình vẽ dưới đây trong đó đưa ra tất cả những hàng của PRODUCT phù hợp và không phù hợp với VENDOR   |  | | --- | |  | |  | |
| 3.2.3.11. Các toán tử tập hợp quan hệ |
| Nhắc lại rằng đại số quan hệ có định hướng tập hợp và bao gồm rất nhiêud các toán tử như toán tử  hợp, giao, trừ và các thuật ngữ tập hợp, bảng và quan hệ có thể dùng lẫn với nhau trong phạm vi bài giảng về cơ sở dữ liệu quan hệ. Như với đại số quan hệ thuần túy, các toán tử tập hợp chỉ làm việc với những quan hệ khả hợp. Trong SQL, điều này có nghĩa là tên của các thuộc tính phải giống nhau và kiểu dữ liệu của chúng cũng phải như nhau. Đây là một lĩnh vực mà các hệ quản trị cơ sở dữ liệu quan hệ khác nhau thì rất khác nhau về định nghĩa khái niệm khả hợp. Ví dụ, một số các hệ quản trị cơ sở dữ liệu quan hệ sẽ coi kiểu dữ liệu VARCHAR(35) và VARCHAR(15) là khả hợp bởi vì mặc dù chúng có độ dài khác nhau nhưng đều dựa trên một kiểu dữ liệu cơ bản. Các hệ quản trị khác sẽ không coi hai kiểu dữ liệu này là khả hợp. Bạn có thể thử với hệ quản trị của bạn đang tiến hành bài tập lớn để xem cái gì được gọi là khả hợp và cái gì là không khả hợp. |
| 3.2.3.11.1. Toán tử hợp |
| Xem xét ngữ cảnh sau đây. Giả sử công ty của chúng ta đã mua lại một công ty khác và người quản lý muốn chắc chắn rằng danh sách khách hàng của công ty được trộn với danh sách khách hàng hiện tại của công ty. Vì có thể một vài khách hàng có thể mua hàng từ cả hai công ty nên hai danh sách khách hàng có thể có những khách hàng chung. Người quản lý không muốn có bất kỳ khách hàng nào bị trùng lặp trong danh sách. Câu lệnh UNION của SQL sẽ tự động loại bỏ những bản ghi trùng lặp từ hai quan hệ toán hạng. nếu bạn muốn bao gồm cả những bản ghi trùng lặp thì dùng câu lệnh UNION ALL. Cú pháp của câu lệnh truy vấn UNION là *query*  UNION  *query.* Về cơ bản câu lệnh UNION kết hợp hai đầu ra của hai câu lệnh truy vấn SELECT. Nhớ rằng đầu ra của hai câu truy vấn này phải khả hợp. Để mô tả truy vấn UNION, hãy kết hợp danh sách các khách hàng hiện tại vơi danh sách khách hàng mới như sau:   |  | | --- | |  | |  | |  | |  |   Kết quả hợp hai bảng như sau:   |  | | --- | |  | |  |   Trong kết quả này khách hàng có tên là Dunne và Olowski xuất hiện trong cả hai bảng vì vậy chỉ xuất hiện một lần trong bảng kết quả. Còn nếu sử dụng UNION ALL thì hai khách hàng đó sẽ xuất hiện hai lần trong bảng kết quả. |
| 3.2.3.11.2. Toán tử giao |
| Toán tử giao có cú pháp như sau  *query*  INTERESCT  *query*. Access không hỗ trợ toán tử giao. Để thực hiện phép toán này trong Access bạn cần sử dụng toán tử IN như sau   |  | | --- | |  | |  |   Một ví dụ về phép giao hai bảng CUSTOMER và INVOICE   |  | | --- | |  | |  | |
| 3.2.3.11.3. Toán tử trừ |
| Toán tử trừ có cú pháp  *query*  MINUS  *query.* Access cũng không hỗ trợ phép toán này. Để thực hiện nó, bạn cần sử dụng toán tử NOT IN. Hầu hết các hệ quản trị đều để tên phép toán này là EXCEPT.   |  | | --- | |  | |  |   Một ví dụ về phép trừ được thể hiện trong hình vẽ dưới đây   |  | | --- | |  | |  | |
| 3.2.3.11.4. Các phép kết nối của SQL |
| Các phép toán kết nối của SQL trộn các hàng từ hai bảng và trả về các hàng mà   1. Có các giá trị giống nhau cho các cột chung (kết nối tự nhiên) hoặc 2. Thoả mãn một điều kiện kết nối nào đó (bằng nhau hoặc không bằng nhau) hoặc 3. Có các giá trị giống nhau tại các cột chung hoặc có cả các giá trị không bằng nhau   Chúng ta đã xem xét các dạng cơ bản của một kết nối SQL trong đó có hai bảng được liệt kê tại mệnh đề FROM và mệnh đề WHERE thể hiện điều kiện kết nối. Một ví dụ cho dạng cơ bản của phép kết nối này được thể hiện sau đây  SELECT P\_CODE, P\_DESCRIPT, P\_PRICE, V\_NAME    FROM PRODUCT, VENDOR  WHERE PRODUCT.V\_CODE = VENDOR.V\_CODE Mệnh đề FROM thể hiện các bảng được kết nối. Nếu ba hoặc nhiều hơn 3 bảng được kết nối thì các phép kết nối sẽ thực hiện hai bảng tại một thời điểm, bắt đầu từ trái sang phải. Điều kiện kết nối được thể hiện trong mệnh đề WHERE, với ví dụ trên một kết nối tự nhiên được thể hiện trên thuộc tính V\_CODE. Cú pháp của phép kết nối được thể hiện ở trên đôi khi được cho là một phép kết nối được viết theo kiểu cổ điển. Bảng tóm tắt các phép toán kết nối được thể hiện dưới đây:   |  |  |  |  | | --- | --- | --- | --- | | **Phân loại kết nối** | **Kiểu kết nối** | **Ví dụ về cú pháp SQL** | **Mô tả** | | Cross | CROSS JOIN | SELECT \* | Kiểu cổ điển. Trả về tích Đề các của T1 và T2 | | FROM T1, T2; | |  |  | SELECT \* | Kiểu mới. Trả về tích Đề các của T1 và T2 | | FROM T1 CROSS JOIN T2; | | Inner | Old Style | SELECT \* | Trả về các hàng thoả mãn điều kiện kết nối trong mệnh đề WHERE-kiểu cổ điển. Chỉ những  hàng có giá trị  thích hợp mới được chọn. | | JOIN | FROM T1, T2 | |  | WHERE T1.C1 = T2.C1 | |  | NATURAL JOIN | SELECT \* | Trả về các hàng có giá trị thích hợp tại các cột giống nhau. Các cột tương  thích nhau phải có cùng tên và cùng kiểu dữ liệu. | | FROM T1 NATURAL JOIN T2 | |  | JOIN USING | SELECT \* | Chỉ trả về các hàng có giá trị thích hợp tại ccs cột được thể hiện trong mệnh đề USING. | | FROM T1 JOIN T2 USING (C1) | |  | JOIN ON | SELECT \* | Chỉ trả về các hàng thoả mãn điều kiện kết nối được thể hiện trong mệnh đề ON. | | FROM T1 JOIN T2 | | ON T1.C1 = T2.C1 | | Outer | LEFT JOIN | SELECT \* | Trả về các hàng có giá trị thích hợp và bao gồm cả những hàng không thích hợp trong bảng bên trái (T1). | | FROM T1 LEFT OUTER JOIN T2 | | ON T1.C1= T2.C1 | |  | RIGHT JOIN | SELECT \* | Trả về các hàng có giá trị thích hợp và bao gồm cả những hàng không thích hợp trong bảng bên  phải (T2). | | FROM T1 RIGHT OUTER JOIN T2 | | ON T1.C1= T2.C1 | |  | FULL JOIN | SELECT \* | Trả về các hàng có giá trị thích hợp và bao gồm cả những hàng không thích hợp trong cả hai bảng bên  phải và bên trái | | FROM T1 FULL OUTER JOIN T2 | | ON T1.C1= T2.C1 |   Phép CROSS JOIN (kết nối chéo) trong SQL tương đương với một tích đề các trong đại số quan hệ chuẩn. Cú pháp của phép kết nối này như sau:   |  |  | | --- | --- | | **Theo kiểu cổ điển** | **Theo kiểu mới** | | SELECT *column-list* | SELECT *column-list* | | FROM *table1, table2*; | FROM *table1 CROSS JOIN table2*; |   Cú pháp của phép kết nối tự nhiên như sau   |  |  | | --- | --- | | **Theo kiểu cổ điển** | **Theo kiểu mới** | | SELECT *column-list* | SELECT *column-list* | | FROM *table1,  table2*  *WHERE table1.C1 = table2.C2;* | FROM *table1* NATURAL JOIN*table2*; |   Phép kết nối tự nhiên sẽ thực hiện các nhiệm vụ sau đây:   * Xác định các thuộc tính chung bằng cách tìm kiếm các thuộc tính có cùng tên và có kiểu  dữ liệu tương thích. * Chỉ chọn những hàng có chung giá trị tại các thuộc tính chung. * Nếu không có các thuộc tính chung, trả về kết nối chéo của hai bảng   Một sự khác nhau quan trọng giữa kết nối tự nhiên và cú pháp cổ điển là kết nối tự nhiên không đòi hỏi việc sử dụng định danh bảng trong các thuộc tính chung.Cách  thứ hai để thể hiện một kết nối là thông qua từ khoá USING. Câu truy vấn này sẽ trả về các hàng có giá trị tương ứng bằng nhau tại các cột được chỉ ra trong mệnh đề USINH. Các cột được liệt kê trong mệnh đề USING phải xuất hiện trong cả hai bảng. Cú pháp của câu lệnh này như sau: SELECT *column-list*   FROM *table1* JOIN *table2* USING *(common-column);*  Ví dụ:  SELECT  INV\_NUMBER, P\_CODE, P\_DESCRIPT, LINE\_UNITS, LINE\_PRICE   FROM  INVOICE JOIN LINE USING (INV\_NUMBER) JOIN PRODUCT USING (P\_CODE);  Cũng như đối với câu lệnh kết nối tự nhiên, câu lệnh này không đòi hỏi phải có định danh tên bảng cho các thuộc tính. Trên thực tế, Oracle 9i sẽ thông báo lỗi nếu bạn đưa tên bảng vào mệnh đề USING. Cả câu lệnh kết nối tự nhiên NATURAL JOIN và JOIN USING  đều sử dụng tên của các thuộc tính chung trong các bảng cần kết nối. Một cách khác để thể hiện một kết nối khi các bảng không có thuộc tính chung là sử dụng câu lệnh JOIN ON. Câu truy vấn này sẽ trả về những bản ghi thoả mãn điều kiện. Điều kiện kết nối sẽ thường bao gồm một biểu thức so sánh bằng của hai cột. Các cột có thể có chung tên hoặc không nhưng hiển nhiên là phải có kiểu dữ liệu tương thích nhau. Cú pháp của câu lệnh này như sau:  SELECT *column-list*   FROM *table1* JOIN *table2* ON *join-condition;*  Một ví dụ  SELECT INVOICE.INV\_NUMBER, P\_CODE, P\_DESCRIPT, LINE\_UNITS, LINE\_PRICE   FROM INVOICE JOIN LINE ON INVOICE.INV\_NUMBER = LINE.INV\_NUMBER    JOIN PRODUCT ON LINE.P\_CODE = PRODUCT.P\_CODE;  Chú ý trong ví dụ trên, không giống như NATURAL JOIN và JOIN USING, mệnh đề JOIN ON đòi hỏi sử dụng định danh bảng cho các thuộc tính chung. Nếu bạn không đưa định danh bảng vào bạn sẽ nhận được một thông báo lỗi “column ambiguously defined” (cột được xác định không rõ ràng).  Luôn  nhớ rằng cú pháp của JOIN ON cho phép bạn thực hiện một phép kết nối thậm chí khi các bảng không có tên thuộc tính chung. Ví dụ để tạo ra một danh sách tất cả các nhân viên cùng với tên của người quản lý bạn có thể sử dụng truy vấn đệ quy dùng mệnh đề JOIN ON như dưới đây:  SELECT E.EMP\_MGR, M.EMP\_LNAME, E.EMP\_NUM, E.EMP\_LNAME   FROM EMP E JOIN EMP M ON E.EMP\_MGR = M.EMP\_NUM  ORDER BY E.EMP\_MGR;  ***Các phép kết nối ngoài OUTER JOIN***  Chúng ta đã xem xét các dạng của kết nối ngoài bên trái và kết nối ngoài bên phải trong các bài giảng trước. Ngoài ra còn có phép kết nối ngoài đầy đủ trong SQL. Một phép kết nối ngoài đầy đủ sẽ trả về không chỉ các hàng thoả mãn điều kiện kết nối (đó là những bản ghi có các giá trị bằng nhau tại những cột chung) nhưng cũng gồm tất cả các hàng với giá trị không bằng nhau ở cả hai bảng bên trái và bên phải. Cú pháp của phép kết nối ngoài đầy đủ như sau SELECT *column-list*    FROM *table1* FULL [OUTER]JOIN *table2* ON *join-condition;* Ví dụ sau đây của phép kết nối ngoài đầy đủ sẽ liệt kê mã sản phẩm, mã người cung cấp và tên người cung cấp của tất cả các sản phẩm và bao gồm cả những hàng (sản phẩm không của các nhà cung cấp) và tất cả những hàng của nhà cung cấp (mà không có sản phẩm nào cả). SELECT  P\_CODE, VENDOR.V\_CODE, V\_NAME FROM VENDOR FULL OUTER JOIN PRODUCT ON VENDOR.V\_CODE = PRODUCT.V\_CODE |
| 3.2.3.12. Truy vấn con và truy vấn tương hỗ |
| Việc sử dụng các phép kết nối cho phép một hệ quản trị cơ sở dữ liệu có thể lấy thông tin từ hai hoặc nhiều bảng. Dữ liệu lấy từ các bảng được xử lý đồng thời. Trên thực tế, thông thường các truy vấn hay cần xử lý dữ liệu dựa trên các dữ liệu đã được xử lý trước. Ví dụ, giả sử bạn muốn tạo ra một danh sách các nhà cung cấp có cung cấp các sản phẩm. (Nhắc lại rằng không phải tất cả các nhà cung cấp trong bảng VENDOR đều cung cấp sản phẩm-một vài trong số đó chỉ là những nhà cung cấp trong tương lai). Câu truy vấn thực hiện yêu cầu này như sau:  SELECT  V\_CODE, V\_NAME   FROM VENDOR  WHERE V\_CODE NOT IN (SELECT V\_CODE    FROM PRODUCT);  Một truy vấn con là một truy vấn dạng câu lệnh SELECT bên trong một truy vấn khác. Nó thường được thể hiện bên trong  một dấu ngoặc. Câu truy vấn đầu tiên trong câu lệnh SQL được biết đến là câu truy vấn bên ngoài, câu truy vấn thứ hai là câu truy vấn bên trong. Câu truy vấn bên trong bao giờ cũng được thực thi đầu tiên, kết quả của câu truy vấn bên trong được sử dụng như đầu vào cho câu truy vấn bên ngoài. Toàn bộ câu lệnh SQL kiểu như vậy được biết đến là truy vấn lồng nhau. Câu truy vấn bên trong còn được gọi là câu truy vấn con. Một truy vấn con có thể trả về một trong các loại kết quả sau đây:   1. Một giá trị đơn (một cột hoặc một hàng). Câu truy vấn con này có thể được dùng ở bất kể nơi nào một giá trị đơn được sử dụng. Ví dụ, ngay ở bên phải của biểu thức so sánh. 2. Một danh sách các giá trị (một cột hoặc nhiều hàng). Kiểu truy vấn con này có thể được dùng ở bất kỳ nơi nào mà một danh sách các giá trị được sử dụng. Ví dụ khi dùng mệnh đề IN. 3. Một bảng ảo (nhiều cột, nhiều hàng gồm tập các giá trị). Câu truy vấn con loại này có thể được sử dụng ở bất kể nơi nào cần một bảng trong câu lệnh. Ví dụ trong câu lệnh FROM 4. Không có giá trị nào cả hay giá trị NULL. Trong những trường hợp kiểu này đầu ra của truy vấn bên ngoài có thể cho kết quả là một thông báo lỗi hoặc một tập rỗng, phụ thuộc vào vị trí của câu truy vấn đó được sử dụng (trong một phép so sánh, một biểu thức hoặc một tập các bảng)   Loại truy vấn con phổ biến nhất sử dụng một câu truy vấn con SELECT bên trong ở bên phải của một biểu thức so sánh ở mệnh đề WHERE. Ví dụ, để tìm thấy tất cả các sản phẩm có giá lớn hơn hoặc bằng giá trung bình của một sản phẩm, câu truy vấn sau đây là cần thiết  SELECT P\_CODE, P\_PRICE FROM PRODUCT  WHERE P\_PRICE >= (SELECT AVG(P\_PRICE)                 FROM PRODUCT);  Câu truy vấn con cũng có thể được sử dụng kết hợp với các phép kết nối. Câu truy vấn dưới đây liệt kê tất cả các khách hàng đặt hàng sản phẩm “Claw hammer”.  SELECT DISTINCT CUS\_CODE, CUS\_LNAME, CUYS\_FNAME    FROM CUSTOMER JOIN INVOICE USING (CUS\_CODE)                  JOIN LINE USING (INV\_NUMBER)                  JOIN PRODUCT USING (P\_CODE)   WHERE P\_CODE = (SELECT P\_CODE                     FROM PRODUCT                    WHERE P\_DESCRIPT = “Claw hammer”);  Lưu ý rằng câu truy vấn trên cũng có thể được viết như sau:  SELECT DISTINCT CUS\_CODE, CUS\_LNAME, CUYS\_FNAME    FROM CUSTOMER JOIN INVOICE USING (CUS\_CODE)                  JOIN LINE USING (INV\_NUMBER)                  JOIN PRODUCT USING (P\_CODE)   WHERE  P\_DESCRIPT = ‘Claw hammer’);  Tuy nhiên, điều gì sẽ xảy ra nếu hai hoặc nhiều hơn phần mô tả sản phẩm có chứa chuỗi “Claw hammer”. Bạn sẽ nhận được một thông báo lỗi bởi vì vế phải của biểu thức so sánh chỉ có thể nhận một giá trị. Để giải quyết vấn đề vừa nêu trên, chúng ta sử dụng toán hạng IN. Câu truy vấn dưới đây sẽ liệt kê tất cả các khách hàng đặt bất kể loại hàng nào là hammer hoặc saw  SELECT DISTINCT CUS\_CODE, CUS\_LNAME, CUYS\_FNAME     FROM CUSTOMER JOIN INVOICE USING (CUS\_CODE)                   JOIN LINE USING (INV\_NUMBER)                   JOIN PRODUCT USING (P\_CODE)  WHERE P\_CODE IN (SELECT P\_CODE                      FROM PRODUCT                    WHERE P\_DESCRIPT LIKE ‘%hammer%’                       OR P\_DESCRIPT LIKE ‘%saw%’);  Câu truy vấn con cũng có thể được sử dụng trong mệnh đề HAVING. Gợi nhớ rằng mệnh đề HAVING được sử dụng để hạn chế đầu ra của một truy vấn GROUP BY bằng cách áp dụng các tiêu chí điều kiện đối với các bản ghi đã được gộp nhóm. Ví dụ, truy vấn liệt kê tất cả các sản phẩm với tổng số lượng bán được lớn hơn lượng bán trung bình của các mặt hàng được thể hiện như dưới đây: SELECT DISTINCT P\_CODE, SUM(LINE\_UNITS)   FROM LINE  GROUP BY P\_CODE HAVING SUM(LINE\_UNITS) > (SELECT AVG(LINE\_UNITS)                              FROM LINE); Trong câu truy vấn con IN sử dụng một toán tử bằng, điều đó có nghĩa là nó chỉ chọn các hàng thích hợp với ít nhất một trong các giá trị trong danh sách. Điều gì sẽ xảy ra nếu bạn cần thực hiện một so sánh không bằng của một giá trị với  một danh sách các giá trị?. Lấy ví dụ, bạn muốn biết sản phẩm nào có giá lớn hơn giá của mọi sản phẩm mà do nhà cung cấp ở Florida cung cấp. Câu truy vấn sau đây giải quyết điều đó  SELECT P\_CODE, P\_ONHAND\*P\_PRICE   FROM PRODUCT   WHERE P\_ONHAND\*P\_PRICE > ALL (SELECT P\_ONHAND\*P\_PRICE                                   FROM PRODUCT                                  WHERE V\_CODE IN (SELECT V\_CODE                                                     FROM VENDOR                                                    WHERE V\_STATE= ‘FL’)); Trong tất cả các trường hợp sử dụng truy vấn con mà chúng ta đã xem xét từ trước đến giờ, câu truy vấn con là một phần của biểu thức điều kiện và nó luôn xuất hiện ở vế phải của một biểu thức. Đó là trường hợp của truy vấn con ở WHERE, HAVING, và IN cũng như toán tử ANY và ALL.  Nhắc lại rằng mệnh đề FROM dùng để liệt kê các bảng cần để lấy dữ liệu ra. Bởi vì đầu ra của một câu lệnh SELECT là một bảng khác (hoặc chính xác hơn là một bảng ảo khác), bạn có thể sử dụng câu truy vấn con SELECT trong mệnh đề FROM. Ví dụ, bạn muốn biết tất cả các khách hàng những người đã mua sản phẩm 13-Q2/P2 và 23109-HB. Vì tất cả việc mua bán các sản phẩm được lưu trong bảng LINE nên dễ dàng tìm thấy khách hàng mua bất kỳ một sản phẩm nào bằng cách tìm kiếm thuộc tính P\_CODE trong bảng LINE. Tuy nhiên, trong trường hợp này, bạn muốn biết tất cả các khách hàng mua cả hai loại hàng chứ không chỉ một loại. Câu truy vấn được thể hiện như dưới đây: SELECT DISTINCT CUSTOMER.CUS\_CODE, CUSTOMER.LNAME   FROM CUSTOMER,(SELECT INVOICE.CUS\_CODE                     FROM INVOICE NATURAL JOIN LINE                    WHERE P\_CODE = ’13-Q2/P2’) CP1,                 (SELECT INVOICE.CUS\_CODE                    FROM INVOICE NATURAL JOIN LINE                   WHERE P\_CODE = ‘23109-HB’) CP2  WHERE CUSTOMER.CUS\_CODE = CP1.CUS\_CODE     AND CP1.CUS\_CODE = CP2.CUS\_CODE; |