Lời nói đầu

Giáo trình này được soạn thảo dùng để hỗ trợ giảng dạy cho môn Nhập môn Cơ sở dữ liệu tại các trường Đại học. Giáo trình được xây dựng trên cuốn sách Fundamental of Relational Databases (Ramon A. Mata-Toledo, Ph.D. và Pauline K. Cushman, Ph.D.).

Tôi xin tri ân đến các bà đã nuôi dạy tôi, các thầy cô đã tận tâm chỉ dạy tôi. Chỉ có cống hiến hết mình cho tri thức tôi mới thấy mình đền đáp được công ơn đó.
Tôi xin cảm ơn gia đình đã hy sinh rất nhiều để tôi có được khoảng thời gian cần thiết thực hiện được giáo trình này.
Tôi đặc biệt cảm ơn bạn Đinh Trương Nhật Hà đã đóng góp công sức rất lớn để giúp hoàn thành tài liệu.

Mặc dù đã dành rất nhiều thời gian và công sức, hiệu chỉnh chi tiết và nhiều lần, nhưng giáo trình không thể nào tránh được những sai sót và hạn chế. Tôi thật sự mong nhận được các ý kiến góp ý từ bạn đọc để giáo trình có thể hoàn thiện hơn. Nội dung giáo trình sẽ được cập nhật định kỳ, mở rộng và viết chi tiết hơn; tùy thuộc những phản hồi, những đề nghị, những ý kiến đóng góp từ bạn đọc.

Các bạn đồng nghiệp nếu có sử dụng giáo trình này, xin gửi cho tôi ý kiến đóng góp phản hồi, giúp giáo trình được hoàn thiện thêm, phục vụ cho công tác giảng dạy chung.

Phiên bản

Cập nhật ngày: **12/07/2017**

Thông tin liên lạc

Mọi ý kiến và câu hỏi có liên quan xin vui lòng gửi về:
Dương Thiên Tứ
91/29 Trần Tấn, P. Tân Sơn Nhì, Q. Tân Phú, Thành phố Hồ Chí Minh
Facebook: https://www.facebook.com/tu.duongthien
E-mail: thientu2000@yahoo.com
Trung tâm: CODESCHOOL – http://www.codeschool.vn
Fanpage: https://www.facebook.com/codeschool.vn

Cơ sở dữ liệu quan hệ

1. Hệ quản trị cơ sở dữ liệu quan hệ

Các hệ quản trị cơ sở dữ liệu dựa trên các mô hình dữ liệu cho phép mô tả dữ liệu ở mức luận lý hay cao hơn. Một hệ quản trị cơ sở dữ liệu dựa trên mô hình dữ liệu quan hệ được gọi là *Hệ Quản Trị Cơ Sở Dữ Liệu Quan Hệ* (RDBMS, Relational Database Management System). Trong kiểu cơ sở dữ liệu này, thông tin tạo thành được biểu diễn dưới dạng một tập hợp *các quan hệ*. Nhờ đó RDBMS có thể được định nghĩa như một tập hợp các quan hệ. Mặc dù một quan hệ có thể được định nghĩa bằng thuật ngữ toán học, tuy nhiên với mục đích thực hành, chúng ta sẽ biểu diễn các quan hệ dưới dạng các bảng hai chiều thỏa các điều kiện sẽ được giải thích sau. Vì lý do đó trong tài liệu này ta sẽ dùng thuật ngữ bảng và quan hệ thay thế nhau. Thực tế, thuật ngữ bảng được dùng trong các ngôn ngữ lập trình, còn thuật ngữ quan hệ được dùng trong lý thuyết cơ sở dữ liệu).

Bạn hãy nhớ rằng dữ liệu trong RDBMS có liên quan *luận lý* đến các bảng. Nghĩa là các bảng là cấu trúc dữ liệu luận lý mà ta giả sử rằng nó chứa dữ liệu cơ sở dữ liệu muốn biểu diễn. Bảng không phải là cấu trúc vật lý. Mỗi bảng có một tên duy nhất. Trong bảng có chứa một số *cột* (columns) hay *thuộc tính* (attribute). Mỗi cột trong bảng có một tên duy nhất. Số lượng cột (hay thuộc tính) trong bảng sẽ cho biết *bậc* (degree) của bảng. Dữ liệu trong bảng sẽ xuất hiện dưới dạng một tập hợp các *dòng* (row) hay *n-bộ* (n-tuples), trong đó n là số thuộc tính của bảng. Khi không rõ số lượng thuộc tính của bảng, thì ta sẽ bỏ qua tiền tố n và ta sẽ xem các dòng của bảng là các dòng hay bộ. Gọi một dòng là một n-tuple có ưu điểm là cho biết dòng đó có n giá trị. Tất cả các dòng trong bảng đều có chung định dạng và biểu diễn một số đối tượng hay mối quan hệ trong thế giới thực. Tổng số dòng trong bảng tại *một thời điểm bất kỳ* được gọi là *bản số* (cardinality) của bảng. Trong một số sách, các thuật ngữ *trường* (field) và *bản ghi* (record) được dùng thay cho các thuật ngữ thuộc tính và dòng.

Chúng ta sẽ gọi nội dung của bảng tại một thời điểm nào đó là *thể hiện* (instance hoặc snapshot) của bảng. Nói chung khi các bảng được định nghĩa, thì số lượng các cột trong bảng sẽ cố định trong quá trình bảng tồn tại. Tuy nhiên, số lượng các dòng trong bảng sẽ thay đổi theo thời gian để phản ánh thế giới thực mà bảng biểu diễn luôn biến đổi. Ví dụ như một số dòng mới có thể được thêm vào hay một số dòng cũ có thể bị xóa bỏ. Chú ý là bảng phải có ít nhất một cột, nhưng không nhất thiết phải có ít nhất một dòng. Bảng không có dòng nào được gọi là bảng rỗng (empty table). Quá trình chèn các bộ vào bảng gọi là *nhập dữ liệu cho bảng* (populating the table).

Với mỗi bảng sẽ có một tập hợp các giá trị có thể nhận được gọi là *miền giá trị* (domain). Miền giá trị chứa tất cả các giá trị nhận được có thể xuất hiện trong cột đó. Chúng ta biểu thị miền giá trị của một cột (hay thuộc tính) bằng *Dom(column-name)*. Mọi giá trị xuất hiện trong cột đều thuộc về miền giá trị của cột đó. Trong một bảng có thể xảy ra trường hợp có hai hay nhiều cột có chung miền giá trị.

Ví dụ

Cho quan hệ EMPLOYEE sau, hãy cho biết bậc và bản số của nó. Với mỗi thuộc tính hãy chỉ rõ miền giá trị có thể.

EMPLOYEE

Id	Last_Name	First_Name	Department	Salary
555294562	Martin	Nicholas	Accounting	55000
397182093	Benakritis	Ben	Marketing	33500
097803123	Adams	Larry	Human Resources	40000

Trong bảng này ta có thấy có năm thuộc tính hay cột: Id, Last_Name, First_Name, Department và Salary. Do đó bậc của quan hệ là năm. Bảng chỉ có ba dòng hay bộ. Do đó bản số của quan hệ là ba.

Miền giá trị có thể có mỗi thuộc tính như sau:

Miền giá trị của thuộc tính Id, ký hiệu là Dom(Id), là tập hợp các ký số. Ví dụ như ở Việt Nam, ta có thể dùng số chứng minh nhân dân của các nhân viên làm Id cho nhân viên đó, và nếu như thế Dom(Id) là tập hợp các số dương có 9 chữ số.

Miền giá trị của thuộc tính Last_Name, là Dom(Last_Name), là tập hợp các họ hợp lệ. Ta giả sử các họ tên chỉ dùng các chữ cái tiếng Anh thông thường. Số lượng ký tự của Last_Name hay First_Name sẽ do DBA quy định, nhưng chỉ có thể dao động trong khoảng 20 đến 256.

Miền giá trị của thuộc tính First_Name, là Dom(First_Name) là tập hợp các tên hợp lệ. Giả sử tên tuân theo quy tắc như họ đã nêu ở trên.

Đối với một số công ty, miền giá trị của thuộc tính Department, Dom(Department), là tập hợp các tên được chọn từ các tên phòng ban hợp lệ. Trong trường hợp này bảng chỉ có ba trong số các giá trị đó.

Miền giá trị của thuộc tính Salary, Dom(Salary), là một tập con của tập số thực không âm. Không âm vì lương âm không có ý nghĩa gì cả.

Ta giả sử rằng mọi ô nhập trong bảng hay quan hệ đều chứa nhiều nhất một trị (tức thỏa điều kiện dạng chuẩn 1, sẽ được thảo luận sau). Nghĩa là ô nhập tại giao điểm của một cột và dòng chỉ chứa tối đa một giá trị. Với mọi quan hệ r, thuộc tính A bất kỳ của r, và mọi bộ t của r, ta sẽ dùng ký hiệu *t(A)* để biểu diễn giá trị tại ô nhập của bộ t dưới cột A, nghĩa là giá trị tại giao điểm của cột A với dòng t.

Ví dụ

Cho quan hệ CUSTOMER_ORDER như dưới đây, thì giá trị của t(A) là gì nếu t và A là một bộ và thuộc tính tùy ý trong quan hệ?

CUSTOMER_ORDER

Id	Date_Ordered	Date_Shipped	Payment_Type
1	08/11/1999	08/12/1999	cash
2	08/12/1999	08/12/1999	purchase order
10	08/14/1999	08/15/1999	credit

Nếu ta gọi t là dòng đầu tiên trong quan hệ thì r(Id) = 1, t(Date_Ordered) = 8/11/1999, t(Date_Shipped) = 8/12/1999 và t(Payment_Type) = cash.

Còn nếu t là dòng thứ hai thì t(Id) = 2, t(Date_Ordered) = 8/12/1999, , t(Date_Shipped) = 8/12/1999 và t(Payment_Type) = purchase order.

Nếu t là dòng thứ ba thì (Id) = 10, t(Date_Ordered) = 8/14/1999, , t(Date_Shipped) = 8/15/1999 và t(Payment_Type) = credit.

Chú ý là các bộ của quan hệ này là 4-tuples. Nghĩa là mỗi bộ có 4 giá trị, mỗi giá trị cho một cột trong bảng.

Từ các định nghĩa trước đây và khái niệm bảng, thì khi nào một quan hệ được biểu diễn theo ý nghĩa của một bảng, ta sẽ giả sử các điều kiện sau thỏa mãn:
- Bảng có tên duy nhất.
- Mỗi cột trong bảng đều mang tên duy nhất. Nghĩa là không có hai cột nào trong một bảng có tên giống nhau.
- Trật tự của các cột trong bảng là không quan trọng.
- Tất cả các dòng trong bảng đều có chung định dạng và có số ô nhập.
- Các giá trị trong từng cột phải có chung miền giá trị (chuỗi, trị nguyên, trị thực, ngày giờ).
- Mọi ô nhập (entry, giao điểm của một dòng và một cột) của mọi bộ của quan hệ chứa tối đa một giá trị. Nghĩa là không có chuyện có danh sách hay tập hợp ở đây.
- Trật tự của các dòng là không quan trọng vì chúng được xác định thông qua nội dung của chúng chứ không phải qua vị trí.
- Không có một cặp dòng nào giống hệt nhau ở mọi vị trí. Về mặt toán học mà nói, thì một quan hệ là một tập hợp các ánh xạ, nên sẽ không có chuyện có hai phần tử trùng nhau. Hơn nữa, mỗi bản ghi đều cần có một thuộc tính đặc biệt để chứa giá trị duy nhất dùng nhận dạng từng dòng.

2. Định nghĩa toán học của quan hệ

Cho một tập hữu hạn các thuộc tính $A_1, A_2, A_3, \dots, A_n$, ta sẽ gọi *lược đồ quan hệ* R (scheme R) là tập hợp được tạo bởi tất cả các thuộc tính này. Nghĩa là $R = \{A_1, A_2, A_3, \dots, A_n\}$. Tương ứng với mỗi thuộc tính là các tập hợp khác rỗng D_i ($1 \le i \le n$) được gọi là miền giá trị của thuộc tính A_i, ký hiệu là $Dom(A_i)$. Gọi D là một tập hợp mới được định nghĩa như là hợp của tất cả các miền giá trị của các thuộc tính. Nói cách khác, $D = D_1 \cup D_2 \cup \dots \cup D_n$. Định nghĩa một *quan hệ* r trên lược đồ quan hệ R là một tập hợp hữu hạn các ánh xạ $\{t_1, t_2, t_3, \dots, t_k\}$ từ R vào D. Các ánh xạ t_i được gọi là các bộ hay n-tuples. Với mỗi bộ này, giá trị dưới từng cột A_i, ký hiệu là $t(A_i)$, phải là một phần tử của miền giá trị A_i. Nghĩa là, nếu t là một bộ của quan hệ r thì $t(A_i) \in Dom(A_i)$, trong đó ký hiệu $\in$ được đọc là "thuộc về". Nếu lược đồ R của quan hệ r là chưa xác định thì ta sẽ gọi quan hệ bằng tên của nó, còn không ta sẽ ký hiệu nó là r(R). Ở đây, chúng ta định nghĩa quan hệ là các tập hợp của các ánh xạ để tránh quan tâm đến thứ tự của thuộc tính, vì ta đã từng nói rằng thứ tự của các cột là không quan trọng.

3. Khóa dự tuyển và khóa chính của quan hệ

Khái niệm khóa (key) là một nội dung quan trọng trong mô hình quan hệ vì nó giúp có được cơ chế cơ bản để tìm các bộ trong mọi bảng của cơ sở dữ liệu. Cho quan hệ r và các thuộc tính của nó là $A_1, A_2, A_3, \dots, A_n$, ta gọi mọi tập con $K = \{A_1, A_2, A_3, \dots, A_k\}$ với ($1 \le k \le n$) của các thuộc tính này là khóa dự tuyển nếu K đồng thời thỏa hai điều kiện sau:

(1) Với mọi cặp bộ khác nhau t_1 và t_2 của quan hệ r sẽ tồn tại một thuộc tính A_j của K để $t_1(A_j) \neq t_2(A_j)$. Điều đó có nghĩa là không có hai bộ khác nhau nào của r sẽ có chung các giá trị tại tất cả các thuộc tính của K. Nói cách khác *ít nhất* phải có một biểu thức sau đúng: $t_1(A_1) \neq t_2(A_1)$, $t_2(A_1) \neq t_2(A_2)$, ... , $t_1(A_k) \neq t_2(A_k)$ với mọi cặp bộ trong quan hệ. Điều kiện này được gọi là *đặc tính duy nhất của khóa* (uniqueness property of key). Một số RDBMS cho phép người dùng tạo các bảng có các cột được định nghĩa với ràng buộc UNIQUE. Kiểu ràng buộc này sẽ giúp tránh các giá trị trùng trong những cột có ràng buộc đó.

(2) Không có tập con thực sự K' nào của K thỏa điều kiện duy nhất trên. Nghĩa là không thể bớt đi phần tử nào của K mà không làm K mất đi đặc tính duy nhất. Điều kiện này được gọi là *đặc tính tối thiểu của khóa* (minimality property of key), để đảm bảo rằng số lượng các thuộc tính tạo nên khóa là ít nhất.

Vì một quan hệ r có thể có nhiều khóa dự tuyển, nên sẽ có một trong các khóa dự tuyển đó được cài đặt thành *khóa chính* (PK, primary key) của quan hệ. Giá trị của khóa chính thường được dùng như cơ chế để nhận diện và tìm kiếm trong quan hệ. Nghĩa là sự khác biệt giữa các dòng trong bảng chủ yếu dựa trên các giá trị khóa chính, chúng ta chỉ tìm thấy một bộ duy nhất trong quan hệ dựa trên giá trị khóa chính của bộ đó. Khi khóa chính đã được chọn lựa thì các khóa dự tuyển còn lại sẽ được gọi là *khóa thay thế* (alternate key). Một RDBMS chỉ cho phép có một khóa chính trong một bảng. Một khóa chính có thể được tạo thành từ một thuộc tính (*khóa chính đơn,* single PK) hay từ nhiều thuộc tính (*khóa chính phức*, composite PK). Các thuộc tính tạo nên khóa (chính hay thay thế) được gọi các *thuộc tính khóa* (prime attributes). Trong tài liệu này chúng ta sẽ gạch chân các thuộc tính thuộc về khóa chính.

Vì khóa chính được dùng để nhận dạng tính duy nhất của các bộ hay dòng trong quan hệ nên không có chuyện một thuộc tính nào của nó có thể là NULL. Yếu tố này tạo thêm cho khóa một điều kiện hoặc *ràng buộc* (constraint), gọi là *ràng buộc toàn vẹn* (integrity constraint). Trong một quan hệ, một giá trị NULL được dùng để biểu diễn thông tin chưa có, chưa biết hoặc không thể sử dụng. Bạn nên nhớ rằng giá trị NULL khác với số 0 cũng không giống với khoảng trống, là hai ký tự có thực trong bảng mã.

Ví dụ

Cho bảng DEPT và các dòng như dưới đây. Hãy cho biết ba dòng bên dưới có thể được chèn vào bảng DEPT hay không. Chú ý DEPARTMENT là khóa của bảng.

DEPT

DEPARTMENT	NAME	LOCATION	BUDGET
20	Sales	Miami	1700000
10	Marketing	New York	2000000

10	Research	New York	1500000	*KHÔNG. Vi phạm đặc tính duy nhất của khóa vì đã có phòng số 10.*
	Accounting	Atlanta	1200000	*KHÔNG. Vi phạm ràng buộc toàn vẹn của khóa, vì khóa không thể NULL.*
15	Computing	Miami	1500000	*CÓ. Không có ràng buộc nào vi phạm cả.*

Cho khóa K của quan hệ r, ta nói *siêu khóa* (superkey) của quan hệ r là mọi tập thuộc tính K' của quan hệ có chứa khóa. Nghĩa là nếu K' là siêu khóa thì K' là tập con của K, K' ⊂ K. Tức là mọi tập thuộc tính của quan hệ r có chứa khóa đều được gọi là siêu khóa. Rõ ràng là siêu khóa thỏa đặc tính duy nhất của khóa nhưng không thỏa đặc tính tối thiểu của khóa.
Các khóa, như đã nói, là cơ chế cơ bản để nhận diện và tìm kiếm các bộ khác nhau trong quan hệ. Do đó khi chọn khóa ta cần chọn các thuộc tính thỏa điều kiện duy nhất và tối thiểu cho tất cả các trường hợp dữ liệu cho phép. Nghĩa là khi chọn khóa không nên chỉ xét trường hợp dữ liệu đang có trong bảng. Chúng ta cũng nên xem xét đến ý nghĩa của các thuộc tính đang chọn.
Cho quan hệ r, một tập con X = {A_1, A_2, A_3, ... , A_k} của các thuộc tính của r, với mọi bộ t của r, ta gọi X-giá trị của t, ký hiệu là t{A_1, A_2, A_3, ... , A_k} là k-bộ {t(A_1), t(A_2), t(A_3), ... , t(A_k)}. Nghĩa là X-giá trị của một bộ t là một k-bộ trong đó các thành phần là các giá trị tại giao điểm của bộ t và các thuộc tính A_1, A_2, A_3, ... , A_k. Nếu thứ tự của các thuộc tính được biết thì X-giá trị của t có thể có được ký hiệu là t(X).
Khóa chính được định nghĩa bằng cách dùng các lệnh DDL và sẽ được RDBMS tự động ép thỏa. Chúng thường được định nghĩa tại thời điểm tạo bảng.

4. Khóa ngoại

Bởi vì các cột có cùng miền giá trị có thể được dùng để liên kết các bảng trong một cơ sở dữ liệu, nên khái niệm khóa ngoại sẽ cho phép RDBMS duy trì tính chất nhất quán giữa các dòng trong hai quan hệ hay trong cùng một quan hệ. Khái niêm khóa ngoại được định nghĩa như sau: cho hai quan hệ r_1 và r_2 trong cùng một cơ sở dữ liệu (trong một số trường hợp đặc biệt, r_1 và r_2 có thể cùng một quan hệ), thì một tập các thuộc tính FK của quan hệ r_1 được gọi *là khóa ngoại* (foreign key) của r_1 (tham chiếu đến r_2) nếu hai điều kiện sau đồng thời được thỏa mãn:
- Các thuộc tính của FK có cùng miền giá trị với các thuộc tính khóa chính của r_2 (một số DBMS chỉ yêu cầu tập thuộc tính trong r_2 thỏa đặc tính duy nhất chứ không bắt buộc là khóa chính). Khi đó FK được gọi là tham chiếu đến (các) thuộc tính khóa của quan hệ r_2.
- Các FK-giá trị trong một bộ của quan hệ r_1 hoặc là NULL hoặc phải xuất hiện trước như PK-giá trị của một bộ nào đó trong quan hệ r_2.

Dưới quan điểm thực hành, khái niệm khóa ngoại bảo đảm rằng các bộ của quan hệ r_1 tham chiếu đến các bộ của quan hệ r_2 đã có sẵn. Điều kiện này trong khóa ngoại gọi là ràng buộc *toàn vẹn tham chiếu* (referential integrity). Một số tác giả gọi bảng chứa khóa ngoại là *bảng con*, còn bảng chứa các thuộc tính được tham chiếu là *bảng cha*. Nếu dùng các thuật ngữ đó ta có thể nói rằng các giá trị của FK trong mỗi dòng của bảng con phải là NULL hoặc phải so trùng giá trị của PK của một bộ nào đó trong bảng cha.

Ví dụ
Xét các bảng sau. Giả sử rằng thuộc tính EMP_DEPT là khóa ngoại của bảng EMPLOYEE tham chiếu đến thuộc tính ID của DEPARTMENT. Hãy cho biết ba dòng bên dưới có thể được chèn vào bảng EMPLOYEE hay không.

DEPARTMENT

ID	NAME	LOCATION
10	Accounting	New York
40	Sales	Miami

EMPLOYEE

EMP_ID	EMP_NAME	EMP_MGR	TITLE	EMP_DEPT
1234	Green		President	40
4567	Gilmore	1234	Senior VP	40
1045	Rose	4567	Director	10
9876	Smith	1045	Accountant	10

9213	Jones	1045	Clerk	30
8997	Grace	1234	Secretary	40
5932	Allen	4567	Clerk	

KHÔNG. Vi phạm ràng buộc toàn vẹn tham chiếu. Không có phòng 30 nào trong bảng DEPARTMENT.
CÓ. Không có ràng buộc nào bị vi phạm.
CÓ. Giá trị NULL là chấp nhận được đối với cột EMP_DEPT. Giá trị NULL cho biết nhân viên này chưa được phân về phòng nào.

Chú ý là trong ví dụ vừa rồi ta dùng từ khóa NULL (từ khóa là từ có ý nghĩa tùy hệ thống và chỉ dùng từ đó với ý nghĩa đã quy định) để diễn tả rằng nhân viên đó chưa được phân về phòng nào.
Các khóa ngoại thường được định nghĩa sau khi tất cả các bảng đã được xây dựng và nhập liệu. Điều này giúp tránh được các rắc rối có thể gặp phải như vấn đề xoay vòng: bảng này tham chiếu đến bảng kia, rồi bảng kia lại tham chiếu về bảng này. Rắc rối ở chỗ là các ràng buộc toàn vẹn được định nghĩa bằng cách dùng các lệnh DDL và được RDBMS tự động ép thỏa.

5. Các toán tử quan hệ

Ta gọi các *toán tử quan hệ* (relational operators) là tập hợp các toán tử cho phép ta thao tác trên các bảng của cơ sở dữ liệu. Còn các tập hợp các toán tử cho phép tạo nên các quan hệ mới từ một tập hợp các quan hệ cho trước được gọi là *đại số quan hệ* (relational algebra). Các toán tử quan hệ được gọi là thỏa *đặc tính bao đóng* (closure preoperty) vì chúng làm việc trên một hay nhiều quan hệ để tạo ra các quan hệ mới. Khi một toán tử quan hệ được dùng để thao tác trên một quan hệ thì ta nói toán tử đó đang được "áp dụng" cho quan hệ đó. Để định nghĩa một toán tử quan hệ ta dùng *phép tính quan hệ* (relational calculus). Để định nghĩa các toán tử quan hệ ta dùng ký hiệu như ∃ (tồn tại), ∀ (với mọi), ∨ (hoặc, hay), ∧ (và), ~ (phủ định), ∈ (thuộc), ⊂ (con), ∅ (tập rỗng), / hoặc ϶ (sao cho), và các ký hiệu phủ định như ∉ (không thuộc), ⊄ (không phải là con). Vì các toán tử quan hệ thường được dùng dưới dạng các biểu thức, nên cần nói rõ rằng biểu thức bên trái của dấu bằng là biểu thức đại số quan hệ của toán tử, còn biểu thức bên phải là định nghĩa phép toán quan hệ của toán tử.

Trong phần này, ta chỉ đề cập đến các toán tử quan hệ Chọn (Selection), Chiếu (Projection) và Kết bằng (Equijoin). Một số tác vụ khác, đặc biệt đối với kiểu luận lý (Boolean) sẽ được đề cập đến ở cuối chương. Mỗi tác vụ trên quan hệ là để trả lời một câu hỏi nảy sinh từ cơ sở dữ liệu. Trong thuật ngữ cơ sở dữ liệu, câu hỏi được gọi là câu *truy vấn* (query). Ta sẽ dùng thuật ngữ này để nói về những tác vụ quan hệ được thực hiện trên các quan hệ nhằm mục đích lấy thông tin từ cơ sở dữ liệu.

5.1. Phép Chọn (Selection)

Phép toán này khi được áp dụng cho mọi quan hệ r sẽ tạo ra một quan hệ khác trong đó các dòng được lấy từ các dòng của r thỏa một số giá trị nào trên thuộc tính chỉ định. Quan hệ kết quả và r sẽ có cùng các thuộc tính. Ta cũng có thể định nghĩa phép toán này như sau. Cho r là một quan hệ trên lược đồ R, A là một thuộc tính cụ thể của r và có một giá trị a của Dom(A). Phép Chọn của r trên thuộc tính A dựa trên giá trị a là một tập các bộ t của quan hệ r thỏa t(A) = a. Nghĩa là, tất cả các dòng của quan hệ mới – quan hệ Chọn – đều có giá trị a ở cột A. Phép Chọn của r trên A được ký hiệu là $\sigma_{A=a}(r)$. Chú ý là mệnh đề A = a của $\sigma_{A=a}(r)$ mang ý nghĩa của t(A) = a. Còn định nghĩa của phép Chọn theo kiểu toán học là như sau:

$\sigma_{A=a}(r) = \{t \in r / t(A) = a\}$

Lược đồ quan hệ mới, $\sigma_{A=a}(r)$ giống như lược đồ của quan hệ r. Nghĩa là như đã nói, $\sigma_{A=a}(r)$ và r có cùng các thuộc tính. Ví dụ sau sẽ minh họa cho thấy phép Chọn làm việc như thế nào. Chú ý là phép Chọn là phép toán một ngôi (unary operator), nghĩa là mỗi lúc nó chỉ làm việc trên *một quan hệ*. Trên quan điểm thực hành, phép toán này được áp dụng cho quan hệ khi ta muốn lấy tất cả các thông tin có thể từ một dòng hoặc nhiều dòng có một cột nào đó thỏa một giá trị cụ thể.

Ví dụ

Cho quan hệ EMPLOYEE của ví dụ trước, hãy tìm tất cả các thông tin của nhân viên làm việc cho phòng 10.

Vì ta muốn lấy tất cả các thông tin về các nhân viên làm việc cho phòng 10 nên ta cần có xác định $\sigma_{EMP\_DEPT=10}$(EMPLOYEE). Chú ý là trong ví dụ này điều kiện cần thỏa cho các bộ quan hệ EMPLOYEE sẽ xuất hiện trong quan hệ Chọn mới là t(EMP_DEPT) = 10. Bảng kết quả sẽ như sau:

$\sigma_{EMP\_DEPT=10}$(EMPLOYEE)

EMP_ID	EMP_NAME	EMP_MGR	TITLE	EMP_DEPT
1234	Green		President	40
4567	Gilmore	1234	Senior VP	40
1045	Rose	4567	Director	10
9876	Smith	1045	Accountant	10

Chú ý là tất cả các dòng của quan hệ mới này đều có giá trị 10 ở cột EMP_DEPT

5.2. Phép Chiếu (Projection)

Phép Chiếu cũng là một phép toán đơn. Nếu như phép Chọn để chọn một tập con các dòng trong quan hệ, thì phép Chiếu để chọn một tập con các cột. Định nghĩa của phép toán này như sau. *Chiếu của quan hệ r lên một tập X của các thuộc tính* của nó, ký hiệu là $\pi_X(r)$ là một quan hệ mới có được bằng cách trước hết loại bớt các cột của r không có trong X rồi bỏ bớt các dòng trùng. Các thuộc tính của X là các cột của quan hệ Chiếu. Quan hệ Chiếu cũng có thể được định nghĩa theo kiểu toán học như sau: Cho r là quan hệ của lược đồ quan hệ R và cho $X = \{A_1, A_2, A_3, \ldots, A_k\}$ là tập con các thuộc tính của nó, khi đó $\pi_X(r) = \{t(X) / t \in r\}$. Chú ý là các ô nhập của dòng của quan hệ Chiếu được tạo nên bằng cách lấy từ từng bộ t của những ô nhập tương ứng với các thuộc tính được định nghĩa trong X. Nghĩa là các ô nhập cho một bộ j của $\pi_X(r)$ được tạo nên bằng cách chọn các ô nhập $t_j(A_1), t_j(A_2), \ldots, t_j(A_k)$ từ bộ j của quan hệ t. Vì các quan hệ đã được định nghĩa dưới dạng các tập hợp nên cần thiết loại bỏ các bộ trùng khỏi $\pi_X(r)$. Dưới quan điểm thực hành, phép toán này được áp dụng khi ta cần có các giá trị khác nhau thuộc về một cột hay các bộ giá trị khác nhau thuộc về hai hay nhiều cột. Ví dụ sau sẽ minh họa cho cách dùng phép Chiếu.

Ví dụ

Dùng bảng DEPARTMENT dưới đây, hãy cho biết có địa điểm của các phòng ban khác nhau? Từ đó hãy cho biết thêm có bao nhiêu địa điểm trong bảng DEPARTMENT?

DEPARTMENT

ID	NAME	LOCATION
10	Accounting	New York
30	Computing	New York
50	Marketing	Los Angeles
60	Manufacturing	Miami

Vì ta muốn xác định những giá trị khác nhau hiện có trong cột LOCATION, nên ta cần dùng đến phép Chiếu của bảng DEPARTMENT trên thuộc tính LOCATION. Nghĩa là ta cần tìm $\pi_{LOCATION}$(DEPARTMENT). Trong trường hợp này r = DEPARTMENT và X = {LOCATION}. Các bộ của phép Chiếu được hình thành nhờ trích giá trị t(LOCATION) của từng bộ của quan hệ DEPARTMENT. Vì có hai bộ có cùng giá trị New York nên sẽ chỉ có một xuất hiện trong phép Chiếu. Tương tự với Miami. Do đó kết quả sẽ như sau:

$\pi_{LOCATION}$(DEPARTMENT)

LOCATION
New York
Los Angeles
Miami

Nhận xét rằng LOCATION là thuộc tính duy nhất của bảng.
Nói chung, phép Chiếu của quan hệ này trên thuộc tính LOCATION không thể cho ta biết tổng số địa điểm có trong bảng DEPARTMENT vì các giá trị trùng đã bị loại trừ. Chú ý là quan hệ Chiếu chỉ có ba địa điểm trong khi quan hệ DEPARTMENT có tới năm.

Ví dụ
Sử dụng lại bảng DEPARTMENT trong ví dụ vừa rồi, hãy liệt kê các phòng khác nhau cũng như địa điểm của chúng?
Trong trường hợp này X = {NAME, LOCATION} và r = DEPARTMENT. Do đó quan hệ kết quả sẽ như sau.

$\pi_{NAME, LOCATION}$(DEPARTMENT)

NAME	LOCATION
Accounting	New York
Computing	New York
Marketing	Los Angeles
Manufacturing	Miami

Quan hệ kết quả không chứa các giá trị trùng vì sự kết hợp khác nhau giữa NAME và LOCATION là duy nhất. Chú ý là các bộ (Manufacturing, Miami) và (Sales, Miami) là khác nhau. Tương tự (Accounting, New York) và (Computing, New York) cũng là hai bộ khác nhau.
Trước khi xem xét phép toán tiếp theo, ta cần dành một chút thời gian để nói về *phép nối* (concatenation operator). Đây là một phép toán trên bộ chứ không phải trên bảng. Cho hai bộ s = (s_1, s_2, …, s_n) và r = (r_1, r_2, …, r_m), khi đó phép nối của r và s là (m+n)-tuple được định nghĩa như sau:
$\overrightarrow{rs}$ = (s_1, s_2, ... , s_n, r_1, r_2, ... , r_m) trong đó ký hiệu $\overrightarrow{rs}$ được đọc là nối của hai bộ r và s. Chú ý là số lượng các giá trị trong các r và s không nhất thiết giống nhau. Ví dụ như nếu r = (a, b, c) và s = (1, 2) thì $\overrightarrow{rs}$ = (a, b, c, 1, 2).

5.3. Phép Kết bằng (Equijoin)

Phép kết bằng (còn gọi là kết tự nhiên, natural join) là phép toán hai ngôi kết hợp hai quan hệ, trong đó hai quan hệ không nhất thiết phải khác nhau. Nói chung, phép toán này kết hợp hai quan hệ dựa trên các thuộc tính chung của chúng. Như vậy, quan hệ kết quả bao gồm tất cả các bộ được nối từ các bộ của quan hệ thứ nhất với các bộ của quan hệ thứ hai có chung các giá trị ở các thuộc tính chung X. Những thuộc tính chung mà ta nói ở đây không nhất thiết phải có tên giống nhau, nhưng chúng phải có miền giá trị và ý nghĩa giống nhau. Ta cũng có thể định nghĩa phép Kết bằng theo kiểu toán học như sau: Cho r là một quan hệ có tập thuộc tính R và s cũng là một quan hệ có tập thuộc tính S. Ngoài ra X là tập thuộc tính chung của R và S, nghĩa là $R \cap S = X$. Phép kết bằng của r và s, ký hiệu là *r Join s* là một quan hệ mới có các thuộc tính là các phần tử của $R \cup S$. Hơn nữa với *mỗi bộ t* của quan hệ *r Join s* sẽ có ba điều kiện sau đồng thời thỏa mãn:
(1) t(R) = t_r đối với các bộ t_r của quan hệ r,
(2) t(S) = t_s đối với các bộ t_s của quan hệ s, và
(3) $t_s(X) = t_r(X)$.
Sau đây là một cách định nghĩa khác của phép Kết bằng:
r Join s = {$\overrightarrow{rs}$ / r $\in$ R và s $\in$ S và r(R $\cap$ S) = s(R $\cap$ S)}
Hai định nghĩa vừa rồi là tương đương nhau. Ví dụ tiếp theo sẽ minh họa cho cách làm việc của phép toán này.

Ví dụ
Hãy kết hai bảng sau dựa trên các thuộc tính chung là DEPT và NAME. Hãy viết một câu truy vấn có thể có để thỏa kết quả của tác vụ này. Hai bảng này có thể kết trên thuộc tính ID được không?

DEPARTMENT

ID	NAME	LOCATION
100	Accounting	Miami
200	Marketing	New York
300	Sales	Miami

EMPLOYEE

ID	NAME	DEPT	TITLE
100	Smith	Sales	Clerk
200	Jones	Marketing	Clerk
300	Martin	Accounting	Clerk
400	Bell	Accounting	Sr. Accountant

Trong trường hợp này, các thuộc tính chung này là DEPT và NAME. Kết hai bảng này, ký hiệu là DEPARTMENT *Join* EMPLOYEE, được trình bày như sau. Vì cả hai bảng đều có thuộc tính ID nên để tránh nhầm lẫn, ta phải dùng thêm tiền tố là tên bảng phía trước cột ID. Tương tự

đối với thuộc tính NAME, cần phải thực hiện điều này trước khi kết các bảng. Điều này không có gì mâu thuẫn với yêu cầu rằng các tên trong bảng phải khác nhau. Kết quả của phép Kết này có thể đáp ứng truy vấn "hiển thị thông tin về các nhân viên kèm theo ID , tên và địa điểm của phòng ban của nhân viên đó".

DEPARTMENT *Join* EMPLOYEE

DEPT_ID	DEPT_NAME	LOCATION	EMP_ID	EMP_NAME	TITLE
100	Accounting	Miami	300	Martin	Clerk
100	Accounting	Miami	400	Bell	Sr. Accountant
200	Marketing	New York	200	Jones	Clerk
300	Sales	Miami	100	Smith	Clerk

Trong phép Kết này chú ý là thuộc tính chung là DEPARTMENT.NAME. Nghĩa là X = {DEPT}. Hơn nữa rõ ràng là phép Kết này thỏa ba điều kiện vừa rồi. Thực ra nếu r là quan hệ DEPARTMENT và s là quan hệ EMPLOYEE thì ra sẽ có các lược đồ tương ứng là:
R = {ID, NAME, LOCATION}
và S = {ID, NAME, DEPT, TITLE}
Rõ ràng là với mọi bộ t của quan hệ Kết ta đều có:
(1) t(R) = t(ID, NAME, LOCATION) = t_r với các bộ của r.
(2) t(S) = t(ID, NAME, TITLE) = t_r với các bộ của s.
(3) $t_r(X) = t_s(X)$
Nếu dùng cả các tên thuộc tính như đã giải thích và chỉ xét bộ đầu tiên của quan hệ kết thì ta có:

DEPT_ID	DEPT_NAME	LOCATION	EMP_ID	EMP_NAME	TITLE
100	Accounting	Miami	300	Martin	Clerk

Nếu gọi t là bộ đầu tiên thì phải chú ý là
t(DEPT_ID, DEPT_NAME, LOCATION) = (100, Accounting, Miami) = t_r
t(EMP_ID, EMP_NAME, DEPT, TITLE) = (300, Martin, Accounting, Clerk) = t_s
t_r(DEPT_NAME) = t_s(DEPT) = Accounting
Bạn có thể kiểm tra để thấy rằng cả ba điều kiện đều thỏa mãn đối với mọi bộ còn lại của quan hệ.
Các bảng DEPARTMENT và EMPLOYEE *không thể* kết trên thuộc tính ID vì thuộc tính ID của bảng EMPLOYEE và thuộc tính ID của bảng DEPARTMENT không cùng ý nghĩa, đơn giản vì ID của nhân viên không thể đem so với ID của phòng ban. Điều này một lần nữa nhắc lại rằng hai bảng vẫn không thể kết với nhau dù chúng có thuộc tính cùng tên. Để kết được hai bảng trên thuộc tính chung, các thuộc tính này cần có chung miền giá trị và chung ý nghĩa.
Thực ra có nhiều kiểu kết khác nhau trong cơ sở dữ liệu. Phép kết bằng hay kết tự nhiên mới được trình bày yêu cầu rằng giá trị của các thuộc tính chung phải bằng nhau. Còn *phép kết đôi* (compsition) là phép kết tự nhiên trong đó có một hay nhiều thuộc tính chung bị xóa. Và *phép kết ba* (theta join) là phép kết mà điều kiện cần thỏa bởi các bộ có thể dùng các phép so sánh như ≤, ≥, ≠, < , hoặc >. Nói chung, hai phép kết sau không phải là được tất cả các RDBMS hỗ trợ, chúng được trình bày trong phần Bài tập.

6. Các phép toán tập hợp trên quan hệ

Vì các quan hệ được định nghĩa như các tập hợp toán học, nên có thể áp dụng các tác vụ hai ngôi cơ bản – các phép toán tập hợp – lên chúng. Các phép toán này bao gồm: Hội (Union), Giao (Intersection), Trừ (Difference) và tích Descartes (Cartesian product). Trong các phép toán này thì có lẽ hội là mạnh và thú vị nhất. Ba phép toán đầu tiên yêu cầu các thuộc tính của các quan hệ tham gia phải *khả hợp* (union compartible). Ta nói hai tập thuộc tính A và B là khả hợp nếu chúng cùng bậc và các miền giá trị có cùng kiểu dữ liệu. Chú ý là định nghĩa này không yêu cầu các tên thuộc tính phải như nhau.
Tất cả các phép toán này đều được hỗ trợ trong lệnh SELECT của SQL. Vì kết quả của mỗi phép toán này đều là một bảng và các thuộc tính của các bảng tham gia không nhất thiết phải như nhau, nên câu hỏi được đặt ra ở đây, là tên các cột của bảng kết quả sẽ như thế nào? Trong tài liệu này chúng ta tự đặt ra một quy ước là bảng kết quả sẽ có các tên cột như bảng "thứ nhất".

6.1. Phép Hội (UNION)

Cho hai quan hệ r(R) và s(S) có các lược đồ khả hợp, khi đó phép Hội của hai quan hệ, ký hiệu là $r \cup s$, là tập hợp các bộ có trong r hoặc có trong s hoặc có trong cả hai quan hệ. Phép Hội cũng có thể được định nghĩa theo kiểu toán học như sau:
$r \cup s = \{t / t \in r \vee t \in s\}$ trong đó ký hiệu $\cup$ được gọi là "hội" (union) và ký hiệu $\vee$ được gọi là "hoặc" (or).
Giống như mọi quan hệ khác, sẽ không có chuyện có các bộ trùng trong quan hệ Hội. Lược đồ của quan hệ Hội, nghĩa là tập thuộc tính của nó, là lược đồ của quan hệ r vì ta đã quy ước tên cột của kết quả là tên cột của bảng thứ nhất (r) tham gia phép toán Hội. Rõ ràng là nếu ta ghi $s \cup r$ thì lược đồ kết quả sẽ như s.

Ví dụ
Cho các quan hệ như dưới đây, hãy tìm hội của hai quan hệ đó. Phép toán này trả lời cho câu truy vấn nào?

C_PROGRAMMER

Employee_Id	Last_Name	First_Name	Project	Department
101123456	Venable	Mark	E-commerce	Sales Department
103705430	Cordam	John	Firewall	Information Technology
101936822	Serrano	Areant	E-commerce	Sales Department

JAVA_PROGRAMMER

Employee_Id	Last_Name	First_Name	Project	Department
101799332	Barnes	James	Web Application	Information Technology
101936822	Serrano	Areant	E-commerce	Sales Department

C_PROGRAMMER ∪ JAVA_PROGRAMMER

Employee_Id	Last_Name	First_Name	Project	Department
101123456	Venable	Mark	E-commerce	Sales Department
103705430	Cordam	John	Firewall	Information Technology
101799332	Barnes	James	Web Application	Information Technology
101936822	Serrano	Areant	E-commerce	Sales Department

Chú ý là ở đây không có các bộ trùng nhau trong quan hệ kết quả. Chú ý thêm là cũng không thể kiểm tra trong quan hệ kết quả để biết được những dòng nào được chọn từ bảng nào. Phép toán này trả lời truy vấn yêu cầu thông tin về tất cả các nhân viên là lập trình viên C hay Java hoặc cả hai.

6.2. Phép Giao (INTERSECTION)

Cho hai quan hệ r(R) và s(S) có các lược đồ khả hợp, khi đó phép Giao của hai quan hệ, ký hiệu là r ∩ s, là tập hợp các bộ có trong cả hai quan hệ. Phép Giao cũng có thể được định nghĩa theo kiểu toán học như sau:
r ∩ s = {t / t ∈ r ∧ t ∈ s} trong đó ký hiệu ∩ được gọi là "giao" (intersection) và ký hiệu ∧ được gọi là "và" (and). Cũng như mọi quan hệ khác, sẽ không có chuyện có các bộ trùng trong quan hệ Giao.

Ví dụ
Cho các quan hệ như ví dụ trước, hãy tìm giao của hai quan hệ đó. Phép toán này trả lời cho câu truy vấn nào?
Các bộ của quan hệ Giao là các bộ có trong cả hai quan hệ. Trong trường hợp này chỉ có một bộ thỏa điều kiện. Phép toán này trả lời truy vấn yêu cầu thông tin về các nhân viên vừa là lập trình viên C vừa là Java.

C_PROGRAMMER ∩ JAVA_PROGRAMMER

Employee_Id	Last_Name	First_Name	Project	Department
101936822	Serrano	Areant	E-commerce	Sales Department

Giao là phép toán có tính giao hoán. Nghĩa là r ∩ s = s ∩ r với mọi quan hệ r và s. Tính chất này tương tự đối với toán tử AND.

6.3. Phép Trừ (DIFFERENCE)

Cho hai quan hệ r(R) và s(R) có các lược đồ là khả hợp, khi đó phép Trừ của hai quan hệ r và s (theo thứ tự này), ký hiệu là r - s, là tập hợp các bộ có trong quan hệ r nhưng không có trong quan hệ s. Phép Trừ cũng có thể được định nghĩa theo kiểu toán học như sau:
r - s = {t / t ∈ r ∧ t ∉ s} trong đó ký hiệu ∉ được đọc là "không thuộc" và ký hiệu ∧ được đọc là "và" (and).

Ví dụ
Dùng các quan hệ trong ví dụ trước. Tìm hiệu của quan hệ C_PROGRAMMER - JAVA_PROGRAMMER và JAVA_PROGRAMMER - C_PROGRAMMER. Phép toán này trả lời cho câu truy vấn nào?
Quan hệ C_PROGRAMMER - JAVA_PROGRAMMER chứa các bộ có trong quan hệ C_PROGRAMMER và không có trong quan hệ JAVA_PROGRAMMER. Quan hệ Trừ như sau.

C_PROGRAMMER - JAVA_PROGRAMMER

Employee_Id	Last_Name	First_Name	Project	Department
101123456	Venable	Mark	E-commerce	Sales Department
103705430	Cordam	John	Firewall	Information Technology

Phép toán này trả lời truy vấn yêu cầu thông tin về các nhân viên là lập trình viên C nhưng không phải là lập trình viên Java.
Quan hệ JAVA_PROGRAMMER - C_PROGRAMMER chứa các bộ có trong quan hệ JAVA_PROGRAMMER và không có trong quan hệ C_PROGRAMMER. Quan hệ Trừ như sau.

JAVA_PROGRAMMER - C_PROGRAMMER

Employee_Id	Last_Name	First_Name	Project	Department
101799332	Barnes	James	Web Application	Information Technology

Tác vụ này trả lời truy vấn yêu cầu thông tin về các nhân viên là lập trình viên Java nhưng không phải là C.
Chú ý là việc đặt quan hệ nào đứng trước trong phép Trừ là quan trọng. Nói chung như bạn thấy, nếu có quan hệ r và s thì r - s và s – r là khác nhau. Điều này cũng có nghĩa là phép Trừ không có tính giao hoán.

6.4 Tích Descartes (Cartesian Product)

Cho hai quan hệ r(R) và s(S) khác rỗng, tích Descartes (còn được gọi là tích chéo) của hai quan hệ, ký hiệu là r ⊗ s , được định nghĩa là một quan hệ trong đó các bộ được tạo thành bởi việc nối từng bộ của quan hệ r với từng bộ của quan hệ s. Tích Descates cũng có thể được định nghĩa theo kiểu toán học như sau:

$r \otimes s = \{\overline{pq} / p \in r \text{ và } q \in s\}$ trong đó ký hiệu $\overline{pq}$ được đọc là "nối của các bộ p và q".
Bậc của quan hệ tích Descartes là tổng của các bậc của các quan hệ thành viên (trong trường hợp này là r và s) sau khi loại đi các tên trùng. Còn bản số của tích Descartes là tích các bản số của các quan hệ thành viên. Bạn nên cẩn thận khi dùng phép toán này, vì đôi khi quan hệ kết quả không có ý nghĩa gì. Phép toán này được áp dụng kèm với phép toán Chọn. Trong tất cả các phép toán tập hợp trên quan hệ thì tích Descartes là phép toán cần nhiều thời gian nhất để thi hành.

Ví dụ

Cho các quan hệ sau, hãy tìm tích Descartes của hai quan hệ đó.

BUYER

ID_NUMBER	ITEM
100	A
234	B
543	C

PRODUCT

CODE	NAME	PRICE
A	Bike	250
B	Spikes	90
C	Goggles	15
D	Gloves	35

Tích Descartes của hai quan hệ này được trình bày dưới đây. Như bạn thấy, quan hệ kết quả có 5 thuộc tính = 2 (từ BUYER) + 3 (từ PRODUCT) và 12 bộ = 3 (từ BUYER) * 4 (từ PRODUCT).

BUYER ⊗ PRODUCT

ID_NUMBER	ITEM	CODE	NAME	PRICE
100	A	A	Bike	250
100	A	B	Spikes	90
100	A	C	Goggles	15
100	A	D	Gloves	35
234	B	A	Bike	250
234	B	B	Spikes	90
234	B	C	Goggles	15
234	B	D	Gloves	35
543	C	A	Bike	250
543	C	B	Spikes	90
543	C	C	Goggles	15
543	C	D	Gloves	35

Chú ý là trong quan hệ này ngoài các bộ có ý nghĩa còn có một số bộ chẳng có ý nghĩa gì. Ví dụ như ở bộ thứ nhất thì khách hàng số 100 mua bike giá $250 là có ý nghĩa, nhưng còn ở bộ thứ hai thì sao?

7. Các phép toán chèn, xóa và cập nhật trên quan hệ

Nội dung của các quan hệ nói chung luôn thay đổi. Các phép toán thường áp dụng làm cho các nội dung thay đổi đó là chèn (INSERT), xóa (DELETE) và cập nhật (UPDATE). Mặc dù các phép toán này có thể được định nghĩa theo kiểu toán học, nhưng như thế sẽ quá phức tạp và khó dùng. Do đó ta sẽ định nghĩa chúng theo kiểu dễ hiểu hơn.

7.1. Chèn một bộ vào bảng

Giả sử có quan hệ r có lược đồ $R = \{A_1, A_2, A_3, \dots, A_n\}$. Khi đó cú pháp của phép toán Chèn như sau:
INSERT INTO *relation-name* $(A_1 = v_1, A_2 = v_2, A_3 = v_3, \dots, A_n = v_n)$
trong đó các giá trị $v_1, v_2, v_3, \dots, v_n$ thuộc về miền giá trị của các thuộc tính $A_1, A_2, A_3, \dots, A_n$. Nếu thứ tự của các thuộc tính đã biết thì ta có thể viết ngắn hơn là:
INSERT INTO *relation-name* $(v_1, v_2, v_3, \dots, v_n)$
Kết quả của phép toán này như tên của nó, cho biết sẽ chèn một bộ t vào quan hệ, trong đó $t(A_i) = v_i$.
Chú ý là không có điều gì đảm bảo rằng phép toán này sẽ thành công, phép toán Chèn có thể thất bại do một trong những lý do sau:
- Giá trị cho trước không thuộc về miền giá trị tương ứng của nó, nghĩa là $v_i \notin Dom(A_i)$.
- Bộ đang được chèn vào không có số lượng các ô nhập tương ứng với số lượng thuộc tính của lược đồ của quan hệ.
- Giá trị của khóa chính bị trùng lắp.
- Có thể có một hay vài giá trị vi phạm đặc tính duy nhất của thuộc tính tương ứng.
- Tên của quan hệ không có trong cơ sở dữ liệu, hoặc tên cột không có trong bảng.

Khi có sự vi phạm của một trong những điều kiện trên thì một thông báo lỗi sẽ được trả về. Tất nhiên thông báo lỗi như thế nào còn tùy vào RDBMS đang dùng. Ví dụ sau sẽ minh họa cho cách dùng phép toán chèn.

Ví dụ
Cho quan hệ PATIENT-ACCOUNT như sau, hãy cho biết những bộ nào sẽ được hay không được chèn vào quan hệ. Giả sử thêm rằng các dữ liệu ký tự phải nằm trong cặp nháy kép.
PATIENT-ACCOUNT (ACCOUNT, AMOUNT-DUE, DEPARTMENT, DOCTOR-ID, TREATMENT-CODE)
trong đó các miền giá trị tương ứng là:
Dom(ACCOUNT) = chuỗi ký tự, dài 9 ký tự.
Dom(AMOUNT-DUE) = số, tối đa 9 chữ số, trong đó có 2 số phần thập phân.
Dom(DEPARTMENT) = {G, T, L, X-rays, I}
Dom(DOCTOR-ID) = chuỗi số, 4 ký tự tối đa, dãy từ 1000 đến 2000.
Dom(TREATMENT-CODE) = {G100, G110, G120, T130, L140, L150, X160, X170, I180, I190, I200}

PATIENT-ACCOUNT

ACCOUNT	AMOUNT-DUE	DEPARTMENT	DOCTOR_id	TREATMENT-CODE
980543990	2456	G	1200	X160
804804308	245.45	T	1123	G100
173402644	589.25	L	1111	I180

a. INSERT INTO patient-account (ACCOUNT = "890345255", AMOUNT-DUE = 256, DEPARTMENT = "G", DOCTOR-ID = "1500", TREATMENT-CODE = "T130")
Bộ này được chèn vì nó chẳng vi phạm ràng buộc nào.
b. INSERT INTO patient-account (ACCOUNT = "980543990", AMOUNT-DUE = 256, DEPARTMENT = "G", DOCTOR-ID = "1500", TREATMENT-CODE = "T130")
Bộ này không được chèn vì nó bị trùng khóa chính với một trong các bộ tồn tại.
c. INSERT INTO patient-account (ACCOUNT = "890345255", AMOUNT-DUE = 256, DEPARTMENT = "G", DOCTOR-ID = "1500")
Bộ này không được chèn vì nó không có đủ số mục nhập tương ứng, chỉ có 4 giá trị, phải 5 mới đủ.
d. INSERT INTO patient-account (ACCOUNT = "5648901155", AMOUN-DUE = 256, DEPARTMENT = "G", DOCTOR-ID = "1500", TREATMENT-CODE = "T130")
Bộ này không được chèn vì có một tên cột bị ghi sai (AMOUN_DUE).
e. INSERT INTO patient-account (ACCOUNT = "721307804", AMOUNT-DUE = 256, DEPARTMENT = "G", DOCTOR-ID = "3500", TREATMENT-CODE = "T130")
Bộ này không được chèn vì giá trị của cột DOCTOR-ID không thuộc về miền giá trị của nó.

7.2. Xóa một bộ khỏi một bảng
Phép toán Xóa sẽ xóa một hay vài bộ chỉ định khỏi một quan hệ. Để xóa bộ nào cần chỉ rõ bộ đó bằng cách dùng các thuộc tính khóa chính. Cho quan hệ r có lược đồ R = {A_1, A_2, A_3, …, A_n}. Khi đó cú pháp của phép toán xóa như sau:
DELETE FROM *relation-name* WHERE *search-codition*
Điều kiện - tìm (search-condition) thường chỉ định các giá trị của các thuộc tính khóa để chỉ ra các bộ ta muốn xóa. Tuy nhiên, khi có nhiều bộ đồng thời bị xóa khi phát một lệnh xóa, điều kiện - tìm có thể được chỉ định bởi giá trị của thuộc tính bất kỳ. Trong tài liệu này ta sẽ biểu diễn điều kiện - tìm dưới dạng ($A_1 = v_1$, $A_2 = v_2$, $A_3 = v_3$, ... , $A_k = v_k$) trong đó các thuộc tính A_1, A_2, A_3, ... , A_k sẽ tạo nên khóa của quan hệ. Nếu các thuộc tính đã được ngầm hiểu thì ta chỉ cần liệt kê những giá trị v_i.
Phép toán Xóa cũng có thể thất bại do một trong những lý do sau:
- Bộ cần xóa không có trong quan hệ.
- Bộ cần xóa đang được tham chiếu bởi khóa ngoài của quan hệ khác. Trong trường hợp này, ta nói lệnh đã vi phạm ràng buộc toàn vẹn.
- Quan hệ không có trong cơ sở dữ liệu, hoặc tên cột không có trong bảng.
Việc xóa bộ cuối cùng của quan hệ cũng được chấp nhận, vì quan hệ rỗng là hợp lệ. Tất nhiên khi có sự vi phạm của một trong những điều trên thì một thông báo lỗi sẽ được trả về.

Ví dụ
Dùng lại quan hệ PATIENT_ACCOUNT như ví dụ trên, hãy cho biết có thể xóa những bộ nào dưới đây.
a. DELETE FROM patient-account WHERE account = "980543990"
Phép toán này sẽ thành công và nó sẽ xóa bộ có khóa chính là "980543990".
b. DELETE FROM patient-account WHERE account = 980543990
Phép toán này sẽ không thành công vì giá trị của thuộc tính ACCOUNT không tìm thấy. Chú ý là ta đã quy ước rằng dữ liệu kiểu chữ cần để trong cặp nháy kép. Còn trong trường hợp này RDBMS sẽ tìm một giá trị số chứ không phải là giá trị chữ. Một số hệ thống có thể sẽ tự động chuyển đổi qua lại giữa các giá trị chữ và số. Tuy nhiên chúng ta phải tự xem xét vấn đề đó trước khi nhờ đến máy.
c. DELETE FROM patient-account WHERE accounts = "980543990"
Phép toán này sẽ không thành công vì RDBMS không thể nhận ra thuộc tính accounts.

7.3. Cập nhật một bộ trong một bảng
Cập nhật một bộ nghĩa là thay đổi giá trị của một hay nhiều thuộc tính của nó. Bộ sẽ được cập nhật cần được chỉ rõ bằng cách dùng khóa của nó hay bằng cách dùng thuộc tính tìm kiếm. Cú pháp của phép toán này như sau:
UPDATE *relation-name* SET *column-name* = *new-value* WHERE *search-condition*

Bạn nên nhận thức rằng sự tồn tại của phép toán cập nhật chỉ là sự thuận tiện của RDBMS dành cho chúng ta mà thôi, bởi vì ta có thể tự làm điều tương tự bằng cách trước hết xóa bộ cũ, sau đó chèn bộ mới với các giá trị mới. Từ đó ta dễ dàng nhận thấy rằng những lý do làm cho phép toán cập nhật bị thất bại cũng sẽ là những lý do làm cho phép chèn và xóa thất bại.

Ví dụ
Giả sử ta cần cập nhật bảng PATIENT-ACCOUNT để thay đổi giá trị của thuộc tính AMOUNT-DUE từ $45.45 thành $200 cho bộ có giá trị của thuộc tính ACCOUNT là "804804308". Khi đó lệnh cập nhật sẽ là:
UPDATE patient-account SET amount-due = 200 WHERE account = "804804308"
Tác vụ này sẽ thành công vì không vi phạm điều gì.

8. Miền giá trị của các thuộc tính

Như ta đã biết, miền giá trị của các thuộc tính sẽ định nghĩa tính chất của các giá trị mà một cột trong bảng có thể chứa. Trong mọi RDBMS, miền giá trị của mọi thuộc tính sẽ dùng đến một khái niệm cần thiết là *kiểu dữ liệu* (data type). Một số tổ chức như ANSI (American National Standard Insitute) và ISO (International Standard Organization) đã định nghĩa một tập các kiểu dữ kiệu cơ bản. Các kiểu dữ liệu này mặc dù được hầu hết các RDBMS hỗ trợ, nhưng vẫn có những khác biệt nhỏ. Bảng sau sẽ trình bày một số kiểu dữ liệu SQL cơ bản trong một số RDBMS cụ thể.

SQL chuẩn	ORACLE	ACCESS	DB2
Character(n) n là số lượng ký tự.	*Char(n)* độ dài cố định tối đa 255 ký tự.	*Text* độ dài cố định tối đa 255 ký tự.	*Character(n).* tương tự *Char(n)* của ORACLE.
Character varying (n) n ký tự. Lưu trữ đúng số ký tự thực tế của nội dung.	*Varchar2(n)* độ dài thay đổi tối đa 2000 ký tự.	*Text* độ dài thay đổi tối đa 255 ký tự. *Memo* độ dài thay đổi tối đa 64000 ký tự.	*Varchar(n)* tương tự *Varchar2(n)* của ORACLE.
Float (p) p là tổng số chữ số.	*Number* phạm vi từ 1.0 x 10^{-130} đến 9.99...9 x 10^{125}, 38 chữ số có nghĩa.	*Single hay Double* tùy thuộc phạm vi giá trị dữ liệu.	*Float* tương tự *Number* của ORACLE.
Decimal(p, s) ít nhất p chữ số, với s định nghĩa bởi nhà cung cấp.	*Number(p,s)* p có phạm vi từ 1 đến 38 còn s từ -84 đến 127	*Integer hay Long Integer* tùy thuộc phạm vi giá trị dữ liệu.	*Integer* Tương tự *Number(38)* của ORACLE.

Các cột có các kiểu dữ liệu *character(n)* hay *character varying(n)*, trong đó n là số tối đa các ký tự có thể lưu trong cột, thường được dùng cho loại dữ liệu chứa chữ hay số không cần tính toán. Ví dụ như tên, địa chỉ, số điện thoại. Điểm khác biệt chính giữa các kiểu dữ liệu character(n) và character varying(n) là cách chúng lưu các chuỗi có ngắn hơn độ rộng cột hay không. Cụ thể là khi dữ liệu thực sự có số lượng ký tự ít hơn n, thì nếu dùng character(n), RDBMS sẽ tự thêm các khoảng trống vào dữ liệu đó cho đủ n ký tự, còn nếu dùng character varying(n) thì RDBMS không làm điều đó. Do đó nếu ta có ý định lưu dữ liệu không có độ rộng cố định, thì nên dùng kiểu character varying(n) cho hiệu quả. Dữ liệu chữ nói chung, nghĩa là bất kể có độ rộng cố định hay không, đều phân biệt chữ hoa với chữa thường. Điều đó có nghĩa là 'abc' sẽ khác 'aBc'. Ngoài ra khoảng trống cũng là ký tự, nên 'abc' đương nhiên khác với ' abc'.
Các cột có kiểu *float(n)*, trong đó n là số lượng chữ số, nói chung thường được dùng để biểu diễn các con số lớn hay các số có độ chính xác cao. Ví dụ như trong Oracle ta có thể dùng các kiểu dữ liệu float để biểu diễn các con số trong phạm vi từ 1.0 x 10^{-10} đến 10 x 10^{+10}.
Các cột có kiểu dữ liệu *decimal(p, s)* được dùng để biểu diễn các số có số chữ số phần thập phân cố định. Giá trị của p (precision) để cho biết các số đó cần tổng cộng tối đa bao nhiêu vị trí, còn s (scale) để cho biết riêng phần thập phân (phần sau dấu chấm) cần bao nhiêu vị trí. Trong trường hợp con số ta biểu diễn là số tròn thì phần thập phân sẽ được tự động gán giá trị không. Ví dụ như để biểu diễn giá trị 214.25 thì kiểu dữ liệu sẽ là decimal(5, 2). Trong những trường hợp này nếu là giá trị 352 thì chỉ cần kiểu decimal(3, 0).
Mặc dù không được trình bày trong bảng trên, nhưng có một kiểu dữ liệu rất hay được dùng và không thể thiếu là *DATE*. Kiểu dữ liệu này cho phép người dùng nhập các thông tin về ngày và giờ. Ngày thường có định dạng DD-MM-YY trong đó DD là phần ngày, MM là phần tháng và YY là phần năm.

Bài tập có lời giải

1. Hãy cho biết bậc và bản số của quan hệ SUPPLIER sau.

SUPPLIER

Supplier-Id	Part	Order-Number	Customer-Number	Quantity-Last-Order	Total
S1002	P1898E	18904-01-12-00	119078090	1000	12800
S8948	P3473	12443-10-11-99	232301248	25	1000
S1000	P2354A	98080-05-08-00	300234732	780	19500
S2993	P1898A	80808-78-08-31	459042354	2	500

Vì bảng có năm cột và bốn dòng nên có bậc 5 và bản số là 4.

2. Giả sử ta đã tạo quan hệ SUPPLIER của bài tập trên trong một RDBMS, vậy bạn sẽ chọn những kiểu dữ liệu gì cho các thuộc tính quan hệ trên?
Kiểu dữ liệu cho thuộc tính Supplier-Id nên là ký tự, vì trong đó có cả chữ cái và chữ số. Nếu thấy rằng năm ký tự là đủ thì ta sẽ chọn character(5) cho thuộc tính này. Nếu Supplier-Id thay đổi chiều dài, bạn cứ chọn varying(6), cho phép tăng đến 6 ký tự cũng đủ. Lý luận

tương tự cho các thuộc tính Part, Order-Number và Customer-Number. Chú ý là Customer_Number trông có vẻ như là số, nhưng thực ra thuộc tính này chẳng dùng để tính toán (giống như số chứng minh), nên chọn kiểu dữ liệu ký tự cho dễ.
Các thuộc tính Quantity_Last_Order và Total nên có kiểu dữ liệu số vì chúng cần tính toán. Cụ thể ở đây ta nên chọn kiểu dữ liệu số nguyên cho Quantity_Last_Order, còn Total nên là số thực.

3. Lâu nay ta dùng lẫn lộn các thuật ngữ bảng và quan hệ. Giả sử bảng dưới đây là một danh sách các học hàm học vị của các Giảng viên của một trường đại học, thì bảng đó có thể được xem như một quan hệ không?

INSTRUCTORS

Last-Name	First-Name	Academic-Degree	Department
Preito	Ray	B.S, M.S, Ph.D	Physics
Renton	Joan	B.A	Biology
Perez	Jose	B.S, M.S, Ph.D	PreMed
Bell	Mardre	B.S, M.S, Ph.D	PreMed

Không, nó không được xem như một quan hệ, vì các nội dung trong cột Academic-Degree không đơn trị, ngoại trừ của Renton. Tóm lại nó chỉ là bảng chứ không phải quan hệ.

4. Xét một quan hệ chỉ có một thuộc tính và có n (n > 0) bộ. Nếu ta chiếu (projection) lên thuộc tính khóa của quan hệ này thì bản số của quan hệ Chiếu kết quả sẽ như thế nào?
Vì có n bộ khác nhau trong quan hệ nên nhất định phải có n giá trị khóa khác nhau, mỗi khóa cho một bộ. Do đó, bản số của quan hệ kết quả sau khi chiếu cũng chính là bản số của quan hệ đầu, nghĩa là n.

5. Giả sử có hai quan hệ: EMPLOYEE và PROJECT. Bảng EMPLOYEE có thuộc tính Id. Giá trị của thuộc tính này nằm trong khoảng từ 1 đến 100. Bảng PROJECT cũng có thuộc tính Id này nằm trong khoảng từ 1 đến 100. Vì các thuộc tính này có miền giá trị như nhau nên bạn có thể nói rằng các thuộc tính này là như nhau không?
Không, ta không thể nói hai thuộc tính này là như nhau chỉ vì chúng có chung miền giá trị. Mặc dù chúng có thể giống nhau về giá trị nhưng có ý nghĩa khác nhau. Một thể hiện Id của nhân viên, giá trị duy nhất dùng nhận dạng từng nhân viên. Một thể hiện mã dự án, giá trị duy nhất dùng nhận dạng từng dự án.

6. Xét lại hai quan hệ trên. Nếu ta kết hai quan hệ đó lại thì RDBMS có cấm ta làm điều đó không?
Câu trả lời là không. RDBMS không bao giờ cấm ta làm điều đó. RDBMS chỉ kiểm tra xem hai vế của phép kết có cùng kiểu dữ liệu hay không, chứ nó không kiểm tra được có cùng ý nghĩa hay không. Mặc dù ta không bị cấm, nghĩa là kết được, nhưng kết quả sẽ chẳng có ý nghĩa gì.

7. Quan hệ STAFF của một cơ sở y tế như sau. Có thể chọn Last_Name làm khóa chính cho quan hệ này hay không?

STAFF

Last-Name	First-Name	Title	Salary	Department
Kyger	Wendy	X-Ray Tech	28000	Radiology
Dillard	Suzanne	X-Ray Tech	28000	Radiology
Shafiroff	Jean	Secretary	16000	General Ofice
Carson	Anna-Christie	Medical Assitant	23000	General Medicine
Sutton	Melissa	X-Ray Tech	28000	Radiology
Renton	Hayley	Receptionist	14000	General Office

Không, Last-Name không thể được chọn làm khóa chính cho quan hệ này mặc dù thể hiện của quan hệ này không cho thấy sự trùng lắp ở Last-Name. Bởi tuy trong thể hiện này không có sự giống nhau ở Last-Name, nhưng điều đó không thể bảo đảm rằng những Last-Name khác nếu thêm vào sẽ không trùng.

8. Vì các quan hệ được định nghĩa là các tập hợp nên sẽ không có chuyện có các bộ trùng. Vậy tại sao ta không dùng toàn bộ tập hợp các thuộc tính của quan hệ làm khóa chính của quan hệ?
Đúng là trong mọi quan hệ, khóa chính luôn là tập con của tập các thuộc tính của quan hệ. Nghĩa là toàn bộ các thuộc tính của quan hệ là siêu khóa của quan hệ đó. Hơn nữa, sẽ không có chuyện có hai bộ giống hệt nhau ở mọi thuộc tính của quan hệ. Tuy nhiên phải nhớ rằng khóa được dùng như cơ chế chủ yếu để nhận diện và truy xuất các bộ trong quan hệ. Vì thế, ta thường dùng các khóa chính có số lượng thuộc tính nhỏ nhất. Ví dụ như trong quan hệ EMPLOYEE, nếu biết mã nhân viên ta sẽ biết được lương của người đó. Còn nếu phải biết tất cả các thông tin về người đó mới có thể lấy được thông tin của người đó thì lấy làm gì nữa. Tóm lại khóa chính luôn là tập hợp con nhỏ nhất của tập các thuộc tính mà vẫn có thể xác định được mọi bộ trong quan hệ.

9. Giả sử ta chuẩn bị nhập dữ liệu cho quan hệ EMPLOYEE có lược đồ như sau. Biết rằng Manager-Id là khóa ngoại tham chiếu đến thuộc tính Id của quan hệ. Ý nghĩa của khóa ngoại đó là: để là một người quản lý (manager) thì bản thân bạn cũng phải là nhân viên (employee). Nếu các bộ (a), (b) và (c) dưới đây là các bộ đầu tiên sẽ được nhập vào bảng thì có vấn đề gì không?

EMPLOYEE

Id	Last-Name	Department	Salary	Manager-Id

a. ("190034890", "Gomez", "Lila", "Human Resources", 25000, "355899234")
b. ("123434643", "Tuset", "Agustin", "Marketing", 35000, NULL)
c. (NULL, "Tuset", "Maria Pia", "Sales", 40000, NULL)

Nếu bộ (a) được nhập vào đầu tiên thì RDBMS sẽ báo lỗi vi phạm khóa ngoại, vì Manager-Id 35589234 hiện chưa có trong cột Id. Vì Manager-Id là khóa ngoại tham chiếu đến Id nên mọi giá trị khác NULL muốn xuất hiện trong cột đó đều phải xuất hiện trong cột Id trước.
Bộ (b) có thể được nhập vào bảng mà không gặp vấn đề gì vì giá trị NULL cho Manager-Id là chấp nhận được. NULL trong trường hợp này có nghĩa là nhân viên này chưa có người quản lý.
Bộ (c) không thể được nhập vào vì giá trị khóa chính của nó mang giá trị NULL. Giá trị cho khóa ngoại thì NULL được nhưng khóa chính thì không bao giờ.

10. Chứng minh phép Chọn có tính giao hoán, nghĩa là nếu có quan hệ r có hai thuộc tính A và B, và nếu a $\in$ Dom(A) và b $\in$ Dom(B) thì:
$\sigma_{A=a}(\sigma_{B=b}(r)) = \sigma_{B=b}(\sigma_{A=a}(r))$
Điều trên có thể được diễn dịch lại thành lời, rằng nếu ta thực hiện phép Chọn tất cả các bộ trong quan hệ r có các giá trị tại cột A bằng a rồi từ quan hệ kết quả đó, ta thực hiện phép Chọn tất cả các bộ có các giá trị tại cột B bằng b, thì kết quả cuối cùng cũng không khác gì nếu từ đầu ta thực hiện phép Chọn tất cả các bộ trong quan hệ r có giá trị tại cột B bằng b rồi từ quan hệ kết quả đó, ta thực hiện phép Chọn tất cả bộ có các giá trị tại cột A bằng a.
Để chứng minh rằng hai tập A và B là bằng nhau, ta cần chứng minh rằng A $\subset$ B và B $\subset$ A. Vì quan hệ Chọn cũng là một tập hợp, để chứng minh cho phát biểu trên, ta cần chứng minh:
(a) $\sigma_{A=a}(\sigma_{B=b}(r)) \subset \sigma_{B=b}(\sigma_{A=a}(r))$
và
(b) $\sigma_{B=b}(\sigma_{A=a}(r)) \subset \sigma_{A=a}(\sigma_{B=b}(r))$
Để chứng minh (a), ta cần lấy một bộ t bất kỳ của $\sigma_{A=a}(\sigma_{B=b}(r))$ và chứng minh rằng bộ này cũng là một bộ trong $\sigma_{B=b}(\sigma_{A=a}(r))$. Ta ó thể chứng minh như sau:
Gọi t $\in\sigma_{A=a}(\sigma_{B=b}(r))$, theo định nghĩa của phép Chọn, t là một phần tử của quan hệ $\sigma_{B=b}(r)$ và t(A) = a. Áp dụng định nghĩa của phép Chọn lần nữa cho $\sigma_{B=b}(r)$, ta có: t $\in$ r $\wedge$ t(A) = a $\wedge$ t(B) = b. Nhóm lại cho thuận tiện, sử dụng thực tế là toán tử $\wedge$ có tính hoán vị và áp dụng định nghĩa của phép Chọn ta có: t $\in$ $\sigma_{A=a}(r)$ $\wedge$ t(B) = b. Áp dụng định nghĩa của phép Chọn lần nữa, ta có: t $\in$ $\sigma_{B=b}(\sigma_{A=a}(r))$.
Vì ta lấy bất kỳ bộ t $\in$ $\sigma_{A=a}(\sigma_{B=b}(r))$ và chứng minh rằng nó cũng là một bộ của $\sigma_{B=b}(\sigma_{A=a}(r))$, nên (a) được chứng minh.
Chứng minh tương tự cho (b).

11. Cho quan hệ EMPLOYEE dưới đây, hãy tìm Last-Name (họ), Department (phòng ban) và Salary (lương) của mọi nhân viên.

EMPLOYEE

SSN	Last-Name	First-Name	Department	Sex	Salary
122904934	Gomez	Luis	General Office	M	40000
789009223	Nadeau	Frank	Engineering	M	57000
802341175	Knupp	Carroll	Security	M	23000
123425434	Nichols	Lois	Engineering	F	30000
209354532	Nichols	James	Maintenance	M	30000
894393344	Nadeau	Peggy	General Office	F	40000
823123453	Smith	Donald	Maintenance	M	35000
634890303	Smith	Faith	Maintenance	F	34000

Để có các thông tin cần thiết, ta cần tìm phép Chiếu quan hệ trên các thuộc tính Last-Name, Department và Salary. Kết quả như sau:

$\pi_{\text{Last-Name, Department, Salary}}$(EMPLOYEE)

Last-Name	Department	Salary
Gomez	General Office	40000
Nadeau	Engineering	57000
Knupp	Security	23000
Nichols	Engineering	30000
Nichols	Maintenance	30000
Nadeau	General Office	40000
Smith	Maintenance	35000
Smith	Maintenance	34000

Chú ý là các bộ (Nichols, Engineering, 30000) và (Nichols, Maintenance, 30000) là khác nhau vì khác phòng ban. Tương tự, các bộ (Smith, Maintenance, 35000) và (Smith, Maintenance, 34000) là khác nhau ở lương.

12. Ngoài bảng EMPLOYEE ở bài tập trên, giả sử có thêm bảng DEPARTMENT dưới đây. Hãy viết truy vấn cho phép tìm ngân sách (budget) của phòng ban nơi Luis Gomez làm việc.

DEPARTMENT

Id	Name	Building	Budget
GOF	General Office	North Tower	2000000
ENG	Engineering	North Tower	3000000
SEC	Security	South Tower	1000000
MAI	Maintenance	East Tower	6000000

Để trả lời câu hỏi trên, trước hết ta phải có đủ các thông tin về nhân viên đó. Thực hiện phép Chọn $\sigma_{\text{Last-Name="Gomez"}}$(EMPLOYEE). Kết quả sẽ như sau:

$\sigma_{\text{Last-Name="Gomez"}}$(EMPLOYEE)

Last-Name	Department	Salary
Gomez	General Office	40000

Chiếu quan hệ này trên thuộc tính tên văn phòng (Name), ta sẽ có kết quả là:

$\pi_{\text{Department}}(\sigma_{\text{Last-Name="Gomez"}}$(EMPLOYEE))

Department
General Office

Tiếp theo, ta sẽ dùng kết quả trả về để thực hiện phép Chọn trên quan hệ DEPARTMENT. Nghĩa là lấy thông tin về bộ có tên là "General Office": SELECTION $_{\text{Name}} = \pi_{\text{Department}}(\sigma_{\text{Last-Name="Gomez"}}$(EMPLOYEE))(DEPARTMENT)
Kết quả sẽ là:

SELECTION $_{\text{Name}} = \pi_{\text{Department}}(\sigma_{\text{Last-Name="Gomez"}}$(EMPLOYEE))(DEPARTMENT)

Name	Building	Budget
General Office	North Tower	2000000

Cuối cùng, chiếu lên thuộc tính Budget để có kết quả:

π_{Budget}(SELECTION $_{\text{Name}} = \pi_{\text{Department}}(\sigma_{\text{Last-Name="Gomez"}}$(EMPLOYEE))(DEPARTMENT))

Budget
2000000

13. Hãy viết lệnh SQL để hiển thị họ (Last-Name) và tên (First-Name) của từng nhân viên, tên phòng ban(Department) cũng như tên tòa nhà (Building) nơi phòng ban của các nhân viên đó tọa lạc.
Để trả lời cho câu hỏi này trước hết ta cần kết hai quan hệ EMPLOYEE và DEPARTMENT dựa trên thuộc tính chung là Department và Name. Sau đó ta sẽ chiếu trên kết quả có được lên những thuộc tính cần có.
Khi kết hai bảng ta sẽ có kết quả như sau:

EMPLOYEE Join DEPARTMENT

SSN	Last-Name	First-Name	Sex	Salary	Department	Building	Budget
122904934	Gomez	Luis	M	40000	General Office	North Tower	2000000
789009223	Nadeau	Frank	M	57000	Engineering	North Tower	3000000
802341175	Knupp	Carroll	M	23000	Security	South Tower	1000000
123425434	Nichols	Lois	F	30000	Engineering	North Tower	3000000
209354532	Nichols	James	M	30000	Maintenance	East Tower	6000000
894393344	Nadeau	Peggy	F	40000	General Office	North Tower	2000000
823123453	Smith	Donald	M	35000	Maintenance	East Tower	6000000
634890303	Smith	Faith	F	34000	Maintenance	East Tower	6000000

Để có kết quả cuối cùng ta chiếu lên kết quả trên và có câu trả lời như sau:

$\pi_{\text{first-Name, Last-Name, Department, Building}}$(EMPLOYEE Join DEPARTMENT)

First-Name	Last-Name	Department	Building
Luis	Gomez	General Office	North Tower
Frank	Nadeau	Engineering	North Tower
Carroll	Knupp	Security	South Tower
Lois	Nichols	Engineering	North Tower

James	Nichols	Maintenance	East Tower
Peggy	Nadeau	General Office	North Tower
Donald	Smith	Maintenance	East Tower
Faith	Smith	Maintenance	East Tower

14. Phép toán Kết cho phép ta hiển thị những dòng chỉ định từ hai bảng có liên quan. Trong các bài tập trên, ta đã hiển thị những dòng có trong từ một bảng và "so trùng" trong bảng khác. Tuy nhiên, ta muốn các dòng từ một bảng xuất hiện trong bảng Kết khi những dòng đó không so trùng trong bảng khác. Khi đó, *phép Kết ngoại* (Outer Join) có thể giúp chúng ta giải quyết vấn đề.
Kết quả của phép Kết ngoại là kết quả của phép Kết bằng cộng thêm những dòng ở bảng thứ nhất thỏa điều kiện kết nhưng không có những dòng tương ứng ở bảng thứ hai.
Có hai dạng Kết ngoại: *Kết ngoại trái* (Left OuterJoin) và *Kết ngoại phải* (Right Outer Join), ta sẽ gọi là lr Ljoin rr và lrRjoin rr, trong đó rl và rr là tên các quan hệ left ralation và right relation tương ứng.
Cho hai quan hệ r(R) và s(S) có chung thuộc tính X, khi đó ta định nghĩa r Ljoin s là quan hệ chứa tất cả các bộ là kết quả của rJoin s cộng thêm các dòng thỏa điều kiện ở quan hệ r, là những dòng không có các dòng tương ứng ở s. Tương tự ta định nghĩa r Rjoin s là quan hệ chứa tất cả các bộ là kết quả của r Join s cộng thêm các dòng thỏa điều kiện ở quan hệ s, là những dòng không có các dòng tương ứng ở r. Trong trường hợp Kết trái sẽ có thể có những giá trị NULL xuất hiện tại những cột lẽ ra xuất hiện những giá trị của s. Tương tự, trường hợp Kết phải sẽ có thể có những giá trị NULL xuất hiện tại những cột lẽ ra xuất hiện những giá trị của r.
Cuối cùng, một tác vụ gọi là *Kết hai bên* (Full OuterJoin) có thể được xem như phép hội (UNION) của Kết trái và Kết phải. (Hiếm có RDBMS nào hỗ trợ tác vụ này, vì nó là dư thừa khi có thể Hội của Kết trái và Kết phải như đã nói).
Xét các bảng SalesAssociate và Customer dưới đây, hãy hiển thị tên và Id của các nhân viên bán hàng cùng với tất cả các tên khách hàng của các nhân viên đó.

SalesAssociate

Last_Name	Id
McGee	11
Gilson	12
Segui	13
Nguyen	14
Dumas	15

Customer

SalesAssociateId	Name
11	WomanSport
11	Baseball for All
12	Sport Equipments
13	Sam’s Sporting Goods
14	The Little Sporting Shop
14	Equipo Deportivo
15	Sportique
	UniSports

Trong trường hợp này các thuộc tính chung là Id (của SalesAssociate) và SalesAssociateId (của Customer). Kết quả Rjoin của SalesAssociate và Customer được trình bày dưới đây. Chú ý là khách hàng UniSports cũng được xuất hiện mặc dù không có SalesAssociateId tương ứng.

SalesAssociate Rjoin Customer

Last_Name	SalesAssociateId	Name
McGee	11	WomanSport
McGee	11	Baseball for All
Gilson	12	Sport Equipments
Segui	13	Sam’s Sporting Goods
Nguyen	14	The Little Sporting Shop
Nguyen	14	Equipo Deportivo
Dumas	15	Sportique
		UniSports

15. Có một phép toán khác cũng thường được dùng để trả lời các truy vấn trong cơ sở dữ liệu là phép Chia (DIVISION). Phép Chia cho phép ta trả lời những truy vấn kiểu như "có những bộ nào trong quan hệ này có mọi giá trị xuất hiện ở một cột nào đó trong quan hệ kia?" Phép Chia có thể được định nghĩa như sau: Cho hai quan hệ r(R) và s(S) với $A \subset R$ và $B \subset S$, trong đó A và B khả hợp. Phép Chia được ký hiệu r[A ÷ B]s trong đó r là *quan hệ bị chia* (devidend relation), còn s là *quan hệ chia* (divisor relation). Kết quả của phép toán này có thể đạt được thông qua việc dùng thuật toán sau:

(1) Xét quan hệ bị chia như một quan hệ hai phần: phần chứa (các) thuộc tính bị chia A và phần còn lại C = R – A.
(2) Tìm chiếu của quan hệ bị chia lên (các) thuộc tính còn lại, tức $\pi_C(R)$. Sau đó với mỗi bộ trong kết quả của phép Chiếu ta thực hiện các bước sau:
   a. Tạo một bảng có tất cả các bộ của quan hệ bị chia có chứa bộ được chọn ở bước trên. Lược đồ của bảng này phải giống với lược đồ của quan hệ bị chia.
   b. Chiếu kết quả mới nhận được lên (các) thuộc tính bị chia. Gọi đây là tập hợp T_i.
(3) Chiếu quan hệ chia lên (các) thuộc tính chia.
(4) Chọn tập hợp T_i chứa tất cả các bộ có từ bước 3.

Sử dụng thuật toán trên với hai bảng BUYER (khách hàng) và PRODUCT (sản phẩm) dưới đây, hãy tìm khách hàng mua mọi sản phẩm. Để trả lời câu hỏi ta cần thực hiện phép Chia BUYER [ITEM ÷ CODE] PRODUCT. Chú ý là các thuộc tính ITEM va CODE phải khả hợp, trong đó ITEM là thuộc tính bị chia còn CODE là thuộc tính chia.

BUYER

Name	Item
Smith	A
Jones	B
Adams	A
Smith	B
Jones	A
Smith	C

PRODUCT

CODE	COST	PRICE
A	5	4
B	4	4
C	6	9

Áp dụng thuật toán cho hai bảng ta có:
(1) Vì quan hệ bị chia (BUYER) chỉ có hai thuộc tính, nó có thể dễ dàng chia làm hai nhóm: thuộc tính bị chia (ITEM) và thuộc tính còn lại (NAME).
(2) Khi chiếu quan hệ BUYER lên thuộc tính còn lại ta có:

π_{NAME}(BUYER)

Name
Smith
Jones
Adams

Chọn bộ Smith ta sẽ có bảng mới T_1 có các bộ lấy từ quan hệ bị chia (BUYER) chứa bộ Smith như sau:

Table T_1

Name	Item
Smith	A
Smith	B
Smith	C

Chiếu bảng này lên thuộc tính bị chia và gọi kết quả là tập hợp T_1 ta có $T_1 = \pi_{ITEM}(T_1) = \{(A), (B), (C)\}$. Chú ý, đây là tập hợp các bộ.
Chọn bộ Jones và tạo bảng mới T_2 có các bộ lấy từ quan hệ bị chia (BUYER) chứa bộ Jones như sau:

Table T_2

Name	Item
Jones	B
Jones	A

Chiếu bảng này lên thuộc tính bị chia và gọi kết quả là tập hợp T_2 ta có $T_2 = \pi_{ITEM}(T_2) = \{(A), (B)\}$.
Chọn bộ Adams ta sẽ có bảng mới T_3 có các bộ lấy từ quan hệ bị chia (BUYER) chứa bộ Adams như sau:

Table T_3

Name	Item
Adams	A

Chiếu bảng này lên thuộc tính bị chia và gọi kết quả là tập hợp T_3 ta có $T_3 = \pi_{ITEM}(T_3) = \{(A)\}$.
(3) Thực hiện phép chiếu của quan hệ chia (PRODUCT) lên thuộc tính chia (CODE), nghĩa là tìm π_{CODE}(PRODUCT) ta có:

π_{CODE}(PRODUCT)

CODE
A
B
C

(4) Rõ ràng T_1 là tập hợp các bộ duy nhất chứa ba bộ khác nhau trong bước vừa rồi.
Vì T_1 là có được khi chọn bộ Smith nên Smith chính là khách hàng duy nhất mua mọi sản phẩm.

16. Cho các bảng dưới đây, hãy tìm tên các người phụ thuộc (dependent) của tất cả các kỹ sư (Engineer).

EMPLOYEE

SSN	Last-Name	Title	Department	No-of-Dependents
123456789	Hopkins	Engineer	Construction	2
987654321	Brando	Architect	Design	1
570345228	Kimball	Engineer	Construction	3
923458004	Housden	Lawyer	Main Office	1

DEPENDENT

EMPSSN	DEPSSN	Last-Name	First-Name	DOB
123456789	792323653	Hopkins	Gerald	11/12/90
123456789	970254356	Hopkins	Brenda	05/12/92
987654321	805298992	Brando	Greta	08/12/85
570345228	234497909	Kimball	Silvia	02/02/87
570345228	807325224	Kimball	George	04/08/90
570345228	345098772	Kimball	Alice	05/23/92
923458004	943003993	Housden	David	07/12/81

Trước hết ta cần tìm tất cả các kỹ sư trong bảng EMPLOYEE, nghĩa là ta cần thực hiện phép $\sigma_{Title=Engineer}$(EMPLOYEE), kết quả như sau:

$\sigma_{Title=Engineer}$(EMPLOYEE)

SSN	Last-Name	Title	Department	No-of-Dependents
123456789	Hopkins	Engineer	Construction	2
570345228	Kimball	Engineer	Construction	3

Từ quan hệ này ta sẽ lấy SSN, Last-Name và Title của mọi kỹ sư. Nghĩa là ta cần chiếu quan hệ trên lên ba thuộc tính đó, ta có:

$\pi_{SSN,Last\text{-}Name,Title}(\sigma_{Title=Engineer}$(EMPLOYEE))

SSN	Last-Name	Title
123456789	Hopkins	Engineer
570345228	Kimball	Engineer

Ta sẽ đặt tên cho bảng $\pi_{SSN,Last\text{-}Name,Title}(\sigma_{Title=Engineer}$(EMPLOYEE)) là ENG viết tắt của Engineers.
Bây giờ ta có thể tính tích Descartes của quan hệ ENG này với bảng DEPPENDENT. Tích Descartes có kết quả dưới đây. Một số tên thuộc tính đã được trình bày rõ ràng để tránh nhầm lẫn.

ENG ⊗ DEPENDENT

SSN	ENG-Last-Name	Title	EMPSSN	DEPSSN	DEP-Last-Name	DEP-First-Name	DOB
123456789	Hopkins	Engineer	123456789	792323653	Hopkins	Gerald	11/12/90
123456789	Hopkins	Engineer	123456789	970254356	Hopkins	Brenda	05/12/92
123456789	Hopkins	Engineer	987654321	805298992	Brando	Greta	08/12/85
123456789	Hopkins	Engineer	570345228	234497909	Kimball	Silvia	02/02/87
123456789	Hopkins	Engineer	570345228	807325224	Kimball	George	04/08/90
123456789	Hopkins	Engineer	570345228	345098772	Kimball	Alice	05/23/92
123456789	Hopkins	Engineer	923458004	943003993	Housden	David	07/12/81
570345228	Kimball	Engineer	123456789	792323653	Hopkins	Gerald	11/12/90
570345228	Kimball	Engineer	123456789	970254356	Hopkins	Brenda	05/12/92
570345228	Kimball	Engineer	987654321	805298992	Brando	Greta	08/12/85

570345228	Kimball	Engineer	570345228	234497909	Kimball	Silvia	02/02/87
570345228	Kimball	Engineer	570345228	807325224	Kimball	George	04/08/90
570345228	Kimball	Engineer	570345228	345098772	Kimball	Alice	05/23/92
570345228	Kimball	Engineer	923458004	943003993	Housden	David	07/12/81

Tích Descartes tự nó không có ý nghĩa, nhưng có thể lấy từ nó một số thông tin đáng giá cho Engineers. Để làm điều này ta sẽ chọn tất cả các bộ có SSN = EMPSSN. Kết quả của tác vụ này như sau:

$\sigma_{SSN=EMPSSN}$(ENG ⊗ DEPENDENT)

SSN	ENG-Last-Name	Title	EMPSSN	DEPSSN	DEP-Last-Name	DEP-First-Name	DOB
123456789	Hopkins	Engineer	123456789	792323653	Hopkins	Gerald	11/12/90
123456789	Hopkins	Engineer	123456789	970254356	Hopkins	Brenda	05/12/92
570345228	Kimball	Engineer	570345228	234497909	Kimball	Silvia	02/02/87
570345228	Kimball	Engineer	570345228	807325224	Kimball	George	04/08/90
570345228	Kimball	Engineer	570345228	345098772	Kimball	Alice	05/23/92

Và kết quả cuối cùng lấy về bằng cách thực hiện phép chiếu trên các thuộc tính SSN, ENG-Last-Name và DEP-First-Name như sau:

$\pi_{SSN,ENG-Last-Name,DEP-First-Name}(\sigma_{SSN = EMPSSN}$(ENG ⊗ DEPPENDENT))

SSN	ENG-Last-Name	DEP-First-Name
123456789	Hopkins	Gerald
123456789	Hopkins	Brenda
570345228	Kimball	Silvia
570345228	Kimball	George
570345228	Kimball	Alice

17. Phép kết (Theta) giữa hai quan hệ r(R) và s(S), thường được ký hiệu là r θ s là kết hợp phép Chọn và tích Descartes. Dùng các bảng dưới đây, tìm các mặt hàng (item) mà mỗi khách hàng có thể mua được dựa trên số tiền họ có.

Product

Id	Name	Price
00123	Basketball	125
00343	Bike	550
00489	Golf Clubs	980

Customer

Id	Credit
C3920	1500
C563	350
C332	200

Diễn đạt cách dùng phép kết Theta như sau:
Tính tích Descartes của hai quan hệ: Customer ⊗ Product. Trong kết quả ta điều chỉnh tên một số thuộc tính để tránh nhầm lẫn.

Customer ⊗ Product

CustomerId	Credit	ProductId	Name	Price
C3920	1500	00123	Basketball	125
C3920	1500	00343	Bike	550
C3920	1500	00489	Golf Clubs	980
C563	350	00123	Basketball	125
C563	350	00343	Bike	550
C563	350	00489	Golf Clubs	980
C332	200	00123	Basketball	125
C332	200	00343	Bike	550
C332	200	00489	Golf Clubs	980

Dùng kết quả có được ta thực hiện phép Chọn với điều kiện chọn là Credit ≥ Price, nghĩa là $\sigma_{Credit \geq Price}$(Customer ⊗ Product)

Kết quả ta sẽ có:

$\sigma_{Credit \geq Price}$(Customer ⊗ Product)

CustomerId	Credit	ProductId	Name	Price
C3920	1500	00123	Basketball	125
C3920	1500	00343	Bike	550
C3920	1500	00489	Golf Clubs	980
C563	350	00123	Basketball	125
C332	200	00123	Basketball	125

Cuối cùng ta thực hiện phép Chiếu kết quả trên lên các thuộc tính CustomerId, ProductId, Name và Price để có thông tin cần thiết:

$\pi_{CustomerId,ProductId, Name,Price}(\sigma_{Credit \geq Price}$(Customer ⊗ Product))

CustomerId	ProductId	Name	Price
C3920	00123	Basketball	125
C3920	00343	Bike	550
C3920	00489	Golf Clubs	980
C563	00123	Basketball	125
C332	00123	Basketball	125

18. Dùng phép Kết như thế nào để diễn đạt phép Chọn?
Để diễn đạt phép Chọn bằng cách dùng phép Kết, ta làm như sau: Cho quan hệ r(R), có thuộc tính A của r và a là một giá trị thuộc miền giá trị dom(A), giả sử ta muốn có kết quả của phép $\sigma_{A=a}(r)$. Để có kết quả của phép Chọn này ta có thể tạo một quan hệ s mới chỉ có một thuộc tính và một bộ. Khi đó (đương nhiên) A là thuộc tính duy nhất và a là bộ duy nhất. Quan hệ mới như sau:

$\sigma_{A=a}(r)$

A
a

Sau đó thực hiện phép Kết bằng giữa hai quan hệ r và $\sigma_{A=a}(r)$ trên thuộc tính chung A, phép Kết này tương đương $\sigma_{A=a}(r)$.

19. Cho các quan hệ DOCTORS (bác sĩ) và NURSES (y tá) , hãy tìm Last-Name (tên) và Supervisor (người quản lý) của các nhân viên y tế ở phòng Cardiology. Cần giả sử thêm gì để kết quả có thể được trình bày trong một bảng đơn?

DOCTORS

Id	Last-Name	Department	Supervisor
120534533	Silver	Cardiology	Dr. Jones
380237302	Roswell	Radiology	Dr. Stewart
293982983	Hartman	Oncology	Dr. Smith
727873233	Stanley	Cardiology	Dr. Sing
982391179	Bartley	Pediatrics	Dr. Spesser
425370983	Jones	Rehab	Dr. Sams

NURSES

Id	Last-Name	Department	Supervisor
930272434	Clinton	Pediatrics	Mrs. Alexander
892390313	Alvarez	Cardiology	Ms. Hussan
321231234	Lewis	Rehab	Mr. Hanson
930111341	Felton	Oncology	Mr. Lenkerd
823907592	Traubaugh	Pediatrics	Mr. Sullivan
392983999	Rissler	Cardiology	Mr. Sanchez

Ta cần phải làm như sau để trả lời câu hỏi trên:
Trước hết ta sẽ tìm tên của các bác sĩ và y tá làm việc ở bộ phận Cardiology.
Phép toán và kết quả của yêu cầu này như sau:

$\sigma_{DEPARTMENT=CARDIOLOGY}$(DOCTORS)

Id	Last-Name	Department	Supervisor
120534533	Silver	Cardiology	Dr. Jones
727873233	Stanley	Cardiology	Dr. Sing

$\sigma_{DEPARTMENT=CARDIOLOGY}$(NURSES)

Id	Last-Name	Department	Supervisor
892390313	Alvarez	Cardiology	Ms. Hussan
392983999	Rissler	Cardiology	Mr. Sanchez

Từ hai quan hệ trên ta sẽ lấy Last-Name và Supervisor của cả bác sĩ và y tá. Phép toán và kết quả như sau:

$\pi_{Last\text{-}Name,Supervisor}(\sigma_{DEPARTMENT=CARDIOLOGY}$(DOCTORS))

Last-Name	Supervisor
Silver	Dr. Jones
Stanley	Dr. Sing

$\pi_{Last\text{-}Name,Supervisor}(\sigma_{DEPARTMENT=CARDIOLOGY}$(NURSES))

Last-Name	Supervisor
Alvarez	Ms. Hussan
Rissler	Mr. Sanchez

Để hai kết quả trên có thể được trình bày trong một bảng ta sẽ thực hiện phép Hội hai quan hệ đó. Để có thể Hội được, thì hai bảng trên phải cùng thuộc tính và các thuộc tính phải có cùng kiểu dữ liệu (tính khả hợp).

Last-Name	Supervisor
Silver	Dr. Jones
Stanley	Dr. Sing
Alvarez	Ms. Hussan
Rissler	Mr. Sanchez

20. Cho hai quan hệ r(R) và s(S) và các phép toán sau, vậy bản số của kết quả sẽ như thế nào?
(a) $s \cup r$
(b) $r \cap s$
(c) r - s
(d) $r \otimes s$
(e) $\sigma_{A=a}(r)$ với $a \in Dom(A)$
(f) $\pi_A(r)$ nếu A là một thuộc tính trong lược đồ của r

(a) Bản số của phép Hội tùy thuộc vào số lượng các bộ giống nhau hiện có trong các quan hệ r và s. Nếu hai quan hệ không có bộ nào giống nhau (disjoint), thì bản số của phép Hội sẽ là tổng của các bản số của các quan hệ. Nếu tất cả các bộ của r đều có trong s thì bản số của phép Hội chính là bản số của s. Trường hợp hai quan hệ có chung nhau một số bộ thì bản số của phép Hội chỉ có thể nhỏ hơn hay bằng tổng bản số của hai quan hệ.

(b) Bản số của phép Giao có thể bằng không nếu hai quan hệ không có bộ chung nào. Nếu quan hệ này có các bộ đều được chứa trong quan hệ kia, thì bản số của phép Giao là số nhỏ hơn trong hai bản số của hai quan hệ. Trường hợp hai quan hệ có chung nhau một số bộ thì bản số của phép Giao chỉ có thể nhỏ hơn hay bằng số nhỏ hơn trong bản số của hai quan hệ.

(c) Nếu kết quả của phép Trừ r - s không trả về rỗng, thì bản số của phép Trừ chỉ có thể nhỏ hơn hay bằng bản số của r. Nếu tất cả các bộ của s đều được chứa trong r thì bản số của phép Trừ chính là hiệu của hai bản số của hai quan hệ kia.

(d) Bản số của tích Descartes của hai quan hệ là tích của hai bản số của hai quan hệ.

(e) Bản số của phép Chọn trên e sẽ nhỏ hơn hay bằng bản số của r.

(f) Bản số của phép Chiếu trên r sẽ nhỏ hơn bản số của r nếu kết quả có trùng lắp trong phép Chiếu. Còn nếu kết quả không trả về hai bộ nào trùng nhau thì bản số của kết quả phép Chiếu cũng chính là bản số của r.

Ngôn ngữ SQL

1. Mở đầu

SQL (Structured Query Language) là ngôn ngữ lệnh mà bạn sử dụng để tương tác với cơ sở dữ liệu. SQL cung cấp cho bạn một cách đơn giản và hiệu quả để đọc/ghi dữ liệu từ/đến cơ sở dữ liệu.
SQL được sử dụng theo hai cách khác nhau: nhúng hoặc tương tác.
- Bạn nhúng các phát biểu SQL trong một chương trình được tạo ra trong một nền tảng lập trình khác (như Java).
- Bạn nhập các phát triển SQL bằng cách sử dụng bàn phím của bạn tại dấu nhắc lệnh SQL hoặc trong một phần mềm có giao diện đồ họa (như SQL Developer), để có được thông tin bạn muốn.

Thông thường, ngôn ngữ SQL được chia thành bốn loại lệnh sau đây:
- Ngôn ngữ định nghĩa dữ liệu (DDL)
  Các lệnh cho phép bạn tạo, sửa và xóa các đối tượng của cơ sở dữ liệu. Các đối tượng cơ sở dữ liệu chủ yếu bao gồm các bảng (table), khung nhìn (view), các thủ tục (procedure), người dùng (user), các trigger, ... Hầu như tất cả các lệnh định nghĩa dữ liệu SQL đều bắt đầu với một trong ba từ khóa sau:
  CREATE thêm các đối tượng cơ sở dữ liệu mới như bảng, người dùng, ...
  ALTER sửa đổi cấu trúc của một đối tượng cơ sở dữ liệu hiện có.
  DROP xóa một đối tượng cơ sở dữ liệu.
- Ngôn ngữ thao tác dữ liệu (DML)
  Các lệnh cho phép bạn thay đổi nội dung cơ sở dữ liệu. Với mục đích này, SQL cung cấp ba lệnh thao tác cơ bản:
  INSERT chèn thêm dòng mới (hồ sơ) vào một bảng.
  UPDATE cập nhật, sửa đổi các giá trị cột của các dòng hiện có.
  DELETE xóa các dòng từ một bảng.
- Ngôn ngữ kiểm soát dữ liệu (DCL)
  Các lệnh liệu được sử dụng để kiểm soát truy cập vào các đối tượng cơ sở dữ liệu khác nhau (bảng, khung nhìn, ...). Các lệnh kiểm soát dữ liệu bao gồm GRANT và REVOKE.
- Ngôn ngữ truy vấn dữ liệu (DQL)
  Chỉ có một lệnh, nhưng là một trong những lệnh quan trọng nhất: SELECT. Đây là lệnh duy nhất trong SQL được sử dụng để lấy (truy vấn) dữ liệu từ cơ sở dữ liệu.

2. Cài đặt

Để thực hiện tất cả các bài tập được cung cấp trong phần này, bạn phải tải về các phần mềm miễn phí sau:
- Oracle Database Express Edition 11g (Oracle Database XE)
- Oracle SQL Developer

Bạn cần tạo một tài khoản Oracle miễn phí để được cho phép tải xuống.

2.1. Oracle Database Express Edition

Oracle Database XE là cơ sở dữ liệu có dung lượng nhỏ. Nó được phát triển, triển khai và phân phối miễn phí;
- Tải về từ địa chỉ: http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html
- Tick chọn Accept License Agreement.
- Chọn Oracle Database Express Edition 11g Release 2 với phiên bản tương thích hệ nền của bạn.
- Nhập tên đăng nhập và mật khẩu của bạn. Nếu bạn không có tài khoản, đăng ký một tài khoản Oracle Web miễn phí và lặp lại quá trình tải xuống. Tải về tập tin .zip.
- Giãn nén tập tin .zip và chạy tập tin setup.exe từ thư mục giải nén để bắt đầu cài đặt.
- Thực hiện theo các hướng dẫn trên màn hình để hoàn tất quá trình cài đặt.

Lưu ý, trong quá trình cài đặt, bạn phải nhập và xác nhận mật khẩu người dùng SYSTEM. Cung cấp một mật khẩu mà bạn có thể dễ dàng nhớ, ví dụ "manager".
Tài khoản SYSTEM/manager được sử dụng để kiểm tra kết nối cơ sở dữ liệu. Màn hình cài đặt cuối cùng sẽ hiển thị một số hiệu cổng, thường là 1521. Những lệnh yêu cầu thông qua mạng sẽ được chuyển đến máy chủ cơ sở dữ liệu thông qua cổng này. Nhớ số hiệu cổng này, vì nó là được yêu cầu trong phần tiếp theo.
Kiểm tra cài đặt Oracle Database XE bằng cách nhấn vào chọn Run SQL Command Line trong nhóm Oracle Database Express Edition 11g. Trên dấu nhắc lệnh SQL, nhập: connect system/manager và nhấn Enter. Nếu quá trình cài đặt thành công, bạn sẽ thấy thông báo Connected và dấu nhắc tiếp, báo rằng cơ sở dữ liệu của bạn đã sẵn sàng và sẵn sàng nhận yêu cầu. Nhập exit và nhấn Enter để thoát khỏi giao diện dòng lệnh.

2.2. SQL Developer

SQL Developer là một công cụ có giao diện người dùng đồ họa (GUI) mà Oracle Corporation cung cấp để truy vấn các cơ sở dữ liệu, duyệt các đối tượng, thực hiện các báo cáo, và chạy các kịch bản lệnh. Nó hỗ trợ Windows, Linux, và Mac OSX. SQL. Ngoài cơ sở dữ liệu Oracle, nó có thể được sử dụng để kết nối và truy cập các cơ sở dữ liệu của bên thứ ba (không phải Oracle) như MySQL, Microsoft SQL Server, Sybase Adaptive Server, Microsoft Access và IBM DB2. Lưu ý rằng SQL Developer không cần bất kỳ trình cài đặt nào vì vậy nó cũng không tạo bất kỳ mục đăng ký nào. Tương tự, xóa thư mục SQL Developer sẽ xóa nó khỏi hệ thống của bạn và bạn không phải chạy bất kỳ trình gỡ cài đặt nào cho nó.
- Tải về từ địa chỉ: http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html
- Lưu và giải nén tệp tin zip vào một thư mục bạn chọn.
- Click vào sqldeveloper.exe từ thư mục được giải nén để chạy SQL Developer.

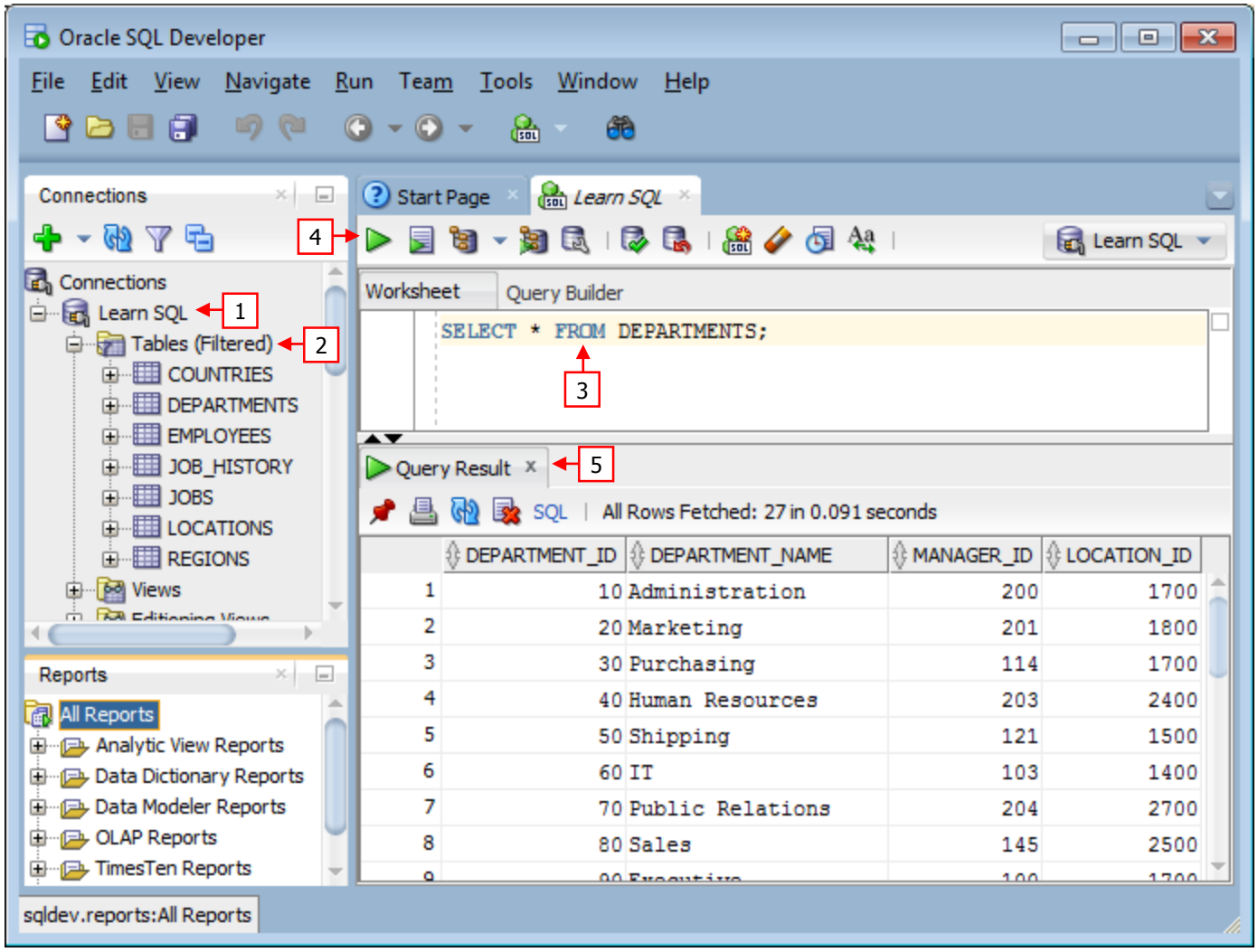
Lưu ý: khi chạy SQL Developer lần đầu, bạn được nhắc chỉ định vị trí JDK. Nếu bạn đã chọn tùy chọn tải xuống SQL Developer kèm với JDK, thì java.exe sẽ được bao gồm trong thư mục con jdk nơi bạn giải nén.

2.3. Tạo kết nối đến cơ sở dữ liệu Oracle mẫu

Một kết nối (connection) là một đối tượng của SQL Developer chỉ định các thông tin cần thiết để kết nối với một cơ sở dữ liệu cụ thể từ một người dùng cụ thể của cơ sở dữ liệu đó. Bạn phải có ít nhất một kết nối cơ sở dữ liệu để sử dụng SQL Developer.
Oracle cung cấp lược đồ mẫu HR (Human Resources) cho các ví dụ trong chương này. Bạn cần mở khóa (unlock) cơ sở dữ liệu HR trong Oracle Database XE.
- Click dấu + trong khung Connections để mở hộp thoại New / Select Database Connection. Tạo một kết nối tên SYSTEM, username/password là system/manager, click nút Connect.
Ghi chú: sau khi nhập đủ thông tin, click nút Test, nếu nhận được thông điệp "Success", kết nối sẵn sàng được sử dụng.
- Giãn nhánh SYSTEM > Other Users, click phải lên user HR và chọn Edit User... Đặt và xác nhận mật khẩu, ví dụ "hr". Tick bỏ chọn "Password Expired" và "Account is Locked". Click nút Apply.
Sau đó, bạn có thể xóa bỏ kết nối SYSTEM và tạo kết nối "Learn SQL":
- Click dấu + trong khung Connections để mở hộp thoại New / Select Database Connection. Tạo một kết nối tên "Learn SQL", với username/password là hr/hr, click nút Save để lưu thông tin kết nối.

2.4. Giao diện SQL Developer

Hình dưới trình bày giao diện của SQL Developer, có đánh dấu những phần quan trọng nhất.



[1] Learn SQL: khung Connections hiển thị tất cả những kết nối bạn tạo để kết nối đến các cơ sở dữ liệu khác nhau.
[2] Tables: bạn sẽ tương tác với bảng cơ sở dữ liệu để thực hành SQL. Khi đã kết nối với cơ sở dữ liệu HR, bạn có thể giãn nút này để duyệt tất cả các bảng trong cơ sở dữ liệu. Bạn có thể thấy các cấu trúc của bảng và dữ liệu được lưu trữ trong mỗi bảng thông qua nút này.
[3] Worksheet: khung Worksheet là nơi bạn nhập các câu lệnh SQL để tương tác với cơ sở dữ liệu của bạn.
[4] Run Statement: nhấn nút này để thực hiện câu lệnh SQL của bạn.
[5] Query Result: kết quả thực hiện phát biểu SQL mà bạn nhập vào Worksheet sẽ xuất hiện trong phần này.

2.5. Tạo kết nối đến Microsoft SQL Server

- SQL Developer dùng JDBC Driver Type 4 jDTS để kết nối SQL Server. Với SQL Developer 4.x, cần jdts-1.3.1, tải về tại:

http://sourceforge.net/projects/jtds/files/.
Giải nén tập tin jdts-1.3.1.jar vào thư mục bạn chọn.
- Trong SQL Developer vào Tools > Preferences..., chọn nhánh Database > Third Party JDBC Drivers.
Click Add Entry... và chỉ đến tập tin driver jdts-1.3.1.jar. Lưu và thoát khỏi Preferences.
- Trong khung Connections, click dấu + để tạo kết nối mới.
Trong hộp thoại New / Select Database Connection, bạn sẽ thấy tab mới SQL Server xuất hiện cạnh tab Oracle.
Chọn tab SQL Server, nhập thông tin đầy đủ:
+ Connection Name: tên database trong SQL Server muốn kết nối.
+ Username: thường là "sa"
+ Password: mật khẩu đăng nhập SQL Server của "sa"
+ Hostname: IP của SQL Server, thường là "localhost".
+ Port: port của SQL Server, mặc định là 1433.
Click Save rồi click Connect.

3. Truy vấn dữ liệu từ cơ sở dữ liệu
Phép Chọn (Selection) trong các toán tử quan hệ khi thực hiện trên quan hệ r sẽ trả về một quan hệ mới gồm đầy đủ các thuộc tính của r, chứa các dòng dữ liệu thuộc r thỏa mãn điều kiện cho trước. Trong SQL, phép Chọn được thực thi thông qua phát biểu SELECT.
Bạn sử dụng lệnh SELECT của SQL để truy vấn dữ liệu từ cơ sở dữ liệu của bạn. Lệnh này giúp bạn rút trích bộ dữ liệu mong muốn, cũng như chỉ định thứ tự trình bày. Trong phần này, bạn sẽ thực hiện tất cả các tùy chọn của lệnh SELECT để lấy dữ liệu từ cơ sở dữ liệu.
Hướng dẫn viết phát biểu SQL hợp lệ, dễ đọc và chỉnh sửa.
+ Lệnh và phát biểu SQL không phân biệt chữ hoa chữ thường.
+ Bạn có thể nhập phát biểu SQL trên một dòng hoặc có thể chia chúng thành nhiều dòng.
+ Các lệnh không được viết tách ra, cũng không được viết tắt.
+ Bạn có thể sử dụng tab và canh hàng để có thể đọc tốt hơn.
+ Đặt dấu chấm phẩy (;) như là một dấu kết thúc trước khi bạn thực hiện phát biểu.
+ Trong cú pháp SQL, văn bản trong cặp ngoặc vuông [] là tùy chọn, và trong cặp ngoặc nhọn {} là bắt buộc. Từ khóa (chẳng hạn như SELECT, FROM ...) được trình bày bằng chữ hoa, trong khi giá trị được do người dùng cung cấp (tên cột/bảng, điều kiện, ...) được hiển thị bằng chữ thường.

- Lệnh SELECT
Cú pháp
```
SELECT [DISTINCT] {* | column [alias], ...}
FROM {table_name}
[WHERE condition(s)]
[GROUP BY expression]
[HAVING group_condition]
[ORDER BY {column, expression} [ASC | DESC]];
```
Giải thích

Mệnh đề	Giải thích
SELECT	từ khóa theo sau bởi ít nhất một cột từ bảng mong muốn.
DISTINCT	mệnh đề tùy chọn, ngăn chặn các kết quả trùng trên cột chỉ định (mặc định là ALL).
*	ký tự đại diện được sử dụng khi chọn tất cả các cột. dấu (|) phân cách giữa các lựa chọn.
column [alias]	danh sách (các) cột chỉ định của một bảng với các tiêu đề tùy chọn.
FROM table_name	tên của bảng mà bạn muốn lấy dữ liệu.
WHERE condition(s)	mệnh đề trong đó chỉ định dữ liệu mong muốn. Điều kiện có thể có tên cột, biểu thức và toán tử so sánh.
GROUP BY expression	chia các dòng trong một bảng thành các nhóm nhỏ hơn.
HAVING group_condition	sử dụng kết hợp với GROUP BY, để trả về chỉ những nhóm được xác định trong điều kiện.
ORDER BY	mệnh đề dùng chỉ định thứ tự hiển thị của tập dữ liệu lấy được.
ASC|DESC	thứ tự các dòng lấy được tăng dần (ASCending) hoặc giảm dần (DESCending).

- Chọn tất cả dữ liệu từ một bảng
Bạn có thể sử dụng lệnh SELECT dưới dạng đơn giản nhất để lấy tất cả dữ liệu từ một bảng. Sử dụng ký tự (*) để lấy dữ liệu từ tất cả các cột. Ví dụ, lấy dữ liệu từ tất cả các cột trong tất cả các dòng có trong bảng Departments. Giả sử rằng bạn đã kết nối đến lược đồ HR bằng cách sử dụng kết nối Learn SQL, nhập phát biểu sau trong khung Worksheet và click vào nút Run Statement. Khung Query Result sẽ xuất hiện, chứa kết quả thực hiện truy vấn.
```
SELECT * FROM Departments;
```
Ghi chú: trong Oracle, bạn có thể sử dụng lệnh DESCRIBE để liệt kê các cột trong một bảng. Ví dụ, để xem danh sách tất cả các cột trong bảng Employees, bạn nhập lệnh:
```
DESCRIBE Employees;
```
Trong các DBMS khác:
SQL Server: `sp_help 'Employees'`
MySQL: `describe Employees`
SQLite: `pragma table_info(Employees);`

- Lấy dữ liệu từ các cột được chọn

Lệnh SELECT cũng cho phép bạn lấy dữ liệu từ các cột chỉ định. Bằng cách chỉ định tên (các) cột mong muốn, phân cách bằng dấu phẩy, bạn có thể giới hạn truy vấn của bạn chỉ lấy kết quả của các cột mà bạn muốn xem. Ví dụ, bạn giới hạn dữ liệu chỉ lấy ba cột (DEPARTMENT_ID, LAST_NAME và MANAGER_ID) từ bảng Employees. Lưu ý rằng kết quả truy vấn của bạn hiển thị các cột theo thứ tự như bạn chỉ định chúng trong câu lệnh SELECT. Cũng lưu ý rằng cả ba cột được phân cách bằng dấu phẩy.

```
SELECT department_id, last_name, manager_id
FROM Employees;
```

- Sử dụng toán tử số học

Bạn có thể tạo các biểu thức số học tùy biến trong phát biểu SELECT với sự trợ giúp của các toán tử số học phổ biến được định nghĩa dưới đây. Ngoài lệnh SELECT, bạn được phép sử dụng các toán tử này trong bất kỳ mệnh đề nào của phát biểu SQL ngoại trừ mệnh đề FROM.

Toán tử	Mô tả
+	Cộng
-	Trừ
*	Nhân
/	Chia

Ví dụ, hiển thị mức lương hàng năm của tất cả nhân viên và phần trăm hoa hồng của họ (commission). Lưu ý rằng cột kết quả không thực sự được tạo ra trong bảng Employees, mà chỉ được tạo ra để hiển thị.

```
SELECT last_name, salary * 12, commission_pct
FROM Employees;
```

- Ưu tiên của các toán tử số học

Trong trường hợp có nhiều toán tử số học trong một phát biểu SQL, phép nhân và phép chia được ưu tiên hơn phép cộng và phép trừ. Nếu các toán tử của cùng cấp ưu tiên, chúng được định trị từ trái sang phải. Trong ví dụ sau, phép nhân được thực hiện đầu tiên và sau đó là kết quả được cộng cho 1000.

```
SELECT last_name, salary, 12 * salary + 1000
FROM Employees;
```

Cặp ngoặc có mức ưu tiên cao nhất, nếu bạn không chắc chắn về thứ tự ưu tiên, hãy sử dụng cặp ngoặc (). Ví dụ:

```
SELECT last_name, salary, 12 * (salary + 1000)
FROM Employees;
```

salary cộng với 1000 trước, sau đó 12 mới nhân cho kết quả.

- Thay đổi tiêu đề cột

Thông thường khi bạn truy vấn dữ liệu từ một bảng, kết quả được hiển thị với các tên cột dùng như các tiêu đề (heading) của bảng kết quả. Đôi khi các tiêu đề bị cắt ngắn hoặc khó hiểu như ví dụ trên. Bạn có thể thay đổi tiêu đề này thành tiêu đề có ý nghĩa bằng cách sử dụng bí danh (alias) của cột. Bí danh chỉ là một tên thay thế cho một cột. Thêm bí danh cột ngay sau tên cột (hoặc biểu thức), với space giữa chúng. Nếu một bí danh chứa dấu cách, ký tự đặc biệt (/ hoặc #) hoặc phân biệt chữ hoa chữ thường, nó phải nằm trong cặp ngoặc kép.
Có hai cách đặt bí danh cho cột, dùng từ khóa AS hoặc đặt bí danh ngay sau cột chỉ định.

```
SELECT last_name, salary, 12 * (salary + 1000) AS "ANNUAL SALARY"
FROM Employees;
```

hoặc

```
SELECT last_name, salary, 12 * (salary + 1000) annual_salary
FROM Employees;
```

- Nối các cột

Trong phát biểu SELECT, bạn có thể nối hai hoặc nhiều cột, các biểu thức số học hoặc các trị hằng vào một cột duy nhất sử dụng toán tử nối (||). Cột kết quả được tạo ra như một biểu thức. Ví dụ, bạn nối họ và tên của nhân viên, với JOB_ID và trình bày kết quả dưới tiêu đề mới: "employees". Mặc dù tiêu đề mới chỉ có một từ, ta đặt vào cặp ngoặc kép như một thói quen.

```
SELECT first_name||' '||last_name||', '||job_id "employees"
FROM Employees;
```

Ghi chú: các DBMS dùng toán tử nối khác nhau
SQL Server: +
MySQL: hàm concat()
SQLite: ||

- Giá trị NULL

Nếu một cột trong bảng thiếu một giá trị trong nó, giá trị đó được gọi là NULL. 0 và space không được định nghĩa là giá trị NULL, vì 0 là một số, và space là một ký tự. Giá trị NULL có thể được định nghĩa là: giá trị không áp dụng, không khả dụng, không xác định, hoặc không được gán. Nếu bạn nhìn vào dữ liệu của cột COMMISSION_PCT trong bảng Employees, bạn sẽ nhận thấy rằng nhân viên không phải là nhân viên bán hàng có các giá trị (null) trong cột, bởi vì phần trăm hoa hồng là *không áp dụng* cho những nhân viên này.
Ví dụ sau tính tiền hoa hồng ra (null) cho nhân viên không có phần trăm hoa hồng, lương nhân với phần trăm hoa hồng (null) dẫn đến các giá trị hoa hồng không hợp lệ (null).

```
SELECT last_name, job_id, salary, commission_pct, salary * commission_pct/100 "Commission"
FROM Employees;
```

Trong ví dụ trên, bạn đã thấy rằng cột commission (hoa hồng) hiển thị (null) cho nhân viên không phải là nhân viên bán hàng, điều này không được coi là cách trình bày dữ liệu tốt. Để ghi đè giá trị mặc định này với văn bản có thể chấp nhận được, bạn dùng hàm NVL.

NVL(expression1, expression2)

Bạn đặt cột nguồn hoặc biểu thức chứa (null) trong biểu thức expression1 và đặt giá trị đích mà bạn muốn thấy thay vì (null) vào expression2. Hàm NVL có thể được sử dụng để chuyển đổi loại dữ liệu bất kỳ. Điều mà bạn cần chú ý là expression2 phải cùng kiểu dữ liệu vốn chứa trong expression1 (nếu khác null). Ví dụ, nếu bạn đang chuyển đổi một cột số, thì bạn phải sử dụng kiểu dữ liệu số trong expression2.
Làm lại ví dụ trước, bạn dùng hàm NVL để chuyển đổi các giá trị (null) trong cột COMMISSION_PCT và kết quả biểu thức COMMISSION - để thành 0.

```
SELECT last_name, job_id, salary, nvl(commission_pct, 0) "Percent", salary * nvl(commission_pct, 0)/100 "Commission"
FROM Employees;
```

- Tránh trùng dữ liệu với DISTINCT

Nhắc lại, phép Chiếu (Projection) lên tập thuộc tính X của quan hệ r là một quan hệ mới bao gồm các thuộc tính X và không chứa các dòng trùng lặp. Phép Chiếu được thực thi bằng cách thêm từ khóa DISTINCT vào mệnh đề SELECT.
Theo mặc định, phát biểu SELECT trả về kết quả của truy vấn mà không loại bỏ các bản ghi trùng lặp. Ví dụ, nếu bạn thực hiện SELECT job_id FROM Employees; bạn sẽ nhận được tất cả các bản ghi bao gồm một số mục trùng lặp trong cột JOB_ID. Để lấy các JOB_ID phân biệt, bạn sẽ phải sử dụng mệnh đề DISTINCT ngay sau từ khóa SELECT.

```
SELECT DISTINCT job_id
FROM Employees;
```

Trong ví dụ trước, bạn chỉ sử dụng một cột (JOB_ID) để hiển thị các giá trị duy nhất. DISTINCT cũng có thể được áp dụng cho tất cả các cột trong phát biểu SELECT. Với DISTINCT được áp dụng cho nhiều cột, tập dữ liệu trả về hiển thị kết hợp phân biệt của các cột đã chọn. Trong ví dụ sau, hiển thị tất cả các bộ khác nhau kết hợp của FIRST_NAME và JOB_ID. Vì không có hai nhân viên có bộ FIRST_NAME và JOB_ID giống nhau, truy vấn lấy tất cả các dòng từ bảng.

```
SELECT DISTINCT first_name, job_id
FROM Employees;
```

- Sắp xếp các bản ghi

Trong tất cả các ví dụ trước, bạn đã lấy dữ liệu mà không chỉ định bất kỳ thứ tự sắp xếp nào. Để sắp xếp dữ liệu lấy được theo thứ tự bạn muốn, sử dụng mệnh đề ORDER BY. Mặc định, mệnh đề này sắp xếp dữ liệu theo thứ tự tăng dần (ASC). Bạn có thể sử dụng tùy chọn DESC để có đầu ra theo thứ tự giảm dần. Nếu được sử dụng, mệnh đề này phải được đặt vào cuối trong phát biểu SELECT. Truy vấn sau được sắp xếp theo LAST_NAME của nhân viên. Một cách khác là chỉ định vị trí của các cột mà bạn muốn sắp xếp dữ liệu. Ví dụ, thay vì nhập vào tên cột (LAST_NAME), bạn có thể sử dụng vị trí của nó: ORDER BY 1. Ngoài ra, bạn có thể thêm nhiều cột vào mệnh đề ORDER BY. Để sắp xếp theo nhiều cột, chỉ cần chỉ định tên các cột được phân cách bởi dấu phẩy (giống như bạn làm khi SELECT nhiều cột). Ví dụ, hiển thị một danh sách nhân viên, được sắp xếp theo LAST_NAME và FIRST_NAME (trước tiên sắp xếp theo LAST_NAME, nếu LAST_NAME giống nhau thì sắp xếp theo FIRST_NAME).
Lưu ý, sắp xếp đầu ra theo một cột không được truy xuất trong truy vấn của bạn là hợp lệ. Trong trường hợp nhiều cột, đầu ra được hiển thị chính xác theo thứ tự chuỗi sắp xếp được chỉ định. Ví dụ, nếu bạn sắp xếp truy vấn bằng LAST_NAME và DEPARTMENT_ID, kết quả sẽ được sắp xếp trước hết bằng LAST_NAME và sau đó là DEPARTMENT_ID.

```
SELECT last_name, department_id, hire_date
FROM Employees
ORDER BY last_name;
```

- Các toán tử so sánh và logic với quy tắc ưu tiên

Các toán tử so sánh và logic được sử dụng trong mệnh đề WHERE của phát biểu SQL. Các toán tử này hỗ trợ định trị một số điều kiện để lấy dữ liệu mong muốn. Giả sử bạn có một bảng tên là Contact có ba bản ghi như thể hiện trong hình dưới đây.

id	name	email	age	message
1	Riaz Ahmed	realtech@cyber.net.pk	30	Feedback a message
2	Daniel Clarke	daniel@gmail.com	25	This is a comment
99	(null)	abc@abc.com	(null)	This is a message

Các bản ghi này được sử dụng trong cột "Ví dụ" trong bảng dưới cho thấy cách sử dụng các toán tử so sánh và logic.
+ Toán tử so sánh

Toán tử	Mô tả	Ví dụ
=	Bằng	SELECT * FROM Contact WHERE name='Riaz Ahmed';
		Trả về bản ghi #1
<> hoặc != hoặc ^=	Không bằng	SELECT * FROM Contact WHERE name <> 'Riaz Ahmed';
		Trả về bản ghi #2
>	Lớn hơn	SELECT * FROM Contact WHERE age > 25;
		Trả về bản ghi #1
>=	Lớn hơn hoặc bằng	SELECT * FROM Contact WHERE age >= 25;
		Trả về bản ghi #1 và #2
<	Nhỏ hơn	SELECT * FROM Contact WHERE id < 2;
		Trả về bản ghi #1
<=	Nhỏ hơn hoặc bằng	SELECT * FROM Contact WHERE id <= 2;
		Trả về bản ghi #1 và #2

BETWEEN ... AND ...	Dãy dữ liệu giữa hai trị	SELECT * FROM Contact WHERE id BETWEEN 2 AND 100;
		Trả về bản ghi #2 và #99
LIKE	So trùng một mẫu	SELECT * FROM Contact WHERE message LIKE '%feed%';
		Trả về bản ghi #1
IN	Tìm kiếm nhiều trị	SELECT * FROM Contact WHERE name IN ('Riaz Ahmed', 'Daniel Clarke');
		Trả về bản ghi #1 và #2
IS NULL	Lấy các giá trị (null)	SELECT * FROM Contact WHERE name IS NULL;
		Trả về bản ghi #3

+ Toán tử logic

Toán tử	Mô tả	Ví dụ
AND	Cả hai điều kiện phải định trị true	SELECT * FROM Contact WHERE name = 'Riaz Ahmed' AND age = 30;
		Trả về bản ghi #1
OR	Một trong hai điều kiện định trị true	SELECT * FROM Contact WHERE name = 'Riaz Ahmed' OR age = 25;
		Trả về bản ghi #1 và #2
NOT	Định trị điều kiện ngược lại	SELECT * FROM Contact WHERE name IS NOT NULL;
		Trả về bản ghi #1 và #2
Kết hợp AND và OR	Có thể dùng trong cùng biểu thức logic	SELECT * FROM Contact WHERE name = 'Riaz Ahmed' AND (age = 25 OR age=99);
		Trả về bản ghi #1

Lưu ý, toán tử NOT khi được sử dụng kết hợp với các toán tử số học sẽ cho kết quả phủ nhận. Ví dụ, nếu bạn thêm toán tử NOT vào các ví dụ trên như sau, bạn sẽ nhận được các bản ghi còn lại so với kết quả trên.

```
NOT name = 'Riaz Ahmed'
NOT BETWEEN 2 AND 100
NOT LIKE '% feed%'
NOT IN ('Riaz Ahmed', 'Daniel Clarke')
name IS NOT NULL
```

Nếu giá trị được so sánh là một chuỗi ký tự, hoặc một ngày, phải đặt nó vào cặp nháy đơn; số không được dùng cặp dấu nháy trừ trường hợp bạn xem chúng như một chuỗi.

- Các quy tắc ưu tiên cho các toán tử so sánh và logic

So sánh và các toán tử logic cũng tuân theo một số nguyên tắc ưu tiên như trường hợp toán tử số học. Bảng dưới đây liệt kê thứ tự định trị cho các toán tử này.

Thứ tự định trị	Toán tử
1	Các toán tử so sánh (=, <>, >, >=, <, <=, BETWEEN, LIKE, IN, IS NULL)
2	AND
3	OR

+ Các toán tử so sánh được định trị đầu tiên, ngay cả dùng NOT phủ định biểu thức.
+ AND có độ ưu tiên cao hơn OR.
+ Các toán tử có mức ưu tiên bằng nhau được định trị từ trái sang phải.

Tương tự như các toán tử số học, ưu tiên cho các toán tử này cũng có thể được chỉ định bằng cách đặt một phần của một biểu thức trong cặp ngoặc đơn. Biểu thức đặt trong cặp ngoặc đơn được định trị đầu tiên.

- Lọc dữ liệu với mệnh đề WHERE

Tất cả phát biểu SELECT được sử dụng trong các ví dụ trước dùng lấy ra tất cả các dòng (bản ghi) từ các bảng chỉ định. Để giới hạn số lượng dòng trả về từ truy vấn, bạn sử dụng mệnh đề WHERE ngay sau mệnh đề FROM. Trong mệnh đề này bạn chỉ định một điều kiện bao gồm ba thành phần: biểu thức (expression), toán tử so sánh (comparison operator) và giá trị (value).

```
... WHERE expression comparison_operator value
```

Ở đây, biểu thức có thể là một cột trong bảng, một giá trị hằng hoặc một biểu thức. Một điều kiện được định trị bằng cách so sánh các dữ liệu được định nghĩa trong biểu thức với giá trị, dùng toán tử so sánh.

Ví dụ, trong phát biểu SQL bên dưới, LOCATION_ID (tên cột) dùng như một biểu thức, (=) là toán tử so sánh, và 1700 là giá trị đang được so sánh với biểu thức. Truy vấn lấy tất cả các bản ghi (với tất cả các cột - *) của các phòng ban được thành lập với LOCATION_ID là 1700.

```
SELECT *
FROM Departments
WHERE location_id = 1700;
```

Ví dụ, dùng toán tử BETWEEN và chỉ định dãy giá trị để lấy danh sách các nhân viên có lương giữa 100 và 10000.

```
SELECT *
FROM Employees
WHERE salary BETWEEN 100 AND 10000
ORDER BY salary;
```

- So sánh các chuỗi ký tự trong mệnh đề WHERE

Để so sánh một chuỗi ký tự trong một mệnh đề WHERE, bạn phải đưa chuỗi vào cặp nháy đơn hoặc nháy kép. Ví dụ, tìm nhân viên có tên là 'JOHN'. Khi bạn thực hiện phát biểu, sẽ không có dòng nào được trả về, bởi vì chuỗi ký tự phân biệt chữ hoa và chữ thường và chuỗi nhập phải giống dữ liệu được lưu trữ trong bảng ('John').
Bạn có thể sử dụng hàm UPPER có sẵn để so trùng với giá trị được cung cấp, như: WHERE UPPER(first_name) = 'JOHN'; Hàm UPPER được sử dụng để chuyển đổi giá trị tại cột FIRST_NAME thành chữ hoa trước khi so sánh nó với giá trị được cung cấp.

```
SELECT first_name, last_name, salary
FROM Employees
WHERE first_name = 'JOHN';
```

- Toán tử BETWEEN

Toán tử BETWEEN được sử dụng trong tình huống bạn đang tìm kiếm bản ghi trong một phạm vi, bao gồm cả hai trị biên. Bạn cung cấp một giá trị biên thấp ngay sau từ khóa BETWEEN và đặt giá trị biên cao ngay sau toán tử logic AND.
Ví dụ, lấy các bản ghi nhân viên có ngày tham gia là từ 01-JAN-06 đến 31-JAN-06, bao gồm cả hai ngày trên. Lưu ý rằng giá trị ngày cũng phải đưa vào cặp nháy đơn và được định nghĩa ở định dạng mặc định là 'dd-MMM-yy'.

```
SELECT first_name, last_name, hire_date
FROM Employees
WHERE hire_date BETWEEN '01 -JAN-06 ' AND '31 -JAN-06';
```

- Toán tử IN

Giả sử bạn muốn xem danh sách các phòng ban thuộc hai vị trí khác nhau, 1800 và 2700. Nếu bạn sử dụng toán tử BETWEEN, bạn sẽ nhận được một danh sách các phòng ban mà bạn không có ý định muốn tìm. Cách khác là sử dụng một danh sách các điều kiện như: location_id = 1800 OR location_id = 2700. Mặc dù nó hợp lệ nhưng điều gì sẽ xảy ra nếu bạn thêm 10 địa điểm vào mệnh đề WHERE. Để giải quyết, bạn dùng toán tử IN, bạn chỉ cần cung cấp một danh sách các giá trị mong muốn trong cặp ngoặc đơn.
Ví dụ, vì LOCATION_ID là số, bạn cung cấp các giá trị mà không cần các cặp nháy đơn. Lưu ý rằng chỉ các ký tự và ngày tháng được sử dụng trong danh sách IN là phải đưa vào cặp dấu nháy đơn.
Lưu ý, sử dụng toán tử IN nếu bạn đang làm việc với danh sách dài các điều kiện. Ngoài ra, toán tử IN cũng có thể chứa một phát biểu SELECT khác để tạo thành mệnh đề WHERE linh hoạt hơn.

```
SELECT department_id, department_name, location_id
FROM Departments
WHERE location_id IN (1800, 2700);
```

- Toán tử LIKE

Trong nhiều tình huống, bạn tìm kiếm các bản ghi trong cơ sở dữ liệu có các giá trị không chính xác như vậy. Sử dụng toán tử LIKE cùng với một chuỗi ký tự mẫu (pattern) bạn có thể dễ dàng tìm thấy kết quả so trùng. Chuỗi ký tự mẫu được xây dựng với sự trợ giúp của hai ký tự đặc biệt: % và _. Ký tự (%) biểu thị cho không có hoặc có nhiều ký tự, trong khi ký tự (_) chỉ biểu thị một ký tự.
Nếu trong chuỗi ký tự mẫu có % và _, thêm ký tự escape (\) trước chúng.
Ví dụ, tìm kiếm tất cả nhân viên có FIRST_NAME bắt đầu bằng chữ "A".

```
SELECT first_name
FROM Employees
WHERE first_name LIKE 'A%';
```

Ví dụ, hiển thị danh sách tất cả nhân viên không chứa 'a' trong FIRST_NAME

```
SELECT first_name
FROM Employees
WHERE first_name NOT LIKE '%a%';
```

Ví dụ, tìm kiếm nhân viên mà FIRST_NAME có chữ 'a' là chữ cái thứ hai.

```
SELECT first_name
FROM Employees
WHERE first_name LIKE '_a%';
```

Lưu ý, một số DBMS phân biệt chữ hoa chữ thường, bạn phải chú ý điều này khi sử dụng toán tử LIKE.

- Toán tử IS NULL

Một giá trị NULL, như đã đề cập trước đây, là một giá trị không có sẵn hoặc không áp dụng được. Nó không phải là số 0 và cũng không phải là ký tự space. Ngoài ra, bạn không thể sử dụng toán tử (=) trong mệnh đề WHERE để so trùng với giá trị NULL. Thủ tục hợp lệ để so trùng giá trị NULL là sử dụng toán tử IS NULL.
Ví dụ, sử dụng toán tử IS NULL để hiển thị bản ghi của các nhân viên bán hàng. Để thu thập các bản ghi còn lại không phải nhân viên bán hàng), sử dụng toán tử NOT logic trong mệnh đề WHERE như sau:

```
SELECT first_name, last_name, commission_pct
FROM Employees
WHERE commission_pct IS NOT NULL;
```

- Toán tử AND/OR

Luôn luôn nhớ các quy tắc sau đây đối với toán tử logic AND và OR:

+ AND sẽ trả về các dòng chỉ khi cả hai điều kiện là true.
+ OR yêu cầu một trong hai điều kiện là true.
+ AND có độ ưu tiên cao hơn OR.

Lưu ý rằng cả hai toán tử này đều có thể được sử dụng cùng nhau trong mệnh đề WHERE của một phát biểu SQL để xây dựng các biểu thức logic phức tạp. Ví dụ, tìm kiếm nhân viên làm việc trong phòng ban 20 là MK_MAN (Marketing Manager). Mệnh đề WHERE trong ví dụ sau

được tạo thành từ hai điều kiện, và từ khóa AND được sử dụng để kết hợp chúng. Kết quả chỉ trả về dòng đáp ứng được hai điều kiện được chỉ định. Nếu một bản ghi có DEPARTMENT_ID là 20, nhưng JOB_ID không phải là MK_MAN, nó không được lấy ra. Tương tự, bản ghi có JOB_ID là MK_MAN ở các phòng ban khác cũng không được lấy ra.

```
SELECT first_name, department_id, job_id
FROM Employees
WHERE department_id = 20 AND job_id = 'MK_MAN';
```

Kết quả có hai bản ghi, bạn có thể thu hẹp kết quả bằng cách thêm các điều kiện lọc khác, như sau:

```
WHERE department_id = 20 AND job_id = 'MK_MAN' AND first_name = 'Michael'
```

Toán tử logic thứ hai mà bạn có thể sử dụng trong câu lệnh SQL là toán tử OR, ít hạn chế hơn nên trả về nhiều dòng hơn. Hầu hết các hệ thống quản lý cơ sở dữ liệu không định trị điều kiện thứ hai trong mệnh đề OR nếu điều kiện đầu tiên đã được đáp ứng, tức là các dòng được trả lại mà không xem xét điều kiện thứ hai nếu điều kiện đầu tiên được định trị là true. Ví dụ, tìm tất cả nhân viên làm việc trong phòng ban 20 hoặc làm việc như một MK_MAN. Bản ghi thứ hai trả về bởi truy vấn không đáp ứng điều kiện job_id là MK_MAN, nhưng vì nó đáp ứng điều kiện đầu tiên (department_id = 20), nó được chọn bởi truy vấn.

```
SELECT first_name, department_id, job_id
FROM Employees
WHERE department_id = 20 OR job_id = 'MK_MAN';
```

- AND/OR sử dụng cùng nhau

Bạn có thể sử dụng số lượng bất kỳ các toán tử AND và OR trong một mệnh đề WHERE duy nhất để tạo lọc dữ liệu phức tạp. Nhưng, đặt cả hai vào một mệnh đề WHERE đôi khi bạn cũng có thể gặp rắc rối. Ví dụ, lấy một danh sách tất cả nhân viên làm việc trong phòng ban 10 hoặc 20 và có thu nhập nhiều hơn hoặc bằng 6000.

```
SELECT first_name||' '||last_name employee, department_id, salary
FROM Employees
WHERE department_id = 10 OR department_id = 20 AND salary >= 6000;
```

Lỗi sai là thứ tự định trị. Như đã đề cập trước đây, toán tử AND có độ ưu tiên cao hơn toán tử OR, giải pháp là sử dụng cặp ngoặc đơn để nhóm các toán tử cho rõ ràng.

```
SELECT first_name||' '||last_name employee, department_id, salary
FROM Employees
WHERE (department_id = 10 OR department_id = 20) AND salary >= 6000;
```

- Thêm chú thích vào câu lệnh SQL

Bạn đã sử dụng các câu lệnh SQL rất đơn giản để lấy thông tin mong muốn từ DBMS. Khi bạn bắt đầu viết các phát biểu SQL dài và phức tạp, người lập trình có kinh nghiệm thường thêm các dòng văn bản mô tả để tham khảo trong tương lai. Một văn bản như vậy được gọi là chú thích (comment), và được nhúng trước hoặc trong phát biểu SQL bằng cách sử dụng hai dấu nối (--) hoặc đưa vào cặp /* và */.

Ví dụ

```
SELECT first_name, last_name  -- This is a comment
FROM Employees;

/* select first_name, last_name from employees */
SELECT first_name, last_name
FROM Employees;
```

4. Chuyển đổi và tổng hợp dữ liệu bằng các hàm

Một hàm (function) có thể được định nghĩa như một tác vụ được thực hiện với mục tiêu chuyển đổi thể hiện của dữ liệu. Giống như các ngôn ngữ máy tính khác, SQL cũng hỗ trợ việc sử dụng hàm để thao tác lên dữ liệu. Hàm thường được áp dụng để:

+ Chuyển đổi các kiểu dữ liệu của các cột
+ Tính toán số liệu
+ Thay đổi định dạng hiển thị
+ Cung cấp kết quả tính toán từ nhóm các dòng

Hàm được chia thành hai loại sau:

- Hàm đơn dòng (single row): các hàm thuộc thể loại này chỉ hoạt động trên từng dòng riêng lẻ và trả về một kết quả duy nhất cho mỗi dòng đã xử lý. Bạn có thể sử dụng các hàm này trong các mệnh đề SELECT, WHERE, ORDER BY.

Phân loại như sau:

+ Hàm ký tự: các hàm này nhận dữ liệu ký tự như đầu vào và có thể trả lại trị ký tự và trị số.
+ Hàm số: các hàm này thao tác dữ liệu số, nhận số nhập và trả về trị số.
+ Hàm ngày: bên cạnh việc trình bày dữ liệu thời gian và ngày ở một số định dạng mong muốn, các hàm này cũng hữu ích trong việc so sánh trị ngày và tính toán thời gian giữa các ngày.
+ Hàm chuyển đổi: với sự trợ giúp của các hàm này bạn có thể chuyển đổi kiểu dữ liệu của một số dữ liệu thành một kiểu dữ liệu khác. Ví dụ, để nối một trị số vào một chuỗi ký tự, trước hết bạn chuyển đổi trị số đó thành kiểu dữ liệu ký tự.

- Hàm tổng hợp (aggregate): ngược lại với các hàm đơn dòng, các hàm này hoạt động trên nhóm các dòng và trả lại một kết quả cho mỗi nhóm. Các hàm này được sử dụng để lấy dữ liệu tổng hợp cho mục đích phân tích và báo cáo nên còn gọi là hàm thống kê.

+ Hàm tính trung bình - AVG
+ Hàm đếm - COUNT
+ Hàm tính trị lớn nhất - MAX
+ Hàm tính trị nhỏ nhất - MIN
+ Hàm tính tổng - SUM

4.1. Các hàm đơn dòng

- Hàm CONCAT
Hàm này tương đương với toán tử nối (||) và được sử dụng để nối tham số thứ nhất với tham số thứ hai.
CONCAT(char1, char2)
Ví dụ sau kết nối tên và họ của tất cả nhân viên.
```
SELECT CONCAT (first_name, last_name) FROM Employees;
```
Ghi chú: để đặt dấu cách giữa hai tên, lồng một hàm CONCAT khác, như sau:
```
SELECT CONCAT(CONCAT(first_name, ' '), last_name) "Employee" FROM Employees;
```
Xem xét ví dụ sau nếu bạn muốn nối trị của hai cột với một số văn bản có ý nghĩa ở giữa:
```
SELECT CONCAT(CONCAT(last_name, 'working as'), job_id) "Name and Job" FROM Employees;
```

- Hàm INITCAP
Hàm INITCAP trả về chuỗi ký tự, với chữ cái đầu tiên của mỗi từ được viết hoa, tất cả các chữ cái khác là chữ thường.
INITCAP(characters)
Ví dụ sau chuyển đổi chữ cái đầu tiên của cột JOB_ID thành chữ hoa, và phần còn lại để chữ thường. Lưu ý rằng các giá trị được lưu trữ trong cột này đang là chữ hoa.
```
SELECT INITCAP(job_id)
FROM Employees;
```

- Hàm LENGTH
Hàm LENGTH trả về chiều dài của chuỗi ký tự. Lưu ý rằng nó là một hàm ký tự, trả về trị số.
LENGTH(characters)
Ví dụ, đếm số ký tự trong FIRST_NAME của nhân viên.
```
SELECT first_name, LENGTH (first_name) "Length"
FROM Employees;
```
Ghi chú: Microsoft SQL Server sử dụng hàm DATALENGTH().

- Hàm LOWER và hàm UPPER
Hàm LOWER trả về chuỗi ký tự nhận được, với tất cả là chữ thường.
LOWER(characters)
Ví dụ, biến đổi tất cả các dữ liệu trong các cột FIRST_NAME và JOB_ID thành chữ thường.
```
SELECT LOWER (first_name||' '||last_name) "Name", LOWER (job_id) "Job"
FROM Employess;
```
Lưu ý: như đã đề cập ở trên, hàm đơn dòng cũng có thể được sử dụng trong mệnh đề WHERE. Ví dụ:
```
SELECT first_name, last_name, job_id FROM Employees WHERE LOWER(job_id) = 'pu_clerk';
```
Hàm UPPER trả về chuỗi ký tự nhận được, với tất cả là chữ hoa.
UPPER(characters)
Ví dụ
```
SELECT UPPER(first_name)
FROM Employees;
```

- Hàm NVL
Khi bạn truy vấn một bảng, các giá trị null được lưu trữ trong một cột (ví dụ, cột commission_pct) có thể hiển thị dưới dạng (null). Sử dụng hàm NVL, bạn có thể thay thế các giá trị null này bằng một trị có ý nghĩa.
NVL(expression1, expression2)
Nếu expression1 là null, thì NVL sẽ trả về expression2. Nếu expression1 không phải là null, thì NVL trả về expression1.
Ví dụ, lấy danh sách nhân viên có LAST_NAME bắt đầu bằng 'B' cùng với COMMISSION_PCT, thay thế các trị (nul) nếu có thành trị 0.
```
SELECT last_name, NVL(commission_pct, 0) commission
FROM Employees
WHERE last_name LIKE 'B%'
ORDER BY last_name;
```
Ghi chú, giá trị (null) phải được thay thế hợp lý; ví dụ thành 0 trong cột số, thành 'None' trong cột ký tự.
```
SELECT city, NVL(state_province, 'None') Province
FROM Locations;
```

- Hàm SUBSTR
Hàm SUBSTR trả về chuỗi ký tự con từ chuỗi ký tự chỉ định, bắt đầu từ ký tự thứ P, lấy L ký tự
SUBSTR(characters, P, L)
Ví dụ, lấy các bản ghi của nhân viên có 'lex' (3 ký tự) trong FIRST_NAME bắt đầu từ vị trí thứ hai.
```
SELECT first_name, last_name, salary
FROM Employees
WHERE substr(first_name,2,3)='lex';
```
Ghi chú: Microsoft SQL Server, MySQL sử dụng hàm SUBSTRING().

- Hàm ROUND
Hàm ROUND là một hàm số làm tròn trị 'n' trong cột, biểu thức hoặc giá trị, đến k (số nguyên) vị trí cho phép bên phải dấu thập phân.

ROUND(n, k)
Nếu không chỉ định k, mặc định k là 0. Nếu k là âm, thì n được làm tròn sang bên trái của dấu thập phân (hàng chục, hàng trăm, …). n có thể có kiểu dữ liệu số bất kỳ hoặc kiểu dữ liệu không phải là số bất kỳ có thể được chuyển đổi ngầm sang một kiểu dữ liệu số.
Ví dụ
```
SELECT ROUND (47.842,2), ROUND (47.842,0), ROUND (47.842, -1)
FROM SYS.DUAL;
```
Kết quả: 47.84, 48 và 50
Lưu ý, bảng DUAL là một bảng được cung cấp với cơ sở dữ liệu Oracle. Nó nằm trong lược đồ SYS nhưng có thể được truy cập bởi bất kỳ người dùng nào. Lợi ích chính của bảng này là bạn có thể sử dụng nó trong phát biểu SELECT để tính toán các biểu thức hằng.
Có hơn 20 hàm số được cung cấp bởi Oracle và các nhà cung cấp khác. Các hàm này chủ yếu được sử dụng cho đại số, hình học, hoặc lượng giác, và do đó, không thường xuyên được sử dụng như các hàm ký tự hoặc các hàm ngày.

- Hàm TRUNC
Hàm TRUNC cũng là một hàm số và hoạt động giống như hàm ROUND. Nó trả về trị n được cắt ngắn đến k chữ số thập phân. Nếu không chỉ định k, mặc định k là 0. k có thể là số âm để cắt ngắn k chữ số trái về phía trái của dấu thập phân. Tương tự như hàm ROUND, nó cũng lấy một tham số bất kỳ kiểu dữ liệu số hoặc tham số bất kỳ kiểu dữ liệu không số có thể được ngầm chuyển sang kiểu dữ liệu số. Nếu bạn không chỉ định k, hàm trả về cùng kiểu dữ liệu với kiểu dữ liệu số của tham số. Nếu bạn truyền k, hàm trả về NUMBER.
TRUNC(n, k)
Ví dụ
```
SELECT TRUNC (47.842310,2), TRUNC (47.842310,0), TRUNC (47.842310, -1)
FROM SYS.DUAL;
```
Kết quả: 47.84, 47 và 40

- Hàm Date Time
Mỗi DBMS cho phép bạn lưu trữ các giá trị ngày và giờ trong bảng bằng cách sử dụng kiểu dữ liệu ngày chỉ định và ở các định dạng chỉ định. Mỗi cài đặt theo định dạng lưu trữ của riêng mình để lưu ngày tháng và thời gian. Ví dụ, hiển thị mặc định của Oracle và định dạng đầu vào cho bất kỳ ngày nào là DD-MON-YY. Không may, lưu trữ mặc định này thay đổi giữa các cài đặt khác nhau và do đó ít khả chuyển.
Tương tự như các hàm ký tự, các hàm ngày tháng và thời gian cũng được sử dụng để thao tác thể hiện dữ liệu. Các hàm này được sử dụng không chỉ để trình bày dữ liệu ngày và giờ trong một số định dạng mong muốn, mà còn hữu ích trong việc so sánh giá trị ngày, và tính toán khoảng cách giữa các ngày.
Ví dụ, trong Oracle lấy ngày hiện tại từ hệ thống.
```
SELECT SYSDATE
FROM DUAL;
```
Ghi chú

DBMS	Hàm	Phát biểu SQL	Định dạng xuất
MySQL	CURDATE	SELECT CURDATE();	1975-04-30 (YYYY-MM-DD)
SQL Server	GETDATE	SELECT GETDATE ();	1975-04-30 14:10:02.047
SQLite	DATE	SELECT DATE('now');	1975-04-30 (YYYY-MM-DD)

- Hàm thao tác ngày
Các ví dụ sau trình bày hàm thao tác ngày cho các cài đặt khác nhau, bắt đầu với Oracle. Bảng Employees chứa một cột ngày tháng có tên là HIRE_DATE. Tất cả các ví dụ trong phần này lấy một danh sách nhân viên được thuê vào năm 2003.
Ta sử dụng các hàm lồng nhau để chuyển đổi phần năm trong cột HIRE_DATE thành ký tự - TO_CHAR(hire_date, 'YYYY'). Sau đó chuyển đổi chuỗi ký tự thành số, sử dụng hàm TO_NUMBER để so trùng với năm chỉ định.
```
SELECT first_name, hire_date
FROM Employees
WHERE to_number (to_char (hire_date, 'YYYY')) = 2003;
```
Ghi chú

DBMS	Phát biểu SQL
Oracle	SELECT first_name, hire_date FROM Employees WHERE hire_date BETWEEN to_date ('01-JAN-2003') AND to_date('31-DEC-2003');
SQL Server	SELECT first_name, hire_date FROM Employees WHERE DATEPART(yy, hire_date) = 2003;
MySQL	SELECT first_name, hire_date FROM Employees WHERE YEAR(hire_date) = 2003;
SQLite	SELECT first_name, hire_date FROM Employees WHERE strftime('%Y', hire_date) = 2003;

Sau đây, chúng ta sẽ tìm hiểu một số hàm ngày của Oracle

- Hàm MONTHS_BETWEEN
Hàm MONTHS_BETWEEN hàm dùng xác định số tháng giữa hai ngày. Đầu ra của hàm này có thể là dương hoặc âm. Để kết quả dương, date1 phải sau date2. Ngược lại, kết quả âm được hiển thị khi date1 sớm hơn date2.
MONTHS_BETWEEN (date1, date2)
Ví dụ, tính thời gian làm việc trong tháng của nhân viên số 200. Kết quả dựa trên sự khác nhau giữa ngày hệ thống hiện tại và ngày được lưu trữ trong cột HIRE_DATE.
```
SELECT first_name, hire_date, MONTHS_BETWEEN(sysdate, hire_date) Months_Employed
FROM Employees
WHERE employee_id = 200;
```

- Hàm ADD_MONTHS
Đôi khi bạn cần phải định trị một ngày trong tương lai hoặc một ngày trong quá khứ. Hàm ADD_MONTHS hỗ trợ bạn trong việc này. Để tìm một ngày trong tương lai, thêm số tháng k nguyên cho date. Tương tự, để có được ngày trước đó, hãy nhập một giá trị âm cho k.
ADD_MONTHS(date, k)
Ví dụ
```
SELECT ADD_MONTHS(sysdate, 1) Month
FROM Dual;
SELECT ADD_MONTHS (sysdate, -1) Month
FROM Dual;
```

- Hàm NEXT_DAY(Oracle)
Hàm NEXT_DAY trả về ngày tiếp theo của tuần được chỉ định trong char. Kiểu trả về luôn là ngày. Tham số char phải là một ngày trong tuần, và có thể được truyền như là tên đầy đủ ('Friday'), viết tắt ('Fri'), hoặc một số đại diện cho ngày trong tuần (6).
NEXT_DAY(date, char)
Ví dụ
```
SELECT 'Next Friday will be on '||NEXT_DAY(sysdate, 'Friday') "Next Friday"
FROM Dual;
```

- Hàm LAST_DAY(Oracle)
Hàm LAST_DAY trả về ngày của ngày cuối cùng của tháng được chỉ định trong tham số date.
LAST_DAY(date)
Phát biểu sau trình bày hai cách sử dụng khác nhau của hàm này và hiển thị dữ liệu trong ba cột. Cột đầu tiên, SYSDATE, trả về dữ liệu hệ thống hiện tại để
bổ sung kết quả xuất hiện trong hai cột tiếp theo. Cột thứ hai hiển thị ngày cuối cùng của tháng hiện tại (dựa trên SYSDATE), trong khi cột cuối hiển thị ngày cuối cùng của tháng có ngày chỉ định truyền như tham số.
```
SELECT sysdate, LAST_DAY(sysdate), LAST_DAY('24-DEC-17')
FROM Dual;
```

- Hàm chuyển đổi
Hàm chuyển đổi cho phép bạn chuyển đổi một kiểu dữ liệu sang một kiểu dữ liệu khác. Ví dụ, bảng của bạn chứa một số dữ liệu được lưu trữ ở định dạng ký tự mà bạn muốn chuyển đổi thành số để thực hiện một số tác vụ toán học. Tương tự, bạn cũng có thể chuyển dữ liệu số thành ký tự, nhưng thể hiện của dữ liệu chuyển đổi đó không hỗ trợ các hàm toán học và tính toán.
Bạn có thể chuyển đổi dữ liệu theo các cách sau:
+ Ngày thành ký tự
+ Số thành ký tự
+ Ký tự thành số
+ Ký tự thành ngày
Oracle cung cấp 38 hàm chuyển đổi để thực hiện các tác vụ khác nhau. Chúng ta chỉ xem xét bốn hàm được sử dụng phổ biến nhất, thực hiện bốn nhiệm vụ chuyển đổi đã đề cập ở trên.
+ TO_CHAR (datetime)
+ TO_CHAR (number)
+ TO_NUMBER
+ TO_DATE
Lưu ý: trong nhiều trường hợp, cơ sở dữ liệu cung cấp các hàm chuyển đổi riêng của mình để chuyển đổi một kiểu dữ liệu thành kiểu khác. Tuy nhiên, có một số trường hợp người dùng muốn chỉ định loại dữ liệu họ cần để chuyển đổi dữ liệu. Đối với các kịch bản như vậy, SQL có hàm ANSI CAST để chuyển đổi kiểu dữ liệu sang các kiểu dữ liệu khác mà bạn chọn: CAST(EXPRESSION AS NEW_DATA_TYPE)
Ví dụ: SELECT CAST (hire_date AS CHAR (25)) AS "Hired On" FROM Employees;

- Kiểu DATETIME và các thành phần
Trước hết, chúng ta xem xét một số loại dữ liệu tiêu chuẩn cho ngày và thời gian. Có ba loại dữ liệu chuẩn SQL để lưu trữ ngày tháng và thời gian:
+ DATE: được định dạng là YYYY-MM-DD và trong phạm vi từ 0001-01-01 đến 9999-12-31
+ TIME: được định dạng là HH: MI: SS.nn và trong phạm vi từ 00:00:00 đến 23:59:61.999
+ TIMESTAMP: được định dạng là YYYY-MM-DD HH:MI:SS.nn và trong phạm vi từ 0001-01-01 00:00:00 đến 9999-12-31 23:59:61.999
Các bảng dưới đây liệt kê các thành phần định dạng ngày tháng và thời gian cho Oracle. Nhắc lại, trong Oracle, định dạng mặc định để hiển thị giá trị ngày là DD-MON-YY.
+ Thành phần ngày tháng

ELEMENT	Mô tả	Kết quả
YYYY	Thể hiện năm đầy đủ (số).	2017
YEAR	Thể hiện năm (chuỗi).	TWENTY SEVENTEEN
MM	Thể hiện hai chữ số của tháng.	11
MONTH	Thể hiện tên đầy đủ của tháng.	NOVEMBER
DD	Thể hiện hai chữ số của ngày	20.
DY	Thể hiện viết tắt 3 chữ của ngày trong tuần.	MON (Monday)
DAY	Thể hiện tên đầy đủ của ngày trong tuần.	MONDAY

+ Thành phần thời gian

ELEMENT	Mô tả
AM hoặc PM	Thể hiện cho chỉ báo kinh tuyến.
A.M. hoặc P.M.	Thể hiện chỉ báo kinh tuyến với dấu chấm.
HH hoặc HH12 hoặc HH24	Thể hiện giờ trong ngày hoặc giờ (1-12) hoặc giờ (0-23).
MI	Thể hiện cho phút (0-59).
SS	Thể hiện cho giây (0-59).

- Hàm TO_CHAR(datetime)

Hàm TO_CHAR (datetime) được sử dụng để chuyển đổi một giá trị ngày thành một chuỗi ký tự.

TO_CHAR (datetime [, 'fmt'])

Việc chuyển đổi ngày có thể diễn ra vì một số lý do:

+ Để so sánh giá trị ngày của các loại dữ liệu khác nhau.
+ Để chuyển đổi một giá trị ngày từ định dạng mặc định của nó sang một định dạng được chỉ định bởi bạn.

Truy vấn sau lấy ngày hệ thống trong định dạng thời gian 24 giờ và năm đầy đủ. Thay đổi truy vấn bằng cách thay thế các thành phần được định nghĩa trong phần trước và lưu ý kết quả. Ví dụ: thay đổi mô hình định dạng thành 'fmDD "of" MONTH YYYY' sẽ mang lại kết quả: '20 of NOVEMBER 2017'. Tiền tố fm (viết tắt của fill mode) được thêm vào để xóa đệm trống.

Ví dụ

```
SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24: MI: SS') AS "Current Date"
FROM Dual;
```

Ghi chú:

+ Mô hình định dạng (fmt) phân biệt chữ hoa chữ thường (mon sẽ trả về nov cho tháng 11).
+ Mô hình định dạng phải đưa vào cặp nháy đơn.
+ Sử dụng chế độ điền (fm) để loại bỏ khoảng trống hoặc các số không ở đầu.

- Hàm TO_CHAR(number)

Hàm TO_CHAR (number) chuyển đổi số n thành chuỗi ký tự, sử dụng định dạng số tùy chọn (fmt).

TO_CHAR (n [, 'fmt'])

Hàm này thường được sử dụng khi bạn dự định nối các giá trị số vào một chuỗi, như thể hiện trong ví dụ dưới đây, chuỗi hoàn chỉnh được tạo ra bằng cách nối hai giá trị số (EMPLOYEE_ID và SALARY) thành chuỗi ký tự. Sử dụng các thành phần được liệt kê trong bảng dưới đây để thiết lập định dạng đầu ra.

+ Thành phần số

ELEMENT	Mô tả	Ví dụ	Kết quả
9	Thể hiện vị trí số và xác định hiển thị chiều rộng.	999999 1234	
0	Được sử dụng để hiển thị các số 0 đầu chuỗi.	099999 001234	
$	Hiển thị các giá trị với ký tự $ đầ chuỗi.	$ 999999	$1234
.	Đặt dấu thập phân ở vị trí chỉ định.	999999.99	1234.00
,	Đặt dấu phẩy ở vị trí chỉ định.	999,999	1,234
MI	Trả về giá trị âm với một dấu (-) theo sau.	999999MI	1234-
PR	Trả về giá trị âm trong cặp <>.	999999PR	<1234>

```
SELECT 'Employee number '||TO_CHAR(employee_id)||' gets '||TO_CHAR(salary,'fm$9,999,999') "Salaries"
FROM Employees;
```

- Hàm TO_NUMBER

TO_NUMBER chuyển đổi một biểu thức ký tự (expr) thành giá trị có kiểu dữ liệu NUMBER.

TO_NUMBER(expr [, 'fmt'])

Đối với một chuỗi ký tự được chuyển đổi thành một số, giá trị expr phải từ 0 đến 9. Ví dụ, chuỗi HELLO không thể chuyển đổi thành một số, trong khi postal/zip code được lưu trữ dưới dạng chuỗi ký tự trong một bảng có thể chuyển đổi thành số bằng cách sử dụng hàm này. Bạn cũng có thể sử dụng các ký hiệu (+), (-) và (.) trong expr để biểu diễn các trị dương, âm và thập phân.

Phát biểu sau tính toán mức lương tăng cho nhân viên số 200 bằng cách chuyển đổi chuỗi ('600.00') rồi cộng vào lương thực tế.

```
SELECT salary, salary+TO_NUMBER('600.00') "INCREMENTED SALARY"
FROM Employees
WHERE employee_id=200;
```

- Hàm TO_DATE

Hàm TO_DATE chuyển đổi kiểu dữ liệu ký tự (char) thành giá trị của kiểu dữ liệu DATE.

TO_DATE(char [, 'fmt'])

fmt là một định dạng mô hình datetime chỉ định định dạng của char, luôn chỉ định fmt với TO_DATE.

Phát biểu sau lấy bản ghi của nhân viên đã được thuê vào ngày 3 tháng 3 năm 2005. hàm TO_DATE chuyển đổi giá trị ('March 3, 2005') sang kiểu dữ liệu date, định dạng nó như là 'Month DD, YYYY ', và sau đó so sánh nó với giá trị cột HIRE_DATE trong bảng Employees. Ngày xuất được trình bày ở định dạng mặc định. Nếu bạn muốn ngày xuất cũng trong cùng một định dạng, thay đổi cột HIRE_DATE trong mệnh đề SELECT như sau: TO_CHAR(hire_date,'Month DD, YYYY') "HIRE DATE"

```
SELECT first_name, salary, hire_date
FROM Employees
WHERE hire_date=TO_DATE('March 3, 2005','Month DD, YYYY');
```

- Tính toán ngày

Vì ngày được lưu trữ dưới dạng số trong bảng cơ sở dữ liệu, bạn có thể thực hiện tính toán trên các giá trị ngày sử dụng toán tử số số học như (+), (-), và (/), như được đề cập trong bảng sau. Giá trị ngày cho SYSDATE trong cột Ví dụ được giả định là 21-NOV-2017 (12 giờ tối).

Toán tử	Mô tả	Kết quả	Ví dụ
Date + Number	Thêm một số ngày vào date.	Date	SYSDATE+5, trả về 26-NOV-17
Date – Number	Trừ một số ngày từ date.	Date	SYSDATE-5, trả về 16-NOV-17
Date – Date	Số ngày giữa hai ngày.	Số ngày	SYSDATE-to_date('01-NOV-17', trả về 20.50
Date + Number / 24	Thêm một số giờ vào date.	Date	SYSDATE+24/24 trả về 22-NOV-17

Ví dụ sau đây trả về số tuần làm việc của mỗi nhân viên bằng cách sử dụng toán tử chia.

```
SELECT first_name, (sysdate - hire_date) / 4 "Weeks"
FROM Employees;
```

4.2. Các hàm tổng hợp

Trọng tâm của phần này là cách dữ liệu có thể được nhóm lại và tổng hợp để cho phép bạn tương tác với nó ở một số mức độ chi tiết cao hơn những gì được lưu trữ trong cơ sở dữ liệu, bằng cách sử dụng các hàm tổng hợp. Các hàm tổng hợp trả lại một kết quả duy nhất tính từ nhóm các dòng, chứ không phải trên các dòng đơn. Bạn sử dụng các hàm này trong các truy vấn SQL của bạn để lấy dữ liệu cho mục đích phân tích và báo cáo. Ví dụ, bạn có thể sử dụng các hàm này để:

+ Định trị tổng số bản ghi trong một bảng, hoặc số dòng đáp ứng một số tiêu chí chỉ định.
+ Lấy thông tin tổng kết như giá trị tổng, trung bình, cao nhất hoặc thấp nhất từ một cột của bảng.

Bạn sẽ sử dụng các hàm tổng hợp sau đây trong phần tiếp theo để đạt được mục tiêu đề cập ở trên. Lưu ý rằng không giống như hàm đơn dòng, các hàm tổng hợp được hỗ trợ bởi tất cả các cài đặt SQL.

Hàm	Cú pháp
AVG	AVG([DISTINCT|ALL]|expr)
COUNT	COUNT([DISTINCT|ALL]|expr)
MAX	MAX([DISTINCT|ALL]|expr)
MIN	MIN([DISTINCT|ALL]|expr)
SUM	SUM([DISTINCT|ALL]|expr)

Các hàm tổng hợp thường được thêm vào danh sách SELECT, nhưng cũng có thể xuất hiện trong mệnh đề ORDER BY và HAVING. Chúng thường được sử dụng kết hợp với GROUP BY trong một truy vấn để phân chia các dòng của một bảng truy vấn thành các nhóm.

- Hàm AVG

Hàm AVG được sử dụng để lấy giá trị trung bình của dữ liệu trong một cột chỉ định. Hàm tính toán giá trị trung bình bằng cách chia tổng giá trị trong cột với số dòng trong bảng. Phát biểu bên dưới sử dụng hàm AVG dưới dạng đơn giản nhất, trả về mức lương trung bình của tất cả nhân viên như là một giá trị duy nhất.

```
SELECT AVG(salary) "Average Salary"
FROM Employees;
```

Ghi chú: để có được giá trị trung bình của nhiều cột, bạn có thể sử dụng nhiều hàm AVG trong một truy vấn, mỗi dấu cách nhau bởi dấu phẩy, như sau: AVG(salary), AVG(commission_pct).

Hàm AVG bỏ qua các dòng cột có chứa các giá trị (null).

Tất cả năm hàm tổng hợp bao gồm các mệnh đề DISTINCT|ALL, mặc định là ALL áp dụng hàm cho tất cả các dòng. Trong ví dụ sau, mức lương trung bình cao hơn do sử dụng mệnh đề DISTINCT. Việc loại trừ các bản trùng cho kết quả trung bình cao hơn.

```
SELECT AVG(DISTINCT salary) "Average Salary"
FROM Employees;
```

Sử dụng hàm AVG bạn cũng có thể định trị giá trị trung bình của một số tiêu chí cụ thể. Ví dụ, truy vấn sau xác định mức lương trung bình của nhân viên trong phòng ban 50. Lọc dữ liệu dựa trên mệnh đề WHERE chỉ lọc lương của nhân viên làm việc trong phòng ban chỉ định.

```
SELECT AVG(salary) "Average Salary"
FROM employees
WHERE department_id=50;
```

- Hàm COUNT

Hàm COUNT đếm tổng số dòng trong một bảng hoặc trả về số dòng thỏa điều kiện được chỉ định trong phần expr. Bạn có thể sử dụng hàm này theo hai định dạng:

COUNT(*) đếm số dòng trong một bảng, bao gồm cả các dòng trùng lặp và các dòng có chứa trị null.
COUNT(expr) trả về số các dòng không null trong cột được chỉ định bởi expr.
Ví dụ, đếm tất cả các dòng, không quan tâm các giá trị, và trả về tổng số bản ghi trong bảng Employees:

```
SELECT COUNT (*)
FROM Employees;
```

Ví dụ, tính số nhân viên có quyền nhận hoa hồng. Để thu hẹp thêm kết quả, bạn có thể thêm mệnh đề WHERE vào phát biểu:

```
SELECT COUNT(commission_pct)
FROM Employees
WHERE department_id = 80;
```

- Hàm MIN và MAX

Hai hàm này được sử dụng để lấy trị cao nhất và trị thấp nhất từ một cột. Thực tế, bạn có thể áp dụng chúng cho dữ liệu ngày và thậm chí là ký tự.
Ví dụ, lấy dữ liệu số cao nhất và thấp nhất

```
SELECT MIN(salary) "Minimum", MAX(salary) "Maximum"
FROM Employees;
```

Ví dụ, lấy dữ liệu ngày cao nhất và thấp nhất:

```
SELECT MIN(hire_date) "Minimum", MAX(hire_date) "Maximum"
FROM Employees;
```

Ví dụ, áp dụng lấy trị cao nhất và thấp nhất cho một cột kiểu ký tự, theo thứ tự alphabet.

```
SELECT MIN(first_name) "Minimum", MAX(first_name) "Maximum"
FROM Employees;
```

Các hàm tổng hợp cũng có thể được dùng lồng với nhau. Ví dụ, tính trung bình của mức lương tối đa của tất cả các phòng ban.

```
SELECT AVG(MAX(salary))
FROM Employees
GROUP BY department_id;
```

- Hàm SUM

Hàm SUM trả về tổng số các giá trị của expr.
SUM(expr)
Ví dụ, SUM(salary) - được áp dụng cho một cột của bảng (SALARY) để tính tổng tiền lương được thanh toán trong phòng ban 80, trong lúc SUM (SALARY * commission_pct) - là một biểu thức lấy tổng số tiền hoa hồng được thanh toán trong bộ phận này bằng cách nhân từng trị của cột SALARY với giá trị % tương ứng trong cột COMMISSION_PCT.

```
SELECT SUM(salary), SUM(salary * commission_pct)
FROM Employees
WHERE department_id =80;
```

- Sử dụng các hàm tổng hợp cùng nhau

Một tình huống cụ thể có thể yêu cầu sử dụng tất cả các hàm này trong một phát biểu SQL. Ví dụ sau tập hợp tất cả năm hàm trong một phát biểu cho thấy một tổng kết về bảng Employees.

```
SELECT COUNT(*) "Total Employees", MIN(salary) "Minimum Salary", MAX(salary) "Maximum Salary", AVG(salary) "Average Salary", SUM(salary) "Total Salary"
FROM Employees;
```

- Mệnh đề GROUP BY

Cú pháp SELECT chứa thêm hai mệnh đề: GROUP BY và HAVING.
Mệnh đề GROUP BY được sử dụng trong phát biểu SELECT để nhóm các dòng trong một bảng thành các nhóm con, điều này cho phép áp dụng các hàm tổng hợp để sinh tổng kết cho các nhóm con này. DBMS áp dụng các hàm tổng hợp cho mỗi nhóm con các dòng và trả về một dòng kết quả duy nhất cho mỗi nhóm. Nếu mệnh đề này bị bỏ qua, thì các hàm tổng hợp sẽ được áp dụng cho tất cả các dòng trong bảng truy vấn - như bạn đã thấy trong các phần trước.
Ví dụ, trong ví dụ về hàm COUNT bạn đã thực hiện SELECT COUNT(*) FROM Employees; để có được tổng số nhân viên từ bảng. Nếu bạn được yêu cầu lấy số nhân viên làm việc trong từng phòng ban, sử dụng mệnh đề GROUP BY giúp bạn có được số liệu tổng kết cho mỗi nhóm.

```
SELECT department_id "Department", count(*) "Number of Employees"
FROM Employees
GROUP BY department_id
ORDER BY department_id;
```

Hướng dẫn cho mệnh đề GROUP BY

+ Sau khi nhóm các dòng, ta thường sắp xếp dữ liệu để hiển thị theo một thứ tự nào đó, vì vậy nên sử dụng mệnh đề ORDER BY ngay sau mệnh đề GROUP BY.
+ Đặt mệnh đề GROUP BY giữa mệnh đề WHERE và HAVING. Bằng cách sử dụng mệnh đề WHERE, bạn loại trừ trước các dòng không thỏa điều kiện WHERE trước khi chia chúng thành các nhóm. Trong khi đó, HAVING lọc dữ liệu sau khi các nhóm được hình thành.
+ Ví dụ hiện tại chỉ sử dụng một cột trong mệnh đề GROUP BY. Tuy nhiên, bạn có thể tạo các nhóm lồng nhau bằng cách thêm nhiều cột tùy theo nhu cầu của bạn.
+ Để cột được sử dụng trong hàm tổng hợp, bạn phải bao gồm tất cả các cột hoặc các biểu thức (được chỉ rõ trong danh sách SELECT) trong mệnh đề GROUP BY.
+ Nếu nhóm cột chứa một dòng với một giá trị (null), một nhóm (null) riêng biệt sẽ được tạo ra cho bản ghi đó. Nhóm (null) này được tạo ra bởi vì DEPARTMENT_ID không được chỉ định trong bảng.

- Sử dụng WHERE và GROUP BY cùng nhau

Như đã nêu ở trên, bạn có thể sử dụng mệnh đề WHERE trong câu lệnh SELECT cùng với mệnh đề GROUP BY để loại trừ các dòng không thỏa điều kiện WHERE trước khi chia chúng thành các nhóm. Để làm điều này, mệnh đề WHERE phải được đặt trước mệnh đề GROUP BY.
Ví dụ, hiển thị danh sách các loại job và tổng tiền lương hàng tháng cho mỗi loại job, không bao gồm nhân viên bán hàng: SA_MAN và SA_REP.

```
SELECT job_id "Job Category", sum(salary) "Salary"
FROM Employees
WHERE job_id NOT LIKE 'SA%'
GROUP BY job_id
ORDER BY sum(salary);
```

Như bạn thấy, mệnh đề WHERE được sử dụng trước GROUP BY và dùng điều kiện để loại trừ các dòng không liên quan. Điều kiện loại trừ hai loại nhân viên (SA_MAN, Sales Manager và SA_REP, Sales Representative). Cũng lưu ý mệnh đề ORDER BY sử dụng biểu thức (được liệt kê trong danh sách SELECT) để sắp xếp đầu ra.

- WHERE với HAVING

WHERE nên được sử dụng để lọc đầu vào GROUP BY, và HAVING chỉ được sử dụng với GROUP BY để lọc dữ liệu sau khi đã nhóm. Bạn không thể sử dụng mệnh đề WHERE để lọc các nhóm. Hai ví dụ sau đây cho thấy các nguyên tắc này:

```
SELECT department_id, AVG(salary) FROM Employees WHERE AVG(salary) > 5000 GROUP BY department_id; -- sai
SELECT department_id, AVG(salary) FROM Employees GROUP BY department_id HAVING AVG(salary)> 5000; -- đúng
```

- Mệnh đề HAVING

Mệnh đề HAVING lọc các nhóm dòng trả về từ GROUP BY. Nếu bạn bỏ qua mệnh đề này, thì DBMS trả về kết quả tổng kết cho tất cả các dòng của nhóm. DBMS thực hiện các bước sau khi nó thấy mệnh đề HAVING trong phát biểu SELECT:

+ Loại trừ các dòng không thỏa điều kiện của mệnh đề WHERE.
+ Chuyển các dòng thành các nhóm.
+ Áp dụng hàm nhóm.
+ Hiển thị các nhóm sao cho điều kiện quy định trong mệnh đề HAVING là true.

Ví dụ sau đây đã được trình bày ở phần trước và được lặp lại ở đây bằng cách thêm mệnh đề HAVING để lọc kết quả trên cơ sở của thông tin tổng hợp. So sánh kết quả của phát biểu này với phát biểu trước. Mệnh đề HAVING đã loại bỏ ba nhóm đầu tiên xuất hiện trong kết quả trước đó.

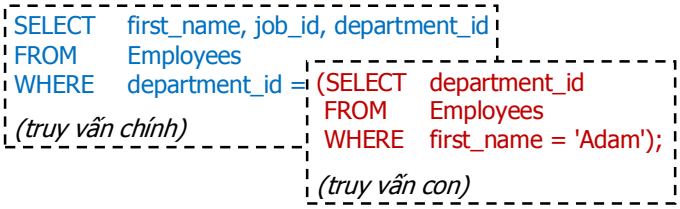
```
SELECT job_id "Job Category", sum(salary) "Salary"
FROM Employees
WHERE job_id NOT LIKE 'SA%'
GROUP BY job_id
HAVING sum(salary) >= 8300
ORDER BY sum(salary);
```

Một điều nữa cần được làm rõ đối với các mệnh đề GROUP BY và HAVING là mệnh đề GROUP BY có thể được sử dụng mà không dùng bất kỳ hàm tổng hợp nào trong danh sách SELECT. Trong ví dụ sau, cả hai mệnh đề này tồn tại mà không dùng bất kỳ hàm tổng hợp nào trong danh sách SELECT. Bởi vì hàm tổng hợp SUM được tham chiếu trong mệnh đề HAVING, mệnh đề GROUP BY cũng được chỉ định.

```
SELECT department_id
FROM Employees
GROUP BY department_id
HAVING sum(salary) < 8000;
```

5. Truy vấn con

Một truy vấn là một tác vụ lấy dữ liệu từ một hoặc nhiều bảng hoặc view. Trong ngữ cảnh hiện tại, phát biểu SELECT cấp cao nhất được gọi là truy vấn chính, và một truy vấn được lồng bên trong một mệnh đề của truy vấn chính được gọi là *truy vấn con* (subquery), như minh họa trong hình dưới đây.



Các truy vấn con chủ yếu được sử dụng trong trường hợp các tiêu chí cho dữ liệu được truy vấn là không xác định.
Hướng dẫn:
+ Một phát biểu SELECT được nhúng trong một mệnh đề của phát biểu SQL khác được gọi là một truy vấn con.
+ Một truy vấn con có thể được đặt trong các mệnh đề SQL sau: WHERE, HAVING, INTO (một mệnh đề của phát biểu INSERT), SET (một mệnh đề của phát biểu UPDATE), FROM (một mệnh đề được sử dụng trong phát biểu SELECT và DELETE).
+ Các truy vấn con phải được đặt trong cặp ngoặc đơn.
+ Các truy vấn con được định nghĩa ở vế phải của toán tử.
+ Sử dụng toán tử so sánh (=, >, >=, <, <=, <>) trong các truy vấn con trả về một dòng. Đối với các truy vấn con trả về nhiều dòng, hãy sử dụng toán tử đa dòng (IN và NOT IN).
+ Truy vấn con thực hiện một lần trước truy vấn chính, và kết quả của nó được sử dụng để hoàn thành điều kiện được định nghĩa trong truy vấn chính hoặc truy vấn bao ngoài.

+ Các truy vấn con không được chứa mệnh đề ORDER BY, ORDER BY được sử dụng trong phát biểu SELECT chính.
+ Bạn có thể đặt số lượng bất kỳ các truy vấn con trong một câu lệnh SELECT duy nhất, được lồng với mức bất kỳ; tuy nhiên, nếu bạn lồng sâu, phải quan tâm đến hiệu suất.

- Tìm hiểu quá trình truy vấn con
Giả sử bạn muốn lấy FIRST_NAME và JOB_ID của các nhân viên làm việc cùng phòng ban với Adam. Cho đến nay, bạn sẽ chỉ đơn giản tạo được hai phát biểu riêng biệt để có được kết quả mong muốn.
Phát biểu đầu tiên như sau:
```
SELECT department_id FROM Employees WHERE first_name = 'Adam';
```
Phát biểu này sẽ trả lại số hiệu phòng ban của Adam, ví dụ 50. Bây giờ, để tìm các nhân viên khác làm việc trong phòng ban này, bạn sẽ chạy phát biểu thứ hai, như sau, để có được đầu ra như yêu cầu:
```
SELECT first_name, job_id FROM nhân viên WHERE department_id = 50;
```
Nếu bạn được yêu cầu tìm kết quả này bằng một câu lệnh chứ không phải hai, câu trả lời là dùng truy vấn con. Với sự giúp đỡ của một truy vấn con, bạn có thể kết hợp hai phát biểu thành một để tạo ra kết quả tương tự.
```
SELECT first_name, job_id, department_id
FROM Employees
WHERE department_id =
(SELECT department_id
FROM Employees
WHERE first_name='Adam')
ORDER BY first_name;
```
Trong phát biểu này, truy vấn đầu tiên (truy vấn con) được thực thi trước và tạo kết quả: 50. Truy vấn thứ hai (truy vấn chính) sau đó được xử lý và sử dụng giá trị (50), được trả về bởi truy vấn con, để hoàn thành mệnh đề WHERE.
Lưu ý rằng truy vấn con được sử dụng trong ví dụ này được gọi là một truy vấn con một dòng, vì nó trả về chỉ một dòng. Trong các truy vấn con như vậy, bạn chỉ có thể sử dụng toán tử so sánh đơn dòng.

- Xử lý truy vấn con nhiều dòng
Các truy vấn con được sử dụng để tìm dữ liệu có các giá trị không xác định, giống như phần được sử dụng trong ví dụ trước để tìm số phòng của Adam. Trong ví dụ đó, truy vấn con trả về một giá trị đơn (số hiệu phòng ban của Adam). Chạy truy vấn tương tự với Jennifer, bạn sẽ gặp phải lỗi *single-row subquery returns more than one row*. Truy vấn con thất bại vì nó tìm thấy hai nhân viên có cùng tên: Jennifer Dilly trong phòng ban 50, và Jennifer Whalen trong phòng ban 10.
Có hai cách để tránh lỗi này.
+ Sửa đổi mệnh đề WHERE như sau:
```
WHERE first_name = 'Jennifer' AND last_name = 'Dilly' (nếu bạn quan tâm phòng ban 50).
```
+ Sử dụng một toán tử so sánh đa dòng, chẳng hạn như IN
```
SELECT first_name||' '||last_name, job_id, department_id
FROM Employees
WHERE department_id IN
(SELECT department_id
FROM Employees
WHERE first_name='Jennifer')
ORDER BY first_name;
```
Cách thứ nhất chỉ lấy thông tin của các nhân viên làm việc theo phòng 50, trong khi cách 2 cũng trả về cả bản ghi của Jennifer Whalen làm việc tại phòng ban 10. Trong trường hợp này, truy vấn con trả lại hai phòng ban (10 và 50).

- Sử dụng nhiều truy vấn con trong một phát biểu SELECT
Bạn có thể định nghĩa nhiều truy vấn con trong một truy vấn lồng bằng cách nối nhiều điều kiện chỉ định sử dụng toán tử logic AND và OR. Ví dụ, lấy tên của tất cả nhân viên có cùng JOB_ID như Jennifer Dilly và làm việc trong cùng phòng ban với cô ấy. Phát biểu sẽ sử dụng hai truy vấn con, nối với nhau bằng cách sử dụng toán tử AND và sẽ được cấu trúc như sau.
```
SELECT first_name||' '||last_name, job_id, department_id
FROM Employees
WHERE department_id IN
(SELECT department_id
FROM Employees
WHERE first_name='Jennifer')
AND job_id =
(SELECT job_id
FROM Employees
WHERE first_name='Jennifer' AND last_name='Dilly')
ORDER BY first_name;
```

- Truy vấn con trong mệnh đề HAVING
Bạn đã thêm các truy vấn con vào mệnh đề WHERE trong hai ví dụ trước. Truy vấn con cũng có thể được sử dụng trong mệnh đề HAVING. Ví dụ, truy vấn thứ nhất lấy danh sách lương trung bình tại mỗi phòng. Truy vấn này giúp hiểu được kết quả của truy vấn thứ hai, hiển thị tất cả các phòng ban có mức lương trung bình cao hơn phòng ban 80. Nó sử dụng một truy vấn con trong mệnh đề HAVING.
```
SELECT department_id, round(avg(salary))
FROM Employees
```

```
GROUP BY department_id
ORDER BY department_id;

SELECT department_id, round(avg(salary))
FROM Employees
GROUP BY department_id
HAVING avg(salary) >
(SELECT avg(salary)
FROM Employees
WHERE department_id=80)
ORDER BY department_id;
```

Từ truy vấn thứ nhất, bạn có thể thấy rằng mức lương trung bình của phòng ban 80 là 8956. Đây là số được chuyển vào truy vấn chính để hoàn thành điều kiện HAVING.

6. Truy vấn dữ liệu từ nhiều bảng

Tính toàn vẹn tham chiếu định nghĩa mối quan hệ giữa các cột và bảng khác nhau trong một cơ sở dữ liệu quan hệ. Nó được gọi là toàn vẹn tham chiếu vì các giá trị trong một cột hoặc tập các cột trong một bảng tham chiếu đến hoặc phải so trùng với các giá trị trong một cột hoặc tập các cột có liên quan trong các bảng khác.
Ví dụ, bảng Departments và Employees có thông tin liên quan. Bảng Departments lưu trữ thông tin về các phòng ban khác nhau. Bản ghi của mỗi phòng ban có một DEPARTMENT_ID. Bảng Employees lưu trữ thông tin của nhân viên cùng với phòng ban của họ. Bảng Employees có một cột (DEPARTMENT_ID) để chỉ phòng ban của mỗi nhân viên. Như bạn thấy, cột DEPARTMENT_ID trong Employees tham chiếu đến cột DEPARTMENT_ID trong Departments. Tính toàn vẹn tham chiếu đơn giản là đảm bảo rằng mọi lúc, bất cứ khi nào bản ghi của nhân viên mới được tạo, nó phải có DEPARTMENT_ID so trùng với một DEPARTMENT_ID trong bảng Departments.

- JOIN

Bất cứ khi nào bạn cần dữ liệu từ nhiều bảng trong cơ sở dữ liệu, bạn sử dụng JOIN (phép kết). Phép kết là một trong những tính năng quan trọng nhất trong SQL, có các loại chính sau đây:
+ Equijoin
+ Inner Join
+ Outer Join
Những điểm cần nhớ:
+ Cơ sở dữ liệu thực hiện một phép kết bất cứ khi nào nhiều bảng xuất hiện trong mệnh đề FROM của truy vấn.
+ Danh sách SELECT của truy vấn có thể chọn bất kỳ cột nào từ bất kỳ bảng nào trong các bảng này. Các cột trong các điều kiện kết cũng không cần phải xuất hiện trong danh sách SELECT.
+ Nếu hai trong số các bảng này có tên cột chung, bạn phải cung cấp đầy đủ tên để tham chiếu đến các cột này (bằng tiền tố tên bảng) trong suốt truy vấn để tránh nhầm lẫn chúng.
+ Để kết các bảng với nhau, bạn cần số điều kiện tham gia tối thiểu bằng số bảng trừ đi 1. Ví dụ, kết bốn bảng, cần tối thiểu ba điều kiện kết.
+ Không giới hạn cho số lượng các bảng có thể được chỉ định trong phát biểu SELECT để tạo kết. Tuy nhiên, tất cả các bảng phải được liệt kê trong mệnh đề FROM, và mối quan hệ của chúng được định nghĩa trong mệnh đề WHERE.
+ Mệnh đề WHERE chứa điều kiện kết cũng có thể chứa các điều kiện khác tham chiếu các cột chỉ trong một bảng. Những điều kiện này có thể lọc thêm các dòng trả về bởi truy vấn kết.

- EQUIJOIN

Equijoin là một trong những phép kết được sử dụng phổ biến nhất với một điều kiện kết có chứa một toán tử (=). Đó là phép kết đơn giản nhất được sử dụng trong mệnh đề FROM hoặc trong mệnh đề WHERE của phát biểu SELECT để hiển thị dữ liệu từ nhiều bảng liên quan.

```
SELECT table.column, table.column …
FROM table1, table2
WHERE table1.column = table2.column;
```

Chú ý: sử dụng tên bảng đầy đủ với các tên cột đủ tiêu chuẩn có thể mất thời gian, đặc biệt là với tên bảng dài. Bạn có thể khắc phục vấn đề này bằng cách sử dụng bí danh (alias) cho bảng như sau:

```
SELECT e.first_name, d.department_name
FROM employees e, departments d;
```

Phát biểu sau đây tương tự như tất cả các phát biểu SELECT trước ngoại trừ hai điều. Trước hết, phát biểu này mang hai cột từ bảng Employees và một cột từ bảng Departments, so với các ví dụ trước đó, nơi tất cả các cột được liệt kê chỉ từ một bảng. Thứ hai, mệnh đề FROM liệt kê hai bảng. Đây là những bảng đang được kết trong truy vấn bằng cách sử dụng mệnh đề WHERE chỉ thị cho DBMS so trùng DEPARTMENT_ID trong bảng Employess với DEPARTMENT_ID trong bảng Departments. Một điều nữa khác với các ví dụ trước là việc sử dụng các tên bảng trong các mệnh đề SELECT và WHERE. Biểu thức này được gọi là các tên cột đủ điều kiện (fully qualified column name) và được sử dụng để thông báo cho DBMS chính xác cột DEPARTMENT_ID mà bạn đang tham chiếu. Luôn luôn bắt buộc sử dụng các định danh như vậy để tránh nhầm lẫn do sự hiện diện của DEPARTMENT_ID trong cả hai bảng.

```
SELECT employees.first_name, employees.department_id, departments.department_name
FROM employees, departments
WHERE employees.department_id=departments.department_id;
```

- INNER JOIN

INNER JOIN tương tự như EQUIJOIN và sử dụng toán tử bằng để thiết lập phép kết giữa hai bảng. Sự khác biệt là nó sử dụng từ khóa INNER JOIN trong cú pháp của nó để chỉ định tường minh phép kết. Từ khóa được sử dụng trong FROM làm cho nó khác với ví dụ trước.

Một sự khác biệt giữa hai phép kết là việc sử dụng mệnh đề ON đặc biệt thay vì WHERE để truyền điều kiện kết tương tự như được chỉ định trong mệnh đề WHERE trước đó. Bạn có thể sử dụng cả hai định dạng cú pháp đơn giản (EQUIJOIN) và chuẩn (INNER JOIN) để tạo ra kết quả tương tự. Tuy nhiên, chuẩn ANSI SQL thích INNER JOIN hơn EQUIJOIN.

```
SELECT table.column, table.column
FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

hoặc

```
SELECT table.column, table.column
FROM table1 JOIN table2
ON table1.column = table2.column;
```

INNER JOIN cũng trả về tất cả các dòng từ các bảng được chỉ định miễn là có một so trùng giữa các cột. Nếu có một số dòng trong bảng DEPARTMENTS không so trùng trong EMPLOYEES, các phòng ban này sẽ không được lấy. Ví dụ: bảng DEPARTMENTS có thêm một số bản ghi phòng khác (120 đến 270) mà không có một bản ghi liên quan trong bảng EMPLOYEES, thì các bản ghi này được loại bỏ bởi truy vấn INNER JOIN.

```
SELECT D.department_id "ID", D.department_name "Name", E.first_name "Employee"
FROM departments D INNER JOIN employees E
ON D.department_id=E.department_id
ORDER BY d.department_id;
```

- OUTER JOIN

Trong hai bài tập trước, những dòng được DBMS trả về phải thỏa mãn điều kiện kết (e.department_id = d.department_id), không hiển thị các phòng ban có số hiệu từ 120 đến 270 vì không có nhân viên nào trong bảng Employees có trong các phòng ban đó. Để nhận được các dòng bị thiếu này, bạn dùng OUTER JOIN trong điều kiện kết. Với Oracle, OUTER JOIN thể hiện bởi một dấu cộng (+) nằm trong cặp ngoặc và được đặt bên kết thiếu thông tin. Toán tử này tạo ra dòng NULL từ bảng thiếu để các dòng từ bảng không thiếu có thể kết đến.

```
SELECT table.column, table.column
FROM table1, table2
ON table1.column = table2.column(+);
```

hoặc

```
SELECT table.column, table.column
FROM table1, table2
ON table1.column(+) = table2.column;
```

```
SELECT D.department_id "ID", D.department_name "Name", E.first_name "Employee"
FROM departments D INNER JOIN Employees E ON
D.department_id=E.department_id(+)
ORDER BY d.department_id;
```

Một số cài đặt sử dụng từ khóa LEFT JOIN và RIGHT JOIN để tạo các OUTER JOIN. Truy vấn với LEFT JOIN lấy tất cả các dòng từ bảng bên trái (table1), với các dòng so trùng với bảng bên phải (table2). Khi không so trùng, kết quả bên phải sẽ hiển thị NULL. RIGHT JOIN thì ngược lại, hiển thị NULL với các dòng không so trùng với bảng bên trái. Trong một số cài đặt, chúng còn được gọi là LEFT OUTER JOIN và RIGHT OUTER JOIN.

Lưu ý: SQLite không có RIGHT OUTER JOIN. Nếu bạn cần phép kết này, đơn giản đảo ngược thứ tự các bảng định nghĩa trong mệnh đề FROM và WHERE.

- JOIN so với truy vấn con

Dữ liệu mà bạn truy xuất bằng phép kết cũng có thể được truy xuất bằng truy vấn con. Tuy nhiên, các truy vấn lồng nhau hoặc các truy vấn với các truy vấn con được lồng nhau thực hiện chậm hơn truy vấn kết nhiều bảng.

7. Tạo bảng

Bảng là đối tượng cơ bản trong cơ sở dữ liệu được tạo ra để lưu dữ liệu. Mỗi bảng trong một cơ sở dữ liệu chứa thông tin cho một loại cụ thể. Ví dụ, bảng Employees, chứa một danh sách nhân viên cùng với thông tin về mỗi nhân viên. Một cơ sở dữ liệu có thể có nhiều bảng để lưu trữ thông tin về từng loại riêng biệt. Ví dụ, Departments là một bảng trong cơ sở dữ liệu chứa thông tin cụ thể của từng phòng ban.

Ngoài việc riêng biệt, bảng cũng có thể liên quan đến nhau bằng cách sử dụng các khóa chỉ định, được gọi là khóa chính (primary key) và khóa ngoại (foreign key). Ví dụ, bảng Employees liên quan đến bảng Departments sử dụng cột DEPARTMENT_ID. Cột này được tạo ra như một khóa ngoại trong bảng Employees, sẽ tham chiếu đến cột cùng tên trong bảng Departments, nơi nó hoạt động như một khóa chính. Loại quan hệ này giúp thu thập dữ liệu từ nhiều bảng (sử dụng JOIN), như bạn đã thấy trong phần trước.

Một bảng là đơn vị cơ bản của tổ chức dữ liệu trong một cơ sở dữ liệu. Mỗi bảng có định nghĩa riêng bao gồm tên bảng và tập hợp các cột. Một cột xác định một thuộc tính của thực thể được mô tả bởi bảng. Ví dụ: cột EMPLOYEE_ID trong bảng Employees tham chiếu đến thuộc tính định danh của một thực thể nhân viên.

Khi bạn tạo một bảng, bạn chỉ định các cột mà bảng sẽ có và cung cấp một tên cho mỗi cột, xác định kiểu dữ liệu và chiều rộng cho mỗi cột. Bạn phải chỉ định một kiểu dữ liệu cho từng cột trong bảng. Ví dụ: kiểu dữ liệu cho EMPLOYEE_ID là NUMBER(6), chỉ định rằng cột này chỉ có thể chứa dữ liệu số lên đến 6 chữ số chiều rộng. Chiều rộng (width) có thể được xác định trước bởi loại dữ liệu, như với DATE.

Sau khi tạo một bảng, bạn có thể chèn, cập nhật, xóa, và truy vấn các dòng bằng cách sử dụng SQL. Một dòng là tập hợp các thông tin cột tương ứng với một bản ghi trong một bảng. Ví dụ: một dòng trong bảng Employees mô tả các thuộc tính của một nhân viên cụ thể.

Trước tiên, bạn phải làm quen với các kiểu dữ liệu được khai báo cho mỗi cột trong quá trình tạo bảng.

- Kiểu dữ liệu

Mỗi giá trị mà bạn lưu trữ trong các bảng cơ sở dữ liệu có một kiểu dữ liệu liên kết với một tập các thuộc tính cố định. Các thuộc tính này giúp cơ sở dữ liệu xử lý các giá trị của một kiểu dữ liệu này khác với các giá trị của một kiểu dữ liệu khác. Khi một bảng được tạo, tất cả các cột của nó được chỉ định các kiểu dữ liệu tương ứng. Các kiểu dữ liệu này xác định miền giá trị mà mỗi cột có thể chứa. Ví dụ, cột DATE không thể chấp nhận giá trị 'ABCD' hoặc ngày 31 tháng 11. Tương tự, một cột số cũng không chấp nhận các giá trị ký tự trong đó.
Các kiểu dữ liệu cũng khác với các DBMS khác nhau. Ngay cả tên của một kiểu dữ liệu trong một DBMS cũng có thể có nghĩa khác trong DBMS khác. Bạn cần xem tài liệu chi tiết về các kiểu dữ liệu được hỗ trợ trên DBMS của bạn.
Một điều quan trọng khác mà bạn phải quan tâm khi thiết kế bảng là việc sử dụng kiểu dữ liệu phù hợp. Bạn có thể bị hậu quả nghiêm trọng trong tương lai nếu bạn chọn sai kiểu dữ liệu. Mặc dù bạn có thể thay đổi các kiểu dữ liệu sau đó, nó cũng có thể gây mất dữ liệu.
Bảng sau liệt kê một số kiểu dữ liệu phổ biến nhất.

Kiểu dữ liệu	Mô tả
CHAR(size)	Lưu trữ dữ liệu ký tự có độ dài cố định, có kích thước size. Ví dụ, cột COUNTRY_ID, trong bảng Countries, được định nghĩa là CHAR(2), có nghĩa là bạn không thể nhập hơn 2 ký tự trong cột này. Trong Oracle, kích thước tối đa của kiểu dữ liệu này là 2000 ký tự. Kích thước mặc định và tối thiểu là 1 ký tự.
VARCHAR2(size)	Lưu các chuỗi ký tự có độ dài thay đổi. Trong một số cài đặt nó được định nghĩa là VARCHAR. Trong Oracle, bạn phải chỉ định một size cho loại dữ liệu này, kích thước tối đa là 4000 ký tự, và tối thiểu là 1 ký tự.
NUMBER[(p[,s])]	Lưu trữ các số có độ chính xác (precision) p và quy mô (scale) s. Trong Oracle, p có thể dao động từ 1 đến 38, trong khi s có thể dao động từ -84 đến 127. Kích thước mặc định của nó là 38. Có thêm một số kiểu dữ liệu số được hỗ trợ bởi các DBMS. Ví dụ: Decimal, Int, Real, Money, Currency, v.v..
DATE	Lưu trữ ngày tháng, kiểu dữ liệu này nằm trong khoảng từ ngày 1 tháng 1 năm 4712 BC đến 31 tháng 12 năm 9999 AD, và chứa các thành phần YEAR, MONTH, DAY, HOUR, MINUTE và SECOND.
BLOB	Binary Large Object, là một kiểu dữ liệu có thể chứa một lượng lớn dữ liệu. BLOB rất tiện dụng để lưu trữ thông tin số hóa (hình ảnh, âm thanh và video).

Lưu ý: các kiểu CHAR và VARCHAR2 lưu trữ chuỗi ký tự, nhưng khác nhau về cách chúng được lưu trữ và lấy ra. Kiểu dữ liệu CHAR chấp nhận một số ký tự cố định, trong khi VARCHAR2 chấp nhận văn bản có độ dài thay đổi khác nhau. Do đó, nếu chuỗi 'Donald Trump' được lưu trong cột được chỉ định là CHAR(30), 30 ký tự đầy đủ được lưu trữ, và văn bản được đệm với space. Trong VARCHAR2, chỉ dữ liệu nhập được lưu mà không có dữ liệu bổ sung. Nếu bạn gán một giá trị cho một cột CHAR hoặc VARCHAR2 vượt quá chiều dài tối đa của cột, giá trị được cắt ngắn.

- CREATE TABLE

Các bảng trong cơ sở dữ liệu được tạo ra bằng cách tạo lệnh CREATE TABLE, là một trong các lệnh định nghĩa ngôn ngữ dữ liệu (DDL). Nhắc lại, lệnh DDL là tập hợp các phát biểu SQL được sử dụng để tạo, sửa đổi và loại bỏ các đối tượng cơ sở dữ liệu. Lệnh này có cú pháp rất dài, tuy nhiên sau đây là phiên bản ngắn và đơn giản để bắt đầu.

```
CREATE TABLE [schema.]{table_name}
(column datatype [DEFAULT value][[CONSTRAINT constraint_name][REFERENCES] constraint_type].
...
...
[table_constraint]);
```

Giải thích

schema	Lược đồ là một nơi hợp lý trong cơ sở dữ liệu để đặt các bảng mới. Nó là một mệnh đề tùy chọn. Theo mặc định, bảng được tạo ra trong lược đồ bạn hiện đang kết nối.
table_name	Đây là một mệnh đề bắt buộc, bạn dùng định nghĩa một tên cho bảng mới. Theo các quy tắc được nêu dưới đây để đặt tên bảng cơ sở dữ liệu: + Tên bảng và tên cột bắt đầu bằng ký tự chữ cái (A-Z hoặc a-z). + Tên có thể dài 1-30 ký tự. + Tên chỉ chứa các ký tự (A-Z hoặc a-z), số (0-9), hoặc ba ký tự đặc biệt (_, $ và #). + Tên bảng phải không trùng lặp tên của bảng khác trong cùng lược đồ. Tương tự, các cột trong cùng một bảng không được sử dụng tên giống nhau. Tuy nhiên, các cột trong các bảng khác nhau có thể có tên giống nhau; tên bảng có thể giống nhau trong các lược đồ khác nhau. + Không bao giờ sử dụng từ dành riêng cho cơ sở dữ liệu (SELECT, USER, ...) để đặt tên bảng. + Sử dụng cùng một thực thể cột trong các bảng khác nhau để dễ dàng tham chiếu. Ví dụ: cột DEPARTMENT_ID trong các bảng Departments, Employees và Job_History. + Tên bảng và cột không phân biệt chữ hoa chữ thường, EMPLOYEES, employees, và Employees cũng giống nhau.
column	Tên của cột. Thực hiện theo các hướng dẫn được cung cấp ở trên cho tên bảng.
datatype	Loại dữ liệu của cột với chiều dài mong muốn.
DEFAULT value	Mệnh đề tùy chọn, định nghĩa một giá trị mặc định cho một cột. Định nghĩa giá trị được thêm vào bảng khi giá trị của cột tương ứng bị bỏ qua tại thời điểm tạo ra bản ghi.
constraint	Những ràng buộc thường được định nghĩa trong khi tạo bảng và có thể đặt vào ở cấp cột hoặc cấp bảng.

- Constraints

Để giữ cơ sở dữ liệu ở trạng thái nhất quán, bạn cần một quy trình đảm bảo rằng chỉ có *dữ liệu hợp lệ* mới được lưu trữ vào bảng. Quy trình này được thực hiện bằng các ràng buộc toàn vẹn - các quy tắc ràng buộc các giá trị trong một cơ sở dữ liệu. Các ràng buộc được áp đặt trên các bảng để thực thi toàn vẹn tham chiếu ngăn chặn xảy ra mất nhất quán dữ liệu.

Hai ưu điểm chính của việc cài đặt ràng buộc là:
+ Thi hành các quy tắc ở cấp độ bảng bất cứ khi nào một bản ghi được chèn vào, cập nhật hoặc bị xóa. Để thực hiện ba hoạt động này thành công, các ràng buộc phải được thỏa mãn.
+ Ngăn chặn việc xóa bảng và bản ghi nếu có bảng và bản ghi phụ thuộc. Những ràng buộc như vậy rất hữu ích để tránh việc xóa do vô ý.
Hầu hết các cơ sở dữ liệu cho phép bạn tạo ra năm loại ràng buộc sau và cho phép bạn khai báo chúng theo hai cách.
+ Như một phần của định nghĩa của một cột riêng, gọi là *đặc tả nội tuyến* (inline specification). Ràng buộc sẽ được *kiểm tra trên cột dữ liệu*.
+ Như một phần của định nghĩa bảng, gọi là *đặc tả ngoại tuyến* (out-of-line specification). Ràng buộc sẽ được *kiểm tra trên bảng*.
Bảng dưới đây trình bày một số loại ràng buộc chính, được hỗ trợ bởi tất cả các DBMS phổ biến. Lưu ý rằng các ràng buộc thường được định nghĩa khi các bảng được tạo. Nếu cần, bạn cũng có thể thi hành chúng ở giai đoạn sau. Tuy nhiên, khuyến nghị là nên lên kế hoạch và triển khai chúng trong giai đoạn đầu, trước khi đưa dữ liệu vào các bảng có liên quan.

Ràng buộc	Mô tả
NOT NULL	Cấm giá trị cột của bảng là NULL. Các cột không có ràng buộc NOT NULL có thể chứa các giá trị NULL theo mặc định. Ràng buộc NOT NULL phải được khai báo nội tuyến. Tất cả các ràng buộc khác có thể được khai báo hoặc nội tuyến hoặc ngoại tuyến. Ví dụ, tạo ràng buộc NOT NULL cho cột PHONE_NUMBER, không nêu rõ tên ràng buộc. CREATE TABLE Employees (..., phone_number varchar2(20) NOT NULL, ...) Ví dụ, ràng buộc NOT NULL được áp dụng cho cột FIRST_NAME. Trong trường hợp này, tên ràng buộc được cung cấp *emp_fname_nn*, viết tắt của *employee first_name not null*. Cung cấp thêm tên ràng buộc có ý nghĩa giúp dễ tham chiếu trong tương lai. CREATE TABLE Employees (..., first_name varchar2 (20) CONSTRAINT emp_fname_nn NOT NULL, ...);
UNIQUE	Kiểm tra tính duy nhất của các cột không tham gia khóa chính. Cấm nhiều dòng có cùng giá trị trong cùng một cột, và cho phép các giá trị NULL nếu nó được dựa trên một cột đơn. Bạn cũng có thể tạo ra khóa phức duy nhất từ việc kết hợp của các cột như là khóa duy nhất. Bạn phải định nghĩa một khóa phức duy nhất là ngoại tuyến. Bạn không thể chỉ định cùng một cột hoặc tập các cột làm vừa là khóa chính vừa là khóa duy nhất. Cột PHONE_NUMBER trong bảng Employees là một cột ứng cử cho ràng buộc này, bởi vì mỗi nhân viên có một số điện thoại duy nhất. Bằng cách làm cho cột này trở nên duy nhất, bạn có thể tránh trùng lặp các giá trị trong cột, thường phát sinh từ lỗi vô tình nhập trùng. Định nghĩa một ràng buộc UNIQUE nội tuyến: CREATE TABLE Employees (..., phone_number varchar2(20) CONSTRAINT emp_phone_uk UNIQUE, ...); Ví dụ sau định nghĩa một khóa phức duy nhất ngoại tuyến trên tổ hợp các cột EMPLOYEE_ID và PHONE_NUMBER. Ràng buộc *emp_id_phone_uk* đảm bảo rằng tổ hợp EMPLOYEE_ID và PHONE_NUMBER không xuất hiện trong bảng hai lần. CREATE TABLE Employees (employee_id number(6), ..., phone_number varchar2(20), hire_date date, ..., CONSTRAINT emp_id_phone_uk UNIQUE (employee_id, phone_number)); Thoạt nhìn, các ràng buộc UNIQUE trông giống ràng buộc PRIMARY KEY, nhưng nó không giống hoặc đồng nghĩa với ràng buộc PRIMARY KEY. Nó khác với khóa chính các điểm sau: + Bạn có thể định nghĩa nhiều cột UNIQUE trong một bảng, nhưng một bảng chỉ có một và chỉ một khóa chính. + Các cột được định nghĩa ràng buộc UNIQUE có thể có các giá trị NULL; các cột khóa chính không thể. + Một khóa chính có thể được định nghĩa như một khóa ngoại trong các bảng khác; bạn không thể tạo mối quan hệ như vậy cho các cột UNIQUE. + Bạn có thể sửa đổi các giá trị cột UNIQUE trong một bảng; các giá trị được lưu trữ trong PRIMARY KEY không thể sửa đổi được.
PRIMARY KEY	Xác định duy nhất mỗi dòng trong một bảng. Bạn sử dụng nó để kết hợp hai ràng buộc trong một khai báo. Khóa chính không thể là NULL và không cho phép trùng lặp. Ràng buộc này về cơ bản được áp dụng để đảm bảo rằng các tác vụ DML khác (như UPDATE và DELETE) được thực hiện thành công. Thiếu khóa chính, hai lệnh này sẽ không bao giờ xác định được dòng mà người dùng có ý định thao tác. Ràng buộc khóa chính có thể được định nghĩa nội tuyến hoặc ngoại tuyến. Khóa chính phức phải được định nghĩa ngoại tuyến. Một ràng buộc khóa chính thường được định nghĩa nội tuyến khi nó được dựa trên một cột: CREATE TABLE Employees (employee_id number(6) CONSTRAINT emp_id_pk PRIMARY KEY,...) Nếu nó được dựa trên nhiều cột, thì bạn phải định nghĩa ràng buộc này ngoại tuyến. Ràng buộc này, được gọi là khóa chính phức (composite primary key), đảm bảo rằng một nhân viên không được đăng ký lại cùng một phòng ban. CREATE TABLE Employees (employee_id number(6), ... , department_id number(4), CONSTRAINT emp_id_pk PRIMARY KEY (employee_id, department_id));
FOREIGN KEY	Loại ràng buộc toàn vẹn này cho phép chúng tra có thể kiểm tra tính tồn tại của dữ liệu (khóa ngoại), bắt buộc phải có trong một bảng khác (gọi là bảng tham chiếu). Điều này ngăn cản việc người dùng nhập một giá trị dữ liệu không có trong một bảng dữ liệu khác. Ràng buộc KEY FOREIGN chỉ định một cột hoặc một tập các cột như là một khóa ngoại. Nó được tạo ra để thiết lập và thực thi các mối quan hệ giữa các bảng. Hãy nhớ các điểm chính sau đây liên quan đến khóa ngoại: + Bảng chứa khóa ngoại được gọi là bảng con, và bảng có chứa khóa tham chiếu được gọi là bảng cha (hay bảng tham chiếu). Một trị khóa ngoại phải so trùng với một trị tồn tại trong bảng cha. + Các cột của khóa ngoại và khóa được tham chiếu phải so trùng theo thứ tự và kiểu dữ liệu.

	+ Bạn có thể định nghĩa ràng buộc khóa ngoại trên một cột khóa đơn, nội tuyến hay ngoại tuyến; khóa ngoại phức phải được chỉ định thuộc tính ngoại tuyến. + Bạn có thể định nghĩa nhiều khóa ngoại trong một bảng. Ngoài ra, một cột đơn có thể tham gia nhiều khóa ngoại. + Vì không có phần nào của khóa chính có thể NULL, do đó, khóa ngoại là một phần của khóa chính cũng không thể NULL. Các mệnh đề sau đây được sử dụng khi bạn xác định các ràng buộc khóa ngoại: FOREIGN KEY từ khóa này chỉ được sử dụng khi bạn định nghĩa ràng buộc ngoại tuyến. REFERENCES được sử dụng với cả khai báo nội tuyến và ngoại tuyến để xác định bảng cha và cột tham chiếu của nó. ON DELETE CASCADE bên cạnh việc thực thi toàn vẹn tham chiếu, khóa ngoại được định nghĩa để ngăn việc vô tình xóa các bản ghi trong bảng cha. Ví dụ, bạn không thể xóa một phòng ban đang có nhân viên. Bản ghi của một phòng ban có thể bị xóa hoặc bằng cách xóa tất cả các bản ghi của các nhân viên thuộc phòng ban đó, hoặc bằng cách sử dụng tùy chọn ON DELETE CASCADE. Nó được sử dụng khi bạn muốn loại bỏ một bản ghi cha kéo theo xóa tất cả các bản ghi con cùng một lúc. Nếu không có tùy chọn này, dòng trong bảng cha không thể xóa nếu nó được tham chiếu đến trong bảng con. Phát biểu sau định nghĩa một khóa ngoại trên cột DEPARTMENT_ID tham chiếu khóa chính trên cột DEPARTMENT_ID trong bảng Departments: CREATE TABLE Employee (employee_id number(4), … , department_id CONSTRAINT emp_deptno_fk REFERENCES Departments(department_id)); Ràng buộc emp_deptno_fk đảm bảo rằng một phòng ban được gán cho một nhân viên trong bảng Employees là có mặt trong bảng Departments. Tuy nhiên, nhân viên có thể có DEPARTMENT_ID là NULL, có nghĩa là họ không liên quan với phòng ban nào. Nếu yêu cầu phòng ban quản lý họ là bắt buộc, bạn có thể tạo một ràng buộc NOT NULL trên cột DEPARTMENT_ID trong bảng Employees, bổ sung cho ràng buộc REFERENCES. Cũng lưu ý rằng ví dụ trên không sử dụng mệnh đề FOREIGN KEY, bởi vì ràng buộc được xác định nội tuyến. Hơn nữa, trong Oracle, kiểu dữ liệu của cột DEPARTMENT_ID cũng không cần thiết vì nó được thực hiện tự động bởi Oracle sử dụng kiểu dữ liệu của khóa được tham chiếu. Phát biểu sau xác định ràng buộc khóa ngoại ngoại tuyến: CREATE TABLE Employee (employee_id number(4), … , department_id, CONSTRAINT emp_deptno_fk FOREIGN KEY (department_id) REFERENCES Departments(department_id)); Cả hai ví dụ trên đã bỏ qua ON DELETE CASCADE, chỉ thị cho DBMS không xóa một phòng ban nếu có ít nhất một nhân viên có liên quan đến nó.
CHECK	Ràng buộc này được áp dụng trên các cột của bảng để đảm bảo rằng giá trị được lưu trữ tuân thủ với điều kiện chỉ định (kiểm tra miền giá trị). Điều này ngăn cản việc một người dùng nhập một giá trị vượt khỏi miền giá trị quy định. Ràng buộc CHECK sử dụng cùng một cú pháp cho cả đặc tả nội tuyến và ngoại tuyến. Tuy nhiên, đặc tả nội tuyến có thể tham khảo chỉ cho cột hiện đang được xác định, trong khi đặc tả ngoại tuyến có thể tham khảo nhiều cột. Phát biểu sau tạo một bảng và định nghĩa một ràng buộc CHECK trong mỗi cột của bảng: CREATE TABLE Departments (department_id NUMBER CONSTRAINT dept_no_chk CHECK (department_id BETWEEN 10 AND 99), department_name VARCHAR2(30) CONSTRAINT dept_name_chk CHECK (department_name = UPPER(department_name)), location VARCHAR2(10) CONSTRAINT dept_loc_chk CHECK (location IN ('SEATTLE', 'TORONTO', 'TOKYO', 'LONDON'))); Mục đích của tất cả ba ràng buộc được định nghĩa trong bảng ở trên là ràng buộc miền giá trị trong cột tương ứng. *dept_no_chk* đảm bảo rằng không có DEPARTMENT_ID nhỏ hơn 10 hoặc lớn hơn 99. *dept_name_chk* đảm bảo rằng tất cả các tên DEPARTMENT_NAME đều là chữ hoa. *dept_loc_chk* giới hạn LOCATION chỉ trong danh sách các thành phố Seattle, Toronto, Tokyo hoặc London. Phát biểu dưới đây tạo ra một bảng với một ràng buộc CHECK ngoại tuyến: CREATE TABLE Employees (… , salary NUMBER(8,2), commission_pct NUMBER(5,2), … , CONSTRAINT emp_sal_chk CHECK (salary * commission_pct <= 5000)); Ràng buộc *emp_sal_chk* thực hiện một điều kiện đặt một biên trên 5000 để giới hạn tổng hoa hồng của mỗi nhân viên bằng cách so sánh tích của SALARY và COMMISSION_PCT với biên chỉ định. Nếu bạn nhập một bản ghi mới trong bảng này, thì tích này được kiểm soát không vượt quá 5000 để phù hợp với ràng buộc. Mặt khác, nếu cột bất kỳ thuộc tích này có một giá trị NULL, thì tích tính toán cũng là NULL, tự động thỏa mãn ràng buộc.

- Tạo bảng

Bây giờ, giả định bạn không chỉ nắm bắt các khái niệm cơ bản về ràng buộc, mà còn quen thuộc với việc sử dụng lệnh CREATE TABLE. Ví dụ sau tạo ra một bảng và thực hiện tất cả năm ràng buộc trên đó.

```
CREATE TABLE department_clone
(
```

```
  department_id number(4) CONSTRAINT pk_dept_id PRIMARY KEY,
  department_name varchar2(3) CONSTRAINT nn_dept_name NOT NULL,
  location_id number(4) CONSTRAINT fk_dept_loc REFERENCES locations (location_id),
  CONSTRAINT uk_dept_name_loc UNIQUE (department_name, location_id)
);
```

Giải thích

Ràng buộc	Giải thích
pk_dept_id	Đây là một ràng buộc nội tuyến xác định cột DEPARTMENT_ID là khóa chính của bảng department_clone. Bằng cách định nghĩa ràng buộc này, bạn loại bỏ khả năng trùng lặp số hiệu cho hai phòng ban khác nhau, và cũng tuyên bố rằng cột này không được chấp nhận các giá trị NULL.
nn_dept_name	Đây cũng là một ràng buộc nội tuyến, được định nghĩa để thực hiện ràng buộc NOT NULL. Nó đảm bảo rằng mỗi phòng ban trong bảng phải có một tên.
fk_dept_loc	Ràng buộc nội tuyến này được tạo ra để thực hiện ràng buộc khóa ngoại. Nó đảm bảo rằng bất kỳ LOCATION_ID nhập vào bảng này phải tồn tại trước trong bảng Locations. Lưu ý rằng trước khi định nghĩa ràng buộc khóa ngoại này bảng Locations phải tồn tại, với một ràng buộc khóa chính trên cột LOCATION_ID.
uk_dept_name_loc	Đây là một ràng buộc duy nhất được định nghĩa ngoại tuyến. Nó sử dụng hai cột từ bảng để tạo thành một khóa phức duy nhất, đảm bảo rằng tổ hợp DEPARTMENT_NAME và LOCATION_ID không xuất hiện trùng lặp trong bảng.

- Tạo bảng từ một bảng khác

Một tình huống thường gặp yêu cầu bạn phải tạo một bảng mới dựa trên một bảng hiện có. Không chỉ phải tạo bảng mới có cùng cấu trúc, bạn còn phải kết hợp tất cả các bản ghi từ bảng hiện tại vào bảng mới. SQL cung cấp một giải pháp cho kịch bản này, một phương pháp khác để tạo một bảng trong đó bạn sử dụng một truy vấn con với mệnh đề AS của CREATE TABLE để tạo bảng và chèn dòng vào nó.

```
CREATE TABLE {table name}
[(column specification, column specification, …)]
{AS subquery};
```

Hãy xem xét một số ví dụ để hiểu cú pháp trên.

```
CREATE TABLE emp2 AS SELECT * FROM Employees;
```

Phát biểu trên tạo ra một bản sao chính xác của bảng Employees. Nó tạo ra bảng mới (emp2) với cùng một cấu trúc và điền vào nó tất cả các bản ghi từ bảng nguồn. Chú ý rằng ví dụ này không sử dụng mệnh đề (column specification). Khi không cung cấp (column specification) thì tên các cột của bảng đích giống như tên các cột trong bảng nguồn. Cũng lưu ý rằng không có ràng buộc toàn vẹn liên quan đến bảng nguồn được kế thừa trong bảng đích.

```
CREATE TABLE emp3 (id, name, hiredate) AS SELECT employee_id,first_name,hire_date FROM Employees;
```

Trong phát biểu này bảng mới được tạo ra bằng cách sử dụng một số cột chỉ định từ bảng nguồn. Trong kiểu tạo bảng này, bạn có thể đặt tên cột trong bảng đích khác với tên cột trong bảng nguồn, nhưng số cột trong bảng mới phải bằng với số cột trong danh sách SELECT của truy vấn con.

```
CREATE TABLE emp4 AS SELECT * FROM Employees WHERE department_id=50;
```

Ở đây, bạn đã tạo bảng mới bằng cách sử dụng tất cả các cột (*) từ bảng nguồn. Tuy nhiên, mệnh đề WHERE được sử dụng để giới hạn việc chèn bản ghi theo điều kiện chỉ định, với hướng dẫn chèn bản ghi của nhân viên làm việc trong phòng ban 50.

Ghi chú, vì truy vấn con dựa trên phát biểu SELECT, bạn có thể sử dụng các mệnh đề khác của lệnh này, bao gồm WHERE và GROUP BY. Bạn cũng có thể sử dụng phép kết trong truy vấn con để chèn dữ liệu từ nhiều bảng. Không phân biệt số lượng các bảng nguồn được định nghĩa trong mệnh đề FROM, dữ liệu sẽ chỉ được lấy vào một bảng duy nhất.

- Thay đổi bảng

Sau khi tạo một bảng, bạn nhận thấy có điều gì đó không ổn với việc tạo bảng và bạn cần phải thay đổi định nghĩa của nó. Sau khi tạo một bảng bạn có thể sửa đổi cấu trúc của nó bằng cách sử dụng lệnh ALTER TABLE. Đây là một lệnh định nghĩa dữ liệu (DDL), cho phép bạn:

+ Thêm một cột mới vào bảng
+ Sửa đổi chiều rộng cột
+ Xóa một cột từ bảng
+ Đổi tên một cột
+ Đổi tên bảng
+ Thêm/xóa các ràng buộc
+ Bật/tắt các ràng buộc
+ Đổi tên các ràng buộc

Ghi chú: đặc biệt cẩn thận khi sử dụng các lệnh DDL vì các lệnh này không thể đảo ngược (rollback) được.

- Thêm cột

Hãy bắt đầu xử lý bằng cách thêm một cột mới vào một bảng bằng cách sử dụng mệnh đề ADD của lệnh ALTER TABLE. Cú pháp của lệnh ALTER TABLE rất giống với CREATE TABLE có thêm ba mệnh đề (ADD, MODIFY và DROP), như được hiển thị bên dưới.

```
ALTER TABLE {table_name}
ADD ({column datatype [DEFAULT value] [constraint_specification]}[, column … ,…]);
```

Ví dụ sau giả định bạn đã có bảng emp2 tạo trong phần trước.

```
ALTER TABLE emp2
ADD (ss_number char(9) CONSTRAINT emp_phone_uk UNIQUE );
```

Trong phát biểu trên, bạn đã thêm một cột mới để ghi lại số SSN (social security number) của nhân viên. Vì số này có định dạng cố định nên ta sử dụng loại dữ liệu CHAR. Mỗi nhân viên có một số SSN duy nhất, do đó, ta tạo ra một ràng buộc UNIQUE cho cột này, để loại bỏ các giá trị trùng lặp. Cũng lưu ý rằng ta cố ý cung cấp một tên ràng buộc sai (emp_phone_uk), sẽ bị bỏ trong phần tiếp theo.

- Sửa đổi cột

```
ALTER TABLE {table_name}
MODIFY ({column datatype [DEFAULT value] }[, column ... , ...]);
```

Sau khi thêm cột SS_NUMBER vào bảng Employees, bạn nhận ra rằng có vài lỗi trong phát biểu ALTER TABLE. Thứ nhất, nên đặt kích thước của cột này là 11 thay vì 9 để lưu số theo định dạng 999-99-9999 vì tính cả hai dấu gạch ngang. Thứ hai, tên của ràng buộc duy nhất cũng sai. Trong phát biểu sau, bạn sẽ giải quyết vấn đề thứ nhất bằng cách thay đổi kích thước chiều rộng cột bằng cách sử dụng mệnh đề MODIFY của lệnh.

```
ALTER TABLE emp2 MODIFY (ss_number char(11));
```

Luôn luôn nhớ các nguyên tắc sau đây khi bạn sửa đổi định nghĩa cột:

+ Giảm độ rộng của cột chỉ khi nó chứa các giá trị NULL.
+ Thay đổi kiểu dữ liệu nếu không có giá trị trong cột.
+ Giá trị DEFAULT có hiệu lực cho việc chèn thêm vào.
+ Sử dụng ràng buộc NOT NULL chỉ khi không có giá trị.

- Xóa cột

Bạn cũng được phép xóa các cột từ một bảng với điều kiện là chúng không chứa giá trị. Cú pháp ở đây cho thấy hai cách dùng mệnh đề này. Sử dụng cách thứ nhất nếu bạn muốn xóa một cột đơn. Đối với kịch bản này, từ khóa COLUMN phải tuân theo sau mệnh đề DROP, tiếp theo là tên cột. Áp dụng cách thứ hai nếu bạn cần phải xóa nhiều cột. Đối với kịch bản này, chỉ sử dụng mệnh đề DROP, sau đó là một danh sách các cột phân cách bằng dấu phẩy đóng trong cặp ngoặc ().

```
ALTER TABLE {table_name} {DROP COLUMN column_name};
```

hoặc

```
ALTER TABLE {table_name} {DROP (col_name1, col_name2, ...)};
```

Ví dụ

```
ALTER TABLE emp2 DROP COLUMN ss_number ;
```

Lưu ý, giảm một cột cũng sẽ giảm các ràng buộc phụ thuộc. Ví dụ, nếu bạn thực hiện thành công phát biểu trên, ràng buộc UNIQUE liên quan (emp_phone_uk) sẽ mất. Bạn có thể xác minh bằng cách click vào tab Constraints để xem.

- Đổi tên cột

Sau khi tạo một bảng, bạn có thể thay đổi tên của các cột của nó với các chú ý sau:

+ Tên mới không được mâu thuẫn với tên của bất kỳ cột nào hiện có trong bảng.
+ Các view, trigger, hàm và thủ tục phụ thuộc có thể trở nên không hợp lệ, và bạn có thể cần phải thay đổi các đối tượng này bằng tên cột mới.

```
ALTER TABLE {table_name}
RENAME COLUMN {old_column TO new_column};
```

Ví dụ sau đổi tên cột SALARY của bảng emp2 thành MONTHLY_SALARY.

```
ALTER TABLE emp2 RENAME COLUMN salary TO monthly_salary;
```

Ghi chú, Microsoft SQL Server thực hiện thủ tục sp_rename để thay đổi tên của một bảng. Ví dụ sau đổi tên bảng test1 thành test2 trong lược đồ HR.

```
EXEC sp_rename 'hr.test1', 'test2';
GO
```

- Đổi tên bảng

Sử dụng mệnh đề RENAME TO của lệnh ALTER TABLE để thay đổi tên của một bảng hiện có.

```
ALTER TABLE previous table name RENAME TO new table name;
```

Phát biểu sau đổi tên một bảng hiện có emp4 thành emp5.

```
ALTER TABLE emp4 RENAME TO emp5;
```

- Thêm ràng buộc sau khi tạo bảng

Để thêm một ràng buộc vào một bảng hiện có, bạn sử dụng mệnh đề ADD CONSTRAINT của lệnh ALTER TABLE.

```
ALTER TABLE {table_name}
{ADD CONSTRAINT [constraint_name] {constraint_type (column | expression)};
```

Các phát biểu sau chỉ ra cách thêm tất cả bốn ràng buộc vào một bảng sau khi nó được tạo ra. Phát biểu thứ ba cho biết thêm cột SS_NUMBER một lần nữa (vì nó đã bị bỏ trong phần trước), để thực hiện ràng buộc UNIQUE.

```
ALTER TABLE emp2 ADD CONSTRAINT pk_emp2_id PRIMARY KEY (employee_id) ;
ALTER TABLE emp2 ADD CONSTRAINT fk_emp2_dept_id FOREIGN KEY (department_id)REFERENCES Departments(department_id);
ALTER TABLE emp2 ADD (ss_number char(11)); -- tạo lại cột ss_number
ALTER TABLE emp2 ADD CONSTRAINT uk_emp2_ss_number UNIQUE (ss_number) ;
ALTER TABLE emp2 ADD CONSTRAINT ck_emp2_salary CHECK (salary > 0);
```

Ghi chú, SQLite chỉ hỗ trợ các mệnh đề RENAME TABLE và ADD COLUMN của lệnh ALTER TABLE. Các biến thể khác, chẳng hạn như DROP COLUMN, ADD và DROP CONSTRAINT không được DBMS hỗ trợ. Để sửa một bảng, bạn sẽ phải tạo một bảng mới với các định nghĩa đúng, chuyển dữ liệu (sử dụng lệnh INSERT INTO) từ bảng cũ, xóa bảng cũ, và đổi tên bảng mới như cũ. Giải pháp này áp dụng cho các cơ sở dữ liệu đơn giản; tuy nhiên, bạn sẽ phải làm nhiều hơn khi xử lý một cơ sở dữ liệu phức tạp.

- Xóa ràng buộc

Sử dụng mệnh đề DROP của lệnh ALTER TABLE để loại bỏ một ràng buộc.

ALTER TABLE {table_name}
DROP PRIMARY KEY | UNIQUE (column) | CONSTRAINT constraint_name
[CASCADE];
Cú pháp của nó chứa các mệnh đề phụ sau đây:
PRIMARY KEY sử dụng từ khóa PRIMARY KEY để xóa ràng buộc khóa chính của một bảng.
UNIQUE chỉ định UNIQUE để xóa ràng buộc duy nhất trên các cột được chỉ định.
CONSTRAINT nếu ràng buộc được xóa không phải là một khóa chính cũng không phải là duy nhất, thì sử dụng mệnh đề CONSTRAINT theo sau bởi tên ràng buộc để xóa một ràng buộc toàn vẹn.
CASCADE tùy chọn CASCADE gây ra ràng buộc phụ thuộc bất kỳ cũng được xóa. Bạn không thể thả một ràng buộc khóa chính hoặc ràng buộc duy nhất là một phần của một ràng buộc toàn vẹn tham chiếu mà không cần xóa khóa ngoại. Để xóa khóa tham chiếu và khóa ngoại, sử dụng mệnh đề CASCADE.
Ví dụ
```
CREATE TABLE dept2 AS SELECT * FROM Departments;
CREATE TABLE emp4 AS SELECT * FROM Employees;
ALTER TABLE dept2 ADD CONSTRAINT pk_dept2_id PRIMARY KEY (department_id);
ALTER TABLE emp4 ADD CONSTRAINT pk_emp4_id PRIMARY KEY (employee_id);
ALTER TABLE emp4 ADD CONSTRAINT fk_emp4_dept_id FOREIGN KEY (department_id) REFERENCES dept2(department_id);

ALTER TABLE dept2 DROP PRIMARY KEY CASCADE;
```
Phát biểu trên xóa ràng buộc khóa chính trên bảng dept2 và giảm ràng buộc khóa ngoại (fk_emp4_dept_id) được định nghĩa trên cột DEPARTMENT_ID trong bảng emp4.
```
ALTER TABLE emp2 DROP UNIQUE (ss_number);
```
Để xóa ràng buộc duy nhất sử dụng từ khóa UNIQUE theo sau bởi tên cột. Ràng buộc này được thêm vào bảng trước đó.
```
ALTER TABLE emp2 DROP CONSTRAINT fk_emp2_dept_id;
```
Điều này xóa ràng buộc khóa ngoại được tạo ra trước đó. Sử dụng mệnh đề CONSTRAINT để xóa một ràng buộc toàn vẹn khác một khóa chính hoặc một ràng buộc duy nhất.

- Bật/tắt ràng buộc
Trong tình huống bạn muốn tắt một ràng buộc hiện tại tạm thời mà không vĩnh viễn bỏ nó, bạn có thể sử dụng các mệnh đề ENABLE/DISABLE của lệnh ALTER TABLE.
ALTER TABLE {table_name}
DISABLE | ENABLE CONSTRAINT constraint_name [CASCADE];
Mệnh đề DISABLE tắt một ràng buộc toàn vẹn, và áp dụng tùy chọn CASCADE với nó sẽ vô hiệu hóa các ràng buộc phụ thuộc. Khi bạn đã muốn dùng lại ràng buộc đã tắt, chỉ đơn giản sử dụng mệnh đề ENABLE. Hai mệnh đề có thể được sử dụng trong cả hai lệnh CREATE và ALTER.
```
ALTER TABLE emp2 DISABLE CONSTRAINT ck_emp2_salary; -- tắt ràng buộc
ALTER TABLE emp2 ENABLE CONSTRAINT ck_emp2_salary; -- bật ràng buộc
```

- Đổi tên ràng buộc
Mệnh đề RENAME CONSTRAINT của lệnh ALTER TABLE cho phép bạn đổi tên ràng buộc bất kỳ hiện có.
ALTER TABLE {table_name}
RENAME CONSTRAINT old_constraint_name TO new_constraint_name;
Lưu ý rằng bạn không thể sử dụng tên của một ràng buộc hiện có sẵn trong cùng lược đồ. Đổi tên một ràng buộc không ảnh hưởng đến các đối tượng phụ thuộc ràng buộc đó. Phát biểu sau đổi tên tên ràng buộc cũ (ck_emp2_salary) trên bảng emp2 thành ck_emp2_min_salary.
```
ALTER TABLE emp2 RENAME CONSTRAINT ck_emp2_salary TO ck_emp2_min_salary;
```

- Xóa bảng
Để loại bỏ một bảng từ cơ sở dữ liệu bạn sử dụng lệnh DROP TABLE. Đó là một lệnh DDL, khi ban hành, vĩnh viễn xóa bảng chỉ định cùng với tất cả các dữ liệu trong bảng.
DROP TABLE {table_name} [CASCADE CONSTRAINTS];
Hãy nhớ những điểm sau đây trước khi sử dụng lệnh quan trọng này:
+ Không có xác nhận kiểu "Bạn có chắc chắn không?" từ cơ sở dữ liệu.
+ Lệnh này là không thể đảo ngược, có nghĩa là một khi lệnh này thực hiện thành công, bạn sẽ mất bảng và dữ liệu của nó ngay lập tức.
+ Các đối tượng phụ thuộc khác (view, stored procedure, function, ...) sẽ trở thành không hợp lệ.
+ Bạn không thể xóa một bảng có các ràng buộc toàn vẹn tham chiếu đang hoạt động. Đối với điều này, bạn phải sử dụng tùy chọn CASCADE CONSTRAINTS để xóa ràng buộc toàn vẹn phụ thuộc.
+ Chỉ chủ sở hữu bảng hoặc người được cấp với các đặc quyền có liên quan mới có thể thực hiện thao tác này.
```
DROP TABLE emp5 CASCADE CONSTRAINTS;
```

8. Thao tác dữ liệu
Trong thuật ngữ cơ sở dữ liệu, thao tác dữ liệu có ý nghĩa đối với ba hành động chính: chèn, cập nhật, và xóa. Chèn (insert) được thực hiện để thêm dữ liệu mới vào bảng, cập nhật (update) được thực hiện để thay đổi dữ liệu lưu trong bảng, và xóa (delete) giúp loại bỏ các dữ liệu không cần thiết. Một khi bạn tạo ra một bảng, bạn thực hiện các hành động này sử dụng ba lệnh thao tác dữ liệu (DML): INSERT, UPDATE, và DELETE.
Chú ý là các lệnh DDL không thể quay ngược (rollback) lại được; những lệnh này ngay lập tức ảnh hưởng đến cơ sở dữ liệu. Ngược lại, lệnh DML (INSERT, UPDATE và DELETE) có thể đảo ngược bằng cách sử dụng lệnh ROLLBACK. Để làm cho thay đổi bền vững và không thể đảo

ngược, sử dụng lệnh COMMIT. Ngược lại, để hoàn tác những thay đổi như chưa bao giờ xảy ra, sử dụng lệnh ROLLBACK. Hai lệnh này được gọi là các lệnh điều khiển giao tác (transaction) vì chúng quản lý những thay đổi được thực hiện bởi các phát biểu DML.

- Thêm dữ liệu
Hành động đầu tiên mà bạn thực hiện trên một bảng, sau khi nó được tạo ra, là thêm các dòng mới vào nó. Để làm điều này, bạn sử dụng lệnh INSERT. Lệnh này có thể được sử dụng theo ba cách:
+ Thêm một dòng hoàn chỉnh vào một bảng
+ Thêm một phần của dòng vào một bảng
+ Thêm dòng từ bảng khác thông qua truy vấn con
Cú pháp sau đây của lệnh INSERT được sử dụng để chỉ thêm một dòng vào một thời điểm cho một bảng. Sử dụng lệnh này khi bạn chèn toàn bộ hoặc một phần của dòng đơn.
```
INSERT INTO {table_name} [(column1, column2, …)]
VALUES (value1, value2, …);
```
Bên cạnh các từ khóa lệnh (INSERT INTO) và tên bảng bắt buộc, cú pháp ở trên yêu cầu thông tin về cột và giá trị. Lưu ý rằng danh sách cột là không bắt buộc. Đây là các cột trong bảng mà bạn muốn chèn vào. Mệnh đề VALUE là bắt buộc lưu các giá trị tương ứng cho cột, và phải được bao trong cặp ngoặc đơn.

- Chèn một dòng hoàn chỉnh
Bạn có thể bỏ qua danh sách cột khi bạn chèn một dòng hoàn chỉnh bằng cách thêm giá trị cho từng cột trong bảng. Trong ví dụ sau, bạn sẽ thêm một bản ghi mới đầy đủ vào bảng Employees. Bởi vì phát biểu mang giá trị cho từng cột, do đó danh sách cột có thể bỏ qua. Tuy nhiên, trong loại thao tác này, các giá trị phải được liệt kê theo thứ tự của các cột được định nghĩa trong bảng. Luôn luôn đưa giá trị ký tự và giá trị ngày vào cặp nháy đơn; giá trị số không cần trong cặp nháy. Khi bạn không biết giá trị cho một cột, chỉ cần sử dụng từ khóa null ở vị trí của nó, như sử dụng cho cột COMMISSION_PCT và cột MANAGER_ID. Nếu bạn bỏ quá sẽ nhận lỗi not enough values.
Việc thực hiện thành công câu lệnh này sẽ trả về thông điệp 1rows inserted trong khung Script Output. Click vào tab Data, sau đó click nút Refresh dưới tab này. Bạn sẽ thấy bản ghi mới. Mặc dù chèn, dòng này vẫn chưa lưu xuống bảng. Trong SQL Develop, bạn có thể thực hiện thay đổi bền vững bằng cách nhấp vào biểu tượng Commit. Cách khác, bạn có thể nhập và thực hiện lệnh COMMIT trong khung Worksheet. Tương tự, click vào biểu tượng Rollback để hoàn tác thay đổi, hoặc nhập và thực hiện lệnh ROLLBACK.
```
INSERT INTO Employees
VALUES (207,'Riaz','Ahmed','RT','123.456.7890','08-DEC-2014','IT_PROG',50000,null,null,50);
```

- Chèn một phần dòng
Bạn cũng có thể thêm một phần các dòng vào một bảng, nghĩa là bạn chỉ cung cấp giá trị cho một số cột chỉ định, và không phải cho tất cả. Điều này có thể được thực hiện bằng cách định nghĩa rõ ràng các cột bạn muốn nhập trị, như thể hiện trong phát biểu sau. Trong ví dụ này, bạn định nghĩa tên của năm cột (employee_id, last_name, email, hire_date, và job_id được đánh dấu là NOT NULLABLE trong cấu trúc bảng) từ bảng Employees mà bạn muốn nhập trị, và cung cấp các giá trị tương ứng cho theo thứ tự các cột được chỉ định - các cột được để trống ở đây sẽ được điền sau bằng lệnh UPDATE. Bạn chỉ có thể bỏ qua những cột đó cho phép giá trị null. Bỏ qua một giá trị cho một cột không cho phép giá trị null và không có giá trị mặc định, sẽ ném một thông báo lỗi, và quá trình chèn dòng sẽ thất bại.
```
INSERT INTO Employees (employee_id, last_name, email, hire_date, job_id)
VALUES (209, 'Sarim', 'SM', '10-DEC-2014', 'IT_PROG');
```
Lưu ý, theo cách tiếp cận này, bạn không bắt buộc phải thực hiện theo thứ tự cột của bảng. Thay vào đó, bạn phải cung cấp các giá trị theo thứ tự của danh sách được định nghĩa trong phát biểu INSERT.
Ví dụ, bạn có thể viết phát biểu này như sau:
```
INSERT INTO Employees (job_id,hire_date,email,last_name,employee_id)
VALUES ('IT_PROG', '10-DEC-2014', 'SM', 'Sarim', 209);
```

- Chèn dòng từ bảng khác
Trong hai ví dụ chèn trước, bạn đã thêm các dòng đơn vào bảng Employees. Có một dạng của lệnh này cho phép bạn thêm nhiều dòng cùng một lúc thông qua một truy vấn con. Trong mẫu này, bạn chèn các dòng vào bảng đích từ bảng nguồn khác và có thể bỏ qua danh sách cột.
```
INSERT INTO {table_name} [(column1, column2, …)]
{subquery};
```
Trong phát biểu sau, chèn giá trị từ tất cả các cột trong tất cả các dòng vào một bảng đích mới có tên là dept3 từ bảng Departments.
```
INSERT INTO dept3 SELECT * FROM departments;
```
Trong phát biểu sau, bạn chỉ định các cột mà bạn muốn đưa dữ liệu vào. Bạn cũng dùng mệnh đề WHERE trong truy vấn con để thiết lập điều kiện.
```
INSERT INTO dept3 (department_id, department_name)
SELECT department_id, department_name
FROM departmentsWHERE department_id <= 50;
```
Bạn sẽ thấy, phát biểu chèn các bản ghi của năm phòng (10 - 50) một lần nữa. Các bản ghi trùng lặp đã được chèn vào do không có ràng buộc khóa chính được định nghĩa trong bảng.
Hai điều bạn phải đảm bảo cho trường hợp cụ thể này là, không chỉ cả hai bảng nguồn và đích tồn tại trong cùng một lược đồ, mà cũng phải có cùng một kết cấu. Trước khi bạn thực hiện phát biểu, phải tạo bảng đích trước, bằng phát biểu sau:
```
CREATE TABLE dept3 (
  department_id number(4),
  department_name varchar2(30),
  manager_id number(6),
  location_id number(4));
```

Lưu ý, sau khi thực hiện phát biểu CREATE TABLE nêu trên, nếu bảng mới không xuất hiện trong danh sách Tables trong khung Connections, click vào kết nối Learn SQL và click vào biểu tượng Refresh ở đầu khung này.

- Cập nhật dữ liệu

Trong khi chèn một bản ghi mới trong phần trước bạn để lại một số cột trống. Lệnh DML UPDATE cho phép bạn điền vào các giá trị null trong các cột của bảng cũng như thay thế giá trị hiện tại bằng giá trị mới.

```
UPDATE {table_name}
SET {column = value} [, column = value, …]
[WHERE condition];
```

Bên cạnh từ khóa UPDATE, cú pháp lệnh này có ba thành phần:

table_name là tên của bảng bạn muốn cập nhật dữ liệu.

column=value tên của cột sẽ được cập nhật với giá trị mới. Toán tử (=) phải được đặt ở giữa. Bạn có thể thêm nhiều cặp cột/giá trị để cập nhật nhiều cột trong bảng như bạn muốn; bắt buộc phải có một cặp. Các cặp phải được phân tách bằng dấu phẩy.

WHERE condition mặc dù tùy chọn, mệnh đề WHERE có một vai trò rất quan trọng trong lệnh này. Bạn thiết lập một điều kiện xác định những dòng trong bảng sẽ được cập nhật. Nếu bạn bỏ qua mệnh đề này, tất cả các dòng trong bảng sẽ được cập nhật.

Trong phát biểu sau, bạn cập nhật FIRST_NAME và PHONE_NUMBER của nhân viên 209. Lưu ý rằng ngoài các giá trị null, bạn cũng có thể thay thế các giá trị hiện có thông qua lệnh này.

Lưu ý, luôn luôn tạo thói quen sử dụng mệnh đề WHERE trong câu lệnh UPDATE, nếu không mọi dòng trong bảng sẽ được cập nhật.

```
UPDATE employees
SET first_name='Muavia', phone_number='999.999.9999', salary=15000
WHERE employee_id=209;
```

- Xóa dữ liệu

Xóa là một hoạt động quan trọng khác được thực hiện trên cơ sở dữ liệu để loại bỏ các dòng từ một bảng và nó được thực hiện bằng cách thực hiện lệnh DML DELETE. Nó cũng phải được sử dụng với mệnh đề WHERE, trừ khi bạn muốn xóa tất cả các bản ghi từ một bảng.

Sử dụng COMMIT hoặc ROLLBACK để thực hiện các thay đổi bền vững.

```
DELETE FROM {table_name}
[WHERE condition];
```

Ví dụ sau sử dụng điều kiện xóa một bản ghi, vì chỉ có một dòng có tên 'Riaz'.

```
DELETE FROM Employees WHERE
first_name='Riaz';
```

Nếu bạn có nhiều bản ghi với cùng một tên (ví dụ như John), thì bạn nên thêm nhiều bộ lọc sử dụng toán tử AND để chỉ xóa bản ghi muốn xóa. Ví dụ, loại bỏ nhiều bản ghi nhưng đảm bảo rằng các nhân viên làm việc trong vai trò Sales Manager không bị xóa.

```
DELETE FROM emp2
WHERE commission_pct > 0 AND job_id <> 'SA_MAN';
```

Ví dụ sau sẽ không xóa bất kỳ bản ghi nào, vì không có nhân viên nào được tuyển dụng vào trước ngày 1 tháng 1 năm 2001.

```
DELETE FROM emp2
WHERE hire_date < TO_DATE('01.01.2001','D.MM.YYYY');
```

Ví dụ sau sẽ xóa tất cả các dòng từ bảng emp3. Câu lệnh cuối cùng cố gắng xóa phòng ban 10. Câu lệnh này sẽ không thành công do việc thực hiện ràng buộc toàn vẹn, và sẽ trả về ràng buộc toàn vẹn (HR.EMP_DEPT_FK) bị vi phạm. Lỗi xảy ra vì DEPARTMENT_ID cung cấp trong mệnh đề WHERE được tham chiếu bởi ràng buộc khóa ngoại được khai báo trong bảng Employees, có chứa bản ghi của một nhân viên (Jennifer Whalen) đăng ký thuộc phòng ban này.

```
DELETE FROM Departments WHERE department_id = 10;
```

Lưu ý, luôn luôn tạo thói quen sử dụng mệnh đề WHERE trong câu lệnh DELETE, nếu không thì mọi dòng trong bảng sẽ bị xóa.

Phụ thuộc hàm

1. Mở đầu

Trong mọi cơ sở dữ liệu quan hệ, kiểm soát dư thừa và bảo đảm nhất quán dữ liệu là hai vấn đề quan trọng nhất của người thiết kế cơ sở dữ liệu và người quản trị cơ sở dữ liệu phải đối mặt.

Dư thừa dữ liệu (data redundancy) sẽ xảy ra khi có một phần dữ liệu được lưu ở nhiều nơi trong cơ sở dữ liệu. Nếu nội dung của phần dữ liệu này có thay đổi thì phải đảm bảo rằng bản sao của dữ liệu đó cũng phải thay đổi theo y hệt. Khi đó phần dữ liệu đó được gọi là nhất quán trong cơ sở dữ liệu. Một cơ sở dữ liệu được gọi là ở *trạng thái nhất quán* (consistency state) nếu mọi dữ liệu đều nhất quán.

Vì các quan hệ là các thực thể luận lý để chứa dữ liệu trong mọi RDBMS, nên để kiểm soát sự dư thừa của dữ liệu cũng như để bảo toàn tính chính xác của nó, ta cần đặc biệt quan tâm đến các ràng buộc áp dụng cho các quan hệ của cơ sở dữ liệu.

Một trong các cách để tìm hiểu về các kiểu ràng buộc khác nhau được áp dụng trên dữ liệu của một hay nhiều quan hệ là thông qua việc dùng các *phụ thuộc hàm* (functional dependencies). Phụ thuộc hàm tồn tại dưới rất nhiều dạng, là những yêu cầu hay hạn chế thường thấy trong thế giới thực, và được thể hiện trong cơ sở dữ liệu. Nói chung các ràng buộc này có thể được chia thành hai nhóm:

- Ràng buộc ngữ nghĩa (semantic constraint) tùy thuộc vào ý nghĩa của thuộc tính trong quan hệ. Ví dụ như trong quan hệ EMPLOYEE (nhân viên) thì sẽ không có nhân viên nào có thể có mức lương hay tuổi âm.
- Ràng buộc thỏa thuận (agreement constraint, còn gọi là concordance constraint) sẽ không phụ thuộc vào giá trị của thuộc tính trong bộ, mà phụ thuộc vào nhiều thuộc tính trên bộ đó. Ví dụ, xét các thuộc tính Department (phòng ban) và Supervisor (trưởng phòng) của quan hệ EMPLOYEE, nếu ta giả sử rằng mỗi nhân viên chỉ làm việc cho một phòng ban, chỉ có một trưởng phòng cho một phòng và mọi phòng đều có trưởng phòng, thì nếu có hai dòng có chung giá trị ở cột Supervisor, khi đó giá trị ở cột Department của hai dòng đó phải giống nhau.

Phụ thuộc hàm là điều quan trọng nhất đối với cả ràng buộc ngữ nghĩa và ràng buộc thỏa thuận.

2. Định nghĩa của phụ thuộc hàm

Giả sử có quan hệ r(R) và hai tập thuộc tính con là A và B. Ta nói (các) thuộc tính A *xác định hàm* (functionally determines) (các) thuộc tính B trên r, ký hiệu là $A \rightarrow B$, nếu và chỉ nếu với mọi bộ t_1 và t_2 trên r, nếu $t_1(A) = t_2(A)$ thì $t_1(B) = t_2(B)$.

Trong ký hiệu $A \rightarrow B$, thuộc tính A, vế trái của phụ thuộc hàm, được gọi là *tập thuộc tính xác định* (determinant); thuộc tính B, vế phải của phụ thuộc hàm được gọi là *tập thuộc tính phụ thuộc* (dependent). Nếu $A \rightarrow B$ (ngầm hiểu là trên quan hệ r), thì ta nói quan hệ r "thỏa phụ thuộc hàm" hay "phụ thuộc hàm được thỏa bởi quan hệ". Rõ ràng trong định nghĩa trên, cả A và B đều là các tập thuộc tính chứ không nhất thiết là các thuộc tính đơn. Khi dùng tập thuộc tính trong phụ thuộc hàm, người ta vẫn dùng vài chữ cái để biểu diễn cho tập thuộc tính đó. Nghĩa là trong phụ thuộc hàm $AB \rightarrow X$, thì AB là viết tắt của $A \cup B$. Các phụ thuộc hàm có vế phải chỉ có một thuộc tính được gọi là các phụ thuộc hàm *dạng đơn* (simple).

Ví dụ

Giả sử có quan hệ Lakes-Of-The-World sau, hãy cho biết các phụ thuộc hàm Continent → Name và Name → Length có thỏa trên quan hệ này hay không. Giả sử thuộc tính Area (diện tích) được đo bằng dặm vuông còn thuộc tính Length (chiều dài) được đo bằng dặm.

Lakes-Of-The-World

Name	Continent	Area	Length
Caspian Sea	Asia-Europe	143244	760
Superior	North America	31700	350
Victoria	Africa	26828	250
Aral Sea	Asia	24904	280
Huron	North America	23000	206
Michigan	North America	22300	307
Tanganyika	Africa	12700	420

a. Phụ thuộc hàm Continent → Name *không thỏa* trên quan hệ này. Ta có thể dễ dàng kiểm tra điều này dựa theo định nghĩa của phụ thuộc hàm. Nếu ta có Continent → Name thì điều đó có nghĩa là với mọi cặp bộ, mà nếu cặp bộ đó có chung giá trị ở cột Continent (lục địa) thì sẽ có chung giá trị ở cột Name (tên hồ). Ví dụ như trong quan hệ Lakes-Of-The-World xét các bộ như sau:
   t_1: (Superior, North America, 31700, 350) và t_2: (Huron, North America, 23000, 206)
   Chú ý rằng $t_1(\text{Continent}) = t_2(\text{Continent}) = \text{North America}$ nhưng $t_1(\text{Name}) = \text{Superior} \neq t_2(\text{Name}) = \text{Huron}$.
   Vì ở hai bộ này các giá trị ở thuộc tính Name là không giống nhau nên ta nói phụ thuộc Continent → Name là không thỏa.
b. Phụ thuộc hàm Name → Length thỏa trong quan hệ. Trong trường hợp này quan hệ rõ ràng được thỏa vì với mọi hồ chỉ có một độ dài tương ứng của nó.

Bạn nên nhớ rằng việc nhận định một thuộc tính có xác định một thuộc tính khác hay không là chỉ dựa vào ý nghĩa của các thuộc tính đó. Theo ý nghĩa đó thì phụ thuộc hàm sẽ gần gũi với thế giới thực hơn và sẽ được thỏa tại mọi thời điểm. Nghĩa là phụ thuộc hàm sẽ đúng với mọi thể hiện của quan hệ. Vì phụ thuộc hàm tùy thuộc vào ngữ nghĩa của các thuộc tính nên phụ thuộc hàm không nên chỉ dựa vào một thể hiện cụ thể của quan hệ. Tức là ta không thể chỉ nhìn vào nội dung hiện thời của quan hệ và dựa trên các giá trị của các thuộc tính A và B để nói rằng $A \rightarrow B$. Cũng nên chú ý thêm là từ định nghĩa phụ thuộc hàm và các thảo luận trên, thì luôn có thể xác định rằng một phụ thuộc hàm nào đó có thỏa hay không dựa trên một thể hiện của quan hệ. Tuy nhiên sẽ không hợp lệ khi chỉ ra một phụ thuộc hàm nào đó dựa trên thể hiện của quan hệ mà không quan tâm đến ý nghĩa của các thuộc tính tham gia.

Thuật toán Thỏa (Satisfies Algorithm) dưới đây thường được dùng để xác định một quan hệ r có thỏa hay không một phụ thuộc hàm $A \rightarrow B$ nào đó. Đầu vào của thuật toán là quan hệ r và phụ thuộc hàm $A \rightarrow B$. Đầu ra của thuật toán là True nếu thỏa $A \rightarrow B$, là False nếu không thỏa $A \rightarrow B$.

Thuật toán Thỏa (Satisfies Algorithm)

(1) Sắp xếp các bộ trong quan hệ r dựa trên (các) thuộc tính A sao cho các bộ có cùng giá trị trên thuộc tính A sẽ nằm cạnh nhau.
(2) Kiểm tra xem các bộ có cùng giá trị trên (các) thuộc tính A có cùng giá trị trên (các) thuộc tính B hay không.
(3) Nếu có hai bộ của r thỏa điều kiện (1) nhưng không thỏa điều kiện (2) thì thuật toán sẽ trả về False, còn không có nghĩa là quan hệ thỏa phụ thuộc hàm nên thuật toán trả về True.

Ví dụ
Với quan hệ Lakes-Of-The-World trên, áp dụng thuật toán Thỏa để kết luận rằng thuộc tính Continent không xác định thuộc tính Name.

Lakes-Of-The-World

Name	Continent	Area	Length
Victoria	Africa	26828	250
Tanganyika	Africa	12700	420
Aral Sea	Asia	24904	280
Caspian Sea	Asia-Europe	143244	760
Superior	North America	31700	350
Huron	North America	23000	206
Michigan	North America	22300	307

Sau khi sắp xếp các bộ của quan hệ dựa trên thuộc tính Continent, rõ ràng thuật toán Thỏa cho biết ngay là phụ thuộc hàm Continent → Name là không được thỏa trên quan hệ: có ít nhất 2 trị bằng nhau trên thuộc tính Continent (Africa), nhưng 2 trị tương ứng trên thuộc tính Name lại khác nhau. Do đó thuật toán sẽ trả về False.

3. Phụ thuộc hàm và khóa

Giả sử có quan hệ r(R) và khóa chính của nó là K, khi đó với mọi giá trị của K ta luôn xác định được có hay không một bộ trong quan hệ có giá trị K. Ngoài ra, nếu có bộ nào đó trong quan hệ có trị K thì ta không những có thể nói rằng bộ đó là duy nhất, mà còn có thể xác định được trị của các thuộc tính còn lại. Nếu sử dụng ký hiệu của phụ thuộc hàm thì ta có thể nói rằng khóa K xác định mọi thuộc tính còn lại của quan hệ. Nghĩa là $K \rightarrow A_i$ với A_i là tập thuộc tính bất kỳ của quan hệ. Vì khóa là một trường hợp đặc biệt của phụ thuộc hàm, nên tất cả các thuộc tính đúng với phụ thuộc hàm đều là các khóa hợp lệ. Một số các thuộc tính đó sẽ được đề cập đến trong các phần tiếp theo.

4. Các tiên đề suy luận cho phụ thuộc hàm

Một quan hệ r có thể thỏa nhiều phụ thuộc hàm. Về mặt lý thuyết, ta có thể dùng thuật toán Thỏa với mọi kết hợp giữa các thuộc tính để biết quan hệ r có thỏa phụ thuộc hàm nào trong số đó hay không. Mặc dù phương pháp này sẽ vét cạn được mọi phụ thuộc hàm có trong quan hệ r, nhưng sẽ đơn điệu và mất thời gian.
Các tiên đề suy luận (inference axioms) cung cấp một cách khác giúp chúng ta suy dẫn được một phụ thuộc hàm nào đó có thỏa trên một quan hệ hay không, không phải dùng thuật toán trên. Giả sử cho tập F gồm các phụ thuộc hàm, các tiên đề suy luận là tập hợp các luật cho biết nếu một quan hệ thỏa các phụ thuộc hàm của F thì quan hệ phải thỏa các phụ thuộc hàm khác. Tập phụ thuộc hàm khác đó được gọi là *suy dẫn* (derived, logically deduced) từ phụ thuộc hàm ban đầu của F.
Các tiên đề suy luận được gọi là *hoàn chỉnh* (complete). Gọi như thế vì nếu có quan hệ r(R) và tập phụ thuộc hàm F được thỏa trên r, thì các tiên đề sẽ cho phép chúng ta suy dẫn ra các phụ thuộc hàm hợp lệ khác cũng thỏa mãn trên r, và nếu các tiên đề được áp dụng đúng thì chúng sẽ không suy dẫn các phụ thuộc sai.
Giả sử có quan hệ r(R) và X, Y, Z cùng W là các tập hợp con của R. Các tiên đề suy luận, còn được gọi là các tiên đề Armstrong, được trình bày như dưới đây. Trong cặp () là tên thường dùng của các tiên đề đó.

Các tiên đề Armstrong

(1) Nếu $Y \subseteq X$ thì $X \rightarrow Y$ (phản xạ*, reflexivity, inclusion)
(2) Nếu $X \rightarrow Y$ thì $XW \rightarrow Y$ và/hoặc $XW \rightarrow YW$ (gia tăng*, augmentation)
(3) Nếu $X \rightarrow Y$ và $Y \rightarrow Z$ thì $X \rightarrow Z$ (bắc cầu, transitivity)
(4) Nếu $X \rightarrow Y$ và $YW \rightarrow Z$ thì $XW \rightarrow Z$ (giả bắc cầu*, pseudotransitivity)
(5) Nếu $X \rightarrow Z$ và $X \rightarrow Y$ thì $X \rightarrow YZ$ (cộng thêm, additivity, union)
(6) Nếu $X \rightarrow YZ$ thì $X \rightarrow Y$ và $X \rightarrow Z$ (chiếu, projectivity, decomposition)
* các tiên đề này gọi là tiên đề cơ bản. Mọi luật suy luận khác đều được suy dẫn từ tập các tiên đề cơ bản.
Chú ý là khi có nhiều thuộc tính ở vế trái hay vế phải của phụ thuộc hàm thì thứ tự của các thuộc tính đó là không quan trọng, nghĩa là AB → CD có thể được ghi là AB → DC hay BA → CD, ghi kiểu nào thì ý nghĩa của phụ thuộc hàm đó đều không đổi.

- Tiên đề *Reflexivity* chỉ ra rằng nếu cho một tập thuộc tính thì chính tập thuộc tính đó sẽ xác định tập con trong tập đó. Điều đó cũng có nghĩa là $X \rightarrow X$ với mọi tập thuộc tính X, lý do $X \subset X$ là đương nhiên. Các phụ thuộc hàm có dạng $X \rightarrow Y$ trong đó $Y \subset X$ được gọi là *phụ thuộc hàm tầm thường* (trivial dependencies) hoặc phụ thuộc hiển nhiên.
Chứng minh Để chứng minh rằng $X \rightarrow Y$, chỉ cần chứng minh rằng không có cặp dòng u và v nào có bộ thuộc tính X giống nhau mà bộ thuộc tính Y lại khác nhau. Điều này là hiển nhiên với $Y \subseteq X$, không bao giờ có hai dòng có bộ thuộc tính X giống nhau mà tập con của bộ thuộc tính X này lại khác nhau. ■

- Tiên đề *Augmentation* chỉ ra rằng có thể tăng cường hay mở rộng vế trái của phụ thuộc hàm hay hai vế của phụ thuộc hàm bằng một hay nhiều thuộc tính. Chú ý là tiên đề không cho phép mở rộng cho vế phải một mình.

Chứng minh Giả sử rằng X → Y, xem xét hai dòng u và v trong T có các thuộc tính trên XZ (tức X ∪ Z) giống nhau. Chúng ta cần chỉ ra rằng u và v không thể khác nhau trên tập thuộc tính YZ. Một khi u và v giống nhau trên tập thuộc tính XZ, chúng chắc chắn giống nhau trên tập thuộc tính X; và vì chúng ta giả định rằng X → Y, thì u và v phải giống nhau trên tập thuộc tính Y. Tương tự, một khi u và v giống nhau trên tập thuộc tính XZ, chúng chắc chắn giống nhau trên tập thuộc tính Z. Do đó, u và v phải giống nhau trên tập thuộc tính Y và trên tập thuộc tính Z. ■

- Tiên đề *Transitivity* chỉ ra rằng nếu một thuộc tính xác định được một thuộc tính khác, và đến lượt thuộc tính này cũng xác định được một thuộc tính khác nữa, thì thuộc tính ban đầu có thể xác định được thuộc tính thứ ba.

- Tiên đề *Pseudotransitivity* là tổng quát hóa tiên đề Transitivity. Chú ý là để có thể áp dụng được tiên đề này, yêu cầu toàn bộ vế phải của phụ thuộc hàm phải xuất hiện như là các thuộc tính xác định của một phụ thuộc hàm khác.

- Tiên đề *Additivity* chỉ ra rằng nếu có hai phụ thuộc hàm có chung tập thuộc tính xác định thì có thể tạo nên được một phụ thuộc hàm mới với tập thuộc tính xác định cũ còn vế phải mới là hội của hai vế phải kia.

- Tiên đề *Projectivity* là đảo của tiên đề Additivity. Tiên đề này chỉ ra rằng tập thuộc tính xác định của một phụ thuộc hàm bất kỳ có thể xác định duy nhất một thuộc tính bất kỳ hoặc một kết hợp bất kỳ các thuộc tính thuộc vế phải phụ thuộc hàm đó.
Chứng minh Chú ý rằng YZ = Y ∪ Z. Như vậy, YZ → Y theo tiên đề Reflexivity. Từ tiên đề Transitivity, X → YZ và YZ → Y dẫn đến X → Y. Tương tự, chúng ta có thể chứng minh X → Z. ■

Bạn nên nhớ rằng các tiên đề suy luận trên cho phép từ một tập nhỏ các luật, chúng ta có thể tìm thêm các phụ thuộc hàm mới để biểu diễn hay thỏa một quan hệ cho dù ban đầu ta không có. Ví dụ, nếu quan hệ r thỏa X → Y và Y → Z thì nó cũng thỏa X → Z, dù phụ thuộc hàm này không được chỉ định rõ và chứng minh thỏa quan hệ r.
Như đã nói, các tiên đề suy luận cho phép chúng ta có được các phụ thuộc hàm mới từ một tập các phụ thuộc hàm ban đầu. Ví dụ tiếp theo sẽ minh họa cho cách sử dụng các tiên đề suy luận cũng như cách áp dụng chúng để suy dẫn ra các phụ thuộc hàm mới.

Ví dụ
Cho tập F = {A → B, C → X, BX → Z}, suy dẫn ra phụ thuộc hàm AC → Z bằng cách áp dụng các tiên đề suy luận. Giả sử rằng tất cả các phụ thuộc tính trên đều thuộc về một quan hệ có lược đồ R.
(1) Từ A → B (cho sẵn) và BX → Z (cho sẵn), áp dụng tiên đề Pseudotransitivity ta có AX → Z. Chú ý là ta có thể áp dụng tiên đề này vì thuộc tính B xuất hiện ở vế phải của một phụ thuộc hàm và ở vế trái của một phụ thuộc hàm khác.
(2) Ta có AX → Z (bước trước) và C → X (cho sẵn), áp dụng tiên đề Pseudotransitivity một lần nữa ta có AC → Z.

Ví dụ trên minh họa cho cách tìm một phụ thuộc hàm thành viên. Thực ra kết quả này còn nói lên rằng nếu quan hệ r thỏa các phụ thuộc hàm của tập F cho trên thì nó phải thỏa phụ thuộc hàm AC → Z.
Ta có thể phát biểu như sau: cho tập F các phụ thuộc hàm của quan hệ có lược đồ R và một phụ thuộc hàm X → Y. Khi đó ta nói F có thành viên X → Y, ký hiệu F |= X → Y, nếu mọi quan hệ r(R) thỏa các phụ thuộc hàm trong F thì cũng thỏa X → Y.

Ví dụ
Giả sử có F = {A → B, C → D} với C ⊂ B. Hãy chứng minh rằng F |= A → D.
(1) Với C ⊂ B, áp dụng tiên đề Refexivity ta có B → C.
(2) Ta có A → B (cho sẵn) và B → C (bước trước), áp dụng tiên đề Transitivity ta có A → C.
(3) Ta có A → C (bước trước) và C → D (cho sẵn), áp dụng tiên đề Transitivity ta có A → D.
Vì A → D có thể được suy dẫn từ tập F nên F |= A → D.

5. Phụ thuộc hàm dư thừa
Giả sử có tập phụ thuộc hàm F, một phụ thuộc hàm A → B của F được gọi là *phụ thuộc hàm dư thừa* của F (redundant with respect to the FDs of F) nếu và chỉ nếu A → B có thể được suy dẫn từ tập phụ thuộc hàm F – {A → B}. Nghĩa là nếu A → B có thể được suy dẫn từ tập phụ thuộc hàm F sau khi bỏ bớt A → B thì nó là dư thừa.
Các phụ thuộc hàm dư thừa là không cần thiết và có thể loại bỏ khỏi tập F. Việc loại bớt các phụ thuộc hàm dư thừa giúp ta cực tiểu hóa tập phụ thuộc hàm.
Việc xác định các phụ thuộc hàm dư thừa trong một tập phụ thuộc hàm là một quá trình xử lý dài dòng và phức tạp, đặc biệt khi số lượng các phụ thuộc hàm trong tập phụ thuộc hàm là lớn. Thuật toán Thành viên (Membership Algorithm) trình bày dưới đây sẽ giúp ta có được một thủ tục để xác định các phụ thuộc hàm dư thừa, tuy nhiên thuật toán khá dài nếu áp dụng cho tập phụ thuộc hàm lớn. Đầu vào của thuật toán là tập phụ thuộc hàm F và một phụ thuộc hàm cụ thể để kiểm tra xem nó có dư thừa trong F hay không.

Thuật toán Thành viên (Membership Algorithm)
Giả sử có F là tập các phụ thuộc hàm có A → B ∈ F. Để xác định xem A → B có dư thừa trong F hay không, ta làm như sau:
(1) Tạm xóa A → B khỏi F và gọi G là tập phụ thuộc hàm mới, nghĩa là G = F – {A → B}.
Nếu G ≠ ∅ thì chuyển sang bước 2, ngược lại kết thúc thuật toán vì rõ ràng A → B không dư thừa.
(2) Khởi động tập thuộc tính T_i (với i = 1) từ tập (các) thuộc tính A (tập thuộc tính xác định của phụ thuộc hàm đang xét).
Nghĩa là $T_i = T_1 = \{A\}$. Tập T_i hiện hành là T_1.
(3) Trong tập G, tìm phụ thuộc hàm X → Y sao cho tất cả tất cả các thuộc tính của tập thuộc tính xác định X đều là các phần tử của tập T_i hiện hành. Có hai kết quả có thể nhận được:
(3-a) Nếu tìm thấy phụ thuộc hàm này thì đưa các thuộc tính của Y (vế phải của phụ thuộc hàm tìm thấy) vào tập T_i tạo thành tập mới $T_{i+1} = T_i \cup \{Y\}$. Bây giờ tập T_{i+1} trở thành T_i hiện hành.

Kiểm tra xem có phải tất cả các thuộc tính của B (vế phải của phụ thuộc hàm đang xét) có phải là thành viên của T_{i+1} hay không. Nếu là thành viên thì kết thúc thuật toán vì phụ thuộc hàm $A \to B$ là dư thừa.

Còn nếu không phải mọi thuộc tính của B đều là thành viên của T_{i+1} thì bỏ $X \to Y$ khỏi G rồi lặp lại bước 3.

(3-b) Nếu G = ∅ hay không có phụ thuộc hàm nào trong G có tất cả các thuộc tính của tập thuộc tính xác định trong T_i hiện hành thì $A \to B$ là không dư thừa.

Như đã nói, nếu một phụ thuộc hàm $A \to B \in F$ là dư thừa thì ta có thể loại bỏ nó khỏi tập F. Ví dụ sau sẽ minh họa cho cách dùng thuật toán Thành viên.

Ví dụ

Cho tập $F = \{X \to YW, XW \to Z, Z \to Y, XY \to Z\}$, hãy cho biết $XY \to Z$ có dư thừa trong F hay không?

(1) Tạm thời xóa phụ thuộc hàm $XY \to Z$ khỏi tập F. Gán $G = \{X \to YW, XW \to Z, Z \to Y\}$

(2) Khởi tạo tập thuộc tính T_1 chứa (các) thuộc tính của tập thuộc tính xác định của phụ thuộc hàm đang xét. Trong trường hợp này vì XY là tập thuộc tính xác định của phụ thuộc hàm $XY \to Z$ nên ta có $T_1 = \{XY\}$.

(3-a) Trong tập G, tìm các phụ thuộc hàm có tất cả các tập thuộc tính xác định của chúng là các phần tử của tập T_1. Vì tập thuộc tính xác định của $X \to YW$ là một phần tử của T_1 ($X \in T_1$) nên ta sẽ đưa vào T_1 các thuộc tính xuất hiện ở vế phải của phụ thuộc hàm này để có tập mới như sau.

$$T_2 = T_1 \cup \{YW\} = \{XY\} \cup \{YW\} = \{XYW\}$$

Vì Z, vế phải của $XY \to Z$ không phải là phần tử của T_2, ($Z \notin T_2$), loại $X \to YW$ khỏi G. Nghĩa là $G = \{XW \to Z, Z \to Y\}$ và lặp lại bước 3.

(3-a) (lần 2) Rõ ràng là bây giờ tất cả các tập thuộc tính xác định của $XW \to Z$ đều là các phần tử của T_2. Do đó ta sẽ đưa các thuộc tính ở vế phải của phụ thuộc hàm này vào tập T_2 để có tập mới $T_3 = T2 \cup \{Z\} = \{XYWZ\}$.

Vì $Z \in T_3$ nên thuật toán dừng tại đây, kết luận $XY \to Z$ là phụ thuộc hàm dư thừa. Nghĩa là có thể bỏ $XY \to Z$ an toàn khỏi F.

Để xác minh xem $XY \to Z$ có thực sự dư thừa không, ta cần loại bỏ nó khỏi F trước khi thử suy dẫn nó từ F bằng cách áp dụng các tiên đề suy luận. Bây giờ F là: $F = \{X \to YW, XW \to Z, Z \to Y\}$

a. Từ $X \to YW$ (cho sẵn), áp dụng tiên đề Projectivity ta có $X \to Y$ và $X \to W$.

b. Từ $X \to W$ (bước trước) và $XW \to Z$ (cho sẵn), áp dụng luật Pseudotransitivity ta có $XX \to Z$.

c. Vì hội các tập thuộc tính tương đương đều bằng chính nó, nghĩa là $X \cup X = X$, nên $X \to Z$.

Do đó phụ thuộc hàm $XY \to Z$ là dư thừa vì rõ ràng nó có thể được suy dẫn từ các phụ thuộc hàm khác của F.

6. Bao đóng, phủ và phụ thuộc hàm tương đương

Cho tập phụ thuộc hàm F, ta có thể tìm được tất cả các phụ thuộc hàm suy dẫn từ F. Tập các phụ thuộc hàm suy dẫn này rất quan trọng đối với quá trình chuẩn hóa quan hệ. Các phần tiếp theo đây sẽ đưa ra các định nghĩa và thuật toán để có được tập các phụ thuộc hàm đó cũng như cách kiểm tra xem một phụ thuộc hàm có thể được suy dẫn từ một tập F cho trước hay không.

6.1. Bao đóng của tập phụ thuộc hàm F

Cho tập phụ thuộc hàm F của lược đồ quan hệ R, ta định nghĩa F^+ là *bao đóng* (closure) của F, là tập tất cả các phụ thuộc hàm được suy dẫn luận lý từ F. Thể hiện toán học là $F^+ = \{X \to Y \mid F \models X \to Y\}$.

Tập bao đóng đồng thời thỏa hai điều sau:

(1) F^+ là tập nhỏ nhất chứa F và thỏa (2).

(2) Mọi áp dụng của các tiên đề suy luận lên mọi phụ thuộc hàm của F chỉ có thể suy dẫn ra các phụ thuộc hàm có trong F^+.

Ví dụ sau minh họa khái niệm này.

Ví dụ

Cho tập $F = \{XY \to Z\}$. Hãy xác định tất cả các phần tử của F^+. Giả sử lược đồ chỉ chứa các thuộc tính được chỉ ra trong các phụ thuộc hàm của F.

Để suy dẫn tất cả các phụ thuộc hàm có thể suy dẫn từ F ta đã tiến hành quá trình sau. Thứ nhất, áp dụng tất cả các tiên đề suy luận cho tất cả các thuộc tính đơn. Thứ hai, áp dụng tất cả các tiên đề suy luận cho tất cả các kết hợp của hai thuộc tính rồi dùng các phụ thuộc hàm của F khi nào có thể. Tiếp theo áp dụng các tiên đề suy luận cho tất cả các kết hợp của ba thuộc tính rồi dùng các phụ thuộc hàm của F khi cần thiết. Cứ tiếp tục quá trình này với các thuộc tính có trong F. Khi đó tập kết quả sẽ như sau.

$F^+ = \{X \to X, Y \to Y, Z \to Z,$
$XY \to X, XY \to Y, XY \to XY, XZ \to X, XZ \to Z, XZ \to XZ, YZ \to Y, YZ \to Z, YZ \to YZ,$
$XYZ \to XY, XYZ \to XZ, XYZ \to YZ, XYZ \to XYZ\}$

6.2. Bao đóng của tập thuộc tính

Số lượng các phần tử trong F^+ có thể rất lớn so với số lượng các thuộc tính trong F. Ví dụ trên cho thấy F chỉ có một phụ thuộc hàm nhưng F^+ có đến 16 phụ thuộc hàm khác nhau.

Với mọi phụ thuộc hàm $X \to Y$, F^+ đều có thể được dùng để xác định $F \models X \to Y$ hay không; tuy nhiên việc tính F^+ là một quá trình dài dòng. Để đơn giản hóa quá trình này ta sẽ dùng một phương pháp khác, trong đó có bước tính X^+, là *bao đóng của tập thuộc tính* (closure of the set of attribute) X trong F. Khái niệm này có thể được định nghĩa như sau:

Giả sử có tập thuộc tính X và tập phụ thuộc hàm F, khi đó bao đóng của tập thuộc tính X trong F, được viết là X^+, là tập các thuộc tính A được suy dẫn từ X bằng cách áp dụng các tiên đề suy luận trên tập phụ thuộc hàm F.

$X^+ = \{A: A \in R \text{ và } X \to A \in F^+\}$

Ví dụ

Cho R = {A, B, C, D, E, G}, F = {A → C, A → EG, B → D, G → E}, X = {A, B}, Y = {C, G, D}.
X^+ = {A, B, C, D, E, G} và Y^+ = {C, G, D, E}
Bao đóng của X luôn là tập khác rỗng vì ít nhất cũng phải có X → X suy từ tiên đề Reflexivity. Nếu đã ngầm hiểu tập phụ thuộc hàm F thì ta chỉ cần nói ngắn gọn rằng X^+ là bao đóng của X. Chú ý là ta giả sử X và tất cả các thuộc tính của các phụ thuộc hàm của F đều được định nghĩa trên cùng một lược đồ R.
Bạn có thể thắc mắc rằng, liệu có mối liên hệ nào giữa các phụ thuộc hàm đã được suy dẫn từ X và các phụ thuộc hàm của F^+ có X đóng vai trò tập thuộc tính xác định hay không. Thực ra nếu $X^+ = \{A_1A_2A_3...A_n\}$ thì theo định nghĩa bao đóng của X, mọi thuộc tính của tập hợp này đều có thể được suy dẫn từ X. Nghĩa là $X \to A_1$, $X \to A_2$, $X \to A_3$, ... , $X \to A_n$. Dùng các phụ thuộc hàm này và lần lượt áp dụng tiên đề Additivity, ta sẽ có phụ thuộc hàm $X \to A_1A_2A_3...A_n$ có vế phải chứa tất cả các thuộc tính $A_i \in X^+$. Vế phải của phụ thuộc hàm này là tập tối đa của các thuộc tính trong F^+ vốn chứa các thuộc tính được suy dẫn từ X. Nghĩa là nếu $X \to Z \in F^+$ thì $Z \to A_1A_2A_3...A_n$. Rõ ràng trong kết luận này, tập hợp ở vế phải chính là X^+. Kết quả này cung cấp cho ta một điều kiện để xác định $F \models X \to Y$ hay không với mọi X → Y. Thực tế, để kiểm tra $F \models X \to Y$, ta chỉ cần chứng minh rằng $Y \in X^+$.
Để tính X^+ ta có thể dùng thuật toán tương tự như thuật toán Thành viên, gọi là thuật toán Bao đóng (Closure Algorithm).

Thuật toán Bao đóng (Closure Algorithm)
Đầu vào của thuật toán này là tập phụ thuộc hàm F và tập thuộc tính X định nghĩa trên cùng một lược đồ. Đầu ra của thuật toán là X^+.
(1) Khởi động tập phụ thuộc hàm G với các thuộc tính của F, nghĩa là G = F.
(2) Khởi động tập thuộc tính T_i (với i = 1) chứa các thuộc tính của X. Nghĩa là $T_i = T_1 = \{X\}$. T_i hiện hành là T_1.
(3) Trong tập G, tìm các phụ thuộc hàm X → Y sao cho tất cả các thuộc tính của tập thuộc tính xác định X đều là các thuộc tính của tập hợp hiện hành T_i. Có hai kết quả có thể nhận được:
  (3-a) Nếu tìm thấy phụ thuộc hàm kiểu này, đưa các thuộc tính của Y vào tập T_i tạo thành tập mới $T_{i+1} = Ti \cup \{Y\}$ và xóa X → Y khỏi G vì vế phải của nó không thể tạo nên các thuộc tính mới cho các T_i sau này nữa. Bây giờ tập T_{i+1} trở thành T_i hiện hành. Lặp lại bước 3.
  (3-b) Nếu G = ∅ hay không có phụ thuộc hàm nào trong G sao cho tất cả các thuộc tính của tập thuộc tính xác định của nó thuộc T_i hiện hành thì chấm dứt thuật toán.
  Khi thuật toán kết thúc, tập hiện hành T_i sẽ chứa tất cả các thuộc tính có thể được suy dẫn từ X. Nghĩa là $T_i = X^+$.
Ví dụ sau minh họa thuật toán Bao đóng.

Ví dụ
Cho tập phụ thuộc hàm F = {A → B, B → C, BC → D, DA → B}. Hãy tìm X^+ với X = {A}. Ý nghĩa của tập hợp này là gì? Phụ thuộc hàm A → DA khi đó có đúng hay không?
(1) Gán G = {A → B, B → C, BC → D, DA → B}.
(2) Gán $T_i = T_1 = \{A\}$.
  (3-a) Vì tập thuộc tính xác định của A → B là một phần tử của T_1 nên ta có $T_2 = T_1 \cup \{B\} = \{AB\}$. Ta có thể xóa A → B khỏi G vì nó không còn có thể tạo thêm được thuộc tính nào từ vế phải cho các T_i sau. Bây giờ ta làm việc với tập G = {B → C, BC → D, DA → B}. Lặp lại bước 3.
  (3-a) (lần 2) Với tập G mới ta thấy tập thuộc tính xác định của B → C là phần tử của T_2. Thêm các thuộc tính vế phải của phụ thuộc hàm này vào T_2 để tạo $T_3 = T_2 \cup \{C\} = \{ABC\}$. Xóa B → C khỏi G. Bây giờ ta có G = {BC → D, DA → B}. Lặp lại bước 3.
  (3-a) (lần 3) Với tập G mới ta thấy tập thuộc tính xác định của BC → D là phần tử của T_3. Thêm các thuộc tính vế phải của phụ thuộc hàm này vào T_3 để tạo $T_4 = T_3 \cup \{D\} = \{ABCD\}$. Xóa BC → D khỏi G. Bây giờ ta có G = {DA → B}. Lặp lại bước 3.
  (3-a) (lần 4) Với tập G mới ta thấy tập thuộc tính xác định của DA → B là phần tử của T_4. Thêm các thuộc tính vế phải của phụ thuộc hàm này vào T_4 để tạo $T_5 = T_4 \cup \{B\} = \{ABCD\}$. Xóa DA → B khỏi G. Bây giờ ta có G = ∅. Lặp lại bước 3.
  (3-b) Vì G = ∅ nên thuật toán chấm dứt và T_i hiện hành là $T_5 = \{ABCD\} = X^+$.

Kết quả này cho thấy thuộc tính A có thể đóng vai trò tập thuộc tính xác định của phụ thuộc hàm có vế phải là một trong những thuộc tính của X^+ hay mọi sự kết hợp của nó.
Các thuộc tính AD là các phần tử của X^+, do đó phụ thuộc hàm A → DA là đúng, Nói cách khác: $F \models A \to DA$
Ta có thể xác minh điều này như sau. Tập phụ thuộc hàm F = {A → B, B → C, BC → D, DA → B}.
a. Ta có A → B (cho sẵn) và B → C (cho sẵn) nên sẽ có A → C (tiên đề Transitivity)
b. Ta có A → B (cho sẵn) và BC → D (cho sẵn) nên có AC → D (tiên đề Pseudotransitivity)
c. Với AC → D (bước trước) và A → C (bước đầu) ta có AA → D (tiên đề Pseudotransitivity)
d. Vì A ∪ A = A nên theo quy ước viết tắt của phép hội ta có A → D
e. Vì A → A (tiên đề Reflexivity) và A → D (bước trước) ta có A → AD (tiên đề Additivity)
Do đó A → DA là đúng.

6.3. Thuật toán tìm khóa
Khóa là tập thuộc tính K mà bao đóng của K đúng bằng R, nghĩa là $K^+ = R$.
Từ định nghĩa này ta thấy có thể tìm khóa bắt đầu từ R vì $R^+ = R$, ta loại dần các phần tử của R để nhận được tập bé nhất mà bao đóng của nó đúng bằng R. Thuật toán tìm khóa sau đây có đầu vào là lược đồ quan hệ R và tập phụ thuộc hàm F trong R, đầu ra là khóa K.

Thuật toán tìm khóa

(1) Gán K = R
(2) Với mỗi A trong K, nếu (K – A) = R thì loại A khỏi K: K = K - A
Khi xét hết các thuộc tính A của K, K chứa khóa cần tìm.

Ví dụ
Cho R = {A, B, C, D, E, G, H, I}, F = {AC → B, BI → ACD, ABC → D, H → I, ACE → BCG, CG → AE}. Tìm K.
(1) K = R = {A, }
(2) Lần lượt loại thử các thuộc tính trong K
Loại A, ta có $\{B, C, D, E, G, H, I\}^+ = R$ vì CG → AE
Loại B, ta có $\{C, D, E, G, H, I\}^+ = R$ vì CG → AE và AC → B
Loại C, ta có $\{D, E, G, H, I\}^+ \neq R$, không loại C.
Loại D, ta có $\{C, E, G, H, I\}^+ = R$ vì CG → AE, AC → B, ABC → D
Loại E, ta có $\{C, G, H, I\}^+ = R$ vì CG → AE, AC → B, ABC → D
Loại G, ta có $\{C, H, I\}^+ \neq R$, không loại G.
Loại H, ta có $\{C, G, I\}^+ \neq R$, không loại H.
Loại I, ta có $\{C, G, H\}^+ = R$, vì CG → AE, AC → B, ABC → D, H → I
Như vậy, K = {C, G, H} là khóa.
Nhận xét:
- Các thuộc tính không xuất hiện trong cả vế trái và vế phải của phụ thuộc hàm trong F phải có trong khóa.
- Các thuộc tính chỉ xuất hiện trong vế trái của phụ thuộc hàm trong F cũng phải thuộc khóa.

6.4. Phủ và tập phụ thuộc hàm tương đương

Như đã nói, với mọi tập phụ thuộc hàm F, tập F^+ có thể chứa một lượng lớn các phụ thuộc hàm. Điều này càng đúng với những tập F có một lượng lớn các phụ thuộc hàm. Do đó nếu có được phương pháp để tính được F^+ từ một tập phụ thuộc hàm có số lượng phụ thuộc hàm ít hơn F thì thật tốt. Tập phụ thuộc hàm thỏa điều kiện này được gọi là *tập phụ thuộc hàm tương đương* (equivalent set). Ta có thể định nghĩa vấn đề này như sau:
Cho hai tập phụ thuộc hàm F và G được định nghĩa trên cùng một lược đồ quan hệ. Chúng ta nói F và G là tương đương nhau nếu và chỉ nếu $F^+ = G^+$. Ta sẽ ký hiệu sự tương đương đó như sau $F \equiv G$. Bất cứ khi nào $F^+ = G^+$, ta đều nói rằng F *phủ* (cover) G và ngược lại. Nếu G phủ F và nếu không có tập con thật sự[1] H nào của G thỏa $H^+ = G^+$ thì ta nói G là *phủ không dư thừa* (nonredundant cover) của F.
Thuật toán để tìm phủ không dư thừa G được trình bày dưới đây. Đầu vào của thuật toán này là tập phụ thuộc hàm F; còn đầu ra là phủ không dư thừa của F.

Thuật toán tìm phủ không dư thừa (NonRedundant Cover Algorithm)

(1) Khởi động G chính là F, tức G = F.
(2) Kiểm tra tính dư thừa của từng phụ thuộc hàm của G bằng cách dùng thuật toán Thành viên cho đến khi không còn phụ thuộc hàm nào của G để kiểm tra nữa.
(3) Bây giờ G sẽ là phủ không dư thừa của F.

Bạn nên nhớ rằng với một tập F cho trước có thể có nhiều phủ không dư thừa vì thứ tự xét các phụ thuộc hàm có thể khác nhau. Nói cách khác, sự có mặt hay vắng mặt một phụ thuộc hàm nào đó có thể quyết định các phụ thuộc hàm khác có dư thừa hay không. Hơn nữa nếu ta có thể tìm được một phủ không dư thừa, thì phủ này có thể là chưa tối thiểu. Nghĩa là có thể có các phủ không dư thừa khác có ít phụ thuộc hàm hơn.

Ví dụ
Tìm phủ không dư thừa G của tập F = {X → YZ, ZW → P, P → Z, W → XPQ, XYQ → YW, WQ → YZ}.
(1) Gán G = F.
(2) a. Kiểm tra tính dư thừa của X → YZ. Bây giờ G = {ZW → P, P → Z, W → XPQ, XYQ → YW, WQ → YZ}
$T_1 = \{X\}$. Vì không có phụ thuộc hàm nào của G có tất cả các tập thuộc tính xác định của nó trong T_1, nên X → YZ là không dư thừa. Chú ý là $\{YZ\} \not\subset T_1$.
b. Kiểm tra tính dư thừa của ZW → P. Bây giờ G = {X → YZ, P → Z, W → XPQ, XYQ → YW, WQ → YZ}
$T_1 = \{ZW\}$. Ta thấy tập thuộc tính xác định của phụ thuộc hàm W → XPQ có tất cả các thuộc tính xác định của nó trong T_1, do đó $T_2 = T_1 \cup \{XPQ\} = \{ZWXPQ\}$. Vì P, vế phải của phụ thuộc hàm ZW → P, là một phần tử của T_2, nên phụ thuộc hàm ZW → P là dư thừa và có thể loại khỏi G.
c. Kiểm tra tính dư thừa của P → Z. Bây giờ G = {X → YZ, W → XYQ, XYQ → YW, WQ → YZ}
$T_1 = \{P\}$. Vì không có phụ thuộc hàm nào của G có tất cả các tập thuộc tính xác định của nó trong T_1, nên P → Z là không dư thừa. Chú ý là $\{Z\} \not\subset T_1$.
d. Kiểm tra tính dư thừa của W → XPQ. Bây giờ G = {X → YZ, P → Z, XYQ → YW, WQ → YZ}
$T_1 = \{W\}$. Vì không có phụ thuộc hàm của G có tất cả các tập thuộc tính xác định của nó trong T_1, nên W → XPQ là không dư thừa. Chú ý là $\{XPQ\} \not\subset T_1$.
e. Kiểm tra tính dư thừa của XYZ → YW. Bây giờ G = {X → YZ, P → Z, XYQ → YW, WQ → YZ}

[1] H là *tập con thật sự* (proper subset) của G nếu và chỉ nếu $H \subset G$ và $H \neq G$.

$T_1 = \{XYZ\}$. Ta thấy tập thuộc tính xác định của phụ hàm $X \to YZ$ có tất cả các tập thuộc tính xác định của nó trong T_1, do đó $T_2 = T_1 \cup \{YZ\} = \{XYZ\}$. Vì không còn phụ thuộc hàm nào có tập thuộc tính xác định của nó trong T_2 nên $XYZ \to YW$ là không dư thừa. Chú ý là $\{YW\} \not\subset T_2$.

f. Kiểm tra tính dư thừa của $WQ \to YZ$. Bây giờ $G = \{X \to YZ, P \to Z, W \to XYQ, XYQ \to YW\}$

$T_1 = \{WQ\}$. Ta thấy tập thuộc tính xác định của phụ thuộc hàm $W \to XPQ$ có tất cả các tập thuộc tính xác định của nó trong T_1, do đó $T_2 = T_1 \cup \{XPQ\} = \{WQXP\}$. Vì tập thuộc tính xác định của $X \to YZ$ là một phần tử của T_2, do đó $T_3 = T_2 \cup \{YZ\} = \{WQXPYZ\}$. Vì YZ, vế phải của phụ thuộc hàm $WQ \to YZ$, là một phần tử của T_3, nên phụ thuộc hàm đó là dư thừa và có thể loại khỏi G.

(3) Không còn phụ thuộc hàm nào trong G để kiểm tra nên phủ không dư thừa F là tập $G = \{X \to YZ, P \to Z, W \to XPQ, XYQ \to YW\}$.

6.5. Thuộc tính dư thừa

Nếu F là một tập phụ thuộc hàm không dư thừa thì không thể làm cho F nhỏ hơn bằng cách bỏ bớt bất kỳ phụ thuộc hàm nào trong đó, vì nếu chúng ta làm như vậy, thì tập kết quả không tương đương với F nữa. Tuy nhiên vẫn có trường hợp có thể giảm bớt kích cỡ của các phụ thuộc hàm của F bằng cách bỏ bớt các thuộc tính dư thừa (extraneous attributes) ở vế trái hoặc bỏ bớt các thuộc tính dư thừa ở vế phải.
Có hai khái niệm có thể phát biểu là:

- Cho F là một tập phụ thuộc hàm trên lược đồ R và cho $A_1A_2 \to B_1B_2$ là một phụ thuộc hàm trong F. Thuộc tính A_1 là một *thuộc tính dư thừa ở vế trái* (extraneous left attribute) trong phụ thuộc hàm $A_1A_2 \to B_1B_2$ nếu và chỉ nếu $F \equiv F - \{A_1A_2 \to B_1B_2\} \cup \{A_2 \to B_1B_2\}$. Nói cách khác, thuộc tính A_1 là một thuộc tính dư thừa ở vế trái trong phụ thuộc hàm $A_1A_2 \to B_1B_2$, nếu có thể bỏ bớt thuộc tính A_1 khỏi tập thuộc tính xác định của $A_1A_2 \to B_1B_2$ mà không làm thay đổi bao đóng của F.
- Tương tự, ta có thể định nghĩa thuộc tính B_1 là một *thuộc tính dư thừa ở vế phải* (extraneous right attribute) trong phụ thuộc hàm $A_1A_2 \to B_1B_2$ nếu và chỉ nếu $F \equiv F - \{A_1A_2 \to B_1B_2\} \cup \{A_1A_2 \to B_2\}$. Nói cách khác, thuộc tính B_1 là một thuộc tính dư thừa ở vế phải trong phụ thuộc hàm $A_1A_2 \to B_1B_2$, nếu có thể bỏ bớt thuộc tính B_1 khỏi tập thuộc tính vế phải của $A_1A_2 \to B_1B_2$ mà không làm thay đổi bao đóng của F. Nếu đã ngầm hiểu tập F, thì chỉ cần nói ngắn gọn là thuộc tính dư thừa thay vì phải nói rõ là thuộc tính dư thừa vế trái hay vế phải. Các phụ thuộc hàm không có thuộc tính dư thừa vế trái được gọi là các phụ thuộc hàm đã *cực tiểu vế trái* (left-reduced FD). Tương tự, các phụ thuộc hàm không có tính dư thừa vế phải được gọi là các phụ thuộc hàm đã *cực tiểu vế phải* (right-reduced FD). Nếu tất cả các phụ thuộc hàm của F đều là các phụ thuộc hàm đã cực tiểu vế trái thì F được gọi là tập cực tiểu vế trái (left-reduced set). Tương tự, nếu tất cả các phụ thuộc hàm của F đều là các phụ thuộc hàm đã cực tiểu vế phải thì F được gọi là tập cực tiểu vế phải (right-reduced set). Trong tài liệu này ta chỉ quan tâm đến các phụ thuộc hàm đã cực tiểu vế trái vì chúng hữu ích hơn.

Thuật toán để bỏ bớt các thuộc tính vế trái sẽ được trình bày dưới đây. Đầu vào của thuật toán này là tập phụ thuộc hàm F, còn đầu ra là phủ G đã được cực tiểu hóa vế trái.

Thuật toán cực tiểu hóa vế trái (LeftReduce Algorithm)

(1) Khởi động G chính là F, tức G = F

(2) Với mỗi phụ thuộc hàm $A_1A_2...A_i...A_n \to Y$ trong G ta thực hiện bước 3 cho đến khi không còn phụ thuộc hàm nào trong G để áp dụng bước này. Thuật toán sẽ chấm dứt khi tất cả các phụ thuộc hàm trong G đều đã được thực hiện trong bước này.

(3) Với mỗi thuộc tính A_i trong tập thuộc tính xác định của phụ thuộc hàm được chọn trong bước trước ta sẽ thực hiện bước 4 cho đến khi tất cả các thuộc tính đều được kiểm tra. Sau khi kiểm tra tất cả các thuộc tính của một phụ thuộc hàm thì lặp lại bước 2.

(4) Kiểm tra xem có phải tất cả các thuộc tính của Y (vế phải của phụ thuộc hàm đang xét) có phải plà các phần tử của bao đóng của $A_1A_2...A_n$ (chú ý là ta đã bỏ thuộc tính A_i khỏi tập thuộc tính xác định của phụ thuộc hàm) hay không. Nếu thuộc về thì bỏ thuộc tính A_i khỏi tập thuộc tính xác định của phụ thuộc hàm đang kiểm tra vì A_i là thuộc tính dư thừa ở vế trái. Còn nếu tất cả các thuộc tính của Y đều không thuộc về bao đóng của $A_1A_2...A_n$ thì thuộc tính A_i không phải là thuộc tính dư thừa vế trái nên nó phải được giữ lại trong tập thuộc tính xác định của phụ thuộc hàm đang xét.

Khi thuật toán chấm dứt, tập G sẽ chứa phủ đã cực tiểu hóa vế trái của F.

Chú ý: Thuật toán có thể chạy nhanh hơn một chút nếu ta có thể nhận ra rằng bước 2 chỉ có thể được áp dụng cho những phụ thuộc hàm có tập thuộc tính xác định có từ hai thuộc tính trở lên.
Ví dụ sau sẽ minh họa cho cách dùng thuật toán này.

Ví dụ
Hãy cực tiểu hóa tập hợp $F = \{X \to Z, XY \to WP, XY \to ZWQ, XZ \to R\}$ bằng cách bỏ đi các thuộc tính dư thừa vế trái.
$F = \{X \to Z, XY \to WP, XY \to ZWQ, XZ \to R\}$
Ta sẽ không cần xét đến các phụ thuộc hàm có tập thuộc tính xác định là một thuộc tính vì chắc chắn rằng không có thuộc tính dư thừa trong đó.

(1) Gán $G = \{X \to Z, XY \to WP, XY \to ZWQ, XZ \to R\}$

(2) (lần 1) Chọn $XY \to WP$ và thực hiện bước 3. [Ta cần lặp lại bước này cho tất cả các phụ thuộc hàm trong G.]

(3-4) (lần 1) Từ phụ thuộc hàm đã chọn ta sẽ chọn một thuộc tính trong tập thuộc tính xác định. Ở đây sẽ chọn X rồi kiểm tra vế phải $Y \to WP$ xem có thuộc về bao đóng của Y không. Rõ ràng bao đóng của $Y^+ = \{Y\}$. Vì $\{WP\} \not\subset Y^+$ nên thuộc tính X không là thuộc tính dư thừa vế trái.

(3-4) (lần 2) Chọn thuộc tính Y trong tập thuộc tính xác định của phụ thuộc hàm $XY \to WP$ rồi kiểm tra vế phải của $X \to WP$ xem có thuộc về bao đóng của X không. Rõ ràng bao đóng của $X^+ = \{XZR\}$. Vì $\{WP\} \not\subset X^+$ nên thuộc tính Y không là thuộc tính dư thừa vế trái.

(2) (lần 2) Chọn $XY \to ZWQ$ và thực hiện bước 3.

(3-4) (lần 1) Từ phụ thuộc hàm đã chọn ta sẽ chọn một thuộc tính trong tập thuộc tính xác định. Ở đây sẽ chọn X rồi kiểm tra vế phải của $Y \rightarrow ZWQ$ xem có thuộc về bao đóng của Y không. Rõ ràng bao đóng của $Y^+ = \{Y\}$. Vì $\{ZWQ\} \not\subset Y^+$ nên thuộc tính X không là thuộc tính dư thừa vế trái.

(3-4) (lần 2) Chọn thuộc tính Y trong tập thuộc tính xác định của phụ thuôc hàm $XY \rightarrow ZWQ$ rồi kiểm tra vế phải của $X \rightarrow ZWQ$ xem có thuộc về bao đóng của X không. Rõ ràng bao đóng của $X^+ = \{XZR\}$. vì $\{ZWQ\} \not\subset X^+$ nên thuộc tính Y không là thuộc tính dư thừa vế trái.

(2) (lần 3) Chọn $XZ \rightarrow R$

(3-4) (lần 1) Từ phụ thuộc hàm đã chọn ta sẽ chọn một thuộc tính trong tập thuộc tính xác định. Ở đây sẽ chọn X rồi kiểm tra vế phải của $Z \rightarrow R$ xem có thuộc về bao đóng của Z không. Rõ ràng bao đóng của $Z^+ = \{Z\}$. Vì $\{R\} \not\subset Z^+$ nên thuộc tính X không là thuộc tính dư thừa vế trái.

(3-4) (lần 2) Chọn thuộc tính Z của tập thuộc tính xác định của phụ thuộc hàm $XZ \rightarrow R$ rồi kiểm tra vế phải của $X \rightarrow R$ xem có thuộc về bao đóng của X không. Rõ ràng bao đóng của $X^+ = \{XZR\}$. Vì $\{R\} \not\subset X^+$ nên thuộc tính Z là thuộc tính dư thừa vế trái, nó sẽ được xóa khỏi phụ thuộc hàm.

Vì không còn phụ thuộc hàm nào để kiểm tra trong G nữa nên ta có phủ đã cực tiểu hóa vế trái của F là $G = \{X \rightarrow Z, XY \rightarrow WP, XY \rightarrow ZWQ, X \rightarrow R\}$.

6.6. Phủ hợp quy

Cho tập phụ thuộc hàm F, khi đó *phủ hợp quy* (canonical cover), được ký hiệu là F_c, là một tập các phụ thuộc hàm đồng thời thỏa các điều kiện sau:

(1) Mọi phụ thuộc hàm của F_c đều ở dạng đơn (simple). Nghĩa là vế phải của mọi phụ thuộc hàm của F_c chỉ có một thuộc tính.

(2) F_c đã được cực tiểu hóa vế trái.

(3) F_c là không dư thừa.

Ví dụ

Cho tập F như ví dụ trên. Hãy tìm phủ hợp quy của F.

Gọi $F_c = \{X \rightarrow Z, XY \rightarrow WP, XY \rightarrow ZWQ, XZ \rightarrow R\}$

Từ ví dụ trên ta biết rằng phủ cực tiểu hóa vế trái của F là $F_c = \{X \rightarrow Z, XY \rightarrow WP, XY \rightarrow ZWQ, X \rightarrow R\}$. Lần lượt áp dụng tiên đề Projectivity ta có thể viết lại tập này như sau:

$F_c = \{X \rightarrow Z, XY \rightarrow W, XY \rightarrow P, XY \rightarrow Z, XY \rightarrow W, XY \rightarrow Q, X \rightarrow R\}$

Chú ý là trong tập này $XY \rightarrow Z$ là dư thừa, đơn giản vì ta đã có $X \rightarrow Z$. Do đó có thể bỏ $XY \rightarrow Z$ khỏi F. Hơn nữa vì F là tập hợp nên ta có thể bỏ bớt phần tử trùng ($XY \rightarrow W$) nên ta có $F_c = \{X \rightarrow Z, XY \rightarrow W, XY \rightarrow P, XY \rightarrow Q, X \rightarrow R\}$. Trong tập này, tất cả các phụ thuộc hàm đều ở dạng đơn, đều đã được cực tiểu hóa vế trái và không dư thừa, nên đó chính là phủ hợp quy của F.

Bài tập có lời giải

1. Cho quan hệ r(R) dưới đây. Cho biết các phụ thuộc hàm sau có thỏa quan hệ hay không.

a. $A \rightarrow B$ b. $A \rightarrow C$ c. $AB \rightarrow C$

d. $C \rightarrow A$ e. $BC \rightarrow A$ f. $AC \rightarrow B$

r

A	B	C
1	4	2
3	5	6
3	4	6
7	3	8
9	1	0

a. $A \rightarrow B$ không thỏa vì có hai bộ (3,5,6) và (3,4,6), trong đó giá trị tại thuộc tính A đều bằng 3 nhưng giá trị tại thuộc tính B lại là 5 và 4.

b. $A \rightarrow C$ thỏa vì hầu hết các bộ có giá trị tại thuộc tính A là duy nhất. Chỉ có hai bộ có giá trị tại A giống nhau là (3,5,6) và (3,4,6) thì cũng có giá trị tại thuộc tính C là giống nhau.

c. $AB \rightarrow C$ thỏa vì thực ra không có hai bộ nào có các giá trị tại các thuộc tính A và B giống nhau.

d. $C \rightarrow A$ thỏa vì hầu hết các bộ giá trị tại thuộc tính C là duy nhất. Chỉ có hai bộ có giá trị tại C giống nhau là (3,5,6) và (3,4,6) thì cũng có giá trị tại thuộc tính A là giống nhau.

e. $BC \rightarrow A$ thỏa thỏa vì thực ra không có hai bộ nào có các giá trị tại các thuộc tính B và C giống nhau.

f. $AC \rightarrow B$ không thỏa vì có hai bộ (3,5,6) và (3,4,6), trong đó giá trị tại thuộc tính A và C đều bằng nhau nhưng giá trị tại thuộc tính B lại là 5 và 4.

2. Hãy chứng minh phép nội suy "nếu $AB \rightarrow C$ và $C \rightarrow A$ thì $C \rightarrow B$" là không đúng ở mọi quan hệ r(R). Giả sử A, B, C và D là các thuộc tính của lược đồ quan hệ r(R).

Ta có thể dùng một quan hệ có hai bộ thỏa $AB \rightarrow C$ và $C \rightarrow A$ nhưng không thỏa $C \rightarrow B$ để phản chứng cho phép nội suy trên. Sau đây là một thể hiện của quan hệ r(R):

r

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2

Rõ ràng AB → C thỏa quan hệ r vì ta có giá trị tại các thuộc tính A và B đều khác nhau. Còn C → A thì hiển nhiên. Tuy nhiên C → B là không thỏa vì cùng là c_1 nhưng tại B ta có b_1 và b_2.

3. Giả sử có quan hệ r(X, Y, Z) đồng thời thỏa cả hai điều kiện sau, vậy bản số của $\pi_Y(\sigma_{X=x}(r))$ là gì?
(1) X → Y thỏa trên r
(2) $\sigma_{X=x}(r)$ là quan hệ khác rỗng
Không mất tính tổng quát ta có thể giả sử X và Y là các thuộc tính đơn, và mọi bộ của r có một giá trị khác nhau dưới thuộc tính Y.
Theo định nghĩa phép chọn ta có $\sigma_{X=x}(r)$ sẽ trả về tất cả các bộ của r có giá trị tại thuộc tính X là x. Vì X → Y nên các bộ kết quả của phép chọn trên cũng có các giá trị bằng nhau tại thuộc tính Y. Do đó phép chiếu $\sigma_{X=x}(r)$ trên thuộc tính Y chỉ có thể trả về một bộ. Do đó nếu ta ký hiệu bản số của quan hệ r là |r| thì ta có $|\pi_Y(\sigma_{X=x}(r))| = 1$. Nếu ta giả sử $\sigma_{X=x}(r)$ có thể rỗng thì $0 \le |\pi_Y(\sigma_{X=x}(r))| \le 1$.

4. Giả sử AB → C, C → D và D → A đều thỏa trên r(R). Vậy các khóa dự tuyển (candidate key) của quan hệ trên là gì? Khóa nào là chính (primary key)? Đâu là các thuộc tính khóa (prime attribute)? Quan hệ này có siêu khóa (superkey) không?
Một tập thuộc tính của r sẽ là khóa dự tuyển của quan hệ r nếu và chỉ nếu tập đó có thể xác định mọi thuộc tính của quan hệ. Trong ví dụ này, mọi khóa dự tuyển nếu có đều phải xác định được các thuộc tính của A, B, C và D. Vậy các khóa dự tuyển là AB và DB.
- AB là khóa dự tuyển (các phụ thuộc hàm cho sẵn là AB → C, C → D và D → A)
Nếu AB là khóa của r thì các phụ thuộc hàm sau đều đúng: AB → A, AB → B, AB → C và AB → D. Sử dụng các tiên đề suy luận và các phụ thuộc hàm đã có sẵn ta có:
(1) AB → A và AB → B (tiên đề Refexivity)
(2) AB → C (cho sẵn)
(3) AB → C và C → D (cho sẵn) nên AB → D (tiên đề Transitivity)
Do đó AB là khóa dự tuyển của quan hệ r vì nó có thể xác định mọi thuộc tính trong r.
- DB là khóa dự tuyển (các phụ thuộc hàm cho sẵn là AB → C, C → D và D → A)
Nếu DB là khóa của r thì các phụ thuộc hàm sau đều đúng: DB → A, DB → B, DB → C và DB → D. Sử dụng các tiên đề suy luận và các phụ thuộc hàm đã có sẵn ta có:
(1) DB → B và DB → D (tiên đề Refexivity)
(2) DB → D (bước trước) và D → A (cho sẵn) nên ta có DB → A (tiên đề Transitivity)
(3) DB → A (bước trước) và AB → C (cho sẵn) nên ta có DBB → C (tiên đề Pseudotransitivity)
(4) Vì BB = B, nên thay vì viết DBB → C ta có thể viết DB → C.
Do đó DB là khóa dự tuyển của quan hệ r vì nó có thể xác định mọi thuộc tính trong r.
Chú ý là có đến hai khóa dự tuyển nhưng chỉ có thể chọn một để làm khóa chính cho quan hệ. Chọn khóa dự tuyển nào làm khóa chính là tùy theo bài toán cụ thể và tùy DBA.
Các thuộc tính khóa của quan hệ này là A, B và D. Nhắc lại, thuộc tính khóa là thuộc tính có tham gia vào khóa.
Số lượng siêu khóa là khá lớn: ABC, ABD, ABCD, DBA, DBC, và DBAC.

5. Cho lược đồ quan hệ r(A, B, C, D) và tập phụ thuộc hàm F có A → B và BC → D. Hãy cho biết phụ thuộc hàm nào dưới đây là có thể suy dẫn từ F.
a. AC → D b. B → D c. AD → B

a. AC → D có thể suy dẫn từ A → B và BC → D và tiên đề Pseudotransitivity.
b. B → D không thể suy dẫn được từ F. Vì ta đã có BC → D nên không thể có B → D. Thể hiện sau của r sẽ minh họa cho điều này.

r

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_2	d_2

Rõ ràng ở đây ta có BC → D nhưng không có B → D.
c. AD → B có thể được suy dẫn từ A → B nhờ tiên đề Augmentation. Điều này có nghĩa là nếu ta chỉ cần A cũng có thể xác định được B thì nếu có thêm D cũng không làm thay đổi ý trên (A → B). Đôi khi thuộc tính D còn bị gọi là thuộc tính dư thừa.

6. Hãy chứng minh rằng tiên đề Transitivity là một trường hợp đặc biệt của tiên đề Pseudotransitivity.
Nếu W = ∅ trong tiên đề Pseudotransiticity (nếu X → Y và YW → Z thì XW → Z) thì ta có thể viết lại tiên đề này như sau: nếu X → Y và Y∅ → Z thì X∅ → Z. Vì X ∪ ∅ = X nên ta có: nếu X → Y và Y → Z thì X → Z.

7. Tập tích lũy (set accumulation) được suy dẫn từ các tiên đề suy luận: nếu X → YZ và Z → W, thì X → YZW. Hãy chứng minh điều này.
(1) X → YZ và (2) Z → W. Dùng tiên đề Augmentation, tăng (2) với YZ để có YZZ → YZW.
Do ZZ = Z, chúng ta có (3) YZ → YZW. Cuối cùng, dùng tiên đề Transitivity, từ (1) và (3), chúng ta có X → YZW.

8. Cho quan hệ r(A, B, C) và tập phụ thuộc hàm F = {AB → C, B → D, D → B}, hãy tìm các khóa dự tuyển của quan hệ. Chỉ rõ các thuộc tính khóa.
Có hai khóa dự tuyển AB và AD. Nghĩa là hai khóa dự tuyển đó có thể xác định được mọi thuộc tính trong quan hệ r.
- AB là khóa dự tuyển
(1) AB → A và AB → B (tiên đề Refexivity)
(2) AB → C (cho sẵn)
(3) AB → B (bước 1) và B → D (cho sẵn) nên AB → D (tiên đề Transitivity)
Vì AB có thể xác định mọi thuộc tính trong quan hệ nên nó là khóa dự tuyển.
- AD là khóa dự tuyển
(1) AD → A và AD → D (tiên đề Refexivity)
(2) AD → D (bước 1) và D → B (cho sẵn) nên AD → B (tiên đề Transitivity)
(3) AD → B (bước 2) và AB → C (cho sẵn) nên AAD → C (tiên đề Pseudotransitivity)
Vì A ∪ A = A nên ta có AD → C.
Vì AD có thể xác định mọi thuộc tính trong quan hệ nên nó là khóa dự tuyển.
Các thuộc tính khóa là A, B và D.

9. Cho F = {XY → W, Y →Z, WZ → P, WQ → QR, Q → X}, hãy chứng minh F |= XY → P bằng cách dùng các tiên đề suy luận.

(1) XY → W (cho sẵn) và WZ → P (cho sẵn) ta có XYZ → P (tiên đề Pseudotransitivity)
(2) XYZ → P (bước trước) và Y → Z (cho sẵn) ta có XYY → P (tiên đề Pseudotransitivity). Vì Y ∪ Y = Y nên ta có XY → P.

10. Cho F như bài tập trên, hãy chứng minh F |= XY → P bằng cách tính $(XY)^+$
(1) G = {XY → W, Y → Z, WZ → P, WQ → QR, Q → X}
(2) T_1 = {XY}
(3-a) Vì tất cả các thuộc tính xác định của XY → W đều thuộc về T_1 nên ta có thể tạo tập T_2 mới từ T_1 cộng thêm vế phải của phụ thuộc hàm này, nghĩa là $T_2 = T_1 \cup \{W\}$ = {XYW}. Khi đó G = G – {XY → W} = {Y → Z, WZ → P, WQ → QR, Q → X}
(3-a) (lần 2) Vì tất cả các thuộc tính xác định của Y → Z đều thuộc về T_2 nên $T_3 = T_2 \cup \{Z\}$ = {XYWZ}. Khi đó G = G – {Y → Z} = {WZ → P, WQ → QR, Q → X}.
(3-a) (lần 3) Vì tất cả các thuộc tính xác định của WZ → P đều thuộc về T_3 nên $T_4 = T_3 \cup \{P\}$ = {XYWZP}. Khi đó G = G – {WZ → P} = {WQ → QR, Q → X}.
(3-b) Vì không còn phụ thuộc hàm nào có tất cả các thuộc tính xác định của nó thuộc về T_4 nên thuật toán chấm dứt tại đây và $(XY)^+$ = {XYZWP}. Vì P là vế phải của XY → P thuộc về $(XY)^+$ nên ta có thể kết luận rằng phụ thuộc hàm XY → P có thể được suy dẫn từ F.
Một cách khác để chứng minh XY → P bằng cách dùng các tiên đề suy luận là:
Ta có F = {XY → W, Y → Z, WZ → P, WQ → QR, Q → X}
(1) Y → Z (cho sẵn) nên ta có XY → Z (tiên đề Augmentation)
(2) XY → Z (bước trước) và WZ → P (cho sẵn) nên ta có XYW → P (tiên đề Pseudotransitivity)
(3) XYW → P (bước trước) và XY → W (cho sẵn) nên ta có XYXY → P (tiên đề Pseudotransitivity).
Vì X ∪ X = X và Y ∪ Y = Y nên ta có thể viết lại là XY → P.

11. Hãy loại bỏ các phụ thuộc hàm dư thừa của F = {X → Y, Y → X, Y → Z, Z → Y, X → Z, Z → X} bằng cách dùng thuật toán Thành viên.

- Kiểm tra tính dư thừa của X → Y (tạm thời bỏ X → Y khỏi F)
(1) G = {Y → X, Y → Z, Z → Y, X → Z, Z → X}
(2) T_1 = {X}
(3-a) Vì thuộc tính xác định của X → Z thuộc về T_1 nên ta có thể thêm vế phải của phụ thuộc hàm đó vào T_1 để có $T_2 = T_1 \cup \{Z\}$ = {XZ}. Chú ý rằng $Y \notin T_2$. Vì vậy bỏ X → Z khỏi G và ta có G = G – {X → Z} = {Y → X, Y → Z, Z → Y, Z → X}.
(3-a) (lần 2) Vì thuộc tính xác định của Z → Y thuộc về T_2 nên $T_3 = T= \cup \{Y\}$ = {XZY}. Vì $Y \in T3$ nên X → Y là phụ thuộc hàm dư thừa trong G và có thể loại bỏ nó đi.
- Kiểm tra tính dư thừa của Y → X (tạm thời bỏ Y → X khỏi F)
(1) G = {Y → Z, Z → Y, X → Z, Z → X}
(2) T_1 = {Y}
(3-a) Vì thuộc tính xác định của Y → Z thuộc về T_1 nên $T_2 = T_1 \cup \{Z\}$ = {YZ}. G = {Z → Y, X → Z, Z → X}.
(3-a) (lần 2) Vì thuộc tính xác định của Z → X thuộc về T_2 nên $T_3 = T_2 \cup \{X\}$ = {YZX}. Vì $X \in T_3$ nên Y → X là phụ thuộc hàm dư thừa trong G và vì thế có thể loại bỏ nó đi. Nghĩa là G = {Y → Z, Z → Y, X → Z, Z → X}
- Kiểm tra tính dư thừa của Y → Z (tạm thời bỏ Y → Z khỏi F)
(1) G = {Z → Y, X → Z, Z → X}
(2) T_1 = {Y}
(3-b) Vì không có phụ thuộc hàm nào trong G có tất cả các thuộc tính xác định đều thuộc về T_1 nên Y → Z là không dư thừa.
- Kiểm tra tính dư thừa của Z → Y (tạm thời bỏ Z → Y khỏi F)
(1) G = {Y → Z, X → Z, Z → X}
(2) T_1 = {Z}
(3-a) Vì thuộc tính xác định của Z → X thuộc về T_1 nên $T_2 = T_1 \cup \{X\}$ = {ZX}. Loại Z → X khỏi G ta có G = {Y → Z, X → Z}

(3-a) (lần 2) Vì thuộc tính xác định của $X \to Z$ thuộc về T_2 nên $T_3 = T_2 \cup \{Z\} = \{ZX\}$. Loại $X \to Z$ khỏi G ta có $G = \{Y \to Z\}$.
(3-b) Vì không có phụ thuộc hàm nào trong G có tất cả các thuộc tính xác định thuộc về T_3 nên $Z \to Y$ là không dư thừa.
- Kiểm tra tính dư thừa của $X \to Z$ (tạm thời bỏ $X \to Z$ khỏi F)
(1) $G = \{Y \to Z, Z \to Y, Z \to X\}$
(2) $T_1 = \{Z\}$
(3-b) Vì không có phụ thuộc hàm nào trong G có tất cả các thuộc tính xác định đều thuộc về T_1 nên $X \to Z$ là không dư thừa.
- Kiểm tra tính dư thừa của $Z \to X$ (tạm thời bỏ $Z \to X$ khỏi F)
(1) $G = \{Y \to Z, Z \to Y, X \to Z\}$
(2) $T_1 = \{Z\}$
(3-a) Vì thuộc tính xác định của $Z \to Y$ thuộc về T_1 nên $T_2 = T_1 \cup \{Y\} = \{ZY\}$. Loại $Z \to Y$ khỏi G ta có $G = \{Y \to Z, X \to Z\}$.
(3-a) (lần 2) Vì thuộc tính xác định của $Y \to Z$ thuộc về T_2 nên $T_3 = T_2 \cup \{Z\} = \{ZY\}$. Loại $Y \to Z$ khỏi G ta có $G = \{X \to Z\}$.
(3-b) Vì không có phụ thuộc hàm nào trong G có tất cả các thuộc tính xác định đều thuộc về T_4 nên $Z \to X$ là không dư thừa.
Vây tập phụ thuộc hàm không dư thừa $F = \{Y \to Z, Z \to Y, X \to Z, Z \to X\}$.

12. Hãy thu giảm tập $F = \{X \to YW, XW \to Z, Z \to Y, XY \to Z\}$ bằng cách bỏ bớt các thuộc tính trái dư thừa.
Không cần xét các phụ thuộc hàm chỉ có một thuộc tính ở vế trái.
(1) $G = \{X \to YW, XW \to Z, Z \to Y, XY \to Z\}$
(2) (lần 1) Chọn $XW \to Z$ và thực hiện bước 3. [Ta cần lặp lại bước này cho tất cả các phụ thuộc hàm trong G.]
(3-4) (lần 1) Từ thuộc tính xác định của phụ thuộc hàm trên ta chọn thuộc tính X và kiểm tra xem Z có thuộc $(W)^+$ trên G hay không. Ta có $(W)^+ = \{W\}$. Vì $\{Z\}$ không phải là con của $(W)^+$ nên ta kết luận X không phải là thuộc tính dư thừa.
(2) (lần 2) Từ thuộc tính xác định của phụ thuộc hàm $XW \to Z$ ta chọn thuộc tính W và kiểm tra xem Z có thuộc $(X)^+$ trên G hay không. Vì $(X)^+ = \{XYWZ\}$ và $\{Z\}$ là con của $(X)^+$ nên ta kết luận W là thuộc tính dư thừa vế trái và có thể loại khỏi vế trái của phụ thuộc hàm $XW \to Z$.
(2) (lần 1) Chọn $XY \to Z$. Tập G mới là $G = \{X \to YW, X \to Z, Z \to Y, XY \to Z\}$
(3-4) (lần 1) Từ thuộc tính xác định của phụ thuộc hàm $XY \to Z$ ta chọn thuộc tính X và kiểm tra xem $\{Z\}$ có thuộc $(Y)^+$ trên G hay không. Vì $(Y)^+ = \{Y\}$ nên $\{Z\}$ không phải là con của $(Y)^+$ và ta kết luận X không phải là thuộc tính dư thừa.
(3-4) (lần 2) Từ thuộc tính xác định của phụ thuộc hàm $XY \to Z$ ta chọn thuộc tính Y và kiểm tra xem $\{Z\}$ có thuộc $(X)^+$ trên G hay không. Vì $(X)^+ = \{XYWZ\}$ nên $\{Z\}$ là con của $(Y)^+$ và ta kết luận Y là thuộc tính dư thừa vế trái và có thể loại khỏi vế trái phụ thuộc hàm $XY \to Z$.
Thuật toán chấm dứt tại đây vì không còn phụ thuộc hàm nào nữa để xét. Sau khi bỏ các phụ thuộc hàm trùng, thì tập $F = \{X \to YW, X \to Z, Z \to Y\}$ là tập cực tiểu vế trái.

13. Hãy chuyển tập cực tiểu vế trái từ bài tập trên thành phủ hợp quy cho F.
Để có được phủ hợp quy cho F, ta cần biến đổi để tất cả các phụ thuộc hàm trong F đều có vế phải là một thuộc tính. Dùng tiên đề Projectivity ta có thể viết lại F như say: $F = \{X \to Y, X \to W, X \to Z, Z \to Y\}$. Rõ ràng $X \to Y$ là dư thừa vì nó có thể được suy dẫn từ $X \to Z$ và $Z \to Y$ bằng cách dùng tiên đề Transitivity. Loại $X \to Y$ khỏi F ta có $F_c = \{X \to W, X \to Z, Z \to Y\}$

Quá trình chuẩn hóa

1. Mở đầu

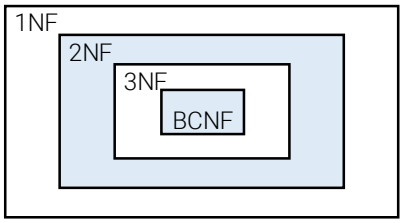
Trong các cơ sở dữ liệu quan hệ, thuật ngữ *chuẩn hóa* (normalization, quá trình này được E. F. Codd giới thiệu năm 1971) là để mô tả một quá trình từng bước và có thể đảo ngược, trong đó một tập các quan hệ sẽ được thay thế bằng một tập các quan hệ khác có dạng đơn giản và cấu trúc chính quy hơn. Mỗi bước, được tham chiếu như một *dạng chuẩn* (normal form), định nghĩa một tập các điều kiện (kiểm tra dạng chuẩn) cần thiết cần thỏa cho các bảng khác nhau của cơ sở dữ liệu. Theo ý này, thì để nói rằng một quan hệ là thuộc một dạng chuẩn nào đó, thì có nghĩa là bảng đó đã thỏa một số điều kiện nào đó. Vì đây là quá trình có thể đảo ngược, nên tập quan hệ gốc ban đầu có thể được phục hồi lại mà không làm mất thông tin. Khi quá trình chuẩn hóa lên được các dạng cao hơn, thì từng tập quan hệ cũng như dữ liệu của nó sẽ phải thỏa thêm các kiểu phụ thuộc hàm khác.

Các mục đích của quá trình chuẩn hóa là:

- Cho phép nó có khả năng biểu diễn mọi quan hệ trong cơ sở dữ liệu.
- Tận dụng được sức mạnh của các thuật toán truy vấn quan hệ dựa trên các toán tử quan hệ cơ bản.
- Giải phóng quan hệ khỏi các phép chèn, cập nhật và xóa dữ liệu sai.
- Giảm bớt sự cần thiết phải xây dựng lại cấu trúc của các quan hệ khi cần đến các kiểu dữ liệu mới.

Hai mục đích đầu tiên thường chỉ áp dụng cho dạng chuẩn một; còn hai mục đích sau áp dụng cho tất cả các dạng chuẩn. Các thuật ngữ này sẽ định nghĩa thật ngắn gọn.

Toàn bộ quá trình chuẩn hóa dựa trên việc phân tích các quan hệ, lược đồ, khóa chính và các phụ thuộc hàm. Khi một quan hệ không thỏa điều kiện một dạng chuẩn nào đó, thì quan hệ đó phải được phân rã thành các quan hệ nhỏ hơn thỏa điều kiện đang kiểm tra. Ban đầu E. F. Codd đề nghị ba dạng chuẩn với các tên gọi dạng chuẩn một, hai và ba. Các dạng chuẩn này thường được viết và gọi tắt là 1NF, 2NF và 3NF. Ngoài ra còn có các dạng chuẩn khác như Boyce-Codd (BCNF, được E. F. Codd giới thiệu năm 1974), dạng chuẩn bốn (4NF) và dạng chuẩn năm (5NF). Mối quan hệ giữa bốn dạng chuẩn đầu được minh họa trong hình dưới. Hình này đôi khi còn được gọi là "củ hành" dạng chuẩn. Trong tài liệu này ta chỉ đề cập đến các dạng chuẩn: 1NF, 2NF, 3NF và BCNF.



2. Dạng chuẩn một (First Normal Form)

Đôi khi trong quá trình thiết kế cơ sở dữ liệu, có khi ta cần nhập vào các mục của bảng (giao của dòng và cột) nhiều hơn một giá trị. Ví dụ, xét bảng PROJECT, trong đó một hay nhiều nhân viên (employee) có thể làm chung trong một dự án (project). Chú ý là với mỗi mã dự án (Pro-ID) thì mỗi "dòng" của bảng có thể có nhiều hơn một giá trị trong các cột Emp-ID, Emp-Name, Emp-Dpt, Emp-Hrly-Rate và Total-Hrs.

PROJECT

Pro-ID	Pro-Name	Pro-Mgr-ID	Emp-ID	Emp-Name	Emp-Dpt	Emp-Hrly-Rate	Total-Hrs
100	E-commerce	789487453	123423479 980808980 234809000 542298973	Heydary Jones Alexander Johnson	MIS TechSupport TechSupport TechDoc	65 45 35 30	10 6 6 12
110	Distance-Ed	820972445	432329700 689231199 712093093	Mantle Richardson Howard	MIS TechSupport TechDoc	50 35 30	5 12 8
120	Cyber	980212343	834920043 380802233 553208932 123423479	Lopez Harrison Olivier Heydary	Engineering TechSupport TechDoc MIS	80 35 30 65	4 11 12 10
130	Nitts	550227043	340783453	Shaw	Cabling	40	27

Để diễn tả bảng này cũng như sự liên quan giữa bảng và quan hê, ta cần đến một số thuật ngữ mới. Ô nhập của bảng có nhiều hơn một trị được gọi là ô nhập *đa trị* (multivalue). Các bảng có các ô nhập đa trị được gọi là các bảng *chưa chuẩn* (unormalized). Trong một bảng chưa chuẩn, một thuộc tính hay một nhóm các thuộc tính có thể có nhiều ô nhập đa trị chỉ cho một lần xuất hiện của giá trị định danh của bảng, gọi là *nhóm lặp lại* (repeating group). Giá trị định danh của bảng (table identifier) chỉ thuộc tính cho phép ta phân biệt giữa các dòng khác nhau trong bảng chưa chuẩn.

Như vậy, bảng PROJECT dưới dạng một bảng chưa chuẩn có các thuộc tính Emp-ID, Emp-Name, Emp-Dpt, Emp-Hrly và Total-Hrs là các nhóm lặp lại. Kiểu bảng này không thể được xét như một quan hệ vì có các ô nhập đa trị.

Để có thể biểu diễn bảng này thành một quan hệ và đưa nó vào RDBMS, ta cần *chuẩn hóa* (normalize) bảng này. Nói cách khác ta cần đưa bảng này về dạng chuẩn một. Định nghĩa của dạng chuẩn một như sau:

Một quan hệ r(R) được gọi là ở dạng chuẩn một (1NF, First Normal Form) nếu và chỉ nếu mọi ô nhập của quan hệ (ô nhập, entry, là điểm giao của một bộ và một cột) có nhiều nhất là một giá trị. Một số tác giả nói rằng một quan hệ được gọi là ở dạng chuẩn một nếu và chỉ nếu

tất cả các thuộc tính của quan hệ đó đều có miền giá trị đơn. Thực ra hai định nghĩa đó là tương tự nhau. Nếu tất cả các quan hệ của cơ sở dữ liệu đều ở dạng chuẩn một thì ta nói cơ sở dữ liệu ở dạng chuẩn một.
Mục đích của việc chuẩn hóa một bảng là để loại bỏ các nhóm lặp lại và bảo đảm rằng tất cả các ô nhập của bảng kết quả chỉ có thể có nhiều nhất là một giá trị. Bạn nên nhận thức rằng nếu chỉ loại bỏ các nhóm lặp lại thì bảng chưa chuẩn cũng không thể tự động trở thành quan hệ, mà ta cần đến vài thao tác nữa trên bảng kết quả để đảm bảo rằng chúng thực sự là các quan hệ. Nói chung, có hai cách để chuẩn hóa. Hai cách này sẽ được trình bày ngay sau đây.
- Cách thứ nhất còn được gọi là "làm phẳng" (flattening) bảng, sẽ xóa các nhóm lặp lại bằng cách điền vào các ô nhập "bị thiếu" cho các "dòng chưa hoàn chỉnh" của bảng bằng những bản sao của các thuộc tính không lặp lại tương ứng. Ví dụ sau sẽ minh họa cho bạn thấy cách làm này.

Ví dụ
Hãy làm phẳng bảng PROJECT. Bảng kết quả có phải là một quan hệ không? Nếu không thì làm sao để chuyển nó thành dạng chuẩn một?
Trong bảng PROJECT, với mỗi dự án, dưới các thuộc tính Emp-ID, Emp-Name, Emp-Dpt, Emp-Hrly và Total-Hrs có nhiều giá trị cho mỗi ô nhập. Để chuẩn hóa bảng này ta chỉ cần điền vào các ô nhập còn lại bằng cách sao chép các thông tin tương ứng từ các thuộc tính không lặp lại. Ví dụ như với dòng chứa các nhân viên Jones, ta chỉ cần điền vào các ô nhập "trống" còn lại bằng cách sao chép các giá trị của các cột Proj-ID, Proj-Name và Proj-Mgr-ID. Bây giờ dòng này chỉ có một giá trị cho mỗi ô nhập của nó. Trong bảng kết quả mới dưới đây chúng ta đã tô xám tất cả các bộ mới của những nhân viên thuộc dự án E-commerce, màu xanh là các giá trị đã sao chép. Ta sẽ lặp lại quá trình tương tự cho các nhân viên của hai dự án còn lại. Bảng PROJECT đã được chuẩn hóa được trình bày như sau:

PROJECT

Pro-ID	Pro-Name	Pro-Mgr-ID	Emp-ID	Emp-Name	Emp-Dpt	Emp-Hrly-Rate	Total-Hrs
100	E-commerce	789487453	123423479	Heydary	MIS	65	10
100	E-commerce	789487453	980808980	Jones	TechSupport	45	6
100	E-commerce	789487453	234809000	Alexander	TechSupport	35	6
100	E-commerce	789487453	542298973	Johnson	TechDoc	30	12
110	Distance-Ed	820972445	432329700	Mantle	MIS	50	5
110	Distance-Ed	820972445	689231199	Richardson	TechSupport	35	12
110	Distance-Ed	820972445	712093093	Howard	TechDoc	30	8
120	Cyber	980212343	834920043	Lopez	Engineering	80	4
120	Cyber	980212343	380802233	Harrison	TechSupport	35	11
120	Cyber	980212343	553208932	Olivier	TechDoc	30	12
120	Cyber	980212343	123423479	Heydary	MIS	65	10
130	Nitts	550227043	340783453	Shaw	Cabling	40	27

Bảng PROJECT đã được chuẩn hóa này không phải là một quan hệ vì nó không có khóa chính. Thuộc tính Proj-ID không còn là thuộc tính định danh duy nhất cho từng dòng nữa. Chú ý là tất cả các dòng trong vùng xám đều có chung Proj-ID. Để chuyển bảng này thành quan hệ ta cần định nghĩa khóa chính. Khóa chính cho bảng này có lẽ là khóa phức từ sự kết hợp (Proj-ID, Emp-ID). Rõ ràng là mọi sự kết hợp khác đều không thỏa để làm khóa chính.
- Cách thứ hai để chuẩn hóa một bảng là phân rã bảng đó thành hai bảng mới thay thế cho bảng gốc. Việc phân rã một quan hệ sẽ cần đến việc tách các thuộc tính của quan hệ cũ để tạo lược đồ cho hai quan hệ mới. Tuy nhiên, trước khi phân rã bảng gốc ta cần nhận diện (các) thuộc tính sẽ đóng vai trò thuộc tính định danh của bảng. Nếu nhận diện được (các) thuộc tính đó, một bảng sẽ chứa (các) thuộc tính định danh của bảng gốc và tất cả các thuộc tính không lặp lại. Còn bảng kia cũng chứa một bản sao (các) thuộc tính định danh đó và tất cả các thuộc tính lặp lại. Để chuyển các bảng này thành các quan hệ, ta chỉ cần chỉ rõ khóa chính của từng bảng. Nếu một trong hai bảng có nhiều hơn một nhóm lặp lại hay nếu các nhóm lặp lại có chứa các nhóm lặp lại khác bên trong chúng thì quá trình này (quá trình phân rã) có thể được tiến hành nhiều lần. Các bộ của các quan hệ mới sẽ là kết quả của phép chiếu từ quan hệ gốc đến các lược đồ tương ứng. Ví dụ sau sẽ minh họa cho cách thứ hai này dùng chuẩn hóa một bảng.

Ví dụ
Hãy chuẩn hóa bảng PROJECT ban đầu bằng cách dùng cách thứ hai (phân rã).
Để chuẩn hóa bảng PROJECT ta cần thay thế nó bằng hai bảng mới. Bảng thứ nhất sẽ chứa thuộc tính định danh của bảng cùng với các nhóm không lặp lại. Các thuộc tính của bảng này là: Proj-ID (thuộc tính định danh), Proj-Name và Proj-Mgr-ID. Còn bảng thứ hai sẽ chứa thuộc tính định danh cùng với các nhóm lặp lại. Các thuộc tính của bảng này là: Proj-ID, Emp-ID, Emp-Name, Emp-Dpt, Emp-Hrly-Rate và Total-Hrs. Để chuyển bảng sau thành quan hệ, ta cần gán khóa chính cho nó. Có hai quan hệ ở 1NF được trình bày dưới đây. Chú ý là với bảng PROJECT_EMPLOYEE, thuộc tính kết hợp (Proj-ID, Emp-ID) sẽ là khóa chính.

PROJECT

Pro-ID	Pro-Name	Pro-Mgr-ID
100	E-commerce	789487453
110	Distance-Ed	820972445
120	Cyber	980212343
130	Nitts	550227043

PROJECT-EMPLOYEE

Pro-ID	Emp-ID	Emp-Name	Emp-Dpt	Emp-Hrly-Rate	Total-Hrs
100	123423479	Heydary	MIS	65	10
100	980808980	Jones	TechSupport	45	6
100	234809000	Alexander	TechSupport	35	6
100	542298973	Johnson	TechDoc	30	12
110	432329700	Mantle	MIS	50	5
110	689231199	Richardson	TechSupport	35	12
110	712093093	Howard	TechDoc	30	8
120	834920043	Lopez	Engineering	80	4
120	380802233	Harrison	TechSupport	35	11
120	553208932	Olivier	TechDoc	30	12
120	123423479	Heydary	MIS	65	10
130	340783453	Shaw	Cabling	40	27

Đến đây có thể bạn sẽ thắc mắc, rằng cách nào trong hai cách trên là tốt hơn. Thực ra cả hai cách đều tốt vì chúng đều thực hiện được yêu cầu chuyển một bảng chưa chuẩn về quan hệ 1NF. Tuy nhiên có thể nói rằng người ta thích cách thứ hai hơn, vì ngoài khả năng chuyển về dạng 1NF, các bảng kết quả còn đạt yêu cầu ít dư thừa. Hơn nữa, bạn sẽ thấy ở phần tiếp theo, cho dù đã thực hiện theo cách thứ nhất, thì sau đó ta cùng tiếp tục áp dụng cách thứ hai cho bảng kết quả.

3. Dữ liệu xấu trong các quan hệ đạt 1NF

Vấn đề dư thừa trong các quan hệ ở 1NF sẽ dẫn đến nhiều kiểu *dữ liệu xấu* (anomalies data) khác nhau. Dữ liệu xấu ở đây có ý rằng sẽ gặp rắc rối về dữ liệu khi thực hiện một số tác vụ quan hệ. Dữ liệu xấu được chia thành ba loại chính: thêm, xóa và sửa. Tên của các phép quan hệ tương ứng là INSERT, DELETE và UPDATE. Chia làm ba là nói theo lý thuyết, chứ trên thực tế có thể chia gọn hơn: thêm/xóa và sửa. Gộp chung thêm và xóa vì thêm/xóa là hai vế không thể tách rời. Tuy nhiên để đơn giản vấn đề ta cứ chia làm ba cho rõ.

Để bạn rõ hơn về dữ liệu xấu trong các quan hệ ở 1NF, ta xét lại phụ thuộc hàm Emp-ID → Emp-Dpt của quan hệ PROJECT_EMPLOYEE trong ví dụ trước. Rắc rối khi thêm dữ liệu sẽ xảy ra trong quan hệ này vì ta không thể thêm thông tin của một nhân viên mới sẽ làm việc cho một phòng ban cụ thể nào đó trừ khi nhân viên đó đã được phân công một dự án. Nhớ rằng khóa phức của quan hệ này là (Proj-ID, Emp-ID). Cũng chú ý rằng ràng buộc toàn vẹn sẽ không cho phép gán giá trị NULL cho các thuộc tính của khóa. Tương tự, rắc rối khi xóa sẽ xảy ra ngay khi ta xóa chẳng hạn bộ cuối cùng trong bảng. Rắc rối là do khi đó ta không chỉ xóa thông tin dự án để kết nối nhân viên đó với một dự án cụ thể, mà còn làm mất luôn thông tin về phòng ban của nhân viên đó. Ngoài hai kiểu rắc rối trên, quan hệ cũng sẽ gặp khó khăn khi cập nhật, vì phòng ban làm việc của một nhân viên có thể sẽ xuất hiện nhiều lần trong bảng. Chính sự dư thừa thông tin này sẽ dẫn đến rắc rối vì nếu nhân viên chuyển sang phòng ban khác thì ta phải đối mặt với hai vấn đề: hoặc phải tìm toàn bộ bảng để tìm nhân viên đó và cập nhật giá trị Emp-Dpt hoặc để sót không cập nhật vài bộ của nhân viên đó và kết quả là cơ sở dữ liệu không còn nhất quán nữa. Với những bảng nhỏ, ta có thể dễ dàng khắc phục vấn đề này, nhưng khi bảng có cả ngàn bộ thì vấn đề không còn dễ dàng nữa.

4. Phụ thuộc một phần

Cho quan hệ r(R), tập thuộc tính X và Y (X, Y ⊂ R) và phụ thuộc hàm X → Y, ta nói thuộc tính Y là *phụ thuộc đầy đủ vào thuộc tính* (fully dependent on attribute) X nếu và chỉ nếu không tồn tại tập con thật sự W của X để W → Y. Nếu có tập con thật sự W của X mà W → Y thì ta nói thuộc tính Y là *phụ thuộc một phần vào thuộc tính* (partially dependent on attribute) X.

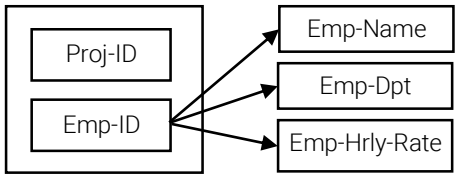
Một cách khác để diễn đạt khái niệm phụ thuộc đầy đủ và phụ thuộc một phần là: Cho phụ thuộc hàm $A_1A_2A_3...A_m \rightarrow B_1B_2 ...B_n$ (m > n), khi đó tập thuộc tính $B_1B_2 ...B_n$ được gọi là phụ thuộc một phần vào các thuộc tính $A_1A_2A_3...A_m$ nếu và chỉ nếu tồn tại một tập con thật sự $A_cA_dA_e...A_k \subset A_1A_2A_3...A_m$ sao cho $A_cA_dA_e...A_k \rightarrow B_1B_2 ...B_n$. Nói cách khác, các thuộc tính $B_1B_2 ...B_n$ là phụ thuộc một phần vào tập thuộc tính xác định nếu có tập con của tập thuộc tính xác định cũng xác định được vế phải của phụ thuộc hàm trên. Còn nếu không có một tập con nào của tập thuộc tính xác định như thế thì ta nói các thuộc tính $B_1B_2 ...B_n$ là phụ thuộc đầy đủ vào tập thuộc tính xác định của phụ thuộc hàm đó.

Ý nghĩa của phụ thuộc một phần là một nội dung rất quan trọng khi muốn chuyển một quan hệ thành 2NF. Ví dụ sau sẽ minh họa cho khái niệm phụ thuộc một phần này.

Ví dụ

Hãy chỉ rõ những phụ thuộc một phần trong quan hệ PROJECT_EMPLOYEE.

Như đã nói, khóa chính của quan hệ này được hình thành bởi các thuộc tính Proj-ID và Emp-ID. Điều đó có nghĩa là hai thuộc tính Proj-ID và Emp-ID có thể xác định mọi thuộc tính của quan hệ. Tuy nhiên có thể thấy rằng chỉ cần thuộc tính Emp-ID là đã xác định được các thuộc tính: Emp-Name, Emp-Dpt, Emp-Hrly-Rate, Total-Hrs. Nói cách khác, các thuộc tính Emp-Name, Emp-Dpt, Emp-Hrly-Rate và Total-Hrs là phụ thuộc một phần vào khóa. Sơ đồ sau sẽ minh họa cho tính phụ thuộc một phần cho những thuộc tính này.

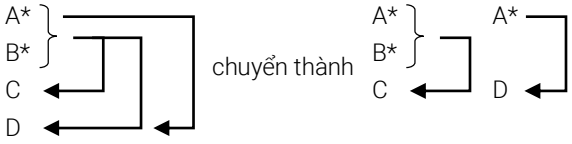


Ví dụ
Hãy tìm các phụ thuộc một phần trong bảng PROJECT (trong kết quả chuẩn hóa 1NF theo cách thứ hai)
Không có phụ thuộc một phần nào trong bảng này vì tập thuộc tính xác định của khóa chỉ có một thuộc tính.

5. Dạng chuẩn hai (Second Normal Form)

Một quan hệ r(R) được gọi là ở dạng chuẩn hai (2NF) nếu và chỉ nếu hai điều kiện sau đồng thời thỏa mãn:
(1) r(R) đã ở 1NF.
(2) Không có thuộc tính không khóa nào phụ thuộc một phần vào khóa, hay nói cách khác, mỗi thuộc tính không khóa trong R đều phụ thuộc đầy đủ vào mọi khóa (kể cả các khóa dự tuyển).

Chú ý là để tìm các thuộc tính không khóa trong R, ta cần chỉ rõ tất cả các thuộc tính khóa của R. Tất cả ở đây bao gồm mọi khóa có thể có. Sau đó các thuộc tính không khóa sẽ được tính bằng R – P trong đó P là tập tất cả các thuộc tính khóa, còn R là lược đồ quan hệ của R. Nếu tất cả các quan hệ của cơ sở dữ liệu đều ở 2NF thì ta nói cơ sở dữ liệu ở 2NF.
Để chuyển một quan hệ về 2NF ta có thể làm theo cách được bày ở hình dưới. Trong sơ đồ này các thuộc tính khóa được đánh dấu bằng các dấu sao, còn các phụ thuộc hàm được biểu diễn bằng các mũi tên. Các khóa phức được móc bằng ngoặc nhọn.



Sơ đồ chuyển thành 2NF

Chú ý là trong hình trên, khóa chính gồm các thuộc tính A và B. Hai thuộc tính này có thể xác định tất cả các thuộc tính còn lại. Nghĩa là các thuộc tính A và B là thuộc tính khóa. Thuộc tính C là phụ thuộc đầy đủ vào khóa. Còn thuộc tính D chỉ phụ thuộc một phần vào khóa, vì ta chỉ cần thuộc tính A cũng đã xác định được nó. Các thuộc tính C và D là các thuộc tính không khóa. Rõ ràng trong sơ đồ này quan hệ cũ đã được thay thế bằng hai quan hệ mới. Quan hệ mới thứ nhất có ba thuộc tính: A, B và C. khóa chính của quan hệ này là AB (cũng là khóa chính của quan hệ cũ). Quan hệ thứ hai chỉ có A và D, A là khóa chính và D bây giờ phụ thuộc đầy đủ vào khóa. Mặc dù sơ đồ này chỉ minh họa cho bốn thuộc tính, tuy nhiên ta có thể tổng quát hóa cho mọi quan hệ cần chuyển về 2NF nếu giả sử rằng C là một tập hợp các thuộc tính phụ thuộc đầy đủ vào khóa và D là một tập hợp các thuộc tính phụ thuộc một phần vào khóa. Ví dụ tiếp theo sẽ minh họa cho cách chuyển một quan hệ về 2NF bằng cách dùng phương pháp trên.

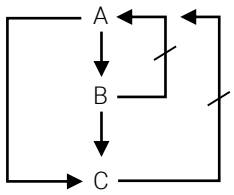
Ví dụ
Hãy chuyển quan hệ PROJECT_EMPLOYEE về quan hệ 2NF.
Phương pháp này sẽ tách quan hệ thành hai quan hệ mới. Quan hệ thứ nhất có khóa chính là khóa chính của quan hệ PROJECT_EMPLOYEE (Proj-ID, Emp-ID), các thuộc tính còn lại của quan hệ này là các thuộc tính phụ thuộc đầy đủ vào khóa. Trong trường hợp này, chỉ có một thuộc tính phụ thuộc đầy đủ vào khóa phức là Total-Hrs. Lược đồ của quan hệ mới này được đặt tên là HOURSE_ASSIGNED như sau:
HOURSE_ASSIGNED (Proj-ID, Emp-ID, Total-Hrs)
Quan hệ thứ hai có khóa chính là Emp-ID vì thuộc tính này hoàn toàn xác định được các thuộc tính Emp-Name, Emp-Dpt và Emp-Hrly-Rate. Lược đồ của quan hệ này như sau:
EMPLOYEE (Emp-ID, Emp-Name, Emp-Dpt, Emp-Hrly-Rate)

6. Dữ liệu xấu trong các quan hệ ở 2NF

Các quan hệ dù đã ở 2NF vẫn có dữ liệu xấu. Để tiện giải thích, ta giả sử rằng phòng ban làm việc của nhân viên có thể xác định được tỷ lệ giờ làm việc của nhân viên đó, nghĩa là Emp-Dpt → Emp-Hrly-Rate. Yếu tố này không được đề cập đến trong dạng chuẩn trước, nhưng không phải là tình huống không thực tế. Dữ liệu xấu khi chèn (insertion anomalies) sẽ xảy ra trong quan hệ EMPLOYEE. Ví dụ như xét tình huống ta muốn nhập tỷ lệ giờ làm việc cho một phòng ban mới, nhưng điều này không thực hiện được cho đến khi một nhân viên mới của phòng ban được nhập vào. Lưu ý là tỷ lệ giờ làm việc của một phòng ban lại độc lập với việc phòng ban đó có nhân viên hay không. Quan hệ EMPLOYEE cũng có thể có dữ liệu xấu khi xóa (deletion anomalies). Dữ liệu xấu sẽ có khi ta ta xóa một nhân viên, mà đó lại là nhân viên duy nhất của một phòng ban nào đó. Trong trường hợp này, chúng ta cũng mất luôn thông tin về tỷ lệ giờ àm việc của phòng ban đó. Cập nhật cũng có thể có dữ liệu xấu (update anomalies) trong quan hệ EMPLOYEE vì có thể có vài nhân viên thuộc cùng một phòng ban nhưng làm ở các dự án khác nhau. Nếu tỷ lệ giờ làm viêc của phòng ban đó thay đổi, chúng ta cần bảo đảm rằng giá trị tương ứng của mọi nhân viên thuộc phòng ban đó cũng thay đổi theo. Nếu không cơ sở dữ liệu sẽ ở tình trạng không nhất quán.

7. Phụ thuộc bắc cầu

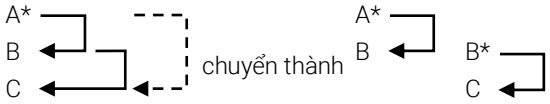
Giả sử A, B và C là các tập thuộc tính của quan hệ r(R). Giả sử thêm rằng các phụ thuộc hàm sau đều thỏa: A → B, ¬(B → A), B → C, ¬(C → A) và A → C. Rõ ràng C → B là không thể xảy ra và cũng không cần. Ký hiệu ¬(B → A), nghĩa là không có phụ thuộc hàm (B → A).
Nếu tất cả các điều kiện đều đúng, thì ta nói thuộc tính C là *phụ thuộc bắc cầu vào thuộc tính* (transitively dependent on attribute) A. Cần nói rõ rằng các phụ thuộc hàm này xác định các điều kiện cho thuộc tính C phụ thuộc bắc cầu vào thuộc tính A. Nếu một trong các điều kiện trên không thỏa mãn thì thuộc tính C không phụ thuộc bắc cầu vào thuộc tính A.
Sơ đồ trong hình dưới chỉ rõ các điều kiện này. Trong sơ đồ này các mũi tên có dạng "→" dùng biểu diễn cho các phụ thuộc hàm. Chú ý là phụ thuộc hàm A → C có thể không được chỉ ra tường minh nhưng nó đương nhiên đúng theo tiên đề Transitivity. Các điều kiện ¬(B → A) và ¬(C → A) cũng cần thiết để bảo đảm rằng các thuộc tính A và B là các thuộc tính không khóa.



Các điều kiện để định nghĩa sự phụ thuộc bắc cầu
của thuộc tính C vào thuộc tính A.

8. Dạng chuẩn ba (Third Normal Form)

Một quan hệ r(R) là ở dạng chuẩn ba (3NF) nếu và chỉ nếu nó đồng thời thỏa các điều kiện sau:
(1) r(R) đã có ở 2NF.
(2) Không có thuộc tính không khóa nào phụ thuộc bắc cầu vào khóa.
Bạn cần phân biệt rõ điều kiện đã được trình bày ở hình trên với điều kiện thứ hai của định nghĩa 3NF. Lưu ý là hình trên chỉ rõ các điều kiện cần thiết để thuộc tính không khóa C có thể phụ thuộc bắc cầu vào khóa A. Định nghĩa 3NF yêu cầu rằng các điều kiện này sẽ không thỏa nếu A là thuộc tính khóa và C là thuộc tính không khóa.
Một cách khác để biểu diễn các điều kiện cho dạng chuẩn ba là:
(1) r(R) đã ở 2NF.
(2) Không có thuộc tính không khóa nào xác định thuộc tính không khóa khác.
Vì hai tập điều kiện này là tương đương, nên ta dùng cái nào là tùy.
Như hai định nghĩa của 3NF đã nói, mục đích của việc chuyển lên 3NF là loại bỏ tất cả các phụ thuộc bắc cầu. Để chuyển một quan hệ từ 2NF lên 3NF ta sẽ làm theo phương pháp trong hình dưới. Trong hình này, giả sử mọi phụ thuộc hàm không được chỉ ra có nghĩa là không có. Dấu * để biểu diễn thuộc tính khóa còn dấu mũi tên để biểu diễn phụ thuộc hàm. Mũi tên không liên tục có ý rằng một phụ thuộc hàm có thể không được cho sẵn tường minh, nhưng nó lcó thể suy dẫn bằng cách dùng các tiên đề suy luận.



Sơ đồ chuyển thành 3NF

Ví dụ sau sẽ minh họa cho cách chuyển một quan hệ lên 3NF bằng cách dùng phương pháp trên.

Ví dụ
Hãy chuyển quan hệ EMPLOYEE của ví dụ trước từ 2NF lên 3NF bằng cách dùng định nghĩa thứ nhất của 3NF.
Quan hệ EMPLOYEE không ở 3NF vì có phụ thuộc bắc cầu của một thuộc tính không khóa vào khóa chính của quan hệ. Trong trường hợp này, thuộc tính không khóa Emp-Hrly-Rate là phụ thuộc bắc cầu vào khóa dựa trên phụ thuộc hàm Emp-Dpt → Emp-Hrly-Rate. Lưu ý là tất cả các điều kiện cần thiết khác cho định nghĩa đều đã được thỏa mãn trong tập các phụ thuộc hàm này. Cụ thể, ta có ¬(Emp-Dpt → Emp-ID) và ¬(Emp-Hrly-Rate → Emp-ID). Để chuyển quan hệ này về một quan hệ ở chuẩn 3NF, ta cần phải loại bỏ mọi phụ thuộc bắc cầu của một thuộc tính không khóa vào khóa. Theo sơ đồ hình trên, ta cần tạo hai quan hệ mới. Lược đồ của quan hệ thứ nhất là:
EMPLOYEE (Emp_ID, Emp-Name, Emp-Dpt)
Lược đồ của quan hệ thứ hai là:
CHARGES (Emp-Dpt, Emp-Hrly-Rate)
Rõ ràng là trong quan hệ thứ hai, thuộc tính Emp-Dpt đã được chọn làm khóa chính như theo yêu cầu của sơ đồ.

Ví dụ
Dùng định nghĩa thứ hai của 3NF, ta có thể chỉ ra quan hệ EMPLOYEE đã ở 3NF hay chưa? Nếu dùng định nghĩa thứ hai của 3NF, thì ta có cần dùng một thủ tục khác để chuyển quan hệ lên 3NF không?
Ta có thể nói rằng quan hệ chưa ở 3NF bằng cách chú ý rằng Emp-Dpt → Emp-Hrly-Rate và cả hai đều là thuộc tính không khóa. Để chuyển quan hệ này lên 3NF ta dùng phương pháp tương tự trên.
Các quan hệ mới mà ta có được thông qua quá trình chuẩn hóa này sẽ không gặp phải những tình trạng xấu về dữ liệu như những dạng chuẩn trước. Nghĩa là ta có thể thêm, xóa và sửa các bộ mà không bị ảnh hưởng như ở 1NF và 2NF.

9. Dữ liệu xấu trong các quan hệ ở 3NF

Dạng chuẩn ba giúp ta tránh được tình trạng dữ liệu xấu gây nên bởi các phụ thuộc bắc cầu trên khóa chính hoặc các phụ thuộc của một thuộc tính không khóa trên thuộc tính không khóa khác. Tuy nhiên các quan hệ ở 3NF vẫn còn có thể gây dữ liệu xấu, đặc biệt khi các quan hệ có hai tập khóa dự tuyển gối (overlapping) nhau, hay khi có một thuộc tính không khóa xác định một thuộc tính khóa. Hai ví dụ sau sẽ minh họa cho điều này.

Ví dụ
Xét CERTIFICATION_PROGRAM (Area, Course, Section, Time, Location)

CERTIFICATION_PROGRAM

Area	Course	Section	Time	Location
East Coast	SQL 101	Introduction	8:00-10:00	Atlanta Educational Center

East Coast	SQL 101	Intermediate	10:00-12:00	New York Educational Center
West Coast	SQL 101	Advanced	8:00-10:00	Los Angeles Educational Center

Quan hệ này là ở 3NF vì không có chuyện các thuộc tính không khóa xác định các thuộc tính khác. Tuy nhiên nếu chỉ có một trung tâm (location) cho mỗi thành phố thì ta sẽ có Location → Area. Phụ thuộc này không vi phạm điều kiện 3NF nhưng sẽ gây dữ liệu xấu. Ví dụ như giả sử nếu ta xóa bộ cuối của quan hệ thì ta sẽ mất thông tin về địa điểm của trung tâm giáo dục.

Ví dụ
Xét quan hệ MANUFACTURER dưới đây, trong đó mỗi nhà máy (manufacurer) có một mã (ID) và tên (Name) duy nhất. Các nhà máy sẽ sản xuất các sản phẩm (được xác định bởi mã duy nhất của sản phẩm, Item-No) với số lượng (quantity được chỉ rõ. Các nhà máy có thể sản xuất nhiều sản phẩm và các nhà máy khác nhau có thể sản xuất các sản phẩm giống nhau.

MANUFACTURER

ID	Name	Item-No	Quantity
M101	Electronics USA	H3552	1000
M101	Electronics USA	J08732	500
M101	Electronics USA	Y23490	200
M322	Electronics-R-Us	H3552	900

Quan hệ MANUFACTURER này có hai khóa dự tuyển: (ID, Item-No) và (Name, Item-No) gối nhau ở thuộc tính Item-No. Quan hệ đang ở 3NF vì chỉ có một thuộc tính không khóa và không thể nói rằng thuộc tính này có thể xác định một thuộc tính không khóa nào khác.
Quan hệ MANUFACTUERE có thể gây dữ liệu xấu khi cập nhật. Chẳng hạn như khi có một nhà máy đổi tên. Nếu giá trị của thuộc tính này không thay đổi trong mọi bộ tương ứng của nó thì cơ sở dữ liệu sẽ ở tình trạng thiếu nhất quán.

10. Dạng chuẩn Boyce-Codd (Boyce-Codd Normal Form)

Để khắc phục tình trạng dữ liệu xấu ở 3NF, ta sẽ xét đến quá trình chuẩn hóa lên dạng chuẩn cao hơn, là dạng chuẩn Boyce-Codd (BCNF). Khái niệm về quá trình này như sau.
Một quan hệ r(R) là ở dạng chuẩn Boyce-Codd nếu và chỉ nếu các điều kiện sau đồng thời thỏa mãn:
(1) Quan hệ đã ở 1NF.
(2) Với mọi phụ thuộc hàm có dạng X → A, ta đều có A ⊂ X hoặc X là một siêu khóa (superkey) của r. Nói cách khác, mọi phụ thuộc hàm hoặc là phụ thuộc tầm thường hoặc X phải là siêu khóa.

Chú ý là định nghĩa của BCNF không nói gì đến phụ thuộc đầy đủ hay một phần. Tuy nhiên từ định nghĩa này ta có thể ghi nhận những điều sau về các thuộc tính không khóa và các thuộc tính khóa của lược đồ quan hệ:
- Tất cả các thuộc tính không khóa phải phụ thuộc đầy đủ vào mọi khóa.
- Tất cả các thuộc tính khóa đều phải phụ thuộc đầy đủ vào mọi khóa không chứa chúng.

Tập hợp các quan hệ ở BNCF là một tập con thực sự của các quan hệ ở 3NF. Nghĩa là mọi BCNF đều là 3NF nhưng không phải tất cả các 3NF đều là BCNF. Điều đó có nghĩa là định nghĩa BCNF chặt chẽ hơn. Ví dụ sau sẽ minh họa cho điều này.

Ví dụ
Dùng quan hệ MANUFACTURER ở ví dụ trước, hãy chuyển nó lên BCNF.
Để chuyển quan hệ này lên BCNF ta có thể phân rã nó thành các lược đồ quan hệ sau:
Set No. 1
MANUFACTURER (ID, Name)
MANUFACTURER-PART (ID, Item-No, Quantity)
hoặc
Set No. 2
MANUFACTURER (ID, Name)
MANUFACTURER-PART (Name, Item-No, Quantity)
Chú ý là tất cả hai quan hệ đều ở BCNF và điều đó có nghĩa là dữ liệu xấu khi cập nhật sẽ không còn nữa. Trong ví dụ này ta đã phân rã quan hệ theo cách thuận tiện nhất để giải thích.

11. Phân rã bảo toàn (lossless decomposition) và không bảo toàn (lossy decomposition)

Khi một quan hệ được phân rã, ta cần bảo đảm rằng dữ liệu trong quan hệ cũ phải được thể hiện đầy đủ ở các quan hệ mới được phân rã ra; nghĩa là, ta cần bảo đảm rằng có thể phục hồi quan hệ gốc từ các quan hệ mới thay thế nó. Nói chung quan hệ gốc có thể được phục hồi bằng cách dùng phép kết tự nhiên (natural join) trên các quan hệ mới. Nếu có thể phục hồi quan hệ gốc thì ta nói phép phân rã là bảo toàn D hay quan hệ gốc đã được phân rã kết bảo toàn trên D (lossless-join decomposition with respect to D), trong đó D là tập phụ thuộc hàm của quan hệ cũ. Nếu quan hệ không thể phục hồi thì ta nói phép phân rã là không bảo toàn.
Khái niệm này có thể được phát biểu như sau:
Giả sử có quan hệ r(R) được thay thế bởi tập các quan hệ $r_1(R_1)$, $r_2(R_2)$, ... , $r_n(R_n)$ trong đó $R = R_1 \cup R_2 \cup ... \cup R_n$ và D là tập các phụ thuộc thỏa trên r. Ta nói phép phân rã là bảo toàn trên D hay là phép phân rã kết bảo toàn trên D nếu và chỉ nếu $r = \pi_{R1}(r)$ join $\pi_{R2}(r)$ join ... join $\pi_{Rn}(r)$. Nghĩa là quan hệ r là bảo toàn nếu nó là phép kết tự nhiên của các phép chiếu của nó lên các R_i của nó. Nếu quan hệ không thể phục hồi từ các phép chiếu tự nhiên thì phép phân rã được gọi là không bảo toàn trên D. Ví dụ sau sẽ minh họa khái niệm này.

Ví dụ
Xét quan hệ r dưới đây cùng với phân rã R_1 và R_2 của nó. Giả sử có X → Y và Z → Y. Vậy phép phân rã này là bảo toàn hay không bảo toàn?

r

X	Y	Z
x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_2	z_3
x_4	y_3	z_4

Phân rã thành

R_1

X	Y
x_1	y_1
x_2	y_2
x_3	y_2
x_4	y_3

R_2

Y	Z
y_1	z_1
y_2	z_2
y_2	z_3
y_3	z_4

Kết thành

R_1 Join R_2

X	Y	Z
x_1	y_1	z_1
x_2	y_2	z_2
x_2	y_2	z_3
x_3	y_2	z_2
x_3	y_2	z_3
x_4	y_3	z_4

Chú ý là kết tự nhiên của các quan hệ R_1 và R_2 chứa các bộ không có trong các quan hệ gốc. Các bộ mới không có trong quan hệ gốc này được gọi là các *bộ phát sinh* (spurious tuples), vì thay thế đó mới sinh ra (và không hợp lệ). Ta tô xám các bộ phát sinh để dễ xem. Vì phép kết tự nhiên giữa quan hệ R_1 và R_2 không phục hồi được quan hệ gốc nên phép phân rã này không bảo toàn.
Phân rã bảo toàn kết là cần thiết để bảo đảm rằng quan hệ có thể phục hồi được. Việc xác định được phân rã có bảo toàn tập phụ thuộc hàm hay không là một quá trình không khó nếu ta dùng thuật toán kết bảo toàn. Thuật toán kết bảo toàn như sau.

Thuật toán kết bảo toàn (Lossless-Join Algorithm)

Thuật toán có hai đầu vào:
- Tập các quan hệ $r_1(R_1)$, $r_2(R_2)$, ... , $r_n(R_k)$ để thay thế cho quan hệ $r(A_1, A_2, A_3, ... , A_n)$, ở đây $R = \{A_1, A_2, A_3, ... , A_n\} = R_1 \cup R_2 \cup ... \cup R_k$.
- Tập phụ thuộc hàm F thỏa trên r.

Đầu ra của thuật toán cho biết phân rã là bảo toàn hay không bảo toàn.
Ta làm như sau:

(1) Xây dựng một bảng có n cột (n là số thuộc tính của quan hệ gốc) và k dòng (k là số quan hệ được phân rã ra từ quan hệ gốc). Đặt tên cho các cột của bảng là $A_1, A_2, A_3, ... , A_n$ còn các dòng là $R_1, R_2, R_3, ... , R_k$.

(2) Điền vào bảng theo cách sau:
Với mỗi thuộc tính A_i, kiểm tra xem thuộc tính này có phải là một trong các thuộc tính của lược đồ quan hệ R_j hay không. Nếu A_i là thuộc tính của lược đồ R_j thì ta điền giá trị a_i vào ô nhập (A_i, R_j) của bảng, còn không ta sẽ điền giá trị b_{ij} vào đó.

(3) Với mỗi phụ thuộc hàm $X \rightarrow Y$ của F ta làm như sau cho đến khi không thể làm tiếp được nữa. Khi đó chuyển qua bước 4.
Tìm hai hay nhiều dòng có cùng giá trị ở (các) thuộc tính tạo nên X (tập thuộc tính xác định của phụ thuộc hàm đang xét). Có hai trường hợp có thể xảy ra:
(3-a) Nếu có hai hay nhiều dòng có cùng giá trị ở (các) thuộc tính tạo nên X thì tạo các giá trị ở cột Y (vế phải của phụ thuộc hàm đang xét) giống nhau ở các dòng đó. Khi tạo giá trị bằng nhau, nếu có một trong các giá trị ở cột Y là a_j thì sử dụng a_i, còn nếu là b_{ij} và b_{kl} thì dùng giá trị nào trong hai giá trị đó cũng được. Sau đó tiếp tục bước 3.
(3-b) Nếu không có hai dòng có chung giá trị ở (các) thuộc tính tạo nên X thì thực hiện tiếp bước 3.

(4) Kiểm tra tất cả các dòng của bảng. Nếu có một dòng chỉ chứa các a_i, tức $a_1a_2a_3...a_n$ thì phép phân rã là bảo toàn. Còn không phép phân rã là không bảo toàn.

Ví dụ
Xét quan hệ r(X, X, Z, W, Q), tập $F = \{X \rightarrow Z, Y \rightarrow Z, Z \rightarrow W, WQ \rightarrow Z, ZQ \rightarrow X\}$ và phép phân rã của r thành các quan hệ $R_1(X, W)$, $R_2(X, Y)$, $R_3(Y, Q)$, $R_4(Z, W, Q)$ và $R_5(X, Q)$. Hãy dùng thuật toán kết bảo toàn cho biết phép phân rã là bảo toàn hay không.

(1-2) Vì quan hệ gốc có 5 thuộc tính và được phân rã thành 5 quan hệ nên ta cần có bảng chứa 5 cột (thuộc tính) và 5 dòng (quan hệ). Các cột của bảng được đặt tên là X, Y, Z, W và Q, còn các dòng được đặt tên là R_1, R_2, R_3, R_4 và R_5. Để tiện theo dõi ta đổi tên các thuộc tính thành A_1, A_2, A_3, A_4 và A_5 như dưới đây.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1					
R_2					
R_3					
R_4					
R_5					

Vì X là một trong các thuộc tính của quan hệ R_1 nên ta điền a_1 vào ô (R_1, A_1), tức là ô (R_1, X). Nói cách khác, ở cột A_1 (tức cột X) và dòng R_1 ta ghi a_1. Tương tự, ta ghi a_4 ở ô nhập (R_1, A_4) tức là ô (R_1, W) vì W cũng là một thuộc tính của quan hệ R_1. Nghĩa là ở cột A_4 (hay W) và dòng R_1 ta ghi a_4. Trong các ô nhập còn lại của dòng R_1 ta ghi các giá trại b_{ij} trong đó i là số cột còn j là số dòng. Các ô nhập này (từ trái sang phải) lần lượt là b_{12}, b_{13}, và b_{15} như bảng sau:

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2					

R_3					
R_4					
R_5					

Tiếp tục quá trình này ta sẽ điền cho các ô nhập còn lại trong bảng. Sau đó bảng sẽ như sau.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{23}	b_{24}	b_{25}
R_3	b_{31}	a_2	b_{33}	b_{34}	a_5
R_4	b_{41}	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	b_{53}	b_{54}	a_5

(3-a) (lần 1) Xét $X \rightarrow Z$ hay $A_1 \rightarrow A_3$ ta sẽ tìm những bộ có cùng giá trị ở cột X (hay A_1). Trong trường hợp này, các dòng R_1, R_2 và R_5 có cùng giá trị a_1. Do đó ta sẽ làm cho các giá trị ở cột Z (hay A_3) giống nhau. Nhớ rằng Z (hay A_3) là vế phải của phụ thuộc hàm đang xét. Lưu ý rằng các ô nhập tương ứng cho các dòng R_1, R_2 và R_5 trong cột Z là b_{13}, b_{23} và b_{53}. Để các giá trị này giống nhau, ta sẽ chọn tùy ý một trong ba giá trị đó và làm cho hai giá trị kia bằng nó. Giả sử chọn b_{13} thì ta sẽ có bảng sẽ như sau. Ta đã tô xám những ô nhập được thay thế bằng giá trị b_{13} của dòng R_1. Lặp lại bước 3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{13}	b_{24}	b_{25}
R_3	b_{31}	a_2	b_{33}	b_{34}	a_5
R_4	b_{41}	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	b_{13}	b_{54}	a_5

(3-a) (lần 2) Xét $Y \rightarrow Z$ hay $A_2 \rightarrow A_3$ ta sẽ tìm những bộ có cùng giá trị ở cột Y (hay A_2). Trong trường hợp này, các dòng R_2 và R_3 có cùng giá trị a_2. Do đó ta sẽ làm cho các giá trị ở cột Z (hay A_3) giống nhau. Nếu chọn b_{13} thì ta sẽ đổi b_{23} thành b_{13}. Sau đó ta sẽ có bảng như sau. Ta đã tô xám những ô nhập được thay thế. Lặp lại bước 3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{13}	b_{24}	b_{25}
R_3	b_{31}	a_2	b_{13}	b_{34}	a_5
R_4	b_{41}	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	b_{13}	b_{54}	a_5

(3-a) (lần 3) Xét $Z \rightarrow W$ hay $A_3 \rightarrow A_4$ ta sẽ tìm những bộ có cùng giá trị ở cột Z (hay A_3) Trong trường hợp này, các dòng R_1, R_2, R_3 và R_5 có cùng giá trị b_{13}. Do đó ta sẽ làm cho các giá trị ở cột W (hay A_4) giống nhau. Vì một trong các giá trị ở cột W (hay A_4) là a_4 (cho R_1), nên tất cả các giá trị ở các b_{ij} sẽ được chuyển thành a_4. Bảng sẽ như sau. Ta cũng đã tô xám những ô nhập được thay thế. Lặp lại bước 3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{13}	a_4	b_{25}
R_3	b_{31}	a_2	b_{13}	a_4	a_5
R_4	b_{41}	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	b_{13}	a_4	a_5

(3-a) (lần 4) Xét $WQ \rightarrow Z$ hay $A_4A_5 \rightarrow A_3$ ta sẽ tìm những bộ có cùng giá trị ở cột A_4 và A_5. Các dòng R_3, R_4 và R_5 có cùng giá trị a_4 và a_5 ở các cột WQ (hay A_4 và A_5) giống nhau. Do đó ta sẽ làm cho các giá trị ở cột Z (hay A_3) giống nhau. Vì một trong các giá trị ở cột Z (hay A_3) là a_3 (cho R_4), nên tất cả các giá trị ở các b_{ij} sẽ được chuyển thành a_3. Bảng sẽ như sau. Ta cũng đã tô xám những ô nhập được thay thế. Lặp lại bước 3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{13}	a_4	b_{25}

R_3	b_{31}	a_2	a_3	a_4	a_5
R_4	b_{41}	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	a_3	a_4	a_5

(3-a) (lần 5) Xét $ZQ \rightarrow X$ hay $A_3A_5 \rightarrow A_1$ ta sẽ tìm những bộ có cùng giá trị ở cột ZQ (hay A_3A_5). Các dòng R_3, R_4 và R_5 có cùng giá trị a_3 và a_5 ở các cột A_3 và A_5, do đó ta sẽ làm cho các giá trị ở cột X (hay A_1) giống nhau. Vì một trong các giá trị ở cột X (hay A_1) là a_1 (cho R_5), nên tất cả các giá trị ở các b_{ij} sẽ được chuyển thành a_1. Bảng sẽ như sau. Ta cũng đã tô xám những ô nhập được thay thế. Lặp lại bước 3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{13}	a_4	b_{25}
R_3	a_1	a_2	a_3	a_4	a_5
R_4	a_1	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	a_3	a_4	a_5

(3-a) (lần 6) Không còn phụ thuộc hàm nào để xét do đó bảng sẽ không thể thay đổi nữa. Ta chuyển qua bước 4.

(4) (lần 1) Bây giờ ta sẽ tìm những dòng có tất cả a_i. Vì có dòng R_3 thành ($a_1a_2a_3a_4a_5$) nên phép phân rã là bảo toàn.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	b_{12}	b_{13}	a_4	b_{15}
R_2	a_1	a_2	b_{13}	a_4	b_{25}
R_3	a_1	a_2	a_3	a_4	a_5
R_4	a_1	b_{42}	a_3	a_4	a_5
R_5	a_1	b_{52}	a_3	a_4	a_5

Ví dụ

Xét quan hệ r(X, Y, Z) và các phân rã của nó R_1(X, Y) và R_2(Y, Z). Giả sử có $X \rightarrow Y$ và $Z \rightarrow Y$. Hãy dùng thuật toán kết bảo toàn để cho biết phép phân rã này có bảo toàn hay không.

(1-2) Vì quan hệ gốc có 3 thuộc tính và được phân rã thành 2 quan hệ nên ta cần có bảng chứa 3 cột và 2 dòng. Ta cũng đổi tên các thuộc tính thành A_1, A_2 và A_3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$
R_1			
R_2			

Lược đồ quan hệ R_1 có các thuộc tính X và Y nên ở các cột X (hay A_1) và Y (hay A_2) ta viết a_1 và a_2.
Lược đồ quan hệ R_2 có các thuộc tính Y và Z nên ở các cột Y (hay A_2) và Z (hay A_3) ta viết a_2 và a_3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$
R_1	a_1	a_2	b_{13}
R_2	b_{21}	a_2	a_3

(3-b) (lần 1) Xét $X \rightarrow Y$ ta sẽ tìm các dòng có chung giá trị ở thuộc tính X. Vì không có cặp dòng nào như thế nên bảng không thay đổi và ta tiếp tục bước 3.

(3-b) (lần 2) Xét $Z \rightarrow Y$ ta sẽ tìm các dòng có chung giá trị ở thuộc tính Z. Vì không có cặp dòng nào như thế nên bảng không thay đổi.

(3-b) (lần 3) Vì không còn phụ thuộc hàm nào trong F nên bảng đương nhiên sẽ giữ nguyên. Ta chuyển qua bước 4.

(4) (lần 1) Vì không có dòng nào trong bảng chứa tất cả a_i nên phép phân rã là không bảo toàn. Nghĩa là bảng gốc không thể được phục hồi từ phép kết tự nhiên của các quan hệ R_1 và R_2.

12. Bảo toàn phụ thuộc hàm

Như đã nói ở phần trước, để một quan hệ r có thể phục hồi từ các phần chiếu của nó thì phép phân rã phải bảo toàn. Hơn thế nữa, phép phân rã còn phải thỏa một đặc tính khác được gọi là *bảo toàn phụ thuộc* (dependency preservation). Những gì đặc tính này yêu cầu là phép phân rã phải thỏa tất cả các phụ thuộc hàm vốn thỏa trong quan hệ gốc. Lý do là tập phụ thuộc hàm được một quan hệ thỏa là để định nghĩa các ràng buộc toàn vẹn mà quan hệ phải thỏa. Bất kỳ phân rã không bảo toàn phụ thuộc của quan hệ gốc sẽ gây lỗi trên RDBMS. Thực tế, mọi cập nhật đến bất cứ quan hệ nào của phân rã đều cần đến kết của tất cả các quan hệ để kiểm tra xem các ràng buộc có bị vi phạm không. Rõ ràng đây là tác vụ mất thời gian và làm giảm hiệu quả của hệ thống. Ví dụ sau sẽ minh họa cho điều này.

Ví dụ

Xét quan hệ r(X, Y, Z) thỏa các phụ thuộc XY → Z và Z → X. Phân rã r(X, Y, Z) thành các quan hệ R_1(XY) và R_2(XZ) là bảo toàn nhưng không bảo toàn phụ thuộc. Xét các thể hiện sau của các quan hệ này.

R_1

Y	Z
y_1	z_1
y_2	z_2

R_2

Z	X
z_1	x_1
z_2	x_1

Chú ý là quan hệ R_2 thỏa Z → X nhưng khi kết hai quan hệ này thì không thỏa phụ thuộc hàm XY → Z.

R_1 Join R_2

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_2

12.1. Chiếu của một tập phụ thuộc hàm lên một tập thuộc tính

Để phát biểu khái niệm của bảo toàn phụ thuộc và để định nghĩa thuật toán cho phép kiểm tra bảo toàn phụ thuộc, ta cần thêm một số thuật ngữ mới. Giả sử có quan hệ r(R) đã được phân rã thành nhiều quan hệ $\rho = (R_1, R_2, \dots, R_k)$ và F là tập phụ thuộc hàm thỏa trên r. Ta định nghĩa phép chiếu của F lên một tập thuộc tính Z, ký hiệu là $\pi_{(Z)}F$, như sau:

$\pi_{(Z)}F = \{X \to Y \in F^+ / XY \subset Z\}$ trong đó dấu "/" được đọc là "sao cho".

Rõ ràng là định nghĩa này không yêu cầu X → Y phải thuộc về F nhưng phải thuộc về F^+ (F^+ là bao đóng của F). Chú ý thêm là cả X và Y đều có thể là tập thuộc tính mà hợp của chúng phải là tập thuộc tính con của Z. Các phụ thuộc hàm của $\pi_{(Z)}F$ được gọi là thỏa bởi các thuộc tính của Z.

Để tính toán phép chiếu $\pi_{(Z)}F$ ta xét các tập thuộc tính con thực sự X của Z ($X \subset Z$ và $X \neq Z$) đóng vai trò tập thuộc tính xác định của một phụ thuộc hàm hoặc là một phần của tập thuộc tính xác định của một phụ thuộc hàm trong F và làm như sau:

(1) Tính X^+.

(2) Với mỗi tập thuộc tính Y của X^+ đồng thời thỏa các điều kiện sau:

a. $Y \subset Z$

b. $Y \subset X^+$ (trên F)

c. $Y \not\subset X$ (điều kiện này để loại trừ các phụ thuộc hàm tầm thường. Các phụ thuộc hàm tầm thường luôn đúng nhờ tiên đề Transitivity)

X → Y là một trong các phụ thuộc hàm của $\pi_{(Z)}F$. Tập các phụ thuộc hàm trong $\pi_{(Z)}F$ được gọi là thỏa bởi các thuộc tính của Z.

Ví dụ

Cho quan hệ r(X, Y, W, Z, Q) và tập F = {X → Z, Y → Q, ZQ → W}, hãy tìm chiếu của F lên tập các thuộc tính {X, Y, W}.

Chỉ có hai thuộc tính trong {X, Y, W} xuất hiện như thuộc tính xác định của F, là X và Y. Do đó các tập con có thể có của {X, Y, W} là {X}, {Y} và {X, Y}.

- Xét thuộc tính X

(1) (lần 1) Tính X^+.
Dùng thuật toán Bao đóng, ta có $X^+ = \{X, Z\}$

(2) (lần 2) Xét thuộc tính Z của X^+, rõ ràng $Z \not\subset \{X, Y, W\}$.
Vì Z không thỏa điều kiện (a) của bước 2 nên sẽ không có phụ thuộc hàm nào trong $\pi_{(Z)}F$.

- Xét thuộc tính Y

(1) (lần 1) Tính Y^+.
Dùng thuật toán Bao đóng, ta có $Y^+ = \{Y, Q\}$.

(2) (lần 2) Xét thuộc tính Q của Y^+, rõ ràng $Q \not\subset \{X, Y, W\}$.
Vì Q không thỏa điều kiện (a) của bước 2 nên sẽ không có phụ thuộc hàm nào trong $\pi_{(Z)}F$.

- Xét thuộc tính XY

(1) (lần 1) Tính $\{XY\}^+$.
Dùng thuật toán Bao đóng, ta có $\{XY\}^+ = \{X, Y, Z, Q, W\}$.

(2) (lần 2) Xét các thuộc tính của $\{XY\}^+ = \{X, Y, Z, Q, W\}$, theo định nghĩa bao đóng và không tính các phụ thuộc hàm tầm thường ta có XY → W, XY → Q và XY → Z. Trong các phụ thuộc hàm đó, chỉ có XY → W thỏa các điều kiện của bước 2a đến 2b, do đó phụ thuộc hàm này có thể được đưa vào $\pi_{(Z)}F$. Phụ thuộc hàm này của $\pi_{(Z)}F$ là phụ thuộc hàm duy nhất được thỏa bởi các thuộc tính {X, Y, W}.

12.2. Kiểm tra bảo toàn phụ thuộc

Cho quan hệ r(R), quan hệ đó được phân rã thành $\rho = \{R_1, R_2, \dots, R_k\}$ và F là tập phụ thuộc hàm trên r(R), ta nói phép phân rã ρ là bảo toàn phụ thuộc nếu và chỉ nếu các điều kiện sau đồng thời thỏa mãn:

(1) $G = \cup_{i=1}^{k} \pi_{Ri}(F)$

(2) $G^+ = F^+$

Nói cách khác, phân rã ρ chỉ bảo toàn tập phụ thuộc hàm F nếu hợp của tất cả các phụ thuộc hàm trong $G = \cup_{i=1}^{k} \pi_{Ri}(F)$ có thể xác định tất cả các phụ thuộc hàm trong F.

Định nghĩa này cũng cung cấp một thủ tục để kiểm tra xem một phân rã của một quan hệ cho trước có bảo toàn các phụ thuộc hàm của quan hệ đó không. Theo đó, ta chỉ cần chiếu F^+ lên tất cả các R_i của nó, rồi hợp các kết quả này lại với nhau và kiểm tra xem kết quả hợp đó có tương đương với F hay không. Tuy nhiên việc tính F^+ như ta đã biết, là mất rất nhiều thời gian. May mắn là có thuật toán ngắn hơn không cần đến việc tính F^+. Nhưng trước khi giới thiệu thuật toán mới này, ta cần định nghĩa tác vụ sau lên một tập các thuộc tính của phụ thuộc hàm.

Một tác vụ R trên tập thuộc tính Z dựa trên tập phụ thuộc hàm F sẽ thay thế thuộc tính Z bằng tập $Z \cup ((Z \cap R)^+ \cap R)$. Mục đích của tác vụ này là đưa vào tập Z tất cả các thuộc tính A sao cho $(Z \cap R) \rightarrow A \in \pi_R(F)$.

Để xác minh xem các tập F và $G = \cup_{i=1}^{k} \pi_{Ri}(F)$ là tương đương mà không cần tính F^+, thuật toán sẽ xét từng $X \rightarrow Y$ trong F và xem thử Y có trong X^+ không bằng cách xét tác dụng của việc tính bao đóng khi chiếu F lên các R_i. Thuật toán như sau.

Thuật toán kiểm tra bảo toàn (Test Preservation Algorithm)

Đầu vào của thuật toán này là phân rã $\rho = (R_1, R_2, \dots, R_k)$ và tập phụ thuộc hàm F. Đầu ra của thuật toán là phát biểu cho biết phép phân rã có bảo toàn phụ thuộc không.

(1) Với mỗi $X \rightarrow Y \in F$ khởi tạo tập các thuộc tính T bằng các thuộc tính của X (là tập thuộc tính xác định của phụ thuộc hàm đang xét). Nghĩa là gán T = X rồi tiếp tục bước 2.

(2) Lặp lại bước 3 cho đến khi tập T không thay đổi nữa. Khi đó làm tiếp bước 4.

(3) Với mỗi quan hệ R_i ($1 \le i \le k$) của phân rã, áp dụng tác vụ R_i tương ứng (trên tập thuộc tính T dựa trên tập phụ thuộc hàm F). Nghĩa là làm như sau: $T = T \cup ((T \cap R_i)^+ \cap R_i)$ và lặp lại bước 3.

(4) Kiểm tra xem Y (vế phải của phụ thuộc hàm đang xét) có là con của T hay không. Có hai trường hợp có thể xảy ra ở đây.
Nếu $Y \not\subset T$ thì chấm dứt thuật toán và xuất kết quả rằng phân rã $\rho = (R_1, R_2, \dots, R_k)$ không bảo toàn phụ thuộc hàm. Còn nếu $Y \subset T$ thì $X \rightarrow Y \in G^+$. Nếu còn phụ thuộc hàm khác trong F cần để kiểm tra thì lặp lại bước 1 với phụ thuộc hàm chưa xét đó. Còn nếu đã hết phụ thuộc hàm trong F thì tiếp tục bước 5.

(5) Hai tập phụ thuộc hàm là tương đương và thuật toán có thể trả về kết quả rằng phân rã $\rho = (R_1, R_2, \dots, R_k)$ là bảo toàn phụ thuộc hàm của quan hệ gốc.

Ví dụ

Hãy cho biết phân rã $\rho = \{R_1(X, Y), R_2(Y, Z), R_3(Z, W)\}$ của quan hệ r(X, Y, Z, W) có bảo toàn phụ thuộc hàm với $F = \{X \rightarrow Y, Y \rightarrow Z, Z \rightarrow W, W \rightarrow Z\}$ hay không.

Để quá trình chứng minh được dễ dàng, ta sẽ tuần tự xét các phụ thuộc hàm theo thứ tự liệt kê của chúng trong tập F.

(1) (lần 1) Xét $X \rightarrow Y$
Xét $X \rightarrow Y$, ta khởi tạo tập thuộc tính T là tập thuộc tính xác định của phụ thuộc hàm đang xét, tức $T = \{X\}$.

(2/3) (lần 1) Xét $X \rightarrow Y$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{X\} \cap \{X, Y\} = \{X\}$
$(T \cap R_1)^+ = \{X\}^+ = \{X, Y, Z, W\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_1)^+ \cap R_1 = \{Z, Y, Z, W\} \cap \{X, Y\} = \{X, Y\}$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{X\} \cup \{X, Y\} = \{X, Y\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{X, Y\}$
$T = \{X, Y\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(3) (lần 2) Xét $X \rightarrow Y$
Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có
$T \cap R_2 = \{X, Y\} \cap \{Y, Z\} = \{Y\}$
$(T \cap R_2)^+ = \{Y\}^+ = \{Y, Z, W, X\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_2)^+ \cap R_2 = \{X, Y, Z, W\} \cap \{Y, Z\} = \{Y, Z\}$
$T \cup ((T \cap R_2)^+ \cap R_2) = \{X, Y\} \cup \{Y, Z\} = \{X, Y, Z\}$
$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{X, Y, Z\}$
$T = \{X, Y, Z\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_3)

(3) (lần 3) Xét $X \rightarrow Y$
Áp dụng tác vụ R_3 $T \cup ((T \cap R_3)^+ \cap R_3)$ trên F ta có
$T \cap R_3 = \{X, Y, Z\} \cap \{Z, W\} = \{Z\}$
$(T \cap R_3)^+ = \{Z\}^+ = \{Z, W, X, Y\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_3)^+ \cap R_3 = \{X, Y, Z, W\} \cap \{Z, W\} = \{Z, W\}$
$T \cup ((T \cap R_3)^+ \cap R_3) = \{X, Y, Z\} \cup \{Z, W\} = \{X, Y, Z, W\}$
$T = T \cup ((T \cap R_3)^+ \cap R_3) = \{X, Y, Z, W\}$
$T = \{X, Y, Z, W\}$

Đến đây thuật toán sẽ lặp lại tác vụ R_i với T mới. Tuy nhiên T không thay đổi nữa. Chúng ta bỏ qua những bước đó vì chúng trả về các kết quả tương tự đã biết. Do đó ta tiếp tục ở bước 4.

(4) (lần 1) Xét $X \rightarrow Y$
Vì Y (vế phải của phụ thuộc hàm đang xét) thỏa $\{Y\} \subset T = \{X, Y, Z, W\}$ nên $X \rightarrow Y \in G^+$.

(1) (lần 1) Xét $Y \rightarrow Z$

Xét $Y \to Z$, ta khởi tạo tập thuộc tính T là tập thuộc tính xác định của phụ thuộc hàm đang xét, tức T = {Y}.

(2/3) (lần 1) Xét $Y \to Z$

Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có

$T \cap R_1 = \{Y\} \cap \{X, Y\} = \{Y\}$

$(T \cap R_1)^+ = \{Y\}^+ = \{Y, Z, W, X\}$ (áp dụng thuật toán Bao đóng)

$(T \cap R_1)^+ \cap R_1 = \{Y, Z, W, X\} \cap \{X, Y\} = \{Y, X\}$

$T \cup ((T \cap R_1)^+ \cap R_1) = \{Y\} \cup \{X, Y\} = \{Y, X\}$

$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Y, X\}$

T = {Y, X} (GIÁ trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(3) (lần 2) Xét $Y \to Z$

Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có

$T \cap R_2 = \{Y, X\} \cap \{Y, Z\} = \{Y\}$

$(T \cap R_2)^+ = \{Y\}^+ = \{Y, Z, W, X\}$ (áp dụng thuật toán Bao đóng)

$(T \cap R_2)^+ \cap R_2 = \{Y, Z, W, X\} \cap \{Y, Z\} = \{Y, Z\}$

$T \cup ((T \cap R_2)^+ \cap R_2) = \{Y, X\} \cup \{Y, Z\} = \{X, Y, Z\}$

$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{X, Y, Z\}$

T = {X, Y, Z} (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_3)

(3) (lần 3) Xét $Y \to Z$

Áp dụng tác vụ R_3 $T \cup ((T \cap R_3)^+ \cap R_3)$ trên F ta có

$T \cap R_3 = \{X, Y, Z\} \cap \{Z, W\} = \{Z\}$

$(T \cap R_3)^+ = \{Z\}^+ = \{Z, W, X, Y\}$ (áp dụng thuật toán Bao đóng)

$(T \cap R_3)^+ \cap R_3 = \{Z, W, X, Y\} \cap \{Z, W\} = \{Z, W\}$

$T \cup ((T \cap R_3)^+ \cap R_3) = \{X, Y, Z\} \cup \{Z, W\} = \{X, Y, Z, W\}$

$T = T \cup ((T \cap R_3)^+ \cap R_3) = \{X, Y, Z, W\}$

T = {X, Y, Z, W}

Đến đây thuật toán sẽ lặp lại tác vụ R_i với T mới. Tuy nhiên T không thay đổi nữa. Chúng ta bỏ qua bước đó vì chúng trả về các kết quả tương tự đã biết. Do đó ta tiếp tục ở bước 4.

(4) (lần 1) Xét $Y \to Z$

Vì Z (vế phải của phụ thuộc hàm đang xét) thỏa $\{Z\} \subset T = \{X, Y, Z, W\}$ nên $Y \to Z \in G^+$.

(1) (lần 1) Xét $Z \to W$

Xét $Z \to W$, ta khởi tạo tập thuộc tính T là tập hợp thuộc tính xác định của phụ thuộc hàm đang xét, tức T = {Z}.

(2/3) (lần 1) Xét $Z \to W$

Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có

$T \cap R_1 = \{Z\} \cap \{X, Y\} = \varnothing$

$(T \cap R_1)^+ = \varnothing^+ = \varnothing$

$(T \cap R_1)^+ \cap R_1 = \varnothing$

$T \cup ((T \cap R_1)^+ \cap R_1) = \{Z\} \cup \varnothing = \{Z\}$

$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Z\}$

T = {Z} (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(3) (lần 2) Xét $Z \to W$

Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có

$T \cap R_2 = \{Z\} \cap \{Y, Z\} = \{Z\}$

$(T \cap R_2)^+ = \{Z\}^+ = \{Z, W, X, Y\}$ (áp dụng thuật toán Bao đóng)

$(T \cap R_2)^+ \cap R_2 = \{Z, W, X, Y\} \cap \{Y, Z\} = \{Z, Y\}$

$T \cup ((T \cap R_2)^+ \cap R_2) = \{Z\} \cup \{Z, Y\} = \{Z, Y\}$

$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{Z, Y\}$

T = {Z, Y} (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_3)

(3) (lần 3) Xét $Z \to W$

Áp dụng tác vụ R_3 $T \cup ((T \cap R_3)^+ \cap R_3)$ trên F ta có

$T \cap R_3 = \{Z, Y\} \cap \{Z, W\} = \{Z\}$

$(T \cap R_3)^+ = \{Z\}^+ = \{Z, W, X, Y\}$ (áp dụng thuật toán Bao đóng)

$(T \cap R_3)^+ \cap R_3 = \{Z, W, X, Y\} \cap \{Z, W\} = \{Z, W\}$

$T \cup ((T \cap R_3)^+ \cap R_3) = \{Z, Y\} \cup \{Z, W\} = \{Z, Y, W\}$

$T = T \cup ((T \cap R_3)^+ \cap R_3) = \{Z, Y, W\}$

T = {Z, Y, W}

Đến đây thuật toán sẽ lặp lại tác vụ R_i với T mới.

(3) (lần 4) Xét $Z \to W$

Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có

$T \cap R_1 = \{Z, Y, W\} \cap \{X, Y\} = \{Y\}$

$(T \cap R_1)^+ = \{Y\}^+ = \{Y, Z, W, X\}$

$(T \cap R_1)^+ \cap R_1 = \{X, Y\}$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{X, Y, W\} \cup \{X, Y\} = \{Z, Y, W\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, Y, W\}$
$T = \{Z, Y, W\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(4) (lần 1) Xét $Z \rightarrow W$
Vì W (vế phải của phụ thuộc hàm đang xét) thỏa $\{W\} \subset T = \{Z, Y, W\}$ nên $Z \rightarrow W \in G^+$.

(1) (lần 1) Xét $W \rightarrow X$
Xét $W \rightarrow X$, ta khởi tạo tập thuộc tính T là tập thuộc tính xác định của phụ thuộc hàm đang xét, tức $T = \{W\}$.

(2/3) (lần 1) Xét $W \rightarrow X$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{W\} \cap \{X, Y\} = \varnothing$
$(T \cap R_1)^+ = \varnothing^+ = \varnothing$
$(T \cap R_1)^+ \cap R_1 = \varnothing$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{W\} \cup \varnothing = \{W\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{W\}$
$T = \{ W\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(3) (lần 2) Xét $W \rightarrow X$
Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có
$T \cap R_2 = \{W\} \cap \{Y, Z\} = \varnothing$
$(T \cap R_2)^+ = \varnothing^+ = \varnothing$
$(T \cap R_2)^+ \cap R_2 = \{W\} \cap \varnothing = \varnothing$
$T \cup ((T \cap R_2)^+ \cap R_2) = \{W\} \cup \varnothing = \{W\}$
$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{W\}$
$T = \{ W\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_3)

(3) (lần 3) Xét $W \rightarrow X$
Áp dụng tác vụ R_3 $T \cup ((T \cap R_3)^+ \cap R_3)$ trên F ta có
$T \cap R_3 = \{W\} \cap \{Z, W\} = \{W\}$
$(T \cap R_3)^+ = \{W\}^+ = \{W, X, Y, Z\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_3)^+ \cap R_3 = \{W, X, Y, Z\} \cap \{Z, W\} = \{Z, W\}$
$T \cup ((T \cap R_3)^+ \cap R_3) = \{W\} \cup \{Z, W\} = \{Z, W\}$
$T = T \cup ((T \cap R_3)^+ \cap R_3) = \{Z, W\}$
$T = \{Z, W\}$

(3) (lần 4) Xét $W \rightarrow X$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{Z, W\} \cap \{X, Y\} = \varnothing$
$(T \cap R_1)^+ = \varnothing^+ = \varnothing$
$(T \cap R_1)^+ \cap R_1 = \varnothing$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, W\} \cup \varnothing = \{Z, W\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, W\}$
$T = \{Z, W\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(3) (lần 4) Xét $W \rightarrow X$
Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có
$T \cap R_2 = \{Z, W\} \cap \{X, Y\} = \{Z\}$
$(T \cap R_2)^+ = \{Z\}^+ = \{Z, W, X, Y\}$
$(T \cap R_2)^+ \cap R_2 = \{Z, W, X, Y\} \cap \{Y, Z\} = \{Y, Z\}$
$T \cup ((T \cap R_2)^+ \cap R_2) = \{Z, W\} \cup \{Y, Z\} = \{Z, W, Y\}$
$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{Z, W, Y\}$
$T = \{Z, W, Y\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_3)

(3) (lần 5) Xét $W \rightarrow X$
Áp dụng tác vụ R_3 $T \cup ((T \cap R_3)^+ \cap R_3)$ trên F ta có
$T \cap R_3 = \{Z, W, Y\} \cap \{Z, W\} = \{Z, W\}$
$(T \cap R_3)^+ = \{ZW\}^+ = \{W, X, Y, Z\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_3)^+ \cap R_3 = \{W, X, Y, Z\} \cap \{Z, W\} = \{Z, W\}$
$T \cup ((T \cap R_3)^+ \cap R_3) = \{Z, W, Y\} \cup \{Z, W\} = \{Z, W, Y\}$
$T = T \cup ((T \cap R_3)^+ \cap R_3) = \{Z, W, Y\}$

(3) (lần 4) Xét $W \rightarrow X$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{Z, W, Y\} \cap \{X, Y\} = \{Y\}$
$(T \cap R_1)^+ = \{Y\}^+ = \{Y, Z, W, X\}$
$(T \cap R_1)^+ \cap R_1 = \{ Y, Z, W, X \} \cap \{X, Y\} = \{X, Y\}$

$T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, W, Y\} \cup \{X, Y\} = \{Z, W, Y, X\}$

$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, W, Y, X\}$

$T = \{Z, W, Y, X\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)

(3) (lần 4) Xét $W \rightarrow X$

Đến đây thuật toán sẽ lặp lại bước 3 với tác vụ R_i. Tuy nhiên T không thay đổi nữa. Chúng ta bỏ qua những bước đó vì chúng trả về các kết quả tương tự đã biết. Do đó ta tiếp tục ở bước 4.

(4) (lần 1) Xét $W \rightarrow X$

Vì X (vế phải của phụ thuộc hàm đang xét) thỏa $\{X\} \subset T = \{Z, W, Y, X\}$ nên $W \rightarrow X \in G^+$.

(5) (lần 1) Xét $W \rightarrow X$

Vì tất cả các phụ thuộc hàm của F đều là thành viên của G^+ nên hai tập phụ thuộc hàm là tương đương nghĩa là phân rã bảo toàn phụ thuộc hàm của F.

Nãy giờ ta đã xem xét cá thủ tục có hệ thống để kiểm tra xem một phân rã có bảo toàn phụ thuộc hàm không, còn trước đây cũng có đã nói đến một phương pháp để kiểm tra phân rã bảo toàn. Bạn nên nhớ rằng hai tính chất này là độc lập với nhau. Nghĩa là có thể có phân rã bảo toàn nhưng không hề bảo toàn phụ thuộc hàm và ngược lại. Bài tập 15 sẽ đề cập đến một thuật toán cho phép ta tìm một phân rã đạt 3NF bảo toàn và cả bảo toàn phụ thuộc hàm. Bài tập 16 đề cập đến một thuật toán tìm một phân rã đạt BCNF bảo toàn.

Bài tập có lời giải

1. Trong bảng EMPLOYEE dưới đây, hãy chỉ rõ thuộc tính định danh, tất cả các thuộc tính lặp lại và không lặp lại của bảng. Làm phẳng bảng và cho biết kết quả đó có phải là quan hệ không. Nếu không thì làm sao để chúng thành quan hệ?

EMPLOYEE

ID	Last-Name	Department	Dependent-Name	Dependent-DOB	Dependent-Sex	Dependent-ID
322135609	Cordani	CS	Mary Cindy John	01/12/60 04/24/65 07/12/68	F F M	800980432 953635262 992556631
423542641	Strange	VP	Fern Victoria	03/28/62 11/12/84	F F	790902462 800234979
536234809	VanKlaveren	Sales	Sadie	08/31/65	F	970073473
632390802	Miller	Sales	Sallie	09/21/45	F	890721289
980772345	Kroeger	MIS	Jonathan	08/15/85	M	943632552

Thuộc tính định danh của bảng là thuộc tính ID. Các thuộc tính không lặp lại là ID, Last-Name và Department. Các thuộc tính lặp lại là Dependent-Name, Dependent-Sex và Dependent-ID.

Để làm phẳng bảng ta điền đầy vào những ô nhập bằng cách sao chép thông tin của các thuộc tính không lặp lại. Bảng được chuẩn hóa như sau:

EMPLOYEE

ID	Last-Name	Department	Dependent-Name	Dependent-DOB	Dependent-Sex	Dependent-ID
322135609	Cordani	CS	Mary	01/12/60	F	800980432
322135609	Cordani	CS	Cindy	04/24/65	F	953635262
322135609	Cordani	CS	John	07/12/68	M	992556631
423542641	Strange	VP	Fern	03/28/62	F	790902462
423542641	Strange	VP	Victoria	11/12/84	F	800234979
536234809	VanKlaveren	Sales	Sadie	08/31/65	F	970073473
632390802	Miller	Sales	Sallie	09/21/45	F	890721289
980772345	Kroeger	MIS	Jonathan	08/15/85	M	943632552

Bảng này không phải là quan hệ vì nó không có khóa chính. Ví dụ như ID không thể xác định được giá trị nào trong dòng đầu tiên. Để bảng này có thể là quan hệ ta cần xác định được một khóa chính phù hợp cho quan hệ. Khóa phức (ID, Dependent-ID) có vẻ là khóa chính phù hợp nhất.

2. Hãy chuẩn hóa bảng ban đầu bằng cách tạo hai quan hệ mới.

Để chuẩn hóa bảng này ta cần tạo hai quan hệ mới. Các thuộc tính của quan hệ thứ nhất là thuộc tính định danh của bảng cộng thêm các thuộc tính không lặp lại. Còn thuộc tính của quan hệ thứ hai là thuộc tính định danh của bảng cộng thêm các thuộc tính lặp lại. Lược đồ của hai quan hệ mới được trình bày dưới đây. Thuộc tính ID (của Employee) đã được đổi tên trong quan hệ Dependent.

Employee (<u>ID</u>, Last-Name, Department)

Dependent (<u>Emp-ID</u>, <u>Dependent-ID</u>, Dependent-Name, Dependent-Sex)

Sau đây là các thể hiện của hai quan hệ trên. Chú ý là ở đây không còn có các dòng trùng. Các khóa chính cũng là được định nghĩa.

Employee

ID	Last-Name	Department
322135609	Cordani	CS

423542641	Strange	VP
536234809	VanKlaveren	Sales
632390802	Miller	Sales
980772345	Kroeger	MIS

Dependent

Emp-ID	Dependent-Name	Dependent-DOB	Dependent-Sex	Dependent-ID
322135609	Mary	01/12/60	F	800980432
322135609	Cindy	04/24/65	F	953635262
322135609	John	07/12/68	M	992556631
423542641	Fern	03/28/62	F	790902462
423542641	Victoria	11/12/84	F	800234979
536234809	Sadie	08/31/65	F	970073473
632390802	Sallie	09/21/45	F	890721289
980772345	Jonathan	08/15/85	M	943632552

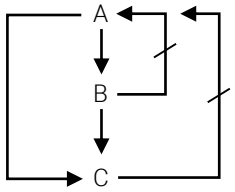
3. Xét lược đồ quan hệ của quan hệ SCHEDULE dưới đây. Dạng chuẩn cao nhất mà quan hệ này đạt được là gì? Kiểu dữ liệ xấu của quan hệ này là gì? Hãy cho một ví dụ để minh họa cho kiểu dữ liệu xấu đó.
SCHEDULE (Student-ID, Class-No, Student-Name, Student-Major, Class-Time, Building-Room, Instructor).
Giả sử có các phụ thuộc hàm sau:

Student-ID → Student-Name
Student-ID → Student-Major
Class-No → Class-Time
Class-No → Building-Room
Class-No → Instructor

Có một số phụ thuộc một phần vào khóa, ví dụ như Class-No → Building-Room hay Student-ID → Student-Major, nên dạng chuẩn cao nhất mà quan hệ này đạt được là 1NF.
Vì quan hệ ở 1NF nên chắc chắn sẽ có dữ liệu xấu khi chèn/xóa hay cập nhật dữ liệu. Ví dụ khi có một sinh viên thôi học thì quan hệ sẽ có dữ liệu xấu khi xóa vì các thông tin về sinh viên như Student-ID, Student-Name và Student-Major (ngành học) sẽ mất. Hơn nữa nếu đó là sinh viên duy nhất trong lớp, thì thông tin về Instructor (giảng viên) cũng mất luôn. Vì có dữ liệu xấu khi xóa nên chắc chắn có dữ liệu xấu khi chèn. Ví dụ như thông tin về Class-Time (giờ học), Building-Room (chỗ học) và Instructor (giảng viên) không thể được chèn vào quan hệ cho đến khi có một sinh viên đăng ký lớp học đó. Dữ liệu xấu khi cập nhật cũng có, ví dụ khi có lớp thay đổi chỗ học. Trong trường hợp đó ta phải bảo đảm rằng tất cả các bộ tương ứng đều được thay đổi theo để tránh trạng thái dữ liệu thiếu nhất quán.

4. Xét lược đồ quan hệ và tập phụ thuộc hàm sau. Dạng chuẩn cao nhất của quan hệ là gì? Hãy chuyển quan hệ này lên dạng chuẩn tiếp theo. Khi đó thông tin có thể được phục hồi không? Cần làm gì để phục hồi nó?
Programmer-Task (Programmer-ID, Programming-Package-ID, Programmer-Package-Name, Total-Hours-Worked-on-Package)
Programmer-Package-ID → Programmer-Package-Name
Dạng chuẩn cao nhất của quan hệ này là 1NF vì có các phụ thuộc một phần trên khóa phức. Ví dụ như Programmer-Package-ID → Programmer-Package-Name.
Dạng chuẩn tiếp theo của quan hệ này là 2NF. Để chuyển lên 2NF ta cần tạo hai quan hệ mới. Quan hệ thứ nhất có khóa như khóa của quan hệ cũ: Programmer-ID, Programmer_Package-ID. Lược đồ của quan hệ thứ nhất như sau:
Programmer-Activity (Programmer-ID, Programming-Package-ID, Total-Hours-Worked-on-Package)
Lược đồ thứ hai có khóa là một phần khóa của lược đồ cũ: Programming-Package-ID. Lược đồ của quan hệ thứ hai như sau:
Package-Info (Programming-Package-ID, Programming-Package-Name)
Thông tin của quan hệ cũ có thể được phục hồi bằng phép kết trên thuộc tính chung Programming-Package-ID.

5. Hãy chứng minh rằng hai định nghĩa cho 3NF là tương đương.
Các điều kiên trong dưới là các điều kiện tổng quát cho các phụ thuộc hàm bắc cầu. Mọi quan hệ thỏa các điều kiện này đều có phụ thuộc hàm bắc cầu và do đó không ở 3NF. Giả sử A là khóa của quan hệ thì đương nhiên các thuộc tính B và C không thể là khóa vì chúng không xác định A. Vậy B và C là các thuộc tính không khóa. Chú ý là cho dù ta trực tiếp có A → C thì cũng không tránh được kết luận rằng quan hệ có phụ thuộc hàm bắc cầu.
Trong hình, cách duy nhất để tránh có A → C là không có B → C. Nếu ta tổng quát hóa tình huống này với mọi thuộc tính không khóa B và C thì cả hai định nghĩa về 3NF đều tương đương.



Các điều kiện để định nghĩa sự phụ thuộc bắc cầu

của thuộc tính C vào thuộc tính A.

6. Xét lược đồ quan hệ (Sales-Transaction-No, Item-No, Item-Price, Item-Quantity-Sold, Seller, Seller-District) và các phụ thuộc hàm dưới đây. Khóa của quan hệ là gì? Hãy chuyển quan hệ về 3NF.

Sales-Transaction-No, Item-No → Item-Quantity-Sold
Item-No → Item-Price
Sales-Transaction-No → Seller
Seller → Seller-District

Khóa của quan hệ này là khóa phức gồm: Sales-Transaction-No, Item-No. Lý do là hai thuộc tính này có thể xác định mọi thuộc tính, ta sẽ chứng minh như sau:

a. Sales-Transaction-No, Item-No → *Sales-Transaction-No* (tiên đề Reflexivity)
b. Sales-Transaction-No, Item-No → *Item-No* (tiên đề Reflexivity)
c. Sales-Transaction-No, Item-No → *Item-Quantity-Sold* (cho sẵn)
d. Từ Sales-Transaction-No, Item-No → Item-No (tiên đề Reflexivity) và Item-No → Item-Price (cho sẵn) nên ta có Sales-Transaction-No, Item-No → *Item-Price* (tiên đề Transitivity)
e. Từ Sales-Transaction-No, Item-No → Sales-Transaction-No (tiên đề Reflexivity) và Sales-Transaction-No → Seller (cho sẵn) nên ta có Sales-Transaction-No, Item-No → *Seller* (tiên đề Transitivity)
f. Từ Sales-Transaction-No, Item-No → Seller (bước e) và Seller → Seller-District (cho sẵn) nên ta có Sales-Transaction-No, Item-No → *Seller-District* (tiên đề Transitivity).

Dạng chuẩn cao nhất của quan hệ này là 1NF vì có các phụ thuộc một phần vào khóa chính.
Để chuyển lên 2NF, chú ý rằng ta cần xét riêng từng phụ thuộc một phần. Do đó, thay vì thay thế quan hệ gốc bằng hai quan hệ mới, ta sẽ thay thế bằng ba quan hệ mới. Các quan hệ đó là:

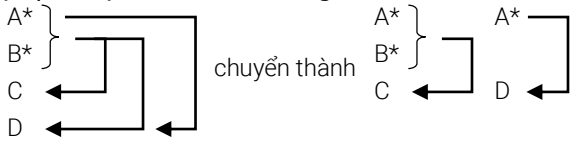
a. Item-Sold (Sales-Transaction-No, Item-No, Item-Quantity-Sold)
b. Item (Item-No, Item-Price)
a. Seller (Sales-Transaction-No, Seller, Seller-District)

Các quan hệ (a) và (b) đều ở 3NF vì chỉ có một thuộc tính không khóa phụ thuộc hoàn toàn vào khóa. Còn quan hệ (c) chưa ở 3NF vì một trong các thuộc tính không khóa là Seller lại phụ thuộc vào một thuộc tính không khóa khác là Seller-District. Quan hệ này có thể được chuyển lên 3NF bằng cách: từ quan hệ này ta có hai quan hệ mới như sau:
Seller-Transaction (Sales-Transaction-No, Seller)
Seller_District (Seller, District)

7. Hãy chuyển quan hệ ở bài tập 3 lên 2NF. Quan hệ và tất cả các phụ thuộc của chúng được viết lại dưới đây.
SCHEDULE (Student-ID, Class-No, Student-Name, Student-Major, Class-Time, Building-Room, Instructor).
Giả sử có các phụ thuộc hàm sau:

Student-ID → Student-Name
Student-ID → Student-Major
Class-No → Class-Time
Class-No → Building-Room
Class-No → Instructor

Trong mọi trường hợp khi muốn chuyển một quan hệ về 2NF ta đều dùng hình dưới.



Tuy nhiên trong trường hợp này giải pháp không đơn giản như vậy. Lý do vì là khóa phức và mỗi thuộc tính khóa đều phụ thuộc một phần vào các thuộc tính không khóa. Do đó thay vì có hai quan hệ mới ta sẽ tạo ít nhất ba quan hệ. Các quan hệ đó là:
STUDENT (Student-ID, Student-Name, Student-Major). Chú ý là mọi thuộc tính trong quan hệ này đều phụ thuộc vào khóa Student-ID.
CLASS (Class-No, Class-Time, Building-Room, Instructor). Chú ý là mọi thuộc tính trong quan hệ này đều phụ thuộc vào khóa Class-No.
Vì quá trình chuẩn hóa có thể đảo ngược nên ta phải bảo đảm rằng có thể phục hồi quan hệ cũ, do đó ta cần thêm một quan hệ nữa để bảo đảm điều vừa rồi. Trong trường hợp này, ta sẽ tạo một quan hệ mới là
STUDENT_CLASS(Student-Id, Class-No)

8. Xét quan hệ r(X, Y, Z, W) và tập F = {Y ↔ W, XY → Z} trong đó ↔ có nghĩa là Y → W và W → Y. Hãy cho biết các khóa dự tuyển của quan hệ? Dạng chuẩn cao nhất của quan hệ là gì?
Quan hệ này có hai khóa dự tuyển là XY và WX.
- XY là khóa dự tuyển vì XY có thể xác định tất cả các thuộc tính như sau:

a. XY → X (tiên đề Refexivity)
b. XY → Y (tiên đề Refexivity)
c. XY → Z (cho sẵn)
d. XY → Y (bước b) và Y → W nên XY → W (tiên đề Transitivity)

- WX là khóa dự tuyển vì WX có thể xác định tất cả các thuộc tính như sau:

a. $WX \rightarrow W$ (tiên đề Refexivity)

b. $WX \rightarrow X$ (tiên đề Refexivity)

c. $XY \rightarrow Z$ (cho sẵn) và $W \rightarrow Y$ (cho sẵn) nên $XW \rightarrow Z$ (tiên đề Pseudotransitivity)

d. $WX \rightarrow W$ (bước a) và $W \rightarrow Y$ (cho sẵn) nên $WX \rightarrow Y$ (tiên đề Transitivity)

Các thuộc tính khóa là X, Y và W. Chỉ có một thuộc tính không khóa là Z.
Quan hệ ở 2NF vì không có thuộc tính không khóa phụ thuộc một phần vào khóa. Trong cả hai trường hợp ta đều có Z phụ thuộc hoàn toàn vào khóa. Quan hệ ở 3NF vì chỉ có một thuộc tính không khóa. Do đó không thể có chuyện thuộc tính không khóa này có thể xác định thuộc tính không khóa khác. Quan hệ không ở BCNF vì vế trái của $W \rightarrow Y$ hay $Y \rightarrow W$ không phải là khóa của quan hệ. Do đó dạng chuẩn cao nhất mà quan hệ đạt được là 3NF.

9. Cho quan hệ r(X, Y, W, Z, P, Q) và tập $F = \{XY \rightarrow W, XW \rightarrow P, PQ \rightarrow Z, XY \rightarrow Q\}$. Giả sử quan hệ r được phân rã thành $R_1(Z, P, Q)$ và $R_2(X, Y, Z, P, Q)$. Hỏi phân rã có bảo toàn không? Hãy dùng thuật toán kết bảo toàn để giải thích.

(1) Xây dựng bảng có sáu cột (mỗi cột cho một thuộc tính) và hai dòng (mỗi dòng cho một quan hệ con). Đặt tên cho cột là tên các thuộc tính và tên cho dòng là tên của các quan hệ con.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$P(A_5)$	$Q(A_6)$
R_1						
R_2						

(2) Điền vào các ô trong bảng như sau:
Các thuộc tính của lược đồ R_1 là: $Z(A_3)$, $P(A_5)$, $Q(A_6)$. Do đó ở các ô tương ứng ta sẽ điền vào a_3, a_5 và a_6. Các ô nhập còn lại là b_{11}, b_{12} và b_{14}. Các thuộc tính của lược đồ R_2 là: $X(A_1)$, $Y(A_2)$, $W(A_4)$, $P(A_5)$, $Q(A_6)$. Do đó ở các ô tương ứng ta sẽ điền vào a_1, a_2, a_4, a_5 và a_6. Ô còn lại là b_{23}.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$P(A_5)$	$Q(A_6)$
R_1	b_{11}	b_{12}	a_3	b_{14}	a_5	a_6
R_2	a_1	a_2	b_{23}	a_4	a_5	a_6

(3) (lần 1) Xét $XY \rightarrow W$ ta sẽ kiểm tra trên hai dòng của bảng xem có các giá trị nào giống nhau ở các cột là vế trái của phụ thuộc hàm không. Vì không có sự giống nhau ở hai dòng trên các cột X và Y nên ta lặp lại bước 3 nhưng xét phụ thuộc hàm khác.

(3) (lần 2) Xét $XW \rightarrow P$ ta sẽ kiểm tra trên hai dòng bảng xem có các giá trị giống nhau ở các cột là vế trái của phụ thuộc hàm không. Vì không có sự giống nhau ở hai dòng trên các cột X và W nên ta lặp lại bước 3 nhưng xét phụ thuộc hàm khác.

(3) (lần 3) Xét $PQ \rightarrow Z$ ta sẽ kiểm tra trên hai dòng của bảng xem có các giá trị giống nhau ở các cột là vế trái của phụ thuộc hàm không. Rõ ràng có sự giống nhau ở hai dòng trên các cột P và Q, mà một trong hai giá trị ở cột Z là a_3 nên ta thay giá trị ở ô kia (b_{23}) bởi giá trị a_3.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$P(A_5)$	$Q(A_6)$
R_1	b_{11}	b_{12}	a_3	b_{14}	a_5	a_6
R_2	a_1	a_2	a_3	a_4	a_5	a_6

Mặc dù chưa xét hết phụ thuộc hàm nhưng có tiếp tục thì bảng cũng không thay đổi, nên ta ngưng tại đây và chuyển qua bước 4.

(4) (lần 1) Vì có một dòng chỉ chứa tất cả a nên phân rã là bảo toàn.

10. Cho quan hệ r(X, Y, Z, W, Q) và tập $F = \{XY \rightarrow WQ, Z \rightarrow Q, W \rightarrow Z, Q \rightarrow X\}$. Giả sử có một phân rã nào đó và ta được một quan hệ con $R_1(X, Y, Z)$. Hãy tìm tập phụ thuộc hàm thỏa quan hệ R_1 bằng cách chiếu tập F lên quan hệ con đó.
Để xác định $\pi_{(X, Y, Z)}(F)$ ta xét các tập con thực sự của lược đồ (X, Y, Z) và tính bao đóng của nó.
Các tập con đó là {X}, {Y}, {Z}, {X, Y}, {Y, Z}, {X, Z}.
Bao đóng của các tập đơn là:
$\{X\}^+ = \{X\}$, $\{Y\}^+ = \{Y\}$, $\{Z\}^+ = \{Z, Q, X\}$
Còn bao đóng của các tập hai phần tử:
$\{XY\}^+ = \{X, Y, W, Q, Z\}$
$\{YZ\}^+ = \{Y, Z, Q, X, W, Q\}$
$\{XZ\}^+ = \{X, Z, Q\}$
Với mỗi bao đóng, ta sẽ chọn $X \rightarrow Y$ nếu Y thuộc về $\{X\}^+$ và các điều kiện sau đồng thời thỏa
(a) $Y \subset \{X, Y, Z\}$,
(b) $Y \in X^+$
(c) $Y \not\subset X$.
Các bao đóng $\{X\}^+$ và $\{Y\}^+$ chỉ có các phụ thuộc hàm tầm thường, không có trong $\pi_{(X, Y, Z)}(F)$.
Do đó từ $\{Z\}^+$ ta chọn $Z \rightarrow X$. Nghĩa là ta đưa phụ thuộc hàm này vào tập phụ thuộc hàm của {X, Y, Z}. Ta cũng có $Z \rightarrow Q$ nhưng thuộc tính Q không phải là một phần của lược đồ {X, Y, Z} theo điều kiện (a) nêu trên.
Từ bao đóng $\{XY\}^+ = \{X, Y, W, Q, Z\}$ ta có $XY \rightarrow Z$

Từ bao đóng $\{YZ\}^+ = \{Y, Z, Q, X, W, Q\}$ ta có $YZ \to X$
Từ bao đóng $\{XZ\}^+ = \{X, Z, Q\}$ ta chỉ có các phụ thuộc hàm tầm thường
Các thành phần của $\pi_{(X, Y, Z)}(F)$ là $\{XY \to Z, YZ \to X, Z \to X\}$
Bỏ bớt các phụ thuộc hàm dư thừa ta có $\pi_{(X, Y, Z)}(F) = \{XY \to Z, Z \to X\}$

11. Xét quan hệ r(X, Y, Z, W) và một phân rã của nó $\rho = \{R_1(X, Y), R_2(Z, W)\}$. Quan hệ này có bảo toàn phụ thuộc hay không nếu $F = \{X \to Y, Z \to W\}$.
Ta sẽ lần lượt xét từng phụ thuộc hàm trong tập phụ thuộc hàm đã cho.
(1) (lần 1) Xét $X \to Y$
Khởi tọa T là các thuộc tính ở vế trái của phụ thuộc hàm đang xét, nghĩa là $T = \{X\}$.
(2/3) (lần 1) Xét $X \to Y$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{X\} \cap \{X, Y\} = \{X\}$
$(T \cap R_1)^+ = \{X\}^+ = \{X, Y\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_1)^+ \cap R_1 = \{X, Y\} \cap \{X, Y\} = \{X, Y\}$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{X\} \cup \{X, Y\} = \{X, Y\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{X, Y\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)
(3) (lần 2) Xét $X \to Y$
Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có
$T \cap R_2 = \{X, Y\} \cap \{Z, W\} = \varnothing$
$(T \cap R_2)^+ = \varnothing^+ = \varnothing$
$(T \cap R_2)^+ \cap R_2 = \varnothing \cap \{Z, W\} = \varnothing$
$T \cup ((T \cap R_2)^+ \cap R_2) = \{X, Y\} \cup \varnothing = \{X, Y\}$
$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{X, Y\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_1)
(3) (lần 3) Xét $X \to Y$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{X, Y\} \cap \{X, Y\} = \{X, Y\}$
$(T \cap R_1)^+ = \{X, Y\}^+ = \{X, Y\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_1)^+ \cap R_1 = \{X, Y\} \cap \{X, Y\} = \{X, Y\}$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{X, Y\} \cup \{X, Y\} = \{X, Y\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{X, Y\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)
Vì tập T không thay đổi nữa nên ta tiếp tục ở bước 4.
(4) (lần 1) Xét $X \to Y$
Vì Y (vế phải của phụ thuộc hàm đang xét) thỏa $\{Y\} \subset T = \{X, Y\}$ nên ta có $X \to Y \in G^+$ trong đó $G = \bigcup_{i=1}^{k} \pi_{Ri}(F)$
(2/3) (lần 1) Xét $Z \to W$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{Z\} \cap \{X, Y\} = \varnothing$
$(T \cap R_1)^+ = \varnothing^+ = \varnothing$
$(T \cap R_1)^+ \cap R_1 = \varnothing \cap \{X, Y\} = \varnothing$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{Z\} \cup \varnothing = \{Z\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Z\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_2)
(3) (lần 2) Xét $Z \to W$
Áp dụng tác vụ R_2 $T \cup ((T \cap R_2)^+ \cap R_2)$ trên F ta có
$T \cap R_2 = \{Z\} \cap \{Z, W\} = \{Z\}$
$(T \cap R_2)^+ = \{Z\}^+ = \{Z, W\}$ (áp dụng thuật toán Bao đóng)
$(T \cap R_2)^+ \cap R_2 = \{Z, W\} \cap \{Z, W\} = \{Z, W\}$
$T \cup ((T \cap R_2)^+ \cap R_2) = \{Z\} \cup \{Z, W\} = \{Z, W\}$
$T = T \cup ((T \cap R_2)^+ \cap R_2) = \{Z, W\}$ (giá trị này của T sẽ được dùng lại ở lần lặp tiếp theo với R_1)
(3) (lần 3) Xét $Z \to W$
Áp dụng tác vụ R_1 $T \cup ((T \cap R_1)^+ \cap R_1)$ trên F ta có
$T \cap R_1 = \{Z, W\} \cap \{X, Y\} = \varnothing$
$(T \cap R_1)^+ = \varnothing^+ = \varnothing$
$(T \cap R_1)^+ \cap R_1 = \varnothing \cap \{X, Y\} = \varnothing$
$T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, W\} \cup \varnothing = \{Z, W\}$
$T = T \cup ((T \cap R_1)^+ \cap R_1) = \{Z, W\}$
Vì tập T không thay đổi nữa nên ta tiếp tục ở bước 4.
(4) (lần 1) Xét $Z \to W$
Vì W (vế phải của phụ thuộc hàm đang xét) thỏa $\{Y\} \subset T = \{Z, W\}$ nên ta có $Z \to W \in G^+$ trong đó $G = \bigcup_{i=1}^{k} \pi_{Ri}(F)$
(5) (lần 1) Các kết quả trước cho thấy cả hai X Y và Z W đều thuộc về G^+, do đó phân rã $\rho = \{R_1(X, Y), R_2(Z, W)\}$ là bảo toàn phụ thuộc.

12. Hãy chứng minh rằng mọi quan hệ có hai thuộc tính đều ở BCNF. Nghĩa là nếu r(X, Y) thì r(X, Y) đạt BCNF.
Ta sẽ xét các trường hợp sau:

a. X là khóa duy nhất của quan hệ. Trong trường hợp này, phụ thuộc hàm không tầm thường $X \rightarrow Y$ có X là siêu khóa vì $X \subset X$.

b. Y là khóa duy nhất của quan hệ. Trong trường hợp này, phụ thuộc hàm không tầm thường $Y \rightarrow X$ có Y là siêu khóa vì $Y \subset Y$.

c. Cả $X \rightarrow Y$ và $Y \rightarrow X$. Khi đó bất kể khóa chính là gì thì ta đều có X hoặc Y là thuộc tính xác định. Khi đó ta lại rơi vào một trong hai trường hợp a hay b đã xét.

13. Xét quan hệ r(A, B, C, D, E) và tập $F = \{AB \rightarrow CE, E \rightarrow AB, C \rightarrow D\}$. Dạng chuẩn cao nhất của quan hệ này là gì?
Vì r là quan hệ nên theo định nghĩa, r ở 1NF.
Để xác định dạng chuẩn của quan hệ này, ta cần xác định tất cả các khóa có thể có của quan hệ. Hai khóa có thể có của quan hệ là AB và E. Xét từng trường hợp:
- AB là khóa

(1) $AB \rightarrow A$ (tiên đề Reflexivity)

(2) $AB \rightarrow B$ (tiên đề Reflexivity)

(3) Từ $AB \rightarrow CE$ (cho sẵn) ta có $AB \rightarrow C$ (tiên đề Projectivity)

(4) Từ $AB \rightarrow CE$ (cho sẵn) ta có $AB \rightarrow E$ (tiên đề Projectivity)

(5) Từ $AB \rightarrow C$ (bước 3) và đã có $C \rightarrow D$ (cho sẵn) ta có $AB \rightarrow D$ (tiên đề Transitivity)

Vì AB có thể xác định mọi thuộc tính trong quan hệ nên AB là khóa.
- E là khóa

Vì $E \rightarrow AB$ (cho sẵn) và AB là khóa nên E cũng là khóa, bởi áp dụng tiên đề Transitivity nó có thể xác định hàm tất cả các thuộc tính của quan hệ.

Như vậy các thuộc tính khóa là A, B và E.
Các thuộc tính không khóa là C và D.
Vậy quan hệ ở 2NF vì không có chuyện phụ thuộc một phần vào khóa ở đây.
Quan hệ không ở 3NF vì có phụ thuộc hàm bắc cầu váo khóa, ví dụ như $C \rightarrow D$, trong đó cả hai thuộc tính đều là các thuộc tính không khóa.

14. Cho quan hệ r(A, B, C) và các phụ thuộc hàm $A \rightarrow B$ và $B \rightarrow C$. Ta dễ thấy r chưa ở 3NF vì có phụ thuộc bắc cầu vào khóa A. Vậy quan hệ có có ở 3NF không nếu có thêm phụ thuộc hàm $C \rightarrow B$?
Khi đó quan hệ vẫn chưa ở 3NF. Phụ thuộc hàm $C \rightarrow B$ là không cần thiết.

15. Thuật toán dưới đây có đầu vào là quan hệ r(R), phủ hợp quy F_c của các phụ thuộc hàm và khóa dự tuyển K của quan hệ. Đầu ra của thuật toán là một phân rã ở dạng chuẩn ba $\rho = (R_1, R_2, R_3, \ldots, R_n)$ bảo toàn và bảo toàn phụ thuộc.

(1) Tìm tất cả các thuộc tính của lược đồ quan hệ r không xuất hiện như một phần của vế trái hoặc một phần của vế phải của mọi phụ thuộc hàm trong F_c. Nếu có thuộc tính như vậy, ta tạo tập A với các thuộc tính nà rồi thực hiện các bước 1-a và 1-b, còn không chuyển qua bước 2.

(1-a) Tạo quan hệ với các thuộc tính của tập A, nghĩa là tạo r(A).

(1-b) Bỏ các thuộc tính có trong tập A khỏi lược đồ của quan hệ r, nghĩa là $R = R - \{A\}$

(2) Tìm các phụ thuộc hàm $X \rightarrow Y$ trong F_c sao cho tất cả các thuộc tính tồn tại trong xuất hiện trong phụ thuộc hàm này. Nếu tìm thấy thì tạo quan hệ r(X, Y) rồi kết thúc thuật toán trên ở bước 3.

(3) Với mỗi $X \rightarrow A$ trong F_c (A là thuộc tính đơn F_c là phủ hợp quy), tạo quan hệ $R_1(X, A)$ có $X \rightarrow A$. Sau khi đã xét hết các phụ thuộc hàm trong F_c, chuyển qua bước 4.

(4) Kết hợp, nếu có thể, tất cả các lược đồ quan hệ có cùng vế trái. Nghĩa là kết hợp $R_1(X, Z)$ có $X \rightarrow Z$ với $R_k(X, W)$ có $X \rightarrow W$ và tạo quan hệ $R_{jk}(X, ZW)$ có $X \rightarrow ZW$. Tiếp tục bước 5.

(5) Nếu tất cả các phần tử của khóa K không xuất hiện như một phần của vế trái trong bất cứ quan hệ nào được tạo ở bước 4 thì tạo quan hệ mới có thuộc tính là khóa K. Tiếp tục với bước 6.

(6) Các quan hệ $R_i(X, Y)$ tạo nên phân rã 3NF bảo toàn và bảo toàn phụ thuộc.

Sau đây là một ví dụ minh họa cho thuật toán trên.

Cho quan hệ r(X, Y, Z, W, Q) và $F_c = \{X \rightarrow Y, XZ \rightarrow W, YW \rightarrow Q\}$. Hãy tìm một phân rã đạt 3NF dùng thuật toán trên.
Trước khi áp dụng thuật toán trên ta cần tìm một khóa dự tuyển của quan hệ cho trước. Trong trường hợp này, khóa dự tuyển là khóa phức XZ. Ta hãy chứng minh điều này.
- XZ là khóa

(1) $XZ \rightarrow X$ (tiên đề Reflexivity)

(2) $XZ \rightarrow Z$ (tiên đề Reflexivity)

(3) Từ $X \rightarrow Y$ (cho sẵn) và $XZ \rightarrow X$ (từ 1) nên $XZ \rightarrow Y$ (tiên đề Transitivity)

(4) $XZ \rightarrow W$ (cho sẵn)

(5) Từ $X \rightarrow Y$ (cho sẵn), $XZ \rightarrow W$ (cho sẵn) và $YW \rightarrow Q$ nên ta có $XXZ \rightarrow Q$ (tiên đề Pseudotransitivity) hay $XZ \rightarrow Q$

Vì XZ xác định hàm tất cả các thuộc tính của quan hệ nên nó là khóa của quan hệ. Áp dụng thuật toán trên ta có:

(1) Vì không có thuộc tính nào trong lược đồ của r(X, Y, Z, W, Q) không xuất hiện ở cả hai vế của tất cả các phụ thuộc hàm của F_c, nên ta chuyển qua bước 2.

(2) Không có phụ thuộc hàm đơn nào có tất cả các thuộc tính của quan hệ cho trước.

(3) Xét $X \to Y$ ta tạo quan hệ $R_1(X, Y)$
Xét $XZ \to W$ ta tạo quan hệ $R_2(X, Z, W)$
Xét $YW \to Q$ ta tạo quan hệ $R_3(Y, W, Q)$
(4) Không có quan hệ nào có chung vế trái của các phụ thuộc hàm, do đó ta chuyển qua bước 5.
(5) Các thuộc tính của khóa (XZ) xuất hiện ở vế trái của phụ thuộc hàm ($XZ \to Z$) tương ứng của quan hệ $R_{XZW}(X, Z, W)$, do đó không cần tạo quan hệ khác có lược đồ là khóa dự tuyển XZ.

Phân rã $\rho = \{R_1(X, Y), R_2(X, Z, W), R_3(Y, W, Q)\}$ là bảo toàn. Ta có thể chứng minh điều này bằng cách áp dụng thuật toán Kết bảo toàn. Bảng cho các bước 1 và 2 như sau:

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}
R_2	a_1	b_{22}	a_3	a_4	b_{25}
R_3	b_{31}	a_2	b_{33}	a_4	a_5

Xét các phụ thuộc hàm của $F_c = \{X \to Y, XZ \to W, YW \to Q\}$ và áp dụng bước 3 ta sẽ có bảng sau:

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$Q(A_5)$
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}
R_2	a_1	a_2	a_3	a_4	a_5
R_3	b_{31}	a_2	b_{33}	a_4	a_5

Vì có một dòng (R_2) chứa toàn a nên phân rã là bảo toàn. Bạn hãy tự chứng minh tiếp phần bảo toàn phụ thuộc.

16. Thuật toán sau nhận hai giá trị đầu vào. Một là quan hệ r(R). Hai là phủ không dư thừa F' của tập phụ thuộc hàm F thỏa trên r(R). Đầu ra của thuật toán là một phân rã bảo toàn đạt BNCF $\rho = (R_1, R_2, R_3, \ldots, R_n)$. Kết quả của phân rã có thể hoặc không thỏa tất cả phụ thuộc hàm thỏa bởi quan hệ gốc. Lưu ý là thuật toán chỉ bảo đảm phân rã là bảo toàn chứ không bảo đảm bảo toàn phụ thuộc của quan hệ gốc. Bạn phải nhận thức rằng thứ tự chọn các phụ thuộc hàm ở bước 2 của thuật toán có ảnh hưởng đến kết quả phân rã.

(1) Khởi tạo một quan hệ S_{in} là quan hệ r(R) cho trước. Nghĩa là S_{in} có lược đồ như lược đồ R của quan hệ r.
Khởi tạo một tập quan hệ S_{out} là tập rỗng, nghĩa là $S_{out} = \{\}$.
Khởi tạo một biến nguyên j bằng không, nghĩa là j = 0.
(2) Tìm phụ thuộc hàm không tầm thường $X \to Y$ trong F' thỏa X không phải là siêu khóa của S_{in}. Có hai kết quả có thể nhận được.
(2-a) Nếu tìm thấy phụ thuộc hàm như vậy thì:
Tăng j, nghĩa là j = j + 1
Tạo quan hệ $R_j(\underline{X}, Y)$ trong đó X là khóa của quan hệ. Quan hệ R_j thỏa $X \to Y$.
Đưa quan hệ $R_j(\underline{X}, Y)$ vào S_{out}, nghĩa là $S_{out} = S_{out} \cup R_j$.
Loại vế phải của $X \to Y$ khỏi S_{in}, nghĩa là $S_{in} = S_{in} - \{Y\}$.
Loại $X \to Y$ khỏi F'.
Nếu còn thuộc tính trong S_{in} và còn phụ thuộc hàm trong F' thì lặp lại bước 2, còn không chuyển qua bước 3.
(2-b) Nếu không tìm thấy phụ thuộc hàm như thế thì chuyển qua bước 3. Chú ý là điều này sẽ xảy ra khi có phụ thuộc hàm $X \to Y$ trong F' mà $XY \not\subset S_{in}$.
(3) Nếu không còn phụ thuộc hàm nào trong F' và không còn thuộc tính nào trong S_{in}, thì các quan hệ của S_{out} chính là phân rã đạt BCNF của quan hệ r(R) đã cho vừa bảo toàn vừa chứa các phụ thuộc hàm giống như của quan hệ gốc. Nếu không còn thuộc tính trong S_{in} nhưng còn các phụ thuộc hàm trong F' thì phân rã là bảo toàn nhưng không bảo toàn phụ thuộc của quan hệ gốc. Phân rã cuối cùng là $S_{out} \cup S_{in}$.

Xét quan hệ r(X, Y, W, Z, P, Q) và $F' = \{XY \to W, XW \to P, PQ \to Z, XY \to Q\}$.

(1) Khởi tạo quan hệ S_{in} như r(R), $S_{in} = \{X, Y, W, Z, P, Q\}$
Khởi tạo một tập quan hệ S_{out} là tập rỗng, nghĩa là $S_{out} = \{\}$
Khởi tạo một biến nguyên j bằng không, nghĩa là j = 0.
(2-a) (lần 1) Xét phụ thuộc hàm không tầm thường $XY \to W$. Phụ thuộc hàm này không phải là siêu khóa.
Tăng j, nghĩa là j = 1.
Tạo quan hệ $R_1(\underline{XY}, W)$ trong đó XY là khóa của quan hệ. Quan hệ R_1 thỏa $XY \to W$.
Đưa quan hệ $R_1(\underline{XY}, W)$ vào S_{out}, $S_{out} = S_{out} \cup R_1 = R_1(\underline{XY}, W)$
Loại vế phải của $XY \to W$ khỏi S_{in}, nghĩa là $S_{in} = S_{in} - \{W\} = \{X, Y, Z, P, Q\}$
Loại $XY \to W$ khỏi F', nghĩa là $F' = \{XW \to P, PQ \to Z, XY \to Q\}$
(2-a) (lần 2) Xét phụ thuộc hàm không tầm thường $XY \to Q$. Phụ thuộc hàm này không phải là siêu khóa.
Tăng j, nghĩa là j = 2.
Tạo quan hệ $R_2(\underline{XY}, Q)$ trong đó XY là khóa của quan hệ. Quan hệ R_2 thỏa $XY \to Q$.
Đưa quan hệ $R_2(XY, Q)$ vào S_{out}, $S_{out} = S_{out} \cup R_2 = \{R_1(\underline{XY}, W), R_2(XY, Q)\}$
Loại vế phải của $XY \to Q$ khỏi S_{in}, nghĩa là $S_{in} = S_{in} - \{Q\} = \{X, Y, Z, P\}$
Bỏ $XY \to Q$ khỏi F', nghĩa là $F' = \{XWW \to P, PQ \to Z\}$

(2-b) (lần 1) Không xét được phụ thuộc hàm nào nữa vì S_{in} = {X, Y, Z, P} và F' = {XW → P, PQ → Z}. Chú ý là ta không có {XWP} ⊂ S_{in} = {X, Y, Z, P} mà cũng chẳng có {PQZ} ⊂ S_{in} = {X, Y, Z, P}

(3) (lần 1) Kết quả phân rã là S_{out} = {R_1(<u>XY</u>, W), R_2(<u>XY</u>, Q), R_3(X, Z, W)}

Kết quả phân rã không bảo toàn phụ thuộc vì XW → P và PQ → Z không có trong R_i nào. Tuy nhiên phân rã S_{out} là bảo toàn. Bảng cuối cùng của việc áp dụng thuật toán kết bảo toàn sau khi xét tất cả các phụ thuộc hàm trong F' được trình bày dưới đây. Chú ý là trong đó có một dòng toàn a.

	$X(A_1)$	$Y(A_2)$	$Z(A_3)$	$W(A_4)$	$P(A_5)$	$Q(A_6)$
R_1	a_1	a_2	a_3	a_4	a_5	
R_2	a_1	a_2	a_3	a_4	a_5	a_6
R_3	a_1	a_2	a_3	a_4	a_5	a_6

Tài liệu tham khảo

(Theo năm xuất bản)

[1] Ramon A. Mata-Toledo, Pauline K. Cushman – **Fundamentals of Relational Databases** (Schaum's Outline Series) – MacGraw-Hill, 2000. ISBN 0-07-137869-3

[2] Toby Teorey et al. – **Database design: know it all** (know it all Series) – Morgan Kaufmann, 2009. ISBN 978-0-12-374630-6

[3] Riaz Ahmed – **SQL - The Shortest Route For Beginners** – 2015. ISBN-13: 978-1-505-80560-4

[4] Peter Lalovsky – **Learn SQL Server Intuitively** – zPL Concept, 2016. ISBN 9780995245105