**TRƯỜNG ĐẠI HỌC SÀI GÒN**

A blue circle with text

Description automatically generated**KHOA CÔNG NGHỆ THÔNG TIN**

**TIỂU LUẬN MÔN: CƠ SỞ DỮ LIỆU NÂNG CAO**

**TÊN TIỂU LUẬN:**

**DỊCH CHƯƠNG, TÌM HIỂU CÔNG CỤ POSTGRESQL VÀ SỬ DỤNG POSTGRESQL ĐỂ QUẢN LÝ CỬA HÀNG BÁN GIÀY**

**Giảng Viên Hướng Dẫn: TRẦN SƠN HẢI**

**THÀNH VIÊN NHÓM 2b:**

|  |  |  |
| --- | --- | --- |
| **1** | **Võ Hồ Ngọc Trâm** | **3121410518** |
| **2** | **Nguyễn Lê Bảo Duy** | **3121410120** |
| **3**  **4** | **Nguyễn Lương Thu Trang**  **Hoàng Trọng Tiến** | **3119410440**  **3121410496** |

**HỌC KỲ 1  
 NĂM HỌC 2023-2024**

***Thành phố Hồ Chí Minh, 19 tháng 12 năm 2023***

**MỤC LỤC**

[**LỜI CẢM ƠN 4**](#_Toc153786967)

[**LỜI MỞ ĐẦU 5**](#_Toc153786968)

[**CHƯƠNG 1: DỊCH CHƯƠNG 6**](#_Toc153786969)

[**1.1. Các đặc điểm của một thiết kế quan hệ tốt 6**](#_Toc153786970)

[**1.1.1. Phân hủy 8**](#_Toc153786971)

[**1.1.2. Phân hủy không mất mát 10**](#_Toc153786972)

[**1.1.3. Lý thuyết chuẩn hóa 11**](#_Toc153786973)

[**1.2. Phân hủy bằng cách sử dụng các phụ thuộc chức năng 11**](#_Toc153786974)

[**1.2.1. Quy ước ký hiệu 12**](#_Toc153786975)

[**1.2.2.Chìa khóa và phụ thuộc chức năng 12**](#_Toc153786976)

[**1.2.3. Sự phân hủy không mất mát và phụ thuộc chức năng 15**](#_Toc153786977)

[**1.3. Biểu mẫu thông thường 16**](#_Toc153786978)

[**1.3.1. Boyce – Codd Dạng Bình Thường 16**](#_Toc153786979)

[**1.3.2. Biểu Mẫu Thông Thường Thứ Ba 19**](#_Toc153786980)

[**1.3.3. So sánh BCNF và 3NF 20**](#_Toc153786981)

[**1.3.4. Biểu mẫu Bình thường Cao hơn 21**](#_Toc153786982)

[**1.4. Lý thuyết phụ thuộc chức năng 21**](#_Toc153786983)

[**1.4.1. Kết thúc một tập hợp các phụ thuộc chức năng 22**](#_Toc153786984)

[**1.4.2. Đóng các bộ thuộc tính 24**](#_Toc153786985)

[**1.4.3. Bìa Canonical 26**](#_Toc153786986)

[**1.4.4. Duy trì sự phụ thuộc 30**](#_Toc153786987)

[**1.5. Các thuật toán phân hủy bằng cách sử dụng các phụ thuộc chức năng 31**](#_Toc153786988)

[**1.5.1. Phân hủy BCNF 31**](#_Toc153786989)

[**1.5.2. Phân hủy 3NF 34**](#_Toc153786990)

[**1.5.3. Tính chính xác của thuật toán 3NF 35**](#_Toc153786991)

[**1.6. Phân hủy bằng cách sử dụng phụ thuộc đa giá trị 36**](#_Toc153786992)

[**1.6.1. Phụ thuộc đa giá trị 37**](#_Toc153786993)

[**1.6.2. Biểu Mẫu Thông Thường Thứ Tư 39**](#_Toc153786994)

[**1.6.3. Phân hủy 4NF 39**](#_Toc153786995)

[**1.7. Các biểu mẫu thông thường khác 41**](#_Toc153786996)

[**1.8. Miền nguyên tử và dạng chuẩn thứ nhất 41**](#_Toc153786997)

[**1.9. Quy trình thiết kế cơ sở dữ liệu 43**](#_Toc153786998)

[**1.9.1. Mô hình E-R và Chuẩn hóa 43**](#_Toc153786999)

[**1.9.2. Đặt tên các thuộc tính và mối quan hệ 44**](#_Toc153787000)

[**1.9.3.Không chuẩn hóa cho hiệu suất 45**](#_Toc153787001)

[**1.9.4. Các vấn đề thiết kế khác 45**](#_Toc153787002)

[**1.10. Mô hình hóa dữ liệu thời gian 46**](#_Toc153787003)

[**1.11. Tóm tắt 49**](#_Toc153787004)

[**CHƯƠNG 2: TÌM HIỂU VÀ GIỚI THIỆU CÔNG CỤ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU - PostgreSQL 52**](#_Toc153787005)

[**2.1. Giới thiệu về PostgreSQL 52**](#_Toc153787006)

[**2.2. Cài đặt và cấu hình PostgreSQL 52**](#_Toc153787007)

[**2.4. Các tính năng quan trọng của PostgreSQL 62**](#_Toc153787008)

[**2.5. Đánh giá về công cụ POSTGRESQL 63**](#_Toc153787009)

[**2.6. Các lợi ích của việc sử dụng công cụ POSTGRESQL 64**](#_Toc153787010)

[**2.7. Cách sử dụng công cụ POSTGRESQL 64**](#_Toc153787011)

[**CHƯƠNG 3: THIẾT KẾ CƠ SỞ DỮ LIỆU CHO CỬA HÀNG BÁN GIÀY 71**](#_Toc153787012)

[**3.1. Sơ đồ luồng dữ liệu (DFD) biểu diễn quy trình của cửa hàng bán giày 71**](#_Toc153787013)

[**3.2. Sơ đồ ERD 75**](#_Toc153787014)

[**3.3. Chuyển sơ đồ ERD sang RDM 76**](#_Toc153787015)

[**3.4. Thiết kế cơ sở dữ liệu 77**](#_Toc153787016)

[**3.4.1. Tạo Database 77**](#_Toc153787017)

[**3.4.2. Tạo bảng và các khóa chính, khóa ngoại 77**](#_Toc153787018)

[**3.4.3. Thêm dữ liệu cho từng bảng 82**](#_Toc153787019)

[**TÀI LIỆU THAM KHẢO 88**](#_Toc153787020)

[**KẾT LUẬN 89**](#_Toc153787021)

# LỜI CẢM ƠN

Để có thể hoàn thành tốt với một kết quả tốt đẹp, ngoài sự cố gắng của nhóm. chúng em đã nhận được sự giúp đỡ, quan tâm và hỗ trợ tận tình từ nhiều phía. Đặc biệt là sự giúp đỡ của thầy Trần Sơn Hải đã cung cấp cho chúng em thêm nhiều kiến thức, kỹ năng và học hỏi được nhiều điều qua môn học Cơ sở dữ liệu nâng cao.

Trước hết, nhóm xin gửi đến quý thầy cô Trường Đại học Sài Gòn lời chào, lời chúc sức khỏe và lời cảm ơn sâu sắc nhất. Nhờ có sự quan tâm, dạy dỗ, tận tình, chu đáo của quý thầy cô, nhóm chúng em đã hoàn thành bài báo cáo tiểu luận cuối kỳ môn Cơ sở dữ liệu nâng cao với tên đề tài tiểu luận là: “DỊCH CHƯƠNG, TÌM HIỂU CÔNG CỤ POSTGRESQL VÀ SỬ DỤNG POSTGRESQL ĐỂ QUẢN LÝ CỬA HÀNG BÁN GIÀY”.

Đặc biệt, nhóm xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Trần Sơn Hải đã giúp đỡ, hướng dẫn nhóm chúng em trong quá trình thực hiện đề tài và hoàn thành bài tiểu luận trong khoảng thời gian qua.

Do kiến thức, kinh nghiệm còn hạn chế và giới hạn về thời gian nên trong quá trình thực hiện tiểu luận sẽ không tránh khỏi những sai sót. Nhóm rất mong nhận được những ý kiến đóng góp của quý thầy cô để nhóm chúng em lấy đó làm kinh nghiệm, nâng cao kiến thức, khắc phục những lỗi sai trong quá trình làm việc và có thể áp dụng những điều đó trong quá trình học tập và làm việc của chúng em sau này.

Nhóm xin chân thành cảm ơn!

Thành phố Hồ Chí Minh, ngày 19 tháng 12 năm 2023

# LỜI MỞ ĐẦU

Trong thời đại số hóa mạnh mẽ ngày nay, quản lý thông tin là một phần không thể thiếu trong hoạt động kinh doanh, và cơ sở dữ liệu quan hệ đóng một vai trò quan trọng trong việc này. Trong tiểu luận "DỊCH CHƯƠNG, TÌM HIỂU CÔNG CỤ POSTGRESQL VÀ SỬ DỤNG POSTGRESQL ĐỂ QUẢN LÝ CỬA HÀNG BÁN GIÀY," chúng ta sẽ bắt đầu một hành trình khám phá và nắm bắt tất cả những khía cạnh quan trọng của việc thiết kế cơ sở dữ liệu quan hệ, và cách PostgreSQL, một hệ quản lý cơ sở dữ liệu mạnh mẽ, có thể được áp dụng để quản lý cửa hàng bán giày.

Cơ sở dữ liệu quan hệ không chỉ đơn giản là một bộ sưu tập dữ liệu; nó là một phần quan trọng của việc tổ chức, lưu trữ và truy xuất dữ liệu, đặc biệt là khi bạn đang quản lý một doanh nghiệp. PostgreSQL đã chứng minh mình là một giải pháp cơ sở dữ liệu mã nguồn mở mạnh mẽ với nhiều tính năng cao cấp, và đã trở thành lựa chọn hàng đầu cho nhiều tổ chức và doanh nghiệp trên toàn thế giới.

Và từ những lý do trên, nhóm chúng em quyết định chọn chương về thiết kế cơ sở dữ liệu quan hệ để dịch đồng thời chọn công cụ PostgreSQL để tìm hiểu và thiết kế nên cơ sở dữ liệu để quản lý cửa hàng bán giày và đó cũng tên đề tài của tiểu luận.

# CHƯƠNG 1: DỊCH CHƯƠNG

**THIẾT KẾ CƠ SỞ DỮ LIỆU QUAN HỆ**

Trong chương này, chúng ta xem xét vấn đề thiết kế lược đồ cho cơ sở dữ liệu quan hệ. Nhiều vấn đề khi làm như vậy tương tự như các vấn đề thiết kế mà chúng tôi đã xem xét trong Chương 6 bằng cách sử dụng mô hình E-R.

Nói chung, mục tiêu của thiết kế cơ sở dữ liệu quan hệ là tạo ra một tập hợp các lược đồ quan hệ cho phép chúng ta lưu trữ thông tin mà không cần dư thừa, nhưng cũng cho phép chúng ta truy xuất thông tin dễ dàng. Điều này được thực hiện bằng cách thiết kế các lược đồ ở *dạng bình thường* thích hợp. Để xác định xem một lược đồ quan hệ có phải là một trong những hình thức bình thường mong muốn hay không, chúng ta cần thông tin về doanh nghiệp trong thế giới thực mà chúng ta đang mô hình hóa với cơ sở dữ liệu. Một số thông tin này tồn tại trong sơ đồ E-R được thiết kế tốt, nhưng cũng có thể cần thêm thông tin về doanh nghiệp.

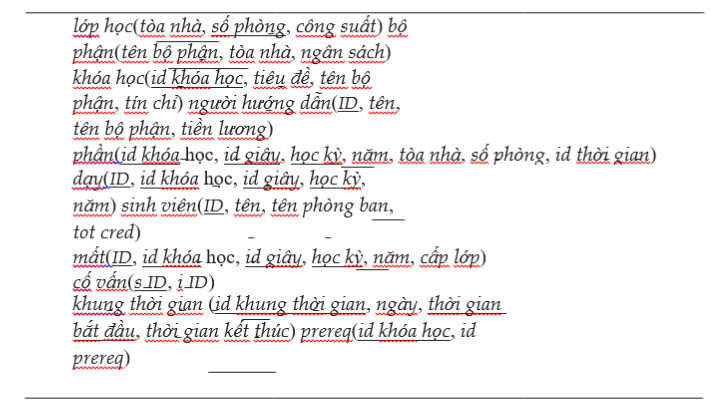
Trong chương này, chúng tôi giới thiệu một cách tiếp cận chính thức để thiết kế cơ sở dữ liệu quan hệ dựa trên khái niệm phụ thuộc chức năng. Sau đó, chúng tôi xác định các biểu mẫu thông thường về các phụ thuộc chức năng và các loại phụ thuộc dữ liệu khác. Tuy nhiên, trước tiên, chúng tôi xem vấn đề thiết kế quan hệ từ quan điểm của các lược đồ bắt nguồn từ một thiết kế quan hệ thực thể nhất định.

## 1.1. Các đặc điểm của một thiết kế quan hệ tốt

Nghiên cứu của chúng tôi về thiết kế mối quan hệ thực thể trong Chương 6 cung cấp một điểm khởi đầu tuyệt vời để tạo ra một thiết kế cơ sở dữ liệu quan hệ. Chúng ta đã thấy trong Phần 6.7 rằng có thể tạo ra một tập hợp các sơ đồ quan hệ trực tiếp từ thiết kế E-R. Sự tốt đẹp (hoặc xấu) của bộ sơ đồ kết quả phụ thuộc vào mức độ tốt của thiết kế E-R ngay từ đầu. Ở phần sau của chương này, chúng ta sẽ nghiên cứu các cách chính xác để đánh giá tính mong muốn của một tập hợp các sơ đồ quan hệ. Tuy nhiên, chúng ta có thể đi một chặng đường dài hướng tới một thiết kế tốt bằng cách sử dụng các khái niệm mà chúng ta đã nghiên cứu. Để dễ tham khảo, chúng tôi lặp lại các lược đồ cho cơ sở dữ liệu trường đại học trong Hình 1.1.

Giả sử rằng chúng ta đã bắt đầu khi thiết kế doanh nghiệp đại học với lược đồ *trong dep*.

*trong dep* (*ID*, *tên*, *tiền lương*, *tên phòng*, *tòa nhà*, *ngân sách*

****

***Hình 1.1****: Lược đồ CSDL cho ví dụ về trường đại học*

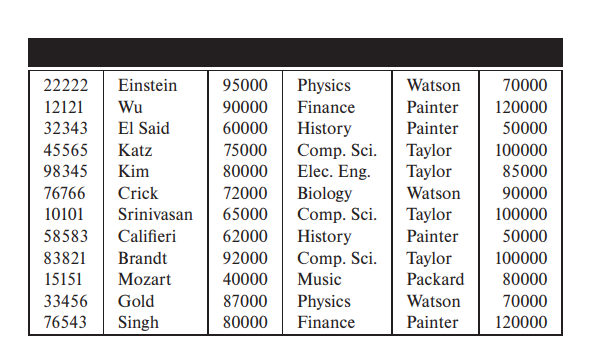
Điều này thể hiện kết quả của việc tham gia tự nhiên vào các mối quan hệ tương ứng với *giảng viên* và *bộ phận*. Điều này có vẻ như là một ý tưởng hay vì một số truy vấn có thể được thể hiện bằng cách sử dụng ít tham gia hơn, cho đến khi chúng tôi suy nghĩ cẩn thận về sự thật về trường đại học đã dẫn đến thiết kế E-R của chúng tôi.

Chúng ta hãy xem xét ví dụ về mối quan hệ *in dep* được hiển thị trong Hình 1.2. Lưu ý rằng chúng tôi phải lặp lại thông tin bộ phận (“tòa nhà” và “ngân sách”) một lần cho mỗi giảng viên trong bộ phận. Ví dụ, thông tin về Comp. Khoa Khoa học (Taylor, 100000) được bao gồm trong bộ giáo viên hướng dẫn Katz, Srinivasan và Brandt.

Điều quan trọng là tất cả các bộ dữ liệu này phải đồng ý về số tiền ngân sách vì nếu không, cơ sở dữ liệu của chúng tôi sẽ không nhất quán. Trong thiết kế ban đầu của chúng tôi sử dụng *người hướng dẫn*  *và người khởi hành*, chúng tôi đã lưu trữ số tiền của mỗi ngân sách chính xác một lần. Điều này cho thấy rằng sử dụng *in dep* là một ý tưởng tồi vì nó lưu trữ số tiền ngân sách một cách dư thừa và có nguy cơ một số người dùng có thể cập nhật số tiền ngân sách trong một bộ nhưng không phải tất cả, và do đó tạo ra sự không nhất quán.

Ngay cả khi chúng tôi quyết định sống với vấn đề dư thừa, vẫn còn một vấn đề khác với sơ đồ *in dep* . Giả sử chúng ta đang tạo ra một bộ phận mới trong sự thống nhất. Trong thiết kế thay thế ở trên, chúng tôi không thể đại diện trực tiếp thông tin liên quan đến một bộ *phận (tên phòng*, *tòa nhà*, *ngân sách*) trừ khi bộ phận đó có ít nhất một giảng viên tại trường đại học. Điều này là do các bộ *trong bảng dep* yêu cầu các giá trị cho *ID*, *tên* và *lương*. Điều này có nghĩa là chúng tôi không thể ghi lại thông tin về bộ phận mới được tạo ra cho đến khi người hướng dẫn đầu tiên được thuê cho bộ phận mới. Trong thiết kế cũ, bộ *phận* lược đồ có thể xử lý việc này, nhưng theo thiết kế sửa đổi, chúng ta sẽ phải tạo một bộ với giá trị null cho việc *xây dựng* và *ngân sách*. Trong một số trường hợp, các giá trị rỗng rất rắc rối, như chúng ta đã thấy trong nghiên cứu của mình. Tuy nhiên, nếu chúng ta quyết định rằng

đây không phải là vấn đề đối với chúng tôi trong trường hợp này, sau đó chúng tôi có thể tiến hành sử dụng thiết kế sửa đổi, mặc dù, như chúng tôi đã lưu ý, chúng tôi vẫn sẽ gặp vấn đề dư thừa.

***Hình 1.2****: Trong quan hệ dep*

### 1.1.1. Phân hủy

Cách duy nhất để tránh vấn đề lặp lại thông tin trong sơ đồ in dep là phân tách nó thành hai sơ đồ (trong trường hợp này, người hướng dẫn và sơ đồ bộ phận). Ở phần sau của chương này, chúng tôi sẽ trình bày các thuật toán để quyết định lược đồ nào phù hợp và lược đồ nào không phù hợp. Nói chung, một lược đồ thể hiện sự lặp lại của sự hình thành có thể phải được phân tách thành một số lược đồ nhỏ hơn.

Không phải tất cả các phân tích sơ đồ đều hữu ích. Hãy xem xét một trường hợp cực đoan trong đó tất cả các lược đồ bao gồm một thuộc tính. Không có mối quan hệ thú vị nào có thể được thể hiện. Bây giờ hãy xem xét một trường hợp ít cực đoan hơn khi chúng tôi chọn phân tích lược đồ nhân viên (Phần 6.8):

employee (ID, name, street, city, salary) vào hai sơ đồ sau:

employee1 (ID, name)

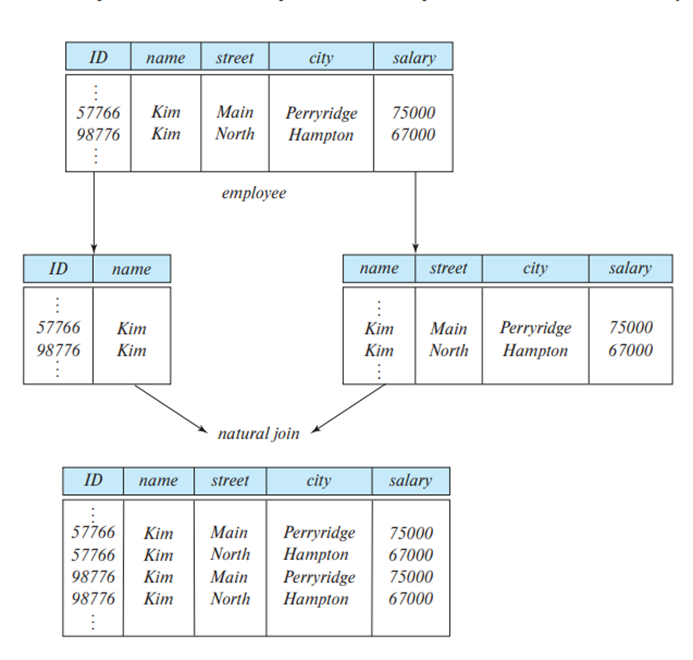
employee2 (name, street, city, salary)

Sự phân rã này phát sinh từ khả năng doanh nghiệp có hai nhân viên cùng tên. Điều này không có khả năng xảy ra trong thực tế, vì nhiều nền văn hóa có một số tên rất phổ biến. Mỗi người sẽ có một mã nhân viên duy nhất, đó là lý do tại sao ID có thể đóng vai trò là khóa chính. Ví dụ, chúng ta hãy giả sử hai nhân viên, cả hai đều tên Kim, làm việc tại trường đại học và có các bộ sau trong mối quan hệ về employee lược đồ trong thiết kế ban đầu:

(57766, Kim, Main, Perryridge, 75000)

(98776, Kim, North, Hampton, 67000

Hình 1.3 cho thấy các bộ này, các bộ kết quả bằng cách sử dụng các lược đồ do sự phân hủy và kết quả nếu chúng ta cố gắng tái tạo các bộ ban đầu chúng ta tham gia tự nhiên. Như chúng ta thấy trong hình, hai bộ dữ liệu gốc xuất hiện trong kết quả cùng với hai bộ dữ liệu mới kết hợp không chính xác các giá trị dữ liệu liên quan đến hai em- ployees tên là Kim. Mặc dù chúng ta có nhiều bộ hơn, nhưng chúng ta thực sự có ít thông tin hơn theo nghĩa sau. Chúng tôi có thể chỉ ra rằng một đường phố, thành phố và mức lương nhất định liên quan đến một người tên Kim, nhưng chúng tôi không thể phân biệt được người họ Kim nào. Do đó, sự phân hủy của chúng tôi không thể đại diện cho một số sự kiện quan trọng nhất định về nhân viên trường đại học



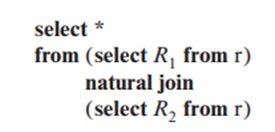
**Hình 1.3** Mất thông tin do phân hủy kém.

Chúng tôi muốn tránh sự phân hủy như vậy. Chúng ta sẽ đề cập đến các vị trí phân hủy như là sự phân hủy mất mát, và ngược lại, đối với những vị trí không phải là sự phân hủy không mất mát.

Đối với phần còn lại của văn bản, chúng tôi sẽ nhấn mạnh rằng tất cả các phân hủy phải là phân hủy không mất mát.

### 1.1.2. Phân hủy không mất mát

Đặt R là sơ đồ quan hệ và để R1 và R2 tạo thành sự phân hủy của R — nghĩa là, xem R, R1 và R2 dưới dạng tập hợp các thuộc tính, R = R1∪ R2. Chúng tôi nói rằng sự phân hủy là một sự phân hủy không mất mát nếu không có sự mất mát thông tin bằng cách thay thế R bằng hai sơ đồ quan hệ R1 và R2. Mất thông tin xảy ra nếu có thể có một thực thể của mối quan hệ r(R) bao gồm thông tin không thể được biểu diễn nếu thay vì thực thể của r(R), chúng ta phải sử dụng các thực thể của r1(R1) và r2(R2). Chính xác hơn, chúng tôi nói rằng sự phân hủy là không mất mát nếu, đối với tất cả các trường hợp cơ sở dữ liệu pháp lý (chúng tôi sẽ chính thức định nghĩa "pháp lý" trong Phần 1.2.2.), mối quan hệ r chứa cùng một tập hợp các bộ dữ liệu như là kết quả của truy vấn SQL sau: 1



Điều này được phát biểu ngắn gọn hơn trong đại số quan hệ như sau:



Nói cách khác, nếu chúng ta chiếu r lên R1 và R2 , và tính toán sự tham gia tự nhiên của các kết quả chiếu, chúng ta sẽ lấy lại chính xác r.

Ngược lại, một sự phân hủy là mất mát nếu khi chúng ta tính toán sự tham gia tự nhiên của các kết quả chiếu, chúng ta nhận được một superset thích hợp của mối quan hệ ban đầu. Điều này được phát biểu ngắn gọn hơn trong đại số quan hệ như sau:



Chúng ta hãy quay lại phân tích lược đồ employeevào *employee1* *và employee2* (Hình 1.3) và trường hợp hai hoặc nhiều nhân viên có cùng tên. Kết quả của *employee1* natural join *employee2* là một superset của *nhân viên* quan hệ ban đầu, nhưng sự phân hủy bị mất vì kết quả tham gia đã mất thông tin về mã định danh nhân viên tương ứng với địa chỉ và mức lương nào.

\_\_\_\_\_\_\_\_\_\_\_\_

1Định nghĩa về lossless được nêu giả định rằng không có thuộc tính nào xuất hiện ở phía bên trái của một phụ thuộc hàm có thể có giá trị null. Điều này được khám phá thêm trong Bài tập 1.10

Có vẻ phản trực giác khi chúng ta có *nhiều bộ dữ liệu hơn* nhưng *ít thông tin hơn*, nhưng đó thực sự là trường hợp. Phiên bản bị phân hủy không thể thể hiện sự *vắng mặt* của mối liên hệ giữa tên và địa chỉ hoặc mức lương, và sự vắng mặt của mối liên hệ thực sự là thông tin.

### 1.1.3. Lý thuyết chuẩn hóa

Bây giờ chúng ta đang ở vị trí để xác định một phương pháp luận chung để tạo ra một tập hợp các sơ đồ mà mỗi sơ đồ đều ở dạng "tốt"; nghĩa là, không bị vấn đề lặp lại thông tin.

Phương pháp thiết kế cơ sở dữ liệu quan hệ là sử dụng một quá trình thường được gọi là **chuẩn hóa**. Mục tiêu là tạo ra một tập hợp các lược đồ quan hệ cho phép chúng ta lưu trữ thông tin mà không cần dư thừa không cần thiết, đồng thời cũng cho phép chúng ta truy xuất thông tin dễ dàng. Cách tiếp cận là:

- Quyết định xem một sơ đồ quan hệ nhất định có ở "dạng tốt" hay không. Có một số hình thức khác nhau (được gọi là *hình thức bình thường*), mà chúng tôi đề cập trong Phần 1.3.

- Nếu một lược đồ quan hệ nhất định không ở dạng "tốt", thì chúng tôi phân hủy nó thành một số lược đồ quan hệ nhỏ hơn, mỗi lược đồ trong số đó ở dạng bình thường thích hợp. Sự phân hủy phải là một sự phân hủy không mất mát.

Để xác định xem một lược đồ quan hệ có phải là một trong những hình thức bình thường mong muốn hay không, chúng ta cần thêm thông tin về doanh nghiệp trong thế giới thực mà chúng ta đang mô hình hóa với cơ sở dữ liệu. Cách tiếp cận phổ biến nhất là sử dụng **các phụ thuộc chức năng**, mà chúng tôi đề cập trong Phần 1.2.

## 1.2. Phân hủy bằng cách sử dụng các phụ thuộc chức năng

Một cơ sở dữ liệu mô hình hóa một tập hợp các thực thể và mối quan hệ trong thế giới thực. Thường có nhiều ràng buộc (quy tắc) đối với dữ liệu trong thế giới thực. Ví dụ: một số ràng buộc dự kiến sẽ có trong cơ sở dữ liệu của trường đại học là:

1. Sinh viên và giảng viên được xác định duy nhất bằng giấy tờ tùy thân của họ.
2. Mỗi học sinh và giáo viên hướng dẫn chỉ có một tên.
3. Mỗi giảng viên và sinh viên (chủ yếu) chỉ liên kết với một bộ phận.2
4. Mỗi bộ phận chỉ có một giá trị cho ngân sách của mình và chỉ có một giá trị xây dựng liên quan.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Một giảng viên, ở hầu hết các trường đại học thực sự, có thể được liên kết với nhiều bộ phận, ví dụ, thông qua một cuộc hẹn chung hoặc trong trường hợp giảng viên phụ trợ. Tương tự, một sinh viên có thể có hai (hoặc nhiều) chuyên ngành chính hoặc chuyên ngành phụ. Các mô hình lược đồ đại học đơn giản hóa của chúng tôi chỉ dành cho bộ phận chính liên quan

Một trường hợp của một mối quan hệ thỏa mãn tất cả các ràng buộc trong thế giới thực như vậy được gọi là một trường **hợp pháp lý** của mối quan hệ; một trường hợp pháp lý của cơ sở dữ liệu là một trường hợp trong đó tất cả các trường hợp quan hệ là các trường hợp pháp lý.

### 1.2.1. Quy ước ký hiệu

Khi thảo luận về các thuật toán cho thiết kế cơ sở dữ liệu quan hệ, chúng ta sẽ cần nói về các mối quan hệ tùy ý và sơ đồ của chúng, thay vì chỉ nói về các ví dụ. Nhắc lại phần giới thiệu về mô hình quan hệ trong Chương 2, chúng tôi tóm tắt ký hiệu của chúng tôi ở đây.

- Nói chung, chúng tôi sử dụng các chữ cái Hy Lạp cho các tập hợp thuộc tính (ví dụ: α). Chúng tôi sử dụng một chữ cái La Mã viết hoa để chỉ một lược đồ quan hệ. Chúng tôi sử dụng ký hiệu r(R) để chỉ ra rằng lược đồ R là cho mối quan hệ r.

Sơ đồ quan hệ là một tập hợp các thuộc tính, nhưng không phải tất cả các tập hợp thuộc tính đều là sơ đồ. Khi chúng ta sử dụng một chữ cái Hy Lạp viết thường, chúng ta đang đề cập đến một tập hợp các thuộc tính có thể hoặc không thể là một lược đồ. Một chữ cái La Mã được sử dụng khi chúng ta muốn chỉ ra rằng tập hợp các thuộc tính chắc chắn là một lược đồ.

- Khi một tập hợp các thuộc tính là một siêu khóa, chúng ta có thể biểu thị nó bằng K. Một siêu khóa liên quan đến một lược đồ quan hệ cụ thể, vì vậy chúng tôi sử dụng thuật ngữ "K là một siêu khóa cho R".

- Chúng tôi sử dụng tên viết thường cho các mối quan hệ. Trong các ví dụ của chúng tôi, những tên này được dự định là thực tế (ví dụ: người hướng dẫn), trong khi trong các định nghĩa và thuật toán của chúng tôi, chúng tôi sử dụng các chữ cái đơn lẻ, như r.

- Do đó, ký hiệu r(R) đề cập đến mối quan hệ r với giản đồ R. Khi chúng ta viết r(R), do đó chúng ta đề cập đến cả mối quan hệ và lược đồ của nó.

- Một quan hệ, có một giá trị cụ thể tại bất kỳ thời điểm nào; chúng tôi gọi đó là một thực thể và sử dụng thuật ngữ "thực thể của r." Khi rõ ràng là chúng ta đang nói về một trường hợp, chúng ta có thể chỉ đơn giản sử dụng tên mối quan hệ (ví dụ: r).

Để đơn giản, chúng tôi giả định rằng tên thuộc tính chỉ có một ý nghĩa trong lược đồ cơ sở dữ liệu.

### 1.2.2.Chìa khóa và phụ thuộc chức năng

Một số loại ràng buộc trong thế giới thực được sử dụng phổ biến nhất có thể được biểu diễn chính thức dưới dạng các khóa (siêu khóa, khóa ứng cử viên và khóa chính) hoặc dưới dạng các depen- dencies chức năng, mà chúng tôi định nghĩa dưới đây

Trong Phần 2.3, chúng tôi đã định nghĩa khái niệm *siêu khóa* là một tập hợp một hoặc nhiều thuộc tính, được tập hợp lại, cho phép chúng tôi xác định duy nhất một bộ trong mối quan hệ. Chúng tôi trình bày lại định nghĩa đó ở đây như sau: Cho *r*(*R*), tập hợp con *K* của *R* là **siêu khóa** của *r*(*R*) nếu, trong bất kỳ trường hợp pháp lý nào của *r*(*R*), đối với tất cả các cặp *t*1 và *t*2 của bộ trong trường hợp *r* nếu *t*1 ≠ t 2, thì *t*1[*K*] ≠ *t*2[*K*]. Nghĩa là, không có hai bộ trong bất kỳ trường hợp pháp lý nào về mối quan hệ *r*(*R*) có thể có cùng giá trị trên tập thuộc tính K.3Nếu không có hai bộ trong r có cùng giá trị trên K, sau K -giá trị xác định duy nhất một bộ trong r.

Trong khi đó, một siêu khóa là một tập hợp các thuộc tính xác định duy nhất toàn bộ một bộ, một

phụ thuộc chức năng cho phép chúng ta thể hiện các ràng buộc xác định duy nhất các giá trị của các thuộc tính nhất định. Xem xét sơ đồ quan hệ r(R), và để α R và β R.

- Cho một thể hiện của r(R), chúng ta nói rằng thể hiện thỏa mãn độ lệch chức năng α → β nếu đối với tất cả các cặp bộ t1 và t2 trong thể hiện sao cho t1[α] = t2[α], nó cũng là trường hợp t1[β] = t2[β].

- Chúng tôi nói rằng sự phụ thuộc chức năng α → β giữ trên lược đồ r(R) nếu, mọi trường hợp pháp lý của r(R) thỏa mãn sự phụ thuộc chức năng.

Sử dụng ký hiệu phụ thuộc chức năng, chúng ta nói rằng K là một siêu khóa cho r(R) nếu phụ thuộc chức năng K → R giữ trên r(R). Nói cách khác, K là một siêu khóa nếu, đối với mọi trường hợp pháp lý của r(R), đối với mọi cặp bộ t1 và t2 từ trường hợp này, bất cứ khi nào t1[K] = t2[K], cũng là trường hợp t1[R] = t2[R] (tức là t1 = t2).4

Các phụ thuộc chức năng cho phép chúng ta thể hiện các ràng buộc mà chúng ta không thể thể hiện với superkey. Trong Phần 1.1, chúng tôi đã xem xét lược đồ:

trong dep (ID, tên, tiền lương, tên phòng, tòa nhà, ngân sách)

trong đó→ ngân sách tên bộ phận phụ thuộc chức năng giữ vì đối với mỗi bộ phận (được xác định bằng tên bộ phận) có một số tiền ngân sách duy nhất.

Chúng tôi biểu thị thực tế là cặp thuộc tính (ID, tên bộ phận) tạo thành một siêu khóa cho dep bằng văn bản:

ID,→ tên bộ phận, tiền lương, tòa nhà, ngân sách

Chúng ta sẽ sử dụng các phụ thuộc chức năng theo hai cách:

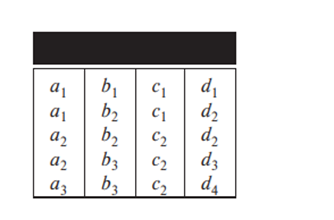
1. Để kiểm tra các trường hợp của các mối quan hệ để xem liệu chúng có đáp ứng một tập hợp F nhất định của các phụ thuộc chức năng hay không.
2. Để xác định các ràng buộc về tập hợp các quan hệ pháp lý. Do đó, chúng ta sẽ chỉ quan tâm đến bản thân mình với những trường hợp quan hệ đáp ứng một tập hợp các de- chức năng nhất định. Nếu chúng ta muốn hạn chế bản thân với các mối quan hệ trên lược đồ r(R) đáp ứng một tập F của các phụ thuộc chức năng, chúng ta nói rằng F giữ trên r(R).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Trong cuộc thảo luận về các phụ thuộc chức năng, chúng tôi sử dụng đẳng thức (=) theo nghĩa toán học thông thường, không phải theo nghĩa logic ba giá trị của SQL. Nói cách khác, khi thảo luận về các phụ thuộc chức năng, chúng tôi giả định không có giá trị null.

4.Lưu ý rằng ở đây chúng ta giả định rằng các mối quan hệ là các tập hợp. SQL xử lý nhiều tập hợp và khai báo **khóa chính** trong SQL cho một

tập hợp các thuộc tính *K* không chỉ yêu cầu *t*1 = t 2 nếu *t*1[*K*] = *t*2[*K*], mà còn không có bộ trùng lặp. SQL cũng yêu cầu các thuộc tính trong tập hợp *K* không thể được gán một giá trị *null* .



**Hình 1.4** Ví dụ mẫu của mối quan hệ *r*.

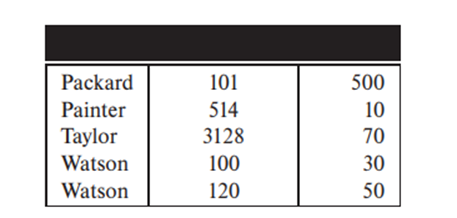
Chúng ta hãy xem xét ví dụ về mối quan hệ *r* của Hình 1.4, để xem những phụ thuộc chức năng nào được thỏa mãn. Quan sát xem *A* → *C* có hài lòng không. Có hai bộ có giá trị *A*  *là*1. Các bộ này có cùng giá trị *C* - cụ thể là *c*1. Tương tự, hai bộ có giá trị *A*  *là*2 có cùng giá trị *C*, *c*2. Không có cặp bộ khác biệt nào có cùng giá trị *A*. Tuy nhiên, phụ thuộc chức năng *C* → *A* không được thỏa mãn. Để thấy rằng nó không phải là, hãy xem xét các bộ *t1*= (*a2*, b3, *c2*, d3) và *t2* = (a3, b3, c2, d4). Hai bộ này có cùng giá trị *C*, *c*2, nhưng chúng có giá trị *A* khác nhau, lần lượt *là* 2 và 3. Do đó, chúng tôi đã tìm thấy một cặp bộ *t1* và *t2* sao cho t1[*C*] = *t2*[*C*], nhưng *t1*[*A*] ≠ *t2*[*A*].

Một số phụ thuộc chức năng được cho là **tầm thường** vì chúng được tất cả mọi người thỏa mãn

quan hệ Ví dụ,→ *A* đượcthỏa mãn bởi tất cả các mối quan hệ liên quan đến thuộc tính *A*. Đọc định nghĩa về sự phụ thuộc chức năng theo nghĩa đen, chúng ta thấy rằng, đối với tất cả các bộ t1 và *t2* sao cho t1 [*A*] = *t2*[*A*], đó là trường hợp *t1*[*A*] = *t2*[*A*]. Tương tự, *AB* → *A* được thỏa mãn bởi tất cả các mối quan hệ liên quan đến thuộc tính *A*. Nói chung, một sự phụ thuộc chức năng của biểu mẫu

αβ → là không đáng kể nếu βα.

Điều quan trọng là phải nhận ra rằng một trường hợp của một mối quan hệ có thể đáp ứng một số chức năng các phụ thuộc không bắt buộc phải giữ lược đồ của mối quan hệ. Trong ví dụ về mối quan hệ *lớp học* của Hình 1.5, chúng ta thấy rằng *số* → *lượng* phòng được thỏa mãn. Tuy nhiên, chúng tôi tin rằng, trong thế giới thực, hai phòng học trong các tòa nhà khác nhau có thể có cùng số phòng nhưng với sức chứa phòng khác nhau. Do đó, vào một thời điểm nào đó, có thể có một ví dụ về mối quan hệ trong *lớp học* trong đó→ *sức chứa*  *số phòng* không được thỏa mãn. Vì vậy, chúng tôi sẽ không bao gồm *số* → *lượng* phòng trong bộ

****

**Hình 1.5** Một ví dụ về mối quan hệ lớp học .

các phụ thuộc chức năng giữ trên lược đồ cho mối quan hệ *lớp học* . Tuy nhiên, chúng tôi hy vọng việc *xây dựng* sự phụ thuộc chức năng, *số lượng phòng* → có thể phù hợp với sơ đồ *lớp học*.

Bởi vì chúng ta giả định rằng các tên thuộc tính chỉ có một ý nghĩa trong lược đồ cơ sở dữ liệu, nếu chúng ta nói rằng một phụ thuộc chức năng α → β giữ như một ràng buộc đối với cơ sở dữ liệu, sau đó đối với bất kỳ lược đồ *R* nào sao cho α R và β *R*, α → β phải giữ.

Cho rằng một tập hợp các phụ thuộc chức năng *F* giữ trên một mối quan hệ *r*(*R*), nó có thể suy ra rằng một số phụ thuộc chức năng khác cũng phải giữ mối quan hệ. Ví dụ, với một lược đồ *r*(*A*, *B*, *C*), nếu các phụ thuộc chức năng *A* → *B* và *B* → *C* giữ *r*, chúng ta có thể suy ra sự phụ thuộc chức năng *A* → *C* cũng phải giữ *r*. Điều này là do, với bất kỳ giá trị nào của *A*, chỉ có thể có một giá trị tương ứng cho *B* và đối với giá trị đó của *B*, chỉ có thể có một giá trị tương ứng cho *C*. Chúng tôi nghiên cứu trong Phần 1.4.1, cách đưa ra những suy luận như vậy.

Chúng ta sẽ sử dụng ký hiệu *F* + để biểu thị sự **đóng** của tập hợp *F*, nghĩa là tập hợp tất cả các phụ thuộc chức năng có thể được suy ra cho tập hợp *F*. *F* + chứa tất cả các phụ thuộc chức năng trong *F* .

### 1.2.3. Sự phân hủy không mất mát và phụ thuộc chức năng

Chúng ta có thể sử dụng các phụ thuộc chức năng để hiển thị khi một số phân hủy nhất định không mất mát. Cho R, R1, R2, và F như trên. R1 và R2 tạo thành sự phân hủy không mất R nếu ít nhất một trong các phụ thuộc chức năng sau nằm

• R1 ∩ R2 → R1

• R1 ∩ R2 → R2

Nói cách khác, nếu R2 tạo thành một siêu khóa cho R1 hoặc R2, sự phân hủy của Rlà sự phân hủy không mất mát. Chúng ta có thể sử dụng đóng thuộc tính để kiểm tra hiệu quả các superkey, như chúng ta đã thấy trước đó.

Để minh họa điều này, hãy xem xét lược đồ

in dep (ID, name, salary, dept name, building, budget)

mà chúng tôi đã phân tách trong Mục 1.1 thành các sơ đồ giảng viên và bộ phận:

instructor (ID, name, dept name, salary)

department (dept name, building, budget)

Hãy xem xét giao điểm của hai sơ đồ này, đó là tên dept. Chúng ta thấy rằng có thể gây ra tên bộ phận→ tên bộ phận, tòa nhà, ngân sách, quy tắc phân hủy không mất mát là sat- isfied.

Đối với trường hợp phân hủy chung của một lược đồ thành nhiều lược đồ cùng một lúc, việc kiểm tra sự phân hủy không mất mát sẽ phức tạp hơn. Xem phần Đọc thêm ở cuối chương này để biết tài liệu tham khảo về chủ đề này.

Mặc dù thử nghiệm phân hủy nhị phân rõ ràng là một điều kiện thích hợp cho sự phân hủy không mất mát, nhưng đó là một điều kiện cần thiết chỉ khi tất cả các ràng buộc là depen- dencies chức năng. Chúng ta sẽ thấy các loại ràng buộc khác sau này (đặc biệt, một loại ràng buộc được gọi là phụ thuộc đa giá trị được thảo luận trong Phần 1.6.1) có thể đảm bảo rằng một thành phần không bị mất ngay cả khi không có phụ thuộc chức năng.

Giả sử chúng ta phân tách lược đồ quan hệ r(R) thành r1(R1) và r2(R2), trong đó R 1 R 2 → R 1.5 Sau đó, các ràng buộc SQL sau đây phải được áp dụng trên lược đồ bị phân hủy để đảm bảo nội dung của chúng phù hợp với lược đồ ban đầu.

• R2 là khóa chính của r1.

Ràng buộc này thực thi sự phụ thuộc chức năng.

• R2 là khóa ngoại từ R2 tham chiếu r1.

Ràng buộc này đảm bảo rằng mỗi bộ trong r2 có một bộ phù hợp trong r1, mà không có nó sẽ không xuất hiện trong tham gia tự nhiên của r1 và r2.

Nếu r1 hoặc r2 bị phân hủy hơn nữa, miễn là sự phân hủy đảm bảo rằng tất cả các thuộc tính trong R1 R 2 là trong một mối quan hệ, ràng buộc khóa chính hoặc khóa ngoại trên r1 hoặc r2 sẽ được kế thừa bởi mối quan hệ đó.

## 1.3. Biểu mẫu thông thường

Như đã nêu trong Phần 1.1.3, có một số dạng thông thường khác nhau được sử dụng trong thiết kế cơ sở dữ liệu quan hệ. Trong phần này, chúng tôi đề cập đến hai trong số những vấn đề phổ biến nhất.

### 1.3.1. Boyce – Codd Dạng Bình Thường

Một trong những dạng chuẩn mong muốn hơn mà chúng ta có thể thu được là dạng chuẩn Boyce-Codd (BCNF). Nó loại bỏ tất cả sự dư thừa có thể được phát hiện dựa trên các depen- dencies chức năng, mặc dù, như chúng ta sẽ thấy trong Phần 1.6, có thể còn lại các loại dư thừa khác.

#### 1.3.1.1. Định nghĩa

Một trong những dạng chuẩn mong muốn hơn mà chúng ta có thể thu được là dạng chuẩn Boyce-Codd (BCNF). Nó loại bỏ tất cả sự dư thừa có thể được phát hiện dựa trên các depen- dencies chức năng, mặc dù, như chúng ta sẽ thấy trong Phần 1.6, có thể còn lại các loại dư thừa khác.

Lược đồ quan hệ R là trong BCNF đối với một tập hợp F của các phụ thuộc chức năng nếu, đối với tất cả các phụ thuộc chức năng trong F + có dạng α → β, trong đó α R và β R, ít nhất một trong các giá trị sau:

• α → β là một phụ thuộc chức năng tầm thường (tức là, β α α).

• α là một siêu khóa cho lược đồ R.

Một thiết kế cơ sở dữ liệu nằm trong BCNF nếu mỗi thành viên của tập hợp các sơ đồ quan hệ liên quan đến thiết kế nằm trong BCNF.

Chúng ta đã thấy trong Phần 1.1 một ví dụ về lược đồ quan hệ không có trong BCNF:

in dep (ID, name, salary, dept name, building, budget)

Phụ thuộc chức năng giữ in dept\_name, nhưng tên bộ phận không phải là một siêu khóa (vì một bộ phận có thể có một số người hướng dẫn khác nhau). Trong phần 1.1 chúng tôi thấy rằng sự phân hủy của dep thành người hướng dẫn và bộ phận là một thiết kế tốt hơn. Lược đồ giảng viên nằm trong BCNF. Tất cả các phụ thuộc chức năng không tầm thường nắm giữ, chẳng hạn như:

ID → name, dept name, salary

bao gồm ID ở phía bên trái của mũi tên và ID là một siêu khóa (thực ra, trong trường hợp này, là khóa chính) cho người hướng dẫn. (Nói cách khác, không có bất kỳ sự kết hợp nào giữa tên, tên bộ phận và tiền lương, mà không có ID, ở phía bên trái.) Do đó, người hướng dẫn nằm trong BCNF.

Tương tự, lược đồ bộ phận nằm trong BCNF vì tất cả các phụ thuộc chức năng không tầm thường, chẳng hạn như:

dept name → building, budget

bao gồm tên bộ phận ở phía bên trái của mũi tên và tên bộ phận là một siêu khóa (và khóa chính) cho bộ phận. Do đó, bộ phận nằm trong BCNF.

Bây giờ chúng tôi đưa ra một quy tắc chung để phân tích các lược đồ không có trong BCNF. Hãy để R là một lược đồ không có trong BCNF. Sau đó, có ít nhất một phụ thuộc chức năng không tầm thường α → β sao cho α không phải là siêu khóa cho R. Chúng tôi thay thế R trong thiết kế của mình bằng hai sơ đồ:

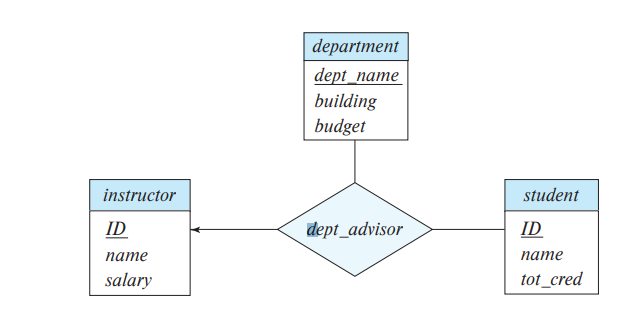
• (αβ)

• (R − (β − α))

Trong trường hợp của dep ở trên, α= dept name, β= {building, budget}, và in dep được thay thế bằng

(α∪β)=(dept name, building,budget)

• (R − (β−α)) = (ID, name, dept name, salary)

**Hình 1.6** Bộ mối quan hệ cố vấn bộ phận

Trong ví dụ này, hóa ra β − α = β. Chúng ta cần nêu quy tắc như chúng ta đã làm để xử lý chính xác các phụ thuộc chức năng có các thuộc tính xuất hiện ở cả hai mặt của mũi tên. Các lý do kỹ thuật cho điều này được đề cập sau trong Phần 1.5.1.

Khi chúng ta phân tách một lược đồ không có trong BCNF, có thể một hoặc nhiều lược đồ kết quả không có trong BCNF. Trong những trường hợp như vậy, cần phải phân hủy thêm, kết quả cuối cùng là một tập hợp các sơ đồ BCNF.

#### 1.3.1.2. BCNF và duy trì sự phụ thuộc

Chúng ta đã thấy một số cách để thể hiện các ràng buộc nhất quán của cơ sở dữ liệu: ràng buộc khóa chính, phụ thuộc chức năng, kiểm tra các ràng buộc, xác nhận và kích hoạt. Kiểm tra các ràng buộc này mỗi khi cơ sở dữ liệu được cập nhật có thể tốn kém và do đó, rất hữu ích để thiết kế cơ sở dữ liệu theo cách mà các ràng buộc có thể được kiểm tra một cách hiệu quả. Đặc biệt, nếu kiểm thử một phụ thuộc chức năng có thể được thực hiện bằng cách xem xét chỉ một mối quan hệ, thì chi phí kiểm thử ràng buộc này là thấp. Chúng ta sẽ thấy rằng, trong một số trường hợp, sự phân hủy thành BCNF có thể ngăn chặn việc kiểm tra hiệu quả của một số thiết bị chức năng nhất định.

Để minh họa điều này, giả sử rằng chúng ta thực hiện một thay đổi nhỏ đối với tổ chức trường đại học của chúng ta. Trong thiết kế của Hình 6.15, một sinh viên có thể chỉ có một cố vấn. Điều này xuất phát từ việc cố vấn tập hợp mối quan hệ là nhiều-đối-một từ sinh viên đến cố vấn. Sự thay đổi "nhỏ" mà chúng ta sẽ thực hiện là một giảng viên chỉ có thể được liên kết với một khởi hành duy nhất, và một sinh viên có thể có nhiều hơn một cố vấn, nhưng không nhiều hơn một từ một bộ phận nhất định.6

Một cách để thực hiện thay đổi này bằng cách sử dụng thiết kế E-R là thay thế tập hợp mối quan hệ advi- sor bằng tập hợp mối quan hệ ternary, cố vấn dept, liên quan đến tập hợp thực thể người hướng dẫn, sinh viên và bộ phận từ nhiều đến một từ cặp {student, instruc- tor} đến bộ phận như thể hiện trong Hình 1.6. Sơ đồ E-R chỉ ra ràng buộc rằng

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6 Sự sắp xếp như vậy có ý nghĩa đối với học sinh có gấp đôi

"một sinh viên có thể có nhiều hơn một cố vấn, nhưng nhiều nhất là một cố vấn tương ứng với một bộ phận nhất định."

Với sơ đồ E-R mới này, sơ đồ cho người hướng dẫn, bộ phận và quan hệ sinh viên không thay đổi. Tuy nhiên, lược đồ bắt nguồn từ mối quan hệ cố vấn phòng ban - bộ tàu bây giờ là:

cố vấn dept (s ID, i ID, tên dept)

Mặc dù không được chỉ định trong sơ đồ E-R, giả sử chúng ta có thêm thông tin rằng "một giảng viên chỉ có thể đóng vai trò là cố vấn cho một bộ phận duy nhất."

Sau đó, các phụ thuộc chức năng sau đây giữ cố vấn phòng ban:

tên bộ phận→ ID i

s ID, dept name → i ID

Sự phụ thuộc chức năng đầu tiên theo yêu cầu của chúng tôi là "một giảng viên chỉ có thể đóng vai trò cố vấn cho một bộ phận." Sự phụ thuộc chức năng thứ hai theo yêu cầu của chúng tôi là "một sinh viên có thể có nhiều nhất một cố vấn cho một bộ phận nhất định."

Lưu ý rằng với thiết kế này, chúng tôi buộc phải lặp lại tên bộ phận một lần cho mỗi lần giảng viên tham gia vào mối quan hệ cố vấn . Chúng tôi thấy rằng cố vấn phòng ban không có trong BCNF vì i ID không phải là một siêu khóa. Theo quy tắc phân hủy BCNF của chúng tôi, chúng tôi nhận được:

(s ID, i ID)

(i ID, tên bộ phận)

Cả hai sơ đồ trên đều là BCNF. (Trên thực tế, bạn có thể xác minh rằng bất kỳ lược đồ nào chỉ có hai thuộc tính đều nằm trong BCNF theo định nghĩa.)

Tuy nhiên, lưu ý rằng trong thiết kế BCNF của chúng tôi, không có lược đồ nào bao gồm tất cả các thuộc tính xuất hiện trong ID phụ thuộc chức năng, tên phòng ban → i ID. Sự phụ thuộc duy nhất có thể được thực thi đối với các mối quan hệ bị phân hủy riêng lẻ là ID → dept name. ID phụ thuộc chức năng, tên phòng ban → i ID chỉ có thể được kiểm tra bằng cách tính toán sự tham gia của các mối quan hệ bị phân hủy.

Bởi vì thiết kế của chúng tôi không cho phép thực thi sự phụ thuộc chức năng này mà không cần tham gia, chúng tôi nói rằng thiết kế của chúng tôi không phải là bảo tồn sự phụ thuộc (chúng tôi cung cấp định nghĩa chính thức về bảo tồn sự phụ thuộc trong Phần 1.4.4). Bởi vì sự phụ thuộc thường được coi là mong muốn, chúng tôi xem xét một hình thức bình thường khác, yếu hơn BCNF, sẽ cho phép chúng tôi duy trì sự phụ thuộc. Hình thức bình thường đó được gọi là hình thức bình thường thứ ba.8

Về mặt kỹ thuật, có thể là một phụ thuộc có các thuộc tính không xuất hiện trong bất kỳ một lược đồ nào vẫn được thực thi ngầm, vì sự hiện diện của các phụ thuộc khác ngụ ý nó một cách logic. Chúng tôi đề cập đến trường hợp đó trong Mục 1.4.4. 8Bạn có thể đã lưu ý rằng chúng tôi đã bỏ qua hình thức bình thường thứ hai. Nó chỉ có ý nghĩa lịch sử và, trong thực tế, một trong những hình thức bình thường thứ ba hoặc BCNF luôn là một lựa chọn tốt hơn. Chúng ta khám phá hình thức bình thường thứ hai trong Bài tập 1.19. Dạng chuẩn đầu tiên liên quan đến các miền thuộc tính, không phải phân hủy. Chúng tôi thảo luận về nó trong

### 1.3.2. Biểu Mẫu Thông Thường Thứ Ba

BCNF yêu cầu tất cả các phụ thuộc không tầm thường phải có dạng α → β, trong đó α là siêu khóa. Dạng pháp tuyến thứ ba (3NF) nới lỏng hạn chế này một chút bằng cách cho phép một số phụ thuộc chức năng không tầm thường nhất định có phía bên trái không phải là siêu khóa. Trước khi chúng tôi xác định 3NF, chúng tôi nhớ lại rằng khóa ứng viên là một siêu khóa tối thiểu - nghĩa là, một siêu khóa không có tập hợp con thích hợp cũng là một siêu khóa.

Lược đồ quan hệ R ở dạng pháp tuyến thứ ba đối với tập hợp F của các phụ thuộc chức năng nếu, đối với tất cả các phụ thuộc chức năng trong F + có dạng α → β, trong đó α R và β R, ít nhất một trong các giá trị sau:

• α → β là một phụ thuộc chức năng tầm thường.

• α là một siêu khóa cho R.

• Mỗi thuộc tính A trong β-α được chứa trong khóa ứng viên cho R.

Lưu ý rằng điều kiện thứ ba ở trên không nói rằng một khóa ứng viên duy nhất phải kết hợp tất cả các thuộc tính trong β− α; mỗi thuộc tính A trong β−α có thể được chứa trong một khóa ứng viên khác .

Hai lựa chọn thay thế đầu tiên giống như hai lựa chọn thay thế trong định nghĩa của BCNF. Giải pháp thay thế thứ ba trong định nghĩa 3NF có vẻ khá không trực quan, và không rõ tại sao nó lại hữu ích. Theo một nghĩa nào đó, nó thể hiện sự nới lỏng tối thiểu các điều kiện BCNF giúp đảm bảo rằng mọi lược đồ đều có sự phân rã bảo tồn phụ thuộc thành 3NF. Mục đích của nó sẽ trở nên rõ ràng hơn sau này, khi chúng ta nghiên cứu phân hủy thành 3NF.

Quan sát thấy rằng bất kỳ lược đồ nào thỏa mãn BCNF cũng thỏa mãn 3NF, vì mỗi phụ thuộc chức năng của nó sẽ thỏa mãn một trong hai lựa chọn thay thế đầu tiên. BCNF có một hình thức bình thường hạn chế hơn so với 3NF.

Định nghĩa của 3NF cho phép các phụ thuộc chức năng nhất định không được phép trong BCNF. Sự phụ thuộc α → β chỉ thỏa mãn lựa chọn thay thế thứ ba của định nghĩa 3NF không được phép trong BCNF nhưng được phép trong 3NF.9

Bây giờ, chúng ta hãy xem xét lại lược đồ cho mối quan hệ cố vấn phòng ban, có các phụ thuộc chức năng sau:

tên bộ phận→ ID i

s ID, dept name → i ID

Trong Phần 1.3.1.2, chúng tôi lập luận rằng sự phụ thuộc chức năng “i ID → dept name” khiến lược đồ cố vấn dept không có trong BCNF. Lưu ý rằng ở đây α = i ID, β = tên dept và β − α = tên dept. Kể từ khi ID phụ thuộc chức năng, tên bộ phận →

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9Những phụ thuộc này là ví dụ về các phụ thuộc chuyển tiếp (xem Bài tập Thực hành 1.18). Định nghĩa ban đầu về

3NF là về các phụ thuộc chuyển tiếp. Định nghĩa chúng tôi sử dụng

i ID giữ cố vấn bộ phận, tên bộ phận thuộc tính được chứa trong khóa ứng viên và do đó, cố vấn bộ phận nằm trong 3NF.

Chúng ta đã thấy sự đánh đổi phải được thực hiện giữa BCNF và 3NF khi không có thiết kế BCNF duy trì sự phụ thuộc. Những đánh đổi này được mô tả chi tiết hơn trong Phần 1.3.3.

### 1.3.3. So sánh BCNF và 3NF

Trong số hai hình thức bình thường cho các lược đồ cơ sở dữ liệu quan hệ, 3NF và BCNF có những lợi thế đối với 3NF ở chỗ chúng ta biết rằng luôn có thể có được một thiết kế 3NF mà không phải hy sinh sự mất mát hoặc bảo tồn phụ thuộc. Tuy nhiên, có những nhược điểm đối với 3NF: Chúng ta có thể phải sử dụng các giá trị null để đại diện cho một số mối quan hệ có ý nghĩa có thể có giữa các mục dữ liệu và có vấn đề lặp lại thông tin.

Mục tiêu của chúng tôi về thiết kế cơ sở dữ liệu với các phụ thuộc chức năng là:

1. BCNF.
2. Không mất mát.
3. Duy trì sự phụ thuộc.

Vì không phải lúc nào cũng có thể đáp ứng cả ba, chúng ta có thể buộc phải lựa chọn giữa

BCNF và bảo tồn sự phụ thuộc với 3NF.

Điều đáng chú ý là SQL không cung cấp cách chỉ định các depen- dencies chức năng, ngoại trừ trường hợp đặc biệt khai báo siêu khóa bằng cách sử dụng khóa chính hoặc các ràng buộc duy nhất. Có thể, mặc dù hơi phức tạp, viết các xác nhận thực thi một phụ thuộc chức năng (xem Bài tập thực hành 1.9); thật không may, hiện tại không có hệ thống cơ sở dữ liệu nào hỗ trợ các xác nhận phức tạp được yêu cầu để thực thi các phụ thuộc chức năng thư viện và các xác nhận sẽ tốn kém để kiểm tra. Do đó, ngay cả khi chúng ta có sự phân hủy duy trì phụ thuộc, nếu chúng ta sử dụng SQL tiêu chuẩn, chúng ta chỉ có thể kiểm tra hiệu quả những phụ thuộc chức năng có phía bên trái là chìa khóa.

Mặc dù kiểm tra các phụ thuộc chức năng có thể liên quan đến tham gia nếu sự phân hủy không phải là bảo tồn phụ thuộc, nếu hệ thống cơ sở dữ liệu hỗ trợ các chế độ xem được vật chất hóa, về nguyên tắc chúng ta có thể giảm chi phí bằng cách lưu trữ kết quả tham gia dưới dạng chế độ xem được vật chất hóa; tuy nhiên, cách tiếp cận này chỉ khả thi nếu hệ thống cơ sở dữ liệu hỗ trợ các biến dạng chính hoặc các ràng buộc duy nhất đối với các chế độ xem được vật chất hóa. Về mặt tiêu cực, có một không gian và thời gian trên cao do chế độ xem được vật chất hóa, nhưng về mặt tích cực, lập trình viên ứng dụng không cần phải lo lắng về việc viết mã để giữ cho dữ liệu dự phòng nhất quán khi cập nhật; đó là công việc của hệ thống cơ sở dữ liệu để duy trì chế độ xem nguyên liệu, nghĩa là giữ cho nó được cập nhật khi cơ sở dữ liệu được cập nhật. (Trong Phần 16.5, chúng tôi phác thảo cách hệ thống cơ sở dữ liệu có thể thực hiện bảo trì chế độ xem được vật chất hóa một cách hiệu quả.) Thật không may, hầu hết các hệ thống cơ sở dữ liệu hiện tại đều hạn chế các ràng buộc đối với các chế độ xem được vật chất hóa hoặc hoàn toàn không hỗ trợ chúng. Ngay cả khi các ràng buộc như vậy được cho phép, vẫn có một yêu cầu bổ sung: cơ sở dữ liệu phải cập nhật chế độ xem và kiểm tra ràng buộc ngay lập tức (như một phần của cùng một giao dịch) khi một mối quan hệ cơ bản được cập nhật. Nếu không, vi phạm ràng buộc có thể được phát hiện tốt sau khi cập nhật đã được hình thành và giao dịch gây ra vi phạm đã được thực hiện.

Tóm lại, ngay cả khi chúng ta không thể có được vị trí giải mã BCNF duy trì sự phụ thuộc, vẫn tốt hơn là chọn BCNF, vì việc kiểm tra các phụ thuộc chức năng khác với các ràng buộc khóa chính là khó khăn trong SQL.

### 1.3.4. Biểu mẫu Bình thường Cao hơn

Sử dụng các phụ thuộc chức năng để phân tách các lược đồ có thể không đủ để tránh lặp lại thông tin không cần thiết trong một số trường hợp nhất định. Hãy xem xét một sự thay đổi nhỏ trong định nghĩa do người hướng dẫn đặt ra, trong đó chúng tôi ghi lại với mỗi người hướng dẫn một bộ tên trẻ em và một bộ số điện thoại cố định có thể được chia sẻ bởi nhiều người. Do đó, số điện thoại và tên con sẽ là các thuộc tính đa giá trị và, theo các quy tắc của chúng tôi để tạo lược đồ từ thiết kế E-R, chúng tôi sẽ có hai lược đồ, một cho mỗi thuộc tính đa giá trị, số điện thoại và tên con:

(CMND, tên trẻ) (CMND, số điện thoại)

Nếu chúng ta kết hợp các sơ đồ này để có được

(CMND, tên trẻ, số điện thoại)

chúng tôi sẽ tìm thấy kết quả trong BCNF vì không có phụ thuộc chức năng không tầm thường nào được giữ. Kết quả là chúng ta có thể nghĩ rằng một sự kết hợp như vậy là một ý tưởng hay. Tuy nhiên, một sự kết hợp như vậy là một ý tưởng tồi, như chúng ta có thể thấy bằng cách xem xét ví dụ về một hướng dẫn với hai trẻ em và hai số điện thoại. Ví dụ, hãy để người hướng dẫn có ID 99999 có hai đứa con tên là "David" và "William" và hai số điện thoại, 512-555-1234 và 512-555-4321. Trong lược đồ kết hợp, chúng ta phải lặp lại các số điện thoại một lần cho mỗi người phụ thuộc:

(99999, David, 512-555-1234)

(99999, David, 512-555-4321)

(99999, William, 512-555-1234)

(99999, William, 512-555-4321)

Nếu chúng ta không lặp lại các số điện thoại, và chúng ta chỉ lưu trữ các số điện thoại đầu tiên và cuối cùng, chúng ta sẽ ghi lại tên người phụ thuộc và số điện thoại, nhưng các bộ kết quả sẽ ngụ ý rằng David tương ứng với 512-555-1234, trong khi William tương ứng với 512-555-4321. Điều này sẽ không chính xác.

Bởi vì các biểu mẫu thông thường dựa trên các phụ thuộc chức năng không đủ khả năng để đối phó với các tình huống như thế này, các phụ thuộc khác và các biểu mẫu thông thường đã được xác định. Chúng tôi đề cập đến những điều này trong Mục 1.6 và Mục 1.1.

## 1.4. Lý thuyết phụ thuộc chức năng

Chúng ta đã thấy trong các ví dụ của mình rằng rất hữu ích khi có thể suy luận một cách có hệ thống về các phụ thuộc chức năng như là một phần của quá trình kiểm tra các lược đồ cho BCNF hoặc 3NF.

### 1.4.1. Kết thúc một tập hợp các phụ thuộc chức năng

Chúng ta sẽ thấy rằng, với một tập F của các phụ thuộc chức năng trên một lược đồ, chúng ta có thể chứng minh rằng một số phụ thuộc chức năng khác cũng giữ trên lược đồ. Chúng tôi nói rằng các phụ thuộc chức năng như vậy được "ngụ ý một cách hợp lý" bởi F. Khi kiểm tra các biểu mẫu thông thường, không cần thiết phải xem xét tập hợp các phụ thuộc chức năng nhất định; thay vào đó, chúng ta cần xem xét tất cả các phụ thuộc chức năng giữ trên lược đồ.

Chính thức hơn, với một lược đồ quan hệ r(R), một phụ thuộc chức năng f trên R được ngụ ý về mặt logic bởi một tập hợp các phụ thuộc chức năng F trên R nếu mọi trường hợp của một quan hệ r(R) thỏa mãn F cũng thỏa mãn f.

Giả sử chúng ta được cung cấp một lược đồ quan hệ r(A, B, C, G, H, I) và tập hợp các phụ thuộc chức năng:

A → B A → C

CG → H CG → I

|||UNTRANSLATED\_CONTENT\_START|||B → H|||UNTRANSLATED\_CONTENT\_END|||

Sự phụ thuộc chức năng:

A → H

được ngụ ý một cách hợp lý. Nghĩa là, chúng ta có thể chỉ ra rằng, bất cứ khi nào một thực thể quan hệ thỏa mãn tập hợp các phụ thuộc chức năng đã cho của chúng ta, A → H cũng phải được thỏa mãn bởi thực thể quan hệ đó. Giả sử rằng t1 và t2 là các bộ sao cho:

t1[A] = t2[A]

Vì chúng ta được cho rằng A → B, nên theo định nghĩa về sự phụ thuộc chức năng thì:

t1[B] = t2[B]

Sau đó, vì chúng ta được cho rằng B → H, nó theo sau từ định nghĩa của chức năng de- pendency rằng:

t1[H] = t2[H ]

Do đó, chúng tôi đã chỉ ra rằng, bất cứ khi nào t1 và t 2 là các bộ sao cho t 1[A] = t2[A], thì phải là t1[H] = t2[H]. Nhưng đó chính xác là định nghĩa của A → H.

Cho F là một tập hợp các phụ thuộc chức năng. Việc đóng F, được biểu thị bằng F +, là tập hợp của tất cả các phụ thuộc chức năng được ngụ ý một cách hợp lý bởi F. Với F, chúng ta có thể tính F + trực tiếp từ định nghĩa chính thức về sự phụ thuộc chức năng. Nếu F lớn, quá trình này sẽ kéo dài và khó khăn. Việc tính toán F + như vậy đòi hỏi các đối số thuộc loại vừa được sử dụng để chỉ ra rằng A → H nằm trong tập hợp các phụ thuộc ví dụ của chúng ta. Các tiên đề, hoặc các quy tắc suy luận, cung cấp một kỹ thuật đơn giản hơn để lý luận về các phụ thuộc chức năng. Trong các quy tắc tiếp theo, chúng tôi sử dụng các chữ cái Hy Lạp (α, β, γ, … ) cho các tập hợp thuộc tính và chữ cái La Mã viết hoa từ đầu bảng chữ cái cho indi-

thuộc tính vidual. Chúng tôi sử dụng αβ để ký hiệu αβ.

Chúng ta có thể sử dụng ba quy tắc sau đây để tìm các phụ thuộc chức năng được ngụ ý một cách hợp lý. Bằng cách áp dụng các quy tắc này nhiều lần, chúng ta có thể tìm thấy tất cả F +, cho F. Tập hợp các quy tắc này được gọi là tiên đề Armstrong để vinh danh người đầu tiên đề xuất nó.

• Quy tắc phản xạ. Nếu α là một tập hợp các thuộc tính và β α, thì α → β giữ.

• Quy tắc tăng cường. Nếu α → β giữ và γ là một tập hợp các thuộc tính, thì γα → γβ

Giữ

• Quy tắc chuyển đổi. Nếu α → β giữ và β → γ giữ, thì α → γ giữ.

Các tiên đề của Armstrong là hợp lý, bởi vì chúng không tạo ra bất kỳ sự phụ thuộc chức năng không chính xác nào. Chúng hoàn chỉnh, bởi vì, đối với một tập hợp F nhất định của các de- pendencies chức năng, chúng cho phép chúng ta tạo ra tất cả F +. Phần Đọc thêm cung cấp các tài liệu tham khảo để chứng minh tính hợp lý và đầy đủ. Mặc dù các tiên đề của Armstrong đã hoàn chỉnh, nhưng thật mệt mỏi khi sử dụng chúng trực tiếp để tính toán F +. Để đơn giản hóa vấn đề hơn nữa, chúng tôi liệt kê các quy tắc bổ sung. Có thể sử dụng các tiên đề của Armstrong để chứng minh rằng các quy tắc này là hợp lý (xem Bài tập Thực hành 1.4, Bài tập Thực hành 1.5 và Bài tập 1.27).

• Quy tắc công đoàn. Nếu α → β giữ và α → γ giữ, thì α → βγ giữ.

• Quy tắc phân hủy. Nếu α → βγ giữ, thì α → β giữ và α → γ giữ.

• Quy tắc Pseudotransitivity. Nếu α → β giữ và γβ → δ giữ, thì αγ → δ giữ.

Chúng ta hãy áp dụng các quy tắc của mình vào ví dụ về lược đồ R = (A, B, C, G, H, I) và tập F của các phụ thuộc chức năng {A → B, A → C, CG → H, CG → I, B → H}. Chúng tôi liệt kê một số thành viên của F + tại đây:

• = aH Vì A → B và B → H giữ, chúng tôi áp dụng quy tắc chuyển tiếp. Quan sát thấy rằng việc sử dụng các tiên đề của Armstrong để chỉ ra rằng A → H nắm giữ dễ dàng hơn nhiều so với việc tranh luận trực tiếp từ các định nghĩa, như chúng ta đã làm trước đó trong phần này.

• CG → HI . Vì CG → H và CG → I, quy tắc hợp nhất ngụ ý rằng CG → HI.

F + = F

áp dụng quy tắc phản xạ/\* Tạo tất cả các phụ thuộc tầm thường \*/

nhắc lại, lặp lại

cho mỗi phụ thuộc chức năng f trong F +

áp dụng quy tắc tăng cường trên f

thêm các phụ thuộc chức năng kết quả vào F +

cho mỗi cặp phụ thuộc chức năng f1 và f2 trong F +

nếu f1 và f2 có thể được kết hợp bằng cách sử dụng transitivity

thêm sự phụ thuộc chức năng kết quả vào F +

cho đến khi F + không thay đổi nữa



**Hình 1.7** Một quy trình để tính F+.

• Agi: Vì A → C và CG → I, quy tắc giả truyền hàm ý rằng

AG → tôi nắm giữ.

Một cách khác để tìm ra AG mà→ tôi nắm giữ như sau: Chúng tôi sử dụng quy tắc augmen- tation trên A → C để suy ra AG → CG. Áp dụng quy tắc chuyển tiếp cho sự phụ thuộc này và CG → I, chúng tôi suy ra AG → I.

Hình 1.7 cho thấy một quy trình thể hiện chính thức cách sử dụng các tiên đề của Armstrong để tính toán F +. Trong quy trình này, khi một phụ thuộc chức năng được thêm vào F +, nó có thể đã có mặt và trong trường hợp đó không có thay đổi nào đối với F +. Chúng ta sẽ thấy một cách tính F + khác trong Phần 1.4.2.

Phía bên trái và bên phải của một phụ thuộc chức năng là cả hai tập hợp con của R. Vì một tập hợp kích thước n có 2n tập hợp con, nên có tổng cộng 2n × 2n = 22n phụ thuộc chức năng có thể có, trong đó n là số thuộc tính trong R. Mỗi lần lặp lại vòng lặp của quy trình, ngoại trừ lần lặp cuối cùng, thêm ít nhất một phụ thuộc chức năng vào F +. Do đó, thủ tục được đảm bảo chấm dứt, mặc dù có thể rất dài.

### 1.4.2. Đóng các bộ thuộc tính

Chúng tôi nói rằng một thuộc tính B được xác định chức năng bởi α nếu α → B. Để kiểm tra xem một tập hợp α có phải là siêu khóa hay không, chúng ta phải đưa ra một thuật toán để tính toán tập hợp các thuộc tính được xác định theo chức năng bởi α. Một cách để làm điều này là tính toán F +, lấy tất cả các phụ thuộc chức năng với α làm phía bên trái và lấy sự kết hợp của phía bên phải của tất cả các phụ thuộc đó. Tuy nhiên, làm như vậy có thể tốn kém, vì F + có thể lớn.

Một thuật toán hiệu quả để tính toán tập hợp các thuộc tính được xác định chức năng bởi α không chỉ hữu ích để kiểm tra xem α có phải là siêu khóa hay không, mà còn cho một số nhiệm vụ khác, như chúng ta sẽ thấy sau trong phần này.

Cho α là một tập hợp các thuộc tính. Chúng tôi gọi tập hợp tất cả các thuộc tính được xác định chức năng bởi α theo một tập hợp F của các phụ thuộc chức năng là đóng α theo F; chúng tôi biểu thị nó bằng α+. Hình 1.8 cho thấy một thuật toán, được viết bằng mã giả, để tính toán α+. Đầu vào là một tập hợp F của các phụ thuộc chức năng và tập hợp α của các thuộc tính. Đầu ra được lưu trữ trong kết quả biến.

Để minh họa cách thuật toán hoạt động, chúng ta sẽ sử dụng nó để tính toán (AG)+ với các phụ thuộc chức năng được xác định trong Phần 1.4.1. Chúng ta bắt đầu với result = AG. Lần đầu tiên chúng tôi thực hiện vòng lặp lặp để kiểm tra từng phụ thuộc chức năng, chúng tôi thấy rằng:

• A → B khiến chúng tôi đưa B vào kết quả. Để thấy được thực tế này, chúng ta quan sát thấy rằng A → B nằm trong F, A & kết quả (là AG), vì vậy kết quả:= kết quả & # 8217; B.

• A → C gây ra kết quả trở thành ABCG.

• CG → H làm cho kết quả trở thành ABCGH .

• CG → I gây ra kết quả trở thành ABCGHI .

Lần thứ hai chúng ta thực hiện vòng lặp lặp lại, không có thuộc tính mới nào được thêm vào kết quả và thuật toán kết thúc.

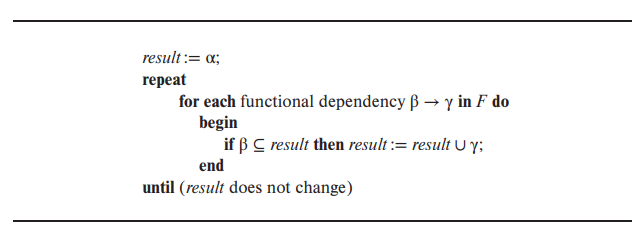
Chúng ta hãy xem tại sao thuật toán trong hình 1.8 là chính xác. Bước đầu tiên là chính xác vì α → α luôn giữ (theo quy tắc phản xạ). Chúng tôi tuyên bố rằng, đối với bất kỳ tập hợp con β nào có kết quả, α → β. Vì chúng ta bắt đầu vòng lặp lại với → kết quả α là đúng, chúng ta có thể thêm γ vào

kết quả chỉ khi kết quả β và β → γ. Nhưng sau đó kết quả → β bằng quy tắc phản xạ, vì vậy α → β

bằng cách chuyển tiếp. Một ứng dụng khác của tính chuyển tiếp cho thấy α → γ (sử dụng α → β và

β → γ). Quy tắc hợp nhất ngụ ý rằng α → kết quả, do đó α chức năng xác định bất kỳ kết quả mới nào được tạo ra trong vòng lặp. Do đó, bất kỳ thuộc tính nào được trả về bởi thuật toán đều nằm trong α+.

Dễ dàng thấy rằng thuật toán tìm thấy tất cả α+. Xem xét một thuộc tính A trong α+ chưa có kết quả tại bất kỳ thời điểm nào trong quá trình thực hiện. Phải có một cách để chứng minh rằng kết quả → A bằng cách sử dụng các tiên đề. Kết quả → A nằm trong chính F (làm cho bằng chứng trở nên tầm thường và đảm bảo A được thêm vào kết quả) hoặc phải có một bước chứng minh bằng cách sử dụng độ chuyển tiếp để hiển thị



**Hình 1.8** Một thuật toán để tính toán α+, đóng α dưới F .

Đối với một số thuộc tính B mà kết quả là → B. Nếu nó xảy ra rằng A = B, sau đó chúng tôi đã chỉ ra rằng A được thêm vào kết quả. Nếu không, B ≠ A được thêm vào. Sau đó, lặp lại lập luận này, chúng ta thấy rằng A cuối cùng phải được thêm vào kết quả.

Nó chỉ ra rằng, trong trường hợp xấu nhất, thuật toán này có thể mất một khoảng thời gian bậc hai trong kích thước của F. Có một thuật toán nhanh hơn (mặc dù phức tạp hơn một chút) chạy theo thời gian tuyến tính với kích thước của F; thuật toán đó được trình bày như một phần của Bài tập thực hành 1.8.

Có một số cách sử dụng thuật toán đóng thuộc tính:

• Để kiểm tra xem α có phải là siêu khóa hay không, chúng tôi tính toán α+ và kiểm tra xem α+ có chứa tất cả các thuộc tính trong

NS =

• Chúng ta có thể kiểm tra xem một phụ thuộc chức năng α → β có giữ được (hay nói cách khác là trong F +) hay không, bằng cách kiểm tra xem β α+. Nghĩa là, chúng ta tính α+ bằng cách sử dụng đóng thuộc tính, và sau đó kiểm tra xem nó có chứa β hay không. Thử nghiệm này đặc biệt hữu ích, như chúng ta sẽ thấy sau này trong

Chương này

• Nó cho chúng ta một cách khác để tính F +: Đối với mỗi γ R, chúng ta tìm thấy sự kết thúc

γ+, và đối với mỗi S, chúng tôi tạo ra sự phụ thuộc chức năng γ → S.

### 1.4.3. Bìa Canonical

Giả sử rằng chúng ta có một tập hợp các phụ thuộc chức năng F trên một lược đồ quan hệ. Khi người dùng thực hiện cập nhật về mối quan hệ, hệ thống cơ sở dữ liệu phải đảm bảo rằng bản cập nhật không vi phạm bất kỳ phụ thuộc chức năng nào, nghĩa là tất cả các thiết lập chức năng trong F đều được thỏa mãn ở trạng thái cơ sở dữ liệu mới.

Hệ thống phải khôi phục bản cập nhật nếu nó vi phạm bất kỳ phụ thuộc chức năng nào trong tập F.

Chúng ta có thể giảm nỗ lực kiểm tra các vi phạm bằng cách kiểm tra một tập hợp các phụ thuộc chức năng đơn giản có cùng kết thúc như tập hợp đã cho. Bất kỳ cơ sở dữ liệu nào đáp ứng tập hợp các phụ thuộc chức năng đơn giản hóa cũng đáp ứng tập hợp ban đầu và ngược lại, vì hai tập hợp có cùng một kết thúc. Tuy nhiên, bộ đơn giản hóa dễ kiểm tra hơn. Chúng ta sẽ thấy cách bộ đơn giản hóa có thể được xây dựng trong giây lát. Đầu tiên, chúng ta cần một số định nghĩa.

Một thuộc tính của một phụ thuộc chức năng được cho là không liên quan nếu chúng ta có thể loại bỏ nó mà không thay đổi việc đóng tập hợp các phụ thuộc chức năng.

• Loại bỏ một thuộc tính từ phía bên trái của một phụ thuộc chức năng có thể làm cho nó trở thành một ràng buộc mạnh mẽ hơn. Ví dụ, nếu chúng ta có AB → C và loại bỏ B, chúng ta nhận được kết quả có thể mạnh hơn A → C. Nó có thể mạnh hơn vì A → C ngụ ý một cách hợp lý AB → → C, nhưng AB C không tự nó ngụ ý một cách hợp lý A → C. Tuy nhiên, tùy thuộc vào tập F phụ thuộc chức năng của chúng ta là gì, chúng ta có thể loại bỏ B khỏi AB → C một cách an toàn. Ví dụ, giả sử rằng bộ

F = {AB → C, A → D, D → C}. Sau đó, chúng ta có thể chỉ ra rằng F một cách hợp lý ngụ ý

A → C, tạo ra B không liên quan trong AB → C.

• Loại bỏ một thuộc tính từ phía bên phải của một phụ thuộc chức năng có thể làm cho nó trở thành một ràng buộc yếu hơn. Ví dụ, nếu chúng ta có AB → CD và loại bỏ C, chúng ta nhận được kết quả có thể yếu hơn AB → D. Nó có thể yếu hơn vì chỉ sử dụng AB → D, chúng ta không còn có thể suy ra AB → C. Tuy nhiên, tùy thuộc vào tập F phụ thuộc chức năng của chúng ta là gì, chúng ta có thể loại bỏ C khỏi AB → CD một cách an toàn. Ví dụ: giả sử rằng F = {AB → CD, A → C}. Sau đó, chúng ta có thể chỉ ra rằng ngay cả sau khi thay thế AB → CD bằng AB → D, chúng ta vẫn có thể suy ra AB → C và do đó AB → CD.

Định nghĩa chính thức về các thuộc tính bên ngoài như sau: Xem xét một tập hợp F của các phụ thuộc chức năng và phụ thuộc chức năng α → β trong F.

• Loại bỏ từ phía bên trái: Thuộc tính A không liên quan trong α nếu A ∈ α và F ngụ ý một cách hợp lý (F − {α → β}) {(α − A) → β}.

• Loại bỏ từ phía bên phải: Thuộc tính A không liên quan trong β nếu A ∈ β và tập hợp các phụ thuộc chức năng (F − {α → β}) {α → (β − A)} một cách hợp lý ngụ ý F .

Hãy cẩn thận với hướng của các hàm ý khi sử dụng định nghĩa về các thuộc tính không liên quan: Nếu bạn đảo ngược tuyên bố, hàm ý sẽ luôn giữ nguyên. Tức là, (F-

{α → β}) {(α − A) → β} luôn ngụ ý một cách logic F, và F cũng luôn ngụ ý một cách logic (F − {α → β}) {α → (β − A)}.

Đây là cách chúng ta có thể kiểm tra một cách hiệu quả nếu một thuộc tính không liên quan. Đặt R là lược đồ quan hệ và đặt F là tập hợp các phụ thuộc chức năng nhất định giữ trên R. Xem xét một thuộc tính A trong một phụ thuộc α → β.

• Nếu A ∈ β, để kiểm tra xem A có không liên quan hay không, hãy xem xét tập hợp

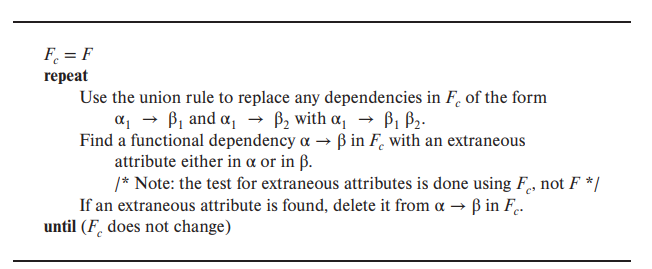
F ′ = (F − {α → β}) {α → (β − A)}

và kiểm tra xem α → A có thể được suy ra từ F ′. Để làm như vậy, hãy tính α+ (đóng

α) dưới F ′; nếu α+ bao gồm A, thì A là không liên quan trong β.

• Nếu A ∈ α, để kiểm tra xem A có ngoại lai hay không, hãy γ = α − {A} và kiểm tra xem γβ có→ thể được suy ra từ F hay không. Để làm như vậy, tính γ+ (đóng γ) dưới F; nếu γ+ bao gồm tất cả các thuộc tính trong β, thì A là không liên quan trong α.

Ví dụ, giả sử F chứa AB → CD, A → E, và E → C. Để kiểm tra xem C có không liên quan trong AB → CD hay không, chúng tôi tính toán đóng thuộc tính của AB theo F ′ = {AB → D, A → E, E → C}. Đóng là abcde, bao gồm CD, vì vậy chúng tôi suy ra rằng C là không liên quan.



**Hình 1.9** Tính toán bìa chính tắc.

Sau khi xác định khái niệm về các thuộc tính bên ngoài, chúng ta có thể giải thích cách chúng ta có thể xây dựng một tập hợp các phụ thuộc chức năng đơn giản hóa tương đương với một tập hợp các phụ thuộc chức năng nhất định.

Một bìa chính tắc Fc cho F là một tập hợp các phụ thuộc sao cho F ngụ ý một cách hợp lý tất cả các phụ thuộc trong Fc và Fc ngụ ý một cách hợp lý tất cả các phụ thuộc trong F. Hơn nữa, Fc phải có các đặc tính sau:

• Không có sự phụ thuộc chức năng nào trong Fc chứa thuộc tính không liên quan.

• Mỗi bên trái của một phụ thuộc chức năng trong Fc là duy nhất. Nghĩa là, không có hai phụ thuộc α1 → β1 và α2 → β2 trong Fc sao cho α1 = α2.

Một bìa chính tắc cho một tập hợp các phụ thuộc chức năng F có thể được tính toán như được ghi chép trong Hình 1.9. Điều quan trọng cần lưu ý là khi kiểm tra xem một thuộc tính có ngoại lệ hay không, kiểm tra sử dụng các phụ thuộc vào giá trị hiện tại của Fc, chứ không phải các depen- dencies trong F. Nếu một phụ thuộc chức năng chỉ chứa một thuộc tính ở phía bên tay phải của nó, ví dụ A → C, và thuộc tính đó được tìm thấy là không liên quan, chúng ta sẽ nhận được một phụ thuộc chức năng với một phía bên tay phải trống rỗng. Các phụ thuộc chức năng như vậy

nên bị xóa.

Vì thuật toán cho phép lựa chọn bất kỳ thuộc tính không liên quan nào, có thể có một số phạm vi chính tắc có thể cho một F nhất định. Bất kỳ Fc nào như vậy đều được chấp nhận như nhau. Bất kỳ nắp chính tắc nào của F, Fc, có thể được chứng minh là có cùng đóng như F; do đó, kiểm tra xem Fc có hài lòng hay không tương đương với kiểm tra xem F có hài lòng hay không.

Tuy nhiên, Fc là tối thiểu theo một nghĩa nhất định - nó không chứa các thuộc tính không liên quan,

và nó kết hợp các phụ thuộc chức năng với cùng một phía bên trái. Kiểm tra rẻ hơn

Fc hơn là để kiểm tra F chính nó.

Bây giờ chúng ta xem xét một ví dụ. Giả sử chúng ta được cung cấp tập F chức năng sau

phụ thuộc vào lược đồ (A, B, C):

A → BC

B → C A → B AB → C

Chúng ta hãy tính toán một bìa kinh điển cho F.

• Có hai phụ thuộc chức năng với cùng một tập hợp các thuộc tính ở phía bên trái của mũi tên:

A → BC

AB

Chúng tôi kết hợp các phụ thuộc chức năng này vào A → BC.

• A không liên quan trong AB → C vì F ngụ ý một cách hợp lý (F − {AB → C}) {B → C}. Khẳng định này là đúng bởi vì B → C đã có trong tập hợp các depen- dencies chức năng của chúng tôi.

• C không liên quan trong A → BC, vì A → BC được ngụ ý một cách hợp lý bởi A → B và B →

Bà nội : chào ông bà, mời ông bà vào nhà uống nước cùng chúng tôi.

Do đó, bìa chính tắc của chúng tôi là:

AB

|||UNTRANSLATED\_CONTENT\_START|||B → C|||UNTRANSLATED\_CONTENT\_END|||

Với một tập hợp F của các phụ thuộc chức năng, có thể là toàn bộ sự suy giảm chức năng trong tập hợp là không liên quan, theo nghĩa là việc giảm nó không làm thay đổi việc đóng F. Chúng ta có thể chỉ ra rằng bìa chính tắc Fc của F không chứa sự phụ thuộc chức năng không liên quan như vậy. Giả sử rằng, ngược lại, có một sự không liên quan như vậy

phụ thuộc chức năng trong Fc. Các thuộc tính bên phải của sự phụ thuộc sau đó sẽ không liên quan, điều này là không thể theo định nghĩa của bìa chính tắc.

Như chúng tôi đã lưu ý trước đó, bìa chính tắc có thể không phải là duy nhất. Ví dụ: hãy xem xét tập hợp các phụ thuộc chức năng F = {A → BC, B → AC và C → AB}. Nếu chúng ta áp dụng thử nghiệm cho các thuộc tính không liên quan đến A → BC, chúng ta thấy rằng cả B và C đều không liên quan theo F. Tuy nhiên, việc xóa cả hai là không chính xác! Thuật toán tìm bìa chính tắc chọn một trong hai và xóa nó. Sau đó,

1. Nếu C bị xóa, chúng tôi nhận được tập hợp F ′ = {A → B, B → AC và C → AB}. Bây giờ, B không phải là không liên quan ở phía bên phải của A → B dưới F ′. Tiếp tục thuật toán, chúng tôi thấy A và B không liên quan ở phía bên phải của C → AB, dẫn đến hai lựa chọn bìa chính tắc:



**Hình 1.10** Kiểm tra bảo tồn phụ thuộc.

Fc = {A → B, B → C, C → A} Fc = {A → B, B → AC, C → B}.

2. Nếu B bị xóa, chúng tôi sẽ nhận được tập hợp {A → C, B → AC và C → AB}. Trường hợp này đối xứng với trường hợp trước, dẫn đến hai lựa chọn bìa chính tắc khác:

Fc = {A → C, C → B, and B → A} Fc = {A → C, B → C, and C → AB}.

Là một bài tập, bạn có thể tìm thêm một bìa kinh điển cho F không?

### 1.4.4. Duy trì sự phụ thuộc

Sử dụng lý thuyết về các phụ thuộc chức năng, có một cách để mô tả việc bảo tồn phụ thuộc đơn giản hơn phương pháp đặc biệt mà chúng tôi đã sử dụng trong Phần 1.3.1.2.

Cho F là một tập hợp các phụ thuộc chức năng trên một lược đồ R, và cho R1, R2, … , Rn là một sự phân hủy của R. Hạn chế của F đến Ri là tập hợp Fi của tất cả các phụ thuộc chức năng trong F + chỉ bao gồm các thuộc tính của Ri. Vì tất cả các phụ thuộc chức năng trong một hạn chế chỉ liên quan đến các thuộc tính của một lược đồ quan hệ, nên có thể kiểm tra một phụ thuộc như vậy để hài lòng bằng cách chỉ kiểm tra một mối quan hệ.

Lưu ý rằng định nghĩa hạn chế sử dụng tất cả các phụ thuộc trong F +, không chỉ các phụ thuộc trong F. Ví dụ, giả sử F = {A → B, B → C}, và chúng ta có sự phân hủy thành AC và AB. Hạn chế của F đến AC bao gồm A → C, vì A → C nằm trong F +, mặc dù nó không nằm trong F.

Tập hợp các hạn chế F1, F2, …, Fn là tập hợp các phụ thuộc có thể được kiểm tra một cách hiệu quả. Bây giờ chúng ta phải hỏi liệu chỉ xét nghiệm các hạn chế có phù hợp hay không. Hãy để F ′ = F 1, F 2, Fn. F ′ là một tập hợp các phụ thuộc chức năng trên lược đồ R, nhưng, nói chung, F ′ ≠ F . Tuy nhiên, ngay cả khi F ′ ≠ F, nó có thể là F ′+ = F +. Nếu sau này là đúng, thì mọi phụ thuộc trong F đều được ngụ ý một cách logic bởi F ′, và nếu chúng ta xác minh rằng F ′ được thỏa mãn, chúng ta đã xác minh rằng F được thỏa mãn. Chúng tôi nói rằng một sự phân hủy có thuộc tính F ′+ = F + là một sự phân hủy bảo tồn phụ thuộc.

Hình 1.10 cho thấy một thuật toán để kiểm tra bảo tồn phụ thuộc. Đầu vào là một tập hợp D = {R1, R2, …, Rn} của các lược đồ quan hệ phân rã và một tập hợp F của các phụ thuộc chức năng. Thuật toán này rất tốn kém vì nó yêu cầu tính toán F +. Thay vào đó,

của việc áp dụng các thuật toán của hình 1.10, chúng tôi xem xét hai lựa chọn thay thế.

Đầu tiên, lưu ý rằng nếu mỗi thành viên của F có thể được kiểm tra trên một trong các mối quan hệ của sự phân hủy, thì sự phân hủy là sự bảo tồn phụ thuộc. Đây là một cách dễ dàng để thể hiện sự bảo tồn sự phụ thuộc; tuy nhiên, nó không phải lúc nào cũng hoạt động. Có những trường hợp, mặc dù sự phân hủy là bảo tồn phụ thuộc, nhưng có một sự phụ thuộc trong F không thể được kiểm tra trong bất kỳ một mối quan hệ nào trong quá trình phân hủy. Do đó, thử nghiệm thay đổi này chỉ có thể được sử dụng như một điều kiện phù hợp dễ kiểm tra; nếu nó không thành công, chúng tôi không thể kết luận rằng sự phân hủy không phải là sự bảo tồn phụ thuộc; thay vào đó chúng tôi sẽ phải áp dụng thử nghiệm chung.

Bây giờ chúng tôi đưa ra một bài kiểm tra thay thế thứ hai để bảo toàn sự phụ thuộc tránh tính toán F +. Chúng tôi giải thích trực giác đằng sau bài kiểm tra sau khi trình bày bài kiểm tra. Thử nghiệm áp dụng quy trình sau đây cho mỗi α → β trong F.

kết quả = α

nhắc lại, lặp lại

cho mỗi Ri trong phân hủy

t = (kết quả, Ri)+ R i kết quả = kết quả , t

cho đến khi (kết quả không thay đổi)

Việc đóng thuộc tính ở đây nằm trong tập hợp các phụ thuộc chức năng F. Nếu kết quả kết hợp tất cả các thuộc tính trong β, thì sự phụ thuộc chức năng α → β được bảo toàn. Thành phần phụ thuộc được bảo toàn khi và chỉ khi quy trình cho thấy rằng tất cả các phụ thuộc trong F được bảo toàn.

Hai ý chính đằng sau bài kiểm tra trước như sau:

• Ý tưởng đầu tiên là kiểm tra từng phụ thuộc chức năng α → β trong F để xem nó có được bảo tồn trong F ′ (trong đó F ′ được xác định trong Hình 1.10) hay không. Để làm như vậy, chúng tôi tính toán đóng α theo F ′; sự phụ thuộc được bảo toàn chính xác khi đóng bao gồm β. Sự phân hủy là sự bảo tồn phụ thuộc nếu (và chỉ nếu) tất cả các phụ thuộc trong F được tìm thấy được bảo tồn.

• Ý tưởng thứ hai là sử dụng một dạng sửa đổi của thuật toán đóng thuộc tính để tính toán đóng theo F ′, mà không thực sự tính toán F ′ đầu tiên. Chúng tôi muốn tránh tính toán F ′ vì tính toán nó khá tốn kém. Lưu ý rằng F ′ là hợp nhất của tất cả Fi, trong đó Fi là giới hạn của F trên Ri. Thuật toán tính toán thuộc tính

đóng cửa của (kết quả, Ri) đối với F, cắt đóng cửa với Ri, và thêm

tập hợp các thuộc tính kết quả; trình tự các bước này tương đương với tính toán

đóng kết quả theo Fi. Lặp lại bước này cho mỗi i bên trong vòng lặp while cho kết quả đóng theo F ′.

Để hiểu tại sao phương pháp đóng thuộc tính được sửa đổi này hoạt động chính xác, chúng tôi lưu ý rằng đối với bất kỳ →γ Ri, γ γ + là phụ thuộc chức năng trong F +, và γ → γ+ Ri là phụ thuộc chức năng trong Fi, sự hạn chế của F + đến Ri. Ngược lại, nếu γ → δ nằm trong Fi, thì δ sẽ là tập hợp con của γ + Ri.

Bài kiểm tra này lấy thời gian đa thức, thay vì thời gian theo cấp số nhân cần thiết để kết hợp F +.

## 1.5. Các thuật toán phân hủy bằng cách sử dụng các phụ thuộc chức năng

Định nghĩa của BCNF có thể được sử dụng trực tiếp để kiểm tra xem một quan hệ có đạt được BCNF hay không. Tuy nhiên, tính toán F+ có thể là một công việc khó khăn. Đầu tiên, chúng tôi sẽ mô tả các bài kiểm tra đơn giản để xác minh xem một quan hệ có đạt được BCNF hay không. Nếu một quan hệ không đạt được BCNF, nó có thể được phân rã để tạo ra các quan hệ đạt được BCNF. Sau đó trong phần này, chúng tôi sẽ mô tả một thuật toán để tạo ra một phân rã không mất mát của một quan hệ, sao cho phân rã đó đạt được BCNF.

### 1.5.1. Phân hủy BCNF

Định nghĩa của BCNF có thể được sử dụng trực tiếp để kiểm tra xem một mối quan hệ có nằm trong BCNF hay không. Tuy nhiên, tính toán F + có thể là một nhiệm vụ tẻ nhạt. Trước tiên, chúng tôi mô tả các bài kiểm tra đơn giản để xác minh xem mối quan hệ có nằm trong BCNF hay không. Nếu một mối quan hệ không nằm trong BCNF, nó có thể được phân hủy để tạo ra các mối quan hệ nằm trong BCNF. Sau đó trong phần này, chúng tôi mô tả một thuật toán để tạo ra sự phân hủy không mất mát của một mối quan hệ, sao cho sự phân hủy nằm trong BCNF.

#### 1.5.1.1. Kiểm tra BCNF

Kiểm tra lược đồ quan hệ R để xem nó có đáp ứng BCNF hay không có thể được đơn giản hóa trong một số trường hợp:

• Để kiểm tra xem sự phụ thuộc không tầm thường α → β có gây ra vi phạm BCNF hay không, hãy tính α+ (đóng thuộc tính của α) và xác minh rằng nó bao gồm tất cả các thuộc tính của R; nghĩa là, nó là một siêu khóa cho R.

• Để kiểm tra xem lược đồ quan hệ R có nằm trong BCNF hay không, nó chỉ kiểm tra các phụ thuộc trong tập hợp F nhất định để biết vi phạm BCNF, thay vì kiểm tra tất cả các phụ thuộc trong F +.

Chúng ta có thể chỉ ra rằng nếu không có phụ thuộc nào trong F gây ra vi phạm BCNF, thì không có phụ thuộc nào trong F + sẽ gây ra vi phạm BCNF.

*result := {R};*

*done := false;*

**while (not** done) **do**

**if** (có một lược đồ Ri kết quả không có trong BCNF)

**then begin**

let α → β là một sự phụ thuộc chức năng không cần thiết mà giữ

trên Ri sao cho α+ không chứa Ri và α∩β=∅ ;

*result := (result − Ri) ∪ (Ri − β) ∪ ( α, β);*

**end**

**else** *done := true;*

**Hình 1.11**  Thuật toán phân hủy BCNF.

Thật không may, quy trình sau không hoạt động khi lược đồ quan hệ được đặt ra. Đó là, nó không cần thiết để sử dụng F khi chúng tôi kiểm tra một lược đồ quan hệ Ri, trong một sự phân hủy của R, vì vi phạm BCNF. Ví dụ, xem xét lược đồ quan hệ (A, B, C, D, E), với các phụ thuộc chức năng F chứa A → B và BC → D. Tư thế này được phân hủy thành (A, B) và (A, C, D, E). Bây giờ, không có sự phụ thuộc trong F chỉ chứa các thuộc tính từ (A, C, D, E), vì vậy chúng ta có thể bị lừa khi nghĩ rằng nó nằm trong BCNF. Trên thực tế, có một phụ thuộc AC → D trong F + (có thể được suy ra bằng cách sử dụng quy tắc giả giao nạp từ hai phụ thuộc trong F) cho thấy rằng (A, C, D, E) không có trong BCNF. Do đó, chúng ta có thể cần một sự phụ thuộc có trong F +, nhưng không có trong F, để chỉ ra rằng một mối quan hệ bị phân hủy không có trong BCNF.

Một bài kiểm tra BCNF thay thế đôi khi dễ dàng hơn tính toán mọi phụ thuộc trong F+. Để kiểm tra xem lược đồ quan hệ Ri trong phân tích R có nằm trong BCNF hay không, chúng tôi áp dụng thử nghiệm này:

• Đối với mỗi tập con α của các thuộc tính trong Ri, hãy kiểm tra xem α+ (đóng thuộc tính của α dưới F) không bao gồm thuộc tính nào của Ri - α, hoặc bao gồm tất cả các thuộc tính của Ri.

Nếu điều kiện bị vi phạm bởi một số thuộc tính α trong Ri, hãy xem xét sự phụ thuộc chức năng sau đây, có thể được chứng minh là có trong F +:

α → (α+ − α) Ri.

Sự phụ thuộc này cho thấy Ri vi phạm BCNF.

#### 1.5.1.2. Thuật toán phân hủy BCNF

Bây giờ chúng ta có thể nêu một phương pháp chung để phân hủy sơ đồ quan hệ để đáp ứng BCNF. Hình 1.11 cho thấy một thuật toán cho nhiệm vụ này. Nếu R không có trong BCNF, chúng ta có thể phân tách R thành một tập hợp các lược đồ BCNF R, R , …, R bằng thuật toán.

Thuật toán sử dụng các phụ thuộc chứng minh vi phạm BCNF để thực hiện phân hủy.

Sự phân hủy mà thuật toán tạo ra không chỉ trong BCNF, mà còn là sự phân hủy không mất mát. Để xem tại sao thuật toán của chúng tôi chỉ tạo ra giải mã lossless- vị trí, chúng tôi lưu ý rằng, khi chúng tôi thay thế lược đồ Ri bằng (Ri -β) và (α, β), sự phụ thuộc α → -β giữ và (Ri -β) - (α, β) = α.

Nếu chúng ta không yêu cầu αβ = ∅, thì các thuộc tính đó trong αβ sẽ không xuất hiện trong lược đồ (Ri − β), và sự phụ thuộc α → β sẽ không còn giữ được nữa.

Dễ dàng thấy rằng sự phân hủy của dep trong Phần 1.3.1 sẽ là kết quả của việc áp dụng thuật toán. Bộ phận phụ thuộc chức năng→ xây dựng tên, ngân sách

thỏa mãn điều kiện αββ = ∅ và do đó sẽ được chọn để phân hủy sơ đồ.

Thuật toán phân hủy BCNF mất thời gian theo cấp số nhân với kích thước của lược đồ ban đầu, vì thuật toán để kiểm tra xem một mối quan hệ trong phân hủy thỏa mãn BCNF có thể mất thời gian theo cấp số nhân hay không. Có một thuật toán có thể tính toán sự phân hủy BCNF trong thời gian đa thức; tuy nhiên, thuật toán có thể "bình thường hóa quá mức", nghĩa là phân hủy một mối quan hệ một cách không cần thiết.

Như một ví dụ dài hơn về việc sử dụng thuật toán phân hủy BCNF, giả sử chúng ta có một thiết kế cơ sở dữ liệu sử dụng quan hệ lớp, có lược đồ như hình dưới đây:

*class (course id, title, dept name, credits, sec id, semester, year, building,*

*room number, capacity, time slot id)*

Tập hợp các phụ thuộc chức năng mà chúng ta cần giữ trên lược đồ này là:

*course id → title, dept name, credits*

*building, room number → capacity*

*course id, sec id, semester, year→ building, room number, time slot id*

Khóa ứng viên cho lược đồ này là {course id, sec id, semester, year}.

Chúng ta có thể áp dụng thuật toán trong hình 1.11 cho ví dụ lớp như sau:

• Sự phụ thuộc chức năng:

*course id → title, dept name, credits*

giữ, nhưng id khóa học không phải là một siêu khóa. Do đó, lớp không nằm trong BCNF. Chúng tôi thay thế lớp học

với hai mối quan hệ với các sơ đồ sau:

*course (course id, title, dept name, credits)*

*class-1 (course id, sec id, semester, year, building, room number*

*capacity, time slot id)*

Các phụ thuộc chức năng không tầm thường duy nhất giữ khóa học bao gồm id khóa học ở phía bên trái của mũi tên. Vì id khóa học là một siêu khóa cho khóa học, khóa học nằm trong BCNF.

• Khóa ứng viên cho lớp 1 là *{course id, sec id, semester, year}*. Hiệu suất chức năng:

building, room number → capacity

giữ lớp 1, nhưng {building, room number} không phải là siêu khóa cho lớp 1. Chúng tôi đặt lại mối quan hệ loại 1 hai với các sơ đồ sau:

*classroom (building, room number, capacity)*

*section (course id, sec id, semester, year,*

*building, room number, time slot id)*

Hai sơ đồ này nằm trong BCNF.

Do đó, sự phân hủy của lớp dẫn đến ba sơ đồ quan hệ khóa học, lớp học và phần, mỗi phần đều nằm trong BCNF. Những điều này tương ứng với các lược đồ mà chúng tôi đã sử dụng trong chương này và các chương trước. Bạn có thể xác minh rằng sự phân hủy là không mất mát và duy trì sự phụ thuộc.

### 1.5.2. Phân hủy 3NF

Hình 1.12 cho thấy một thuật toán để tìm ra sự phân rã không phụ thuộc, không mất mát thành 3NF. Tập hợp các phụ thuộc Fc được sử dụng trong thuật toán là một bìa chính tắc cho F. Lưu ý rằng thuật toán xem xét tập hợp các lược đồ Rj, j = 1, 2, … , i; ban đầu

i = 0, và trong trường hợp này, tập hợp trống.

Hãy để chúng tôi áp dụng thuật toán này vào ví dụ của chúng tôi về cố vấn phòng ban từ Phần 1.3.2, nơi chúng tôi đã chỉ ra rằng:

*dept advisor (s ID, i ID, dept name)*

nằm trong 3NF mặc dù không nằm trong BCNF. Thuật toán sử dụng các phụ thuộc chức năng sau trong F:

*f1: i ID → dept name*

*f2: s ID, dept name → i\_ID*

Không có thuộc tính ngoại lai trong bất kỳ phụ thuộc chức năng nào trong F, vì vậy Fc chứa f1 và f2. Thuật toán sau đó tạo ra như R1 lược đồ, (i ID dept name), và như R2 lược đồ (s ID, dept name, i ID). Thuật toán sau đó phát hiện ra rằng R2 chứa một khóa ứng viên, vì vậy không có sơ đồ quan hệ nào được tạo thêm.

cho Fc là bìa chính tắc cho F;

*i:= 0;*

*cho mỗi phụ thuộc chức năng α → β trong Fc*

*i := i + 1;*

*Ri :=αβ*

nếu không có sơ đồ Rj, j = 1, 2, … , i chứa khóa ứng viên cho R

sau đó

*:= i + 1;*

*Ri := any candidate key for R;*

/\* Tùy chọn, loại bỏ các mối quan hệ dư thừa \*/

nhắc lại, lặp lại

nếu bất kỳ lược đồ Rj nào được chứa trong một lược đồ Rk khác

sau đó

/\* Delete Rj \*/

*Rj := Ri;*

*i := i - 1;*

cho đến khi không còn Rj s nào có thể bị xóa

*return (R1, R2, … , Ri)*

**Hình 1.12** Sự bảo tồn phụ thuộc, phân hủy không mất mát thành 3NF.

Tập hợp các lược đồ kết quả có thể chứa các lược đồ dự phòng, với một lược đồ Rk chứa tất cả các thuộc tính của lược đồ Rj khác. Ví dụ, R2 ở trên chứa tất cả các thuộc tính từ R1. Thuật toán xóa tất cả các lược đồ như vậy được chứa trong một lược đồ khác. Bất kỳ sự phụ thuộc nào có thể được kiểm tra trên Rj bị xóa cũng có thể được kiểm tra trên mối quan hệ tương ứng Rk và sự phân hủy là không mất mát ngay cả khi Rj bị xóa.

Bây giờ chúng ta hãy xem xét lại lược đồ của mối quan hệ lớp của Mục 1.5.1.2 và áp dụng thuật toán phân hủy 3NF. Tập hợp các phụ thuộc chức năng mà chúng tôi liệt kê ở đó là một trang bìa chính tắc. Do đó, thuật toán cung cấp cho chúng ta ba sơ đồ khóa học, lớp học và phần tương tự.

Ví dụ trước minh họa một thuộc tính thú vị của thuật toán 3NF. Đôi khi, kết quả không chỉ ở 3NF, mà còn ở BCNF. Điều này gợi ý một phương pháp thay thế để tạo ra một thiết kế BCNF. Đầu tiên sử dụng thuật toán 3NF. Sau đó, đối với bất kỳ lược đồ nào trong thiết kế 3NF không có trong BCNF, hãy phân hủy bằng thuật toán BCNF. Nếu kết quả không phải là duy trì sự phụ thuộc, hãy quay lại thiết kế 3NF.

### 1.5.3. Tính chính xác của thuật toán 3NF

Thuật toán 3NF đảm bảo duy trì các phụ thuộc bằng cách xây dựng rõ ràng một lược đồ cho từng phụ thuộc trong một bìa chính tắc. Nó đảm bảo rằng sự phân hủy là một phân hủy không mất mát bằng cách đảm bảo rằng ít nhất một lược đồ chứa khóa ứng viên cho lược đồ đang được phân hủy. Bài tập thực hành 1.16 cung cấp một số thông tin chi tiết về bằng chứng cho thấy điều này đảm bảo sự phân hủy không mất mát.

Thuật toán này còn được gọi là thuật toán tổng hợp 3NF, vì nó lấy một tập hợp các sự phụ thuộc và thêm một lược đồ tại một thời điểm, thay vì phân hủy lược đồ ban đầu nhiều lần. Kết quả không được xác định duy nhất, vì một tập hợp các phụ thuộc chức năng có thể có nhiều hơn một bìa chính tắc. Thuật toán có thể phân hủy một mối quan hệ ngay cả khi nó đã ở 3NF; tuy nhiên, sự phân hủy vẫn được đảm bảo ở 3NF.

Để thấy rằng thuật toán tạo ra một thiết kế 3NF, hãy xem xét một lược đồ Ri trong quá trình phân hủy. Nhớ lại rằng khi chúng ta kiểm tra 3NF, cần phải xem xét các phụ thuộc chức năng có phía bên phải bao gồm một thuộc tính duy nhất. Do đó, để thấy rằng Ri nằm trong 3NF, bạn phải thuyết phục bản thân rằng bất kỳ phụ thuộc chức năng nào γ → B giữ trên Ri đều thỏa mãn định nghĩa của 3NF. Giả sử rằng sự phụ thuộc tạo ra Ri trong thuật toán tổng hợp là→ αβ. B phải nằm trong α hoặc β, vì B nằm trong Ri và αβ → được tạo ra Ri. Chúng ta hãy xem xét ba trường hợp có thể xảy ra:

• B có trong cả α và β. Trong trường hợp này, sự phụ thuộc α → β sẽ không có trong

Fc vì B sẽ là ngoại lai trong β. Do đó, trường hợp này không thể giữ được.

• B tính bằng β nhưng không phải α. Xem xét hai trường hợp:

° γ là siêu khóa. Điều kiện thứ hai của 3NF được thỏa mãn.

° γ không phải là siêu khóa. Sau đó α phải chứa một số thuộc tính không có trong γ. Bây giờ, vì γ → B nằm trong F +, nó phải có thể dẫn xuất từ Fc bằng cách sử dụng thuật toán đóng thuộc tính trên γ. Dẫn xuất không thể sử dụng αβ →, bởi vì nếu nó đã được sử dụng,

α phải được chứa trong việc đóng thuộc tính của γ, điều này là không thể, vì chúng ta giả định γ không phải là siêu khóa. Bây giờ, sử dụng α → (β − {B}) và γ → B, chúng ta

có thể thu được α → B (vì γ αβ, và γ không thể chứa B vì γ → B

không tầm thường). Điều này có nghĩa là B không liên quan ở phía bên phải của

α → β, không thể thực hiện được vì α → β nằm trong bìa chính tắc Fc. Do đó, nếu B tính bằng β, thì γ phải là siêu khóa, và điều kiện thứ hai của 3NF phải được thỏa mãn.

• B nằm trong α nhưng không β.

Vì α là một khóa ứng cử viên, giải pháp thay thế thứ ba trong định nghĩa của 3NF được thỏa mãn.

Điều thú vị là, thuật toán chúng tôi mô tả để phân tích thành 3NF có thể được bổ sung trong thời gian đa thức, mặc dù kiểm tra một lược đồ nhất định để xem liệu nó có thỏa mãn 3NF là NP -hard (có nghĩa là rất khó có khả năng một thuật toán đa thức-thời gian sẽ được phát minh cho nhiệm vụ này).

## 1.6. Phân hủy bằng cách sử dụng phụ thuộc đa giá trị

Một số sơ đồ quan hệ, mặc dù chúng nằm trong BCNF, dường như không được chuẩn hóa đầy đủ, theo nghĩa là chúng vẫn phải chịu vấn đề lặp lại thông tin. Hãy xem xét một biến thể của tổ chức trường đại học nơi một giảng viên có thể được liên kết với nhiều bộ phận và chúng tôi có mối quan hệ:

inst (ID, dept name, name, street, city)

Người đọc sắc sảo sẽ nhận ra lược đồ này là lược đồ không BCNF vì sự phụ thuộc chức năng

*ID → name, street, city*

và bởi vì giấy tờ tùy thân không phải là chìa khóa cho inst.

Hơn nữa, giả định rằng một giảng viên có thể có một số địa chỉ (ví dụ: một ngôi nhà mùa đông và một ngôi nhà mùa hè). Sau đó, chúng tôi không còn muốn thực thi sự phụ thuộc chức năn*g "ID→ street, city"* nữa, tuy nhiên, chúng tôi vẫn muốn thực thi *"ID → name"* (nghĩa là trường đại học không làm việc với các giảng viên hoạt động dưới nhiều bí danh!). Theo thuật toán phân hủy BCNF, chúng tôi thu được hai sơ đồ:

*r1 (ID, name)*

*r2 (ID, dept name, street, city)*

Cả hai đều thuộc BCNF (hãy nhớ rằng một giảng viên có thể được liên kết với nhiều bộ phận và một bộ phận có thể có một số giảng viên, và do đó, không phải "ID

→ dept name” hoặc “dept name → ID” giữ).

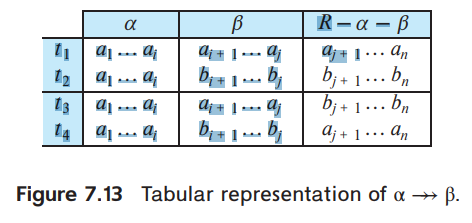
Mặc dù r2 nằm trong BCNF, có sự dư thừa. Chúng tôi lặp lại thông tin địa chỉ của mỗi nơi cư trú của một giảng viên một lần cho mỗi bộ phận mà giảng viên có liên quan. Chúng ta có thể giải quyết vấn đề này bằng cách phân tách r2 thành:

*r21 (dept name, ID)*

*r22 (ID, street, city)*

nhưng không có ràng buộc nào khiến chúng ta phải làm điều này.

Để giải quyết vấn đề này, chúng ta phải xác định một hình thức ràng buộc mới, được gọi là sự phụ thuộc tương đương. Như chúng ta đã làm cho các phụ thuộc chức năng, chúng ta sẽ sử dụng các phụ thuộc đa giá trị để xác định một biểu mẫu bình thường cho các lược đồ quan hệ. Dạng bình thường này, được gọi là dạng bình thường thứ tư (4NF), hạn chế hơn BCNF. Chúng ta sẽ thấy rằng mọi lược đồ 4NF cũng nằm trong BCNF nhưng có những lược đồ BCNF không có trong đó.



**Hình 1.13** Biểudiễn dạng bảng của α →→ β.

### 1.6.1. Phụ thuộc đa giá trị

Các phụ thuộc chức năng loại trừ một số bộ nhất định khỏi mối quan hệ. Nếu A → B, thì chúng ta không thể có hai bộ có cùng giá trị A nhưng các giá trị B khác nhau. Mặt khác, các phụ thuộc đa giá trị không loại trừ sự tồn tại của một số bộ nhất định. Thay vào đó, chúng yêu cầu các bộ khác có dạng nhất định phải có mặt trong mối quan hệ. Vì lý do này, các phụ thuộc chức năng đôi khi được gọi là các phụ thuộc tạo ra sự bình đẳng và các phụ thuộc đa giá trị được gọi là các depen- dencies tạo ra bộ.

Cho r(R) là một lược đồ quan hệ và cho α R và β R. Sự phụ thuộc đa giá trị

α →→ β

giữ trên R nếu, trong bất kỳ trường hợp pháp lý nào của mối quan hệ r(R), đối với tất cả các cặp bộ t1 và t2 trong r

sao cho t1[α] = t2[α], tồn tại các bộ t3 và t4 trong r sao cho

t1[α] = t2[α] = t3[α] = t4[α] t3[β] = t1[β]

t3[R − β] = t2[R − β]

t4[β] = t2[β]

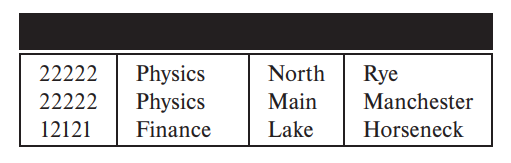
t4[R − β] = t1[R − β]

Định nghĩa này ít phức tạp hơn so với vẻ ngoài của nó. Hình 1.13 đưa ra hình ảnh dạng bảng về t1, t2, t3 và t4. Theo trực giác, sự phụ thuộc đa giá trị α →→ β nói rằng mối quan hệ giữa α và β độc lập với mối quan hệ giữa α và R − β. Nếu sự phụ thuộc đa giá trị α →→ β được thỏa mãn bởi tất cả các quan hệ trên lược đồ R, thì α →→ β là sự phụ thuộc đa giá trị tầm thường trên lược đồ R. Do đó, α →→ β là tầm thường nếu

β α hoặc β α = R. Điều này có thể được nhìn thấy bằng cách nhìn vào Hình 1.13 và xem xét hai trường hợp đặc biệt β α và β α = R. Trong mỗi trường hợp, bảng giảm xuống chỉ còn hai cột và chúng ta thấy rằng t1 và t2 có thể phục vụ trong vai trò của t3 và t4.

Để minh họa sự khác biệt giữa các phụ thuộc chức năng và đa giá trị, chúng tôi

xem xét lược đồ r2 một lần nữa, và một mối quan hệ ví dụ trên lược đồ đó được thể hiện trong Hình 1.14. Chúng ta phải lặp lại tên bộ phận một lần cho mỗi địa chỉ mà giảng viên có, và chúng ta phải lặp lại địa chỉ cho mỗi bộ phận mà giảng viên có liên quan. Sự lặp lại này là không cần thiết, vì mối quan hệ giữa người hướng dẫn



**Hình 1.14** Một ví dụ về dự phòng trong một mối quan hệ trên sơ đồ BCNF

và địa chỉ của anh ấy độc lập với mối quan hệ giữa người hướng dẫn đó và một người chia tay. Nếu một giảng viên có ID 22222 được liên kết với bộ phận Vật lý, chúng tôi muốn bộ phận đó được liên kết với tất cả các địa chỉ của giảng viên đó. Do đó, mối quan hệ của Hình 1.15 là bất hợp pháp. Để làm cho mối quan hệ này hợp pháp, chúng ta cần thêm các bộ (Physics, 22222, Main, Manchester) và (Math, 22222, North, Rye) vào mối quan hệ của Hình 1.15.

So sánh ví dụ trên với định nghĩa của chúng ta về sự phụ thuộc đa giá trị, chúng ta thấy rằng chúng ta muốn sự phụ thuộc đa giá trị:

*ID →→ street, city*

để giữ. (ID →→ dept\_name phụ thuộc đa giá trị cũng sẽ hoạt động. Chúng ta sẽ sớm thấy rằng chúng tương đương nhau.)

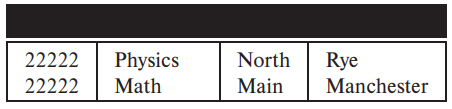
Như với các phụ thuộc chức năng, chúng ta sẽ sử dụng các phụ thuộc đa giá trị theo hai cách:

1. Để kiểm tra các mối quan hệ để xác định xem chúng có hợp pháp theo một tập hợp các phụ thuộc chức năng và đa giá trị nhất định hay không.

2. Để xác định các ràng buộc đối với tập hợp các quan hệ pháp lý; do đó, chúng ta sẽ chỉ quan tâm đến những mối quan hệ đáp ứng một tập hợp các phụ thuộc chức năng và đa giá trị nhất định.

Lưu ý rằng, nếu một mối quan hệ r không thỏa mãn một phụ thuộc đa giá trị nhất định, chúng ta có thể cấu trúc một mối quan hệ r ′ đáp ứng sự phụ thuộc đa giá trị bằng cách thêm bộ vào r. Cho D biểu thị một tập hợp các phụ thuộc chức năng và đa giá trị. Kết thúc D+

của D là tập hợp của tất cả các phụ thuộc chức năng và đa giá trị được ngụ ý một cách hợp lý bởi D. Như chúng ta đã làm cho các phụ thuộc chức năng, chúng ta có thể tính toán D+ từ D, sử dụng các định nghĩa chính thức về các phụ thuộc chức năng và các phụ thuộc đa giá trị. Chúng ta có thể giải quyết được.



**Hình 1.15** Một mối quan hệ bất hợp pháp r2.

với lý do như vậy cho các phụ thuộc đa giá trị rất đơn giản. May mắn thay, các phụ thuộc đa giá trị xảy ra trong thực tế dường như khá đơn giản. Đối với các phụ thuộc phức tạp, tốt hơn là lý luận về các tập hợp các phụ thuộc bằng cách sử dụng một hệ thống các quy tắc suy luận.

Từ định nghĩa về sự phụ thuộc đa giá trị, chúng ta có thể rút ra các quy tắc sau cho α, β R:

• Nếu α → β, thì α →→ β. Nói cách khác, mọi phụ thuộc chức năng cũng là một phụ thuộc đa giá trị.

• Nếu α →→ β, thì α →→ R − α − β

Phần 28.1.1 phác thảo một hệ thống các quy tắc suy luận cho các phụ thuộc đa giá trị.

### 1.6.2. Biểu Mẫu Thông Thường Thứ Tư

Hãy xem xét lại ví dụ của chúng ta về giản đồ BCNF:

*r2 (ID, dept name, street, city)*

trong đó ID →→ street phụ thuộc đa giá trị, thành phố nắm giữ. Chúng ta đã thấy trong các đoạn mở đầu của Phần 1.6 rằng, mặc dù sơ đồ này nằm trong BCNF, nhưng thiết kế không lý tưởng, vì chúng ta phải lặp lại thông tin địa chỉ của người hướng dẫn cho từng bộ phận. Chúng ta sẽ thấy rằng chúng ta có thể sử dụng sự phụ thuộc đa giá trị đã cho để cải thiện thiết kế cơ sở dữ liệu bằng cách phân hủy lược đồ này thành phân hủy dạng chuẩn thứ tư.

Lược đồ quan hệ R ở dạng pháp tuyến thứ tư (4NF) đối với tập hợp D của các phụ thuộc chức năng và đa giá trị nếu, đối với tất cả các phụ thuộc đa giá trị trong D+ của dạng α →→ β , trong đó α R và β R, ít nhất một trong các giá trị sau:

• α →→ β là một sự phụ thuộc đa giá trị tầm thường.

• α là một siêu khóa cho R.

Một thiết kế cơ sở dữ liệu nằm trong 4NF nếu mỗi thành viên của tập hợp các sơ đồ quan hệ liên kết thiết kế nằm trong 4NF.

Lưu ý rằng định nghĩa của 4NF khác với định nghĩa của BCNF chỉ trong việc sử dụng các phụ thuộc đa giá trị. Mỗi lược đồ 4NF nằm trong BCNF. Để thấy thực tế này, chúng tôi lưu ý rằng, nếu lược đồ R không có trong BCNF, thì có một phụ thuộc chức năng không tầm thường α → β giữ trên R, trong đó α không phải là siêu khóa. Vì α → β ngụ ý α →→ β, R không thể có trong 4NF.

Cho R là một lược đồ quan hệ, và cho R1, R2, … , Rn là một sự phân hủy của R. Để kiểm tra xem mỗi lược đồ quan hệ Ri trong phân hủy có nằm trong 4NF hay không, chúng ta cần tìm ra những phụ thuộc đa giá trị nào giữ trên mỗi Ri. Nhớ lại rằng, Fora thiết lập F của depen- dencies chức năng, hạn chế Fi của F đến Ri là tất cả các phụ thuộc chức năng trong F + chỉ bao gồm các thuộc tính của Ri. Bây giờ hãy xem xét một tập hợp D của cả hai depen- dencies chức năng và đa giá trị. Hạn chế của D đến Ri là tập hợp Di bao gồm:

1. Tất cả các phụ thuộc chức năng trong D+ chỉ bao gồm các thuộc tính của Ri.

2. Tất cả các phụ thuộc đa giá trị của biểu mẫu:

*α →→ β ∩ Ri*

*trong đó α Ri và α →→ β tính bằng D+.*

### 1.6.3. Phân hủy 4NF

Sự tương tự giữa 4NF và BCNF áp dụng cho thuật toán để phân tích lược đồ thành 4NF. Hình 1.16 cho thấy thuật toán phân hủy 4NF. Nó giống hệt với thuật toán phân hủy BCNF trên Hình 1.11, ngoại trừ việc nó sử dụng các phụ thuộc đa giá trị và sử dụng hạn chế từ D+ đến Ri.

Nếu chúng ta áp dụng thuật toán của Hình 1.16 cho (ID, tên phòng, đường phố, thành phố), chúng ta thấy rằng

*ID→→ dept\_name* là một phụ thuộc đa giá trị không tầm thường và ID không phải là một siêu khóa cho lược đồ. Theo thuật toán, chúng tôi thay thế nó bằng hai lược đồ:

*(ID, dept name)*

*(ID, street, city)*

Cặp lược đồ này, nằm trong 4NF, loại bỏ sự dư thừa mà chúng tôi gặp phải.

Như trường hợp khi chúng ta chỉ xử lý các phụ thuộc chức năng, chúng ta quan tâm đến các phân hủy không mất mát và duy trì các phụ thuộc. Thực tế hạ thấp về sự phụ thuộc đa giá trị và không mất mát cho thấy thuật toán trên Hình 1.16 chỉ tạo ra sự phân hủy không mất mát:

result := {R};

done := false;

compute D+; Với lược đồ Ri, hãy để Di biểu thị giới hạn của D+ đến Ri

while (not done) do

nếu (có một lược đồ Ri kết quả không nằm trong 4NF w.r.t. Di )

Vậy thì hãy bắt đầu

let α →→ β là một phụ thuộc đa giá trị không tầm thường

mà giữ trên → Ri sao cho α Ri không nằm trong Di, và αβ = ∅;

result := (result − Ri

) ∪ (Ri − β) ∪ (α, β);

end

else done := true;

**Hình 1.16** Thuật toán phân hủy 4NF.

• Cho r(R) là một lược đồ quan hệ và cho D là một tập hợp các phụ thuộc chức năng và đa giá trị trên R. Cho r1(R1) và r2(R2) tạo thành sự phân hủy của R. Thành phần khử giá trị này của R là không mất mát khi và chỉ khi ít nhất một trong các phụ thuộc đa giá trị sau nằm trong D+:

*R1 ∩ R2 →→ R1*

*R1 ∩ R2 →→ R2*

Nhớ lại rằng chúng tôi đã tuyên bố trong Phần 1.2.3 rằng, nếu f R1 ∩ R2 → R1 or R1 ∩ R2 → R2,, thì r1(R1) và r2(R2) tạo thành sự phân hủy không mất r(R). Thực tế trước đây về các phụ thuộc đa giá trị là một tuyên bố tổng quát hơn về tính không mất mát. Nó nói rằng, đối với mỗi sự phân hủy không mất mát của r(R ) thành hai lược đồ r 1 (R1) và r2(R2), một trong hai phụ thuộc R1 ∩ R2 →→ R1 or R1 ∩ R2 →→ R2 phải giữ. Để thấy rằng điều này

là đúng, trước tiên chúng ta cần chỉ ra rằng nếu ít nhất một trong những phụ thuộc này được giữ, thì

ΠR (r), R (r) = r và tiếp theo chúng ta cần chứng minh rằng nếu ΠR (r), R (r) = r thì r(R)

phải đáp ứng ít nhất một trong những phụ thuộc này. Xem phần Đọc thêm để biết tài liệu tham khảo về bằng chứng đầy đủ.

Vấn đề bảo tồn phụ thuộc khi chúng ta phân hủy một lược đồ quan hệ trở nên phức tạp hơn khi có sự hiện diện của các phụ thuộc đa giá trị. Mục 28.1.2 theo đuổi chủ đề này.

Một phức tạp khác phát sinh từ thực tế là một phụ thuộc đa giá trị chỉ có thể giữ trên một tập hợp con thích hợp của lược đồ đã cho, không có cách nào để thể hiện sự phụ thuộc đa giá trị đó trên lược đồ đã cho. Sự phụ thuộc đa giá trị như vậy có thể xuất hiện như là kết quả của sự phân hủy. May mắn thay, những trường hợp như vậy, được gọi là phụ thuộc đa giá trị nhúng, là rất hiếm. Xem phần Đọc thêm để biết chi tiết.

## 1.7. Các biểu mẫu thông thường khác

Dạng chuẩn tắc thứ tư hoàn toàn không phải là dạng chuẩn tắc "cuối cùng". Như chúng ta đã thấy trước đó, các phụ thuộc đa giá trị giúp chúng ta hiểu và loại bỏ một số dạng lặp lại thông tin không thể hiểu được về các phụ thuộc chức năng. Có các loại ràng buộc được gọi là các phụ thuộc liên kết tổng quát hóa các phụ thuộc đa giá trị và dẫn đến một dạng bình thường khác gọi là dạng bình thường liên kết dự án (PJNF). PJNF được gọi là hình thức bình thường thứ năm trong một số sách. Có một lớp các ràng buộc thậm chí còn tổng quát hơn dẫn đến một dạng bình thường được gọi là dạng bình thường khóa miền (DKNF).

Một vấn đề thực tế với việc sử dụng các ràng buộc tổng quát này là chúng không chỉ khó lý luận mà còn không có bộ quy tắc suy luận hợp lý và đầy đủ để lý luận về các ràng buộc. Do đó PJNF và DKNF được sử dụng khá hiếm. Chương 28 cung cấp thêm chi tiết về các biểu mẫu thông thường này.

Dễ thấy bởi sự vắng mặt của nó từ cuộc thảo luận của chúng tôi về các dạng bình thường là dạng bình thường thứ hai (2NF). Chúng tôi đã không thảo luận về nó vì nó chỉ có lợi ích lịch sử. Chúng tôi chỉ đơn giản là định nghĩa nó và để bạn thử nghiệm nó trong Bài tập thực hành 1.19. Hình thức bình thường đầu tiên đề cập đến một vấn đề khác với các hình thức bình thường mà chúng ta đã thấy cho đến nay. Điều này sẽ được thảo luận trong phần tiếp theo.

## 1.8. Miền nguyên tử và dạng chuẩn thứ nhất

Mô hình E-R cho phép các tập hợp thực thể và các tập hợp mối quan hệ có các thuộc tính có một số mức độ cấu trúc phụ. Cụ thể, nó cho phép các thuộc tính đa giá trị như số điện thoại trong Hình 6.8 và các thuộc tính tổng hợp (chẳng hạn như địa chỉ thuộc tính với các thuộc tính component street, city và state). Khi chúng tôi tạo các bảng từ các thiết kế E-R có chứa các loại thuộc tính này, chúng tôi sẽ loại bỏ cấu trúc phụ này. Đối với các thuộc tính tổng hợp, chúng tôi cho phép mỗi thành phần là một thuộc tính theo đúng nghĩa của nó. Đối với các cống phẩm đa giá trị, chúng tôi tạo một bộ cho mỗi mục trong một tập hợp đa giá trị.

Trong mô hình quan hệ, chúng tôi chính thức hóa ý tưởng này rằng các thuộc tính không có bất kỳ cấu trúc phụ nào. Vùng chức năng là nguyên tử nếu các phần tử của vùng chức năng được coi là các đơn vị không thể chia nhỏ. Ta nói rằng giản đồ quan hệ R ở dạng pháp tuyến thứ nhất (1NF) nếu các miền của tất cả các thuộc tính của R là nguyên tử.

Một tập hợp các tên là một ví dụ về một giá trị phi nguyên tử. Ví dụ: nếu lược đồ của một nhân viên quan hệ bao gồm một thuộc tính con có các phần tử miền là tập hợp các tên, lược đồ sẽ không ở dạng bình thường đầu tiên.

Các thuộc tính tổng hợp, chẳng hạn như địa chỉ thuộc tính với các thuộc tính thành phần

và thành phố cũng có các miền phi nguyên tử.

Số nguyên được giả định là nguyên tử, vì vậy tập hợp các số nguyên là một miền nguyên tử; tuy nhiên, tập hợp tất cả các tập hợp số nguyên là một miền không phải nguyên tử. Sự khác biệt là chúng ta thường không coi các số nguyên có các phần phụ, nhưng chúng ta coi các tập hợp số nguyên có các phần phụ - cụ thể là các số nguyên tạo nên tập hợp. Nhưng vấn đề quan trọng không phải là bản thân miền đó là gì, mà là cách chúng ta sử dụng các phần tử miền trong cơ sở dữ liệu của mình. Miền của tất cả các số nguyên sẽ không phải là nguyên tử nếu chúng ta coi mỗi số nguyên là một danh sách các chữ số có thứ tự.

Để minh họa thực tế về điểm này, hãy xem xét một tổ chức chỉ định số nhận dạng em-ployees theo hình thức sau: Hai chữ cái đầu tiên chỉ định bộ phận và bốn chữ số còn lại là một số duy nhất trong bộ phận cho nhân viên. Ví dụ về các số này sẽ là “CS001” và “EE1127”. Các số nhận dạng như vậy có thể được chia thành các đơn vị nhỏ hơn và do đó không phải là nguyên tử. Nếu lược đồ quan hệ có một thuộc tính có miền bao gồm các số nhận dạng được mã hóa như trên, lược đồ sẽ không ở dạng bình thường đầu tiên.

Khi số nhận dạng như vậy được sử dụng, bộ phận của một nhân viên có thể được tìm thấy bằng cách viết mã phá vỡ cấu trúc của một số nhận dạng. Làm như vậy đòi hỏi phải lập trình thêm và thông tin được mã hóa trong chương trình ứng dụng thay vì trong cơ sở dữ liệu. Các vấn đề khác phát sinh nếu số nhận dạng đó được sử dụng làm khóa chính: Khi nhân viên thay đổi bộ phận, số nhận dạng của nhân viên phải được thay đổi ở mọi nơi xảy ra, có thể là một nhiệm vụ khó khăn hoặc mã diễn giải số sẽ cho kết quả sai.

Từ cuộc thảo luận này, có vẻ như việc chúng ta sử dụng các định danh khóa học như "CS- 101", trong đó "CS" chỉ ra bộ phận Khoa học Máy tính, có nghĩa là miền định danh khóa học không phải là nguyên tử. Vùng chức năng như vậy không phải là nguyên tử đối với con người sử dụng hệ thống này. Tuy nhiên, ứng dụng cơ sở dữ liệu vẫn coi miền là nguyên tử, miễn là nó không cố gắng phân tách mã định danh và diễn giải các phần của mã định danh dưới dạng viết tắt của bộ phận. Lược đồ khóa học lưu trữ tên bộ phận dưới dạng một thuộc tính riêng biệt và ứng dụng cơ sở dữ liệu có thể sử dụng giá trị thuộc tính này để tìm bộ phận của khóa học, thay vì diễn giải các ký tự cụ thể của mã định danh khóa học. Do đó, lược đồ trường đại học của chúng ta có thể được coi là ở dạng bình thường đầu tiên.

Việc sử dụng các thuộc tính có giá trị đã đặt có thể dẫn đến các thiết kế có lưu trữ dữ liệu dư thừa, từ đó có thể dẫn đến sự không nhất quán. Ví dụ, thay vì có mối quan hệ giữa người hướng dẫn và các phần được thể hiện như một mối quan hệ riêng biệt, một nhà thiết kế cơ sở dữ liệu có thể bị cám dỗ để lưu trữ một tập hợp các định danh phần khóa học với mỗi người hướng dẫn và một tập hợp các định danh người hướng dẫn với mỗi phần. (Các khóa chính của phần và người hướng dẫn được sử dụng làm định danh.) Bất cứ khi nào dữ liệu liên quan đến người hướng dẫn dạy phần nào được thay đổi, việc cập nhật phải được thực hiện ở hai nơi: trong tập hợp người hướng dẫn cho phần đó và trong tập hợp các phần cho hướng dẫn. Việc không thực hiện cả hai bản cập nhật có thể khiến cơ sở dữ liệu ở trạng thái không nhất quán. Chỉ giữ một trong những bộ này sẽ tránh được thông tin lặp lại; tuy nhiên chỉ giữ một trong số này sẽ làm phức tạp một số truy vấn, và không rõ nên giữ lại bộ nào trong số hai bộ này.

Một số loại giá trị phi nguyên tử có thể hữu ích, mặc dù chúng nên được sử dụng cẩn thận. Ví dụ, các thuộc tính có giá trị tổng hợp thường hữu ích và các thuộc tính có giá trị đặt cũng hữu ích trong nhiều trường hợp, đó là lý do tại sao cả hai đều được hỗ trợ trong mô hình E-R. Trong nhiều lĩnh vực mà các thực thể có cấu trúc phức tạp, việc buộc biểu diễn dạng chuẩn đầu tiên đại diện cho một gánh nặng không cần thiết đối với ứng dụng lập trình viên, người phải viết mã để chuyển đổi dữ liệu thành dạng nguyên tử. Ngoài ra còn có chi phí thời gian chạy của việc chuyển đổi dữ liệu qua lại từ dạng nguyên tử. Do đó, sự hỗ trợ cho các giá trị không phải nguyên tử có thể rất hữu ích trong các vùng như vậy. Trên thực tế, các hệ thống cơ sở dữ liệu hiện đại hỗ trợ nhiều loại giá trị phi nguyên tử, như chúng ta sẽ thấy trong Chương 29 giới hạn bản thân trong các mối quan hệ ở dạng bình thường đầu tiên, và do đó tất cả các miền đều là nguyên tử.

## 1.9. Quy trình thiết kế cơ sở dữ liệu

Cho đến nay chúng tôi đã xem xét các vấn đề chi tiết về các hình thức bình thường và bình thường hóa. Trong phần này, chúng tôi nghiên cứu cách chuẩn hóa phù hợp với quy trình thiết kế cơ sở dữ liệu tổng thể.

Trước đó trong chương bắt đầu từ Phần 1.1.1, chúng tôi giả định rằng một lược đồ quan hệ *r*(*R*) được đưa ra và chúng tôi đã tiến hành chuẩn hóa nó. Có một số cách mà chúng tôi có thể đưa ra lược đồ *r*(*R*):

1. *r*(*R*) có thể đã được tạo ra trong việc chuyển đổi sơ đồ E-R thành một tập hợp các sơ đồ quan hệ.
2. *r*(*R*) có thể là kết quả của một thiết kế đặc biệt của các mối quan hệ mà sau đó chúng tôi kiểm tra để xác minh rằng nó đáp ứng một hình thức bình thường mong muốn.

Trong phần còn lại của phần này, chúng tôi xem xét ý nghĩa của các phương pháp này. Chúng tôi cũng kiểm tra một số vấn đề thực tế trong thiết kế cơ sở dữ liệu, bao gồm phi chuẩn hóa cho mỗi hình thức và các ví dụ về thiết kế xấu không được phát hiện bằng cách chuẩn hóa.

### 1.9.1. Mô hình E-R và Chuẩn hóa

Khi chúng ta xác định một sơ đồ E-R một cách cẩn thận, xác định tất cả các tập hợp thực thể một cách chính xác, các sơ đồ quan hệ được tạo ra từ sơ đồ E-R không cần phải bình thường hóa thêm nữa. Tuy nhiên, có thể có các phụ thuộc chức năng giữa các thuộc tính của một tập hợp thực thể. Ví dụ: giả sử một tập hợp thực thể *người hướng dẫn* có các thuộc tính *dept name* và *dept address*, và có một function dependency *dept name* → *dept address*. Sau đó, chúng ta sẽ cần bình thường hóa mối quan hệ được tạo ra từ *người hướng dẫn*.

Hầu hết các ví dụ về các phụ thuộc như vậy phát sinh từ thiết kế sơ đồ E-R kém. Trong ví dụ trước, nếu chúng ta đã thiết kế sơ đồ E-R chính xác, chúng ta sẽ tạo ra một tập hợp thực thể *bộ phận* với *địa chỉ bộ phận* thuộc tính và tập hợp mối quan hệ giữa *người hướng dẫn* và *bộ phận*. Tương tự, một tập hợp mối quan hệ liên quan đến nhiều hơn hai tập hợp thực thể có thể dẫn đến một lược đồ có thể không ở dạng bình thường mong muốn. Vì hầu hết các bộ mối quan hệ là nhị phân, những trường hợp như vậy là tương đối hiếm. (Trên thực tế, một số biến thể E-R-diagram thực sự làm cho nó khó khăn hoặc không thể chỉ định các tập hợp mối quan hệ phi nhị phân.) Các phụ thuộc chức năng có thể giúp chúng tôi phát hiện thiết kế E-R kém. Nếu các sơ đồ tái lation được tạo không ở dạng bình thường mong muốn, vấn đề có thể được khắc phục trong sơ đồ E-R. Nghĩa là, việc chuẩn hóa có thể được thực hiện chính thức như một phần của mô hình hóa dữ liệu. Thay đổi, bình thường hóa có thể được để lại cho trực giác của nhà thiết kế trong quá trình mô hình hóa E-R, và nó có thể được thực hiện chính thức trên các sơ đồ quan hệ được tạo ra từ mô hình E-R.

Một độc giả cẩn thận sẽ lưu ý rằng để chúng tôi minh họa nhu cầu về các phụ thuộc tương đương và dạng chuẩn thứ tư, chúng tôi phải bắt đầu với các sơ đồ không có nguồn gốc từ thiết kế E-R của chúng tôi. Thật vậy, quá trình tạo ra một thiết kế E-R có xu hướng tạo ra các thiết kế 4NF. Nếu một sự phụ thuộc đa giá trị giữ và không được ngụ ý bởi sự phụ thuộc chức năng tương ứng, nó thường phát sinh từ một trong các nguồn sau:

• Một bộ mối quan hệ nhiều-nhiều.

• Một thuộc tính đa giá trị của một tập hợp thực thể.

Đối với tập hợp mối quan hệ nhiều-nhiều, mỗi tập hợp thực thể liên quan có lược đồ riêng và có một lược đồ bổ sung cho tập hợp mối quan hệ. Đối với một thuộc tính đa giá trị, một lược đồ riêng biệt được tạo bao gồm thuộc tính đó và khóa chính của tập hợp thực thể (như trong trường hợp thuộc tính *số điện thoại* của *người hướng dẫn* tập hợp thực thể).

Cách tiếp cận quan hệ phổ quát đối với thiết kế cơ sở dữ liệu quan hệ bắt đầu bằng một tổng kết rằng có một sơ đồ quan hệ duy nhất chứa tất cả các thuộc tính quan tâm. Lược đồ đơn này xác định cách người dùng và ứng dụng tương tác với cơ sở dữ liệu.

### 1.9.2. Đặt tên các thuộc tính và mối quan hệ

Một tính năng mong muốn của thiết kế cơ sở dữ liệu là giả định vai trò duy nhất, có nghĩa là mỗi tên thuộc tính có một ý nghĩa duy nhất trong cơ sở dữ liệu. Điều này ngăn cản chúng ta sử dụng cùng một thuộc tính để có ý nghĩa khác nhau trong các lược đồ khác nhau. Ví dụ: chúng ta có thể cân nhắc sử dụng *số* thuộc tính cho số điện thoại trong lược đồ *giảng viên* và số phòng trong lược đồ *lớp học*. Sự tham gia của một mối quan hệ trên *người hướng dẫn* lược đồ với một người trong *lớp học* là vô nghĩa. Mặc dù người dùng và nhà phát triển ứng dụng có thể làm việc cẩn thận để đảm bảo sử dụng đúng *số* trong từng trường hợp, nhưng việc có tên thuộc tính khác nhau cho số điện thoại và số phòng sẽ giúp giảm lỗi của người dùng.

Mặc dù việc giữ tên cho các thuộc tính không tương thích là một ý tưởng hay, nhưng nếu các mối quan hệ khác nhau có cùng ý nghĩa, thì có thể nên sử dụng cùng một tên thuộc tính. Vì lý do này, chúng tôi đã sử dụng cùng một tên thuộc tính "*tên*" cho cả tập hợp *người hướng dẫn* và thực thể *sinh viên*. Nếu đây không phải là trường hợp (tức là, nếu chúng ta sử dụng các quy ước đặt tên khác nhau cho tên người hướng dẫn và học sinh), thì nếu chúng ta muốn khái quát hóa các tập thực thể này bằng cách tạo một tập thực thể *cá nhân*, chúng ta sẽ phải đổi tên thuộc tính. Do đó, ngay cả khi chúng ta hiện không có khái quát về *sinh viên* và *người hướng dẫn*, nếu chúng ta thấy trước khả năng như vậy, tốt nhất là sử dụng cùng một tên trong cả hai bộ thực thể (và các mối quan hệ).

Mặc dù về mặt kỹ thuật, thứ tự của các tên thuộc tính trong lược đồ không quan trọng, nhưng đó là một quy ước để liệt kê các thuộc tính khóa chính trước tiên. Điều này làm cho việc đọc đầu ra mặc định (như từ chọn \*) dễ dàng hơn.

Trong các lược đồ cơ sở dữ liệu lớn, các tập hợp mối quan hệ (và các lược đồ bắt nguồn từ đó) thường được đặt tên thông qua việc ghép nối tên của các tập hợp thực thể liên quan, có lẽ với dấu gạch nối hoặc dấu gạch dưới ở giữa. Chúng tôi đã sử dụng một vài tên như vậy, ví dụ, *inst sec* và *student sec*. Chúng tôi đã sử dụng các tên *dạy* và *lấy* thay vì sử dụng các tên ghép nối dài hơn. Điều này có thể chấp nhận được vì bạn không khó để nhớ các tập hợp thực thể liên quan cho một vài tập hợp mối quan hệ. Chúng ta không thể luôn tạo ra các mối quan hệ được đặt tên bằng cách ghép nối đơn giản; ví dụ, một người quản lý hoặc làm việc cho mối quan hệ giữa các nhân viên sẽ không có ý nghĩa nhiều nếu nó được gọi là *nhân viên nhân viên*! Tương tự, nếu có nhiều tập hợp mối quan hệ có thể có giữa một cặp tập hợp thực thể, các tên tập hợp mối quan hệ phải bao gồm các phần bổ sung để xác định tập hợp mối quan hệ.

Các tổ chức khác nhau có các quy ước khác nhau để đặt tên cho các bộ thực thể. Ví dụ, chúng tôi có thể gọi một tập hợp thực thể gồm *sinh viên* hoặc *học sinh*. Chúng tôi đã chọn sử dụng dạng số ít trong các thiết kế cơ sở dữ liệu của mình. Sử dụng số ít hoặc số nhiều là chấp nhận được, miễn là quy ước được sử dụng nhất quán trên tất cả các bộ thực thể.

Khi các lược đồ phát triển lớn hơn, với số lượng bộ mối quan hệ ngày càng tăng, việc sử dụng cách đặt tên nhất quán các thuộc tính, mối quan hệ và thực thể giúp công việc của nhà thiết kế cơ sở dữ liệu và lập trình viên ứng dụng trở nên dễ dàng hơn nhiều

### 1.9.3.Không chuẩn hóa cho hiệu suất

Đôi khi các nhà thiết kế cơ sở dữ liệu chọn một lược đồ có thông tin dư thừa; nghĩa là nó không được chuẩn hóa. Họ sử dụng sự dư thừa để cải thiện hiệu suất cho các ứng dụng cụ thể. Hình phạt được trả cho việc không sử dụng lược đồ chuẩn hóa là công việc bổ sung (về thời gian mã hóa và thời gian thực hiện) để giữ cho dữ liệu dư thừa nhất quán.

Ví dụ: giả sử tất cả các điều kiện tiên quyết của khóa học phải được hiển thị cùng với thông tin khóa học, mỗi khi một khóa học được truy cập. Trong lược đồ chuẩn hóa của chúng tôi, điều này đòi hỏi phải tham gia *tất nhiên* với *prereq*.

Một giải pháp thay thế để tính toán phép nối trên máy bay là lưu trữ một mối quan hệ chứa tất cả các thuộc tính của *khóa học* và *prereq*. Điều này giúp hiển thị thông tin khóa học "đầy đủ" nhanh hơn. Tuy nhiên, thông tin cho một khóa học được lặp lại cho mọi điều kiện tiên quyết của khóa học và tất cả các bản sao phải được ứng dụng cập nhật, bất cứ khi nào điều kiện tiên quyết của khóa học được thêm hoặc bỏ. Quá trình lấy lược đồ chuẩn hóa và làm cho nó không chuẩn hóa được gọi là phi chuẩn hóa và các nhà thiết kế sử dụng nó để điều chỉnh hiệu suất của các hệ thống để hỗ trợ các hoạt động quan trọng về thời gian.

Một giải pháp thay thế tốt hơn, được hỗ trợ bởi nhiều hệ thống cơ sở dữ liệu hiện nay, là sử dụng lược đồ nor- malized và lưu trữ thêm join of *course* và *prereq* như một chế độ xem cụ thể. (Hãy nhớ lại rằng một chế độ xem được vật chất hóa là một chế độ xem có kết quả được lưu trữ trong cơ sở dữ liệu và được cập nhật khi các mối quan hệ được sử dụng trong chế độ xem được cập nhật.) Giống như từ chối, việc sử dụng các chế độ xem được vật chất hóa có chi phí không gian và thời gian; tuy nhiên, nó có lợi thế là giữ cho chế độ xem được cập nhật là công việc của hệ thống cơ sở dữ liệu, không phải là lập trình viên ứng dụng.

### 1.9.4. Các vấn đề thiết kế khác

Có một số khía cạnh của thiết kế cơ sở dữ liệu không được giải quyết bằng cách bình thường hóa và do đó có thể dẫn đến thiết kế cơ sở dữ liệu kém. Dữ liệu liên quan đến thời gian hoặc khoảng thời gian có một số vấn đề như vậy. Chúng tôi đưa ra ví dụ ở đây; rõ ràng, các thiết kế như vậy nên tránh. Hãy xem xét một cơ sở dữ liệu của trường đại học, nơi chúng tôi muốn lưu trữ tổng số hướng dẫn trong mỗi bộ phận trong các năm khác nhau. *Tổng số* mối quan hệ (*tên phòng*, *năm*, *kích thước*) có thể được sử dụng để lưu trữ thông tin mong muốn. Sự phụ thuộc chức năng duy nhất vào điều này mối quan hệ là *tên dept*,→ *quy mô* *năm* và mối quan hệ nằm trong BCNF.

Một thiết kế thay thế là sử dụng nhiều mối quan hệ, mỗi mối quan hệ lưu trữ thông tin kích thước cho một năm khác nhau. Giả sử các năm quan tâm là 2017, 2018 và 2019; sau đó chúng ta sẽ có các mối quan hệ của mẫu *TOTAL INT 2017*, *TOTAL INT 2018*, *TOTAL INT 2019*, tất cả đều nằm trên lược đồ (*tên phòng*, *kích thước*). Sự phụ thuộc chức năng duy nhất ở đây trên mỗi mối quan hệ sẽ là→ *kích thước* *tên dept*, vì vậy các mối quan hệ này cũng nằm trong BCNF.

Tuy nhiên, thiết kế thay thế này rõ ràng là một ý tưởng tồi - chúng ta sẽ phải tạo ra một mối quan hệ mới mỗi năm, và chúng ta cũng sẽ phải viết các truy vấn mới mỗi năm, để mỗi mối quan hệ mới vào tài khoản. Các truy vấn cũng sẽ phức tạp hơn vì chúng có thể phải đề cập đến nhiều mối quan hệ.

Tuy nhiên, một cách khác để đại diện cho cùng một dữ liệu là có một *năm* quan hệ duy nhất (*tên phòng*, *tổng inst 2017*, *tổng inst 2018*, *tổng inst 2019*). Ở đây, các phụ thuộc chức năng duy nhất là từ *tên phòng* ban đến các thuộc tính khác, và một lần nữa mối quan hệ là trong BCNF. Thiết kế này cũng là một ý tưởng tồi vì nó có các vấn đề tương tự như thiết kế trước đó - cụ thể là chúng ta sẽ phải sửa đổi lược đồ quan hệ và viết các truy vấn mới mỗi năm. Các truy vấn cũng sẽ phức tạp hơn, vì chúng có thể phải đề cập đến nhiều thuộc tính.

Các đại diện như các đại diện trong mối quan hệ *năm phòng*, với một cột cho mỗi giá trị của một thuộc tính, được gọi là bảng chéo; chúng được sử dụng rộng rãi trong các bảng tính và báo cáo và trong các công cụ phân tích dữ liệu. Mặc dù các đại diện như vậy rất hữu ích để hiển thị cho người dùng, nhưng vì những lý do vừa được đưa ra, chúng không được mong muốn trong thiết kế cơ sở dữ liệu. SQL bao gồm các tính năng để chuyển đổi dữ liệu từ biểu diễn quan hệ thông thường sang tab chéo, để hiển thị, như chúng ta đã thảo luận trong Phần 11.3.1.

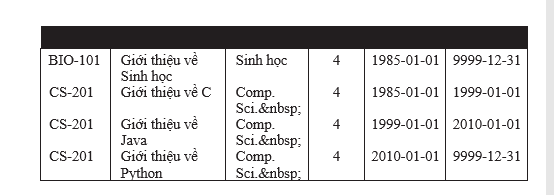
## 1.10. Mô hình hóa dữ liệu thời gian

Giả sử chúng ta giữ lại dữ liệu trong tổ chức trường đại học của chúng ta không chỉ cho thấy địa chỉ của từng giảng viên, mà còn tất cả các địa chỉ cũ mà trường đại học biết. Sau đó, chúng tôi có thể hỏi các câu hỏi, chẳng hạn như "Tìm tất cả các giảng viên sống ở Princeton vào năm 1981." Trong trường hợp này, chúng tôi có thể có nhiều địa chỉ cho giảng viên. Mỗi địa chỉ có một ngày bắt đầu và ngày kết thúc liên quan, cho biết thời điểm giảng viên cư trú tại địa chỉ đó. Một giá trị đặc biệt cho ngày kết thúc, ví dụ: null hoặc một giá trị trong tương lai, chẳng hạn như 9999- 12-31, có thể được sử dụng để chỉ ra rằng người hướng dẫn vẫn cư trú tại địa chỉ đó.

Nói chung, dữ liệu thời gian là dữ liệu có khoảng thời gian liên quan trong đó chúng hợp lệ.10

Mô hình hóa dữ liệu thời gian là một vấn đề khó khăn vì nhiều lý do. Ví dụ: giả sử chúng ta có một tập hợp thực thể *giảng viên* mà chúng ta muốn liên kết một địa chỉ thay đổi theo thời gian. Để thêm thông tin thời gian vào một địa chỉ, sau đó chúng ta sẽ phải tạo một thuộc tính đa giá trị, mỗi thuộc tính có giá trị là một giá trị tổng hợp chứa một địa chỉ và một khoảng thời gian. Ngoài các giá trị thuộc tính thay đổi theo thời gian, bản thân các thực thể có thể có thời gian hợp lệ liên quan. Ví dụ: một thực thể sinh viên có thể có thời gian hợp lệ kể từ ngày sinh viên vào trường đại học cho đến ngày sinh viên tốt nghiệp (hoặc rời trường đại học). Các mối quan hệ cũng có thể có thời gian hợp lệ liên quan. Ví dụ, mối quan hệ *prereq* có thể ghi lại khi một khóa học trở thành điều kiện tiên quyết cho một khóa học khác. Do đó, chúng ta sẽ phải thêm các khoảng thời gian hợp lệ vào các giá trị thuộc tính, tập hợp thực thể và tập hợp mối quan hệ. Thêm chi tiết như vậy vào sơ đồ E-R khiến việc tạo và hiểu trở nên rất khó khăn. Đã có một số đề xuất mở rộng ký hiệu E-R đến

10Có những mô hình dữ liệu thời gian khác phân biệt giữa **thời gian hợp lệ và thời**  **gian giao dịch**, sau này ghi lại khi một sự kiện được ghi lại trong cơ sở dữ liệu. Chúng tôi bỏ qua các chi tiết như vậy



**Hình 1.17** Một phiên bản thời gian của mối quan hệ *khóa học*

*chỉ định một cách đơn giản rằng một giá trị thuộc tính hoặc mối quan hệ thay đổi theo thời gian, nhưng không có tiêu chuẩn nào được chấp nhận.*

*Trong thực tế, các nhà thiết kế cơ sở dữ liệu quay trở lại các cách tiếp cận đơn giản hơn để thiết kế cơ sở dữ liệu tiến độ. Một cách tiếp cận thường được sử dụng là thiết kế toàn bộ cơ sở dữ liệu (bao gồm thiết kế E-R và thiết kế quan hệ) bỏ qua các thay đổi tạm thời. Sau đó, nhà thiết kế nghiên cứu các mối quan hệ khác nhau và quyết định mối quan hệ nào cần theo dõi sự thay đổi theo thời gian.*

*Bước tiếp theo là thêm thông tin thời gian hợp lệ vào mỗi mối quan hệ như vậy bằng cách thêm thời gian bắt đầu và kết thúc làm thuộc tính. Ví dụ, hãy xem xét mối quan hệ của khóa học. Tiêu đề của khóa học có thể thay đổi theo thời gian, có thể được xử lý bằng cách thêm một khoảng thời gian hợp lệ; lược đồ kết quả sẽ là:*

*course (course id, title, dept name, credits, start, end)*

*Một ví dụ về mối quan hệ được thể hiện trong Hình 1.17. Mỗi bộ có một khoảng thời gian hợp lệ được liên kết với nó. Lưu ý rằng theo tiêu chuẩn SQL:2011, khoảng thời gian được đóng ở phía bên tay trái, nghĩa là bộ dữ liệu hợp lệ khi bắt đầu thời gian, nhưng mở ở phía bên tay phải, nghĩa là bộ dữ liệu hợp lệ cho đến ngay trước khi kết thúc thời gian, nhưng không hợp lệ khi kết thúc thời gian. Điều này cho phép một bộ có cùng thời gian bắt đầu với thời gian kết thúc của một bộ khác, mà không bị chồng chéo. Nói chung, các điểm cuối bên trái và bên phải được đóng được biểu thị bằng [ và ], trong khi các điểm cuối bên trái và bên phải đang mở được biểu thị bằng ( và ). Các khoảng thời gian trong SQL:2011 có dạng [start, end), nghĩa là chúng được đóng ở bên trái và mở ở bên phải, Lưu ý rằng 9999-12-31 là ngày cao nhất có thể theo tiêu chuẩn SQL.*

*Có thể thấy trong Hình 1.17 rằng tiêu đề của khóa học CS-201 đã thay đổi nhiều lần. Giả sử rằng vào ngày 1 tháng 1 năm 2020, tiêu đề của khóa học được cập nhật lại thành “Giới thiệu về Scala”. Sau đó, giá trị thuộc tính cuối của bộ có tiêu đề “Giới thiệu về Python” sẽ được cập nhật lên 2020-01-01 và một bộ mới (CS-201, Intro. to Scala, Comp. Sci., 4, 2020-01-01, 9999-12-31) sẽ được thêm vào mối quan hệ.*

*Khi chúng tôi theo dõi các giá trị dữ liệu theo thời gian, các phụ thuộc chức năng mà chúng tôi cho là nắm giữ, chẳng hạn như:*

*course id → title, dept name, credits*

có thể không giữ được nữa. Hạn chế sau đây (được thể hiện bằng tiếng Anh) sẽ được giữ nguyên: "Một *id khóa học* chỉ có một *tiêu đề* và giá trị *tên dept* tại bất kỳ thời điểm nào *t*."

Các phụ thuộc chức năng tại một thời điểm cụ thể được gọi là các phụ thuộc chức năng theo thời gian. Chúng tôi sử dụng thuật ngữ ảnh chụp nhanh dữ liệu để chỉ giá trị của dữ liệu tại một thời điểm cụ thể. Do đó, ảnh chụp nhanh dữ liệu *khóa học* cung cấp giá trị của tất cả các thuộc tính, chẳng hạn như tiêu đề và bộ phận, của tất cả các khóa học tại một thời điểm cụ thể.

Về mặt chính thức, sự phụ thuộc chức năng theo thời gian α → β giữ trên sơ đồ quan hệ *r*(*R*) nếu, đối với tất cả các trường hợp pháp lý của *r*(*R*), tất cả các ảnh chụp nhanh của *r* thỏa mãn sự phụ thuộc chức năng α → β.

Khóa chính ban đầu cho một mối quan hệ thời gian sẽ không còn xác định duy nhất một bộ. Chúng tôi có thể cố gắng khắc phục sự cố bằng cách thêm các thuộc tính thời gian bắt đầu và kết thúc vào khóa chính, đảm bảo không có hai bộ có cùng giá trị khóa chính. Tuy nhiên, giải pháp này là không chính xác, vì có thể lưu trữ dữ liệu với các khoảng thời gian hợp lệ chồng chéo, sẽ không bị bắt bằng cách chỉ thêm các thuộc tính thời gian bắt đầu và kết thúc vào ràng buộc khóa chính. Thay vào đó, phiên bản tạm thời của ràng buộc khóa phải đảm bảo rằng nếu bất kỳ hai bộ nào có cùng giá trị khóa chính, khoảng thời gian hợp lệ của chúng không trùng lặp. Chính thức, nếu *r*.*A* là khóa chính thời gian của mối quan hệ *r*, thì bất cứ khi nào hai bộ *t*1 và *t*2 trong *r* sao cho *t*1.*A* = *t*2.*A*, giá trị của chúng khoảng thời gian của *t*1 và *t*2 không được trùng nhau.

Các ràng buộc khóa ngoại cũng phức tạp hơn khi mối quan hệ tham chiếu là một mối quan hệ thời gian. Một khóa ngoại thời gian phải đảm bảo rằng không chỉ mỗi bộ trong mối quan hệ tham chiếu, giả sử *r*, có một bộ phù hợp trong mối quan hệ tham chiếu, giả sử *s*, mà cả khoảng thời gian của chúng cũng được tính đến. Không bắt buộc phải có một bộ phù hợp trong *s* với chính xác cùng một khoảng thời gian, cũng không phải là một bộ duy nhất trong *s* có một khoảng thời gian chứa khoảng thời gian của *r* tuple. Thay vào đó, chúng tôi cho phép khoảng thời gian của bộ *r* được bao phủ bởi một hoặc nhiều bộ. Về mặt chính thức, một khoảng thời gian ràng buộc khóa ngoại từ *r*.*A* đến *s*.*B* đảm bảo những điều sau: cho mỗi bộ *t* trong *r*, với khoảng thời gian hợp lệ (*l*, *u*), có một tập con *st* của một hoặc nhiều bộ trong *s* sao cho mỗi bộ *si* ∈ *st* có *si*.*B* = *t*.*A*, và tiếp theo là sự kết hợp của các khoảng thời gian của tất cả các *si* chứa (*l*, *u*).

Một bản ghi trong bảng điểm của học viên nên đề cập đến tiêu đề khóa học tại thời điểm học viên tham gia khóa học. Do đó, quan hệ tham chiếu cũng phải ghi lại thời gian hình thành, để xác định một bản ghi cụ thể từ quan hệ *khóa học*. Trong lược đồ đại học của chúng tôi, *lấy*. *id khóa học* là một khóa *học tham khảo khóa học* nước ngoài. Giá trị *năm* và *học kỳ* của một bộ *có* thể được ánh xạ đến một ngày đại diện, chẳng hạn như ngày bắt đầu của học kỳ; giá trị ngày kết quả có thể được sử dụng để xác định một bộ trong phiên bản thời gian của quan hệ *khóa học* có khoảng thời gian hợp lệ chứa ngày được chỉ định. Về cơ bản, một bộ có thể được liên kết với một khoảng thời gian hợp lệ từ ngày bắt đầu học kỳ cho đến ngày kết thúc học kỳ, và các bộ *khóa* học có *id khóa* học phù hợp và thời gian hợp lệ chồng chéo có thể được truy xuất; miễn là các bộ *khóa* học không được cập nhật trong một học kỳ, sẽ chỉ có một bản ghi như vậy.

Thay vì thêm thông tin thời gian vào mỗi mối quan hệ, một số nhà thiết kế cơ sở dữ liệu tạo cho mỗi mối quan hệ một mối quan hệ *lịch* sử tương ứng lưu trữ lịch sử cập nhật cho các bộ. Ví dụ, một nhà thiết kế có thể để lại mối quan hệ *khóa học* không thay đổi, nhưng tạo *lịch sử khóa học* quan hệ có chứa tất cả các thuộc tính của *khóa học*, với thuộc tính *dấu thời gian* tùy theo điều kiện cho biết khi nào một bản ghi được thêm vào bảng *lịch sử khóa học* . Tuy nhiên, một chương trình như vậy có những hạn chế, chẳng hạn như không có khả năng liên kết một bản ghi *take* với tiêu đề khóa học chính xác.

Tiêu chuẩn SQL:2011 đã thêm hỗ trợ cho dữ liệu thời gian. Đặc biệt, nó cho phép các thuộc tính hiện có được khai báo để chỉ định một khoảng thời gian hợp lệ cho một bộ. Ví dụ, đối với mối quan hệ *khóa học* mở rộng mà chúng ta đã thấy ở trên, chúng ta có thể khai báo

period for validtime (start, end)

để chỉ định rằng bộ dữ liệu hợp lệ trong khoảng thời gian được chỉ định bởi *bắt đầu* và *kết thúc* (là các thuộc tính thông thường khác).

Các khóa chính tạm thời có thể được khai báo trong SQL:2011, như được minh họa bên dưới, sử dụng sơ đồ *khóa học* mở rộng:

primary key (course id, validtime without overlaps)

SQL:2011 cũng hỗ trợ các ràng buộc khóa ngoại thời gian cho phép một khoảng thời gian được chỉ định cùng với các thuộc tính quan hệ tham chiếu, cũng như với các thuộc tính quan hệ tham chiếu. Hầu hết các cơ sở dữ liệu, ngoại trừ IBM DB2, Teradata và một vài cơ sở dữ liệu khác, không hỗ trợ các ràng buộc khóa chính thời gian. Theo hiểu biết tốt nhất của chúng tôi, hiện tại không có hệ thống cơ sở dữ liệu nào hỗ trợ các ràng buộc khóa nước ngoài tạm thời (Teradata cho phép chỉ định chúng, nhưng ít nhất là vào năm 2018, không thực thi chúng). Một số cơ sở dữ liệu không hỗ trợ trực tiếp các ràng buộc khóa chính thời gian cách giải quyết thấp để thực thi các ràng buộc đó. Ví dụ, mặc dù Postgre SQL không hỗ trợ các ràng buộc khóa chính thời gian nguyên bản, các ràng buộc như vậy có thể được thực hiện bằng cách sử dụng tính năng ràng buộc loại trừ được hỗ trợ bởi Postgre SQL. Ví dụ: hãy xem xét mối quan hệ *khóa học*, có khóa chính là *id khóa học*. Trong SQL Postgre, chúng ta có thể thêm một thời gian hợp *lệ* thuộc tính, thuộc loại tsrange; loại dữ liệu tsrange của SQL Postgre lưu trữ một phạm vi dấu thời gian với dấu thời gian bắt đầu và kết thúc. PostgreSQL hỗ trợ &&oper- ator trên một cặp phạm vi, trả về true nếu hai phạm vi chồng chéo và false nếu không. Khóa chính thời gian có thể được thực thi bằng cách thêm ràng buộc loại trừ sau (một loại ràng buộc được hỗ trợ bởiSQL) vào quan hệ *khóa học* như sau:

exclude (course id with =, validtime with &&)

Ràng buộc trên đảm bảo rằng nếu hai bộ *khóa học* có cùng giá trị *id khóa học*, thì khoảng thời gian hợp *lệ* của chúng không trùng lặp.

Các phép toán đại số quan hệ, chẳng hạn như select, project hoặc join, có thể được mở rộng để lấy các quan hệ thời gian làm đầu vào và tạo các quan hệ thời gian làm đầu ra. Các hoạt động lựa chọn và chiếu trên các bộ đầu ra quan hệ thời gian có thời gian hợp lệ giữa các lần giống như các bộ đầu vào tương ứng của chúng. Một tham gia thời gian là hơi khác nhau: thời gian hợp lệ của một bộ trong kết quả tham gia được định nghĩa là giao điểm của thời gian hợp lệ của các bộ mà từ đó nó được bắt nguồn. Nếu thời gian hợp lệ không giao nhau, bộ này sẽ bị loại bỏ khỏi kết quả. Theo hiểu biết tốt nhất của chúng tôi, không có cơ sở dữ liệu nào hỗ trợ kết nối tạm thời nguyên bản, mặc dù chúng có thể được thể hiện bằng các truy vấn SQL một cách rõ ràng xử lý các điều kiện thời gian. Các vị từ, chẳng hạn như *chồng chéo, chứa, trước và sau và* các hoạt động như *giao lộ* và *sự khác biệt* trên các cặp khoảng thời gian được hỗ trợ bởi một số hệ thống cơ sở dữ liệu.

## 1.11. Tóm tắt

• Chúng tôi đã chỉ ra những cạm bẫy trong thiết kế cơ sở dữ liệu và cách thiết kế hệ thống lược đồ cơ sở dữ liệu theo cách tránh những cạm bẫy đó. Những cạm bẫy bao gồm thông tin lặp đi lặp lại và không có khả năng đại diện cho một số thông tin.

• Chương 6 cho thấy sự phát triển của thiết kế cơ sở dữ liệu quan hệ từ E-R thiết kế và khi nào các sơ đồ có thể được kết hợp một cách an toàn.

• Phụ thuộc chức năng là các ràng buộc nhất quán được sử dụng để xác định hai dạng chuẩn được sử dụng rộng rãi, dạng chuẩn Boyce – Codd (BCNF) và dạng chuẩn thứ ba (3NF).

• Nếu sự phân hủy là bảo toàn phụ thuộc, tất cả các phụ thuộc chức năng có thể được suy ra một cách logic bằng cách chỉ xem xét các phụ thuộc áp dụng cho một mối quan hệ. Điều này cho phép kiểm tra tính hợp lệ của một bản cập nhật mà không cần phải tính toán sự tham gia của các mối quan hệ trong quá trình phân hủy.

• Bảo hiểm chính tắc là một tập hợp các phụ thuộc chức năng tương đương với một tập hợp các phụ thuộc chức năng nhất định, được giảm thiểu theo một cách cụ thể để loại bỏ các thuộc tính không liên quan.

• Thuật toán để phân hủy các mối quan hệ thành BCNF đảm bảo một sự phân hủy không mất mát. Có các lược đồ quan hệ với một tập hợp các phụ thuộc chức năng nhất định mà không có sự phân hủy BCNF duy trì phụ thuộc.

• Một nắp chính tắc được sử dụng để phân tách sơ đồ quan hệ thành 3NF, đây là một sự nới lỏng nhỏ của điều kiện BCNF. Thuật toán này tạo ra các thiết kế vừa không mất mát vừa duy trì sự phụ thuộc. Các mối quan hệ trong 3NF có thể có một số dư thừa, nhưng đó được coi là một sự đánh đổi chấp nhận được trong trường hợp không có sự phân hủy duy trì sự phụ thuộc vào BCNF.

• Các phụ thuộc đa giá trị chỉ định các ràng buộc nhất định không thể được chỉ định chỉ với các phụ thuộc chức năng. Dạng pháp tuyến thứ tư (4NF) được xác định bằng cách sử dụng khái niệm phụ thuộc đa giá trị. Mục 28.1.1 cung cấp chi tiết về lý luận về các phụ thuộc đa giá trị.

• Các hình thức bình thường khác tồn tại, bao gồm PJNF và DKNF, loại bỏ các hình thức dự phòng tinh tế hơn. Tuy nhiên, chúng rất khó sử dụng và hiếm khi được sử dụng. Chương 28 cung cấp chi tiết về các biểu mẫu thông thường này. Hình thức bình thường thứ hai chỉ có lợi ích lịch sử vì nó không mang lại lợi ích nào trên 3NF.

• Thiết kế quan hệ thường dựa trên các miền nguyên tử đơn giản cho mỗi thuộc tính. Đây được gọi là biểu mẫu bình thường đầu tiên.

• Thời gian đóng một vai trò quan trọng trong các hệ thống cơ sở dữ liệu. Cơ sở dữ liệu là mô hình của thế giới thực. Trong khi hầu hết các cơ sở dữ liệu mô hình hóa trạng thái của thế giới thực tại một thời điểm (tại thời điểm hiện tại), cơ sở dữ liệu thời gian mô hình hóa các trạng thái của thế giới thực theo thời gian.

• Có thể có những thiết kế cơ sở dữ liệu xấu mặc dù không mất mát, bảo tồn sự phụ thuộc và ở dạng bình thường thích hợp. Chúng tôi đã trình bày các ví dụ về một số thiết kế như vậy để minh họa rằng chuẩn hóa dựa trên phụ thuộc chức năng, mặc dù rất quan trọng, không phải là khía cạnh duy nhất của thiết kế quan hệ tốt.

• Để cơ sở dữ liệu lưu trữ không chỉ dữ liệu hiện tại mà cả dữ liệu lịch sử, cơ sở dữ liệu cũng phải lưu trữ cho mỗi bộ dữ liệu đó trong khoảng thời gian mà bộ dữ liệu đó hợp lệ hoặc hợp lệ. Sau đó, nó trở nên cần thiết để xác định dependencies chức năng thời gian để đại diện cho ý tưởng rằng sự phụ thuộc chức năng nắm giữ tại bất kỳ thời điểm nào nhưng không phải trên toàn bộ mối quan hệ. Tương tự, thao tác nối cần được sửa đổi để chỉ nối các bộ với các khoảng thời gian chồng chéo một cách thích hợp.

• Khi xem xét các vấn đề trong chương này, lưu ý rằng lý do chúng ta có thể xác định các cách tiếp cận nghiêm ngặt đối với thiết kế cơ sở dữ liệu quan hệ là mô hình dữ liệu quan hệ dựa trên nền tảng toán học vững chắc. Đó là một trong những ưu điểm chính của mô hình quan hệ so với các mô hình dữ liệu khác mà chúng tôi đã nghiên cứu.

# CHƯƠNG 2: TÌM HIỂU VÀ GIỚI THIỆU CÔNG CỤ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU - PostgreSQL

## 2.1. Giới thiệu về PostgreSQL



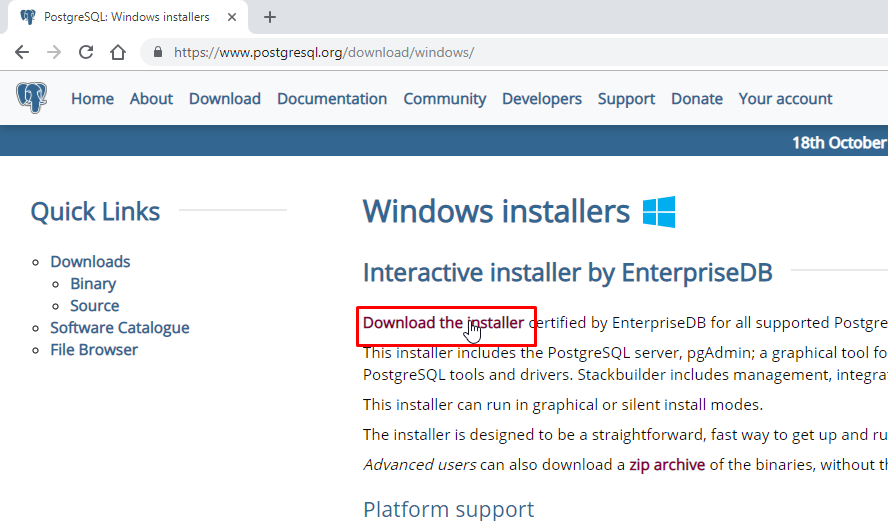
*Hình 2.1: Logo hệ quản trị cơ sở dữ liệu PostgreSQL*

* PostgreSQL là một hệ thống quản trị cơ sở dữ liệu quan hệ-đối tượng (object-relational database management system) có mục đích chung, hệ thống cơ sở dữ liệu mã nguồn mở tiên tiến nhất hiện nay.
* PostgreSQL được phát triển từ năm 1986 như một phần của dự án POSTGRES tại Đại học California ở Berkeley và cho tới nay đã có hơn 35 năm phát triển tích cực trên nền tảng cốt lõi.
* PostgreSQL ban đầu được thiết kế để chạy trên các nền tảng tương tự UNIX. Tuy nhiên, PostgreSQL sau đó cũng được điều chỉnh linh động để có thể chạy được trên nhiều nền tảng khác nhau như Mac OS X, Solaris và Windows.
* Ngoài việc là một hệ quản trị cơ sở dữ liệu, PostgreSQL còn là một phần mềm mã nguồn mở miễn phí. Mã nguồn của phần mềm khả dụng theo license của PostgreSQL, một license nguồn mở tự do. Theo đó, bạn sẽ được tự do sử dụng, sửa đổi và phân phối PostgreSQL dưới mọi hình thức.
* Một điều nữa là PostgreSQL không yêu cầu quá nhiều công tác bảo trì bởi có tính ổn định cao. Do đó, nếu bạn phát triển các ứng dụng dựa trên PostgreSQL, chi phí sở hữu sẽ thấp hơn so với các hệ thống quản trị dữ liệu khác.

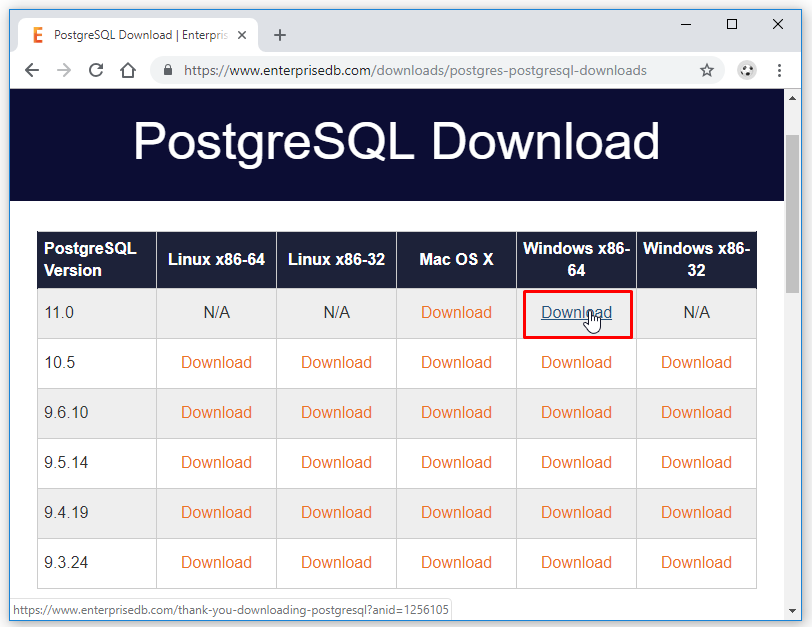
## 2.2. Cài đặt và cấu hình PostgreSQL

Hướng dẫn cài đặt và cấu hình PostgreSQL trên Windows:

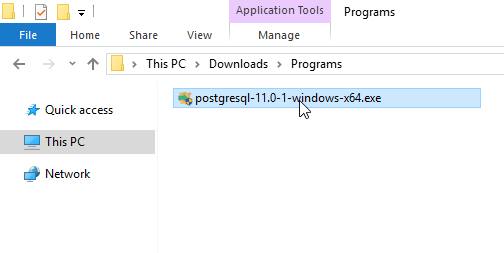
* **Bước 1: Download PostgreSQL**
  + Truy cập <https://www.postgresql.org/download/windows/> để download bản PostgreSQL cho hệ điều hành Windows.
  + Ở đây mình sẽ download phiên bản cài đặt.



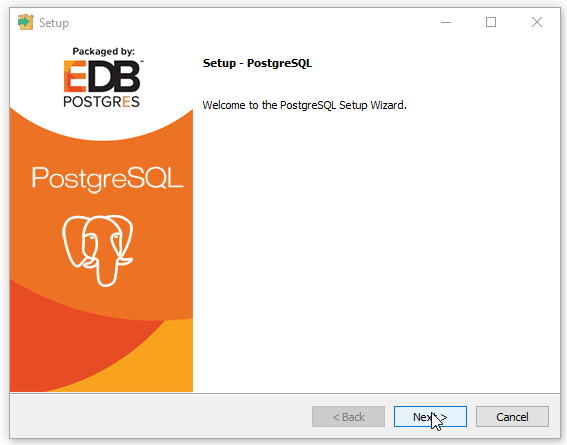
* + Lựa chọn version bạn muốn cài đặt, máy mình là windows 64 bit nên mình sẽ cài phiên bản Windows x86-64



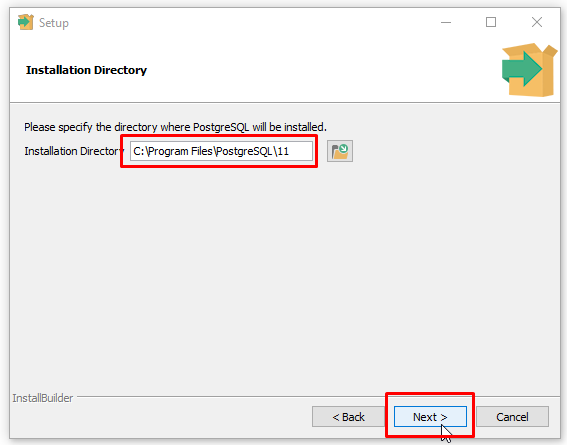
* + Click đúp vào file .exe vừa tải về.



* **Bước 2: Cài đặt PostgreSQL**
  + Nhập các thông tin cấu hình PostgreSQL khi cài đặt:



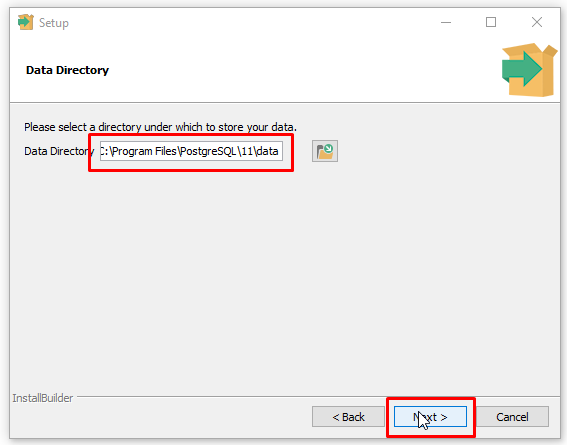
* + Chọn nơi cài đặt PostgreSQL



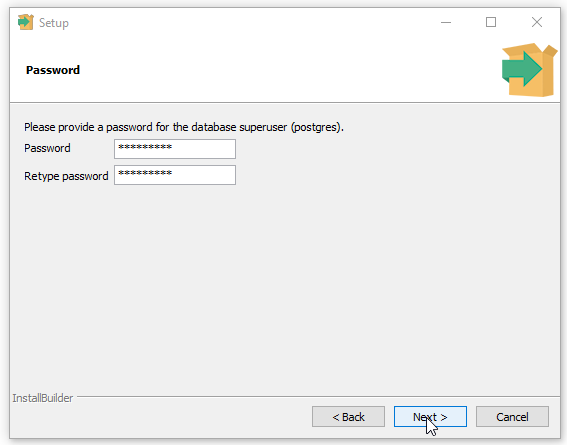
* + Chọn các component cài đặt (nếu bạn chỉ cần server để lưu và truy vấn dữ liệu thì chỉ cần chọn PostgreSQL Server). Ở đây mình chọn tất cả.



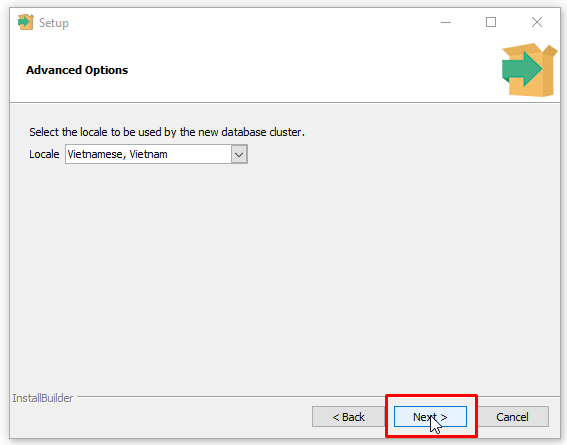
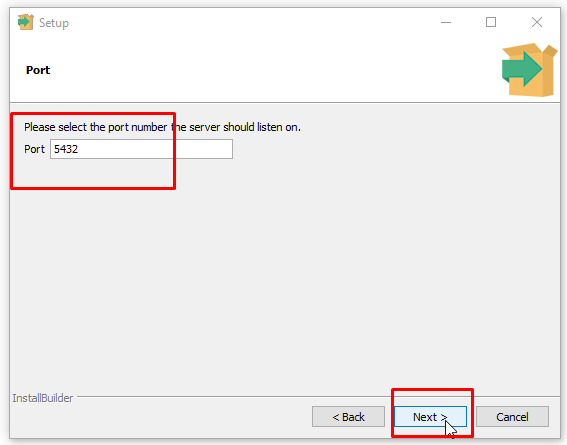
* + Chọn nơi lưu trữ data (database của bạn sẽ được lưu tại đây)



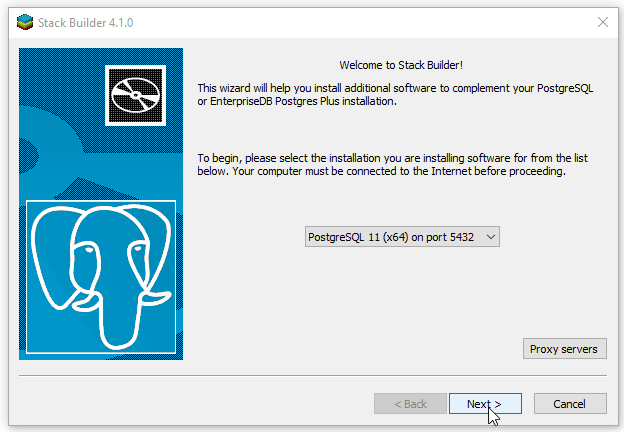
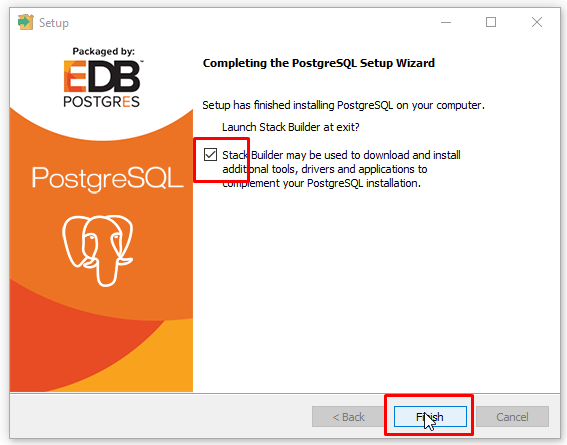
* + Nhập mật khẩu cho user postgres (postgres là user mặc định, có quyền cao nhất – superuser)



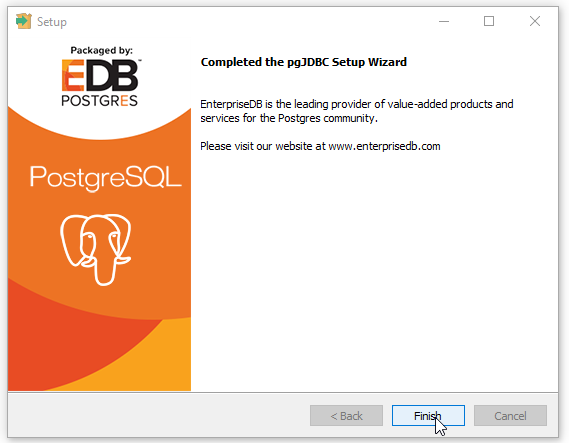
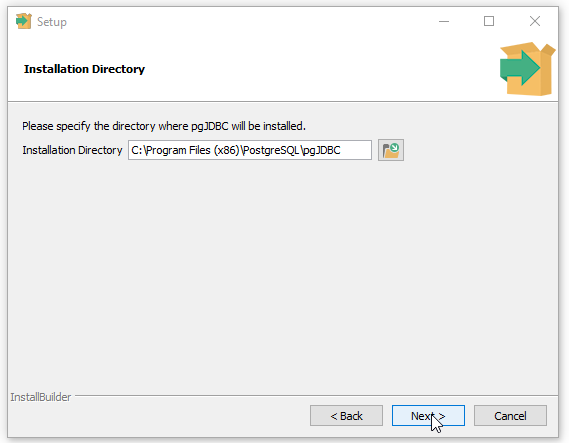
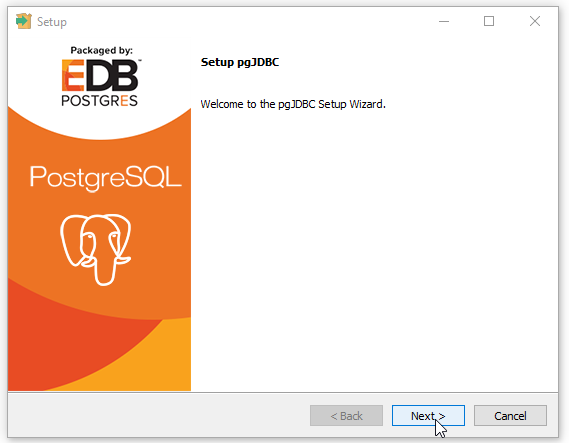
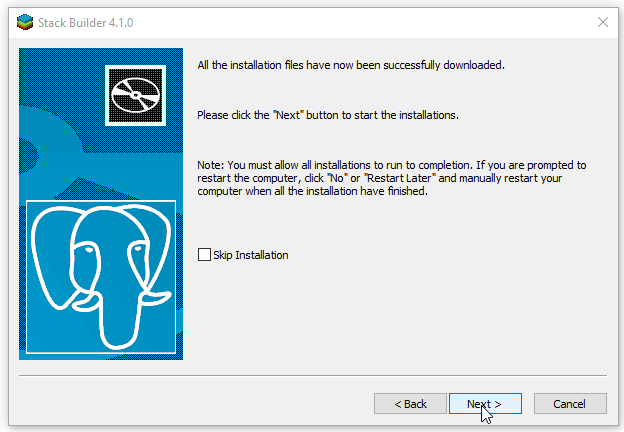
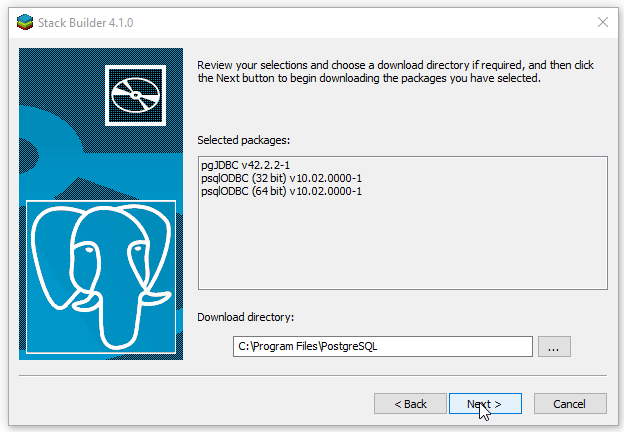
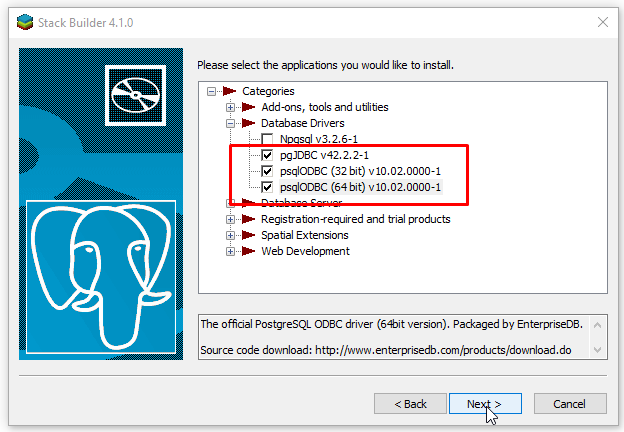
* + Chọn cổng / port cho PostgreSQL (port mặc định cho postgreSQL là 5432, nếu chưa bị trùng với port nào thì các bạn cứ để mặc định cho dễ nhớ nhé)



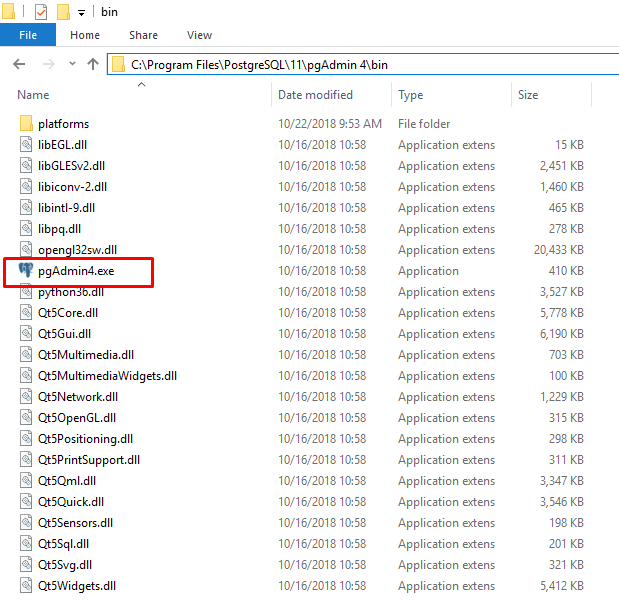
* + Tick vào ô cài đặt Stack Builder để download và cấu hình các driver.



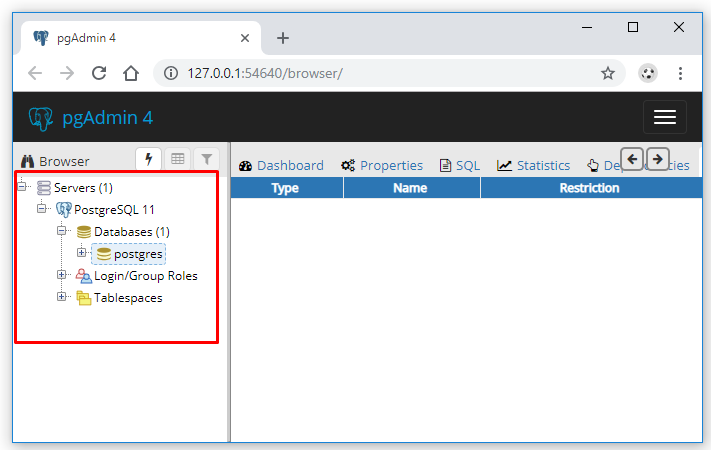
* + Chọn driver để cài đặt (cho phép kết nối với Postgre qua các ngôn ngữ khác)



* + Như thế là đã hoàn thành cài đặt PostgreSQL.
* **Bước 3: Mở PostgreSQL**
  + Bây giờ chúng ta mở pgAdmin 4 nên và kết nối tới PostgreSQL server.



* + pgAdmin 4 được chạy trên server, nên khi click vào nó sẽ tự động mở giao diện pgAdmin trên trình duyệt hoặc các bạn cũng có thể truy cập qua url <http://127.0.0.1:54640/browser/> (Nếu nó bắt đăng nhập thì các bạn nhập account postgres + mật khẩu đã tạo bên trên nhé.



## 2.4. Các tính năng quan trọng của PostgreSQL

* **Công cụ PostgreSQL cung cấp một số tính năng cơ bản quan trọng sau đây:**

+ Hỗ trợ chuẩn SQL: PostgreSQL tuân thủ chuẩn SQL của ANSI và cung cấp một số tính năng mạnh mẽ như truy vấn SELECT, INSERT, UPDATE, DELETE và JOIN để thao tác dữ liệu trong cơ sở dữ liệu.

+ Kiểu dữ liệu đa dạng: PostgreSQL cung cấp một loạt các kiểu dữ liệu, bao gồm kiểu nguyên thủy như số nguyên, số thực, Boolean, ký tự và kiểu ngày tháng, cùng với kiểu dữ liệu phức tạp như mảng, JSON, hình học địa lý, XML và nhiều hơn nữa. Điều này cho phép lưu trữ và xử lý dữ liệu đa dạng theo nhu cầu của ứng dụng.

+ Chỉ mục và tìm kiếm: PostgreSQL hỗ trợ các chỉ mục để tăng tốc độ truy vấn và tìm kiếm dữ liệu. Nó cung cấp các loại chỉ mục như b-tree, hash, GiST (Generalized Search Tree) và GIN (Generalized Inverted Index), cho phép tối ưu hóa truy vấn và tìm kiếm dữ liệu hiệu quả.

+ Ràng buộc và giao thức: PostgreSQL cho phép xác định ràng buộc dữ liệu như ràng buộc khóa ngoại, ràng buộc duy nhất và ràng buộc kiểm tra để đảm bảo tính toàn vẹn dữ liệu. Ngoài ra, nó hỗ trợ các giao thức ACID (Atomicity, Consistency, Isolation, Durability) để đảm bảo tính nhất quán và an toàn của giao dịch.

+ Truy vấn phức tạp: PostgreSQL cung cấp một loạt các tính năng truy vấn phức tạp như truy vấn JOIN, truy vấn con, truy vấn phân tích và truy vấn chức năng. Nó hỗ trợ cú pháp phong phú và mạnh mẽ, cho phép truy vấn và phân tích dữ liệu theo các yêu cầu phức tạp.

+ Hỗ trợ đa ngôn ngữ: PostgreSQL hỗ trợ nhiều ngôn ngữ lập trình như C/C++, Java, Python, Ruby và nhiều ngôn ngữ khác thông qua các giao diện lập trình ứng dụng (API) như libpq, JDBC và ODBC. Điều này cho phép phát triển ứng dụng dễ dàng trên nhiều nền tảng và ngôn ngữ khác nhau.

## 2.5. Đánh giá về công cụ POSTGRESQL

* **Ưu điểm:**
* PostgreSQL nổi tiếng với tính ổn định và độ tin cậy cao. Nó hỗ trợ các tính năng ACID (Atomicity, Consistency, Isolation, Durability) để đảm bảo tính toàn vẹn của dữ liệu.
* PostgreSQL tuân thủ chuẩn SQL nghiêm ngặt, giúp đơn giản hóa việc chuyển đổi từ các hệ thống quản lý cơ sở dữ liệu khác.
* Với cộng đồng người dùng rộng lớn, bạn có thể dễ dàng tìm thấy tài liệu, hướng dẫn và hỗ trợ từ cộng đồng.
* PostgreSQL là mã nguồn mở, miễn phí sử dụng và phát triển. Điều này có nghĩa là có một cộng đồng lớn hỗ trợ và phát triển liên tục, cũng như khả năng tuỳ chỉnh mở rộng theo nhu cầu cụ thể.
* **Nhược điểm:**
* Mặc dù PostgreSQL có hiệu suất tốt, nhưng trong một số trường hợp cụ thể, nhất là khi xử lý lượng dữ liệu lớn, có thể cần tinh chỉnh để tối ưu hóa hiệu suất.
* PostgreSQL có thể khó sử dụng đối với người mới bắt đầu hoặc không có kinh nghiệm với hệ thống quản lý cơ sở dữ liệu.
* PostgreSQL có thể tiêu tốn một số tài nguyên hệ thống khá lớn, đặc biệt khi xử lý các tải đồng thời lớn hoặc yêu cầu hiệu suất cao.

## 2.6. Các lợi ích của việc sử dụng công cụ POSTGRESQL

- Được sử dụng để chạy trang web và ứng dụng web động.

- Cho phép lưu lại nhật ký và hình thành cơ sở dữ liệu hỗ trợ sửa lỗi.

- Mã nguồn mở để người dùng chỉnh sửa, nâng cấp.

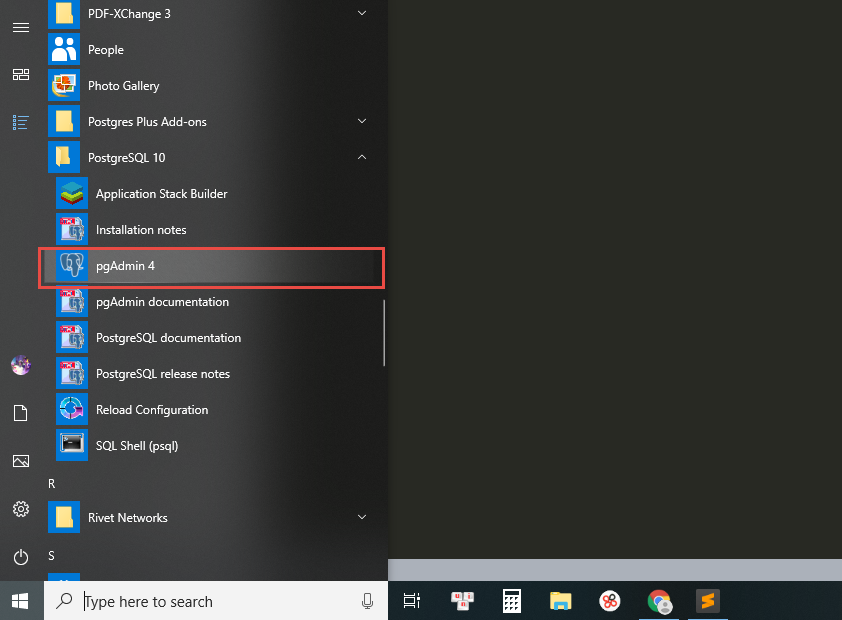
- Hỗ trợ khách hàng dựa trên vùng địa lý.

- Cách sử dụng đơn giản.

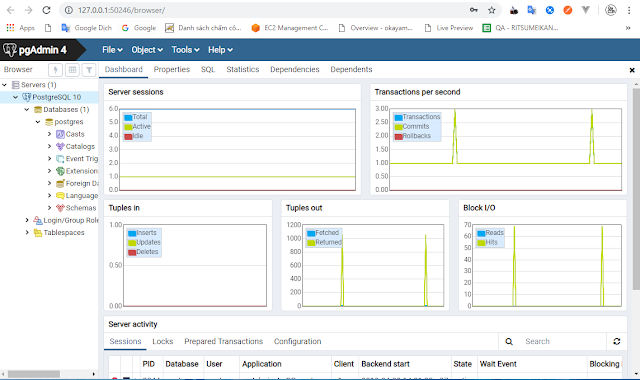
- PostgreSQL có một bộ tính năng rất mạnh. Điều này bao gồm khả năng Multi-Version Concurrency Control ( MVCC) và hiệu suất cơ sở dữ liệu thu hồi tại thời điểm của nó. Cơ sở dữ liệu cũng có các điều khiển truy cập chi tiết, không gian bảng và khả năng sao chép không đồng bộ.

## 2.7. Cách sử dụng công cụ POSTGRESQL

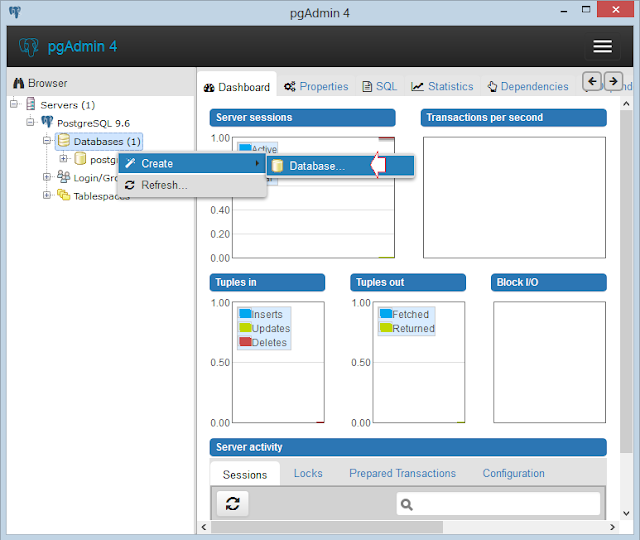
* Ở bước trên bạn đã cài đặt thành công PostGresql, bây giờ bạn có thể mở và sử dụng nó  
  Vào cửa sổ start ở góc trái màn hình máy tính

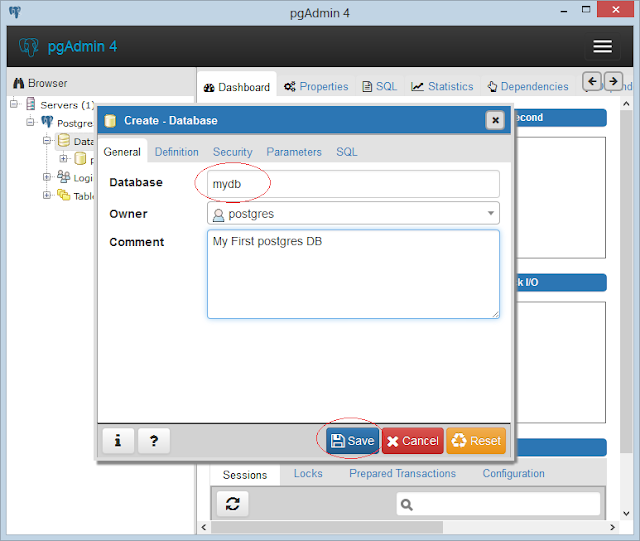


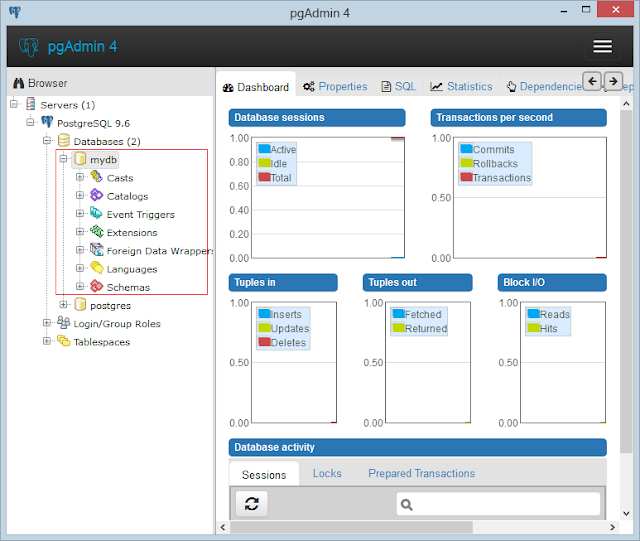
* Chạy chương trình pgAdmin 4 nó sẽ yêu cầu mật khẩu, nhập mật khẩu ở bước 1 cài đặt vào được màn hình như dưới.



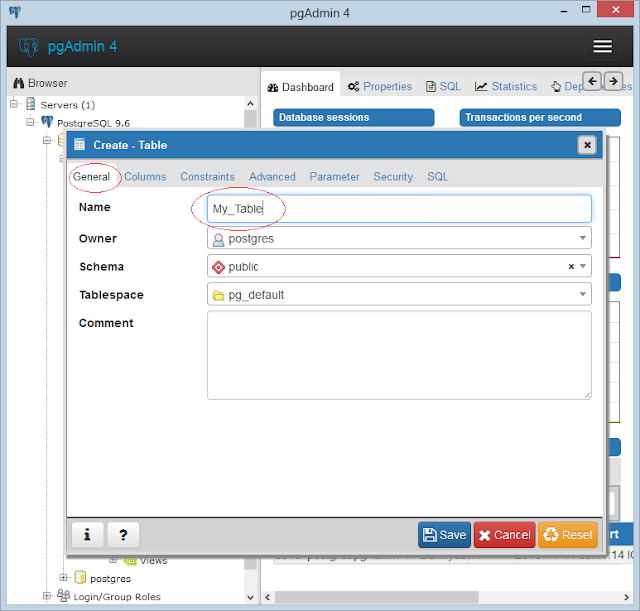
* Tạo database như sau:

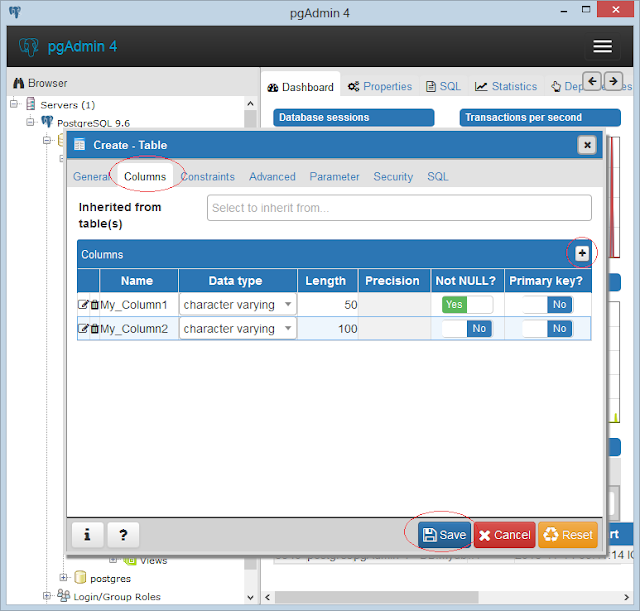


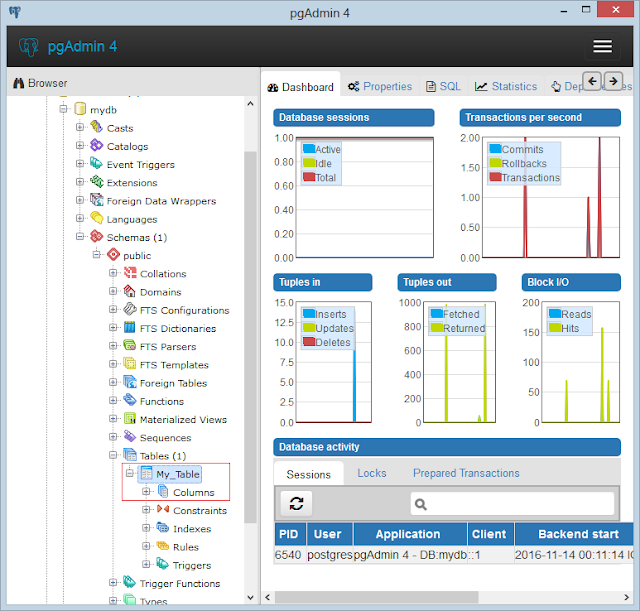




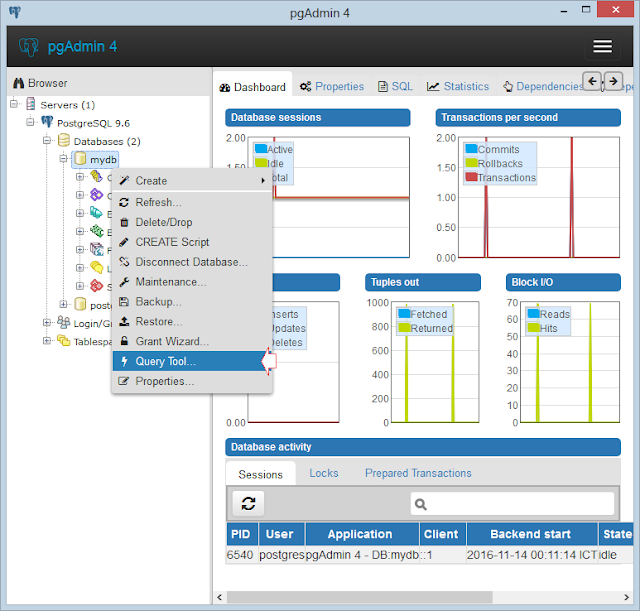
* Cách tạo bảng:



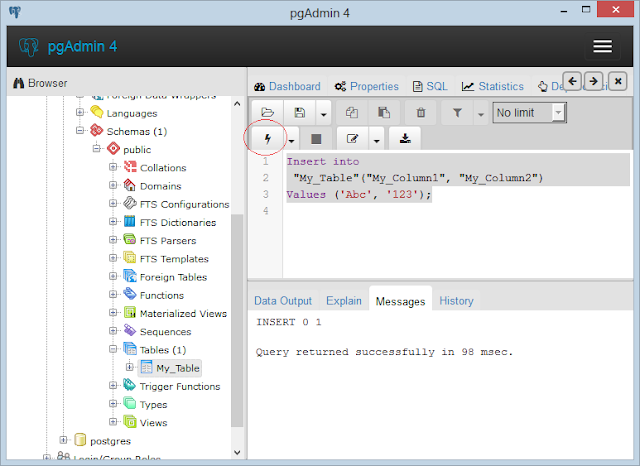


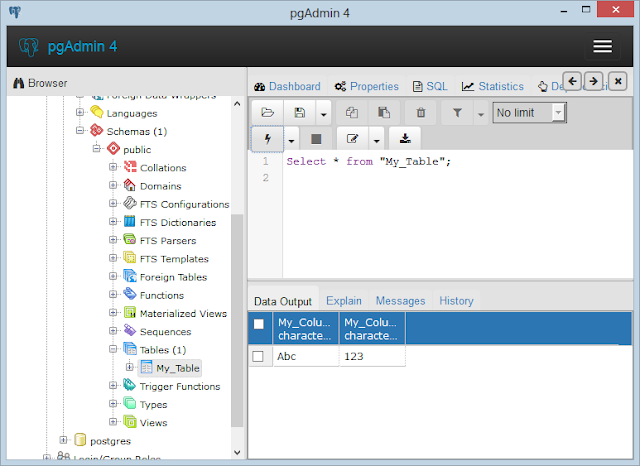


* Công cụ truy vấn



|  |  |
| --- | --- |
| 1  2  3  4  5 | Insert into    "My\_Table"("My\_Column1", "My\_Column2")    Values ('Abc', '123'); |

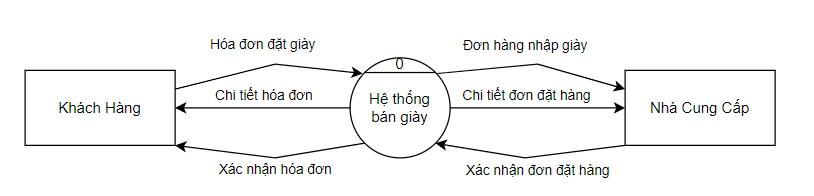




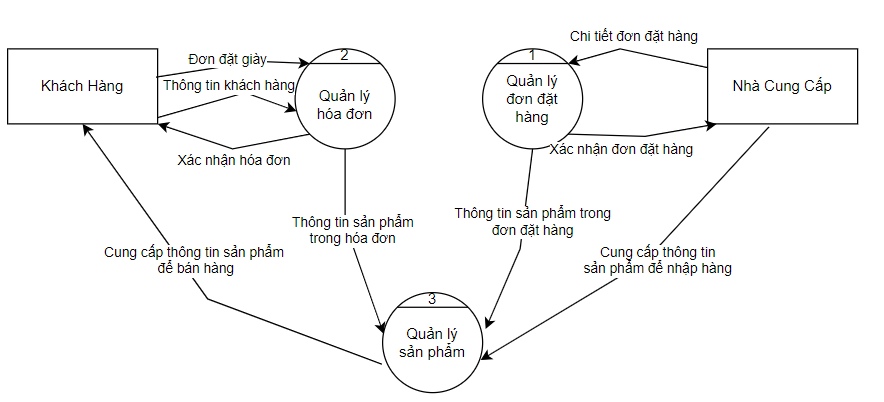
# CHƯƠNG 3: THIẾT KẾ CƠ SỞ DỮ LIỆU CHO CỬA HÀNG BÁN GIÀY

## 3.1. Sơ đồ luồng dữ liệu (DFD) biểu diễn quy trình của cửa hàng bán giày

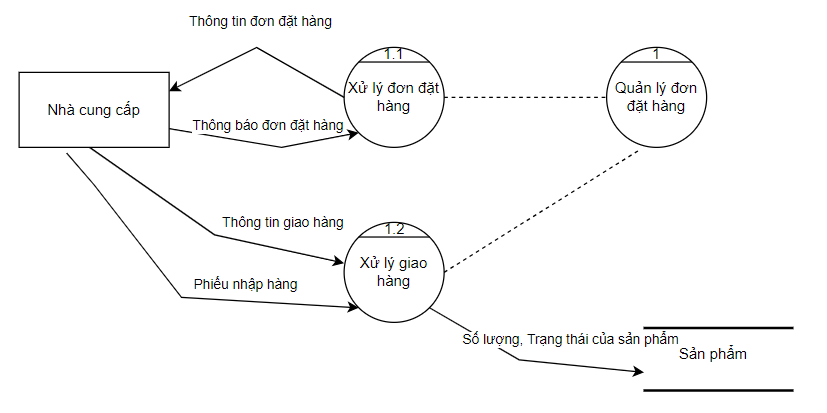
**DFD ở mức 0**

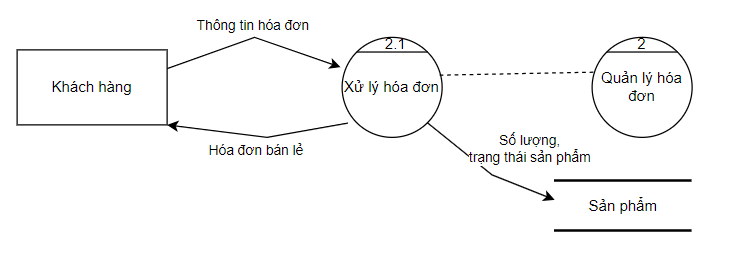
****

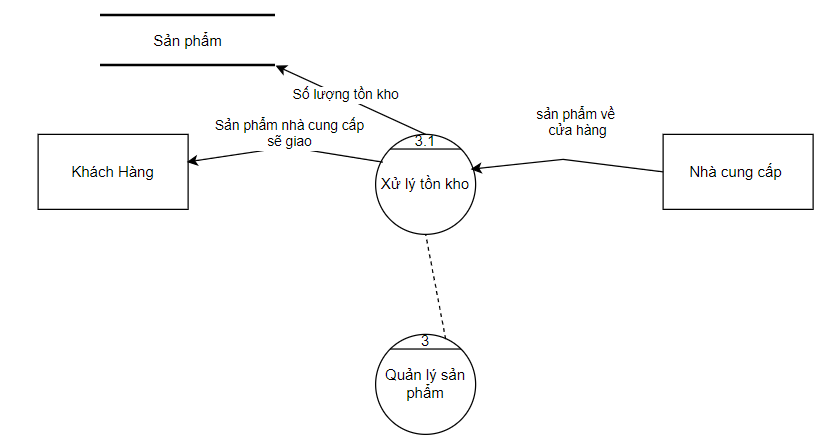
**DFD ở mức 1**

****

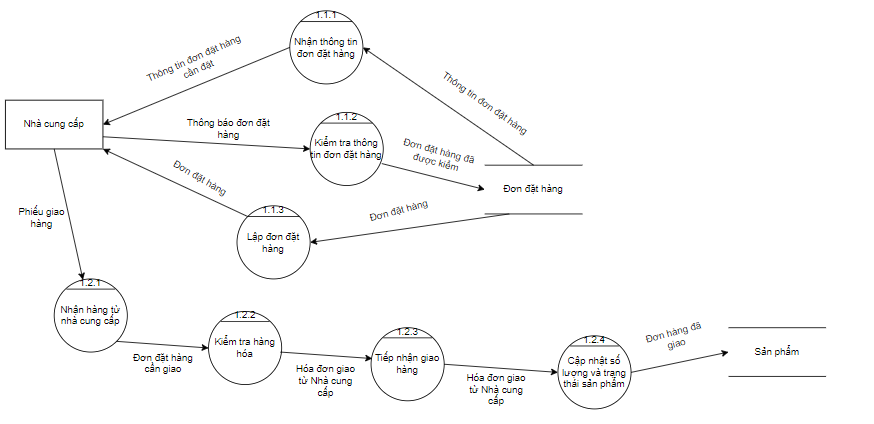
**DFD ở mức 2**

****

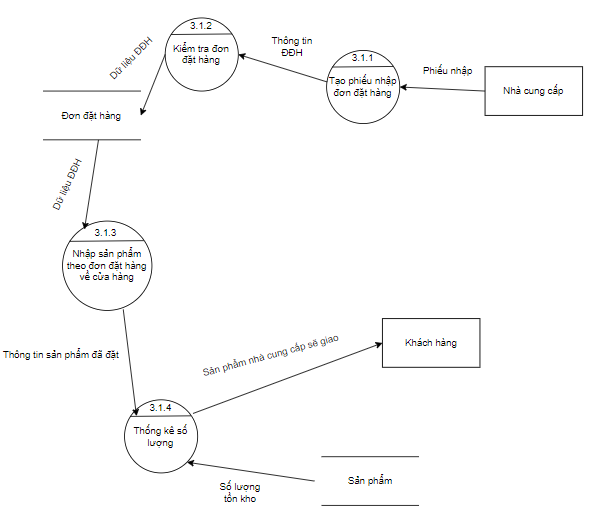
****

****

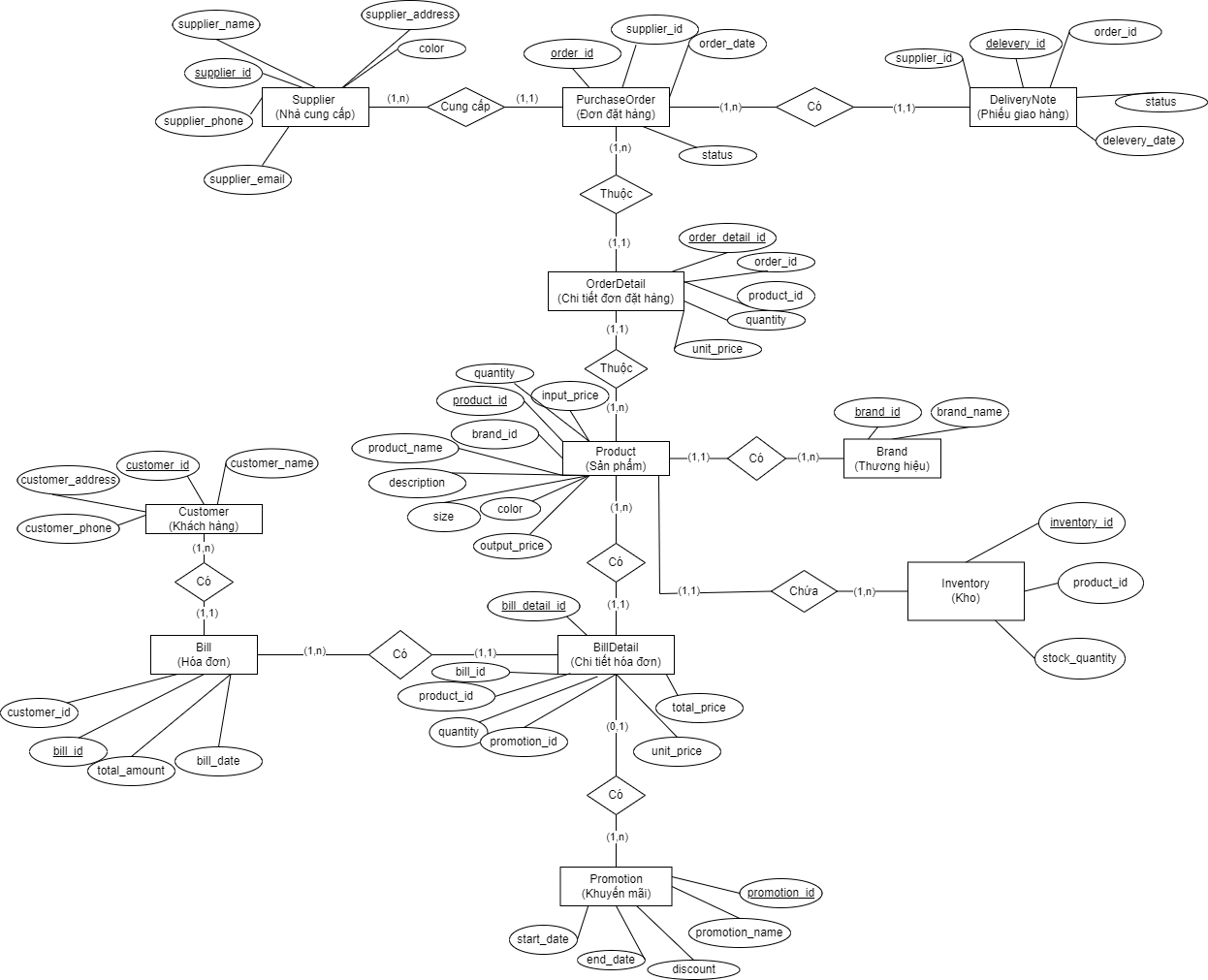
**DFD ở mức 3**

****

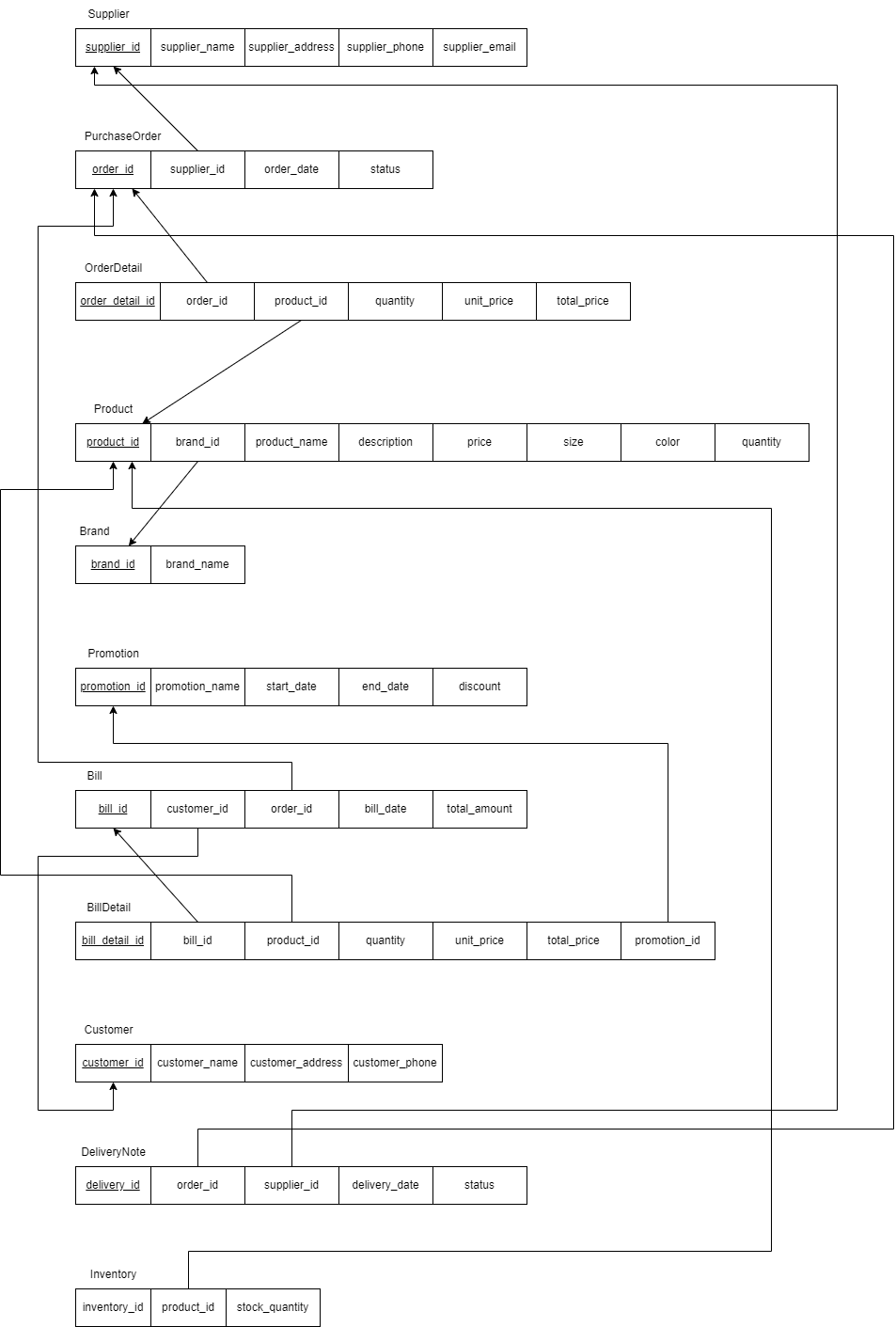
****

****

## 3.2. Sơ đồ ERD

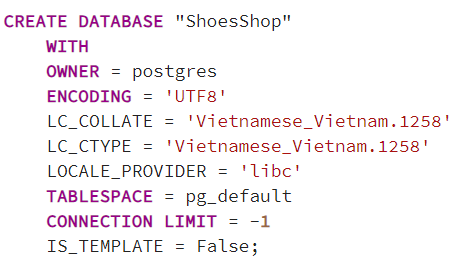
****

## 3.3. Chuyển sơ đồ ERD sang RDM

****

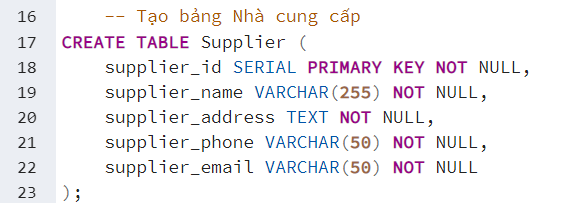
## 3.4. Thiết kế cơ sở dữ liệu

### 3.4.1. Tạo Database

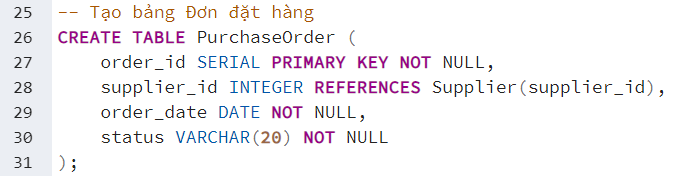
****

### 3.4.2. Tạo bảng và các khóa chính, khóa ngoại

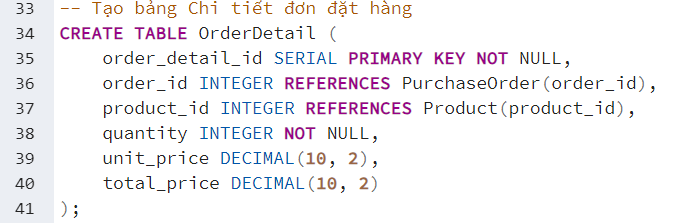
#### 3.4.2.1. Tạo bảng nhà cung cấp

****

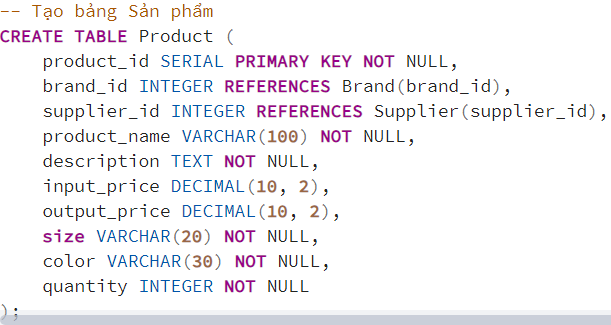
#### 3.4.2.2. Tạo bảng đơn đặt hàng

****

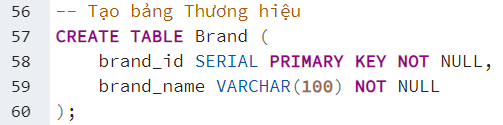
#### 3.4.2.3. Tạo bảng chi tiết đơn đặt hàng

****

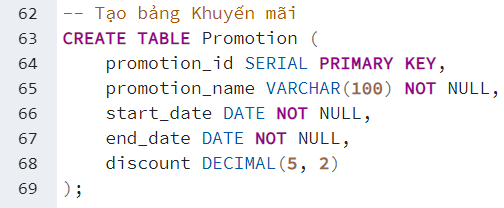
#### 3.4.2.4. Tạo bảng sản phẩm

****

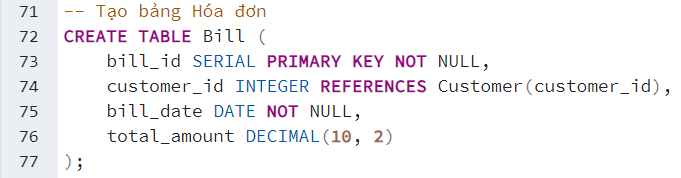
#### 3.4.2.5. Tạo bảng thương hiệu

****

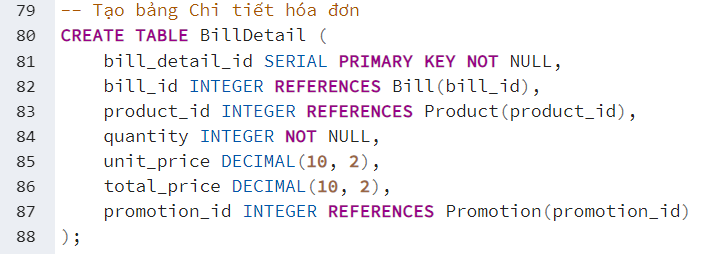
#### 3.4.2.6. Tạo bảng khuyến mãi

****

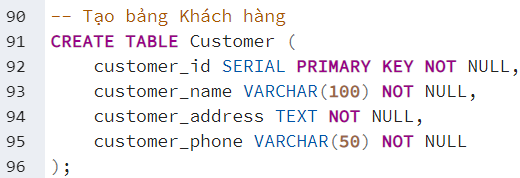
#### 3.4.2.7. Tạo bảng hóa đơn

****

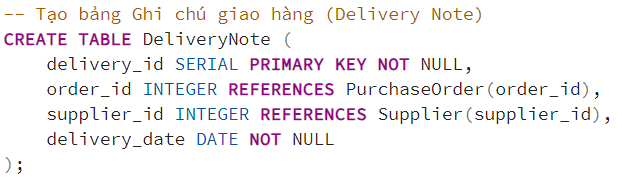
#### 3.4.2.8. Tạo bảng chi tiết hóa đơn

****

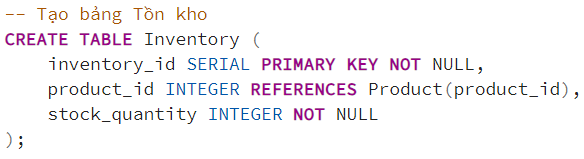
#### 3.4.2.9. Tạo bảng khách hàng

****

#### 3.4.2.10. Tạo bảng ghi chú giao hàng

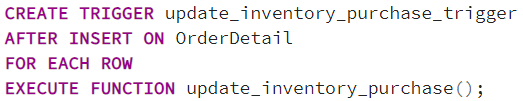
****

#### 3.4.2.11. Tạo bảng tồn kho

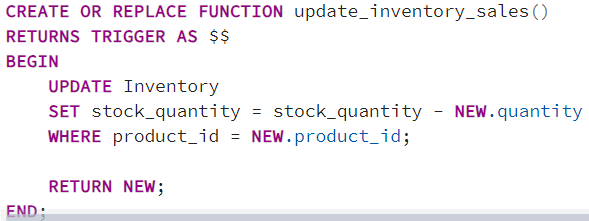
****

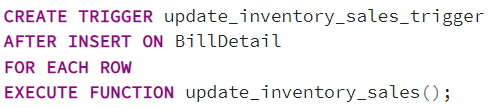
#### 3.4.2.12. Tạo trigger cập nhật số lượng tồn kho khi có đơn đặt hàng

****

****

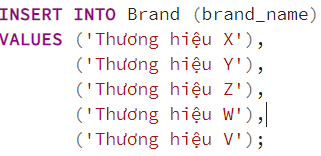
#### 3.4.2.13. Tạo trigger cập nhật số lượng tồn kho khi có hóa đơn bán hàng

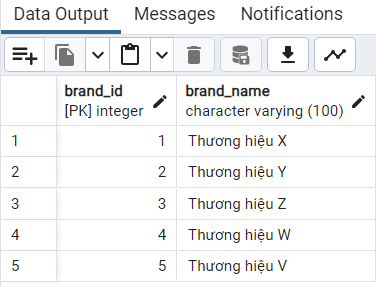
****

****

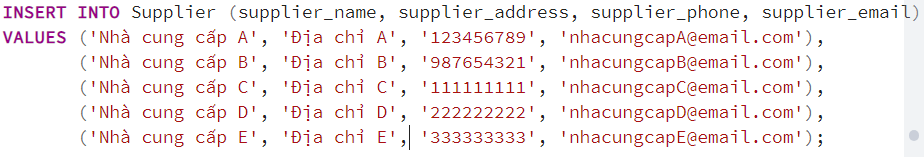
### 3.4.3. Thêm dữ liệu cho từng bảng

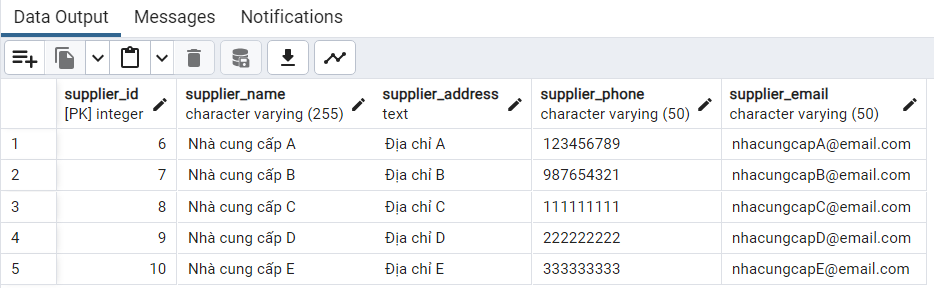
#### 3.4.3.1. Thêm dữ liệu vào bảng thương hiệu

****

****

#### 3.4.3.2. Thêm dữ liệu vào bảng nhà cung cấp

****

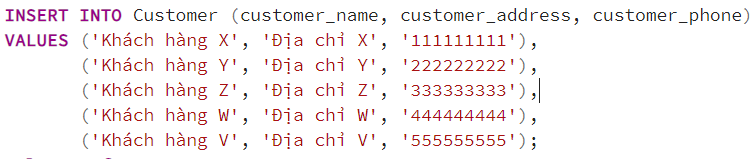
****

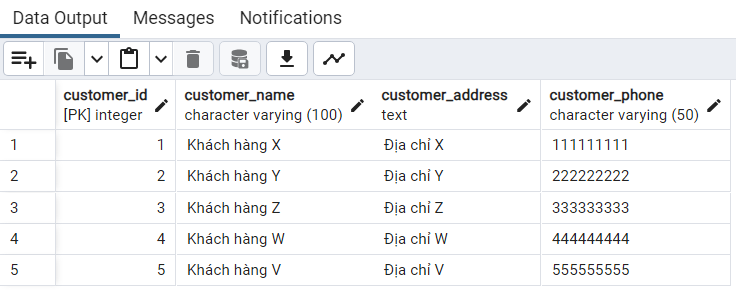
#### 3.4.3.3. Thêm dữ liệu vào bảng Sản phẩm

****

****

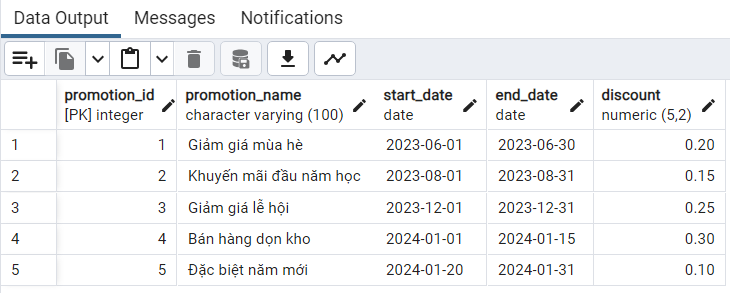
#### 3.4.3.4. Thêm dữ liệu vào bảng Khách hàng

****

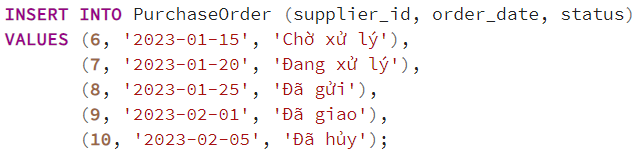
****

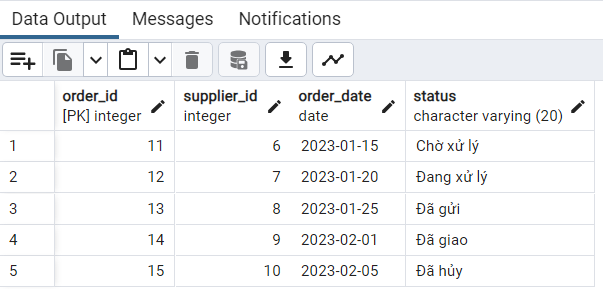
#### 3.4.3.5. Thêm dữ liệu vào bảng Khuyến mãi

****

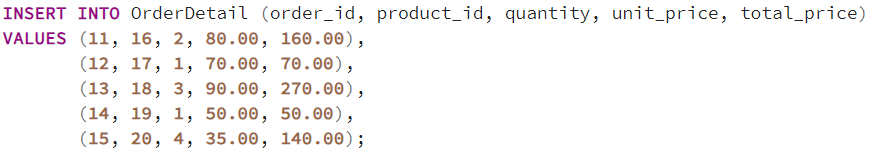
****

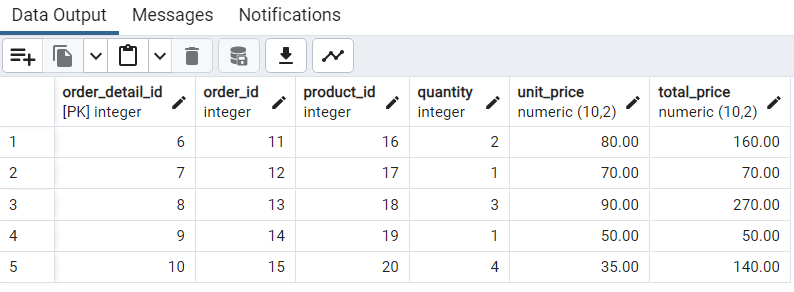
#### 3.4.3.6. Thêm dữ liệu vào bảng Đơn đặt hàng

****

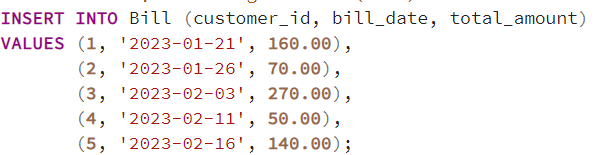
****

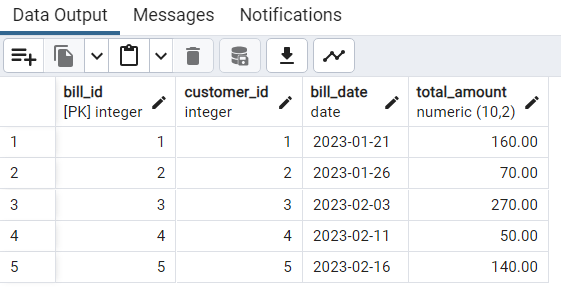
#### 3.4.3.7. Thêm dữ liệu vào bảng Chi tiết đơn đặt hàng

****

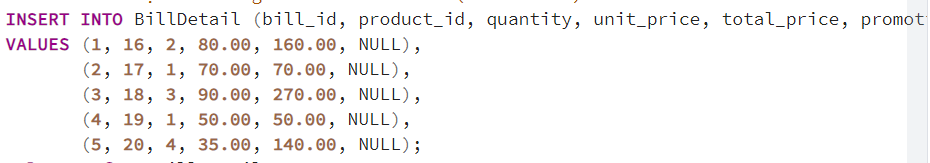
****

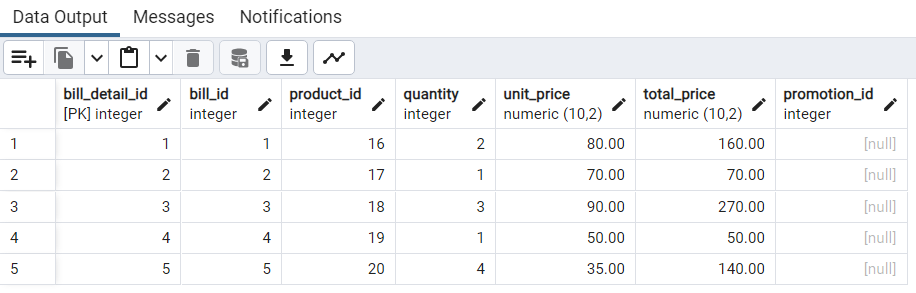
#### 3.4.3.8. Thêm dữ liệu vào bảng Hóa đơn

****

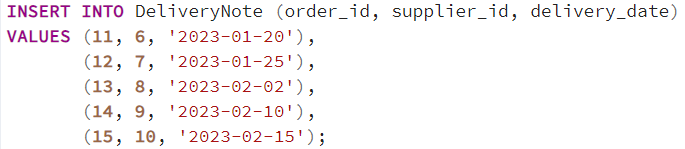
****

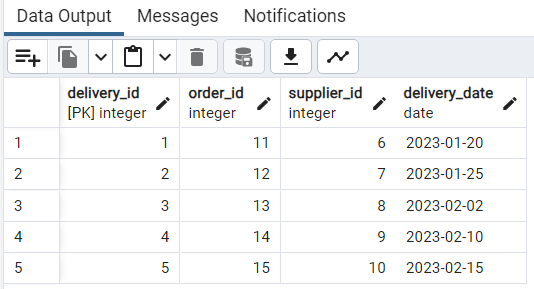
#### 3.4.3.9. Thêm dữ liệu vào bảng Chi tiết hóa đơn

****

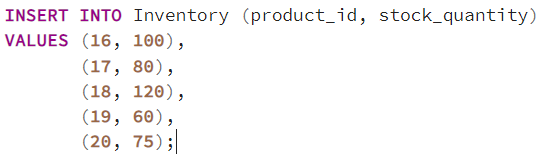
****

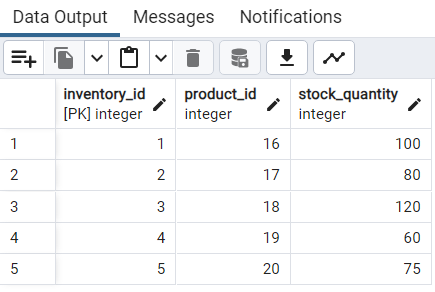
#### 3.4.3.10. Thêm dữ liệu vào bảng Phiếu giao hàng

****

****

#### 3.4.3.11. Thêm dữ liệu vào bảng tồn kho

****

****

# TÀI LIỆU THAM KHẢO

1. https://www.postgresql.org/about/
2. https://bizflycloud.vn/tin-tuc/postgresql-la-gi-tim-hieu-ve-co-so-du-lieu-ma-nguon-mo-tien-tien-nhat-the-gioi-20180919175924611.htm
3. https://www.youtube.com/watch?v=n2Fluyr3lbc
4. https://bkhost.vn/blog/postgresql/#postgresql\_co\_nhung\_tinh\_nang\_noi\_bat\_nao
5. <https://stackjava.com/postgresql/huong-dan-cai-dat-va-cau-hinh-postgresql-tren-windows.htm>
6. <https://appmaster.io/vi/blog/postgresql-la-gi>
7. https://wiki.tino.org/postgresql-la-gi/

# KẾT LUẬN

Qua bài báo cáo này của nhóm chúng em đã thực hiện, em đã được tiếp thu thêm nhiều kiến thức mới như là hiểu thêm về thiết kế CSDL quan hệ là như thế nào, các quy trình để thiết kế,.., cách cài đặt và sử dụng hệ quản trị cơ sở dữ liệu PostgreSQL đồng thời áp dụng nó để thiết kế nên một cơ sở dữ liệu quản lý cửa hàng bán giày. Tuy nhiên, bên cạnh đó nhóm em vẫn còn nhiều thiếu sót trong việc thiết kế các mô hình chưa được tốt, cách thiết kế CSDL chưa được hoàn chỉnh và mở rộng hơn mong thầy và các bạn thông cảm cho chúng em.