

CƠ SỞ DỮ LIỆU

BÀI GIẢNG

**DÀNH CHO SINH VIÊN CÔNG NGHỆ
THÔNG TIN**

NGUYỄN QUỲNH CHI

Hà nội 11/2014

GIỚI THIỆU

Học phần Cơ sở dữ liệu cung cấp các kiến thức để sinh viên nắm được các mức trìn̄u tượng hóa cơ sở dữ liệu, các mô hình cơ sở dữ liệu, các ngôn ngữ biểu diễn và xử lý dữ liệu, lý thuyết về cơ sở dữ liệu quan hệ, quy trình thiết kế một hệ thống cơ sở dữ liệu, các quá trình chuẩn hoá, truy vấn dữ liệu. Đồng thời, học phần cũng cung cấp cho sinh viên những kỹ năng để áp dụng những lý thuyết để thiết kế một cơ sở dữ liệu trong thực tế và xây dựng ứng dụng cơ sở dữ liệu dựa trên một hệ quản trị cơ sở dữ liệu có sẵn.

Đối tượng chính của bài giảng này là sinh viên ngành Công nghệ thông tin hệ đại học, hệ cao đẳng hoặc sinh viên của các ngành khác cũng có thể dùng tài liệu này để tham khảo nhưng khối lượng sẽ được lược bỏ đi một phần tuỳ vào từng ngành và hệ đào tạo.

Sinh viên cần hoàn thành các môn học: Toán rời rạc, Nhập môn logic, Cấu trúc dữ liệu và giải thuật trước khi tham gia học môn học này.

Đây là một môn học tính điểm trung bình sau khi kết thúc cuối kỳ học, trong đó kiểm tra cuối kỳ chiếm 70%, bài tập lớn làm theo nhóm (khoảng 3 hoặc 4 người/nhóm) chiếm 20%, quá trình tham dự trên lớp chiếm 10%.

Tổng số gồm 3 tín chỉ trong đó 40 tiết lý thuyết giảng trên lớp trong đó 8 tiết cho việc giảng viên giải đáp thắc mắc về bài tập và bài tập lớn và 4 tiết cuối cùng dùng để sinh viên thuyết trình bài tập lớn trên lớp, ôn tập và trao đổi với giảng viên.

Yêu cầu đọc sách để chuẩn bị bài và làm bài tập lớn theo hướng dẫn của giảng viên trước mỗi buổi tham gia lớp học. Nói chung sinh viên được khích động đặt các câu hỏi và phát biểu ý kiến riêng với những vấn đề đặt ra trong quá trình nghe giảng trên lớp, tránh thái độ thụ động ngồi nghe.

Nội dung của môn học sẽ được trình bày trong 14 bài học với nội dung được thể hiện trong mục lục của bài giảng.

MỤC LỤC

GIỚI THIỆU	1
MỤC LỤC	2
Bài 1: Khái niệm chung về cơ sở dữ liệu-phần 1	9
Khái niệm về cơ sở dữ liệu	9
Khái niệm về hệ quản trị cơ sở dữ liệu	9
Các hệ thống cơ sở dữ liệu truyền thống	10
Tổng quan các thành phần của một hệ quản trị cơ sở dữ liệu.....	12
Kiến trúc của một hệ quản trị cơ sở dữ liệu.....	13
Khái niệm dữ liệu và thông tin	14
Dữ liệu phái sinh và dữ liệu vật lý	15
Bài 2: Khái niệm chung về cơ sở dữ liệu-phần 2	16
Sự cần thiết của việc thiết kế cơ sở dữ liệu.....	16
Các vai trò cần thiết trong một môi trường cơ sở dữ liệu	16
Các ưu điểm của hệ quản trị cơ sở dữ liệu	17
Mô hình trừu tượng 3 lớp.....	17
Mức bên ngoài	18
Mức khái niệm	19
Mức trừu tượng bên trong	19
Mức trừu tượng vật lý	19
Khái niệm lược đồ, ánh xạ và thể hiện của cơ sở dữ liệu	19
Sự độc lập dữ liệu.....	21
Các ngôn ngữ cơ sở dữ liệu	21
Ngôn ngữ định nghĩa dữ liệu.....	22
Ngôn ngữ thao tác dữ liệu	22

Ngôn ngữ truy vấn.....	22
Các ngôn ngữ thẻ hệ thứ tư	22
Bài 3 và 4: Các mô hình dữ liệu	24
Giới thiệu chung	24
Quá trình thiết kế một cơ sở dữ liệu	25
Bước 1. Phân tích yêu cầu của bài toán	25
Bước 2. Thiết kế cơ sở dữ liệu mức khái niệm	25
Bước 3. Thiết kế cơ sở dữ liệu ở mức logic	25
Bước 4. Cải thiện các lược đồ	25
Bước 5. Thiết kế cơ sở dữ liệu vật lý	25
Bước 6.Thiết kế an toàn bảo mật cho hệ thống.....	26
Mô hình thực thể liên kết	26
Các ký pháp cho mô hình E-R được thể hiện trong hình vẽ dưới đây	27
Một ví dụ về lược đồ E-R (ERD) được thể hiện trong hình vẽ dưới đây.....	29
Một ví dụ khác về lược đồ E-R- phức tạp hơn.....	30
Các thuộc tính trong mô hình E-R	30
Các liên kết trong mô hình E-R.....	31
Các ràng buộc ánh xạ lực lượng liên kết (mapping cardinality) trong mô hình E-R	32
Các ràng buộc tham gia trong mô hình E-R.....	33
Khóa của một tập thực thể.....	33
Tập các mối quan hệ	35
Một số vấn đề cần quan tâm khi thiết kế mô hình E-R.....	35
Ảnh hưởng của ràng buộc ánh xạ lực lượng liên kết lên các khóa	35
Vấn đề đặt vị trí cho các thuộc tính của mối quan hệ	37
Các vấn đề thiết kế khác cần xem xét	39

Vấn đề tập thực thể hay các thuộc tính	39
Việc coi một đối tượng là một tập thực thể hay tập mối quan hệ	41
Việc coi một đối tượng là tập thực thể yếu hay tập thực thể mạnh.....	41
Bài 5: Giới thiệu về mô hình hóa dữ liệu-phần 2	43
Mô hình thực thể liên kết mở rộng	43
Cụ thể hóa (Specializations).....	43
Tổng quát hóa.....	44
Sự kế thừa thuộc tính	45
Các ràng buộc trên việc tổng quát hóa	46
Tích hợp	47
Các mối quan hệ nhiều ngôi.....	49
Lược đồ E-R với các chỉ thị vai trò.....	51
Giới thiệu về ngôn ngữ tạo mô hình thống nhất UML	51
Sự liên quan giữa lược đồ lớp của UML và lược đồ E-R	51
Ràng buộc toàn vẹn tham chiếu.....	53
Bài 6: Mô hình dữ liệu quan hệ	55
Mô hình dữ liệu quan hệ	55
Quan hệ là gì?.....	57
Các lược đồ quan hệ và Các thẻ hiện của quan hệ	58
Thiết kế logic.....	59
Việc ánh xạ mô hình thực thể liên kết sang mô hình quan hệ	59
Bước 1: dùng cho việc ánh xạ các thực thể thông thường (thực thể khỏe).....	59
Bước 2: Ánh xạ các thực thể yếu	61
Bước 3: Ánh xạ các quan hệ hai ngôi	62
Bước 4: Ánh xạ các thực thể liên kết (hay thực thể kết hợp).....	64

Bước 5: Ánh xạ các quan hệ một ngôi (đê quy).....	66
Bước 6: Ánh xạ các quan hệ nhiều ngôi	67
Bước 7: Ánh xạ các mối liên kết lớp cha/lớp con	68
Bài 7 Ngôn ngữ truy vấn	73
Ngôn ngữ đại số quan hệ	73
Năm phép toán cơ bản của đại số quan hệ	74
Phép chọn	74
Phép chiếu	74
Phép hợp.....	75
Phép trừ	76
Phép tích đề các	77
Biểu thức đại số quan hệ	78
Bài 8 Ngôn ngữ truy vấn quan hệ - phần 2.....	82
Các toán tử bổ sung trong đại số quan hệ	82
Phép giao	83
Phép kết nối	83
Phép kết nối ngoài	85
Phép nửa kết nối	86
Phép chia	87
Tính hữu ích của các toán tử dư thừa (mở rộng).....	88
Một số các truy vấn sử dụng chỉ năm phép toán cơ bản	88
Toán tử đặt lại tên.....	89
Các truy vấn để thực hành việc sử dụng tất cả các toán tử của đại số quan hệ.....	90
Bài 9: Giới thiệu về Phụ thuộc hàm	91
Mục đích của sự chuẩn hóa và phụ thuộc hàm	91

Dữ thừa dữ liệu và dị thường khi cập nhật.....	92
Thuộc tính kết nối không tồn thất thông tin.....	93
Bảo toàn các phụ thuộc hàm	94
Định nghĩa phụ thuộc hàm	94
Xác định các phụ thuộc hàm	95
Tóm tắt về các đặc tính của phụ thuộc hàm	96
Các luật suy diễn cho các phụ thuộc hàm	97
Tính toán bao đóng	98
Định nghĩa bao đóng	98
Thuật toán Closure {trả về X^+ trên F}	99
Ví dụ sử dụng thuật toán tính Bao đóng Closure	99
Phủ và sự tương đương của tập phụ thuộc hàm.....	101
Phủ không dư thừa	101
Các thuộc tính dư thừa	103
Tập phụ thuộc hàm tối giản trái và tối giản phải.....	103
Phủ đơn vị và Phủ tối thiểu	105
Bài 10: Giới thiệu về chuẩn hoá	106
Chuẩn hoá dựa trên khoá chính	106
Xác định khoá cho một lược đồ quan hệ.....	106
Chuẩn hoá dựa trên khoá chính.....	107
Các yêu cầu chuẩn hoá.....	108
Dạng chưa phải chuẩn 1 (Non-first normal form-N1NF)	108
Dạng chuẩn 1 (First Normal Form-1NF)	109
Dạng chuẩn hai (2NF)	109
Bài 11: Giới thiệu về chuẩn hoá-phần 2	110

Dạng chuẩn 3NF	110
Tại sao cần chuẩn 3NF	110
Chuẩn Boyce-Codd BCNF.....	111
Phân tách lược đồ quan hệ về các dạng chuẩn.....	112
Bảo toàn các phụ thuộc hàm	112
Thuật toán kiểm tra tính bảo toàn phụ thuộc hàm.....	113
Thuật toán kiểm tra tính kết nối không tồn thắt thông tin.....	116
Bài 12: Giới thiệu về chuẩn hoá-phần 3	119
Thuật toán số 1 cho việc phân tách về 3 NF	119
Thuật toán số 2 cho việc phân tách về 3NF	120
Tại sao lại sử dụng 3NF.2 mà không sử dụng 3NF.1.....	124
Thuật toán số 3 để phân tách một lược đồ quan hệ về 3NF	124
Giới thiệu một kỹ thuật khác để kiểm tra tính bảo toàn phụ thuộc hàm	124
Bài 13: Giới thiệu về ngôn ngữ truy vấn có cấu trúc SQL.....	126
Lịch sử phát triển của ngôn ngữ SQL.....	126
Giới thiệu về ngôn ngữ SQL.....	127
Cách sử dụng các lệnh của ngôn ngữ định nghĩa dữ liệu DDL trong SQL.....	128
Các qui định về ký pháp cho câu lệnh SQL được trình bày trong bài giảng	129
Câu lệnh tạo cấu trúc bảng trong SQL	130
Ngôn ngữ thao tác dữ liệu DML của SQL	133
Lưu trữ các thay đổi tới một bảng	137
Tóm tắt các câu lệnh DML không truy vấn của SQL	138
Câu lệnh truy vấn của DML của SQL.....	139
Sử dụng một câu truy vấn con SELECT để thêm các hàng vào một bảng	140
Câu lệnh truy vấn lựa chọn với sự hạn chế của điều kiện.....	140

Bài 14: Giới thiệu về ngôn ngữ SQL -phần 2.....	144
Các toán tử đặc biệt trong SQL.....	144
Các bảng ảo - tạo các khung nhìn	145
Kết nối các bảng trong cơ sở dữ liệu.....	146
Kết nối đệ quy: là kết nối một bảng với chính bảng đó.	147
Kết nối ngoài	148
Các toán tử tập hợp quan hệ.....	151
Các phép kết nối của SQL.....	154
Các phép kết nối ngoài OUTER JOIN.....	158
Truy vấn con và truy vấn tương hỗ	158

Bài 1: Khái niệm chung về cơ sở dữ liệu - phần 1

Khái niệm về cơ sở dữ liệu

Theo nhận thức chung nhất của nhiều độc giả, một **cơ sở dữ liệu** đơn giản là một bộ tập hợp các dữ liệu liên quan tới nhau. Định nghĩa này khá mơ hồ bởi vì nếu áp dụng định nghĩa này, chúng ta có thể xem một trang sách là một cơ sở dữ liệu bởi vì nó bao gồm các dữ liệu là những từ nằm trong trang sách đó và rõ ràng các từ này có quan hệ với nhau vì chúng cùng mô tả nội dung một chủ đề cụ thể nào đó đang được thể hiện trong trang sách đó.

Lưu ý rằng khái niệm “dữ liệu” trong một cơ sở dữ liệu có thể bao phủ một phạm vi rất rộng các đối tượng khác nhau từ các số, văn bản, đồ họa, video, v.v...

Một định nghĩa cụ thể hơn nữa của một cơ sở dữ liệu bao gồm các đặc tính không thường minh được cân nhắc cùng nhau để định nghĩa một cơ sở dữ liệu. Chúng ta cùng xem xét cách nhìn nhận khái niệm cơ sở dữ liệu theo cách cụ thể này.

Một cơ sở dữ liệu thể hiện các khía cạnh khác nhau của một thế giới thực. Sự trừu tượng của một thế giới thực thường được coi là một thế giới nhỏ hoặc vũ trụ của một vấn đề nào đó. Một cách khác, một cơ sở dữ liệu được coi là một bộ thu thập dữ liệu với các ý nghĩa gắn kết. Các dữ liệu ngẫu nhiên thường không thể coi là một cơ sở dữ liệu mặc dù chúng là những ngoại lệ.

Một cơ sở dữ liệu được thiết kế, xây dựng, lớn dần và được sử dụng cho một mục đích cụ thể nào đó. Nó sẽ có một tập các người sử dụng tiềm năng và được sử dụng cho các ứng dụng cụ thể ngay từ khi thiết kế ban đầu. Ví dụ một cơ sở dữ liệu quản lý thông tin của sinh viên trong một trường học, được dùng với mục đích quản lý các hoạt động chính của sinh viên trong trường bao gồm một số chức năng chính là quản lý điểm số các môn học, quản lý thi đua, được sử dụng bởi nhóm người dùng tiềm năng là sinh viên, các cán bộ quản lý và giáo viên trong trường...

Khái niệm về hệ quản trị cơ sở dữ liệu

Một cơ sở dữ liệu được quản lý bởi một hệ quản trị cơ sở dữ liệu, thường được tham khảo tới như một hệ thống cơ sở dữ liệu.

Một hệ quản trị cơ sở dữ liệu được cho là sẽ phải cung cấp những tính năng quan trọng sau đây:

1. Cho phép người sử dụng tạo ra một cơ sở dữ liệu mới. Việc này sẽ được thực hiện thông qua một ngôn ngữ định nghĩa dữ liệu (Data Definition Languages-DDLs).
2. Cho phép người sử dụng truy vấn cơ sở dữ liệu thông qua ngôn ngữ thao tác dữ liệu (Data Manipulation Languages-DMLs).

3. Hỗ trợ việc lưu trữ một khối lượng rất lớn dữ liệu mà không gây mất mát và tổn thất thông tin. Kích cỡ điển hình là từ nhiều gigabytes trở lên và lưu trữ chúng hiệu quả trong một khoảng thời gian rất dài. Đương nhiên để lưu trữ tốt trong khoảng thời gian dài đó thì cần phải duy trì, cập nhật thông tin tốt và hiệu quả. Đồng thời, duy trì tính bảo mật và tính toàn vẹn dữ liệu trong các xử lý được thực hiện trong hệ thống.
4. Kiểm soát truy nhập dữ liệu từ nhiều người sử dụng cùng một lúc.

Các hệ thống cơ sở dữ liệu truyền thống

Các hệ thống cơ sở dữ liệu thương mại đầu tiên xuất hiện vào những năm của thập kỷ 60. Chúng phát triển từ hệ thống lưu trữ theo kiểu tệp truyền thống, theo kiểu tệp này thì các dữ liệu được lưu trữ trong các tệp tách rời nhau và được lưu trữ trong bộ lưu trữ vật lý. Các hệ thống tệp này có cung cấp các đặc tính (3) của hệ quản trị cơ sở dữ liệu được mô tả ở phần trên nhưng chúng không hoặc cung cấp rất ít các tính năng (4).

Hơn nữa, các hệ thống tệp này không cung cấp các chức năng hỗ trợ trực tiếp cho các tính năng của mục (2) ở trên ví dụ như chúng không hỗ trợ các ngôn ngữ truy vấn. Hệ thống này cũng không hỗ trợ trực tiếp chức năng ở mục (1) ở trên, việc hỗ trợ cho các lược đồ quan hệ rất hạn chế, chỉ cho phép tạo cấu trúc thư mục cho các tệp dữ liệu mà không cho phép thay đổi hay tạo mới cấu trúc của các tệp.

Một vài hệ thống cơ sở dữ liệu ban đầu quan trọng hơn là những hệ thống trong đó dữ liệu được cấu tạo bởi nhiều mục nhỏ và nhiều truy vấn hoặc thay đổi có thể thực hiện được. Ví dụ như các hệ thống đặt vé máy bay hay các hệ thống ngân hàng.

Nói đến sự phát triển vượt bậc của các hệ thống cơ sở dữ liệu phải nhắc đến một bài báo nổi tiếng được viết bởi Codd năm 1970, một bài báo có ảnh hưởng rất lớn tới sự thay đổi các hệ thống cơ sở dữ liệu (tham khảo tới bài báo Codd, E.F., “A relational model for large shared data banks”, Communications of ACM, 13:6, pp. 377-387). Trong bài báo này, Codd đề xuất rằng các hệ thống cơ sở dữ liệu nên đưa ra cho người sử dụng một khung nhìn về dữ liệu trong đó dữ liệu được tổ chức dưới dạng các bảng được gọi là các quan hệ. Bên trong sự mô tả dữ liệu theo kiểu này, một cấu trúc dữ liệu phức tạp được thiết lập cho phép các truy vấn của người sử dụng được đáp ứng nhanh chóng. Nhưng không giống như những người sử dụng các hệ thống cơ sở dữ liệu trước đây, người dùng của một hệ thống quan hệ không cần quan tâm tới cấu trúc lưu trữ của dữ liệu. Các câu truy vấn sau đó có thể được thể hiện trong một ngôn ngữ bậc cao, là loại ngôn ngữ làm tăng hiệu suất đáng kể cho những người lập trình cơ sở dữ liệu.

Các hệ thống cơ sở dữ liệu ban đầu có kích cỡ rất lớn, một loại trong số chúng là hệ quản trị cơ sở dữ liệu. Ban đầu hệ quản trị cơ sở dữ liệu rất lớn, có giá thành cao và chạy trên các máy tính mainframe lớn. Kích cỡ bộ lưu trữ hàng gigabytes dữ liệu trước đây là rất lớn nên cần các máy tính lớn. Ngày nay, một gigabyte có thể được lưu trữ trên một đĩa đơn do công

nghệ ngày càng phát triển, vì vậy hệ quản trị cơ sở dữ liệu có thể chạy trên một máy tính cá nhân là hoàn toàn khả thi. Hệ thống ngày càng nhỏ dần theo thời gian do công nghệ điện tử càng phát triển.

Ngày nay, hệ quản trị cơ sở dữ liệu dựa trên mô hình dữ liệu bắt đầu xuất hiện như một công cụ chung cho các ứng dụng của máy tính cũng như việc các bảng tính và các bộ xử lý văn bản đã trở thành công cụ chung trước đây.

Mặt khác, một gigabyte ngày nay không còn được coi là dữ liệu có kích cỡ lớn nữa. Các hệ cơ sở dữ liệu lớn phải chứa hàng trăm gigabytes hoặc nhiều hơn. Khi bộ nhớ lưu trữ trở nên rẻ hơn, con người thường tìm thấy các lý do mới để lưu trữ nhiều dữ liệu hơn. Chẳng hạn, một chuỗi các cửa hàng bán lẻ thường lưu trữ tới terabytes (1 terabytes = 1000 gigabytes hoặc 10^{12} bytes) thông tin để lưu lại lịch sử của mỗi giao dịch mua bán trong một khoảng thời gian rất dài. Dữ liệu thì không phải chỉ ở dạng văn bản và số như trước kia, mà nay có nhiều dạng mới như dạng âm thanh, hình ảnh thường chiếm không gian lưu trữ rất lớn ví dụ như một giờ của video sẽ chiếm một gigabyte. Hay các cơ sở dữ liệu lưu trữ các hình ảnh vệ tinh sẽ chiếm nhiều petabytes dữ liệu trong đó 1 petabyte=1000 gigabytes hay 10^{15} bytes. Như vậy một xu hướng hiện nay là dữ liệu ngày càng lớn.

Để quản lý được các cơ sở dữ liệu lớn như vậy đòi hỏi nhiều công nghệ hiện đại

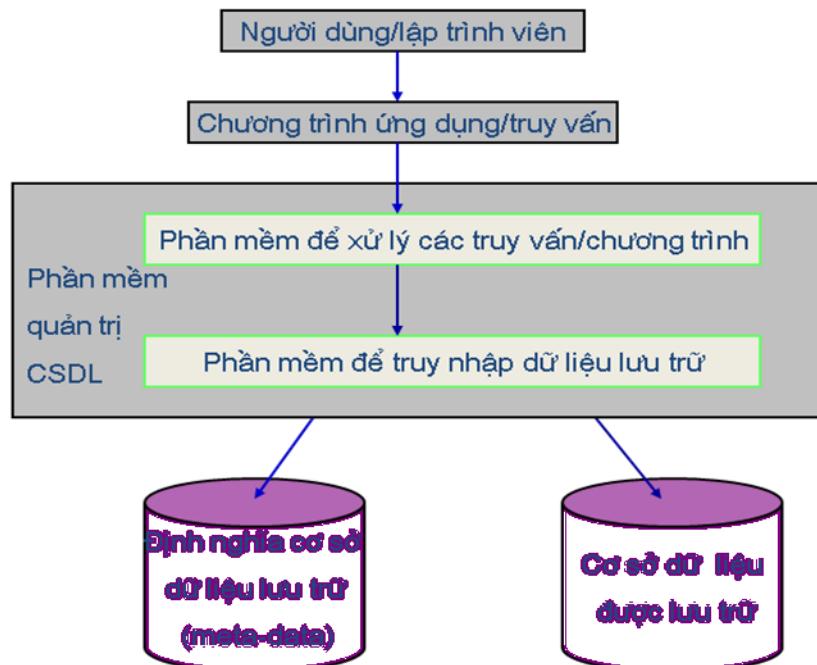
- Các cơ sở dữ liệu hiện đại của các kích cỡ gần đây nhất được lưu trữ trên một mảng các đĩa (các thiết bị lưu trữ thứ cấp)
- Các cơ sở dữ liệu hầu như không bao giờ cho rằng “dữ liệu” sẽ vừa với bộ nhớ trong. Các hệ thống cũ cũ hơn thường chỉ có các thiết bị lưu trữ thứ cấp dưới dạng các đĩa từ (công nghệ tương tự)

Hai hướng mới cho phép các hệ thống cơ sở dữ liệu có thể quản lý được khối lượng dữ liệu lớn hơn một cách nhanh hơn là:

1. Lưu trữ mức độ cấp 3: các cơ sở dữ liệu lớn nhất hiện nay đòi hỏi nhiều hơn chỉ lưu trữ trên các đĩa (cấp 2). Các thiết bị cấp 3 có xu hướng lưu trữ theo đơn vị terabyte và có thời gian truy nhập dài hơn thời gian truy nhập của đĩa từ truyền thống. Thời gian truy nhập của một đĩa truyền thống là nằm trong khoảng 10-20 m giây. Trong khi đó của thiết bị mới này sẽ mất vài giây. Các thiết bị này liên quan tới việc chuyển các đối tượng mà trên đó dữ liệu được lưu trữ tới một thiết bị đọc nào đó thông qua một dạng giám sát bằng robot nào đó. Việc sử dụng đĩa CDs như một phương tiện lưu trữ mức độ này.
2. Tính toán song song: Khả năng lưu trữ khối lượng dữ liệu không lồ là rất quan trọng nhưng nó sẽ ít được sử dụng nếu như chúng ta không thể truy nhập vào khối lượng lớn dữ liệu đó một cách nhanh chóng. Các cơ sở dữ liệu rất lớn đòi hỏi bộ cải thiện tốc độ. Việc cải thiện tốc độ được thực hiện bằng nhiều cách trong cơ sở dữ liệu hiện đại ngày nay bao gồm:

- a. Các cấu trúc chỉ mục
- b. Cơ chế song song hóa- liên quan tới cả song song hóa bộ vi xử lý cũng như song song hóa bản thân dữ liệu. Trong một phạm vi nào đó, các hệ thống cơ sở dữ liệu phân tán cũng có thể được cho vào như một bộ cải thiện tốc độ mặc dù theo một cách thức hơi khác, chúng ta sẽ xem xét vấn đề này trong các bài giảng sau.

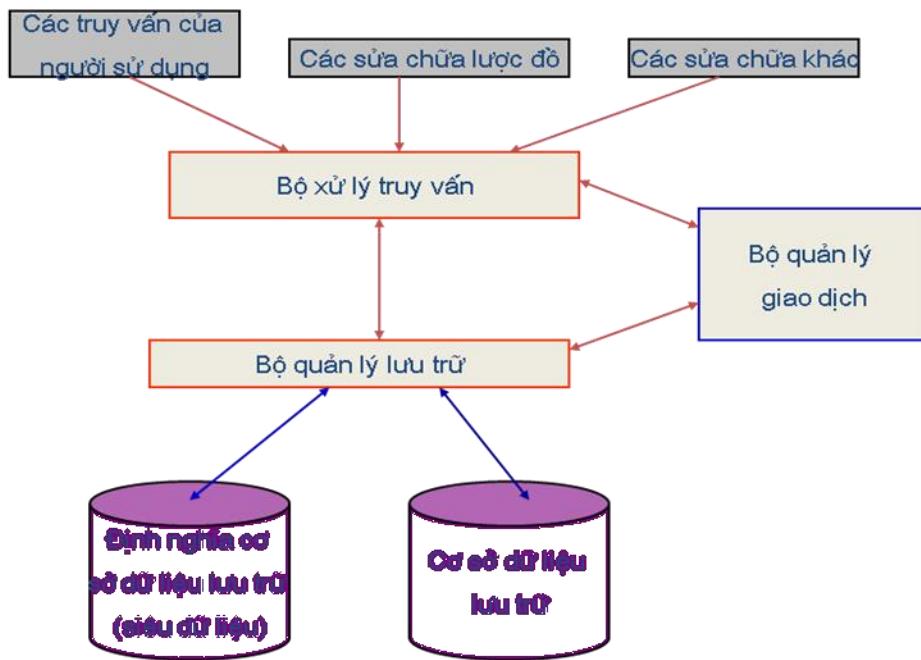
Tổng quan các thành phần của một hệ quản trị cơ sở dữ liệu



Cơ sở dữ liệu được lưu trữ và siêu dữ liệu: Cơ sở dữ liệu được lưu trữ nằm trên các thiết bị cấp 3 hoặc thứ cấp (cấp 2). Trong một vài thời điểm, một số phần của cơ sở dữ liệu sẽ được lưu trữ thêm trên các bộ lưu trữ đệm nhưng chúng ta sẽ bỏ qua vấn đề này trong phạm vi bài giảng này.

Siêu dữ liệu là dữ liệu về dữ liệu. Trong trường hợp này siêu dữ liệu là các mô tả về các thành phần dữ liệu của cơ sở dữ liệu. Vị trí tương đối của các trường trong một bản ghi, các thông tin về lược đồ, thông tin về chỉ mục và nhiều thứ khác nữa. Với một cơ sở dữ liệu nào đó, một hệ quản trị cơ sở dữ liệu có thể duy trì nhiều bộ chỉ mục khác nhau được thiết kế để cung cấp truy nhập nhanh tới dữ liệu ngẫu nhiên. Hầu hết các chỉ mục này được thể hiện như dạng cây nhị phân trong các cơ sở dữ liệu hiện đại. Cây nhị phân có xu hướng thấp và béokhiến việc truy nhập nhanh từ gốc đến lá.

Kiến trúc của một hệ quản trị cơ sở dữ liệu



Bộ quản lý lưu trữ: trong một hệ thống cơ sở dữ liệu đơn giản, bộ quản lý lưu trữ chỉ là đơn giản là một hệ thống tệp của một hệ điều hành nào đó. Trong các hệ thống lớn hơn, để hiệu quả, các kho lưu trữ của hệ quản trị cơ sở dữ liệu được quản lý trên đĩa một cách trực tiếp.

Bộ quản lý lưu trữ bao gồm hai thành phần cơ bản (1) quản lý vùng đệm và (2) quản lý tệp

- Quản lý tệp làm nhiệm vụ theo dõi vị trí của các tệp trên các đĩa và lấy ra được các khối hoặc khai thác một tệp yêu cầu từ bộ quản lý đệm. Các đĩa thường bị chặn vào các vùng có không gian liên tục kích cỡ từ 2^{12} đến 2^{14} bytes (khoảng 4000 đến 16,000 bytes/1 khối dữ liệu)
- Bộ quản lý đệm quản lý bộ nhớ chính. Các khối dữ liệu được lấy ra từ các đĩa thông qua bộ quản lý tệp và lựa chọn một trang trong bộ nhớ chính để lưu trữ chứ không chặn lại. Thuật toán thiết lập trang sẽ quyết định một trang sẽ tồn tại bao lâu trong bộ nhớ chính. Tuy nhiên, bộ quản lý các giao dịch có thể ép một trang trong bộ nhớ chính quay trở về đĩa (chúng ta sẽ xem xét những chi tiết của vấn đề này trong các bài sau)

Bộ quản lý truy vấn làm nhiệm vụ chuyển đổi truy vấn hoặc thao tác cơ sở dữ liệu, đang được thực hiện bằng ngôn ngữ bậc cao (ví dụ SQL) thành một chuỗi các yêu cầu cho các dữ liệu được lưu trữ như các bộ phận của một quan hệ hoặc các phần của một chỉ mục của một quan hệ. Phần khó khăn nhất của việc xử lý truy vấn thường là quá trình tối ưu hóa truy

vấn, cái liên quan tới việc thiết lập một dạng chiến lược tốt để thực thi truy vấn. Chúng ta sẽ bàn luận tới tối ưu hóa truy vấn một cách chi tiết hơn trong những phần sau của bài giảng.

Bộ quản lý giao dịch: tồn tại một số đảm bảo cho một hệ quản trị cơ sở dữ liệu phải có khi thực hiện các phép toán trên một cơ sở dữ liệu. Những đảm bảo này thường được nhắc tới như các thuộc tính ACID được mô tả như sau

- Tính nguyên vẹn: được thể hiện theo nguyên tắc hoặc tất cả công việc của một giao dịch được thực hiện hoặc không công việc nào của nó được thực hiện
- Tính đồng nhất: có nghĩa là dữ liệu không thể ở một trạng thái không đồng nhất
- Tính biệt lập: có nghĩa là các giao dịch đồng thời phải được tách riêng một cách biệt lập khỏi nhau trên cả phương diện ảnh hưởng và tính hiển thị
- Tính duy trì: những thay đổi tới cơ sở dữ liệu gây ra bởi một giao dịch không được mất đi thậm chí hệ thống bị hỏng ngay sau khi giao dịch hoàn thành.

Khái niệm dữ liệu và thông tin

Xét một ví dụ về dữ liệu về các con số như sau

Data: 0 11,500
5 12,300
10 12,800
15 10,455
20 12,200
25 13,900
30 14,220

Thể hiện dưới dạng “thô” như trên, dữ liệu này có rất ít ý nghĩa. Trong trường hợp này, nó đơn giản giống như một cặp danh sách các số nguyên. Ngữ cảnh để làm nền cho dữ liệu không có.

Bằng việc xử lý dữ liệu, chúng ta chuyển đổi nó thành một dạng có ý nghĩa hơn. Trong ví dụ này quá trình xử lý bao gồm chủ yếu là đặt dữ liệu vào ngữ cảnh (cái thường được làm bằng cách thêm dữ liệu vào. Mặc dù những dữ liệu thêm vào này thực sự là siêu dữ liệu). Böyle giờ dữ liệu bắt đầu có ý nghĩa hơn như sau:

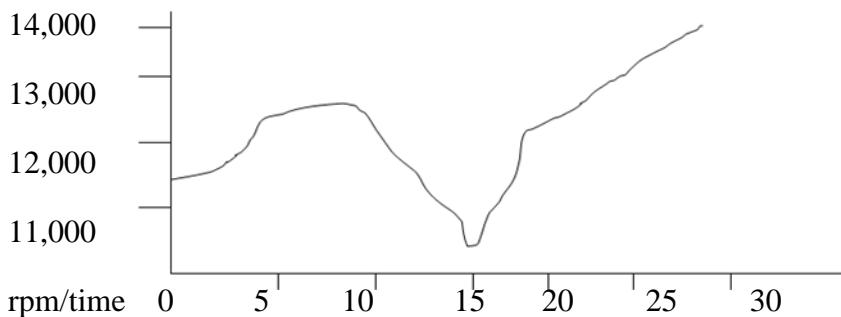
Thông tin: Dữ liệu RPM: Roebling Road 10/4/2003- Yamaha Heavy

Vòng 12: thời gian rpm

0	11,500
5	12,300
10	12,800
15	10,455

20	12,200
25	13,900
30	14,220

Với cùng dữ liệu được mô tả ở trên, xem xét cách xử lý sau đây chuyển đổi dữ liệu đó sang dạng đồ thị



Đồ thị: một phần vòng 12 - Roebling Road 10/4/2003- Yamaha Heavy

Dữ liệu phái sinh và dữ liệu vật lý

Một cơ sở dữ liệu sẽ chứa rất nhiều dữ liệu vì nó mô hình hóa toàn bộ các tính năng và đặc điểm của một công ty đối tượng được thể hiện trong cơ sở dữ liệu này. Ví dụ, xem xét một cơ sở dữ liệu sinh viên của học viện công nghệ bưu chính viễn thông, chúng ta sẽ biểu diễn tên, số chứng minh thư, ngành học của mỗi sinh viên cùng với tập hợp những lớp học và điểm số tương ứng mà mỗi sinh viên đó tham gia. Tại một thời điểm nào đó trong quá trình hoạt động của hệ thống, một người có đủ thẩm quyền truy nhập sẽ nhập dữ liệu của sinh viên vào cơ sở dữ liệu theo một cách thức nào đó, hoặc là tự động hoặc là cách thủ công bằng tay. Nếu chúng ta giả sử rằng cơ sở dữ liệu này cần lưu trữ điểm trung bình của mỗi sinh viên thì giá trị của các điểm trung bình sẽ xuất phát từ đâu? Liệu có cần nhập dữ liệu này vào từ người sử dụng không? Thường thì chúng ta không cần thiết phải làm như vậy, mà hệ quản trị cơ sở dữ liệu chỉ cần tính toán điểm trung bình từ các điểm số khác (thực ra là chỉ cần một chương trình phần mềm đơn giản chạy trên nền hệ quản trị là có thể làm được việc này, chúng ta sẽ bàn luận về vấn đề này trong phần sau của bài giảng). Vì vậy, điểm trung bình của một sinh viên sẽ được suy ra từ những dữ liệu khác liên quan tới sinh viên này. Nếu dữ liệu dùng để tính giá trị trung bình này thay đổi giá trị theo một cách thức nào đó thì giá trị của điểm số trung bình cũng thay đổi theo.

Phụ thuộc vào mức độ phức tạp của tầng ứng dụng và của hệ quản trị cơ sở dữ liệu, khôi dữ liệu phái sinh (được tính toán) từ những dữ liệu khác nằm trong cơ sở dữ liệu có thể lớn hơn rất nhiều khôi lượng của dữ liệu được nhập vào trong thực tế (hay gọi là dữ liệu vật lý). Một thực tế khác là những cân nhắc khi nào và liệu dữ liệu phái sinh này có trở thành dữ liệu vật lý bởi vì không có hạn chế cho việc dữ liệu phái sinh sẽ trở thành thành viên chính thức được nhập vào cơ sở dữ liệu.

Bài 2: Khái niệm chung về cơ sở dữ liệu-phần 2

Sự cần thiết của việc thiết kế cơ sở dữ liệu

Đối với các hệ thống được xây dựng để phục vụ nhu cầu của người sử dụng, các hoạt động thiết kế cơ sở dữ liệu là cần thiết. Một cơ sở dữ liệu được thiết kế không cẩn thận sẽ tạo ra lỗi và lỗi này có thể dẫn tới việc đưa ra các kết luận không đúng đắn, dẫn tới những thiệt hại nặng nề cho tổ chức. Mặt khác, một cơ sở dữ liệu được thiết kế tốt sẽ sinh ra một hệ thống làm việc hiệu quả cung cấp các thông tin một cách chính xác hỗ trợ cho quá trình đưa ra quyết định đúng đắn dẫn tới thành công.

Các vai trò cần thiết trong một môi trường cơ sở dữ liệu

Vai trò người quản trị dữ liệu(DA): là người có trách nhiệm cho việc quản lý các tài nguyên dữ liệu bao gồm việc lập kế hoạch cho cơ sở dữ liệu, phát triển và duy trì các chuẩn hóa và thủ tục, thiết kế cơ sở dữ liệu mức khái niệm và mức logic.

Vai trò quản trị hệ thống cơ sở dữ liệu (DBA): là người chịu trách nhiệm với việc lưu trữ vật lý của cơ sở dữ liệu bao gồm thiết kế cơ sở dữ liệu vật lý và cài đặt, kiểm soát bảo mật và toàn vẹn của dữ liệu, bảo trì các hệ thống thao tác dữ liệu và đảm bảo công suất thỏa mãn người sử dụng cho các ứng dụng. vai trò DBA có thiên hướng liên quan tới kỹ thuật hơn là vai trò của DA.

Vai trò người thiết kế cơ sở dữ liệu: trong các dự án thiết kế cơ sở dữ liệu có kích cỡ lớn, chúng ta có thể phân biệt hai loại thiết kế: người thiết kế cơ sở dữ liệu logic và người thiết kế cơ sở dữ liệu vật lý.

- Người thiết kế cơ sở dữ liệu logic liên quan tới việc xác định dữ liệu (các thực thể và các thuộc tính tương ứng), các mối quan hệ giữa dữ liệu, các ràng buộc trên dữ liệu được lưu trữ trong cơ sở dữ liệu.
- Người thiết kế cơ sở dữ liệu vật lý phụ thuộc nhiều vào hệ quản trị cơ sở dữ liệu đích. Có nhiều cách để cài đặt cơ chế của cơ sở dữ liệu. Người thiết kế cơ sở dữ liệu vật lý phải nhận thức được toàn bộ các tính năng của hệ thống quản trị cơ sở dữ liệu.

Vai trò người phát triển ứng dụng: khi một cơ sở dữ liệu được cài đặt, các chương trình ứng dụng mà cung cấp các tính năng cần thiết cho người sử dụng cuối cùng cần được xây dựng. Đây là trách nhiệm của người phát triển ứng dụng.

Vai trò người sử dụng cuối: chính là khách hàng sử dụng cơ sở dữ liệu và có thể được phân thành hai nhóm người sử dụng dựa trên việc sử dụng hệ thống này như thế nào

- Nhóm thứ nhất được gọi là nhóm sử dụng không có kinh nghiệm, thường là không có nhận thức về khái niệm hệ quản trị cơ sở dữ liệu là gì. Họ truy nhập vào cơ sở dữ liệu thông qua những chương trình ứng dụng được viết phục vụ mục đích của họ, những chương trình này khiến cho việc sử dụng hệ thống càng đơn giản càng tốt. Họ thường không biết gì, không có một chút kiến thức nào về cơ sở dữ liệu hoặc hệ quản trị cơ sở dữ liệu.
- Nhóm thứ hai là nhóm sử dụng có kinh nghiệm hơn, là những người quen thuộc với cấu trúc của cơ sở dữ liệu và các phương tiện thực hiện được các chức năng do hệ quản trị cơ sở dữ liệu cung cấp. Họ thường sử dụng một ngôn ngữ truy vấn ở mức cao kiểu như SQL để thực hiện các thao tác cần thiết hoặc thậm chí viết các chương trình ứng dụng nhằm mục đích chuyên dụng đó.

Các ưu điểm của hệ quản trị cơ sở dữ liệu

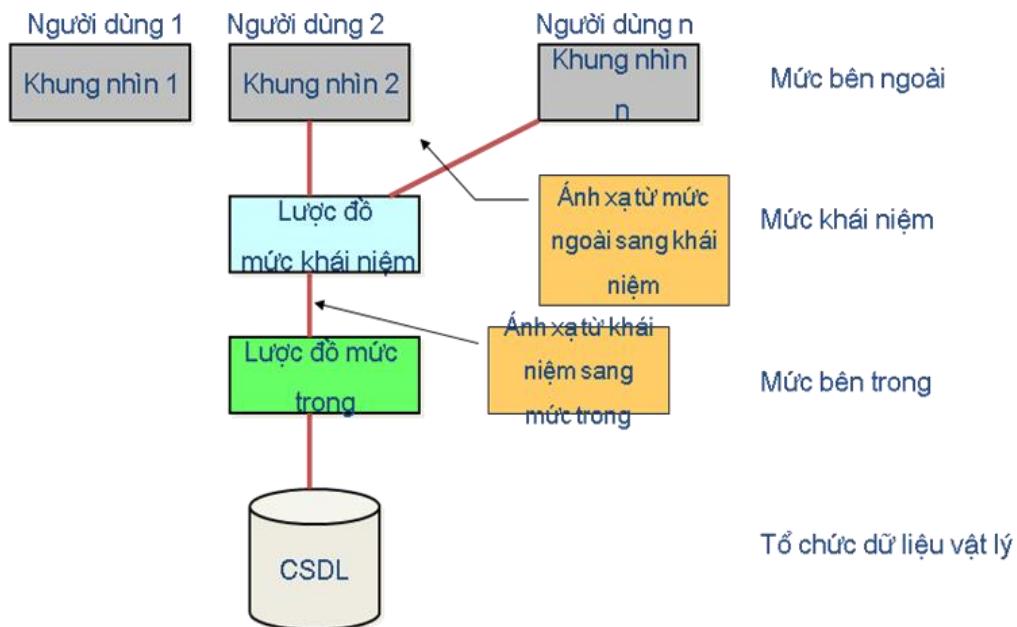
Kiểm soát sự dư thừa dữ liệu	Kinh tế khi tăng khối lượng dữ liệu
Đảm bảo dữ liệu đồng nhất	Tạo cân bằng cho các yêu cầu bị xung đột
Cung cấp thêm thông tin từ dữ liệu	Cải thiện tính truy nhập của dữ liệu
Hỗ trợ sự sẵn có của khối lượng lớn dữ liệu	Cải thiện năng suất thực hiện của hệ thống
Hỗ trợ chia sẻ dữ liệu	Cải thiện sự bảo dưỡng
Cải thiện sự toàn vẹn dữ liệu	Tăng xử lý đồng thời
Cải thiện bảo mật dữ liệu	Cải thiện việc sao lưu và phục hồi
Chuẩn hóa thông tin và dữ liệu	Cải thiện tính đáp ứng tới các truy vấn

Bên cạnh những ưu điểm của các hệ quản trị cơ sở dữ liệu, các hệ thống này cũng có những nhược điểm như:

- Tính phức tạp
- Kích cỡ lớn
- Tốn chi phí mua và bảo trì
- Thêm chi phí cho các phần cứng hỗ trợ
- Chi phí chuyển đổi hệ thống
- Hạn chế về năng suất thực hiện (cho một số trường hợp cụ thể)
- Ảnh hưởng rất lớn đối với các trường hợp hỏng

Mô hình trùu tượng 3 lớp

Để hiểu được quá trình thiết kế một hệ thống cơ sở dữ liệu, chúng ta cần xem xét một khái niệm mô hình trùu tượng 3 lớp trong một hệ thống cơ sở dữ liệu. Mô hình ba lớp này được thể hiện trong hình vẽ dưới đây



Mô hình gồm 3 mức: Mức bên ngoài, mức khái niệm, mức trong

Mức bên ngoài là mức khung nhìn của người dùng đối với cơ sở dữ liệu, mô tả các cách nhìn khác nhau, những yêu cầu khác nhau của người dùng đối với hệ thống dữ liệu. Ở mức này, mỗi khung nhìn là một cách thể hiện một phần của cơ sở dữ liệu tương ứng với người sử dụng. Mỗi người sử dụng cần hệ thống theo một cách thức khác nhau và có một cách khác nhau, quen thuộc với mỗi người để biểu diễn thế giới thực được mô tả trong hệ thống. Khung nhìn từ bên ngoài bao gồm các thực thể, các thuộc tính của chúng và mối quan hệ giữa chúng trong thế giới thực mà người sử dụng đang quan tâm. Các thực thể khác, thuộc tính khác và các mối quan hệ khác có thể tồn tại nhưng không liên quan tới thế giới nhỏ đang cần mô tả đó thì người sử dụng sẽ không cần nhận thức sự tồn tại đó và không cần đưa chúng vào cơ sở dữ liệu.

Thường thì các khung nhìn khác nhau sẽ có cách biểu diễn khác nhau cho cùng một dữ liệu. Ví dụ một khung nhìn thể dữ liệu ngày dưới dạng (tháng, ngày, năm) trong khi một khung nhìn khác thể hiện ngày dưới dạng (ngày, tháng, năm).

Một vài khung nhìn có thể bao gồm các dữ liệu phái sinh hoặc dữ liệu tính toán từ những dữ liệu khác. Những dữ liệu phái sinh này thực tế không được lưu trữ trong cơ sở dữ liệu mà chúng được tạo ra khi cần thiết. Ví dụ một khung nhìn có thể cần tuổi của một người. Tuy nhiên dữ liệu về tuổi không cần thiết phải lưu trữ trong cơ sở dữ liệu vì nó được cập nhật hàng. Thay vào đó nó có thể được tính toán từ dữ liệu được lưu trữ trong cơ sở dữ liệu biểu diễn ngày sinh của người đó và ngày tháng của hệ thống.

Mức khái niệm là khung nhìn của những người thiết kế cơ sở dữ liệu. Mức này sẽ mô tả những dữ liệu cần thiết nào sẽ được lưu trữ trong đó và mối quan hệ giữa chúng. Ở mức này, cấu trúc logic của toàn bộ cơ sở dữ liệu, được nhìn thấy bởi người quản trị cơ sở dữ liệu, được xác định. Nó thể hiện một khung nhìn đầy đủ về các yêu cầu dữ liệu của một tổ chức mà không phụ thuộc vào bất kỳ một cách thức lưu trữ nào. Nói một cách khác, mức khái niệm không quan tâm tới việc lưu trữ vật lý của dữ liệu trong hệ thống, chỉ quan tâm tới việc xác định dữ liệu cần lưu trữ.

Mức khái niệm hỗ trợ cho mỗi khung nhìn từ bên ngoài trong đó bất kỳ dữ liệu sẵn có nào tới người sử dụng cũng cần được lưu lại hoặc có khả năng sinh ra dữ liệu khác ở mức khái niệm. Mức này sẽ không chứa đựng những thông tin liên quan tới sự phụ thuộc vào cách lưu trữ vật lý trong hệ thống. Ví dụ, một thực thể có thể được xác định bằng cách thể hiện nó như một số nguyên tại mức này nhưng số byte mà nó chiếm giữ không cần được xác định cụ thể tại mức này.

Mức trùu tượng bên trong thể hiện sự biểu diễn vật lý của cơ sở dữ liệu trong máy tính. Mức này sẽ mô tả cách thức dữ liệu được lưu trữ trong cơ sở dữ liệu. Mức trùu tượng này cũng sẽ mô tả sự cài đặt vật lý cần thiết để đạt được công suất chạy tối ưu và việc sử dụng không gian lưu trữ tốt nhất. Nó bao gồm cả các cấu trúc dữ liệu và việc tổ chức các tệp dữ liệu được sử dụng để lưu dữ liệu trong các thiết bị lưu trữ. Nó cung cấp giao diện với các phương thức truy nhập của hệ điều hành (Đó là các kỹ thuật quản lý tệp liên quan tới việc lưu trữ và lấy ra các bản ghi dữ liệu) để đưa dữ liệu vào các thiết bị lưu trữ, xây dựng các tệp chỉ mục, lấy dữ liệu ra và các công việc khác.

Mức trùu tượng vật lý: là mức nằm bên dưới mức bên trong, mức này được quản lý bởi hệ điều hành dưới sự chỉ dẫn của hệ quản trị cơ sở dữ liệu. Các chức năng của hệ quản trị cơ sở dữ liệu và hệ điều hành ở mức vật lý này không có ranh giới rõ ràng và sẽ thay đổi từ hệ thống này sang hệ thống khác. Một vài hệ quản trị cơ sở dữ liệu tận dụng các ưu điểm của các phương pháp truy nhập của hệ điều hành, trong khi một số hệ quản trị khác sẽ chỉ sử dụng các phương pháp cơ bản và tự tạo ra những kiểu tổ chức tệp của riêng chúng.

Mức trùu tượng vật lý dưới hệ quản trị cơ sở dữ liệu bao gồm các thành phần mà chỉ được biết đến bởi hệ điều hành, ví dụ như việc tạo chuỗi thực hiện các công việc được diễn ra thế nào và liệu các trường của một bản ghi trong cơ sở dữ liệu có được lưu trữ thành các byte liền nhau trên đĩa không.

Khái niệm lược đồ, ánh xạ và thể hiện của cơ sở dữ liệu

Lược đồ cơ sở dữ liệu là một mô tả tổng quát toàn bộ cơ sở dữ liệu. Có ba loại lược đồ khác nhau và chúng được định nghĩa dựa trên các mức độ trùu tượng của kiến trúc trùu tượng ba lớp.

- Ở mức độ cao nhất, có rất nhiều lược đồ mức ngoài. Mỗi lược đồ này được gọi là một lược đồ con dữ liệu, liên quan tới các khung nhìn khác nhau của dữ liệu

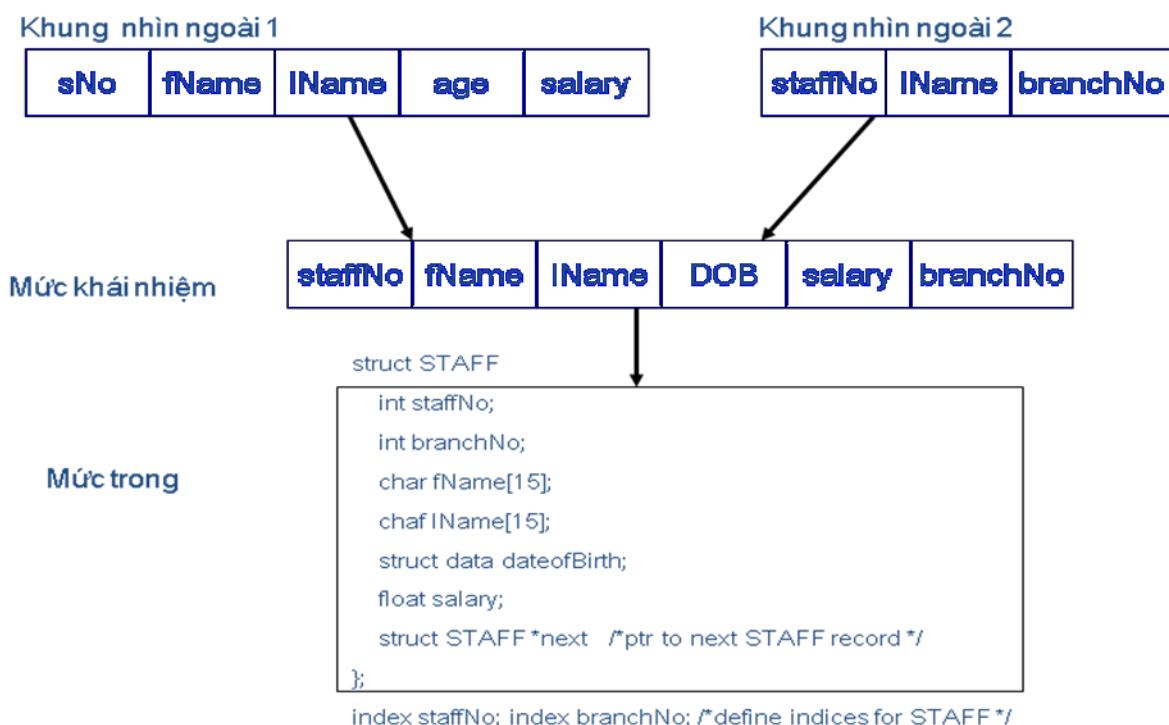
- Ở mức khái niệm có một lược đồ khái niệ, sẽ mô tả tất cả các thực thể, thuộc tính và các mối quan hệ cùng với những ràng buộc toàn vẹn của chúng.
- Mức trừu tượng thấp nhất sẽ có một lược đồ trong, sẽ là một mô tả hoàn thiện ở mô hình bên trong, bao gồm định nghĩa các bản ghi lưu trữ, các phương thức biểu diễn, v.v...

Hệ quản trị cơ sở dữ liệu chịu trách nhiệm ánh xạ giữa ba loại lược đồ này. Nó phải kiểm tra tính đồng nhất giữa ba loại đó; nói một cách khác, hệ quản trị cơ sở dữ liệu phải kiểm tra xem mỗi lược đồ ngoài có thể suy ra từ lược đồ khái niệm và nó phải sử dụng các thông tin trong lược đồ khái niệm để ánh xạ từ lược đồ ngoài vào các thành phần trong lược đồ trong.

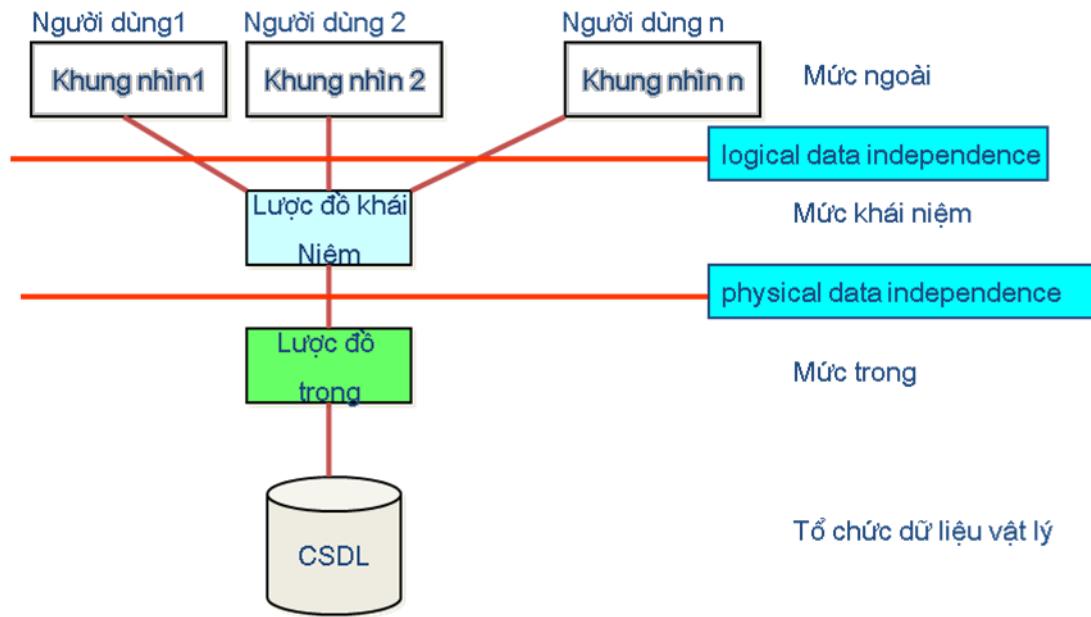
Lược đồ mức khái niệm có sự gắn kết với lược đồ bên trong thông qua một chuyển đổi được gọi là phép ánh xạ mức khái niệm/mức trong. Việc ánh xạ này cho phép hệ quản trị cơ sở dữ liệu tìm thấy bản ghi thực tế hoặc sự kết hợp của các bản ghi trong bộ lưu trữ vật lý đóng góp một bản ghi logic trong lược đồ khái niệm, cùng với các ràng buộc được gắn vào các phép toán liên quan tới bản ghi đó.

Mỗi lược đồ bên ngoài có liên kết với lược đồ khái niệm với một phép chuyển đổi được gọi là phép ánh xạ bên ngoài/khai niệm. Phép chuyển đổi này cho phép hệ quản trị cơ sở dữ liệu ánh xạ các tên trong khung nhìn của người dùng lên các phần liên quan với lược đồ khái niệm.

Ví dụ về việc chuyển đổi lược đồ giữa các mức trừu tượng được thể hiện như sau



Sự độc lập dữ liệu



Một trong những mục tiêu chính của việc đưa ra kiến trúc trùu tượng 3 lớp là để cung cấp sự độc lập về dữ liệu, có nghĩa là lược đồ ở mức cao sẽ không bị ảnh hưởng bởi sự thay đổi của lược đồ ở mức thấp hơn.

Tương ứng với ba mức độ trùu tượng sẽ có hai loại độc lập dữ liệu: độc lập dữ liệu mức logic (mức khái niệm) và mức vật lý.

Độc lập dữ liệu mức khái niệm thể hiện ở sự không bị ảnh hưởng của các lược đồ mức ngoài đối với các thay đổi của lược đồ khái niệm. Điều này có nghĩa là nếu các thực thể, hay thuộc tính ở lược đồ mức khái niệm thay đổi, thì cũng không làm thay đổi nội dung các khung nhìn của người sử dụng.

Độc lập dữ liệu mức vật lý thể hiện ở sự không bị ảnh hưởng của lược đồ khái niệm đối với những thay đổi của lược đồ mức vật lý.

Các ngôn ngữ cơ sở dữ liệu

Một ngôn ngữ con dữ liệu bao gồm hai phần: một ngôn ngữ định nghĩa dữ liệu (Data Definition Language) và một ngôn ngữ thao tác dữ liệu (Data Manipulation Language).

Ngôn ngữ định nghĩa dữ liệu được dùng để xác định lược đồ cơ sở dữ liệu chẵng hạn dùng để định nghĩa một quan hệ (một thực thể), các thuộc tính của thực thể đó, các kiểu dữ liệu của mỗi thuộc tính. Ngôn ngữ thao tác dữ liệu được dùng để đọc và cập nhật dữ liệu. Các ngôn ngữ này được gọi là ngôn ngữ con dữ liệu bởi vì chúng không bao gồm các cấu trúc lập trình cần thiết cho việc tính toán như cấu trúc điều kiện hoặc câu lệnh lặp, những cấu trúc được cung cấp bởi các ngôn ngữ lập trình bậc cao.

Hầu hết các hệ quản trị cơ sở dữ liệu đều có một môi trường cho phép nhúng các ngôn ngữ con vào ngôn ngữ lập trình mức cao như COBOL, Pascal, C, C++, Java hoặc Visual Basic, những loại ngôn ngữ lập trình bậc cao này được gọi là ngôn ngữ chủ. Hầu hết các ngôn ngữ con cũng cung cấp một phiên bản tương tác hay không nhúng vào ngôn ngữ khác mà được đưa vào trực tiếp từ một thiết bị đầu cuối.

Ngôn ngữ định nghĩa dữ liệu là một ngôn ngữ cho phép một quản trị cơ sở dữ liệu DBA hoặc người dùng mô tả và đặt tên các thực thể, các thuộc tính và các mối quan hệ cần thiết cho ứng dụng, cùng với các ràng buộc về bảo mật và toàn vẹn liên quan. Kết quả của việc thực thi hay biên dịch một câu lệnh định nghĩa dữ liệu là một tập các bảng được thu thập lưu trữ trong các tệp đặc biệt được gọi tên là bảng liệt kê các thành phần của hệ thống (system catalog). Tệp này cũng thường được gọi với cái tên là **tùy diễn dữ liệu** hay **thư mục dữ liệu**.

Ngôn ngữ thao tác dữ liệu là một ngôn ngữ cung cấp một tập các thao tác hỗ trợ cho các phép toán thao tác dữ liệu trên các dữ liệu được lưu trữ trong cơ sở dữ liệu. Các thao tác của một ngôn ngữ thao tác dữ liệu thường bao gồm

- Chèn thêm một dữ liệu mới vào cơ sở dữ liệu
- Thay đổi dữ liệu được lưu trữ trong cơ sở dữ liệu
- Lấy dữ liệu từ cơ sở dữ liệu ra
- Xóa dữ liệu từ cơ sở dữ liệu

Ngôn ngữ truy vấn là một phần quan trọng của ngôn ngữ thao tác dữ liệu liên quan tới việc lấy dữ liệu ra từ cơ sở dữ liệu. Các ngôn ngữ thao tác dữ liệu được phân biệt bởi các cấu trúc lấy dữ liệu bên trong của nó, và được phân chia làm hai loại chính: có thủ tục và không thủ tục.

Loại ngôn ngữ thao tác dữ liệu có thủ tục là các ngôn ngữ mà trong đó người dùng có thông báo với hệ thống những dữ liệu nào cần thiết và cách thức chính xác để lấy dữ liệu ra. Loại ngôn ngữ thao tác dữ liệu không có thủ tục là các ngôn ngữ trong đó người dùng chỉ thông báo cho hệ thống dữ liệu nào được yêu cầu và để hệ thống tự xác định cách thức lấy dữ liệu đó ra cho người sử dụng. Thông thường thì các ngôn ngữ thao tác dữ liệu có thủ tục sẽ được nhúng vào các ngôn ngữ lập trình mức cao. Các ngôn ngữ thao tác dữ liệu có thủ tục có xu hướng tập trung vào từng bản ghi đơn trong khi loại ngôn ngữ không thủ tục có xu hướng thực hiện trên một tập các bản ghi.

Các ngôn ngữ thế hệ thứ tư

Chúng ta không có khái niệm về cái gì góp phần vào hình thành một ngôn ngữ thế hệ thứ tư. Cảm giác chung nó là một ngôn ngữ lập trình rất nhanh, những yêu cầu được thực hiện với hàng trăm dòng lệnh trong ngôn ngữ thế hệ thứ ba sẽ được thể hiện chỉ trong một vài dòng mã nguồn của ngôn ngữ thế hệ thứ tư.

Ngôn ngữ thế hệ thứ ba là thuộc loại thủ tục, trong khi ngôn ngữ thế hệ thứ tư là loại không có thủ tục (đặc tính của không có thủ tục được trình bày rõ ở trên). Loại ngôn ngữ thế hệ thứ tư bao gồm các ngôn ngữ làm việc trên bảng tính và trên cơ sở dữ liệu. SQL và QBE (hai loại ngôn ngữ con dữ liệu liên quan tới bài giảng này) là hai ví dụ của ngôn ngữ thế hệ thứ tư.

Bài 3 và 4: Các mô hình dữ liệu

Giới thiệu chung

Một mô hình dữ liệu là một tập tích hợp các khái niệm nhận thức để mô tả và thao tác dữ liệu, các mối quan hệ giữa dữ liệu và các ràng buộc trên dữ liệu của một tổ chức.

Một mô hình dữ liệu là một cách biểu diễn các đối tượng trong thế giới thực và các sự kiện cũng như các liên hệ giữa chúng. Nó là một khái niệm trừu tượng tập trung vào những khía cạnh cần thiết, sống còn của một tổ chức và cần bù qua những thuộc tính ngẫu nhiên.

Một mô hình dữ liệu phải cung cấp các khái niệm cơ bản và các ký pháp cho phép người thiết kế cơ sở dữ liệu và người dùng trao đổi với nhau những hiểu biết về dữ liệu của tổ chức một cách chính xác và không đa nghĩa.

Một mô hình dữ liệu có thể được cho rằng cấu thành bởi ba thành phần:

- 1- Một thành phần cấu trúc bao gồm một tập các luật mà dựa trên đó các cơ sở dữ liệu được thiết lập.
- 2- Một thành phần thao tác, định nghĩa các loại thao tác được phép thực hiện trên dữ liệu (bao gồm cả những thao tác được dùng để cập nhật hoặc thu thập dữ liệu từ cơ sở dữ liệu và cho việc thay đổi cấu trúc cơ sở dữ liệu)
- 3- Có thể có một tập các luật thể hiện tính toàn vẹn để đảm bảo rằng dữ liệu được lưu trữ trong cơ sở dữ liệu là hoàn toàn chính xác tuyệt đối.

Dựa vào kiến trúc trừu tượng ba lớp được trình bày ở trên, chúng ta có thể xác định ba loại mô hình dữ liệu khác nhau tương ứng.

- 1- Một mô hình dữ liệu mức ngoài để biểu diễn khung nhìn của từng người dùng trong tổ chức.
- 2- Một mô hình dữ liệu mức khái niệm để biểu diễn khung nhìn mức logic, độc lập với hệ quản trị cơ sở dữ liệu cụ thể.
- 3- Một mô hình dữ liệu bên trong để biểu diễn lược đồ mức khái niệm theo một cách thức mà hệ quản trị cơ sở dữ liệu có thể hiểu được.

Nhiều loại mô hình dữ liệu khác nhau tồn tại, chúng đã được lý thuyết hóa, được sử dụng, được phát triển và được cài đặt qua nhiều năm. Chúng được phân chia thành ba loại chính: hướng đối tượng, hướng bản ghi và vật lý. Có nhiều mô hình dữ liệu hướng bản ghi bao gồm mô hình dữ liệu quan hệ, mô hình dữ liệu mạng, mô hình dữ liệu phân cấp. Chúng ta chỉ tập trung một mô hình dữ liệu quan hệ trong bài giảng này.

Các mô hình dữ liệu ngữ nghĩa thường cố gắng thể hiện ý nghĩa của một cơ sở dữ liệu. Thực tế, chúng cung cấp một cách tiến cận tới việc mô hình hóa dữ liệu mức khái niệm. Qua nhiều năm, đã có một vài mô hình dữ liệu ngữ nghĩa khác nhau được đề xuất. Trong số những mô hình đó, mô hình dữ liệu thực thể liên kết được sử dụng phổ biến nhất, thường được biết đến với cái tên đơn giản là mô hình dữ liệu E-R.

Mô hình E-R thường được sử dụng làm phương tiện trao đổi giữa những người thiết kế cơ sở dữ liệu và những người sử dụng cuối trong các quá trình phát triển một cơ sở dữ liệu. Nó bao gồm một tập các công cụ thiết lập mô hình mở rộng, mà một số trong đó chúng ta sẽ không tìm hiểu như mục tiêu chính của bài học. Mô hình này thể hiện cho chúng ta một bức tranh bên trong về thiết kế cơ sở dữ liệu khái niệm, chúng ta sẽ không nhắc đến các đầu vào và đầu ra của mô hình E-R.

Một mô hình khái niệm khác cũng đang trở nên phổ biến hơn là ngôn ngữ định nghĩa đối tượng (ODL), một cách tiếp cận hướng đối tượng để thiết kế cơ sở dữ liệu, và đang phát triển thành một chuẩn cho các hệ thống cơ sở dữ liệu hướng đối tượng.

Quá trình thiết kế một cơ sở dữ liệu có thể chia thành sáu bước cơ bản sau. Các mô hình ngữ nghĩa dữ liệu liên quan nhiều tới ba bước đầu tiên.

Bước 1. Phân tích yêu cầu của bài toán: Đây là bước đầu tiên trong quá trình thiết kế một ứng dụng cơ sở dữ liệu để có thể hiểu được dữ liệu nào cần được lưu trữ trong cơ sở dữ liệu, ứng dụng nào cần được xây dựng để sử dụng chúng, và các thao tác dữ liệu nào cần được thực hiện thường xuyên và các yêu cầu về tốc độ thực hiện của hệ thống. Đây thường là tiến trình không chính thức liên quan tới những trao đổi với các nhóm người dùng và nghiên cứu môi trường hiện tại. Tiến hành tìm hiểu các ứng dụng hiện có cần được thay thế hoặc bổ trợ cho hệ thống cơ sở dữ liệu.

Bước 2. Thiết kế cơ sở dữ liệu mức khái niệm: Thông tin được thu thập trong bước phân tích yêu cầu được dùng để phát triển một bản mô tả tổng quát các dữ liệu cần được lưu trữ trong cơ sở dữ liệu, cùng với các ràng buộc cần thiết trên những dữ liệu này.

Bước 3. Thiết kế cơ sở dữ liệu ở mức logic: Một hệ quản trị cơ sở dữ liệu phải được lựa chọn để cài đặt một cơ sở dữ liệu và để chuyển đổi bản thiết kế cơ sở dữ liệu mức khái niệm sang lược đồ cơ sở dữ liệu với mô hình dữ liệu của hệ quản trị cơ sở dữ liệu đã được lựa chọn.

Bước 4. Cải thiện các lược đồ: Trong bước này các lược đồ được phát triển ở bước 3 được phân tích để phát hiện ra các vấn đề tiềm ẩn. Tại bước này, các lược đồ sẽ được chuẩn hóa. Việc chuẩn hóa một cơ sở dữ liệu được dựa trên một lý thuyết toán học rất mạnh và đẹp đẽ. Chúng ta sẽ bàn luận về việc chuẩn hóa này vào những buổi giảng sau.

Bước 5. Thiết kế cơ sở dữ liệu vật lý: Tại giai đoạn này, khôi lượng công việc tiềm ẩn và các cách thức truy nhập được mô phỏng để xác định những điểm yếu tiềm ẩn trong cơ sở dữ

liệu khái niệm. Quá trình này thường là nguyên nhân tạo ra các tệp chỉ mục hoặc/và các quan hệ phân cụm. Trong các tình huống sống còn, toàn bộ mô hình khái niệm sẽ cần được cấu trúc lại.

Bước 6.Thiết kế an toàn bảo mật cho hệ thống: Các nhóm người dùng khác nhau được xác định và các vai trò khác nhau của họ được phân tích sao cho cách thức truy nhập tới dữ liệu có thể xác định được.

Thông thường quá trình phát triển sẽ có bước thứ bảy hay là bước cuối cùng, được gọi là giai đoạn chuốt lại hệ thống. Trong giai đoạn này, hệ thống cơ sở dữ liệu sẽ được thực hiện (mặc dù có thể nó chỉ được chạy trên chế độ mô phỏng) và sẽ được trau chuốt, cải thiện để đáp ứng được nhu cầu thực thi trong môi trường mong đợi. Hình vẽ dưới đây sẽ tóm tắt các bước chính trong quá trình thiết kế cơ sở dữ liệu.

Mô hình thực thể liên kết

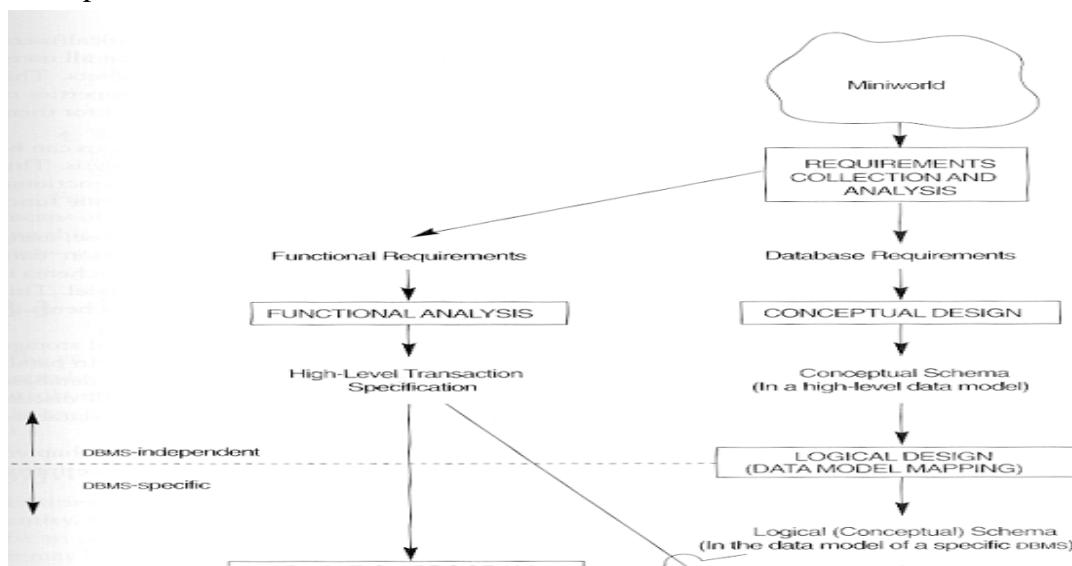
Mô hình này sử dụng ba khái niệm cơ bản là: tập các thực thể, tập các liên kết (mối quan hệ) và các thuộc tính.

Một thực thể là một điều gì đó hoặc một đối tượng nào đó trong thế giới thực mà có thể phân biệt được với các đối tượng khác. Một thực thể có thể hiện thực như là một người hoặc một quyển sách hoặc có thể trừu tượng như một khoản vay từ nhà băng hoặc một khái niệm nào đó.

Một thực thể được biểu diễn bởi một tập các thuộc tính. Các thuộc tính là các thuộc tính mô tả hoặc các đặc điểm được sở hữu bởi một thực thể. Một tập các thực thể là một tập hợp các thực thể cùng loại, có chung các thuộc tính. Ví dụ, tập hợp tất cả các người là khách hàng của một ngân hàng nào đó có thể được định nghĩa như một tập hợp các khách hàng.

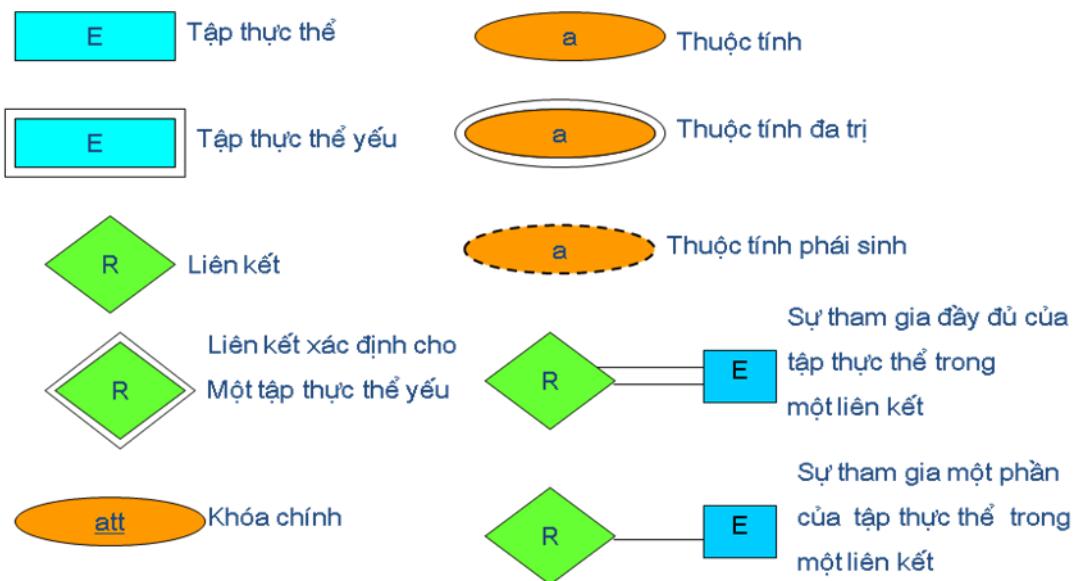
Các tập thực thể không nhất thiết phải không giao nhau. Ví dụ, chúng ta có thể định nghĩa tập các thực thể của tất cả những người là khách hàng của một ngân hàng. Một thực thể người nào đó có thể là một nhân viên, một khách hàng, hoặc cả hai hoặc không phải cả hai.

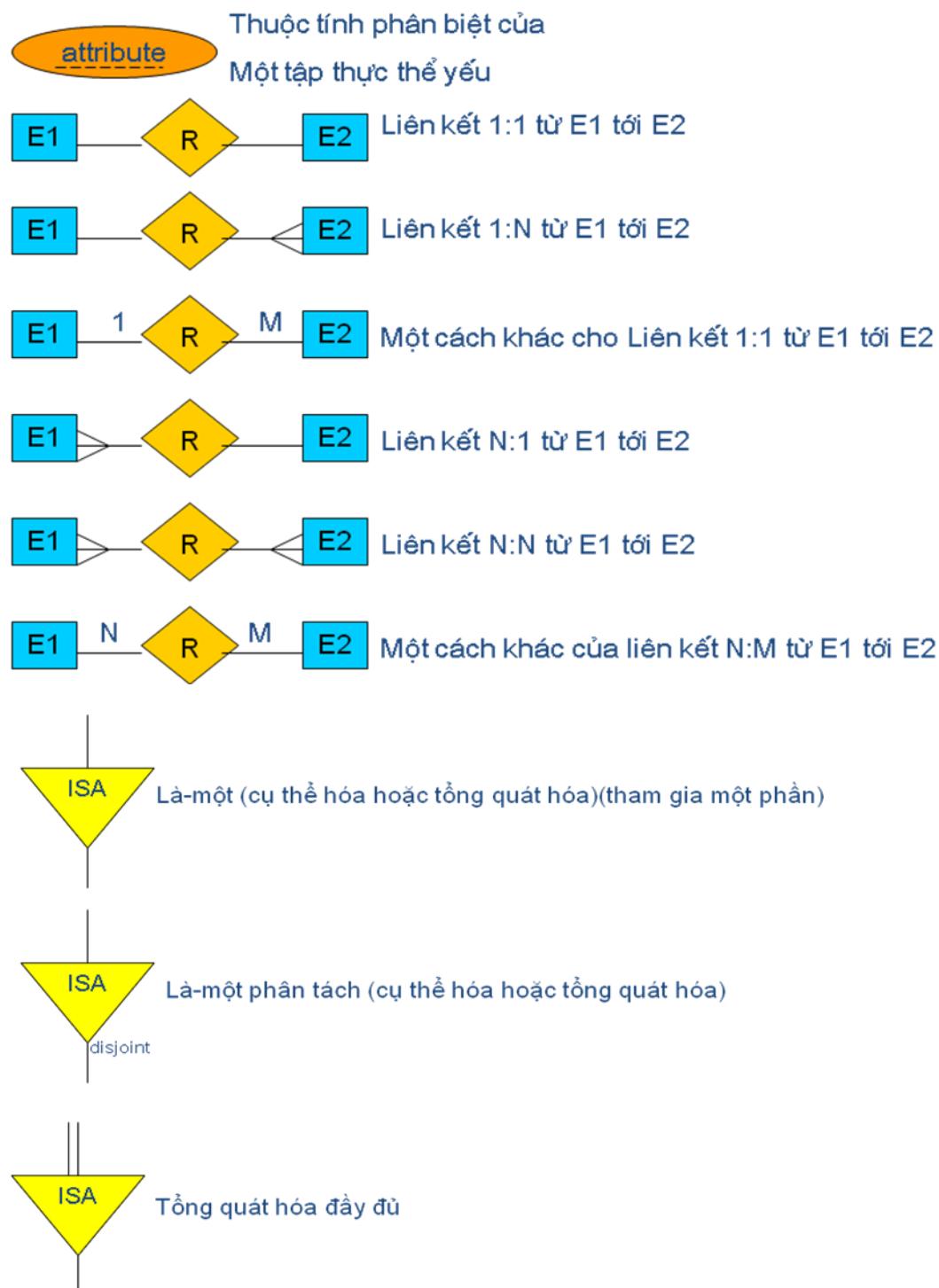
Với mỗi một thuộc tính, tập các giá trị cho phép của thuộc tính đó được gọi là miền giá trị của thuộc tính (đôi khi còn được gọi là tập giá trị). Một cách hình thức hơn, một thuộc tính của một tập các thực thể là một hàm ánh xạ từ một tập các thực thể vào một miền giá trị. Vì một tập

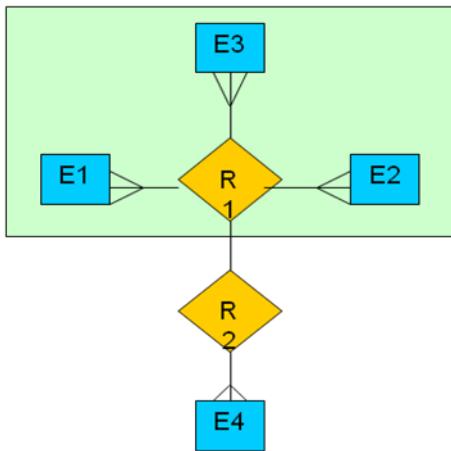


thực thể có thể có vài thuộc tính, mỗi thực thể trong tập có thể được mô tả bởi một tập có dạng một cặp < thuộc tính, giá trị dữ liệu>, mỗi cặp này cho mỗi thuộc tính của tập thực thể. Một cơ sở dữ liệu bao gồm một tập các thực thể.

Các ký pháp cho mô hình E-R được thể hiện trong hình vẽ dưới đây







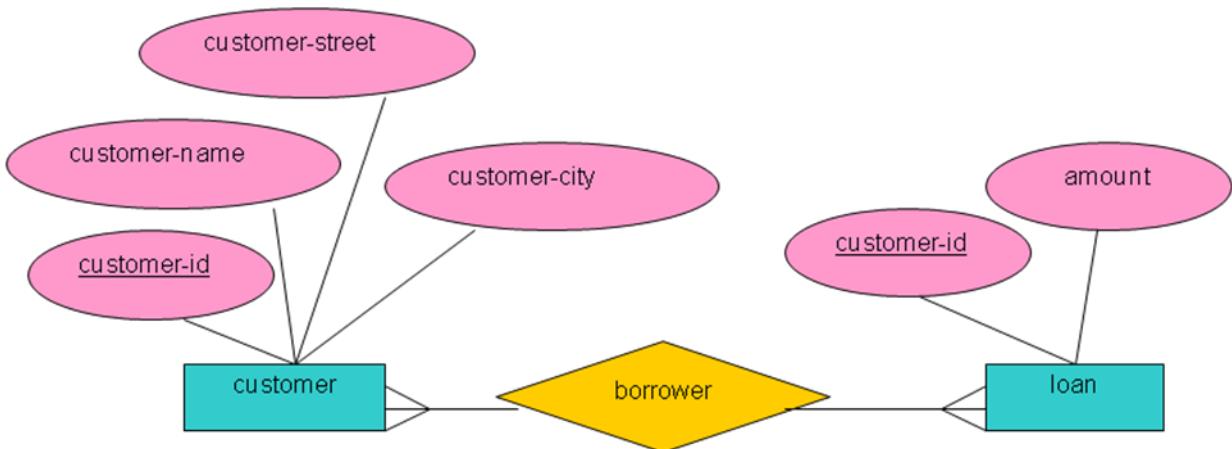
Kết ợp: hộp chữ nhật bao quanh một liên kết

Được coi như một thực thể

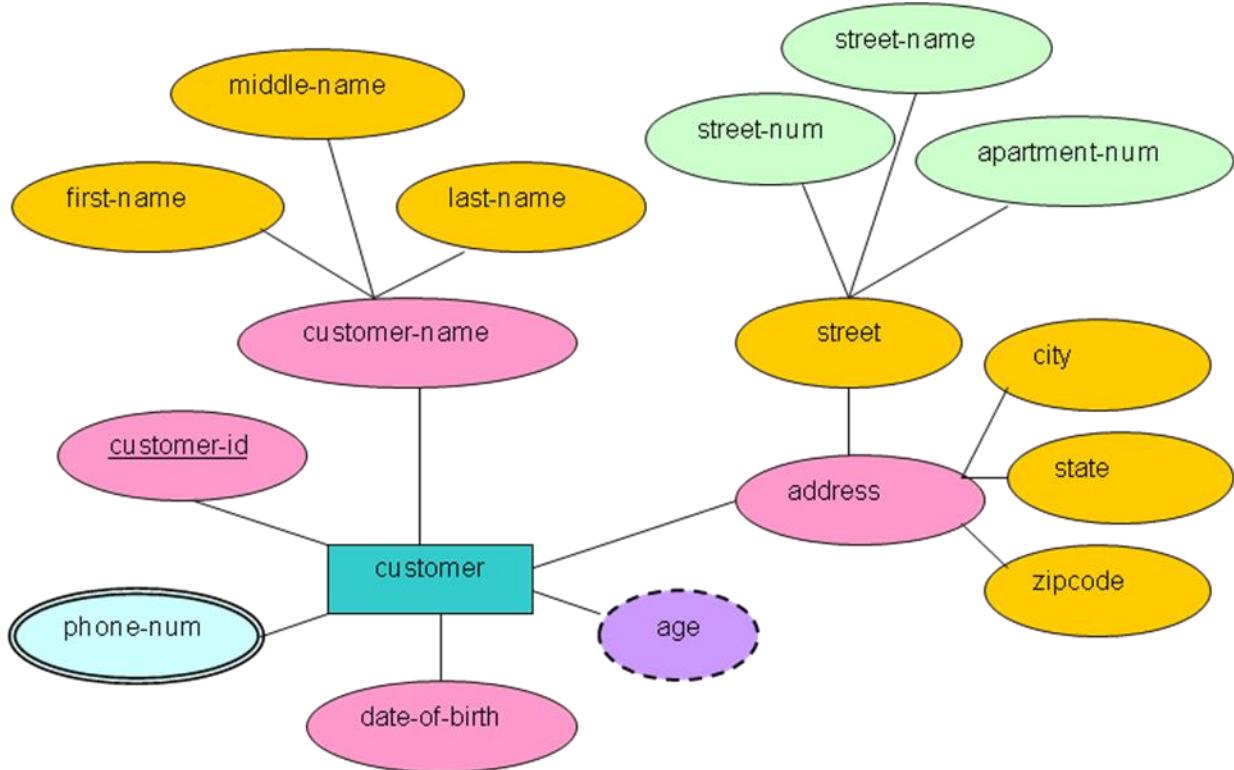
Ràng buộc có cấu trúc: (min, max) của

Số lượng thực thể tham gia một liên kết

Một ví dụ về lược đồ E-R (ERD) được thể hiện trong hình vẽ dưới đây



Một ví dụ khác về lược đồ E-R- phức tạp hơn



Các thuộc tính trong mô hình E-R

Nhu được sử dụng trong mô hình E-R, một thuộc tính có thể được phân chia làm hai loại:

- Thuộc tính đơn hoặc thuộc tính ghép: một thuộc tính đơn không bao gồm các thành phần cấu thành trong khi các thuộc tính ghép bao gồm các phần con cấu thành. Ví dụ, xem xét một thuộc tính tên. Nếu tên biểu diễn một thuộc tính đơn thì chúng ta có thể coi bộ ba cấu thành tên là họ, tên đệm và tên gọi là một thuộc tính nguyên tố, không phân chia được nữa. Mặt khác, nếu coi tên là một thuộc tính ghép thì chúng ta có lựa chọn thao tác với thuộc tính này là một tên đầy đủ hoặc có thể thao tác với từng thành phần cấu thành của tên. Ví dụ, chúng ta có thể chỉ thao tác với tên họ, cái mà không thể thực hiện với một thuộc tính đơn.
- Thuộc tính đơn trị hoặc đa trị: Một thuộc tính đơn trị có thể có nhiều nhất một giá trị tại một thời điểm thể hiện cụ thể. Một thuộc tính đa trị có thể có nhiều giá trị khác nhau tại một thời điểm cụ thể. Ví dụ, xem xét một lớp học tại Học viện công nghệ bưu chính viễn thông (giả sử học theo hệ thống tín chỉ, mỗi môn học được tổ chức thành một lớp trong một kỳ học nào đó). Tại một thời điểm nào đó, số sinh viên đăng ký học một lớp (theo hệ thống học tín chỉ) là đơn trị, tạm cho là nhận giá trị 100, nhưng không thể nhận giá trị đa trị vừa là 100, 80 và 45! Mặt khác, một vài thuộc tính có thể chứa nhiều giá trị cùng một lúc. Một ví dụ là thuộc tính miêu tả số điện thoại của một sinh viên, tại một thời điểm nào đó một sinh viên có thể có một

vài số điện thoại khác nhau vì vậy thuộc tính đa trị là tốt nhất mô tả chính xác mô hình này. Thường thì hay thiết lập cả cận trên và cận dưới của các giá trị cho một thuộc tính tại một thời điểm.

- Thuộc tính phái sinh: Đây là một thuộc tính mà giá trị của nó được phát sinh (hay được tính toán) từ các giá trị của các thuộc tính liên quan hoặc các thực thể liên quan. Ví dụ, giả sử rằng thực thể khách hàng của một ngân hàng bao gồm một thuộc tính tên là loans-held, chứa số lượng các khoản vay của một khách hàng từ một ngân hàng. Giá trị của thuộc tính này có thể được tính bằng cách đếm số lượng thực thể các khoản vay liên quan tới mỗi khách hàng.
- Thuộc tính rỗng: một thuộc tính nhận giá trị rỗng khi một thực thể không có giá trị cho nó. Giá trị rỗng thường là các trường hợp đặc biệt có thể kiểm soát theo các cách khác nhau dựa vào từng tình huống. Ví dụ có thể phiên dịch ngữ nghĩa của giá trị rỗng này như sau: thuộc tính không được áp dụng cho thực thể này hoặc thực thể này có giá trị cho thuộc tính đó nhưng chúng ta không biết giá trị đó. Chúng ta sẽ xem xét các hệ thống khách nhau xử lý các giá trị rỗng như thế nào và các cách hiểu khác nhau liên quan tới giá trị đặc biệt này

Các liên kết trong mô hình E-R

Một mối quan hệ hay liên kết là một sự liên hệ giữa một vài thực thể. Ví dụ chúng ta có thể định nghĩa một mối liên kết thể hiện bạn là một sinh viên của một lớp học nào đó. Mối quan hệ này xác nhận rằng bạn đã đăng ký học lớp đó.

Về mặt hình thức, một mối quan hệ là một tập các quan hệ cùng loại. Nó là một quan hệ toán học của n tập thực thể (có thể giao nhau, $n \geq 2$)

Nếu E_1, E_2, \dots, E_n là các tập thực thể thì một tập quan hệ R là một tập con của:

$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\} \text{ Với } (e_1, e_2, \dots, e_n) \text{ là một quan hệ.}$$

Mỗi liên hệ giữa các tập thực thể được gọi là sự tham gia; có nghĩa là các tập thực thể E_1, E_2, \dots, E_n tham gia vào quan hệ R

Một thể hiện quan hệ trong một lược đồ E-R biểu diễn một mối liên hệ giữa các thực thể được định danh trong một tập đoàn đang được mô hình hóa trong thực tế.

Một mối quan hệ cũng có thể có các thuộc tính được gọi là các thuộc tính mô tả. Ví dụ, xem xét lại ngữ cảnh ngân hàng, giả sử rằng chúng ta có một tập các mối quan hệ tên là depositor (đặt cọc) với các tập thực thể customer (khách hàng) và account (tài khoản). Chúng ta có thể muốn liên hệ với mối quan hệ depositor một thuộc tính mô tả có tên là access-date (ngày truy nhập) để mô tả ngày gần nhất mà một khách hàng truy nhập vào tài khoản của họ.

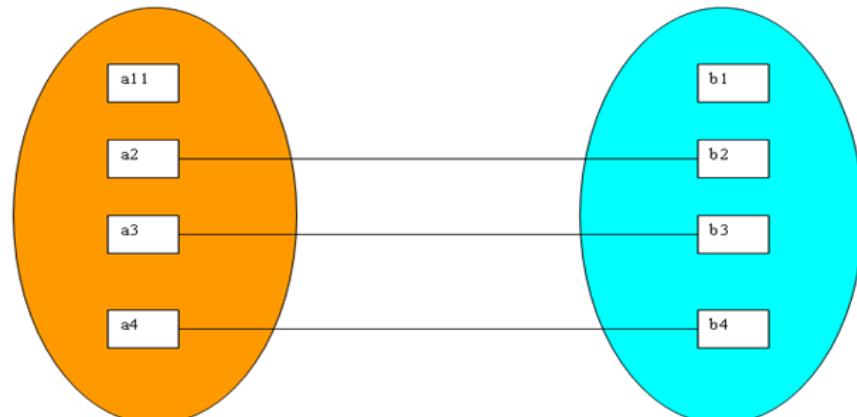
Như đã đề cập đến trước đây, một giá trị được lưu trữ trong một cơ sở dữ liệu nào đó thường có những ràng buộc được thiết lập để đảm bảo rằng chúng mô hình hóa toàn bộ thế giới thực của tập đoàn đang được thể hiện trong cơ sở dữ liệu một cách chính xác. Mô hình E-R có khả năng mô hình các loại ràng buộc này. Chúng ta sẽ tập trung vào hai loại ràng buộc quan trọng là ánh xạ lực lượng liên kết giữa các tập thực thể trong mối quan hệ và ràng buộc về số lượng thực thể tham gia vào mỗi quan hệ.

Các ràng buộc ánh xạ lực lượng liên kết (mapping cardinality) trong mô hình E-R

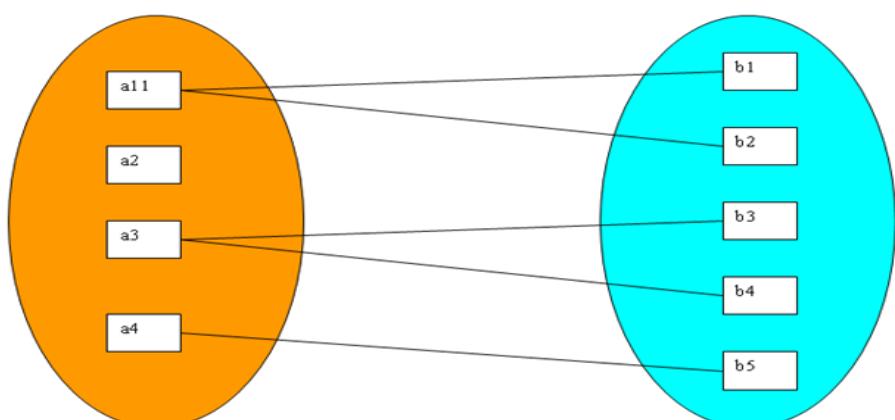
Ràng buộc về ánh xạ lực lượng liên kết hiện số lượng các thực thể mà một thực thể khác có thể liên hệ với thông qua một tập quan hệ (liên kết). Ràng buộc này có ích nhất trong khi mô tả các mối quan hệ hai ngôi, mặc dù chúng có thể có ích trong mô tả các quan hệ liên quan tới nhiều hơn hai tập thực thể (quan hệ nhiều ngôi). Chúng ta sẽ chỉ tập trung vào các mối quan hệ hai ngôi tại thời điểm này.

Với một tập quan hệ hai ngôi R giữa tập thực thể A và B, ánh xạ lực lượng sẽ rơi vào một trong các loại sau:

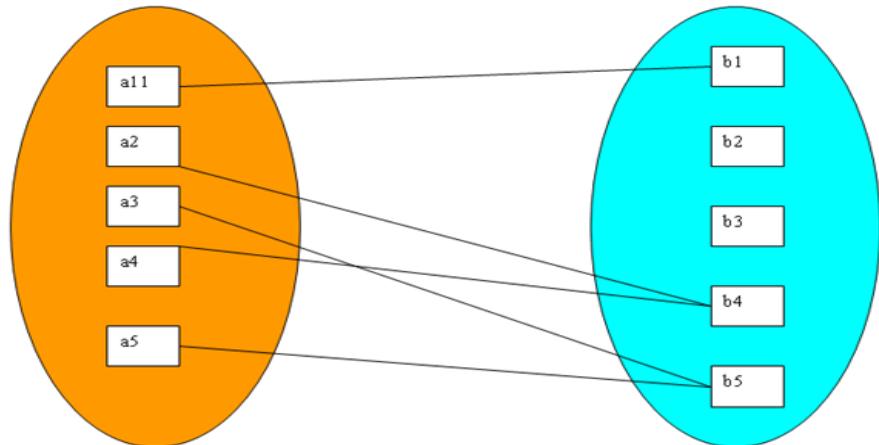
- Liên kết (1:1): một tới một từ A đến B



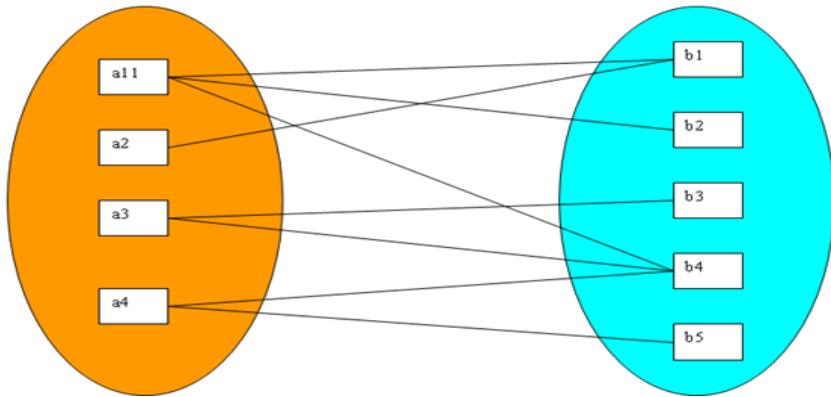
- Liên kết (1:N) hay (1:M): một tới nhiều từ A đến B



- Liên kết (N:1) hay (M:1): nhiều tới một từ A đến B.



- Liên kết (N:N) hay (M:M) hay (N:M): nhiều tới nhiều từ A đến B



Các ràng buộc tham gia trong mô hình E-R

Sự tham gia của một tập thực thể E trong một tập các mối quan hệ R được cho là đầy đủ nếu mọi thực thể của E tham gia vào ít nhất một mối quan hệ của R. Nếu chỉ có một vài thực thể của E tham gia vào một mối quan hệ trong R thì sự tham gia của tập thực thể E tới tập các mối quan hệ R được cho là một phần.

Ví dụ lại xem xét hệ thống ngân hàng, chúng ta thấy rằng mỗi thực thể khoản vay có liên hệ tới ít nhất một khách hàng qua mối quan hệ borrower (vay mượn). Vì vậy, sự tham gia của thực thể *loan* trong tập quan hệ borrower là đầy đủ. Ngược lại, một cá nhân có thể là một khách hàng của ngân hàng không liên quan tới chuyện họ có khoản vay ngân hàng này không. Vì vậy có thể nói rằng chỉ một số các thực thể customer (khách hàng) có liên hệ tới một thực thể *loan* thông qua mối quan hệ borrowers. Vì vậy, sự tham gia của thực thể customer trong mối quan hệ borrower là một phần.

Khóa của một tập thực thể

Chúng ta phải có một cơ chế để phân biệt các thực thể trong một tập các thực thể nào đó. Về mặt khái niệm, mỗi thực thể đơn đều khác nhau, từ một khía cạnh một cơ sở dữ liệu, tuy nhiên, các sự khác nhau giữa chúng phải được thể hiện thông qua các thuộc tính của chúng. Vì vậy, các giá trị của thuộc tính của một thực thể phải được xác định sao cho chúng có thể

xác định duy nhất thực thể đó. Nói một cách khác, không có hai thực thể nào trong một tập các thực thể được phép có giá trị của tất cả các thuộc tính là hoàn toàn giống nhau.

Một khóa cho phép chúng ta xác định một tập các thuộc tính đủ để phân biệt các thực thể với nhau. Khóa cũng giúp cho việc xác định duy nhất các mối quan hệ và vì vậy phân biệt các mối quan hệ với nhau.

Có một số khái niệm khóa bao gồm: Khóa chính, siêu khóa và khóa dự phòng.

Một siêu khóa là một tập gồm một hoặc nhiều thuộc tính, được lựa chọn, cho phép chúng ta xác định duy nhất một thực thể trong một tập các thực thể. Giả sử rằng chúng ta có một tập thực thể mô hình các sinh viên trong một lớp học với lược đồ sau:

Students(SS#, name, address, age, major, minor, gpa, spring-sch)

Trong số các thuộc tính liên quan tới mỗi sinh viên, ta phải xác định một tập các thuộc tính xác định duy nhất mỗi sinh viên. Ta định nghĩa tập thuộc tính này là (SS#, name, major, minor). Tập các thuộc tính này sẽ xác định một siêu khóa cho tập thực thể Students. Lưu ý rằng tập các thuộc tính (SS#, name) cũng xác định một siêu khóa cho tập thực thể này bởi vì với bộ thuộc tính thứ hai này chúng ta vẫn có thể xác định duy nhất mỗi sinh viên trong tập thực thể. Khái niệm về một siêu khóa là một định nghĩa không đầy đủ của một khóa bởi vì siêu khóa này còn chứa nhiều thuộc tính dư thừa, như trong ví dụ vừa xét.

Nếu một tập K là một siêu khóa của tập thực thể E thì mọi tập cha của K cũng là siêu khóa. Chúng ta chỉ quan tâm tới những siêu khóa thỏa mãn không có tập con nào của K là siêu khóa. Những siêu khóa nhỏ nhất kiểu như vậy được gọi là khóa dự bị (candidate key).

Với mỗi một tập thực thể E cho trước, nhiều tập khác nhau các thuộc tính tồn tại, thỏa mãn là khóa dự bị. Cho dù có một khóa dự bị hay có nhiều khóa dự bị, người thiết kế cơ sở dữ liệu chỉ chọn một khóa dự bị làm khóa chính, khái niệm mà mọi người thường gọi là khóa của một tập thực thể.

Một khóa (khóa chính, khóa dự bị hay siêu khóa) là một thuộc tính của tập thực thể chứ không phải của một thực thể nào đó. Bất kể hai thực thể khác nhau trong một tập thực thể sẽ không thể có cùng giá trị trên tất cả các thuộc tính cấu thành các thuộc tính khóa tại cùng một thời điểm. Ràng buộc này thể hiện trên các giá trị cho phép của một thực thể trong một tập được gọi là **ràng buộc khóa**

Người thiết kế cơ sở dữ liệu phải cẩn thận khi lựa chọn tập các thuộc tính cấu thành khóa của một tập thực thể để đảm bảo (1) chắn chắn rằng một tập các thuộc tính xác định duy nhất thực thể và (2) rằng tập thuộc tính khóa sẽ không bao giờ hoặc rất hiếm khi bị thay đổi.

Tập các mối quan hệ

Khóa chính của một tập thực thể cho phép bạn phân biệt các thực thể khác nhau trong một tập. Chúng ta phải có một cơ chế tương tự để cho phép chúng ta phân biệt các mối quan hệ trong một tập các mối quan hệ.

Cho R là một tập các mối quan hệ liên quan tới các tập thực thể E_1, E_2, \dots, E_n . Gọi K_i là tập các thuộc tính cấu thành khóa chính của tập thực thể E_i . Chúng ta giả thiết rằng:

- (1) Tên của tất cả các thuộc tính trong tất cả các khóa chính là duy nhất, điều này sẽ khiến cho ký pháp dễ hiểu hơn và trong thực tế, nếu các tên không duy nhất thì cũng không có vấn đề gì nghiêm trọng cả.
- (2) Mỗi tập thực thể sẽ chỉ tham dự một lần duy nhất trong mối quan hệ

Việc cấu thành khóa chính cho một tập các mối quan hệ phụ thuộc vào tập các thuộc tính liên quan tới tập các mối quan hệ R theo các cách sau đây:

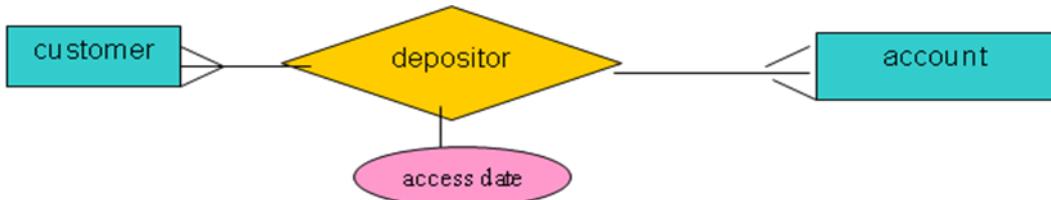
- (a) Nếu tập mối quan hệ R không có thuộc tính nào liên quan tới nó thì tập các thuộc tính: $K_1 \cup K_2 \cup \dots \cup K_n$ mô tả một mối quan hệ riêng biệt trong tập R .
- (b) Nếu tập mối quan hệ R có các thuộc tính liên quan a_1, a_2, \dots, a_m thì tập các thuộc tính $K_1 \cup K_2 \cup \dots \cup K_n \cup \{a_1, a_2, \dots, a_m\}$ sẽ mô tả một mối quan hệ riêng trong tập R .

Trong cả hai trường hợp, tập các thuộc tính $K_1 \cup K_2 \cup \dots \cup K_n$ đều hình thành một siêu khóa cho tập các mối quan hệ.

Một số vấn đề cần quan tâm khi thiết kế mô hình E-R

Ảnh hưởng của ràng buộc ánh xạ lực lượng liên kết lên các khóa

Cấu trúc của khóa chính cho tập các mối quan hệ phụ thuộc vào sự ánh xạ lực lượng liên kết. Xem xét trường hợp được thể hiện trong lược đồ sau đây:



Lược đồ E-R này thể hiện một quan hệ N-N cho quan hệ depositor với một thuộc tính access-date liên quan tới tập mối quan hệ giữa hai thực thể customer và account. Khóa chính của mối quan hệ này sẽ bao gồm hợp của các khóa chính của hai tập thực thể customer và account.

Để làm rõ hơn nữa tình huống này, chúng ta xem xét đến hai lược đồ dữ liệu của hai tập thực thể này như sau:

Customer (customer-id, customer-name, address, city)

Account (account-number, balance)

Một quan hệ N-N giữa hai tập thực thể này có nghĩa là một khách hàng có thể có nhiều tài khoản và tương tự như vậy một tài khoản có thể được quản lý bởi nhiều khách hàng. Để xác định duy nhất một mối quan hệ giữa hai thực thể trong customer và account, một phép hợp các khóa chính của cả hai tập thực thể cần thiết lập.

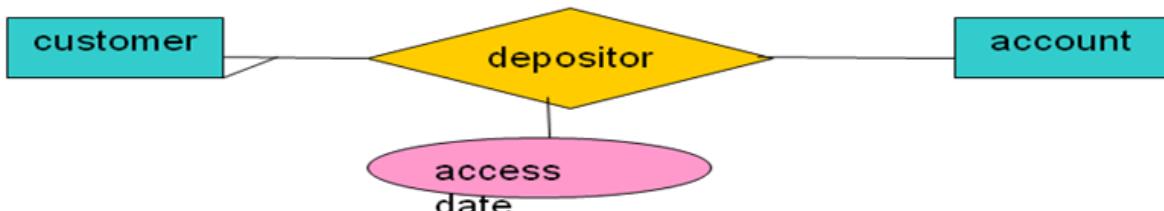
Để nhìn thấy lần chuyển tiền cuối cùng (last deposit) vào một tài khoản cụ thể nào đó, chúng ta cần xác định người chuyển tiền vì với tài khoản này có một số người có thể chuyển tiền vào. Lược đồ cho mối quan hệ depositor như sau:

Depositor (customer-id, account-number, access-date)

Bây giờ chúng ta xem xét trường hợp một khách hàng chỉ được phép chuyển tiền vào một tài khoản duy nhất. Điều đó có nghĩa là mỗi quan hệ depositor sẽ là quan hệ N-1 từ customer tới account thể hiện trong hình vẽ dưới đây. Trong trường hợp này khóa chính của mối quan hệ depositor sẽ chỉ bao gồm khóa chính của tập thực thể customer. Để cho rõ hơn, xem xét lược đồ của các tập thực thể dưới đây

Customer (customer-id, customer-name, address, city)

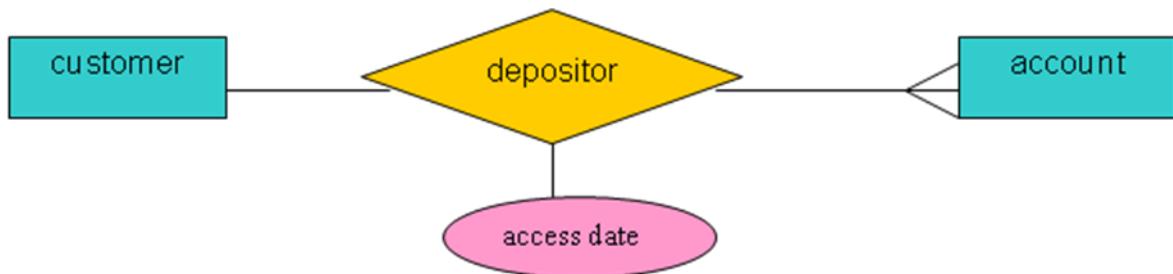
Account (account-number, balance)



Một mối quan hệ N-1 có nghĩa là một khách hàng nào đó chỉ có thể có duy nhất một tài khoản nên khóa chính của mối quan hệ depositor chỉ đơn giản là khóa chính của thực thể customer bởi vì với một khách hàng xác định họ chỉ có thể thực hiện một lần chuyển tiền gần nhất tới tài khoản duy nhất họ có thể truy nhập, vì thế việc chỉ ra số tài khoản cụ thể đó là không cần thiết để xác định một lần chuyển tiền duy nhất bởi người khách hàng đó. Lược đồ của tập các mối quan hệ depositor đó trong trường hợp này như sau

Depositor (customer-id, access-date)

Bây giờ xét đến trường hợp mối quan hệ depositor này là N-1 từ account tới customer.



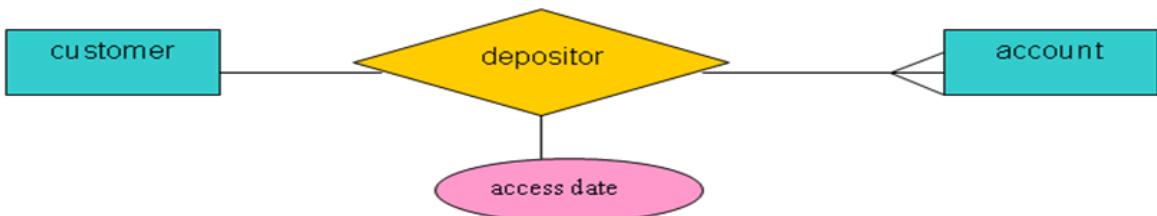
Một mối quan hệ N-1 từ account tới customer có nghĩa là mỗi tài khoản được làm chủ bởi nhiều nhất là một khách hàng nhưng mỗi khách hàng có thể có nhiều hơn một tài khoản. Trong tình huống này, khóa chính của mối quan hệ depositor chỉ đơn giản bao gồm khóa chính của thực thể account bởi vì chỉ có thể có nhiều nhất một lần chuyển tiền gần nhất tới một tài khoản nào đó xác định và chỉ có nhiều nhất một khách hàng có thể thực hiện việc chuyển tiền. Chúng ta không cần xác định duy nhất khách hàng nào đã thực hiện chuyển tiền vì chỉ có thể có một và chỉ một khách hàng có khả năng chuyển tiền vào một tài khoản xác định mà thôi. Lược đồ cho mối quan hệ depositor như sau:

Depositor (account-id, access-date)

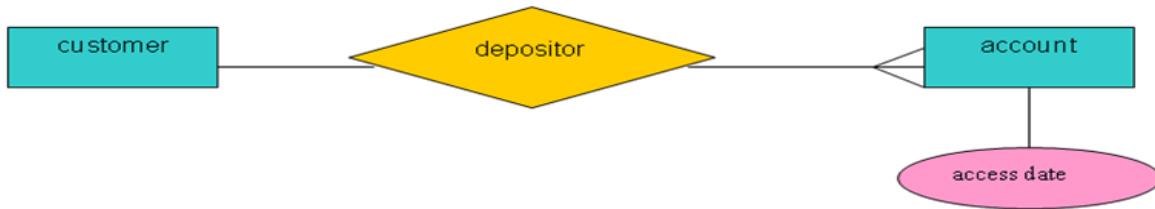
Vấn đề đặt vị trí cho các thuộc tính của mối quan hệ

Phần trước đã xem xét sự ảnh hưởng của lực lượng liên kết của một tập các mối quan hệ ảnh hưởng tới việc cấu thành một khóa chính cho tập các mối quan hệ đó. Tương tự trong phần này, chúng ta sẽ xem xét sự ảnh hưởng của yếu tố này tới việc đặt vị trí của các thuộc tính liên quan với mối quan hệ.

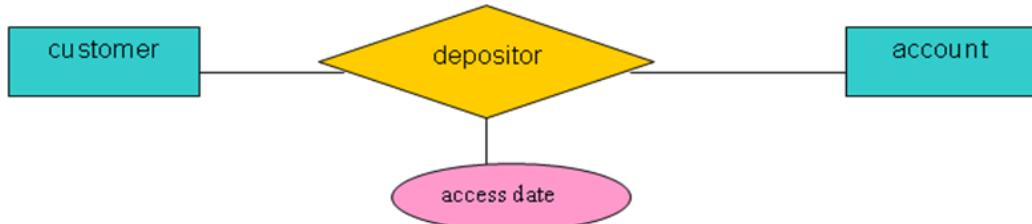
Các thuộc tính của một tập quan hệ dạng 1-1 hoặc 1-N có thể đặt vào một trong các tập thực thể tham gia quan hệ liên kết, hơn là được đặt vào bản thân tập các mối quan hệ đó. Ví dụ với quan hệ depositor thể hiện trong hình vẽ



Thuộc tính access-date có thể được đặt liên quan tới thực thể account mà không làm tổn thất thông tin như hình vẽ dưới đây. Sở dĩ làm được như vậy bởi vì một tài khoản cụ thể nào đó có thể thuộc sở hữu của nhiều nhất là một khách hàng, và tài khoản đó có thể có nhiều nhất một access-date, ngày được lưu trữ trong account.

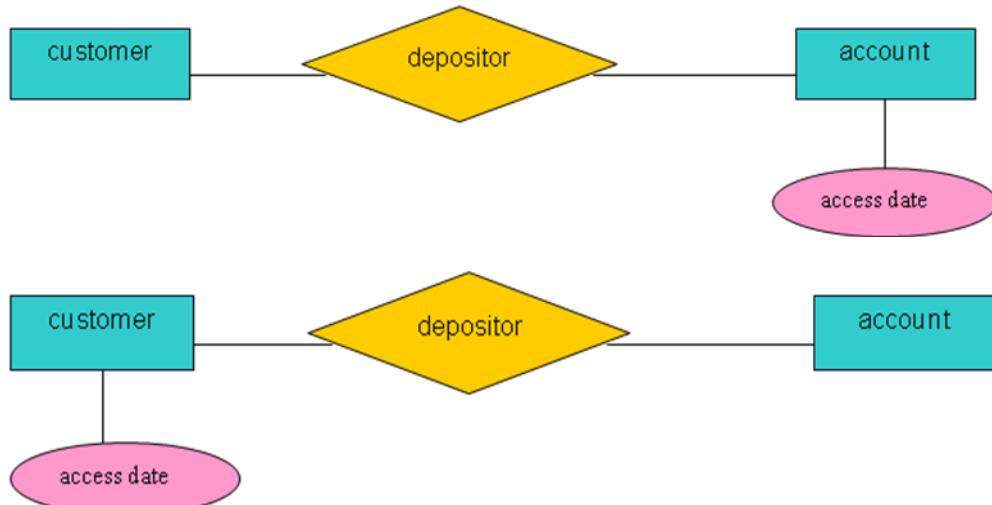


Tiếp đến xét đến trường hợp sau đây

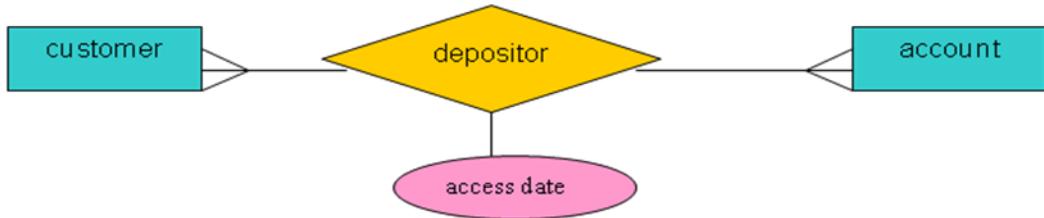


Thuộc tính access-date có thể gắn vào hoặc thực thể customer hoặc tập thực thể account mà không làm tổn thất thông tin. Trong tình huống này một tài khoản cụ thể nào đó có thể được làm chủ bởi nhiều nhất một khách hàng và một khách hàng nào đó có thể sở hữu duy nhất một tài khoản. Vì vậy, nếu thuộc tính access-date được lưu trữ với tập customer thì nó phải tham chiếu tới lần truy nhập cuối cùng của khách hàng đó tới tài khoản duy nhất mà họ có. Tương tự, nếu thuộc tính access-date được lưu trữ trong thực thể account thì nó sẽ tham chiếu tới lần truy nhập cuối cùng tới tài khoản đó bởi người khách hàng duy nhất sở hữu nó.

Vì vậy, một trong hai lược đồ sau đây đều đúng trong tình huống này



Khi tập các mối quan hệ này có ràng buộc N-N, tình huống này sẽ rõ ràng hơn nhiều. Tình huống được thể hiện trong hình vẽ dưới đây



Một tài khoản có thể được sở hữu bởi nhiều khách hàng, chúng ta thấy rằng gắn thuộc tính access-date với tập thực thể tham gia nào cũng không mô hình hóa được tình huống này mà không làm tổn thất thông tin. Nếu chúng ta cần lưu trữ ngày truy nhập cuối cùng của một khách hàng cụ thể nào đó tới một tài khoản cụ thể thì thuộc tính access-date này nhất thiết phải là một thuộc tính của tập mối quan hệ depositor, chứ không thể là thuộc tính của bất kể tập thực thể tham gia nào. Ví dụ, nếu access-date là một thuộc tính của account thì chúng ta không thể xác định được khách hàng nào đã thực hiện việc chuyển tiền vào tài khoản đó. Còn nếu access-date là một thuộc tính của customer, chúng ta cũng không thể xác định được tài khoản nào khách hàng đã truy nhập vào lần cuối.

Các vấn đề thiết kế khác cần xem xét

Khái niệm của một tập thực thể và một tập các mối quan hệ còn chưa rõ ràng. Chúng ta có thể định nghĩa một tập các thực thể và các mối quan hệ giữa chúng theo nhiều cách khác nhau. Ta sẽ cùng xem qua một số cách tiếp cận khác nhau để mô hình hóa dữ liệu.

Trong một phạm vi nào đó, đây là chỗ “nghệ thuật” thiết kế cơ sở dữ liệu thể hiện sự hiềm hóc của nó. Đôi khi một số kịch bản thiết kế khác nhau nhìn qua có vẻ như giống nhau và thậm chí cải thiện và chuẩn hóa vẫn phù hợp để tiến hành nhưng đôi khi không còn giống nhau nữa. Chỉ một thiết kế cẩn thận mới có thể loại bỏ những vấn đề mà chúng ta sẽ bàn tới sau đây.

Vấn đề tập thực thể hay các thuộc tính

Xem xét một tập thực thể

Employee(emp-name, telephone-number, age)

Việc một máy điện thoại telephone sẽ được coi là một thực thể (với các thuộc tính *telephone-number, location, manufacturer, serial-num*, và một số thuộc tính khác nữa) sẽ dễ dàng gây ra tranh cãi. Trong tình huống này thì thực thể Employee phải được định nghĩa lại như sau: *Employee (emp-name, age)*. Sau đó phải tạo ra một tập thực thể mới *Telephone(telephone-number, location, manufacturer, serial-num, ...)* và một tập các mối quan hệ phải được tạo ra để xác định mối liên hệ giữa các nhân viên và các máy điện thoại mà họ sở hữu *Emp-Phone(emp-name, telephone-number, age, location, manufacturer, serial-num)*.

Bây giờ chúng ta phải cân nhắc cái gì là sự khác nhau chính giữa hai cách định nghĩa thực thể Employee.

Việc coi máy điện thoại như một thuộc tính *telephone-number* ngụ ý rằng mỗi nhân viên sẽ có một số điện thoại duy nhất (chú ý rằng điều này là đúng đắn nếu không thuộc tính số điện thoại sẽ phải là một phần của khóa chính cho mỗi nhân viên và ở đây không đề cập tới các thuộc tính đa trị).

Việc coi máy điện thoại như một thực thể cho phép các nhân viên có thể sở hữu vài cái điện thoại (bao gồm cả 0 điện thoại). Tuy nhiên, chúng ta có thể dễ dàng chuyển thuộc tính *telephone-number* thành thuộc tính đa trị cho phép một nhân viên có nhiều máy điện thoại. Vì thế rõ ràng đây không phải là điểm khác biệt chính trong hai cách biểu diễn này.

Sự khác nhau chính ở đây là việc coi máy điện thoại như một thực thể mô hình hóa tình huống này tốt hơn vì như vậy chúng ta có thể muốn lưu trữ thêm thông tin về máy điện thoại đó như được thể hiện trong ví dụ trên. Nếu chúng ta sử dụng cách tiếp cận ban đầu và muốn biến điện thoại thành một thuộc tính của một nhân viên và chúng ta muốn duy trì những thông tin thêm này về máy điện thoại của họ thì thực thể Employee sẽ có lược đồ mới như sau:

Employee(emp-name, telephone-number, age, location, manufacturer,...)

Đây không phải là một lược đồ tốt vì thuộc tính *age* (tuổi) trong lược đồ trên liệu liên quan tới *employee* (nhân viên) hay tới *telephone* (máy điện thoại)? Trong tình huống này chúng ta đang cố gắng mô hình hóa hai tập thực thể khác nhau trong cùng một tập thực thể.

Ngược lại, nếu coi thuộc tính *emp-name* như một thực thể thì lại không thích hợp; khó có thể cho rằng tên của một nhân viên lại là một thực thể. Vì vậy, việc coi tên của một nhân viên *emp_name* như một thuộc tính của tập thực thể *Employee* là hoàn toàn thích hợp. Vậy cái gì nên coi là một thuộc tính và cái gì nên coi là một thực thể? Thật không may, vấn đề này không có một câu trả lời đơn giản. Sự phân biệt giữa hai vai trò này phụ thuộc chính vào cấu trúc của ngữ cảnh trong thế giới thực cần mô hình hóa dữ liệu và dựa vào ngữ nghĩa liên quan tới các thuộc tính trong bài toán.

Một lỗi chung thường gặp là việc sử dụng khóa chính của một tập thực thể như một thuộc tính của một thực thể khác để thể hiện mối liên kết giữa hai thực thể thay vì sử dụng một mối quan hệ. Quay lại ví dụ về ngân hàng, việc coi *customer_id* là một thuộc tính của thực thể *loan* là không hợp lý thậm chí ngay cả khi mỗi *loan* chỉ có duy nhất một khách hàng liên quan tới nó. Quan hệ *borrower* là một cách đúng đắn để biểu diễn mối quan hệ giữa *loan* và *customer* vì nó sẽ tạo liên kết một cách tường minh hơn là cách thể hiện không tường minh thông qua một thuộc tính.

Việc coi một đối tượng là một tập thực thể hay tập mối quan hệ

Việc thể hiện một đối tượng tốt nhất dưới dạng một tập thực thể hay một tập mối quan hệ thường không rõ ràng. Xem xét ví dụ về ngân hàng. Chúng ta đã mô hình một khoản vay như một thực thể. Một lựa chọn khác là mô hình khoản vay *loan* như một mối quan hệ giữa khách hàng và các chi nhánh branches của ngân hàng với *loan-number* và *amount* là các thuộc tính mô tả. Mỗi khoản vay được biểu diễn như một quan hệ giữa một khách hàng và một chi nhánh ngân hàng.

Nếu mỗi khoản vay chỉ được sở hữu bởi duy nhất một khách hàng và có liên hệ tới duy nhất một chi nhánh ngân hàng thì chúng ta có thể mô hình khoản vay là một mối quan hệ. Tuy nhiên, với thiết kế kiểu này chúng ta không thể biểu diễn thuận tiện cho tình huống mà nhiều khách hàng có thể cùng chung sở hữu một khoản vay. Để giải quyết tình huống này, chúng ta cần định nghĩa một mối quan hệ riêng cho mỗi người sở hữu khoản vay chung. Sau đó chúng ta cần dùng lại tất cả các giá trị cho các thuộc tính *loan-number* và *amount* trong mỗi mối quan hệ loại này. Hiển nhiên mỗi quan hệ đó phải có cùng giá trị cho các thuộc tính mô tả.

Hai vấn đề phát sinh như kết quả của việc dùng lặp lại các giá trị là:

1. Dữ liệu được lưu trữ ở nhiều nơi (đúng nghĩa của khái niệm dùng lặp lại)
2. Các cập nhật nhiều khả năng làm dữ liệu trong một trạng thái không đồng nhất nơi mà các giá trị trong hai tập khác nhau không có giá trị giống nhau mặc dù chúng phải giống hệt nhau. Chúng ta sẽ xem nhưng phức tạp mà việc dùng lặp lại giá trị này gây ra cũng như các kỹ thuật giải pháp (lý thuyết chuẩn hóa) sau đó trong bài giảng này. Chú ý là vấn đề này không xuất hiện trong phiên bản ban đầu bởi vì *loan* được biểu diễn bởi một tập thực thể trong trường hợp này.

Một hướng dẫn có thể có trong việc quyết định nên sử dụng một tập thực thể hay một tập các mối quan hệ là cần thiết kế một tập thực thể để mô tả một hành động xảy ra giữa các thực thể. Cách tiếp cận này cũng có ích trong việc quyết định các thuộc tính nào đó có thể thích hợp hơn với cách biểu diễn như là mối quan hệ.

Việc coi một đối tượng là tập thực thể yếu hay tập thực thể mạnh

Một tập thực thể có thể không đủ các thuộc tính để hình thành một khóa chính. Một tập thực thể như vậy được gọi với tên là thực thể yếu. Một tập thực thể có khóa chính được gọi là một tập thực thể khỏe mạnh. Một ví dụ về khái niệm này: cân nhắc một thực thể *payment*, có ba thuộc tính: *payment-number*, *payment date* và *payment amount*. Payment number thường là các số liên tiếp, bắt đầu từ 1 và được sinh ra riêng rẽ cho mỗi khoản nợ, Vì vậy, mặc dù mỗi thực thể *payment* là khác nhau, việc trả tiền cho các khoản nợ khác nhau có thể chung cùng một payment number (mã số trả tiền), vì vậy tập này không có một khóa chính và chỉ là một tập thực thể yếu.

Để một tập thực thể yếu trở nên có ý nghĩa, nó phải liên hệ với một tập thực thể khác được gọi là tập thực thể xác định hay tập thực thể sở hữu. Mỗi thực thể yếu phải được liên quan tới một tập thực thể xác định như vậy.

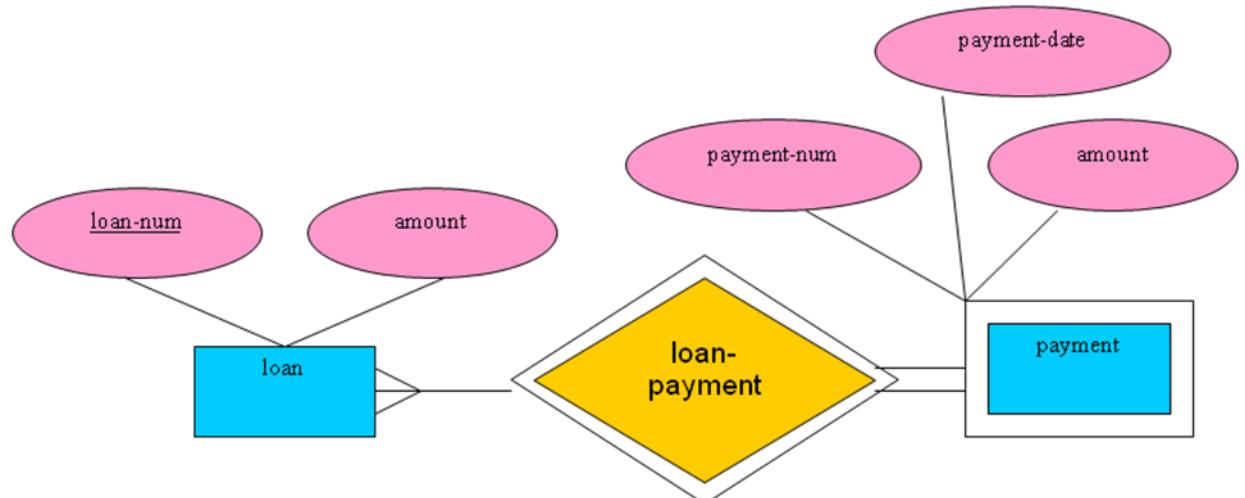
Thực thể yếu được nói là tồn tại phụ thuộc vào tập thực thể xác định. Tập xác định này được gọi là tập sở hữu thực thể yếu mà nó xác định. Mỗi quan hệ liên kết tập thực thể yếu với tập thực thể xác định được gọi là mối quan hệ xác định. Mỗi quan hệ này là quan hệ N-1 từ tập thực thể yếu tới tập xác định và sự tham gia của tập thực thể yếu trong quan hệ này là đầy đủ.

Mặc dù một tập thực thể yêu không có khóa chính, nhưng chúng ta cần một phương thức để phân biệt tất cả các thực thể này trong một tập thực thể yêu mà phụ thuộc vào một thực thể khỏe mạnh cụ thể nào đó. Tập các thuộc tính của một thực thể yêu cho phép phân biệt các thực thể được gọi là thuộc tính phân biệt (đôi khi được gọi là khóa bán phần). Ví dụ, thuộc tính phân biệt của tập thực thể *payment* ở trên là thuộc tính *payment-number*, vì với mỗi khoản nợ, một payment number xác định duy nhất một việc trả tiền riêng biệt cho khoản nợ này.

Khóa chính của một tập thực thể yếu được cấu thành bởi khóa chính của tập thực thể xác định công với thuộc tính phân biệt của tập thực thể yếu. Với trường hợp ở trên, khóa chính của tập thực thể *payment* sẽ là $\{loan-number, payment-number\}$, trong đó *loan-number* là khóa chính của tập thực thể xác định và *payment-number* là thuộc tính phân biệt của tập thực thể yếu *payment*.

Trong một lược đồ E-R, một tập thực thể yếu được biểu diễn bằng một hình chữ nhật với đường viền kép và quan hệ xác định cho một tập thực thể yếu được biểu diễn bằng một hình thoi có đường viền kép.

Một ví dụ về tập thực thể yếu được thể hiện trong hình vẽ dưới đây



Bài 5: Giới thiệu về mô hình hóa dữ liệu- phần 2

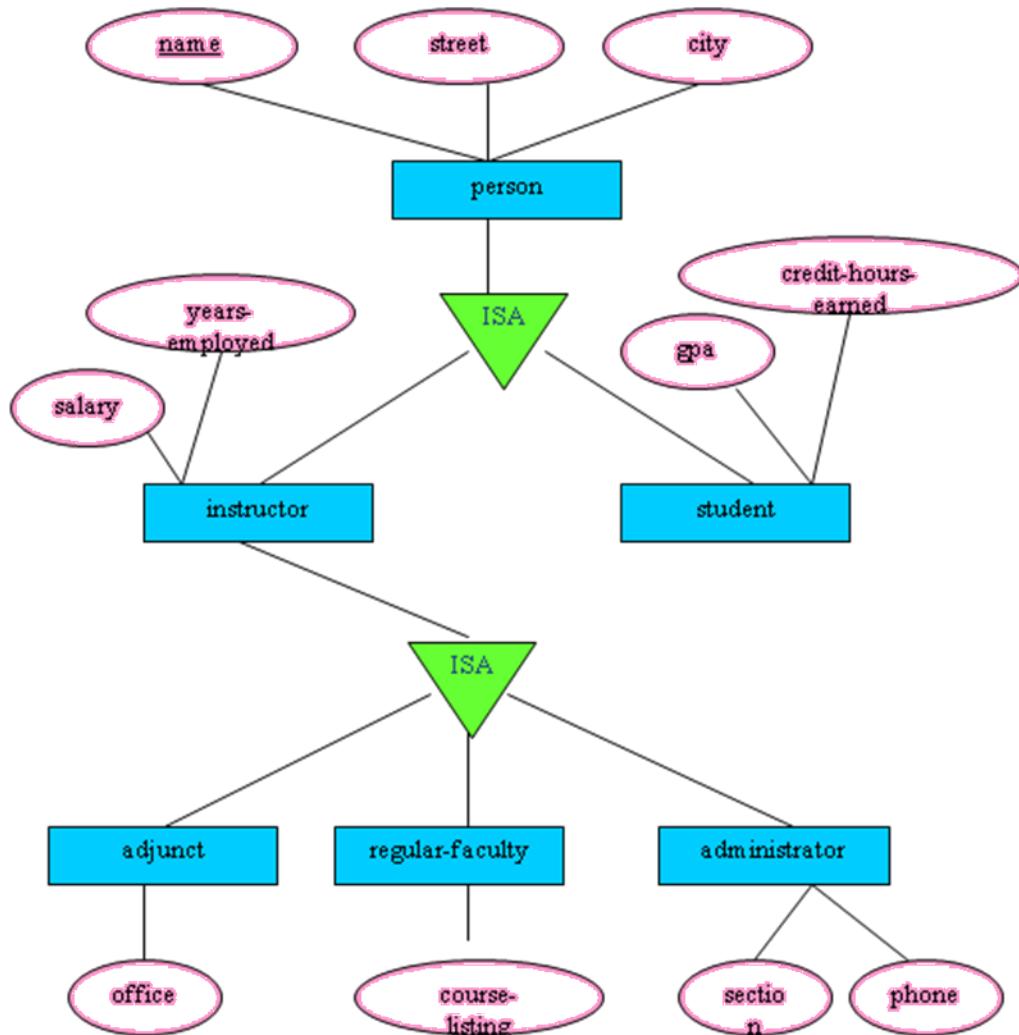
Mô hình thực thể liên kết mở rộng

Một số đặc tính của tình huống trong thế giới thực có thể khó mô hình hóa nếu chỉ sử dụng những đặc tính của mô hình E-R đã được xem xét từ trước đến nay. Một số khái niệm chung đòi hỏi phải mở rộng mô hình E-R để thực hiện cơ chế mô hình hóa cho các đặc tính này. Một lần nữa, chúng ta sẽ không bàn luận đến tất cả chúng mà chỉ xem xét mức tổng quan về một số các đặc tính mở rộng quan trọng hơn trong chúng.

Cụ thể hóa (Specializations)

Một tập thực thể bao gồm các phân nhóm thực thể mà được phân biệt với các thực thể khác theo một cách nào đó. Cụ thể hơn, một tập con các thực thể trong một tập thực thể có thể có các thuộc tính mà không chung với các thực thể khác trong tập thực thể đó. Một ví dụ minh họa, xem xét một tập thực thể *person* (người) với các thuộc tính *name*, *street*, và *city*. Một *person* có thể được phân chia nhỏ hơn thành một trong những loại sau: *student* (sinh viên) hoặc *instructor* (giảng viên). Mỗi loại người này được mô tả bởi một tập các thuộc tính bao gồm tất cả các thuộc tính của tập thực thể *person*, cộng với một vài các thuộc tính bổ sung thêm vào nữa. Thực thể *student* có thể mô tả thêm bởi thuộc tính *gpa* (*điểm trung bình*), và *credit-hours-earned* (*số lượng tín chỉ đã tích lũy*) trong khi đó thực thể *instructor* không được thể hiện bằng các thuộc tính này mà bằng một tập thuộc tính khác như *salary* (*lương*) và *years-employed* (*số năm làm việc*).

Quá trình thiết kế các phân nhóm trong một tập thực thể được gọi là quá trình cụ thể hóa. Việc cụ thể hóa của thực thể *person* cho phép chúng ta phân biệt giữa các loại người theo tiêu chí họ là sinh viên hay là giảng viên. Việc cụ thể hóa này có thể được lặp đi lặp lại nhiều lần vì vậy có thể có cụ thể hóa trong một quá trình cụ thể hóa khác. Trong phạm vi một lược đồ E-R, việc cụ thể hóa được biểu diễn bởi một hình tam giác với nhãn là ISA (là một), là một cách viết tắt của quan hệ “là một” (“is-a”) giữa nhóm con và nhóm cha. Mỗi quan hệ ISA được mô tả trong lược đồ như trong hình vẽ sau đây



Tổng quát hóa

Việc phát triển từ một tập thực thể ban đầu tới các mức tiếp theo là các phân nhóm các thực thể thể hiện các tiếp cận thiết kế từ trên-xuống trong đó các sự khác biệt được thể hiện tường minh.

Quá trình thiết kế cùng một bài toán như vậy có thể được tiến hành theo cách tiếp cận từ dưới lên, trong đó nhiều tập thực thể được đồng bộ vào một thực thể ở mức cao hơn trên cơ sở các thuộc tính chung. Nói một cách khác, đầu tiên chúng ta có thể xác định các thực thể *students(name, address, city, gpa, credit-hours-earned)* và một tập thực thể *instructors(name, address, city, salary, years-employed)*. Việc tìm điểm chung giữa các thuộc tính được gọi là **tổng quát hóa**, nó là một quan hệ tồn tại giữa một tập thực thể ở mức cao hơn với một hoặc nhiều tập thực thể ở mức thấp hơn.

Trong ví dụ chúng ta đang xét, *person* là tập thực thể ở mức cao và *instructor* và *student* là các tập thực thể ở mức thấp. Thực thể ở mức cao biểu diễn một lớp cha và thực thể ở mức thấp biểu diễn một lớp con, Vì vậy, *person* là một lớp cha của lớp con *instructor* và *student*.

Cho tất cả các mục đích thực tế, việc tổng quát hóa chỉ là một khái niệm ngược với việc cụ thể hóa và cả hai quá trình này đều có thể được dùng (hầu như là có thể hoán đổi cho nhau được) trong việc thiết kế các lược đồ cho một kịch bản có thực. Lưu ý trong lược đồ E-R, không thấy có sự khác biệt giữa tổng quát hóa và cụ thể hóa mà chỉ khác nhau ở cách bạn xem hình vẽ đó từ trên xuống hay từ dưới lên.

Sự khác nhau của hai cách tiếp cận này thường được thể hiện bởi những điểm bắt đầu và mục đích chung của chúng: Cụ thể hóa xuất phát từ một tập thực thể riêng biệt; nó nhấn mạnh sự khác nhau giữa các thực thể trong cùng một tập bằng cách tạo ra các tập thực thể phân biệt nhau ở mức thấp hơn. Những tập thực thể mức thấp này có thể có các thuộc tính hoặc tham gia vào các mối quan hệ mà không áp dụng đối với tất cả các thực thể trong tập thực thể ở mức cao hơn.

Trên thực tế, lý do mà một người thiết kế có thể cần để sử dụng quá trình cụ thể hóa là để thể hiện những đặc tính riêng của kịch bản trong thế giới thực. Ví dụ, nếu *instructor* và *student* không có các thuộc tính mà thực thể *person* có hoặc không tham gia vào các quan hệ mà *person* tham gia thì sẽ không cần phải thực hiện việc cụ thể hóa.

Tổng quát hóa xuất phát từ việc nhận ra rằng một số các tập thực thể có chung một số các đặc tính (về tên gọi, chúng được mô tả bởi cùng các thuộc tính và tham gia vào cùng các tập mối quan hệ). Trên cơ sở tính chung, tổng quát hóa tổng hợp những tập thực thể này thành một tập thực thể ở mức cao hơn. Quá trình tổng quát hóa được sử dụng để nhấn mạnh tính tương đồng giữa các tập thực thể ở mức thấp hơn và dấu đi những sự khác nhau. Nó cũng cho phép một sự thể hiện kinh tế trong đó các thuộc tính chung không bị lặp lại.

Sự kế thừa thuộc tính

Một tính chất không thể thiếu được của các thực thể mức thấp hơn, những thực thể được tạo ra bởi sự cụ thể hóa hoặc tổng quát hóa, là sự kế thừa thuộc tính. Thuộc tính của các tập thực thể ở mức cao hơn được nói là kế thừa bởi các tập thực thể ở mức thấp hơn. Trong ví dụ đã xét ở trên, *instructor* và *student* đều kế thừa tất cả các thuộc tính của *person*.

Một tập thực thể mức thấp hơn cũng kế thừa sự tham gia trong các tập mối quan hệ trong đó các tập thực thể ở mức cao hơn tham gia. Nó cũng kế thừa tất cả các thuộc tính và các mối quan hệ thuộc về tập thực thể mức cao hơn định nghĩa nó.

Các tập thực thể ở mức cao sẽ không kế thừa bất kỳ một thuộc tính nào hoặc mối quan hệ napf được định nghĩa trong tập thực thể ở mức thấp hơn. Thông thường, một phân cấp các tập thực thể sẽ được phát triển trong đó thực thể ở mức cao nhất xuất hiện trên đỉnh của phân cấp.

Nếu, trong một phân cấp như vậy, một tập thực thể nào đó có thể liên quan tới vai trò một thực thể ở mức thấp hơn trong một mối quan hệ ISA thì sự kế thừa được gọi là kế thừa

đơn. Mặt khác, nếu một thực thể nào đó liên quan như một tập thực thể mức thấp trong nhiều hơn một mối quan hệ ISA thì sự kế thừa đó được coi là đa kế thừa.

Các ràng buộc trên việc tổng quát hóa

Để mô hình hóa các tình huống trong thực tế chính xác hơn, người thiết kế dữ liệu có thể thiết lập các ràng buộc trên một tổng quát hóa (hoặc cụ thể hóa). Loại đầu tiên của ràng buộc liên quan tới việc xác định các thực thể nào có thể là thành viên của tập thực thể mức thấp hơn. Thành viên này có thể được định nghĩa theo một trong hai cách sau đây:

Mệnh đề xác định: Trong loại này, thành viên được đánh giá trên cơ sở xác định xem một thực thể có thỏa mãn một mệnh đề (điều kiện) tường minh nào đó không. Ví dụ, giả sử rằng tập thực thể ở mức cao *account* có thuộc tính *account-type*. Tất cả các thuộc tính được đánh giá dựa trên việc định nghĩa thuộc tính *account-type*. Chỉ những thực thể thỏa mãn mệnh đề *account-type* = “*savings account*” sẽ được phép thuộc vào tập thực thể mức thấp hơn *savings-account*. Vì tất cả các thực thể mức thấp hơn được đánh giá dựa trên cùng thuộc tính, loại tổng quát hóa này được gọi là **thuộc tính xác định**.

Người dùng xác định: Tập các thực thể mức thấp do người dùng xác định không bị ràng buộc bởi các điều kiện thành viên mà người dùng cơ sở dữ liệu gán các thực thể tới các tập thực thể nào đó. Ví dụ: giả sử sau 3 tháng làm việc tại một ngân hàng, một nhân viên được gán tới một trong năm nhóm khác nhau. Các đội sẽ được biểu diễn như năm tập thực thể mức thấp hơn của tập thực thể mức cao hơn là *Employee*. Một nhân viên nào đó sẽ không được gán tới một nhóm làm việc cụ thể nào đó một cách tự động trên cơ sở một việc định nghĩa một điều kiện tường minh. Thay vào đó, người sử dụng có trách nhiệm tạo ra các nhóm dựa trên từng cá nhân các thành viên, có thể là tùy ý.

Một loại ràng buộc tổng quát hóa thứ hai liên quan tới việc liệu các thực thể có thuộc vào nhiều hơn một tập thực thể ở mức thấp hơn không trong một quá trình tổng quát hóa. Các tập thực thể ở mức thấp hơn có thể là một trong các loại sau:

Không giao nhau: Một ràng buộc không giao nhau đòi hỏi rằng một thực thể không thể thuộc vào nhiều hơn một tập thực thể ở mức thấp hơn. Trong ví dụ trên, một thực thể *account* có thể thỏa mãn một điều kiện cho thuộc tính *account-type* tại một thời điểm. Ví dụ, một *account-type* có thể hoặc là *checking account* (tài khoản thường dùng) hoặc một *saving account* (tài khoản tiết kiệm) nhưng không thể là cả hai được.

Giao nhau: Trong tổng quát hóa có giao nhau, một thực thể có thể thuộc vào nhiều hơn một tập thực thể ở mức thấp hơn trong cùng một quá trình tổng quát hóa. Lấy ví dụ, xem xét nhóm làm việc trong ngân hàng từ phần trước. Giả sử rằng các quản lý nào đó có thể tham gia vào nhiều hơn một nhóm làm việc. Một nhân viên nào đó (hay một quản lý nào đó) vì thế có thể xuất hiện trong nhiều hơn một tập thực thể nhóm mà là tập thực thể ở mức thấp hơn của *employee*. Chú ý là việc giao nhau của thực thể ở mức thấp là trường hợp thông thường hay nhắc đến nếu như không chú thích gì thêm, một ràng buộc không giao nhau phải

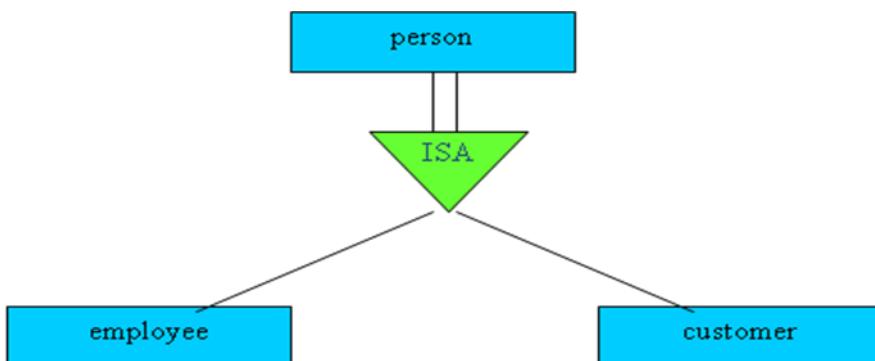
được thể hiện một cách tường minh trong quá trình tổng quát hóa (hay cụ thể hóa). Trong mô hình E-R một ràng buộc không giao nhau được thể hiện bằng từ “disjoint” đặt cạnh biểu tượng hình tam giác như trong hình vẽ minh họa dưới đây. Ý nghĩa của lược đồ bây giờ đã rõ ràng: các nhân viên và các khách hàng là các phân nhóm cụ thể hóa của tập người nói chung và ràng buộc không giao nhau ý rằng một nhân viên cũng không phải là một khách hàng. Nếu ràng buộc không giao nhau này được bỏ đi thì một nhân viên có thể là một khách hàng (hoặc nhìn từ một hướng khác, một người trong công ty này có thể vừa là khách hàng vừa là một nhân viên).

Ràng buộc dựa trên tính toàn bộ: đây là lọa ràng buộc cuối cùng của quá trình tổng quát hóa hoặc cụ thể hóa, xác định liệu một thực thể của tập thực thể ở mức cao phải thuộc vào ít nhất một tập trong các tập thực thể ở mức thấp hơn hay không. Loại ràng buộc này có thể được rơi vào một trong hai dạng dưới đây:

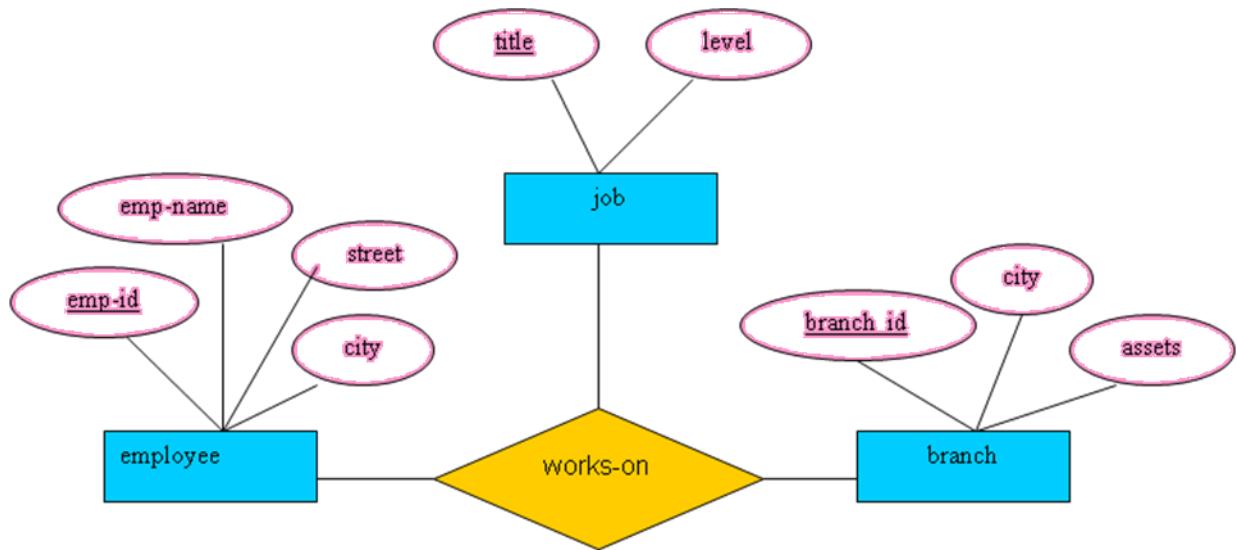
Tổng quát hóa/cụ thể hóa toàn bộ: mỗi thực thể ở mức cai phải thuộc vào một thực thể ở mức thấp hơn.

Tổng quát hóa/cụ thể hóa một phần: Một vài thực thể ở mức cao có thể không thuộc vào một tập thực thể nào mức thấp hơn nào. Tổng quát hóa một phần là trường hợp ngầm định nếu như không có phát biểu gì về loại ràng buộc này trong lược đồ. (Nhắc lại là sự tham gia toàn bộ được thể hiện trong lược đồ E-R bằng một đường kết nối kép-vì vậy nó cũng sẽ được sử dụng để thể hiện sự tổng quát hóa toàn bộ. Trong ví dụ dưới đây tổng quát hóa là toàn bộ và có giao nhau có nghĩa là mỗi người trong công ty phải xuất hiện như là một nhân viên hoặc là một khách hàng và cũng có thể là cả hai.

Ví dụ với các ràng buộc này thể hiện trong hình vẽ dưới đây

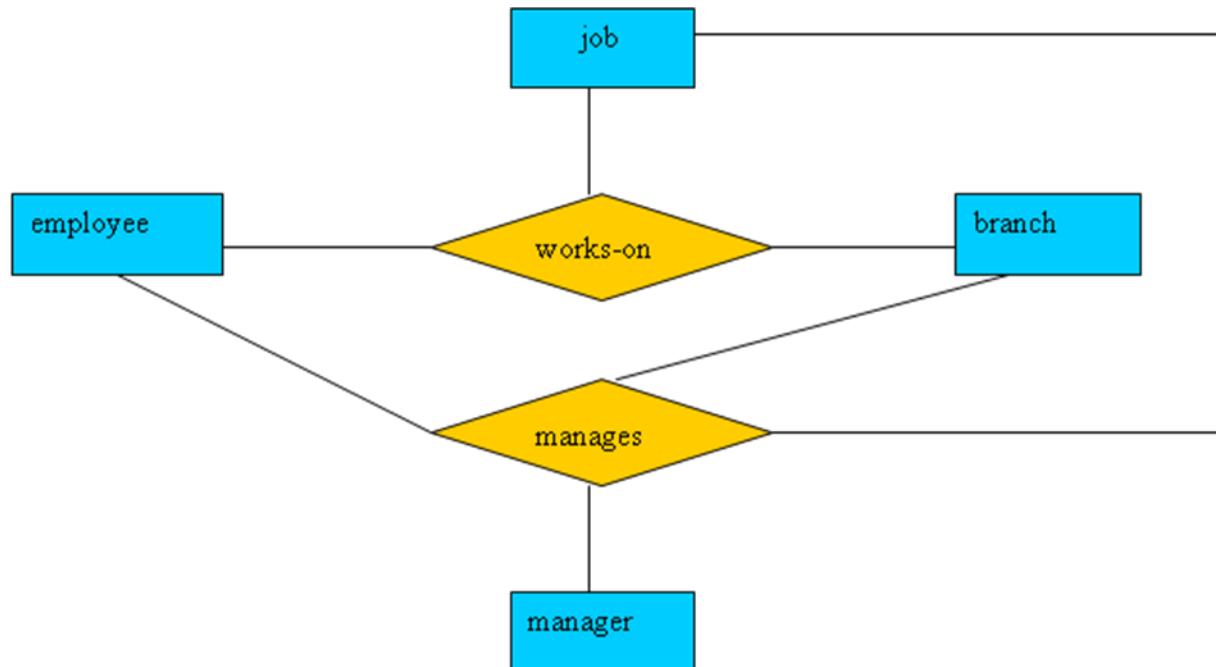


Tích hợp: Một trong những hạn chế của mô hình E-R là nó không thể biểu diễn các mối quan hệ trong các mối quan hệ. Để hiểu được tại sao việc này lại quan trọng chúng ta xem xét mối quan hệ ba ngôi *works-on* giữa *employee* (nhân viên), *branch* (chi nhánh ngân hàng-văn phòng) và *job* (công việc) được thể hiện trong lược đồ E-R dưới đây



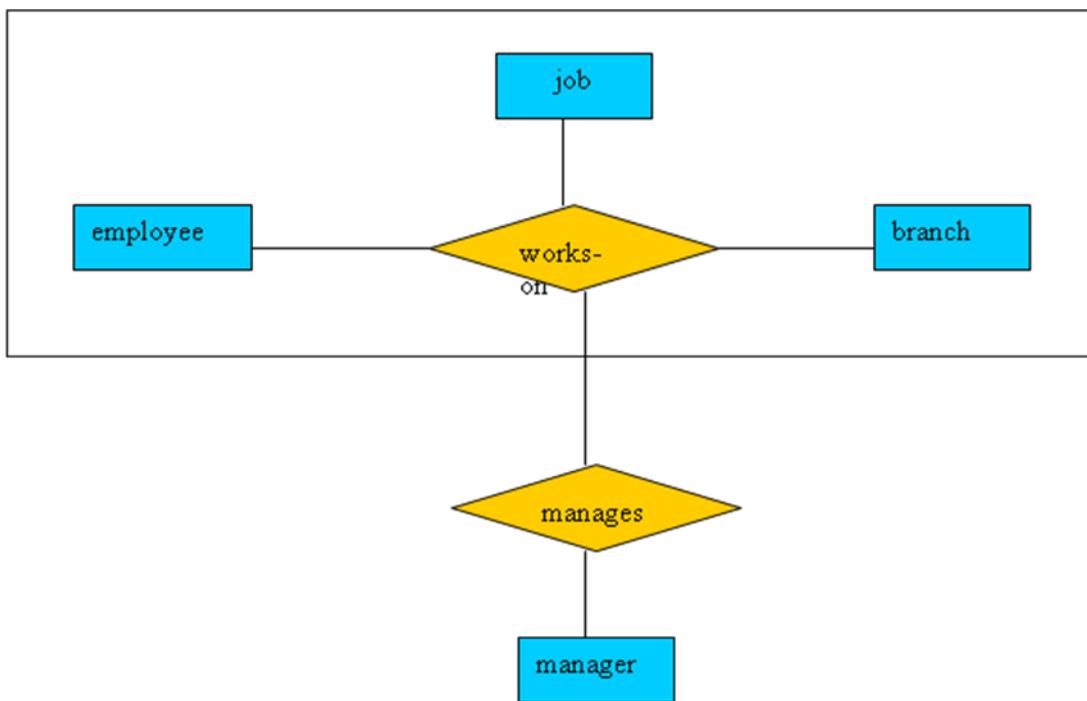
Với ngữ cảnh này, giả sử rằng chúng ta muốn ghi nhận các quản lý cho mỗi nhiệm vụ được thực hiện bởi một nhân viên tại một văn phòng chi nhánh nào đó; có nghĩa là chúng ta muốn lưu trữ các người quản lý cho quan hệ ba ngôi (*employee, branch, job*). Giả thiết rằng chỉ có một tập thực thể *manager*. Một cách để thực hiện việc này là tạo ra một mối quan hệ bốn ngôi như hình vẽ sau đây.

Câu hỏi đặt ra là tại sao không thể hiện mối quan hệ hai ngôi giữa *manager* và *employee*? Câu trả lời như sau: một mối quan hệ sẽ không cho phép chúng ta thể hiện (*branch, job*) của một *employee* (nhân viên) được quản lý bởi người quản lý nào.



Khi bạn nhìn vào lược đồ E-R mô hình hóa tình huống này, dường như tập các mối quan hệ *works-on* và *manages* có thể kết hợp thành một tập mối quan hệ duy nhất. Tuy nhiên, chúng ta không thể thực hiện điều đó bởi vì một bộ ba *employee*, *branch*, *job* có thể không có một người quản lý. Tuy nhiên, rõ ràng là có một thông tin dư thừa trong hình vẽ trên vì mỗi bộ ba *employee*, *branch*, *job* của mối quan hệ *manages* cũng giống của *works-on*. Nếu người quản lý là một giá trị chứ không phải là một thực thể, ta có thể biến *manager* thành một thuộc tính đa trị của mối quan hệ *works-on*. Tuy nhiên, việc biến đổi đó sẽ làm việc tìm kiếm khó khăn hơn (cả về logic lẫn về chi phí thực hiện), lấy ví dụ, bộ ba *employee*-*branch*-*job* sẽ do người quản lý nào chịu trách nhiệm. Tuy nhiên, sự lựa chọn này không sẵn có trong trường hợp người quản lý là một thực thể *manager*.

Cách tốt nhất để mô hình loại tình huống này là sử dụng **tích hợp (hay kết hợp)**. Kết hợp là một sự trừu tượng thông qua việc các mối quan hệ được coi như các thực thể ở mức cao. Vì vậy, trong ví dụ trên, chúng ta có thể coi tập quan hệ *works-on* (liên quan tới các tập thực thể *employee*, *branch*, và *job*) như một tập thực thể ở mức cao được gọi tên là *works-on*. Một tập thực thể như vậy được đổi xử theo cách thức giống như các tập thực thể khác. Ta có thể tạo một quan hệ hai ngôi *manages* giữa *works-on* và *manager* để thể hiện ai quản lý các nhiệm vụ này. Lược đồ E-R dưới đây sẽ thể hiện một sự tích hợp trong mô hình E-R.

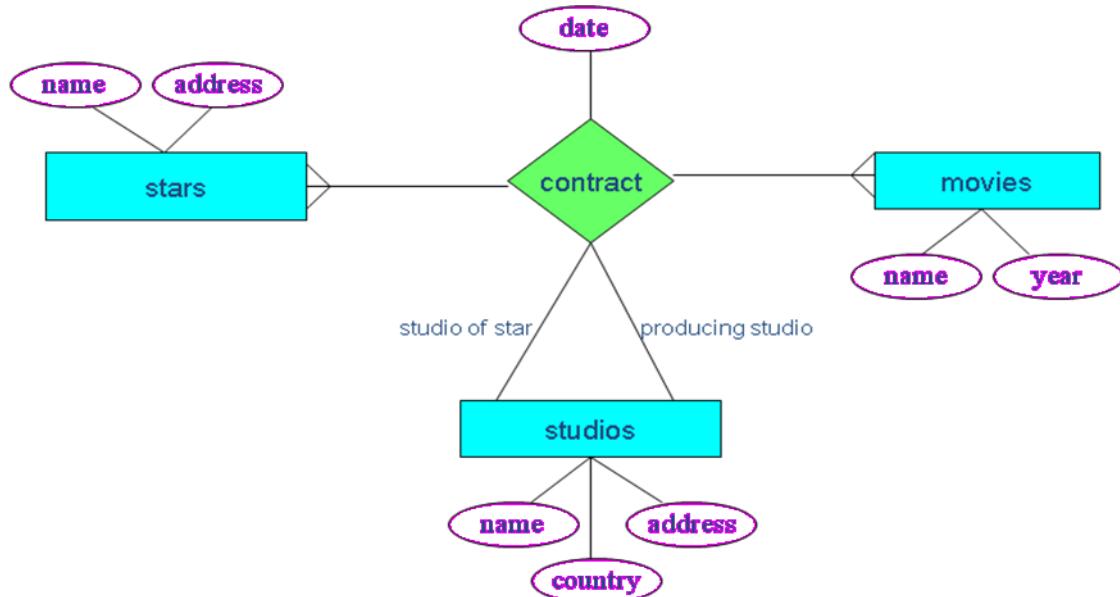


Các mối quan hệ nhiều ngôi.

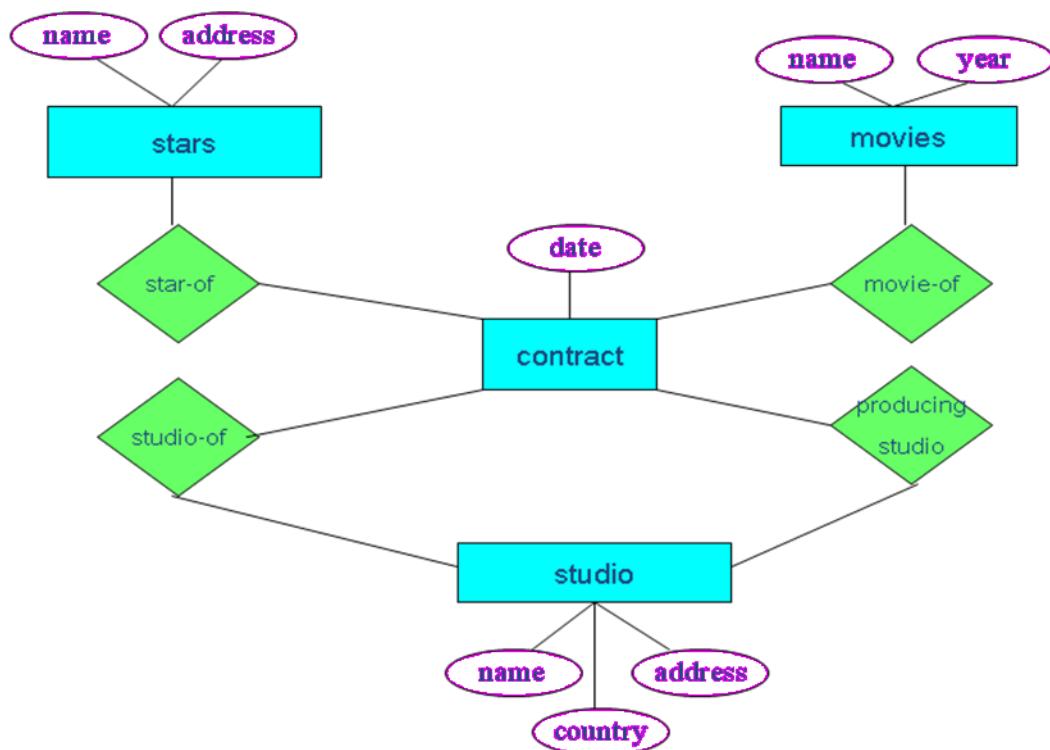
Hầu hết các mối quan hệ chúng ta đã xem xét từ trước đến giờ đều là các quan hệ hai ngôi, là những quan hệ liên quan tới hai tập thực thể. Mọi mối quan hệ liên quan tới nhiều hơn hai tập thực thể có thể được chuyển đổi tới một tập các mối quan hệ hai ngôi, $n_{\text{ngôi}} - 1$. Điều này

là rất có ích trong khi mô hình E-R không hạn chế các quan hệ tới quan hệ hai ngôi nhưng các mô hình dữ liệu khác ví dụ như ngôn ngữ định nghĩa đối tượng có hạn chế này.

Để mô tả cho sự chuyển đổi từ mối quan hệ nhiều ngôi sang một tập các mối quan hệ hai ngôi, xem xét một lược đồ E-R ví dụ sau đây

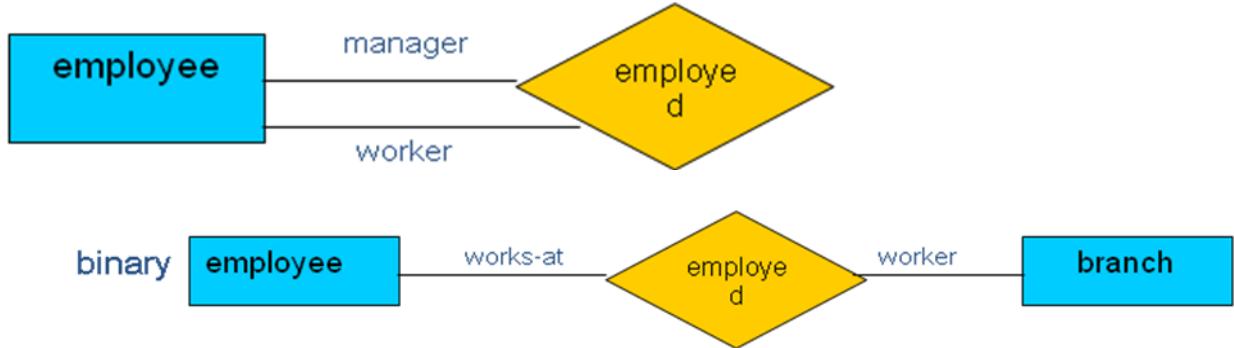


Được chuyển thành một tập các quan hệ hai ngôi như hình vẽ dưới đây



Lược đồ E-R với các chỉ thị vai trò

Các vai trò trong một lược đồ E-R được biểu diễn bằng việc gán nhãn lên các đường kết nối giữa các tập thực thể với các tập các mối quan hệ. Các vai trò này có thể được xác định cho các mối quan hệ đệ quy, nhị phân, và không nhị phân.

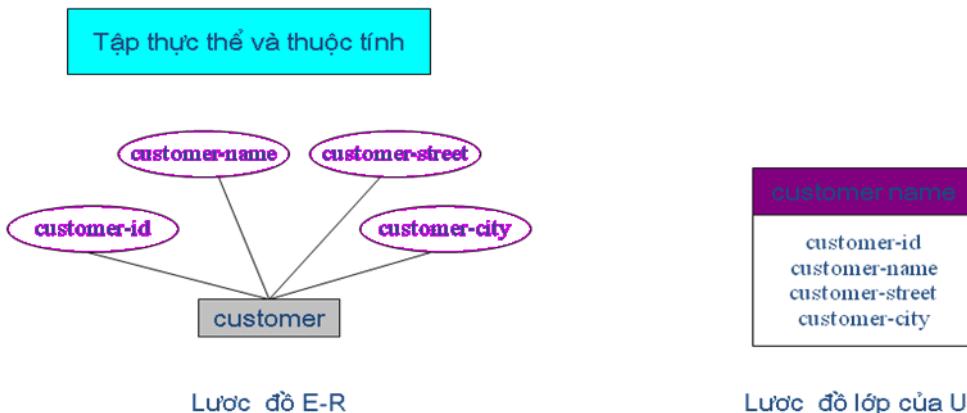


Giới thiệu về ngôn ngữ tạo mô hình thống nhất UML

Một số các thành phần của UML là

- 1- Lược đồ lớp (class): một lược đồ lớp tương tự như một lược đồ E-R
- 2- Lược đồ use case: được sử dụng để thể hiện sự tương tác giữa người sử dụng và hệ thống, cụ thể là các bước của công việc mà người dùng cần thực hiện (ví dụ như việc rút tiền từ một tài khoản ngân hàng hoặc đăng ký một khóa học)
- 3- Lược đồ hoạt động (activity): thể hiện luồng công việc giữa các thành phần của một hệ thống
- 4- Lược đồ cài đặt (implementation) thể hiện các thành phần của hệ thống và sự kết nối giữa chúng ở cả mức độ phần mềm và phần cứng

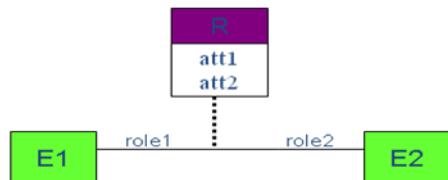
Sự liên quan giữa lược đồ lớp của UML và lược đồ E-R được thể hiện trong các hình vẽ dưới đây



Các mối quan hệ



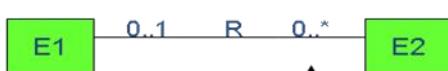
att1 *att2*



Lược đồ E-R

Lược đồ lớp của UML

Các ràng buộc về số lượng tham gia

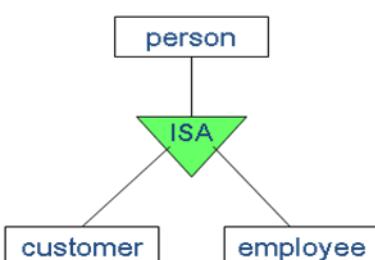


Lưu ý: Vị trí của các ràng buộc này là ngược nhau giữa hai mô hình. Trong UML ràng buộc 0..1 ở bên trái có nghĩa là thực thể E2 có thể tham gia nhiều nhất một mối quan hệ trong khi mỗi thực thể E1 có thể tham gia tới nhiều mối quan hệ, nói một cách khác, mối quan hệ này là nhiều 1 từ E2 tới E1

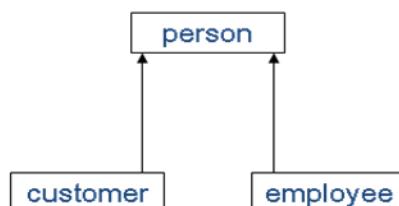
Lược đồ E-R

Lược đồ UML

Khái quát hóa và Cụ thể hóa



Khái quát hóa giao nhau



Lược đồ E-R

Lược đồ lớp của UML

Ràng buộc toàn vẹn tham chiếu

Các ràng buộc tham chiếu có thể được hiểu đơn giản là việc đảm bảo rằng một thuộc tính nào đó có một giá trị khác rỗng. Tuy nhiên, các ràng buộc toàn vẹn tham chiếu thường liên quan tới các mối quan hệ giữa các tập thực thể. Chúng ta cùng xem xét ví dụ về ngân hàng và một quan hệ nhiều-một giữa khách hàng và tài khoản được biểu diễn như hình vẽ dưới đây.

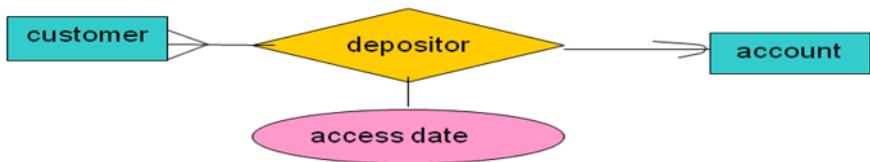


Quan hệ *depositor* này thể hiện một thực tế là không có tài khoản nào có thể được chuyển tiền vào bởi nhiều hơn một khách hàng (và một khách hàng cũng chỉ có thể chuyển tiền vào nhiều tài khoản khác nhau). Quan trọng hơn, quan hệ này không phát biểu rằng một tài khoản phải được chuyển tiền vào bởi một khách hàng cũng như không phát biểu rằng một khách hàng phải thực hiện một việc chuyển tiền vào một tài khoản. Hơn nữa, nó cũng không thể hiện rằng nếu một tài khoản được chuyển tiền vào bởi một khách hàng thì một khách hàng sẽ được lưu trữ trong cơ sở dữ liệu.

Một ràng buộc toàn vẹn tham chiếu yêu cầu rằng mỗi thực thể “được tham chiếu tới” bởi một mối quan hệ phải tồn tại trong cơ sở dữ liệu. Có một số phương pháp được sử dụng để đảm bảo tính ràng buộc toàn vẹn tham chiếu:

1. Việc xóa bỏ một thực thể được tham chiếu đến là không được phép. Nói một cách khác, nếu Kristi thực hiện một lần chuyển tiền vào tài khoản số hiệu 456 thì hệ quả là chúng ta sẽ không thể xóa thông tin liên quan tới Kristi hoặc của tài khoản 456.
 2. Nếu một thực thể được tham chiếu được xóa bỏ thì tất cả các bản ghi tham chiếu tới thực thể bị xóa bỏ đó cũng bị xóa. Nói một cách khác, nếu chúng ta xóa thông tin về Kristi thì chúng ta phải xóa tất cả các thông tin về tài khoản của những tài khoản mà cô ta (một mình) đã chuyển tiền vào đó. Lưu ý là trong một ví dụ cụ thể mà chúng ta đang xem xét quan hệ là nhiều-một có nghĩa là nếu Kristi đã chuyển tiền vào một tài khoản thì cô ta sẽ là khách hàng duy nhất làm việc đó. Trường hợp này không giống với quan hệ nhiều-nhiều.

Các ràng buộc toàn vẹn tham chiếu có thể được mô hình trong E-R. Thông thường chúng được thể hiện bằng một mũi tên đầu cong như trong hình vẽ dưới đây. Mũi tên cong thể hiện tồn tại một ràng buộc trên tập thực thể *account* thông qua mối quan hệ của nó *depositor* với tập thực thể *customers*



Bài 6: Mô hình dữ liệu quan hệ

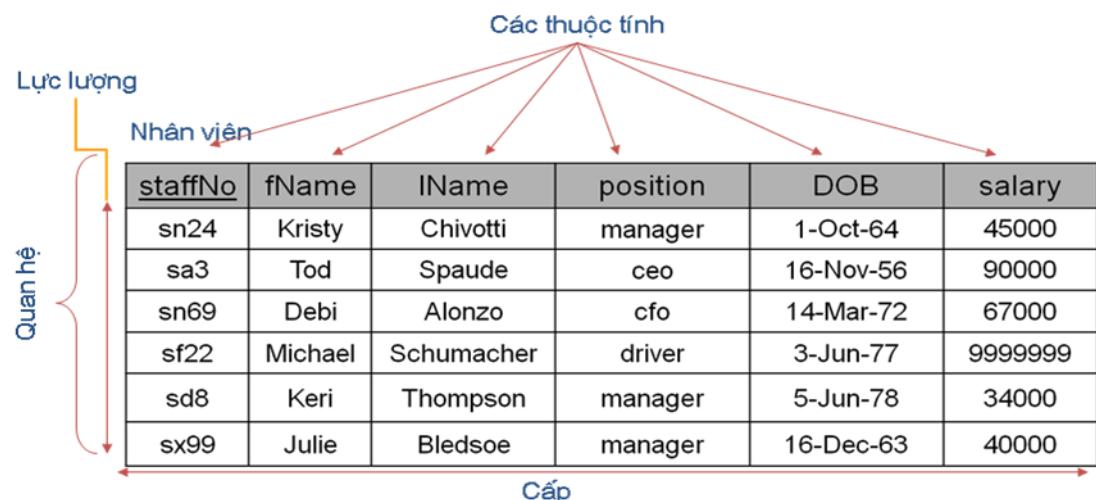
Mô hình dữ liệu quan hệ

Mô hình dữ liệu quan hệ được phát triển dựa trên khái niệm về quan hệ toán học. Nhà khoa học đề xuất ra mô hình quan hệ này, có tên là Codd, là một nhà toán học có sự phạm và ông đã sử dụng một thuật ngữ từ lĩnh vực này, liên quan chủ yếu đến lý thuyết tập hợp và logic mệnh đề để làm tên của mô hình này.

Mô hình dữ liệu quan hệ bao gồm những khái niệm cơ bản sau:

- Quan hệ: một quan hệ là một bảng (ma trận) với các hàng và các cột. Các quan hệ sẽ lưu giữ thông tin về các đối tượng được mô hình hóa trong một cơ sở dữ liệu.
- Thuộc tính: một thuộc tính là một cột được đặt tên của một quan hệ. Một thuộc tính là một đặc tính nào đó của một thực thể (hay một mối quan hệ) được mô hình hóa trong cơ sở dữ liệu. Các thuộc tính có thể xuất hiện theo bất kỳ trật tự nào trong một quan hệ.
- Miền giá trị: Một miền giá trị là một tập các giá trị có thể của một hoặc nhiều thuộc tính. Mỗi thuộc tính được xác định trên một miền giá trị nào đó. Các miền giá trị có thể khác nhau cho mỗi thuộc tính hoặc hai hay nhiều thuộc tính có thể được xác định trên cùng một miền giá trị.
- Bộ: Một bộ là một hàng của một quan hệ. Các bộ có thể xuất hiện theo bất kỳ trật tự nào trong một quan hệ và quan hệ sẽ vẫn giống nhau vì vậy thể hiện cùng ý nghĩa.
- Cấp: của một quan hệ là số lượng các thuộc tính mà nó có
- Lực lượng: Lực lượng của một quan hệ là số lượng các bộ mà nó có.
- Cơ sở dữ liệu quan hệ: Một tập hợp các quan hệ được chuẩn hóa với các tên phân biệt nhau.

Một ví dụ về quan hệ thể hiện ở hình vẽ dưới đây:



Các định nghĩa miền giá trị cho ví dụ trên

Thuộc tính	Tên miền	Ý nghĩa	Định nghĩa miền
staffNo	Staffnumbers	Tập của tất cả các số hiệu có thể của nhân viên	Ký tự: kích cỡ 4, phải bắt đầu bằng chữ cái.
fName, lName	Name	Tập tất cả các tên có thể của một người	Ký tự: kích cỡ 20
DOB	Date	Ngày sinh của một người	Ngày: trong khoảng từ 01-Jan-20, Định dạng: dd-mmm-yy
salary	Salaries	Các giá trị có thể của lương nhân viên	Dạng tiền tệ: 7 chữ số, trong khoảng 10,000-9,999,999
position	Alljobs	Tập tất cả các vị trí có thể có của nhân viên trong một công ty	Lựa chọn một trong tập: {ceo, cfo, coo, manager, asst. manager, driver, secretary}

Các thuật ngữ tương đương có thể thay thế được lẫn nhau trong mô hình quan hệ

Thuật ngữ chính thống	Lựa chọn 1	Lựa chọn 2

Quan hệ	Bảng	Tập
Bộ	Hàng	Bản ghi
Thuộc tính	Cột	Trường

Quan hệ là gì?

Để hiểu được ý nghĩa thực sự của khái niệm quan hệ, chúng ta cần nhắc lại một số khái niệm toán học cơ bản.

- Cho hai tập D_1 và D_2 với $D_1 = \{2, 4\}$ và $D_2 = \{1, 3, 5\}$, tích Δ các của hai tập này được kí hiệu là $D_1 \times D_2$, là tập của tất cả các cặp có thứ tự với phần tử đầu tiên là một phần tử của D_1 và phần tử thứ hai là của D_2

$$D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$$

- Mọi tập con của tích Δ các này là một quan hệ vì vậy chúng ta có thể sinh ra một quan hệ R sao cho $R = \{(2, 3), (4, 3)\}$
- Ngoài ra chúng ta có thể xác định một điều kiện nào đó để lựa chọn các phần tử từ $D_1 \times D_2$ cho quan hệ R ví dụ $R = \{(x, y) | x \in D_1, y \in D_2, \text{and } y = 3\}$
- Cho ba tập D_1 , D_2 , và D_3 với $D_1 = \{2, 4\}$, $D_2 = \{1, 3\}$, and $D_3 = \{3, 6\}$, tích Δ các của ba tập này được kí hiệu là $D_1 \times D_2 \times D_3$, là tập của tất cả các cặp có thứ tự với phần tử đầu tiên là một phần tử của D_1 và phần tử thứ hai là của D_2 , phần tử thứ ba là của D_3 .
 - $D_1 \times D_2 \times D_3 = \{(2, 1, 3), (2, 1, 6), (2, 3, 3), (2, 3, 6), (4, 1, 3), (4, 1, 6), (4, 3, 3), (4, 3, 6)\}$

Mọi tập con của tích Δ các này là một quan hệ.

- Một cách tổng quát, nếu D_1 , D_2 , ... D_n là n tập thì tích Δ các của chúng được định nghĩa là $D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) | d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n\}$ và nhìn chung được viết với ký phong $\prod D_i$
- Một lược đồ quan hệ là một quan hệ có tên được xác định bởi một tập các thuộc tính và các cặp tên miền $R_i = \{A_1:d_1, A_2:d_2, \dots, A_n:d_n | d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n\}$
- Một lược đồ cơ sở dữ liệu quan hệ là một tập các lược đồ quan hệ, mỗi lược đồ có một tên riêng $R = \{R_1, R_2, \dots, R_n\}$

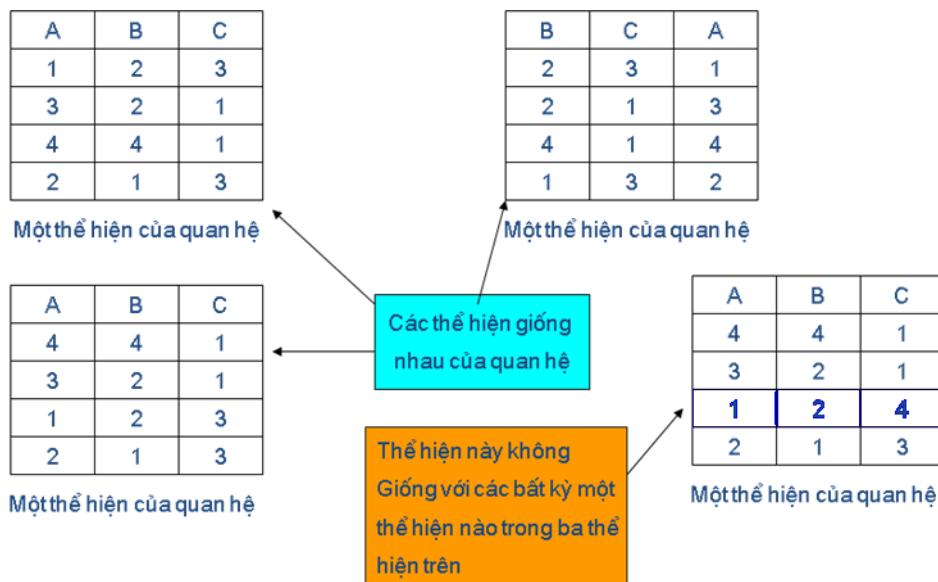
Một quan hệ có các đặc tính sau:

1. Quan hệ có một định danh hay tên phân biệt với tên của các quan hệ khác trong lược đồ quan hệ.
2. Mỗi thuộc tính chứa một giá trị nguyên tố (không phân chia được nữa).
3. Mỗi thuộc tính có một tên riêng biệt.
4. Các giá trị của một thuộc tính đều xuất phát từ một miền giá trị.
5. Mỗi bộ là phân biệt nhau; không có hai bộ nào giống hệt nhau.
6. Trật tự của các thuộc tính không quan trọng.
7. Trật tự của các bộ cũng không quan trọng trên lý thuyết. (Tuy nhiên, trong thực tế, trật tự này có thể ảnh hưởng đến hiệu quả truy nhập vào các bộ, Chi tiết sẽ xem xét trong những phần sau).

Các lược đồ quan hệ và Các thể hiện của quan hệ

Khái niệm về lược đồ quan hệ và thể hiện (instance) của quan hệ là khác nhau và cần được phân biệt. Lược đồ bao gồm tên và các thuộc tính cho quan hệ và khá bất biến. Còn một thể hiện của quan hệ là một tập các bộ của quan hệ đó và thể hiện này có thể thay đổi thường xuyên. Hầu hết các cập nhật và mỗi quá trình chèn thêm hay xóa các bộ sẽ làm thay đổi thể hiện của quan hệ đó. Một cơ sở dữ liệu hiện thời (snapshot database) thể hiện trạng thái hiện tại của thế giới thực được lưu trữ trong cơ sở dữ liệu đó. Tại một thời điểm bất kỳ, nó mô hình hóa thể hiện hiện tại của thế giới thực. Nếu thế giới thực thay đổi, cơ sở dữ liệu cũng sẽ thay đổi để duy trì cách biểu diễn cái thể hiện thế giới thực hiện tại đó.

Ví dụ về các quan hệ giống nhau và khác nhau thể hiện trong hình vẽ dưới đây:



Thiết kế logic

Trong quá trình thiết kế logic bạn chuyển đổi thiết kế mức khái niệm sang các lược đồ cơ sở dữ liệu quan hệ. Như vậy đầu vào cho quá trình này chính là lược đồ thực thể liên kết (E-R diagrams) và đầu ra của quá trình này là các lược đồ quan hệ.

Việc ánh xạ các lược đồ thực thể liên kết sang các quan hệ là một quá trình khá dễ dàng với một tập các luật được định nghĩa khá rõ. Trong thực tế nhiều công cụ CASE (công cụ trợ giúp cho công nghệ phần mềm) có thể thực hiện tự động nhiều bước của quá trình chuyển đổi. Tuy nhiên, việc bạn hiểu rõ các bước này thực sự quan trọng bởi ba lý do sau đây

- 1- Các công cụ CASE thường không thể mô hình hóa các quan hệ dữ liệu phức tạp ví dụ như các quan hệ ba ngôi và các quan hệ giữa các lớp cha/lớp con. Các bước này sẽ được thực hiện thủ công bằng tay.
- 2- Có rất nhiều lựa chọn hợp lý nên bạn phải lựa chọn một cách thủ công bằng tay
- 3- Bạn cần được chuẩn bị để thực hiện việc kiểm tra chất lượng đối với những kết quả thu được sau khi chạy các công cụ CASE.

Trong các bước bạn sẽ cần phải tuân thủ theo để ánh xạ các lược đồ thực thể liên kết sang các lược đồ quan hệ, rất có ích nếu bạn nhớ rằng chúng ta đã định nghĩa ba loại thực thể sau:

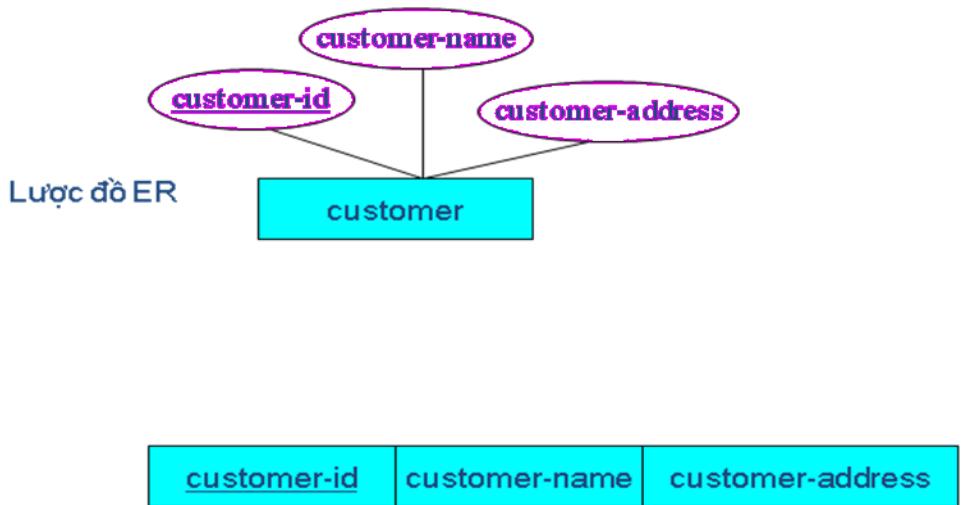
- Thực thể thông thường là các thực thể có thể tồn tại độc lập và thường thể hiện các đối tượng của thế giới thực ví dụ như con người hoặc sản phẩm. Nó được thể hiện trong mô hình thực thể liên kết bằng các hình chữ nhật với một đường viền đơn.
- Thực thể yếu là các thực thể không thể tồn tại một mình mà phải đi cùng với một mối quan hệ xác định bởi một loại thực thể xác định nó (thực thể chủ- khoe). Các thực thể yếu được biểu diễn bởi một hình chữ nhật với đường viền kép.
- Thực thể kết hợp (hay còn được gọi là các danh động từ) được hình thành từ những mối quan hệ nhiều-nhiều giữa các loại thực thể khác nhau. Các thực thể kết hợp này được biểu diễn bởi một hình chữ nhật với đường viền đơn và được bao quanh bởi một biểu tượng quan hệ hình thoi.

Việc ánh xạ mô hình thực thể liên kết sang mô hình quan hệ

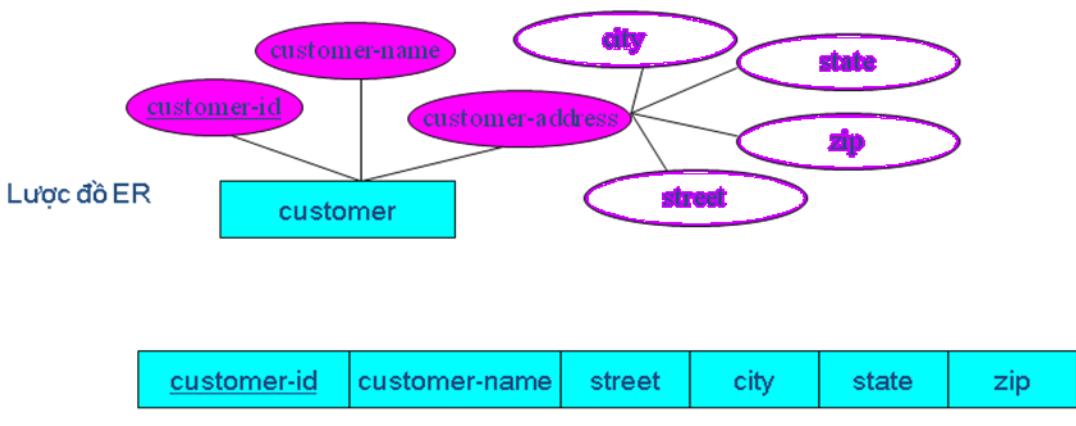
Bước 1: dùng cho việc ánh xạ các thực thể thông thường (thực thể khỏe)

- Mỗi thực thể thông thường trong mô hình thực thể liên kết sẽ được chuyển đổi thành một lược đồ quan hệ.
- Tên của quan hệ thường là tên của loại thực thể.
- Mỗi thuộc tính đơn của loại thực thể trở thành một thuộc tính của lược đồ quan hệ
- Thuộc tính định danh trở thành khóa chính của quan hệ tương ứng

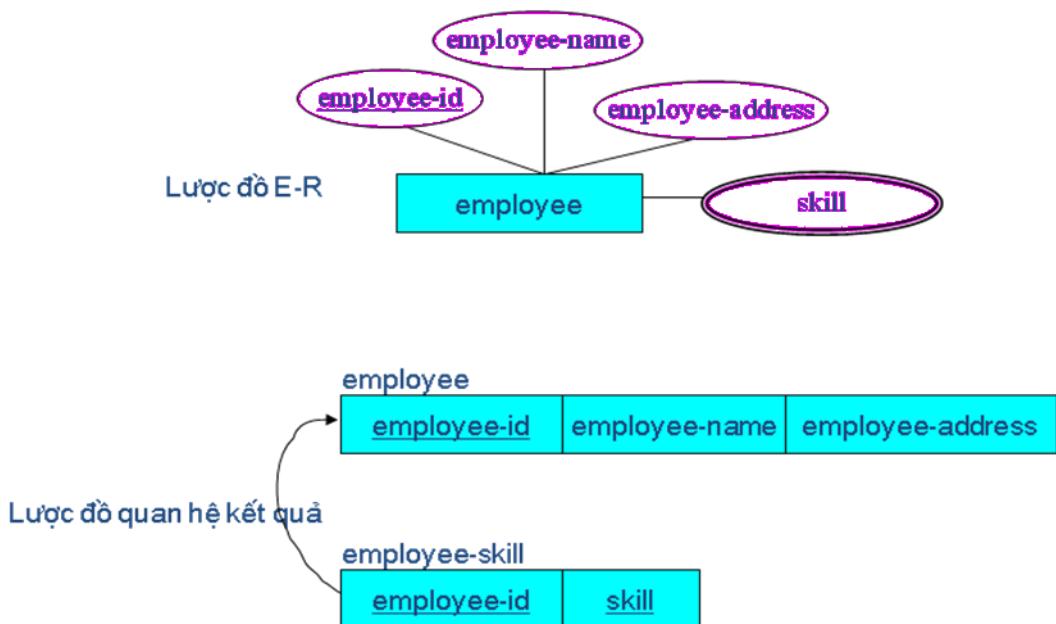
Ví dụ cho bước 1



Các thuộc tính ghép: khi một loại thực thể thường có một thuộc tính ghép, chỉ những thuộc tính đơn của thuộc tính ghép này được đưa vào lược đồ quan hệ mới. Ví dụ thể hiện trong hình dưới đây



Với trường hợp có thuộc tính đa trị: khi một thực thể thường có một thuộc tính đa trị, hai lược đồ quan hệ mới được tạo ra (không phải một cái). Lược đồ quan hệ đầu tiên chứa tất cả các thuộc tính của thực thể loại trừ thuộc tính đa trị. Lược đồ thứ hai sẽ bao gồm hai thuộc tính cấu thành khóa chính của lược đồ quan hệ thứ hai. Thuộc tính thứ nhất trong khóa chính này là khoá chính của lược đồ thứ nhất, cái trở thành khoá ngoại trong lược đồ thứ hai. Thuộc tính thứ hai là thuộc tính đa trị này. Tên của lược đồ thứ hai nên thể hiện nghĩa của thuộc tính đa trị. Ví dụ cho trường hợp thuộc tính đa trị được thể hiện trong hình vẽ dưới đây

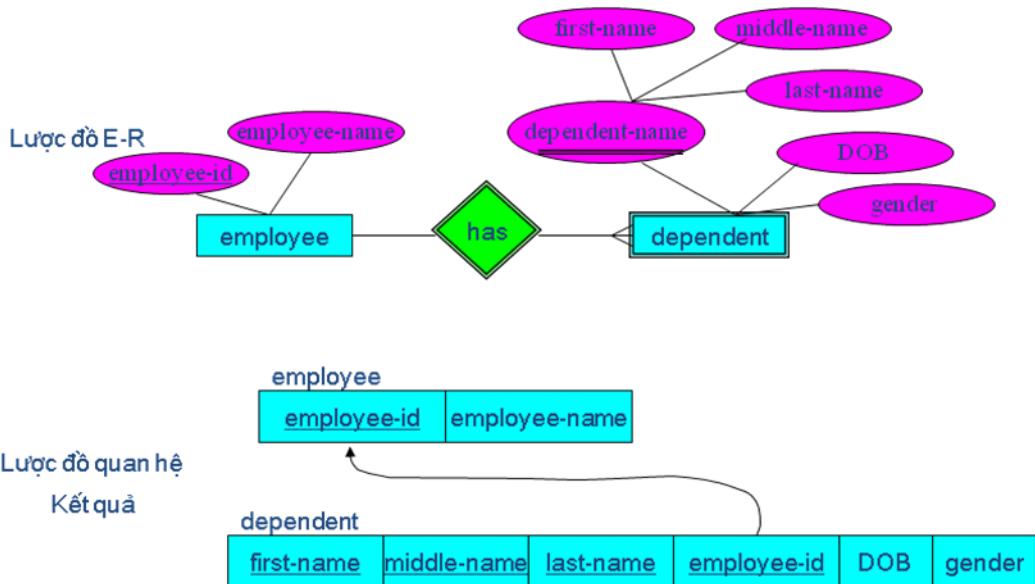


Lưu ý trong các lược đồ quan hệ trước đó được tạo ra bởi thuộc tính đa trị *skill*, lược đồ quan hệ kết quả *employee-skill* chỉ có các thuộc tính khóa. Mỗi bộ đơn giản lưu lại thực tế là một nhân viên nào đó sẽ có một kỹ năng tương ứng. Điều này cung cấp cho người thiết kế cơ sở dữ liệu cơ hội để gợi ý cho người dùng rằng các thuộc tính mới có thể được thêm vào quan hệ này. Ví dụ, các thuộc tính *years-experience* và/hoặc *certificate-date* có thể là các giá trị mới thích hợp với quan hệ này.

Bước 2: Ánh xạ các thực thể yếu

- Nhắc lại rằng một thực thể yếu không thể tồn tại độc lập, chỉ có thể tồn tại thông qua một quan hệ xác định với một loại thực thể khác được gọi là chủ thể.
- Một thực thể yếu không có một định danh đầy đủ mà phải có một thuộc tính được gọi là định danh một phần cho phép phân biệt các thể hiện khác nhau của loại thực thể yếu này cho mỗi thể hiện của thực thể chủ.
- Thủ tục sau đây giả sử rằng bạn đã tạo một lược đồ quan hệ liên quan tới loại thực thể xác định này. Nếu bạn chưa thực hiện- cần làm ngay bây giờ trước khi đi tiếp.
- Đối với mỗi loại thực thể yếu, tạo một lược đồ quan hệ mới và đưa tất cả các thuộc tính đơn (hoặc các thành phần đơn của các thuộc tính ghép) vào thành thuộc tính của lược đồ quan hệ này.
- Sau đó thêm khóa chính của quan hệ xác định vào thành một thuộc tính khóa ngoài trong lược đồ quan hệ mới.
- Khóa chính của lược đồ quan hệ mới là sự kết hợp của khóa chính của quan hệ xác định và định danh một phần của loại thực thể yếu.

Ví dụ về ánh xạ các thực thể yếu được thể hiện ở hình vẽ dưới đây



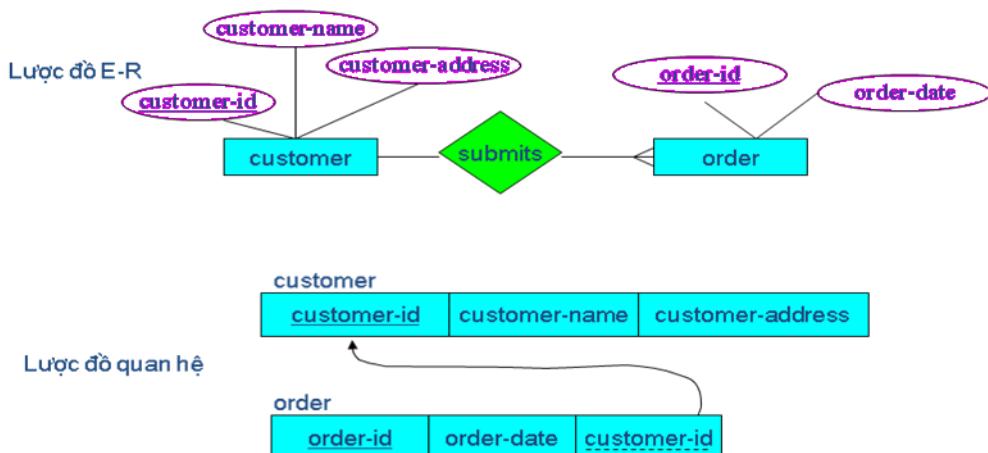
Bước 3: Ánh xạ các quan hệ hai ngôi

Thủ tục cho việc ánh xạ các quan hệ thành mô hình quan hệ phụ thuộc vào số ngôi của quan hệ (một ngôi, hai ngôi, ba ngôi, v.v...) và loại của các mối quan hệ (quan hệ 1-1, 1-nhiều hay nhiều-nhiều). Chúng ta sẽ xem xét việc ánh xạ cho các loại quan hệ đó trong phần này. Lưu ý là quan hệ lão 1-nhiều và nhiều-1 là đối xứng nhau.

Ánh xạ các quan hệ 1-nhiều hai ngôi

- Với mỗi quan hệ 1-nhiều hai ngôi, đầu tiên tạo một lược đồ quan hệ cho mỗi loại thực thể tham gia vào mối quan hệ sử dụng các thủ tục ở bước 1
- Sau đó, thêm thuộc tính khóa chính (hoặc các thuộc tính) của thực thể bên phía 1 của mối quan hệ thành một khóa ngoại cho quan hệ nằm ở bên phía nhiều của mối quan hệ (khóa chính lấy từ bên phía nhiều của mối quan hệ)

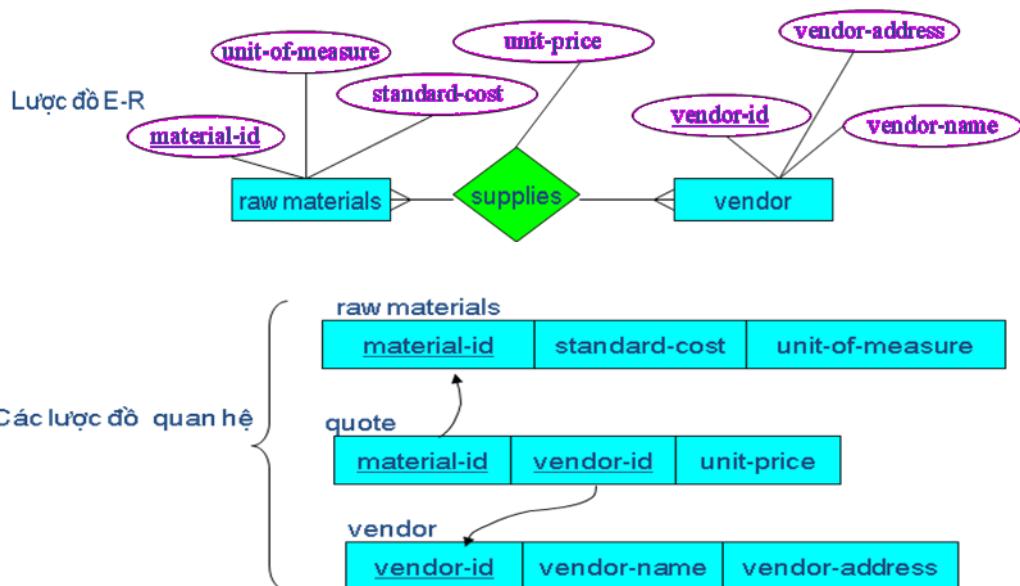
Ví dụ về cho quan hệ hai ngôi 1-nhiều



Ánh xạ cho các quan hệ nhiều-nhiều hai ngôi

Cho mỗi quan hệ hai ngôi nhiều-nhiều giữa hai thực thể loại A và B, đầu tiên phải tạo thêm một lược đồ quan hệ mới C. Khóa chính của một lược đồ C là sự kết hợp của các khóa chính của các tập thực thể tham gia quan hệ và các khóa chính này cũng là khóa ngoại của C. Các thuộc tính không phải là khóa mà liên quan tới quan hệ nhiều-nhiều giữa A và B cũng được đưa vào lược đồ quan hệ C.

Ví dụ việc chuyển đổi sang lược đồ quan hệ cho loại quan hệ nhiều-nhiều

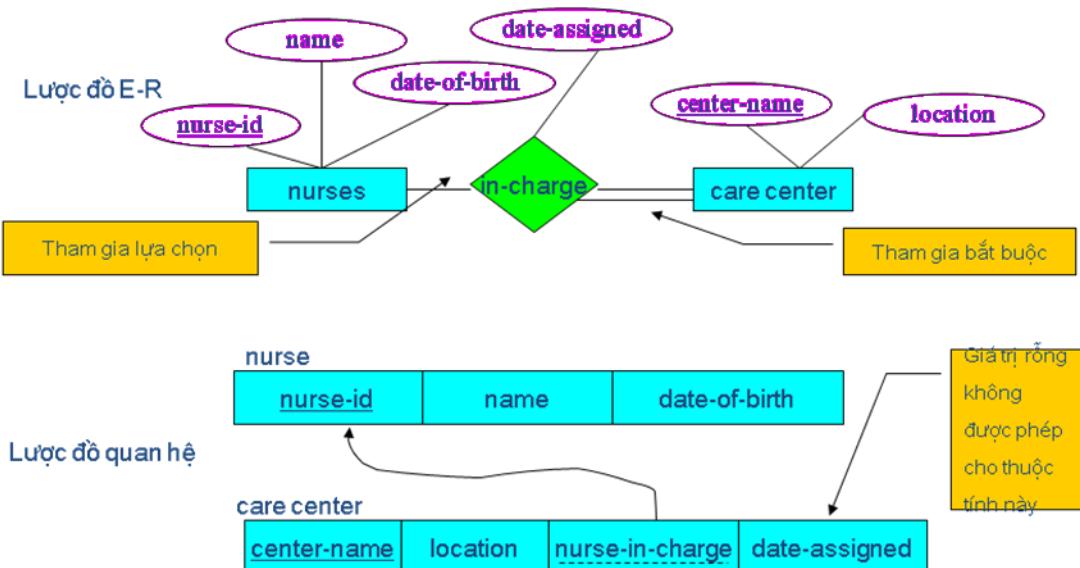


Ánh xạ sang lược đồ quan hệ cho loại quan hệ 1-1: quá trình này cần qua hai bước

- 1- Hai quan hệ được tạo ra, mỗi quan hệ liên quan tới một loại thực thể tham gia mỗi quan hệ đó.
- 2- Khóa chính của một quan hệ sẽ thành khóa ngoài trong quan hệ còn lại.

Trong một quan hệ 1-1, sự tham gia vào liên kết trong một hướng thường là một lựa chọn, trong khi với hướng kia là bắt buộc (nhắc lại loại ràng buộc tham gia liên kết). Bạn nên thêm vào quan hệ của bên có tham gia bắt buộc khóa ngoài của tập thực thể còn lại mà có tham gia không bắt buộc vào mối liên kết 1-1. Cách này sẽ tránh việc lưu trữ các giá trị rỗng cho thuộc tính khóa ngoài. Mọi thuộc tính liên quan tới bản thân quan hệ cũng được đưa vào cùng quan hệ đó như là khóa ngoài.

Ví dụ về việc chuyển đổi quan hệ hai ngôi 1-1 thể hiện ở hình vẽ dưới đây



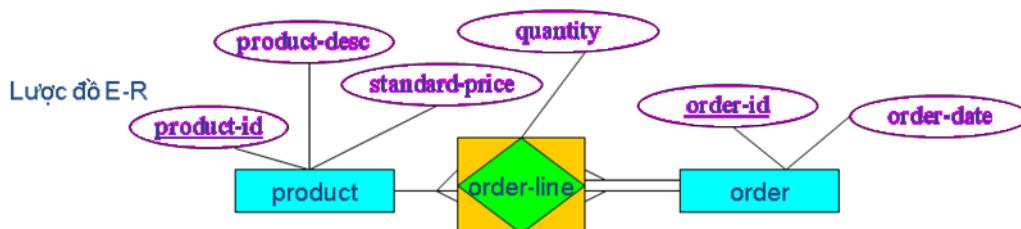
Bước 4: Ánh xạ các thực thể liên kết (hay thực thể kết hợp)

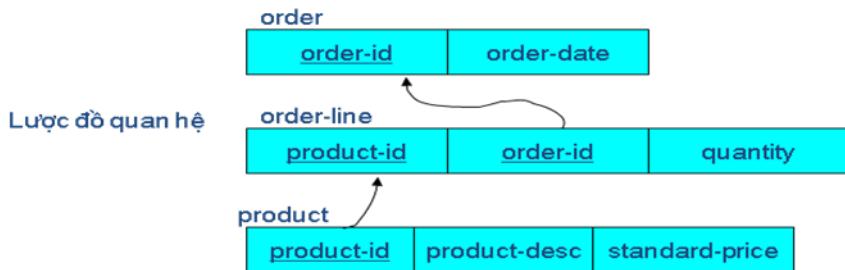
Ánh xạ một thực thể liên kết sang một lược đồ quan hệ tương tự thủ tục chuyển đổi một quan hệ nhiều-nhiều. Hai bước sau cần được thực hiện:

- 1- Ba quan hệ được tạo ra, trong đó hai quan hệ liên quan tới mỗi loại thực thể tham gia mỗi quan hệ đó và quan hệ thứ ba cho thực thể liên kết. Quan hệ được hình thành từ thực thể liên kết được gọi là quan hệ liên kết.
- 2- Hành động trong bước này phụ thuộc vào việc có gán một định danh cho thực thể liên kết trong lược đồ E-R hay không. Hai trường hợp sau xảy ra:
 - a. Một định danh không được gán
 - b. Một định danh được gán

Chúng ta sẽ xem xét từng trường hợp riêng biệt.

Trường hợp không gán định danh: khóa chính ngầm định cho quan hệ liên kết này bao gồm các thuộc tính khóa chính từ hai quan hệ còn lại. Những thuộc tính này sẽ là khóa ngoài tham chiếu tới hai quan hệ đó. Một ví dụ về trường hợp này được thể hiện dưới đây nhưng lưu ý sự giống nhau của ví dụ này với ví dụ của trường hợp chuyển đổi cho quan hệ nhiều-nhiều.



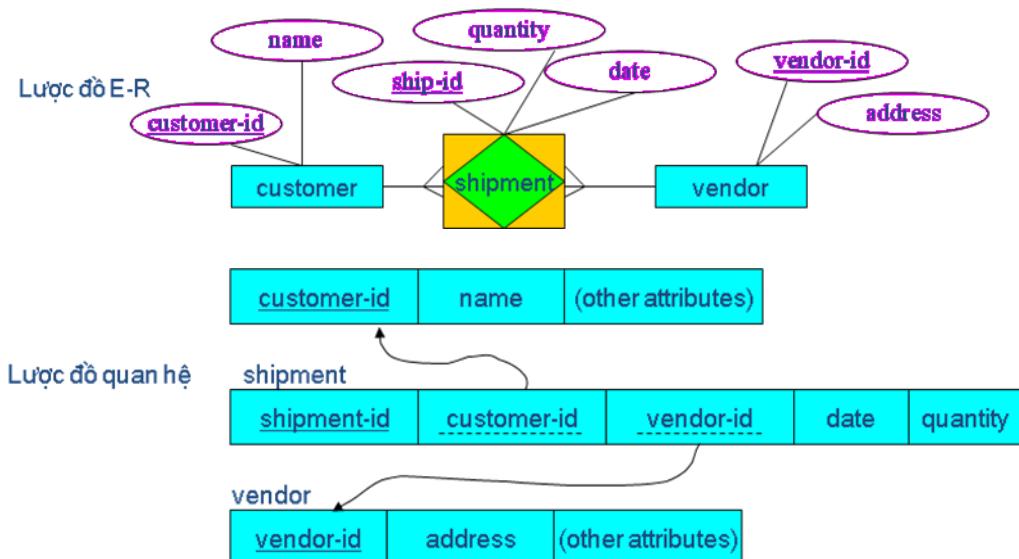


Đôi khi người xây dựng mô hình dữ liệu sẽ gán một định danh tới loại thực thể liên kết trong lược đồ E-R. Có hai lý do chính có thể xảy ra:

- 1- Thực thể liên kết có một định danh tự nhiên mà gần gũi với người sử dụng cuối.
- 2- Định danh ngầm định (bao gồm các định danh cho mỗi loại thực thể tham gia liên kết) có thể không xác định các thể hiện của thực thể liên kết một cách duy nhất.

Dù trong trường hợp nào trong hai trường hợp trên, quá trình để ánh xạ thực thể liên kết sẽ được thay đổi như sau.

Trường hợp một định danh được gán: như trên, một quan hệ liên kết mới được tạo ra để thể hiện thực thể liên kết. Khóa chính cho quan hệ này là một định danh được gán trên lược đồ E-R (hơn là giá khía ngầm định như trong trường hợp trước). Khóa chính cho hai loại thực thể tham gia sau đó sẽ được thêm vào làm khóa ngoài trong quan hệ liên kết này. Một ví dụ về trường hợp này xuất hiện như trong hình vẽ dưới đây.



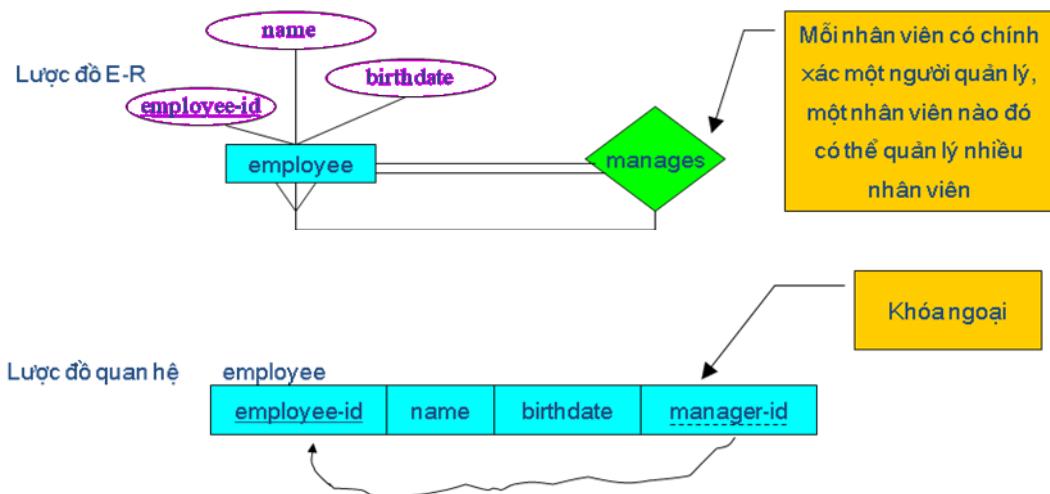
Bước 5: Ánh xạ các quan hệ một ngôi (đệ quy)

Nhắc lại rằng một quan hệ đệ quy được định nghĩa như một quan hệ giữa các thể hiện của cùng một loại thực thể. Hai trường hợp quan trọng nhất của các mối quan hệ một ngôi này là loại quan hệ 1-nhiều và nhiều-nhiều. Chúng ta sẽ bàn luận đến hai loại này tách biệt nhau vì chúng được xử lý khác nhau.

Ánh xạ quan hệ đệ quy loại 1-nhiều

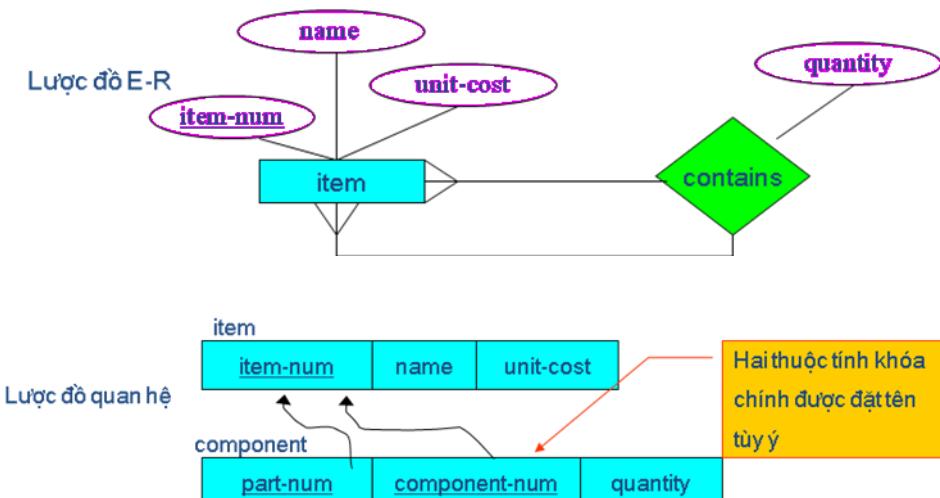
Tập thực thể trong quan hệ một ngôi được ánh xạ thành một lược đồ quan hệ sử dụng một thủ tục được mô tả trong bước 1. Tiếp đến một thuộc tính khóa ngoại được thêm vào cùng quan hệ có tham chiếu tới các giá trị của khóa chính (khóa ngoại này phải có cùng miền giá trị với khóa chính).

Một khóa ngoại đệ quy là một khóa ngoại của một quan hệ mà tham chiếu tới giá trị khóa chính của cùng quan hệ đó. Một ví dụ về trường hợp ánh xạ loại này được thể hiện trong hình vẽ dưới đây:



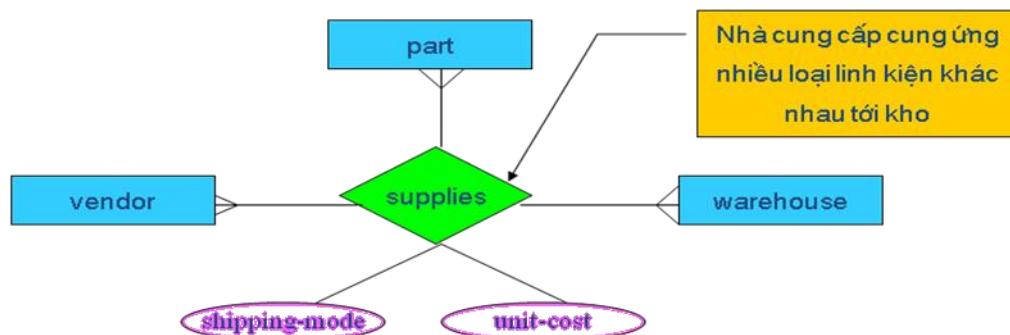
Ánh xạ quan hệ đệ quy loại nhiều-nhiều

Với loại quan hệ đệ quy loại này, hai lược đồ quan hệ sẽ được tạo ra: một thể hiện tập thực thể và lược đồ quan hệ liên kết để thể hiện mối quan hệ nhiều-nhiều. Khóa chính của quan hệ liên kết bao gồm hai thuộc tính. Những thuộc tính này (không nhất thiết phải có cùng tên) đều lấy giá trị của chúng từ các khóa chính của quan hệ còn lại. Các thuộc tính không khóa của quan hệ được thêm vào quan hệ liên kết. Ví dụ của trường hợp này về mối quan hệ phát sinh hóa đơn cho các nhiên vật liệu của các linh kiện được lắp ráp từ những linh kiện khác được thể hiện trong hình vẽ dưới đây.

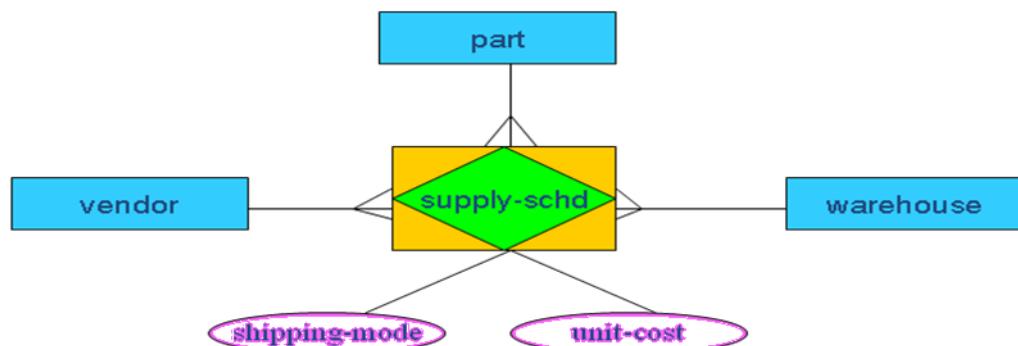


Bước 6: Ánh xạ các quan hệ nhiều ngôi

Nhắc lại rằng một mối quan hệ ba ngôi được định nghĩa như một mối quan hệ giữa ba tập thực thể như được biểu diễn trong hình vẽ dưới đây

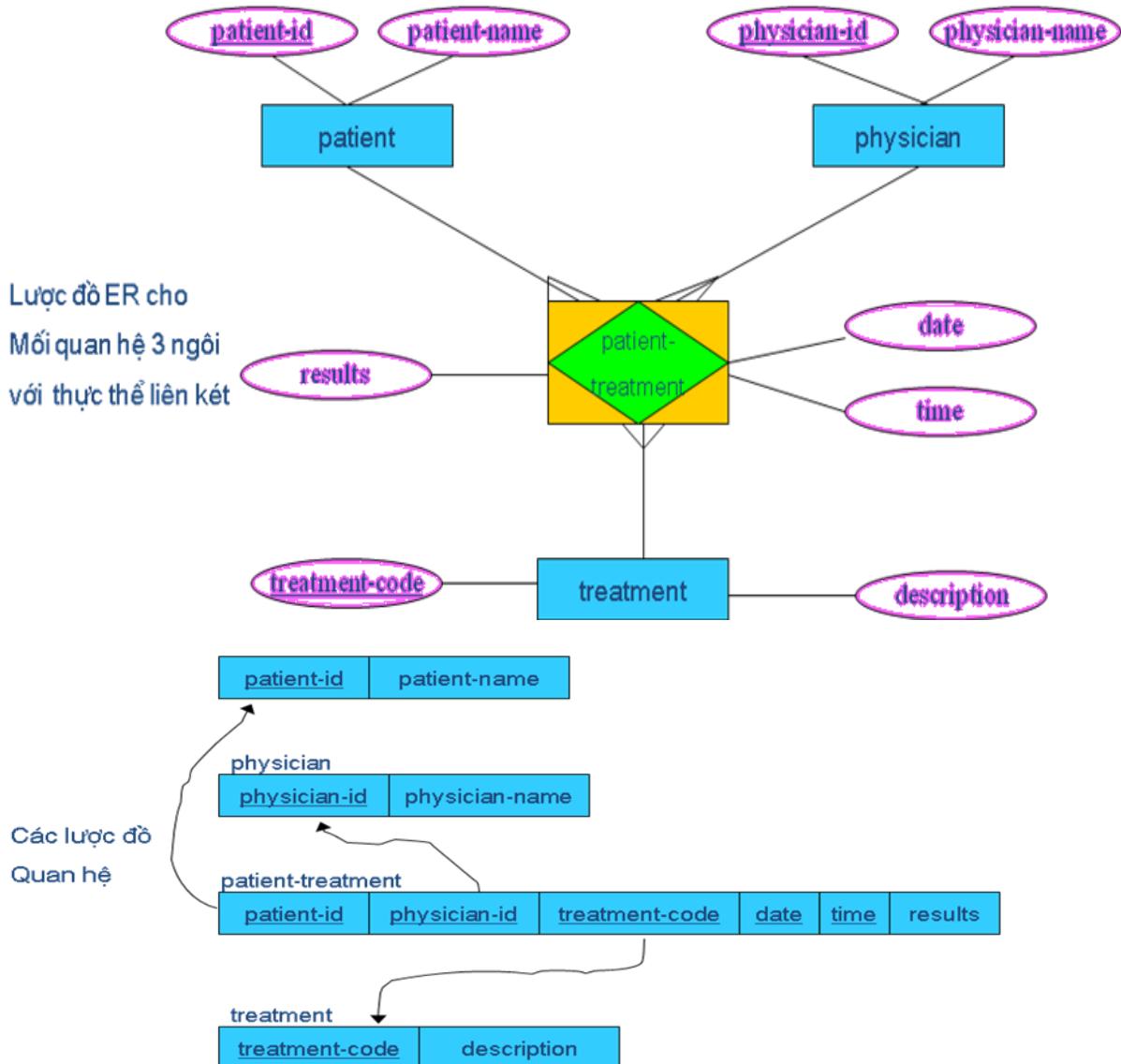


Một nháy nhỏ về quan hệ nhiều ngôi: tất cả nên chuyển đổi về các thực thể liên kết trước khi được xử lý sâu hơn. Một ví dụ về việc chuyển đổi về thực thể liên kết cho lược đồ E-R trên thể hiện trong hình vẽ dưới đây.



Để ánh xạ một thực thể liên kết nối ba tập thực thể loại thường, ta cần tạo ra một quan hệ liên kết mới. Khóa chính ngầm định cho quan hệ này bao gồm các thuộc tính khóa chính của các loại thực thể tham gia liên kết (trong một số trường hợp cần thêm các thuộc tính

khác để hình thành một khóa chính duy nhất). Những thuộc tính này hoạt động với vai trò các khóa ngoại tham chiếu tới từng khóa chính của các tập thực thể tham gia liên kết. Mỗi thuộc tính của loại thực thể liên kết này trở thành thuộc tính trong quan hệ liên kết mới. Ví dụ của quá trình ánh xạ các quan hệ nhiều ngôi được thể hiện trong hình vẽ dưới đây.



Bước 7: Ánh xạ các mối liên kết lớp cha/lớp con

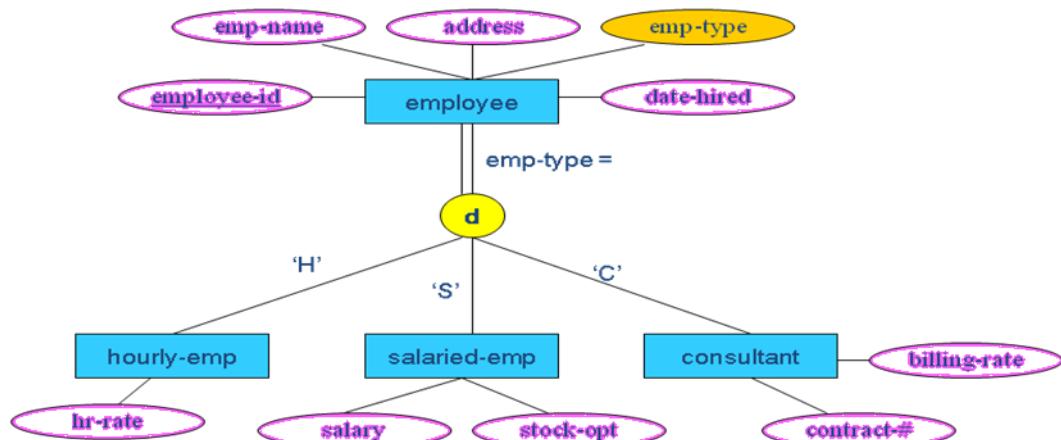
Mô hình dữ liệu quan hệ không hỗ trợ trực tiếp các mối liên kết loại này. May thay, có nhiều chiến lược khác nhau mà người thiết kế cơ sở dữ liệu có thể dùng để thể hiện loại mối liên kết này trong mô hình dữ liệu quan hệ. Chúng ta sẽ xem xét một trong những kỹ thuật chung nhất được sử dụng để mô hình hóa loại liên kết cha/con. Kỹ thuật đó được thể hiện qua các bước dưới đây

- 1- Tạo ra một lược đồ quan hệ riêng biệt cho tập thực thể lớp cha và cho mỗi tập thực thể con.

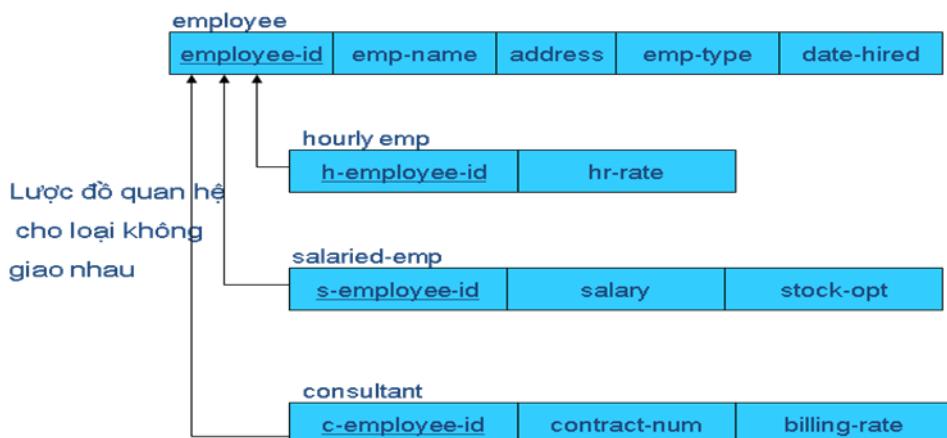
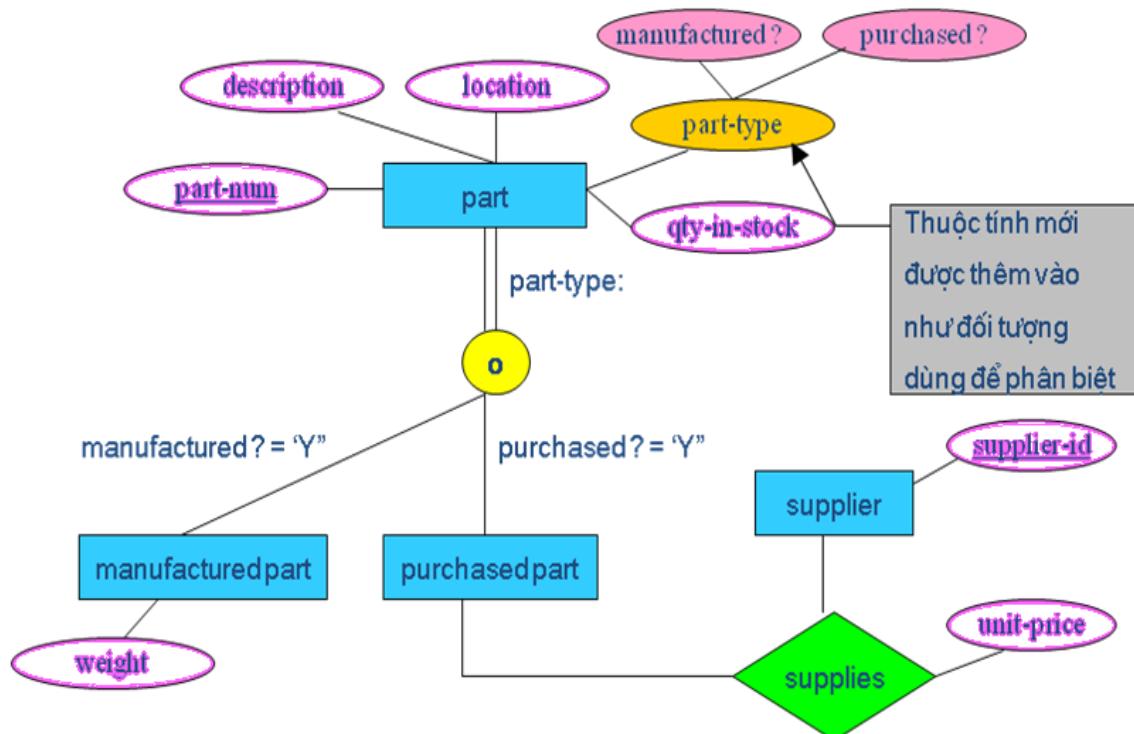
- 2- Gán cho lược đồ quan hệ tương ứng với lớp cha các thuộc tính chung của tất cả các thành viên lớp con bao gồm cả khóa chính.
- 3- Gán cho lược đồ quan hệ của mỗi lớp con khóa chính của lớp cha và chỉ những thuộc tính duy nhất cho từng loại tập thực thể con.
- 4- Gán một (hoặc nhiều) thuộc tính của tập thực thể lớp cha thực hiện chức năng như một chỉ thị phân biệt các loại lớp con.

Xác định các thuộc tính phân biệt các lớp con: Với một mối quan hệ cha/con nào đó, nếu muốn thêm một thể hiện của lớp cha vào cơ sở dữ liệu thì thể hiện này sẽ nên được thêm vào lớp con nào? Một cách tiếp cận chung sử dụng một đối tượng phân biệt. Một đối tượng phân biệt các tập thực thể con là một thuộc tính của tập thực thể cha mà giá trị của nó xác định được các loại tập thực thể con (được sử dụng khi việc xác định các lớp con dựa trên các mệnh đề). Có hai trường hợp xảy ra: loại tập thực thể con không giao nhau và loại các tập thực thể con giao nhau.

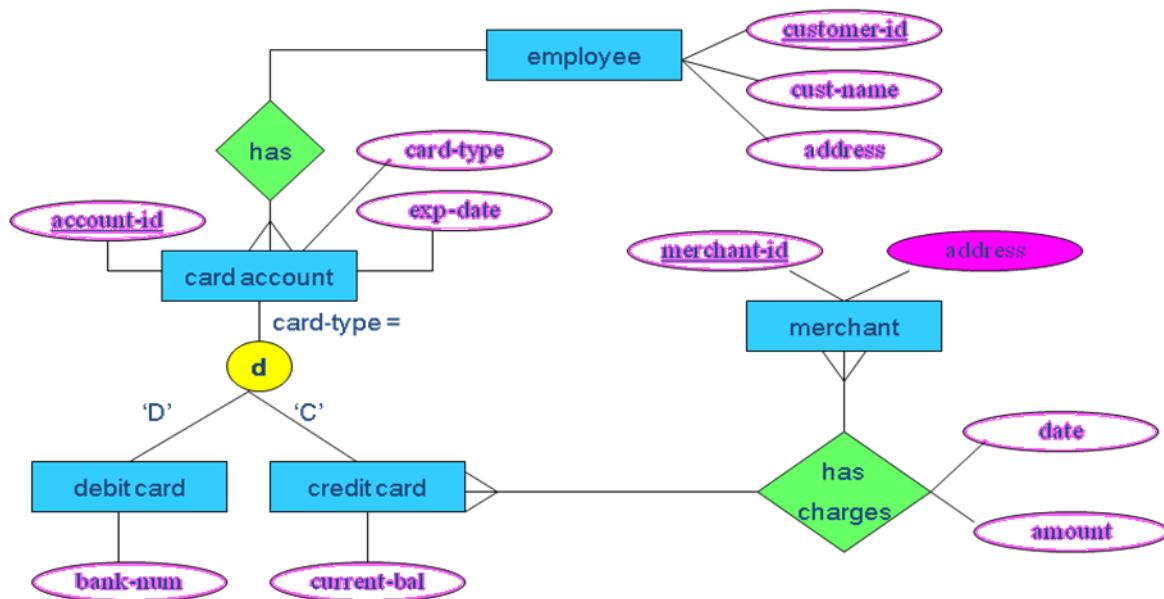
Ví dụ về đối tượng phân biệt trong lược đồ E-R cho loại tập thực thể không giao nhau



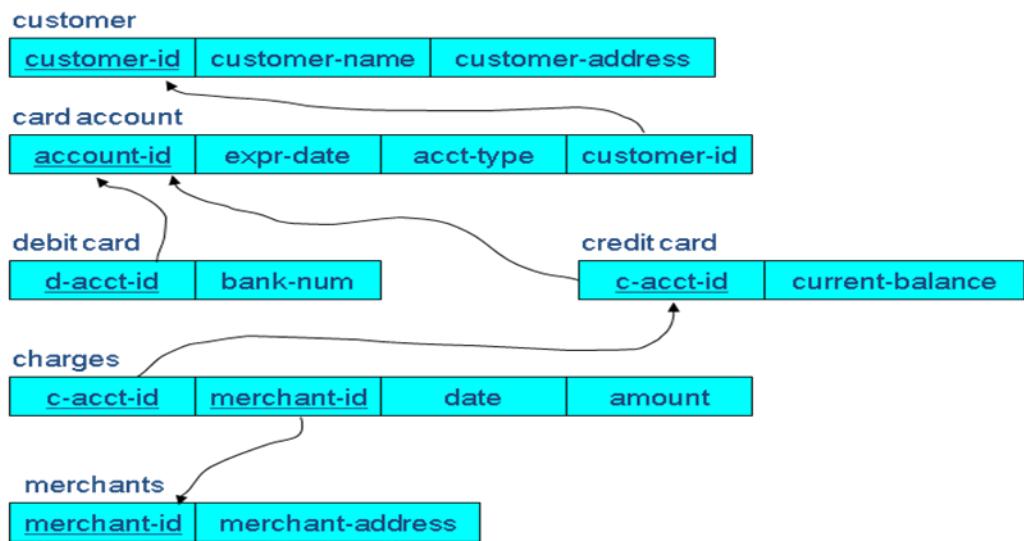
Ví dụ về đối tượng phân biệt trong lược đồ E-R cho loại tập thực thể giao nhau



*****Bài tập thực hành thêm ở nhà: hãy chuyển đổi lược đồ E-R sau sang các lược đồ quan hệ



Giải pháp cho bài tập thực hành trên



Bài 7 Ngôn ngữ truy vấn

Một ngôn ngữ truy vấn là một ngôn ngữ trong đó một người sử dụng cơ sở dữ liệu có thể yêu cầu thông tin từ cơ sở dữ liệu. Hầu hết các ngôn ngữ truy vấn đều ở một mức cao hơn các ngôn ngữ lập trình chuẩn như C hay Java. Các ngôn ngữ truy vấn thuộc nhóm ngôn ngữ thế hệ thứ tư.

Các ngôn ngữ truy vấn có thể được phân loại chung thành hai nhóm: ngôn ngữ có thủ tục và ngôn ngữ phi thủ tục. Một ngôn ngữ truy vấn thủ tục đòi hỏi người sử dụng xác định một chuỗi các phép toán trên cơ sở dữ liệu để tính kết quả mong muốn (Người sử dụng sẽ xác định cụ thể cần cái gì và thực hiện như thế nào). Một ngôn ngữ phi thủ tục đòi hỏi người sử dụng mô tả kết quả mong muốn mà không cần xác định chuỗi các thao tác cần thiết để đạt được kết quả đó (Người sử dụng chỉ xác định cụ thể cần cái gì).

Hầu hết các hệ thống cơ sở dữ liệu quan hệ có sẵn trên thị trường cung cấp một ngôn ngữ truy vấn thuộc loại trung gian lai ghép giữa hai loại trình bày trên. Ngôn ngữ truy vấn lai bao gồm các thành phần của cả hai cách tiếp cận có thủ tục và phi thủ tục tới ngôn ngữ truy vấn. Chúng ta chỉ xem xét các ngôn ngữ truy vấn quan hệ thuần túy. Những ngôn ngữ này là chính thống và thiếu nhiều cú pháp mở rộng có sẵn trong các ngôn ngữ trên thị trường hiện nay nhưng nó mô tả những kỹ thuật nền tảng được sử dụng bởi tất cả các ngôn ngữ truy vấn để lấy dữ liệu ra từ cơ sở dữ liệu.

Khi chúng ta xem xét các ngôn ngữ thuần túy này, mặc dù các ngôn ngữ chuẩn này không chứa các đặc tính mở rộng, một ngôn ngữ truy vấn hoàn chỉnh cung cấp các cách thức để thêm và xóa các bộ từ các quan hệ cũng như cho phép thay đổi các bộ đang có. Ngôn ngữ có thủ tục được xem xét đến là ngôn ngữ đại số quan hệ. Ngôn ngữ phi thủ tục có thể kể đến trong phạm vi môn học này là tính toán bộ và tính toán vị từ.

Ngôn ngữ đại số quan hệ

Là một ngôn ngữ truy vấn thủ tục. Nó bao gồm các phép toán tập hợp hoặc là loại hai ngôi hoặc là một ngôi, có nghĩa là các toán hạng của các phép toán tập hợp này sẽ là một quan hệ hoặc hai quan hệ. Mỗi một phép toán tập hợp này sẽ sinh ra một quan hệ là kết quả đầu ra.

Năm phép toán cơ bản của đại số quan hệ và một số phép toán mở rộng bổ sung dựa trên năm phép toán cơ bản này sẽ được chúng ta xem xét trong phạm vi bài giảng này. Một phép toán đặt lại tên *rename* đôi khi cũng được coi là phép toán cơ bản. Năm phép toán cơ bản bao gồm: phép chọn, phép hợp, phép chiếu, phép trừ, và tích Đè các. Chúng ta sẽ xem xét mỗi phép toán riêng rẽ trước khi kết hợp các phép toán đó vào một biểu thức phức tạp hơn.

Năm phép toán cơ bản của đại số quan hệ là phép chọn, phép chiếu, phép hợp, phép trừ và phép tích. Đề các. Một số các phép toán bổ sung khác cũng được định nghĩa trong đại số quan hệ bao gồm: phép giao, kết nối tự nhiên, phép chia, phép nửa kết nối và kết nối ngoài. Chúng ta sẽ lần lượt xem xét từng phép toán một trước khi dùng chúng trong một biểu thức đại số phức tạp.

Phép chọn

Là một phép toán một ngôi được ký hiệu bởi một chữ cái Hy lạp là sigma, σ

Dạng chung của phép chọn này như sau: $\sigma_{(mệnh đề)}(\text{thể hiện một quan hệ})$

Lược đồ của quan hệ kết quả: giống hết như quan hệ toán hạng.

Kích cỡ của quan hệ kết quả (số bộ) $\leq |\text{quan hệ toán hạng}|$

Ví dụ:

$\sigma_{(\text{major} = "CS")}(\text{students})$

$\sigma_{(\text{major} = "CS" \wedge \text{hair-color} = "brown")}(\text{students})$

$\sigma_{(\text{hours-attempted} > \text{hours-earned})}(\text{students})$

Phép chọn này sẽ lựa chọn ra những bộ từ thể hiện của một quan hệ mà thỏa mãn một mệnh đề cụ thể nào đó. Nhìn chung, một mệnh đề có thể chứa bất kể một toán tử so sánh logic nào trong số $=, \neq, <, \leq, >, \geq$ hơn nữa một số các mệnh đề có thể kết hợp chúng với các phép toán liên kết như *and* (\wedge), *or* (\vee), và *not* (\neg). Phép toán lựa chọn có thể được hình dung như một lát cắt ngang của quan hệ toán hạng. Ví dụ về toán tử lựa chọn được thể hiện dưới đây với một quan hệ R

R

A	B	C	D
a	a	yes	1
b	d	no	7
c	f	yes	34
a	d	no	6
a	c	no	7
b	b	no	69
c	a	yes	24
d	d	yes	47
h	d	yes	34
e	c	no	26
a	a	yes	5

$r = \sigma_{(A = 'a')}(\text{R})$

A	B	C	D
a	a	yes	1
a	d	no	6
a	c	no	7
a	a	yes	5

$r = \sigma_{(A = 'a' \wedge C = 'yes')}(\text{R})$

A	B	C	D
a	a	yes	1
a	a	yes	5

an empty
relation

$r = \sigma_{(B = 'm')}(\text{R})$

A	B	C	D

Phép chiếu

Là một loại phép toán một ngôi được ký hiệu bởi một chữ cái Hy Lạp pi: π

Dạng chung của phép toán này là: $\pi_{(\text{danh sách các thuộc tính})}(\text{thể hiện của quan hệ})$

Lược đồ của quan hệ kết: được xác định bởi $\langle \text{danh sách các thuộc tính} \rangle$

Kích cỡ của quan hệ kết quả (số bộ): $\leq |\text{quan hệ toán hạng}|$

Ví dụ:

$\pi_{(\text{student-id, name, major})}(\text{students})$

$\pi_{(\text{name, advisor})}(\text{students})$

$\pi_{(\text{name, gpa, hours-attempted})}(\text{students})$

Phép chiếu này có thể được hình dung như việc sinh ra một lát cắt theo chiều dọc của quan hệ toán hạng. Nếu phép toán sinh ra các bộ giống hệt nhau, những bộ này sẽ thường được loại bỏ khỏi quan hệ kết quả trong việc giữ chức năng set-like. Ví dụ về phép toán này với một thể hiện của quan hệ R như sau:

R

A	B	C	D
a	a	yes	1
b	d	no	7
c	f	yes	34
a	d	no	6
a	c	no	7
b	b	no	69
c	a	yes	24
d	d	yes	47
h	d	yes	34
e	c	no	26
a	a	yes	5

$r = \pi_{(A, C)}(R)$

A	C
a	yes
b	no
c	yes
a	no
d	yes
h	yes
e	no

$r = \pi_{(A, D)}(R)$

A	D
a	1
b	7
c	34
a	6
a	7
b	69
c	24
d	47
h	34
e	26
a	5

$r = \pi_{(C)}(R)$

C
yes
no

Phép hợp

Là một phép toán hai ngôi được ký hiệu bằng biểu tượng hợp \cup .

Dạng chung của phép toán này là: $r \cup s$, với r và s là hai quan hệ khả hợp.

Lược đồ của quan hệ kết quả chính là lược đồ của các quan hệ toán hạng.

Kích cỡ của quan hệ kết quả (số bộ): $\leq \max\{ |r| + |s| \}$

Ví dụ:

$$r \cup s \quad \pi_{(a, b)}(r) \cup \pi_{(a, b)}(s)$$

Phép hợp này cung cấp một phương tiện để trích lọc thông tin nằm trên hai quan hệ toán hạng, hai quan hệ này phải khả hợp với nhau. Khả hợp là một đặc tính mà hai điều kiện sau đây phải được thỏa mãn:

1. Hai quan hệ $r(R)$ và $s(S)$ trong biểu thức $r \cup s$ phải cùng cấp, có nghĩa là chúng phải có cùng số lượng thuộc tính.

2. Miền giá trị của thuộc tính thứ (i) của r và thuộc tính thứ (i) của s phải giống nhau, cho mọi giá trị của i.

Ví dụ về toán tử hợp (union) với thể hiện quan hệ R, T, S, X được thể hiện dưới đây.

T			$r = R \cup T$ Không hợp lệ bởi vì R và T không tương thích			$r = R \cup S$		
A	B	D	A	B		E	F	G
a	a	1	a	a		a	a	1
b	d	7	b	d		b	d	7
c	f	34	c	f		c	f	34
a	d	6	a	d		a	d	6
a	c	7	a	c		a	c	7

s			$r = T \cup X$					
X	Y	Z	A	B		E	F	G
a	m	4	a	a		a	m	4
b	c	22	b	d		b	c	22
a	d	16	c	f		a	d	16
a	c	7	a	d				

x								
X	Y	Z	A	B				
a	m	4	a	a				
b	c	22	b	d				
a	d	16	c	f				
a	c	7	a	d				

Phép trừ

Là một phép toán hai ngôi được ký hiệu với biểu tượng: – .

Dạng chung của phép toán này là: $r - s$, với r và s là phải khả hợp.

Lược đồ của quan hệ kết quả: là lược đồ của quan hệ toán hạng.

Kích cỡ của quan hệ kết quả (số bộ): $\leq |$ quan hệ r $|$ (lực lượng của quan hệ r)

Ví dụ: $r - s$

Phép toán trừ này cho phép trích lọc các thông tin được chứa trong một quan hệ mà không chứa trong quan hệ thứ hai. Giống như phép toán hợp, phép trừ đòi hỏi rằng hai quan hệ toán hạng là khả hợp. Ví dụ về phép toán này được thể hiện như dưới đây.

R		
A	B	D
a	a	1
b	d	7
c	f	34
a	d	6
a	c	7

T	
A	B
a	a
b	d
c	f
a	d
a	c

$r = R - T$
Không hợp lệ bởi vì R
và T không khả hợp

A		B
c	f	
a	d	

r = R - S		
E	F	G
a	a	1
b	d	7
c	f	34
a	d	6

r = S - R		
E	F	G
a	m	4
b	c	22
a	d	16
a	c	7

S

X	Y	Z
a	m	4
b	c	22
a	d	16
a	c	7

X

A		B
a	a	
b	d	
a	c	

$r = X - T$

A		B

Quan hệ rỗng

Phép tích đề các

Là phép toán một ngôi với ký hiệu là: \times .

Dạng chung của phép toán này là: $r \times s$ (không có hạn chế trên r và s)

Lược đồ của quan hệ kết quả: lược đồ của $r \times s$ với việc đặt lại tên một số thuộc tính.

Kích cỡ của quan hệ kết quả (số bộ): $> |\text{quan hệ } r| \text{ và } > |\text{quan hệ } s|$

Ví dụ: $r \times s$

Phép tích đề các cho phép kết hợp hai quan hệ bất kỳ thành một quan hệ đơn. Gợi nhớ rằng một quan hệ được định nghĩa là một tập con của tích đề các tập miền giá trị vì vậy điều này khiến bạn có một số ý tưởng về cách hành xử của phép toán tích đề các. Ví dụ về phép toán này với hai quan hệ T và X được thể hiện trong hình vẽ dưới đây.

A		B
a	a	
b	d	

X

A		B
a	a	
b	d	
a	c	
c	a	

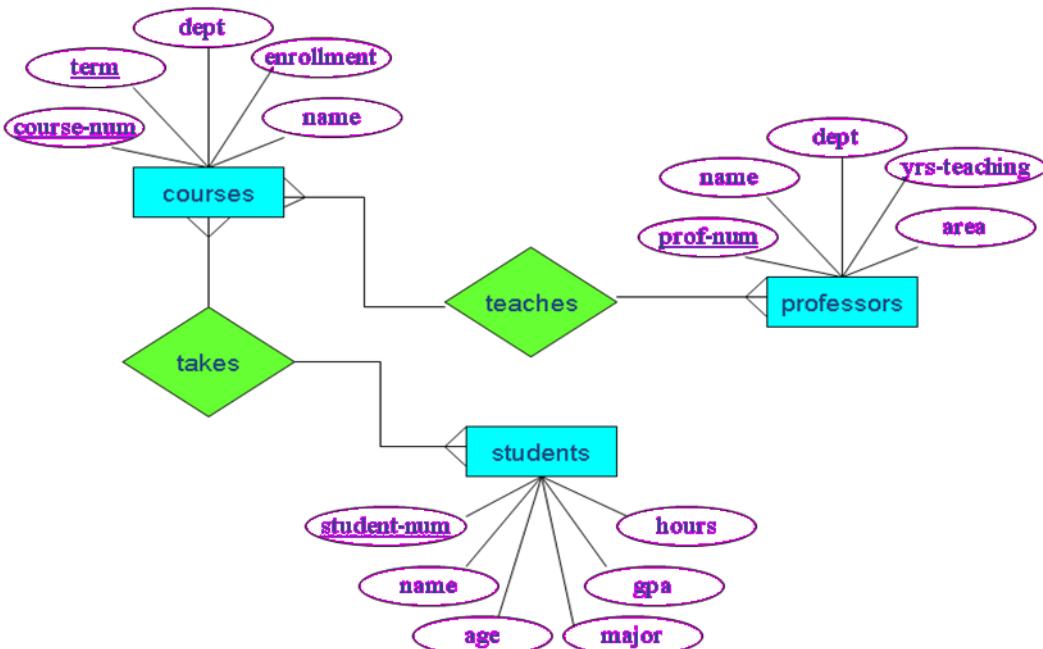
$r = T \times X$

T.A	T.B	X.A	X.B
a	a	a	a
a	a	b	d
a	a	a	c
a	a	c	a
b	d	a	a
b	d	b	d
b	d	a	c
b	d	c	a

R				$r = R \times S$						
A	B	C	D	A	B	C	D	X	Y	Z
a	a	1	yes	a	a	1	yes	a	m	4
b	d	7	yes	a	a	1	yes	b	c	22
c	f	34	no	a	a	1	yes	a	d	16
				a	a	1	yes	a	c	7
				b	d	7	yes	a	m	4
				b	d	7	yes	b	c	22
				b	d	7	yes	a	d	16
				b	d	7	yes	a	c	7
				c	f	34	no	a	m	4
				c	f	34	no	b	c	22
				c	f	34	no	a	d	16
				c	f	34	no	a	c	7

Biểu thức đại số quan hệ

Trong khi mỗi một toán tử đại số quan hệ cơ bản trong số năm toán tử trình bày ở trên có thể được sử dụng riêng rẽ để hình thành một truy vấn, sức mạnh thề hiện của chúng thực sự được cải thiện đáng kể khi chúng được kết hợp với nhau để hình thành các biểu thức truy vấn. Trước khi giới thiệu những phép toán bổ sung trong đại số quan hệ chúng ta sẽ xem xét việc tạo ra các tổ hợp phức tạp hơn của năm phép toán cơ bản này. [Việc này cũng sẽ khiến cho bạn đánh giá cao hơn các phép toán bổ sung này].



Sử dụng các kỹ thuật để chuyển đổi một lược đồ E-R sang thành một tập các lược đồ quan hệ chúng ta có các lược đồ kết quả như sau:

$S = STUDENTS(s\#, name, age, major, gpa, hours_completed)$

$C = COURSES(c\#, term, name, dept, enrollment)$

$P = PROFESSORS(p\#, name, dept, yrs_teaching, area)$

$TA = TAKES(s\#, c\#, term, grade)$

$TE = TEACH(p\#, c\#, term)$

Khi bạn bắt đầu viết một câu truy vấn trong một ngôn ngữ truy vấn mới, việc trực quan thông qua thể hiện dữ liệu của các quan hệ toán hạng cần thiết cho một phép toán mà bạn đang thực hiện đôi khi rất có ích. Bạn nên tự trải nghiệm điều này qua thể hiện của các quan hệ trên sao cho bạn có thể hình dung trực quan được tuy nhiên, đây cũng là thứ mà bạn sẽ cần loại bỏ khi bạn trở nên có kinh nghiệm hơn trong việc tạo các câu truy vấn bởi vì bạn không muốn làm ảnh hưởng đến bản thiết kế các truy vấn bởi sự trực quan một thể hiện quan hệ mà nó có thể không chứa tất cả các bộ có thể có cho câu truy vấn trong quá trình thực hiện.

Ví dụ truy vấn 1: Tìm tên của tất cả các sinh viên học chuyên ngành công nghệ thông tin (Computer Science majors). Cách tiếp cận truy vấn này như sau:

- Đầu tiên chọn tất cả các sinh viên mà học chuyên ngành CS

$$r = \sigma_{(major = "Computer Science")}(S)$$

- Tiếp đến chiểu kết quả trên lên thuộc tính *name*:

$$\text{result} = \pi_{(name)}(r)$$

Biểu thức truy vấn hoàn chỉnh là:

$$\text{result} = \pi_{(name)}(\sigma_{(major = "Computer Science")}(S))$$

Ví dụ truy vấn 2: Tìm số hiệu sinh viên s# và tên của tất cả các sinh viên đã hoàn thành hơn 90 giờ học (theo hệ thống học tín chỉ). Cách tiếp cận truy vấn này như sau:

- Đầu tiên chọn tất cả các sinh viên đã hoàn thành hơn 90 giờ học

$$r = \sigma_{(hours_completed > 90)}(S)$$

- Tiếp đến chiểu kết quả trên lên thuộc tính mã hiệu sinh viên và tên sinh viên

$$\text{result} = \pi_{(s\#, name)}(r)$$

Biểu thức truy vấn hoàn chỉnh là

$$\text{result} = \pi_{(s\#, name)}(\sigma_{(hours_completed > 90)}(S))$$

Ví dụ truy vấn 3: Tìm tên của tất cả các sinh viên dưới 20 tuổi mà đã hoàn thành nhiều hơn 80 giờ học. Cách tiếp cận truy vấn này như sau:

- Đầu tiên chọn tất cả các sinh viên đã hoàn thành hơn 80 giờ học và ít hơn 20 tuổi

$$r = \sigma_{((hours_completed > 80) \wedge (age < 20))}(S)$$

- Tiếp đến chiểu kết quả trên lên thuộc tính tên sinh viên

$$\text{result} = \pi_{(\text{name})}(\text{r})$$

Biểu thức truy vấn hoàn chỉnh là:

$$\text{result} = \pi_{(\text{name})}(\sigma_{((\text{hours_completed} > 80) \wedge (\text{age} < 20))}(\text{S}))$$

Ví dụ truy vấn 4: Tìm tên của tất cả các lớp học do khoa công nghệ thông tin hoặc khoa vật lý tổ chức. Cách tiếp cận truy vấn này như sau:

- Đầu tiên lựa chọn tất cả các lớp học được khoa công nghệ thông tin tổ chức

$$\text{r} = \sigma_{((\text{dept} = \text{Computer Science}) \vee (\text{dept} = \text{Physics}))}(\text{C})$$

- Tiếp đó chiết kết quả trên lên thuộc tính tên

$$\text{result} = \pi_{(\text{name})}(\text{r})$$

Biểu thức truy vấn hoàn chỉnh là:

$$\text{result} = \pi_{(\text{name})}(\sigma_{((\text{dept} = \text{Computer Science}) \vee (\text{dept} = \text{Physics}))}(\text{C}))$$

Ví dụ truy vấn 5: Tìm tên của tất cả các giáo sư đã dạy một lớp trong kỳ mùa thu 2002 (Fall 2002). Cách tiếp cận truy vấn như sau:

- Đầu tiên đưa toàn bộ thông tin về giáo sư vào cùng với thông tin của khóa học.
- Tiếp đến, lựa chọn chỉ những giáo sư và những lớp học liên quan tới nhau và thỏa mãn điều kiện.
- Cuối cùng, chiết kết quả trên lên tên sinh viên.

Biểu thức truy vấn hoàn chỉnh là:

$$\text{result} = \pi_{(\text{P.name})}(\sigma_{((\text{TE.term} = \text{Fall 2002}) \wedge (\text{P.p\#} = \text{TE.p\#}))}(\text{P} \times \text{TE}))$$

Ví dụ truy vấn 6: Tìm tên của tất cả các sinh viên đã tham gia một lớp học trong kỳ mùa thu 2003 mà được giảng bởi một giáo sư có hơn 20 năm kinh nghiệm giảng dạy.

Cách tiếp cận truy vấn này như sau:

- Đầu tiên ghép thông tin về giáo sư và lớp học vào nhau
- Tiếp đến, lựa chọn chỉ những sinh viên, giáo sư và khóa học liên quan và thỏa mãn điều kiện từ kết quả trên
- Cuối cùng, chiết kết quả trên lên tên sinh viên.

Biểu thức truy vấn hoàn chỉnh là:

$$\text{result} = \pi_{(\text{S.name})}(\sigma_{((\text{TA.term} = \text{Fall 2003}) \wedge (\text{P.yrs_teaching} > 20) \wedge (\text{S.s\#} = \text{TA.s\#}) \wedge (\text{P.p\#} = \text{TE.p\#}) \wedge (\text{TA.c\#} = \text{TE.c\#}) \wedge (\text{TA.term} = \text{TE.term}))}(\text{S} \times \text{P} \times \text{TA} \times \text{TE}))$$

Ví dụ truy vấn 7: Tìm tên của tất cả các giáo sư hoặc thuộc khoa công nghệ thông tin hoặc có kinh nghiệm giảng dạy trên 20 năm.

Biểu thức truy vấn hoàn chỉnh là:

$$\text{result} = [\pi_{(\text{name})}(\sigma_{(\text{dept} = \text{Computer Science})(\text{P})})] \cup [\pi_{(\text{name})}(\sigma_{(\text{yrs_teaching} > 20)}(\text{P}))]$$

$$\text{result} = \pi_{(\text{name})}(\sigma_{((\text{dept} = \text{Computer Science}) \vee (\text{yrs_teaching} > 20))}(\text{P}))$$

Ví dụ truy vấn 8: Tìm mã hiệu của các sinh viên chỉ đăng ký học vào kỳ mùa xuân năm 2003.

Biểu thức truy vấn hoàn chỉnh như sau

$$\text{result} = [\pi_{(\text{TA.s\#})}(\sigma_{(\text{TA.term} = \text{Spring 2003})(\text{TA})})] - [\pi_{(\text{TA.s\#})}(\sigma_{(\text{TA.term} \neq \text{Spring 2003})}(\text{TA}))]$$

Lưu ý biểu thức truy vấn sau đây không đúng cho câu truy vấn này

result = $\pi_{(TA.s\#)}(\sigma_{(TA.term = Spring 2003)}(TA))$

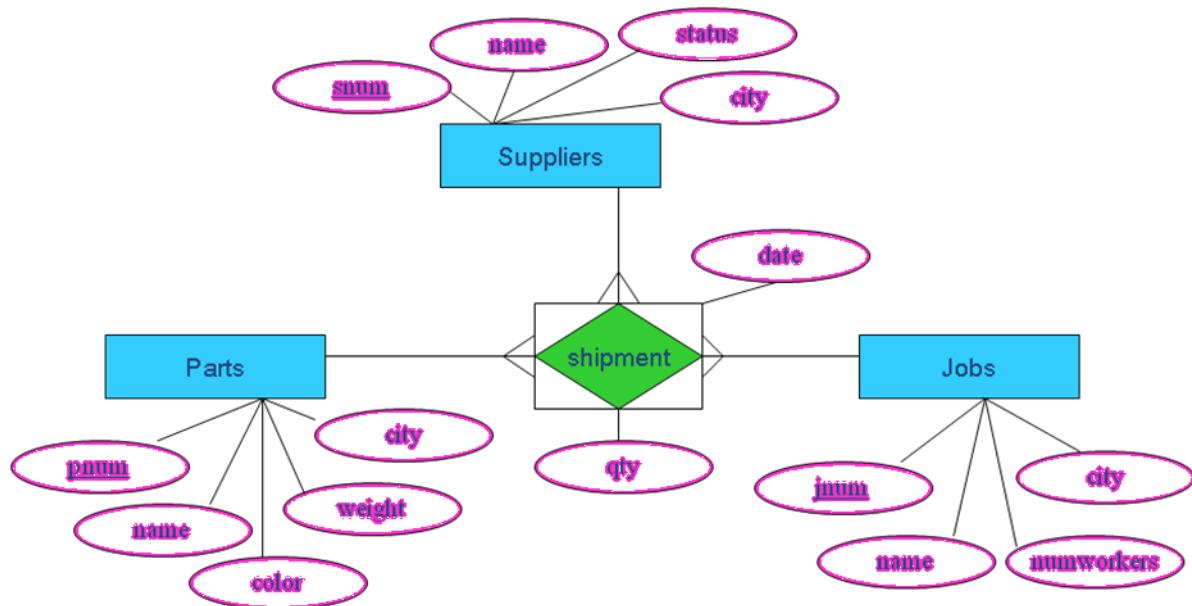
bởi vì nó chỉ liệt kê những sinh viên có tham gia đăng ký học vào kỳ mùa xuân 2003 và cũng có thể các sinh viên đó có tham gia vào các kỳ học khác nữa.

Bài 8 Ngôn ngữ truy vấn quan hệ - phần 2

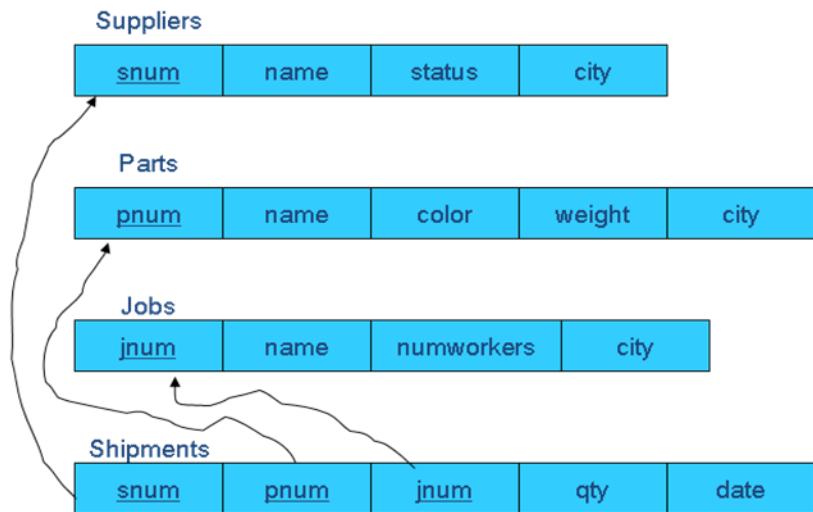
Các toán tử bổ sung trong đại số quan hệ

Có thể chứng minh được rằng (mặc dù chúng ta sẽ không đề cập tới việc chứng minh trong bài giảng này) năm phép toán cơ bản được trình bày ở bài trước là đủ để thể hiện câu truy vấn trong đại số quan hệ. Tuy nhiên quá trình chứng minh không phát biểu rằng một số câu truy vấn phức tạp sẽ đòi hỏi các biểu thức truy vấn khó và dài dòng. Đó chính là nguyên nhân một số các phép toán khác ra đời trong đại số quan hệ để cung cấp sức mạnh thể hiện biểu thức truy vấn và cung cấp một sự đơn giản hóa trong việc thể hiện những câu truy vấn phức tạp. Chúng ta sẽ xem xét một số các phép toán quan trọng và chung nhất và sẽ chỉ ra cách định nghĩa chúng theo cách thể hiện qua các phép toán cơ bản.

Chúng ta lại xem xét lược đồ E-R cho một cơ sở dữ liệu để làm ví dụ cho các phép toán mở rộng này (như hình vẽ dưới đây)



Lược đồ cơ sở dữ liệu tương ứng với lược đồ E-R trên được thể hiện trong hình vẽ dưới đây



Phép giao

Là một phép toán hai ngôi, được ký hiệu bởi biểu tượng \cap .

Dạng chung của phép toán này là: $r \cap s$ với r và s là hai quan hệ khả hợp

Lược đồ của quan hệ kết quả: là lược đồ của quan hệ toán hạng.

Kích cỡ của quan hệ kết quả (số bộ): $\leq |r|$

Định nghĩa: $r \cap s \equiv r - (r - s)$

Ví dụ: $(\pi_{(p\#)}(SPJ)) \cap (\pi_{(p\#)}(P))$

Phép giao sinh ra một tập các bộ mà chúng xuất hiện ở cả hai quan hệ toán hạng.

Ví dụ của toán tử giao với thể hiện của R và S được thể hiện như dưới đây

R			
A	B	C	D
a	a	yes	1
b	d	no	7
c	f	yes	34
a	d	no	6

r = R ∩ S			
A	B	C	D
a	a	yes	1
c	f	yes	34

r = R ∩ T			
A	B	C	D
a	r	no	31

S			
E	F	G	H
Như chúng ta đã thấy trong các biểu thức truy vấn trước đó có liên quan tới tích Đè các, chúng ta phải cung cấp thêm các phép toán lựa chọn.	yes	yes	34
nhưng chúng không có ý nghĩa giống như kiểu	no	no	56
một vận chuyển của một linh kiện cụ thể nào đó được kết hợp với thông tin của linh kiện	n	no	30

E	F	G	H
a	r	no	31
b	f	yes	30

nhưng những thông tin này không phải của linh kiện cần được vận chuyển). Điều này thường xuyên xảy ra đến mức mà việc kết hợp tích Đề các và các phép toán lựa chọn được phát triển thành một phép toán kết nối (join operation). Có nhiều loại phép toán kết nối khác nhau bao gồm kết nối theta, kết nối bằng, kết nối tự nhiên, kết nối ngoài và nửa kết nối. Chúng ta sẽ xem xét lần lượt mỗi phép kết nối này và các điều kiện được chúng sử dụng.

Phép kết nối Theta và kết nối bằng

Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là $r \bowtie_{(predicate)} s$ trong đó predicate biểu diễn một mệnh đề nào đó.

Lược đồ của quan hệ kết quả là ghép nối các quan hệ toán hạng.

Định nghĩa $r \bowtie_{(predicate)} s \equiv \sigma_{(predicate)}(r \times s)$

Ví dụ: phép kết nối bằng $r \bowtie_{(color='blue' \wedge size=3)} s$ và phép kết nối theta $r \bowtie_{(color='blue' \wedge size>3)} s$. Phép kết nối theta là một cách viết ngắn gọn cho một tích Đề các và một phép chọn được thực hiện sau đó. Phép kết nối bằng là một trường hợp đặc biệt của phép kết nối theta trong đó tất cả các điều kiện trong mệnh đề đều sử dụng điều kiện bằng.

Cả phép toán kết nối bằng và kết nối theta đều không loại bỏ các bộ dư thừa vì vậy việc loại bỏ các bộ dư thừa phải được quản lý một cách tường minh thông qua các mệnh đề điều kiện.

Ví dụ về toán tử kết nối theta được thể hiện như dưới đây với các thể hiện của quan hệ R và S.

R

A	B	C	D
a	a	yes	1
b	d	no	7
c	f	yes	34
a	d	no	6

S

E	F	G	H
a	a	yes	1
b	r	yes	3
c	f	yes	34
m	n	no	56

$r = R \bowtie_{(R.B < S.F)} S$

A	B	C	D	E	F	G	H
a	a	yes	1	b	r	yes	3
a	a	yes	1	c	f	yes	34
a	a	yes	1	m	n	no	56
b	d	no	7	b	r	yes	3
b	d	no	7	c	f	yes	34
b	d	no	7	m	n	no	56
c	f	yes	34	b	r	yes	3
c	f	yes	34	m	n	no	56
a	d	no	6	b	r	yes	3
a	d	no	6	c	f	yes	34
a	d	no	6	m	n	no	56

Phép kết nối tự nhiên

Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là $r * s$.

Lược đồ của quan hệ kết quả là ghép nối các quan hệ toán hạng trong đó các thuộc tính với tên chung chỉ được xuất hiện một lần trong lược đồ.

Định nghĩa $r * s \equiv r \bowtie_{(r.commonattributes=s.commonattributes)} s$

Ví dụ: phép kết nối tự nhiên $s * spj * p$

Phép kết nối tự nhiên thực hiện một phép kết nối bằng trên tất cả các thuộc tính có chung tên trùng nhau của hai quan hệ toán hạng. Cấp của quan hệ kết quả của một kết nối bằng là tổng của cấp quan hệ các toán hạng trừ đi số lượng các thuộc tính chung của hai quan hệ toán hạng. Nói một cách khác, một thể hiện của mỗi thuộc tính chung được lọa bỏ khỏi quan hệ kết quả.

Phép kết nối tự nhiên có lẽ là phép toán phổ biến nhất trong số các dạng phép kết nối. Nó thực sự rất có ích trong việc loại bỏ các bộ dư thừa. Những thuộc tính có chung tên giữa các quan hệ toán hạng thường được gọi với cái tên là các **thuộc tính kết nối**.

Ví dụ về toán tử kết nối tự nhiên được thể hiện như dưới đây với thể hiện của quan hệ R và S

R

A	B	C	D
a	a	yes	1
b	r	no	7
c	f	yes	34
a	m	no	6

$r = R * S$

A	B	C	D	M	G	H
a	a	yes	1	a	yes	1
a	a	yes	1	f	yes	34
a	m	no	6	n	no	56

S

B	M	G	H
a	a	yes	1
b	r	yes	3
a	f	yes	34
m	n	no	56

$r = R * T$

A	B	C	D	G	H
b	r	no	7	yes	30

T

A	B	G	H
a	f	no	31
b	r	yes	30

Phép kết nối ngoài

Là loại phép toán hai ngôi sử dụng ký hiệu và dạng chung với ba loại kết nối ngoài như sau:

Kết nối ngoài bên trái: $r \supset s$

Kết nối ngoài bên phải: $r \subset s$

Kết nối ngoài đầy đủ: $r \supset \supset s$

Lược đồ quan hệ kết quả là sự ghép nối các quan hệ toán hạng.

Định nghĩa:

$r \supset s \equiv$ kết nối tự nhiên của r và s với các bộ của r không tương ứng với s vẫn được giữ lại trong kết quả. Tất cả các giá trị bị khuyết ở thuộc tính của s đều được gán cho giá trị rỗng.

$r \subset s \equiv$ kết nối tự nhiên của r và s với các bộ của s không tương ứng với r vẫn được giữ lại trong kết quả. Tất cả các giá trị bị khuyết ở thuộc tính của r đều được gán cho giá trị rỗng.

$r \triangleright \triangleleft s$ \equiv kết nối tự nhiên của r và s với các bộ từ r và s không tương ứng với nhau vẫn được giữ lại trong kết quả. Tất cả các giá trị bị khuyết ở thuộc tính của r hoặc s đều được gán cho giá trị rỗng.

Ví dụ: Cho $r(A,B) = \{(a, b), (c, d), (b, c)\}$ và cho $s(A,C) = \{(a, d), (s, t), (b, d)\}$

thì $r \triangleright \triangleleft s = (A, B, C) = \{(a, b, d), (b, c, d), (c, d, null)\}$,

$r \triangleright s = (A, B, C) = \{(a, b, d), (b, c, d), (s, null, t)\}$, và

$r \triangleright \triangleleft \underline{s} = (A, B, C) = \{(a, b, d), (b, c, d), (s, null, t), (c, d, null)\}$,

Ví dụ về phép kết nối ngoài thông qua các thể hiện của quan hệ R và S được thể hiện như dưới đây

R

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

$r = R \triangleright \triangleleft S$

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	null
7	8	9	null
null	6	7	12

S

B	C	D
2	3	10
2	3	11
6	7	12

$r = R \triangleright \triangleleft S$

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	null
7	8	9	null

$r = R \triangleright \triangleleft S$

B	C	D	A
2	3	10	1
2	3	11	1
6	7	12	null

Phép nửa kết nối

Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là $r \triangleright \triangleright_{(predicate)} s$ trong đó predicate là một mệnh đề.

Lược đồ của quan hệ kết quả là lược đồ của r.

Định nghĩa $r \triangleright \triangleright_{(predicate)} s \equiv \pi_{(\text{các thuộc tính của } r)} (r \triangleright \triangleleft_{(predicate)} s)$

Phép toán nửa kết nối thực hiện một phép kết nối với hai quan hệ toán hạng và chiếu lên các thuộc tính của quan hệ toán hạng bên trái. Ưu điểm chính của phép toán nửa kết nối là nó sẽ giảm số lượng các bộ cần phải quản lý để hình thành phép kết nối. Điều này thực sự rất có ích trong môi trường phân tán. Trong dạng chung của phép kết nối này, phép nửa kết nối là dạng nửa kết nối theta, nửa kết nối bằng và nửa kết nối tự nhiên có thể được định nghĩa theo một cách thức tương tự.

Ví dụ về toán tử nửa kết nối được thể hiện như sau

R

A	B	C	D
a	a	yes	1
b	r	no	7
c	f	yes	34
a	m	no	6

$r = R \triangleright_{(R.B < S.M)} S$

A	B	C	D
a	a	yes	1
c	f	yes	34
a	M	No	6

S

B	M	C
a	e	yes
b	r	yes
a	f	no
r	n	no

T

B	G	D
a	4	d
b	7	e
a	4	f
m	2	g

$r = S \triangleright_{(S.B = T.B \wedge T.G = 4)} T$

B	M	C
a	e	yes
a	f	no

Phép chia

Là loại phép toán hai ngôi, dùng ký hiệu và dạng chung là $r \div s$ với $r(\{A\})$ và $s(\{B\})$.

Lược đồ quan hệ kết quả: C với $C = A - B$

Định nghĩa: $r \div s \equiv \pi_{(A-B)}(r) - (\pi_{(A-B)}((\pi_{(A-B)}(r) \times s) - r))$

Ví dụ: Cho $r(A,B,C) = \{(a,b,c), (a,d,d), (a,b,d), (a,c,c), (a,d,d)\}$ và $s(C) = \{(c), (d)\}$ thì

$$r \div s = t(A,B) = \{(a,b)\}$$

Yêu cầu cho phép toán chia

- Quan hệ r được xác định trên tập các thuộc tính A và quan hệ s được xác định trên tập thuộc tính B sao cho $B \subseteq A$.
- Cho C là tập các thuộc tính trong $A-B$

Với các ràng buộc này phép toán chia được định nghĩa là: một bộ t là của $r \div s$ nếu cho mỗi bộ t_s trong s có một bộ t_r trong r thỏa mãn cả hai điều kiện sau đây:

$$t_r[C] = t_s[C] \text{ và } t_r[A-B] = t[A-B]$$

Ví dụ về toán tử chia được thể hiện như dưới đây

R			
A	B	C	D
a	f	yes	1
b	r	no	1
a	f	yes	34
e	g	yes	34
a	m	no	6
b	r	no	34

r = R ÷ S		
A	B	C
a	f	yes
b	r	no

r = R ÷ T		
A	B	
a	f	

r = R ÷ V		
A		
a		

r = R ÷ W		
A		
A		

S		
D		
1		
34		

T		
C	D	
yes	1	
yes	34	

U		
C	D	
no	1	
no	34	

V		
B	C	D
f	yes	1
f	yes	34
m	no	6

B		
B	C	D
f	yes	1
g	yes	69

Tính hữu ích của các toán tử dư thừa (mở rộng)

Các toán tử dư thừa của đại số quan hệ sở dĩ là dư thừa bởi vì tất cả chúng đều được định nghĩa dựa trên năm phép toán cơ bản. Tuy nhiên tính hữu ích của chúng được mô tả tốt nhất qua phép toán chia. Xem xét câu truy vấn dựa trên cơ sở dữ liệu linh kiện-công việc-vận chuyển được cho ở đầu bài để thấy được sự hữu ích này.

Cây truy vấn: tìm số hiệu của các nhà cung cấp mà chuyển mọi linh kiện. Một giải pháp cho câu truy vấn này chỉ dùng năm toán tử cơ bản là:

- $T1 = \pi_{(s\#, p\#)}(spj) //$ tất cả các cặp $(s\#, p\#)$ cho các vận chuyển thực tế
- $T2 = \pi_{(p\#)}(p) //$ tất cả $(p\#)$ {tất cả các linh kiện tồn tại cho dù được chuyển hoặc không}
- $T3 = \pi_{(s\#)}(T1) \times T2 //$ tất cả $s\#$ trong $T1$ cặp với mỗi bộ trong $T2$ { $spj.s\#, p.p\#$ }
- $T4 = T3 - T1 //$ tất cả các bộ trong $T3$ mà không trong $T1$
- $T5 = T1 - T4 //$ tất cả các bộ trong $T1$ mà không trong $T4$.
- $T6 = \pi_{(s\#)}(T5) //$ giải pháp

Giải pháp hoàn chỉnh là: $\pi_{(s\#)}(spj) - (\pi_{(s\#)}((\pi_{(s\#)}(spj) \times \pi_{(p\#)}(p)) - \pi_{(s\#, p\#)}(spj)))$

Giải pháp sử dụng các phép toán mở rộng (dư thừa)

$$(\pi_{(s\#, p\#)}(spj)) \div (\pi_{(p\#)}(p))$$

Một số các truy vấn sử dụng chỉ năm phép toán cơ bản

1. Tìm tất cả số hiệu của các nhà cung cấp nằm tại Milan hoặc chuyển hàng tới bất kể công việc nào với số lượng lớn hơn 40

$$[\pi_{(s\#)}(\sigma_{(city = Milan)}(S))] \cup [\pi_{(s\#)}(\sigma_{(qty > 40)}(SPJ))]$$

2. Tìm tất cả mã hiệu của các nhà cung cấp mà chỉ vận chuyển các linh kiện màu đỏ

$$[\pi_{(S.name)}(\sigma_{((SPJ.s\#=S.S\#) \wedge (SPJ.p\#=P.p\#) \wedge (\text{color}=red))}(SPJ \times S \times P))] \\ - [\pi_{(S.name)}(\sigma_{((SPJ.s\#=S.S\#) \wedge (SPJ.p\#=P.p\#) \wedge (\text{color} \neq red))}(SPJ \times S \times P))]$$

3. Tìm tên của các nhà cung cấp cùng thành phố với công việc mà họ vận chuyển các linh kiện đến cho.

$$T1 = (S \times SPJ \times J)$$

$T2 = \sigma_{(S.s\#=SPJ.s\#)}(T1)$ // lựa chọn các bộ có cùng mã nhà cung cấp s#

$T3 = \sigma_{(J.j\#=SPJ.j\#)}(T2)$ // lựa chọn các bộ có cùng mã công việc j#

$T4 = \sigma_{(J.city = S.city)}(T3)$ // lựa chọn các bộ có cùng thành phố

$T5 = \pi_{(S.name)}(T4)$ // chiếu lên tập thuộc tính cuối cùng

4. Tìm số hiệu của tất cả các linh kiện được vận chuyển bởi cả hai nhà cung cấp “S1” và “S2”.

Lưu ý biểu thức sau đây không đúng bởi điều kiện lựa chọn của phép chọn luôn bằng false $\pi_{(p\#)}(\sigma_{((s\#=S1) \wedge (s\#=S2))}(SPJ))$

Biểu thức đúng cho truy vấn này như sau

$$[\pi_{(p\#)}(\sigma_{(s\#=S1)}(SPJ))] - ([\pi_{(p\#)}(\sigma_{(s\#=S1)}(SPJ))] - [\pi_{(p\#)}(\sigma_{(s\#=S2)}(SPJ))])$$

5. Tìm số hiệu các nhà cung cấp mà cung cấp một linh kiện màu xanh và một linh kiện màu đỏ.

Lưu ý biểu thức sau đây là không đúng vì điều kiện lựa chọn nếu viết như vậy không thể hiện được yêu cầu của câu truy vấn và đồng thời giá trị trả về của biểu thức điều kiện luôn là false nên kết quả của truy vấn sẽ là rỗng.

$$\pi_{(s\#)}(\sigma_{((color = blue) \wedge (SPJ.p\#=P.p\#) \wedge (\text{color}=red))}(P \times SPJ))$$

Cách đúng để thực hiện truy vấn đó trong đại số quan hệ như sau

$$T1 = \pi_{(s\#)}(\sigma_{((color = blue) \wedge (SPJ.p\#=P.p\#))}(P \times SPJ))$$

$$T2 = \pi_{(s\#)}(\sigma_{((color = red) \wedge (SPJ.p\#=P.p\#))}(P \times SPJ))$$

$$T3 = T2 - T1$$

$$T4 = T2 - T3$$

6. Tìm tất cả các cặp (s#, j#) của những nhà cung cấp và công việc có cùng thành phố nhưng nhà cung cấp đó không vận chuyển linh kiện nào tới cho công việc này.

$$T1 = \pi_{(s\#, j\#)}(\sigma_{(S.city = J.city)}(S \times J)) // tất cả cặp (s#,j#) ở cùng một thành phố$$

$$T2 = \pi_{(s\#, j\#)}(\sigma_{((S.city = J.city) \wedge (SPJ.j\#=J.j\#) \wedge (SPJ.s\#=S.s\#))}(S \times SPJ \times J))$$

//T2 chứa tất cả những cặp (s#,j#) thể hiện việc vận chuyển linh kiện của các nhà cung cấp tới công việc tương ứng trong cùng thành phố.

$$T3 = T1 - T2$$

Toán tử đặt lại tên

Không giống như các quan hệ cơ bản trong cơ sở dữ liệu, các quan hệ trung gian được sản sinh ra từ các kết quả đánh giá của một truy vấn, không có tên để tham chiếu tới. Trừ khi các quan hệ trung gian này được lưu trữ một cách tường minh, nó sẽ không tồn tại sau khi truy vấn này thực thi xong. Tuy nhiên trong nhiều trường hợp, việc lưu trữ các quan hệ trung gian này lại rất có ích vì nó có thể chứa một tập các bộ là kết quả của các truy vấn liên

quan hoặc nó sẽ chứa đựng một tập các bộ có thể được sử dụng để đánh giá một truy vấn khác. Lưu trữ các quan hệ trung gian đôi khi có nghĩa là một công việc tương tự không cần thiết phải thực hiện lại.

Phép toán đặt tên được biểu diễn bởi một chữ cái Hy lạp thường rho (ρ) và nó được sử dụng để đặt tên cho cả các quan hệ và các thuộc tính. Dạng chung đầu tiên của phép toán đặt tên cho các quan hệ là $\rho_{\text{tên mới}}(\text{quan hệ})$. Ví dụ $\rho_x(r)$ đặt lại tên cho quan hệ r thành x .

Dạng thứ hai cho phép toán đặt lại tên cho cả quan hệ và thuộc tính giả sử rằng quan hệ toán hạng có cấp n $\rho_{\text{tên mới}}(\text{quan hệ} (A_1, A_2, \dots, A_N))$ (quan hệ). Ví dụ $\rho_{x(\text{one}, \text{two}, \dots, \text{last})}(r)$ sẽ đổi tên quan hệ r thành quan hệ x và n thuộc tính của quan hệ x thành $\text{one}, \text{two}, \dots, \text{last}$.

Các truy vấn để thực hành việc sử dụng tất cả các toán tử của đại số quan hệ

- Liệt kê tất cả các cặp số hiệu nhà cung cấp ở cùng một thành phố

$$\pi_{(s.s\#, x.s\#)}(s \triangleright \triangleleft_{(s.\text{city} = x.\text{city})}(\rho_x(s)))$$

- Liệt kê mọi vận chuyển liên quan tới một linh kiện màu xanh

$$spj \triangleright \triangleleft_{(spj.pnum = p.pnum)}(\sigma_{(\text{color} = \text{green})}(p))$$

- Liệt kê số hiệu của tất cả các nhà cung cấp mà vận chuyển một linh kiện được sản xuất ở cùng một thành phố với nhà cung cấp.

$$\pi_{(s.s\#)}(\sigma_{(s.\text{city} = p.\text{city})}(s * p * spj))$$

- Liệt kê tên của các nhà cung cấp mà vận chuyển tất cả các linh kiện màu xanh

$$\pi_{(s.name)}(s * (\pi_{(s\#, p\#)}(spj) \div \pi_{(p\#)}(\sigma_{(\text{color} = \text{blue})}(p))))$$

- Liệt kê số hiệu của các nhà cung cấp mà chỉ vận chuyển các linh kiện màu xanh

$$(\pi_{(s\#)}(spj * (\sigma_{(\text{color} = \text{blue})}(p)))) - (\pi_{(s\#)}(spj * (\sigma_{(\text{color} \neq \text{blue})}(p))))$$

$$(\pi_{(s\#)}(spj \triangleright \triangleleft(\sigma_{(\text{color} = \text{blue})}(p)))) - (\pi_{(s\#)}(spj \triangleright \triangleleft(\sigma_{(\text{color} \neq \text{blue})}(p))))$$

Bài 9: Giới thiệu về Phụ thuộc hàm

Mục đích của sự chuẩn hóa và phụ thuộc hàm

Chuẩn hóa là một kỹ thuật sinh ra một tập các quan hệ với các thuộc tính mong muốn với các yêu cầu cho trước về dữ liệu cần mô hình hóa của một tập đoàn. Quá trình chuẩn hóa đầu tiên được phát triển bởi Codd năm 1972. Việc chuẩn hóa thường được thực hiện như một chuỗi các kiểm tra trên một quan hệ để xác định xem nó có thoả mãn hay vi phạm các yêu cầu của một dạng chuẩn cho trước nào đó.

Codd khởi đầu định nghĩa ba dạng chuẩn hóa có tên gọi là dạng chuẩn một (1NF), chuẩn hai (2NF) và dạng chuẩn 3 (3NF). Boyce và Codd sau đó cùng giới thiệu một chuẩn mạnh hơn 3NF được gọi là chuẩn Boyce-Codd (BCNF) vào năm 1974.

Bốn dạng chuẩn đều dựa trên sự phụ thuộc hàm giữa các thuộc tính của một quan hệ. Một phụ thuộc hàm mô tả mối quan hệ giữa các thuộc tính trong một quan hệ. Ví dụ nếu A và B là các thuộc tính hoặc tập các thuộc tính của quan hệ R, B phụ thuộc hàm vào A (ký hiệu là $A \rightarrow B$) nếu mỗi giá trị của A liên hệ tới duy nhất một giá trị của B.

Năm 1977 và 1979, một dạng chuẩn thứ tư và thứ năm được giới thiệu sau dạng chuẩn BCNF. Tuy nhiên, chúng gặp phải tình huống khá hiếm. Các dạng chuẩn mức cao khác lần lượt được giới thiệu nhưng tất cả các dạng đó đều dựa trên các sự phụ thuộc hơn là dựa trên phụ thuộc hàm.

Một lược đồ quan hệ bao gồm một số các thuộc tính và một lược đồ cơ sở dữ liệu quan hệ bao gồm một số các quan hệ. Các thuộc tính có thể gộp nhóm cùng nhau để tạo thành một lược đồ quan hệ theo quan điểm chung của các nhà thiết kế cơ sở dữ liệu hoặc bằng cách ánh xạ từ mô hình thực thể liên kết sang lược đồ quan hệ. Cho dù dùng kiểu tiếp cận gì, một phương pháp chính thức thường được dùng để giúp cho các nhà thiết kế xác định các nhóm thuộc tính tối ưu cho mỗi quan hệ trong lược đồ cơ sở dữ liệu.

Quá trình chuẩn hóa là một phương pháp chính thống để xác định các quan hệ dựa trên khoá chính và khoá dự bị và các phụ thuộc hàm giữa các thuộc tính của chúng. Chuẩn hóa còn hỗ trợ cho người thiết kế thông qua một chuỗi các kiểm thử mà có thể dùng cho mỗi quan hệ sao cho một lược đồ quan hệ có thể được chuẩn hóa tới một dạng cụ thể nào đó nhằm ngăn chặn các dị thường khi cập nhật có thể xảy ra.

Mục đích chính của việc thiết kế cơ sở dữ liệu quan hệ là gộp nhóm các thuộc tính thành vào các quan hệ để tối thiểu sự dư thừa dữ liệu và vì vậy sẽ giảm không gian lưu trữ tệp cần thiết để cài đặt những quan hệ cơ sở này. Chúng ta cùng xem xét lược đồ quan hệ *staffbranch* trong bảng dưới đây. Quan hệ này chứa một sự dư thừa dữ liệu. Thông tin chi tiết của một branch (chi nhánh ngân hàng) sẽ bị lặp lại cho mỗi thành viên của Staff (nhân viên) làm việc tại chi nhánh đó. Ngược lại với trường hợp các lược đồ quan hệ staff và

branch được tách riêng ra sau đó. Trong trường hợp này, chi tiết về branch chỉ xuất hiện duy nhất một lần cho mỗi chi nhánh.

staffbranch

staff#	sname	position	salary	branch#	baddress
SL21	Kristy	manager	30000	B005	22 Deer Road
SG37	Debi	assistant	12000	B003	162 Main Street
SG14	Alan	supervisor	18000	B003	163 Main Street
SA9	Traci	assistant	12000	B007	375 Fox Avenue
SG5	David	manager	24000	B003	163 Main Street
SL41	Anna	assistant	10000	B005	22 Deer Road

staff

staff#	sname	position	salary	branch#
SL21	Kristy	manager	30000	B005
SG37	Debi	assistant	12000	B003
SG14	Alan	supervisor	18000	B003
SA9	Traci	assistant	12000	B007
SG5	David	manager	24000	B003
SL41	Anna	assistant	10000	B005

branch

branch#	baddress
B005	22 Deer Road
B003	163 Main Street
B007	375 Fox Avenue

Dư thừa dữ liệu và dị thường khi cập nhật

Các quan hệ có dư thừa dữ liệu có thể gây ra vấn đề được gọi là dị thường khi cập nhật, có thể được phân ra làm các loại dị thường khi chèn thêm, khi xoá hoặc khi thay đổi.

Dị thường khi chèn thêm

- Để chèn thêm các thông tin chi tiết cho các nhân viên mới vào quan hệ staffbranch, chúng ta phải thêm vào thông tin chi tiết cho branch tương ứng tại mỗi bản ghi mà nhân viên mới đó được lưu trữ. Ví dụ, nếu nhân viên mới được đưa vào branch B007 thì chúng ta phải thêm cả địa chỉ chính xác của B007. Lược đồ quan hệ thứ hai (mà chứa staff và branch) không rơi vào trường hợp này.
- Để chèn thêm chi tiết cho một branch mới mà hiện tại chưa có nhân viên nào, chúng ta sẽ cần chèn thêm giá trị null cho các thuộc tính của staff như staff number. Tuy nhiên, vì thuộc tính staff number là khoá chính, nó sẽ vi phạm tính toàn vẹn của khoá và sẽ không được phép. Vì vậy, chúng ta không thể nhập thông tin cho một chi nhánh mới mà không có một nhân viên nào.

Dị thường khi xoá bộ.

Nếu chúng ta xoá một bộ từ quan hệ staffbranch thể hiện thành viên cuối cùng của staff tại chi nhánh đó, các chi tiết về chi nhánh đó cũng sẽ bị loại bỏ khỏi cơ sở dữ liệu. Ví dụ, nếu xoá nhân viên tên là Traci khỏi quan hệ staffbranch thì thông tin về chi nhánh B007 cũng sẽ bị xoá đi. Điều này không xảy ra đối với lược đồ cơ sở dữ liệu (staff, branch) bởi vì các thông tin chi tiết về nhân viên được lưu trữ tách riêng khỏi chi tiết về các chi nhánh.

Dị thường khi thay đổi giá trị thuộc tính

Nếu chúng ta muốn thay đổi giá trị của một trong các thuộc tính của một chi nhánh cụ thể nào đó trong quan hệ staffbranch, lấy ví dụ là địa chỉ của một chi nhánh B003, thì sẽ cần cập nhật các bộ của mỗi nhân viên làm việc tại chi nhánh đó.

Nếu việc thay đổi này không được tiến hành trên tất cả các bộ tương ứng của quan hệ staffbranch thì cơ sở dữ liệu sẽ trở nên không thống nhất. Ví dụ chi nhánh B003 sẽ xuất hiện trong cơ sở dữ liệu với hai địa chỉ khác nhau cho hai nhân viên khác nhau.

Các ví dụ của ba loại dị thường cập nhật của quan hệ staffbranch đã chỉ ra được rằng việc phân tách lược đồ cơ sở dữ liệu này thành hai quan hệ staff và branch sẽ tránh được những dị thường này. Khi phân tách một lược đồ thành một tập các lược đồ nhỏ hơn, chúng ta cần quan tâm tới hai thuộc tính quan trọng sau:

1. Thuộc tính kết nối không tồn thất thông tin đảm bảo rằng mọi thể hiện của quan hệ ban đầu có thể được xác định từ các thể hiện liên quan trong các quan hệ nhỏ hơn.
2. Thuộc tính bảo toàn sự phụ thuộc hàm đảm bảo rằng một ràng buộc trên quan hệ ban đầu có thể được bảo tồn bằng cách sử dụng đơn giản một số ràng buộc trên mỗi quan hệ nhỏ hơn. Nói một cách khác, các quan hệ nhỏ hơn không cần kết nối với nhau để kiểm tra xem một ràng buộc trên các quan hệ ban đầu có bị vi phạm hay không.

Thuộc tính kết nối không tồn thất thông tin

Xem xét lược đồ quan hệ SP sau đây và sự phân tách nó thành hai lược đồ S1 và S2.

SP

s#	p#	qty
S1	P1	10
S2	P2	50
S3	P3	10

S1

s#	qty
S1	10
S2	50
S3	10

S2

p#	qty
P1	10
P2	50
P3	10

S1 ▷◁ S2

s#	p#	qty
S1	P1	10
S1	P3	10
S2	P2	10
S3	P1	10
S3	P3	10

Những bộ mới này không xuất hiện trong quan hệ ban đầu. Tuy vậy chúng ta không thể nói bộ nào là hợp lệ hay không hợp lệ. Khi sự phân tách xảy ra quan hệ SP ban đầu có thể bị mất.

Bảo toàn các phụ thuộc hàm

Ví dụ $R = (A, B, C)$ với $F = \{AB \rightarrow C, C \rightarrow A\}$ và $\gamma = \{(B, C), (A, C)\}$.

Rõ ràng là $C \rightarrow A$ được bảo toàn trong lược đồ quan hệ (A,C). Làm thế nào để $AB \rightarrow C$ được bảo toàn mà không cần kết nối hai lược đồ quan hệ trong γ . Câu trả lời là không thể vì thế các phụ thuộc hàm không được bảo toàn trong γ .

Định nghĩa phụ thuộc hàm

Với toàn bộ những bàn luận về phụ thuộc hàm trong phần này, chúng ta giả thiết rằng một lược đồ quan hệ có các thuộc tính (A, B, C, ..., Z) và toàn bộ cơ sở dữ liệu được mô tả bởi một quan hệ $R = (A, B, C, \dots, Z)$. Giả thiết này có nghĩa là mỗi thuộc tính trong cơ sở dữ liệu có một tên duy nhất.

Một phụ thuộc hàm là một tài sản về ngữ nghĩa của các thuộc tính trong một quan hệ. Các ngữ nghĩa này thể hiện một thuộc tính có quan hệ với thuộc tính khác như thế nào và xác định các phụ thuộc hàm giữa các thuộc tính đó. Khi một phụ thuộc hàm xuất hiện, sự phụ thuộc này được cụ thể hóa như một ràng buộc giữa các thuộc tính.

Xem xét một quan hệ với các thuộc tính A và B với thuộc tính B là phụ thuộc hàm vào thuộc tính A. Nếu chúng ta biết giá trị của A và kiểm tra quan hệ phụ thuộc này, chúng ta sẽ tìm thấy chỉ một giá trị duy nhất của B trong số tất cả các bộ mà chứa giá trị cho trước đó

của A, tại mọi thời điểm. Lưu ý với mỗi một giá trị cho trước của B có thể có nhiều giá trị khác nhau tương ứng của A.



Đối tượng xác định của một phụ thuộc hàm là thuộc tính hoặc một nhóm các thuộc tính nằm bên trái của mũi tên trong một phụ thuộc hàm. Đối tượng (thuộc tính) hệ quả của một phụ thuộc hàm là một thuộc tính hoặc một nhóm các thuộc tính nằm bên phải mũi tên trong một phụ thuộc hàm. Trong hình vẽ mô tả một phụ thuộc hàm ở trên, A là đối tượng xác định và B là đối tượng hệ quả của A. Ta nói rằng A xác định B hay B phụ thuộc hàm vào A.

Xác định các phụ thuộc hàm

Quay lại ví dụ về quan hệ nhân viên trình bày trong phần trước. Thể hiện của quan hệ này có một phụ thuộc hàm staff# → position. Tuy nhiên, phụ thuộc theo chiều ngược lại rõ ràng là không thể. Mỗi quan hệ giữa staff# và position là quan hệ 1:1 có nghĩa là mỗi một nhân viên có một vị trí tương ứng duy nhất. Nói một cách khác, quan hệ giữa position và staff# là 1:nhiều có nghĩa là có nhiều nhân viên có cùng một vị trí. Hay position không xác định staff# hoặc staff# không phụ thuộc hàm vào position.

Với mục đích chuẩn hóa chúng ta chỉ quan tâm tới việc xác định các phụ thuộc hamg giữa các thuộc tính của một quan hệ loại 1:1.

Khi xác định các phụ thuộc hàm giữa các thuộc tính trong một quan hệ, việc phân biệt rõ ràng giữa các giá trị được nhận bởi một thuộc tính tại một thời điểm nào đó và tập tất cả các giá trị có thể nhận được bởi thuộc tính đó tại các thời điểm khác nhau là rất quan trọng. Nói một cách khác, một phụ thuộc hàm là một đặc điểm chung của một lược đồ quan hệ chứ không phải là một đặc điểm riêng của một thể hiện cụ thể của lược đồ đó.

Lý do mà chúng ta cần xác định các phụ thuộc hàm thỏa mãn cho tất cả các giá trị có thể của các thuộc tính của một quan hệ là vì điều đó thể hiện các loại ràng buộc toàn vẹn mà chúng ta cần xác định. Những ràng buộc loại đó thể hiện sự hạn chế trên các giá trị mà một quan hệ có thể đưa ra một cách hợp lệ. Nói cách khác, chúng xác định những thể hiện hợp lệ có thể của một quan hệ.

Quay lại ví dụ trước về quan hệ staffbranch để xác định các phụ thuộc hàm trong lược đồ quan hệ này. Để xác định các phụ thuộc hàm không thay đổi theo thời gian, chúng ta cần hiểu rõ ngữ nghĩa của các thuộc tính khác nhau trong mỗi lược đồ quan hệ. Ví dụ, nếu chúng ta biết rằng vị trí của mỗi nhân viên và chi nhánh của họ sẽ xác định lương của họ. Không có cách nào để biết được ràng buộc kiểu này trừ khi bạn đã quá quen thuộc với tổ chức này, đây cũng chính là nhiệm vụ của giai đoạn phân tích yêu cầu của bài toán và giai đoạn thiieeys kề mức khái niệm.

$\text{staff\#} \rightarrow \text{sname, position, salary, branch\#, baddress}$

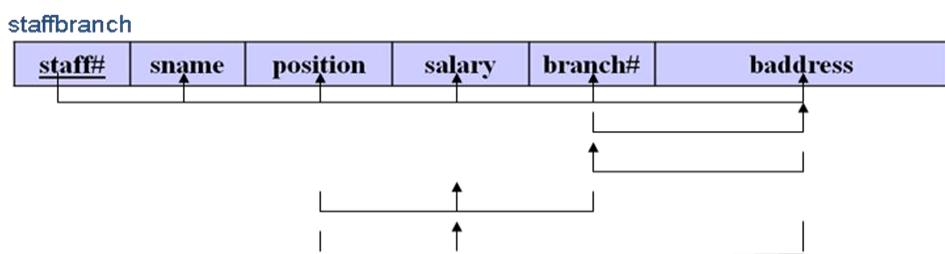
$\text{branch\#} \rightarrow \text{baddress}$

$\text{baddress} \rightarrow \text{branch\#}$

$\text{branch\#, position} \rightarrow \text{salary}$

$\text{baddress, position} \rightarrow \text{salary}$

Thông thường trong nhiều quyển sách giáo trình cho môn học này, một ký pháp lược đồ được sử dụng để biểu diễn các phụ thuộc hàm. Một ví dụ của ký pháp này cho lược đồ quan hệ *staffbranch* với các phụ thuộc hàm được xác định ở trên của quan hệ này được mô tả dưới đây



Phụ thuộc hàm hiển nhiên

Khi xác định các phụ thuộc hàm thể hiện trong tất cả các giá trị có thể của các thuộc tính liên quan tới phụ thuộc hàm đó, chúng ta cũng muốn bỏ qua tất cả các phụ thuộc hàm hiển nhiên đúng. Một phụ thuộc hàm được cho là hiển nhiên nếu và chỉ nếu về phái của phụ thuộc hàm là một tập con của vé trái. Nói một cách khác, phụ thuộc hàm đó đương nhiên đúng, không thể không đúng. Ví dụ: sử dụng lược đồ quan hệ trước, các phụ thuộc đương nhiên bao gồm:

$$\{ \text{staff\#, sname} \} \rightarrow \text{sname}$$

$$\{ \text{staff\#, sname} \} \rightarrow \text{staff\#}$$

Mặc dù các phụ thuộc hàm hiển nhiên luôn đúng, nhưng chúng không cung cấp thêm các thông tin cần thiết nào về các ràng buộc toàn vẹn đối với quan hệ. Vì thế với phạm vi của việc chuẩn hóa, các phụ thuộc hàm này được bỏ qua.

Tóm tắt về các đặc tính của phụ thuộc hàm

Các đặc tính chính của các phụ thuộc hàm mà có ích cho việc chuẩn hóa là

- 1- Tồn tại một mối quan hệ 1-1 giữa các thuộc tính trong đối tượng xác định và đối tượng hệ quả.

- 2- Phụ thuộc hàm là bất biến theo thời gian, có nghĩa là nó có thỏa mãn trong tất cả các thể hiện có thể của một quan hệ.
- 3- Các phụ thuộc hàm là không hiển nhiên. Tất cả các phụ thuộc hàm hiển nhiên đúng đều được bỏ qua.

Các luật suy diễn cho các phụ thuộc hàm

Chúng ta sẽ gọi F là tập các phụ thuộc hàm mà được xác định trên một lược đồ quan hệ R. Thường thì người thiết kế lược đồ xác định cụ thể các phụ thuộc hàm hiển nhiên về mặt ngữ nghĩa, tuy nhiên thường thì có nhiều các phụ thuộc hàm khác cũng thỏa mãn với các thể hiện quan hệ mà đã thỏa mãn các phụ thuộc hàm trong F. Những phụ thuộc hàm phát sinh này là các phụ thuộc hàm được suy diễn hoặc rút ra từ các phụ thuộc hàm trong F.

Tập của tất cả các phụ thuộc hàm được suy diễn ra từ một tập phụ thuộc hàm F được gọi là bao đóng của F và được ký hiệu là F^+ .

Ký pháp: $F \models X \rightarrow Y$ thể hiện rằng phụ thuộc hàm $X \rightarrow Y$ được suy diễn ra bởi tập phụ thuộc hàm F. Một cách chính thức, $F^+ \equiv \{X \rightarrow Y \mid F \models X \rightarrow Y\}$

Một tập các luật suy diễn là cần thiết để suy diễn ra tập các phụ thuộc hàm trong F^+ . Ví dụ, nếu Kristi già hơn Debi và Debi già hơn Traci thì bạn có thể suy diễn ra Kristi già hơn Traci. Bạn đã thực hiện việc suy diễn này như thế nào? Bạn chỉ cần sử dụng một luật bắc cầu để thực hiện suy diễn này mà không cần phải động não một chút nào.

Tập sáu luật suy diễn được biết đến nhiều nhất để áp dụng cho các phụ thuộc hàm như sau:

Luật suy diễn IR1: luật phản xạ - nếu $X \sqsupseteq Y$, thì $X \rightarrow Y$

Luật suy diễn IR2: luật tăng trưởng – nếu $X \rightarrow Y$, thì $XZ \rightarrow YZ$

Luật suy diễn IR3: luật bắc cầu – nếu $X \rightarrow Y$ và $Y \rightarrow Z$, thì $X \rightarrow Z$

Luật suy diễn IR4: luật chiếu – nếu $X \rightarrow YZ$, thì $X \rightarrow Y$ và $X \rightarrow Z$

Luật suy diễn IR5: luật cộng thêm – nếu $X \rightarrow Y$ và $X \rightarrow Z$, thì $X \rightarrow YZ$

Luật suy diễn IR6: luật giả bắc cầu – nếu $X \rightarrow Y$ và $YZ \rightarrow W$, thì $XZ \rightarrow W$

Ba luật suy diễn đầu tiên được biết đến như tiên đề Armstrong và đóng vai trò là một tập luật cần thiết và đầy đủ cho việc tạo ra bao đóng của một tập các phụ thuộc hàm.

Một ví dụ về việc chứng minh sử dụng các luật suy diễn

Cho một lược đồ quan hệ $R = (A, B, C, D, E, F, G, H, I, J)$ và $F = \{AB \rightarrow E, AG \rightarrow J, BE \rightarrow I, E \rightarrow G, GI \rightarrow H\}$. Hỏi $F \models AB \rightarrow GH$?

Chứng minh

1. $AB \rightarrow E$, được cho trong F
2. $AB \rightarrow AB$, dùng luật phản xạ IR1
3. $AB \rightarrow B$, dùng luật chiếu IR4 từ bước 2
4. $AB \rightarrow BE$, dùng luật cộng thêm IR5 từ các bước 1 và 3
5. $BE \rightarrow I$, được cho sẵn trong F
6. $AB \rightarrow I$, dùng luật IR3 từ bước 4 và 5
7. $E \rightarrow G$, cho sẵn trong F
8. $AB \rightarrow G$, dùng luật bắc cầu IR3 từ bước 1 và 7
9. $AB \rightarrow GI$, dùng luật cộng thêm IR5 từ bước 6 và 8
10. $GI \rightarrow H$, cho sẵn trong F
11. $AB \rightarrow H$, dùng luật bắc cầu IR3 từ bước 9 và 10
12. $AB \rightarrow GH$, dùng luật cộng thêm IR5 từ bước 8 và 11- kết quả được chứng minh

Bài thực hành: Dùng cùng tập F, chứng minh rằng $F \vDash BE \rightarrow H$

Tính toán bao đóng

Định nghĩa bao đóng

Một cách nhìn nhận khác về bao đóng của một tập các phụ thuộc hàm F là: F^+ là tập phụ thuộc hàm nhỏ nhất chứa F mà hệ tiên đề Armstrong không thể được áp dụng để sinh ra một phụ thuộc hàm mà không thuộc tập này (hay mọi phụ thuộc hàm được sinh ra nhờ hệ tiên đề Armstrong đều thuộc tập phụ thuộc hàm này).

Bao đóng của F là hữu hạn nhưng có kích cỡ tăng theo cấp số nhân so với số thuộc tính của R. Ví dụ, cho một lược đồ quan hệ $R = (A, B, C)$ và $F = \{AB \rightarrow C, C \rightarrow B\}$, F^+ sẽ bao gồm 29 phụ thuộc hàm (kể cả các phụ thuộc hàm hiển nhiên). Vì vậy, để xác định liệu một phụ thuộc hàm $X \rightarrow Y$ có thỏa mãn trong lược đồ quan hệ R với tập phụ thuộc hàm F hay không, chúng ta cần xác định xem liệu $F \vDash X \rightarrow Y$ không hoặc chính xác hơn là $X \rightarrow Y$ có thuộc F^+ hay không? Tuy nhiên, chúng ta muốn thực hiện việc này mà không cần sinh ra tất cả các thành phần của bao đóng mà vẫn kiểm tra được liệu $X \rightarrow Y$ có nằm trong tập đó không.

Một kỹ thuật để thực hiện điều này là sinh ra bao đóng của tập thuộc tính X là X^+ chứ không phải bao đóng của F trong đó X là tập xác định của một phụ thuộc hàm trong F. Một thuật toán để sinh ra X^+ được thể hiện như sau. X^+ được gọi là bao đóng của X trên tập phụ thuộc hàm F.

Thuật toán Closure {trả về X^+ trên F}

Đầu vào: tập các thuộc tính X, và một tập các phụ thuộc hàm F

Đầu ra: X^+ trên F

Closure (X, F)

{

$X^+ \leftarrow X;$

repeat

$\text{old}X^+ \leftarrow X^+;$

 for mỗi phụ thuộc hàm $W \rightarrow Z$ trong F do

 if $W \subseteq X^+$ then $X^+ \leftarrow X^+ \cup Z;$

 until ($\text{old}X^+ = X^+$);

}

Ví dụ sử dụng thuật toán tính Bao đóng Closure

Cho $F = \{A \rightarrow D, AB \rightarrow E, BI \rightarrow E, CD \rightarrow I, E \rightarrow C\}$, Tìm $(AE)^+$

vòng 1

$X^+ = \{A, E\}$

dùng $A \rightarrow D$, $A \subseteq X^+$, vì vậy thêm D vào X^+ , $X^+ = \{A, E, D\}$

dùng $AB \rightarrow E$, không

dùng $BI \rightarrow E$, không

dùng $CD \rightarrow I$, không

dùng $E \rightarrow C$, $E \subseteq X^+$, vì vậy thêm C vào X^+ , $X^+ = \{A, E, D, C\}$

Có thay đổi xảy ra với X^+ vì vậy cần một vòng lặp nữa

Vòng 2

$X^+ = \{A, E, D, C\}$

dùng $A \rightarrow D$, có nhưng không có thay đổi

dùng $AB \rightarrow E$, không

dùng $BI \rightarrow E$, không

dùng $CD \rightarrow I$, $CD \subseteq X^+$, vì vậy thêm I vào X^+ , $X^+ = \{A, E, D, C, I\}$

dùng $E \rightarrow C$, có nhưng không có thay đổi

có thay đổi xảy ra với X^+ vì vậy cần một vòng lặp nữa

Vòng 3

$X^+ = \{A, E, D, C, I\}$

dùng $A \rightarrow D$, có nhưng không có thay đổi

dùng $AB \rightarrow E$, không

dùng $BI \rightarrow E$, không

dùng $CD \rightarrow I$, có nhưng không có thay đổi

dùng $E \rightarrow C$, có nhưng không có thay đổi

Không có thay đổi xảy ra với X^+ vì vậy thuật toán dừng.

Vậy $(AE)^+ = \{A, E, C, D, I\}$

Điều này có nghĩa là các phụ thuộc hàm sau sẽ thuộc F^+ : $AE \rightarrow AECDI$.

Khi bao đóng của một tập các thuộc tính X được tạo ra, nó trở thành một phép thử đơn giản để có thể nói rằng một phụ thuộc hàm nào đó với về trái là X có thuộc F^+ hay không. Thuật toán **Member** sau đây sẽ xác định xem một tập các phụ thuộc hàm F nào đó có suy diễn ra một phụ thuộc hàm cụ thể hay không

Thuật toán Member {xác định các thành viên của F^+ }

Đầu vào: một tập các phụ thuộc hàm F , và một phụ thuộc hàm $X \rightarrow Y$

Đầu ra: true nếu $F \models X \rightarrow Y$, false nếu trái lại

Member ($F, X \rightarrow Y$)

{

 if $Y \subseteq \text{Closure}(X, F)$

 then trả về true;

 else trả về false;

}

Phủ và sự tương đương của tập phụ thuộc hàm

Một tập các phụ thuộc hàm F được phủ bởi một tập các phụ thuộc hàm G (hay nói một cách khác là G phủ F) nếu mọi phụ thuộc hàm trong G đều nằm trong F^+ . Có thể nói rằng F được phủ nếu mọi phụ thuộc hàm trong F có thể được suy diễn từ G.

Hai tập phụ thuộc hàm F và G là tương đương nếu $F^+ = G^+$. Có thể nói rằng mọi phụ thuộc hàm trong G có thể được suy diễn từ F và mọi phụ thuộc hàm trong F có thể được suy diễn từ G. Để xác định xem liệu G có phủ F hay không, ta tính X^+ trên G cho mỗi phụ thuộc hàm $X \rightarrow Y$ trong F. Nếu $Y \subseteq X^+$ cho mỗi X thì G phủ F.

Phần này chúng ta bàn luận đến lý do tại sao cần khái niệm Phủ tập hợp. Thuật toán Member có thời gian chạy chương trình phụ thuộc vào kích cỡ của tập phụ thuộc hàm được sử dụng như đầu vào cho thuật toán. Vì vậy, tập phụ thuộc hàm đó càng nhỏ thì việc thực thi thuật toán càng nhanh.

Số phụ thuộc hàm càng ít đòi hỏi càng ít không gian nhớ vì vậy chi phí duy trì bảo dưỡng càng thấp khi việc cập nhật cơ sở dữ liệu xảy ra. Nhiều loại phủ tồn tại từ loại phủ không dư thừa đến các phủ tối thiểu. Chúng ta sẽ không xem xét tất cả các loại trong chúng. Ý tưởng chính là sinh ra một tập các phụ thuộc hàm G tương đương với tập ban đầu F nhưng lại có số lượng phụ thuộc hàm càng ít càng tốt (biểu tượng cho các trường hợp tối thiểu).

Phủ không dư thừa

Một tập các phụ thuộc hàm F được cho là không dư thừa nếu không có tập con G thực sự nào của F mà G tương đương với F. Nếu tồn tại một tập G như vậy thì F được gọi là dư thừa.

F là một phủ không dư thừa của G nếu F là một phủ của G và F không dư thừa.

Thuật toán Nonredundant {sinh ra một phủ không dư thừa}

Đầu vào: một tập các phụ thuộc hàm G

Đầu ra: một phủ không dư thừa của G

Nonredundant (G)

{

$F \leftarrow G;$

 for mỗi phụ thuộc hàm $X \rightarrow Y \in G$ do

 if Member($F - \{X \rightarrow Y\}$, $X \rightarrow Y$)

 then $F \leftarrow F - \{X \rightarrow Y\};$

 return (F);

}

Ví dụ về việc sinh ra một phủ không dư thừa

Cho $G = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C\}$, hãy tìm một phủ không dư thừa của G .

$F \leftarrow G$

Member($\{B \rightarrow A, B \rightarrow C, A \rightarrow C\}, A \rightarrow B$)

Closure($A, \{B \rightarrow A, B \rightarrow C, A \rightarrow C\}$)

$A^+ = \{A, C\}$, vì vậy $A \rightarrow B$ là không dư thừa

Member($\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}, B \rightarrow A$)

Closure($B, \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$)

$B^+ = \{B, C\}$, vì vậy $B \rightarrow A$ là không dư thừa

Member($\{A \rightarrow B, B \rightarrow A, A \rightarrow C\}, B \rightarrow C$)

Closure($B, \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$)

$B^+ = \{B, A, C\}$, vì vậy $B \rightarrow C$ là dư thừa $F = F - \{B \rightarrow C\}$

Member($\{A \rightarrow B, B \rightarrow A\}, A \rightarrow C$)

Closure($A, \{A \rightarrow B, B \rightarrow A\}$)

$A^+ = \{A, B\}$, vì vậy $A \rightarrow C$ là không dư thừa

Return $F = \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$

Nếu $G = \{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C\}$, giống hệt tập như trước nhưng với trật tự các phủ thuộc hàm khác. Một phủ khác sẽ được sinh ra!

$F \leftarrow G$

Member($\{A \rightarrow C, B \rightarrow A, B \rightarrow C\}, A \rightarrow B$)

Closure($A, \{A \rightarrow C, B \rightarrow A, B \rightarrow C\}$)

$A^+ = \{A, C\}$, vì vậy $A \rightarrow B$ là không dư thừa

Member($\{A \rightarrow B, B \rightarrow A, B \rightarrow C\}, A \rightarrow C$)

Closure($A, \{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$)

$A^+ = \{A, B, C\}$, vì vậy $A \rightarrow C$ là dư thừa $F = F - \{A \rightarrow C\}$

Member($\{A \rightarrow B, B \rightarrow C\}, B \rightarrow A$)

Closure($B, \{A \rightarrow B, B \rightarrow C\}$)

$B^+ = \{B, C\}$, vì vậy $B \rightarrow A$ là không dư thừa

Member($\{A \rightarrow B, B \rightarrow A\}, B \rightarrow C$)

Closure($B, \{A \rightarrow B, B \rightarrow A\}$)

$$B^+ = \{B, A\}, \text{ vì vậy } B \rightarrow C \text{ là không dư thừa}$$

Return $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$

Ví dụ trên thể hiện rằng một tập các phụ thuộc hàm cho trước có thể có nhiều hơn một phủ không dư thừa. Cũng có những trường hợp trong đó các phủ không dư thừa của một tập các phụ thuộc hàm G chứa những phụ thuộc hàm không nằm trong G . Ví dụ: nếu $G = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C\}$ thì $F = \{A \rightarrow B, B \rightarrow A, AB \rightarrow C\}$ là một phủ không dư thừa của G tuy nhiên F chứa những phụ thuộc hàm không thuộc G .

Các thuộc tính dư thừa

Nếu F là một tập các phụ thuộc hàm không dư thừa, điều này có nghĩa là không có phụ thuộc hàm dư thừa nào trong F và vì vậy F không thể nhỏ hơn bằng cách loại bỏ các phụ thuộc hàm. Nếu các phụ thuộc hàm được loại bỏ khỏi F thì một tập G sẽ được sinh ra và G không tương đương với F . Tuy nhiên có một cách để giảm kích cỡ toàn bộ tập F bằng cách loại bỏ các thuộc tính từ các phụ thuộc hàm trong F .

Nếu F là tập các phụ thuộc hàm trên lược đồ quan hệ R và $X \rightarrow Y \in F$ thì thuộc tính A được gọi là dư thừa trong $X \rightarrow Y \in F$ nếu:

1. $X = AZ, X \neq Z$ và $\{F - \{X \rightarrow Y\}\} \cup \{Z \rightarrow Y\} \equiv F$, hoặc
2. $Y = AW, Y \neq W$ và $\{F - \{X \rightarrow Y\}\} \cup \{X \rightarrow W\} \equiv F$

Nói một cách khác, một thuộc tính A là dư thừa trong $X \rightarrow Y$ nếu A có thể được loại bỏ khỏi vế trái hoặc vế phải của phụ thuộc hàm mà không làm thay đổi F^+ .

Ví dụ: cho $F = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow D\}$, thuộc tính C là dư thừa trong vế phải của $A \rightarrow BC$ vì $A^+ = \{A, B, C, D\}$ khi $F = F - \{A \rightarrow C\}$. Tương tự như vậy, B là thuộc tính dư thừa của vế trái của $AB \rightarrow D$ vì $AB^+ = \{A, B, C, D\}$ khi $F = F - \{AB \rightarrow D\}$.

Tập phụ thuộc hàm tối giản trái và tối giản phải

Cho F là một tập các phụ thuộc hàm trên lược đồ R và cho $X \rightarrow Y \in F$.

$X \rightarrow Y$ được gọi là tối giản trái nếu X không chứa các thuộc tính dư thừa A . Một phụ thuộc hàm tối giản vế trái cũng được gọi là một phụ thuộc hàm đầy đủ.

$X \rightarrow Y$ được gọi là tối giản phải nếu Y không chứa các thuộc tính dư thừa A .

$X \rightarrow Y$ được gọi là tối giản nếu nó tối giản trái và tối giản phải và Y khác rỗng.

Thuật toán tối giản vế trái dưới đây sẽ sinh ra một tập các phụ thuộc hàm tối giản vế trái

Thuật toán Left-Reduce {trả về phiên bản tối giản vế trái của F }

Đầu vào: tập phụ thuộc hàm G

Đầu ra: một phủ tối giản trái của G

Left-Reduce (G)

{

$F \leftarrow G;$

 for mỗi phụ thuộc hàm $X \rightarrow Y$ trong G do

 for mỗi thuộc tính A trong X do

 if Member($F, (X-A) \rightarrow Y$)

 then loại bỏ A khỏi X trong $X \rightarrow Y$ của F

 return(F);

}

Thuật toán tối giản về phải dưới đây sẽ sinh ra một tập các phụ thuộc hàm tối giản về phải.

Thuật toán Right-Reduce {trả về một bản tối giản về phải của F }

Đầu vào: tập các phụ thuộc hàm G

Đầu ra: một phủ tối giản phải của G

Right-Reduce (G)

{

$F \leftarrow G;$

 For mỗi phụ thuộc hàm $X \rightarrow Y$ trong G do

 for mỗi thuộc tính A trong Y do

 if Member($F - \{X \rightarrow Y\} \cup \{X \rightarrow (Y-A)\}, X \rightarrow A$)

 then loại bỏ A khỏi Y từ $X \rightarrow Y$ trong F

 return(F);

}

Thuật toán tối sau đây sinh ra một tập các phụ thuộc hàm tối giản

Thuật toán Reduce {trả về phiên bản tối giản của F }

Đầu vào: tập các phụ thuộc hàm G

Đầu ra: một phủ tối giản của G

Reduce (G)

{

$F \leftarrow \text{Right-Reduce}(\text{Left-Reduce}(G));$

Loại bỏ tất cả các phụ thuộc hàm có dạng $X \rightarrow \text{null}$ từ F // nếu G chứa một phụ thuộc hàm dư thừa $X \rightarrow Y$ thì mọi thuộc tính trong Y sẽ là dư thừa vì vậy sẽ tối giản tới $X \rightarrow \text{null}$, vì vậy những phụ thuộc hàm loại này cần phải loại bỏ.

```
return(F);  
}
```

Trật tự thuật toán thực hiện tối giản rất quan trọng. Tập các phụ thuộc hàm phải được tối giản về trái trước rồi mới tối giản về phải. Ví dụ sau mô tả những gì xảy ra nếu trật tự này bị vi phạm.

Ví dụ: Cho $G = \{B \rightarrow A, D \rightarrow A, BA \rightarrow D\}$, G là tối giản phải nhưng không phải tối giản trái. Nếu chúng ta tối giản trái G để sinh ra $F = \{B \rightarrow A, D \rightarrow A, B \rightarrow D\}$, chúng ta có F sẽ tối giản trái nhưng không tối giản phải. $B \rightarrow A$ là dư thừa bên về phải vì $B \rightarrow D \rightarrow A$.

Phủ tối thiểu

Định nghĩa Phủ tối thiểu: Một tập phụ thuộc hàm F là tối thiểu nếu

1. Mọi phụ thuộc hàm đều có về phải là một thuộc tính
2. F là không dư thừa
3. Không có phụ thuộc hàm nào dạng $X \rightarrow A$ có thể được thay thế bởi một trong dạng $Y \rightarrow A$ với $Y \subseteq X$ và vẫn là một tập tương đương, nói cách khác F là tối giản trái.

Ví dụ: $G = \{A \rightarrow BCE, AB \rightarrow DE, BI \rightarrow J\}$ thì một phủ tối thiểu của G là $F = \{A \rightarrow B, A \rightarrow C, A \rightarrow D, A \rightarrow E, BI \rightarrow J\}$

Thuật toán tìm phủ tối thiểu của một tập phụ thuộc hàm được thể hiện như dưới đây

Thuật toán MinCover {trả về phủ tối thiểu của F }

Đầu vào: tập phụ thuộc hàm F

Đầu ra: một phủ tối thiểu của F

MinCover (F)

```
{
```

```
    G ← F;
```

Thay thế mỗi phụ thuộc hàm $X \rightarrow A_1A_2...A_n$ trong G bởi n phụ thuộc hàm $X \rightarrow A_1, X \rightarrow A_2, ..., X \rightarrow A_n$

```
    Left-Reduce(G);
```

```
    Nonredundant(G);
```

```
    return(G);
```

```
}
```

Lưu ý là trong thuật toán tìm phủ tối thiểu trình bày ở trên, hai bước Left-Reduce(G) và Nonredundant(G) không đổi trật tự được cho nhau vì sau khi thực hiện Left-Reduce(G) vẫn có thể sinh ra phụ thuộc hàm dư thừa.

Bài 10: Giới thiệu về chuẩn hóa

Chuẩn hóa dựa trên khoá chính

Xác định khoá cho một lược đồ quan hệ

- Nếu R là một lược đồ quan hệ với các thuộc tính A_1, A_2, \dots, A_n và một tập các phụ thuộc hàm F trong đó $X \subseteq \{A_1, A_2, \dots, A_n\}$ thì X là một khoá của R nếu:
 - $X \rightarrow A_1 A_2 \dots A_n \in F^+$, và
 - không có tập con thực sự nào $Y \subseteq X$ mà $Y \rightarrow A_1 A_2 \dots A_n \in F^+$.

về cơ bản mà nói, định nghĩa này cho thấy bạn phải tạo ra bao đóng của tất cả các tập con có thể có của R và quyết định xem tập nào suy diễn được ra tất cả các thuộc tính của lược đồ.

Một ví dụ về việc xác định khoá:

cho một quan hệ $r = (C, T, H, R, S, G)$ với tập phụ thuộc hàm $F = \{C \rightarrow T, HR \rightarrow C, HT \rightarrow R, CS \rightarrow G, HS \rightarrow R\}$

Bước 1: Tạo (A_i^+) với $1 \leq i \leq n$

$$\begin{aligned} C^+ &= \{CT\}, & T^+ &= \{T\}, & H^+ &= \{H\} \\ R^+ &= \{R\}, & S^+ &= \{S\}, & G^+ &= \{G\} \end{aligned}$$

Nhìn vào kết quả cho thấy không có thuộc tính đơn nào là khoá cho r

Bước 2: Tạo $(A_i^+ A_j^+)$ với $1 \leq i \leq n, 1 \leq j \leq n$

$$\begin{aligned} (CT)^+ &= \{C, T\}, & (CH)^+ &= \{CHTR\}, & (CR)^+ &= \{CRT\} \\ (CS)^+ &= \{CSGT\}, & (CG)^+ &= \{CGT\}, & (TH)^+ &= \{THRC\} \\ (TR)^+ &= \{TR\}, & (TS)^+ &= \{TS\}, & (TG)^+ &= \{TG\} \\ (HR)^+ &= \{HRCT\}, & (HS)^+ &= \{HSRCTG\}, & (HG)^+ &= \{HG\} \\ (RS)^+ &= \{RS\}, & (RG)^+ &= \{RG\}, & (SG)^+ &= \{SG\} \end{aligned}$$

Tập thuộc tính (HS) là khoá của R

Bước 3: Tạo $(A_i A_j A_k)^+$ với $1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n$

$(CTH)_+ = \{CTHR\}$, $(CTR)_+ = \{CTR\}$
 $(CTS)_+ = \{CTSG\}$, $(CTG)_+ = \{CTG\}$
 $(CHR)_+ = \{CHRT\}$, $(CHS)_+ = \{CHSTRG\}$
 $(CHG)_+ = \{CHGTR\}$, $(CRS)_+ = \{CRSTG\}$
 $(CRG)_+ = \{CRGT\}$, $(CSG)_+ = \{CSGT\}$
 $(THR)_+ = \{THRC\}$, $(THS)_+ = \{THSRCG\}$
 $(THG)_+ = \{THGRC\}$, $(TRS)_+ = \{TRS\}$
 $(TRG)_+ = \{TRG\}$, $(TSG)_+ = \{TSG\}$
 $(HRS)_+ = \{HRSCTG\}$, $(HRG)_+ = \{HRGCT\}$
 $(HSG)_+ = \{HSGRCT\}$, $(RSG)_+ = \{RSG\}$
 các siêu khoá được thể hiện bằng màu đỏ.

Bước 4: Tạo $(A_i A_j A_k A_r)^+$ với $1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n, 1 \leq r \leq n$

$(CTHR)_+ = \{CTHR\}$, $(CHS)_+ = \{CHSRG\}$
 $(CTHG)_+ = \{CTHGR\}$, $(CRS)_+ = \{CHRSTG\}$
 $(CHRG)_+ = \{CHRG\}$, $(CRSG)_+ = \{CRSGT\}$
 $(THRS)_+ = \{THRSCG\}$, $(THR)_+ = \{THRGC\}$
 $(TRSG)_+ = \{TRSG\}$, $(HRSG)_+ = \{HRSGCT\}$
 $(CTRS)_+ = \{CTRS\}$, $(CTSG)_+ = \{CTSG\}$
 $(CSHG)_+ = \{CSHGTR\}$, $(THSG)_+ = \{THSGRC\}$
 $(CTR)_+ = \{CTR\}$

Siêu khoá được thể hiện bằng màu đỏ.

Bước 5: Tạo $(A_i A_j A_k A_r A_s)^+$ với $1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n, 1 \leq r \leq n, 1 \leq s \leq n$

$(CTHRS)_+ = \{CTHSRG\}$
 $(CTHRG)_+ = \{CTHGR\}$
 $(CTHSG)_+ = \{CTHSGR\}$
 $(CHRSG)_+ = \{CHRG\}$
 $(CRSG)_+ = \{CRSG\}$
 $(THRSG)_+ = \{THRSGC\}$

Siêu khoá được thể hiện bằng màu đỏ.

Bước 6: Tạo $(A_i A_j A_k A_r A_s A_t)^+$ với $1 \leq i \leq n, 1 \leq j \leq n, 1 \leq k \leq n, 1 \leq r \leq n, 1 \leq s \leq n, 1 \leq t \leq n$

$(CTHRSG)_+ = \{CTHSRG\}$

Siêu khoá được thể hiện bằng màu đỏ.

Nhìn chung, cho 6 thuộc tính chúng ta sẽ phải xét số trường hợp là

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} + \binom{6}{4} + \binom{6}{5} + \binom{6}{6} = 6 + 15 + 20 + 15 + 1 = 63$$

*** Bài tập thực hành ở nhà: tìm tất cả các khoá của $R=(A, B, C, D)$ với $F = \{A \rightarrow B, B \rightarrow C\}$

Chuẩn hoá dựa trên khoá chính

Chuẩn hoá là một kỹ thuật chính thống cho việc phân tích các quan hệ dựa trên khoá chính (hoặc khoá dự bị) và các thuộc tính và các phụ thuộc hàm. Kỹ thuật này liên quan tới một

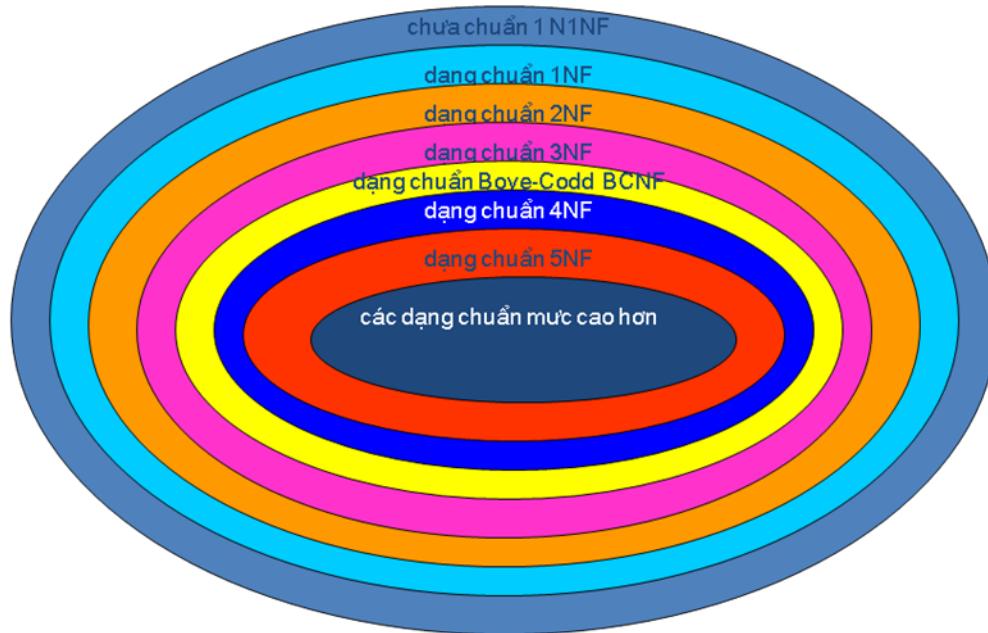
chuỗi các luật có thể được dùng để kiểm tra từng quan hệ sao cho một cơ sở dữ liệu có thể được chuẩn hoá tới một mức độ nào đó.

Khi một yêu cầu không được thoả mãn, quan hệ đã vi phạm yêu cầu đó được phân chia ra thành một tập các quan hệ khác mà mỗi trong số chúng đều thoả mãn các yêu cầu của sự chuẩn hoá đó. Việc chuẩn hoá thường được thực thi như một chuỗi các bước. Mỗi bước liên quan tới một dạng chuẩn cụ thể có những thuộc tính được biết rõ.

Mỗi quan hệ giữa các dạng chuẩn hoá được mô tả trong hình vẽ dưới đây.

Các yêu cầu chuẩn hoá

Đối với mô hình quan hệ, một vấn đề rất quan trọng và thiết yếu là phải nhận ra được một quan hệ vừa được tạo ra đã ở dạng chuẩn một (1NF) chưa. Tất cả các dạng chuẩn mức cao hơn sau đó là tùy theo từng trường hợp, có thể có hoặc không. Tuy nhiên để tránh hiện tượng dị thường khi cập nhật mà chúng ta đã bàn luận đến trong bài trước, người thiết kế cơ sở dữ liệu thường được khuyến cáo phải đưa toàn bộ các quan hệ trong cơ sở dữ liệu về ít nhất là dạng chuẩn 3 (3NF). Như hình vẽ dưới đây mô tả thì một số quan hệ ở 1NF cũng sẽ ở 2NF, một số ở 2NF thì cũng ở 3NF, và cứ như vậy với các mức chuẩn cao hơn.



Chúng ta sẽ xem các yêu cầu cho mỗi dạng chuẩn và một kỹ thuật phân tách để đạt được các lược đồ quan hệ trong các dạng chuẩn đó.

Dạng chưa phải chuẩn 1 (Non-first normal form-N1NF)

Các quan hệ ở dạng chưa chuẩn 1 là những quan hệ chứa một hoặc nhiều thuộc tính chưa phải nguyên tố. Nói một cách khác, trong một quan hệ và trong một bộ có nhiều thuộc địa trị. Một số mở rộng quan trọng tới mô hình quan hệ trong đó các quan hệ N1NF được sử

dụng. Với hầu hết các phần còn lại trong phạm vi bài giảng chúng ta sẽ không bàn luận đến chúng một cách chi tiết. Các cơ sở dữ liệu quan hệ về thời gian và một số loại về không gian thường rơi vào loại N1NF này.

Dạng chuẩn 1 (First Normal Form-1NF)

Một quan hệ trong đó mỗi giá trị thuộc tính đều ở dạng nguyên tố thì quan hệ đó ở dạng 1NF. Chúng ta sẽ chỉ quan tâm tới các quan hệ dạng 1NF cho hầu hết các phần của môn học này. Lưu ý là các quan hệ chưa phải thuộc dạng chuẩn 1 nếu chúng chứa các thuộc tính ghép, các thuộc tính đa trị và các thuộc tính dẫn xuất. Khi gặp những thuộc tính đa trị ở mức khái niệm, nhắc lại rằng trong quá trình chuyển đổi sang mô hình dữ liệu thì cần tạo một bảng riêng biệt cho thuộc tính đa trị đó.

Trước khi chuyển sang xem xét dạng chuẩn 2NF, ta tìm hiểu một số thuật ngữ mới.

Một khoá là một siêu khoá với các thuộc tính ràng buộc là nếu loại bỏ bất kể thuộc tính nào từ khoá này thì sẽ biến khoá đó không còn là một siêu khoá nữa. Nói một cách khác, khoá có số thuộc tính là nhỏ nhất.

Một khoá dự bị cho một quan hệ là một tập các khoá nhỏ nhất cho lược đồ quan hệ đó.

Khoá chính cho một quan hệ là một khoá dự bị được lựa chọn ra. Tất cả các khoá dự bị còn lại trở thành khoá phụ (hay khoá thứ cấp).

Một thuộc tính khoá (chính) là thuộc tính của lược đồ quan hệ R mà là thành viên của một khoá dự bị nào đó của R.

Thuộc tính không khoá là bất kỳ thuộc tính nào của R mà không phải là thành viên của một khoá dự bị nào cả.

Dạng chuẩn hai (2NF)

đây là dạng chuẩn dựa trên khái niệm của một phụ thuộc hàm đầy đủ. Một phụ thuộc hàm $X \rightarrow Y$ là một phụ thuộc hàm đầy đủ nếu loại bỏ bất kể thuộc tính A nào từ A sẽ làm cho phụ thuộc hàm này không còn đúng nữa.

với mọi thuộc tính $A \in X$, $X - \{A\}$ không $\rightarrow Y$

Một phụ thuộc hàm $X \rightarrow Y$ là một phụ thuộc hàm một phần nếu một thuộc tính A nào đó có thể bị loại khỏi X mà phụ thuộc hàm đó vẫn đúng.

với một thuộc tính $A \in X$, $X - \{A\} \rightarrow Y$

Định nghĩa chuẩn 2NF: một lược đồ quan hệ R ở dạng 2 NF với một tập phụ thuộc hàm F nếu nó ở dạng 1NF và mọi thuộc tính không khoá đều phụ thuộc hàm đầy đủ vào mọi khoá của R. Một cách khác để phát biểu định nghĩa trên là: không tồn tại một thuộc tính không

khoá nào mà phụ thuộc hàm một phần vào một khoá nào đó của R. Nói một cách khác, không có thuộc tính không khoá chỉ phụ thuộc vào một phần của khoá của R.

Ví dụ về chuẩn 2NF: Cho quan hệ $R = (A, D, P, G)$, $F = \{AD \rightarrow PG, A \rightarrow G\}$ và $K = \{AD\}$

Thì R không ở dạng 2NF bởi vì G phụ thuộc một phần vào khoá AD do $AD \rightarrow G$ nhưng $A \rightarrow G$.

Phân tách R thành:

$$\begin{array}{ll} R1 = (A, D, P) & R2 = (A, G) \\ K1 = \{AD\} & K2 = \{A\} \\ F1 = \{AD \rightarrow P\} & F2 = \{A \rightarrow G\} \end{array}$$

Bài 11: Giới thiệu về chuẩn hoá-phân 2

Dạng chuẩn 3NF

Dạng chuẩn 3NF dựa trên khái niệm về phụ thuộc bắc cầu.

Định nghĩa phụ thuộc bắc cầu: cho một quan hệ R và một tập phụ thuộc hàm F của R, một tập con $X \subseteq R$ và một thuộc tính A của R. A được nói là phụ thuộc hàm bắc cầu vào X nếu tồn tại $Y \subseteq R$ mà $X \rightarrow Y$, Y không $\rightarrow X$ và $Y \rightarrow A$ và $A \notin X \cup Y$.

Một cách định nghĩa khác cho một phụ thuộc hàm bắc cầu là: một phụ thuộc hàm $X \rightarrow Y$ trong một lược đồ quan hệ R là phụ thuộc bắc cầu nếu có một tập thuộc tính $Z \subseteq R$ mà Z không phải là tập con của bất kỳ khoá nào của R nhưng $X \rightarrow Z$ và $Z \rightarrow Y$.

Định nghĩa chuẩn 3NF: Một lược đồ quan hệ R ở dạng 3NF với một tập phụ thuộc hàm F nếu nó ở dạng 2NF và với bất kỳ phụ thuộc hàm $X \rightarrow A$ thì hoặc là (1) X là một siêu khoá của R hoặc (2) A là một thuộc tính khoá.

Định nghĩa 3NF cách khác: Một lược đồ quan hệ R ở dạng 3NF với tập phụ thuộc hàm F nếu nó ở dạng 2NF và không có thuộc tính không khoá nào phụ thuộc hàm bắc cầu vào một khoá của R.

Ví dụ: Cho $R = (A, B, C, D)$ với $K = \{AB\}$, $F = \{AB \rightarrow CD, C \rightarrow D, D \rightarrow C\}$ thì R không ở dạng 3NF vì $C \rightarrow D$ và C không phải là siêu khoá của R. Một cách khác, R không ở dạng 3NF vì $AB \rightarrow C$ và $C \rightarrow D$ và vì D là một thuộc tính không khoá nhưng lại phụ thuộc bắc cầu vào khoá AB.

Tại sao cần chuẩn 3NF

Chúng ta cùng phân tích xem tại sao lại cần dạng chuẩn 3NF thông qua việc xét cơ sở dữ liệu ví dụ sau đây với thể hiện dữ liệu của lược đồ quan hệ ở bảng sau đó.

assign(flight, day, pilot-id, pilot-name)

$K = \{\text{flight day}\}$

$F = \{\text{pilot-id} \rightarrow \text{pilot-name}, \text{pilot-name} \rightarrow \text{pilot-id}\}$

flight	day	pilot-id	pilot-name
112	Feb.11	317	Mark
112	Feb. 12	246	Kristi
114	Feb.13	317	Mark

Vì $\{\text{flight day}\}$ là khoá nên rõ ràng là $\{\text{flight day}\} \rightarrow \text{pilot-name}$. Nhưng trong F ta có $\text{pilot-name} \rightarrow \text{pilot-id}$, và ta có $\{\text{flight day}\} \rightarrow \text{pilot-id}$. Bây giờ giả sử rằng ta thêm một bảng ghi mới vào bảng trên như bảng sau thì $\text{pilot-name} \rightarrow \text{pilot-id}$ bị vi phạm. Một phụ thuộc hàm bắc cầu tồn tại vì $\text{pilot-id} \rightarrow \text{pilot-name}$ và pilot-id không phải là siêu khoá.

flight	day	pilot-id	pilot-name
112	Feb.11	317	Mark
112	Feb. 12	246	Kristi
114	Feb.13	317	Mark
112	Feb. 11	319	Mark

Chuẩn Boyce-Codd BCNF

Là một chuẩn đẹp hơn dạng chuẩn 3NF.

Định nghĩa: một lược đồ quan hệ R ở dạng chuẩn BCNF với một tập các phụ thuộc hàm F nếu với bất kỳ phụ thuộc hàm dạng $X \rightarrow A$ nào và $A \not\subseteq X$, thì X là một siêu khoá của R. Ví dụ: Cho $R = (A, B, C)$ với $F = \{AB \rightarrow C, C \rightarrow A\}$ $K = \{AB\}$ thì R không ở dạng BCNF vì $C \rightarrow A$ và C không phải là một siêu khoá của R.

Lưu ý rằng sự khác nhau duy nhất trong định nghĩa của dạng chuẩn 3 và chuẩn BCNF là BCNF bỏ đi sự cho phép A trong $X \rightarrow A$ phải là thuộc tính khoá. Một khía cạnh thú vị đối với BCNF là Boyce và Codd ban đầu dự định dạng xây dựng dạng chuẩn này là một dạng đơn giản hơn chuẩn 3NF. Nói một cách khác, nó đã được mong muốn là nằm giữa 2NF và 3NF tuy nhiên sau đó nó được nhanh chóng chứng minh rằng thậm chí còn mạnh hơn cả 3 NF và vì vậy nó đứng ở vị trí giữa 3NF và 4NF.

Trong thực tế, hầu hết các lược đồ quan hệ mà đã ở dạng chuẩn 3NF thì cũng ở dạng chuẩn BCNF. Chỉ nếu $X \rightarrow A$ trong lược đồ nhưng X không phải là một siêu khoá hoặc A là một thuộc tính khoá, thì lược đồ này ở dạng 3NF nhưng không ở dạng BCNF.

Phân tách lược đồ quan hệ về các dạng chuẩn

Mục đích cơ bản của thiết kế cơ sở dữ liệu quan hệ là việc đảm bảo rằng mọi quan hệ trong cơ sở dữ liệu đều hoặc ở dạng chuẩn 3NF hoặc ở dạng chuẩn BCNF. 1NF và 2 NF không loại bỏ được một số lượng đủ các dị thường khi cập nhật để làm nên một sự khác biệt đáng kể trong khi 3NF và BCNF loại bỏ được hầu hết các dị thường khi cập nhật. Như chúng ta đã đề cập đến trước đây, thêm vào việc đảm bảo lược đồ quan hệ hoặc ở 3NF hoặc ở BCNF, người thiết kế phải đảm bảo rằng việc phân tách các lược đồ ban đầu trong cơ sở dữ liệu về hai dạng này phải thoả mãn hai thuộc tính là (1) kết nối không mất mát thông tin và (2) các phụ thuộc hàm được bảo toàn sau khi phân tách.

Hiện tại cũng có một số các thuật toán phân tách đảm bảo chuẩn 3NF và đảm bảo không tồn thất thông tin và bảo toàn phụ thuộc hàm. Tuy nhiên, không có thuật toán nào đảm bảo phân tách được về BCNF mà vẫn đảm bảo kết nối không tồn thắt thông tin và bảo toàn phụ thuộc hàm. Chỉ có một thuật toán đảm bảo không tồn thắt thông tin về dạng BCNF nhưng không đảm bảo sự bảo toàn phụ thuộc hàm. Vì lý do này mà nhiều khi, chuẩn 3NF được coi là một dạng chuẩn mạnh trong khả năng có thể của một số lược đồ quan hệ vì nếu cố gắng đưa về BCNF vì có thể dẫn tới sự không bảo toàn các phụ thuộc hàm. Chúng ta sẽ xem xét hai thuộc tính này một cách kỹ càng trong phần tiếp sau đây.

Bảo toàn các phụ thuộc hàm

Bất kể khi nào một cập nhật được thực hiện đối với cơ sở dữ liệu, hệ quản trị cơ sở dữ liệu phải có khả năng kiểm tra xem việc cập nhật đó sẽ không gây ra một thể hiện dữ liệu không hợp lệ với các phụ thuộc hàm trong F^+ . Để kiểm tra các cập nhật này theo một cách có hiệu quả, cơ sở dữ liệu phải được thiết kế với một tập các lược đồ cho phép việc kiểm tra lại có thể thực hiện được mà không cần đến các phép toán kết nối. Nếu một phụ thuộc hàm nào đó không được bảo toàn thì cách duy nhất để có ràng buộc này là thực hiện phép kết nối hai hoặc nhiều hơn hai quan hệ trong quá trình phân tách này để nhận được một quan hệ bao gồm tất cả các thuộc tính về trái và về phải của phụ thuộc hàm bị mất đó trong cùng một bảng, sau đó có thể kiểm tra lại sự phụ thuộc vẫn được bảo toàn sau khi cập nhật. Hiện nhiên là việc này đòi hỏi rất nhiều công để hiện thực hoá một cách có hiệu quả.

Một cách không chính thức, việc bảo toàn các phụ thuộc hàm có nghĩa là nếu $X \rightarrow Y$ từ F xuất hiện hoặc tường minh trong một lược đồ quan hệ của phép tách hoặc có thể suy diễn ra từ các phụ thuộc hàm xuất hiện trong một số lược đồ khác của phép tách thì tập phụ thuộc hàm ban đầu được bảo toàn trong lược đồ phân tách này.

Một điều quan trọng cần nhớ là bảo toàn tập phụ thuộc hàm không có nghĩa là mọi phụ thuộc hàm trong F đều có mặt một cách tường minh trong một lược đồ quan hệ nào đó của

phép tách, mà tốt hơn nên hiểu là hợp của tất cả các phụ thuộc hàm của từng lược đồ quan hệ trong phép tách phải tương đương với F (khái niệm tương đương ở đây có nghĩa là bao đóng của chúng là như nhau chứ chúng không nhất thiết phải giống hệt nhau).

Phép chiêu của một tập các phụ thuộc hàm lên một tập các thuộc tính Z, ký hiệu là $F[Z]$ (đôi khi còn gọi là $\pi_Z(F)$), là một tập các phụ thuộc hàm $X \rightarrow Y$ trong F^+ sao cho $X \cup Y \subseteq Z$.

Một lược đồ phân tách $\gamma = \{R_1, R_2, \dots, R_m\}$ là bảo toàn phụ thuộc hàm trên tập F nếu hợp của các phép chiêu của F lên mỗi R_i ($1 \leq i \leq m$) trong γ tương đương với F: $(F[R_1] \cup F[R_2] \cup \dots \cup F[R_m])^+ = F^+$

Thông thường thì có thể tìm thấy lược đồ phân tách D bảo toàn phụ thuộc hàm với tập phụ thuộc hàm F sao cho mỗi lược đồ quan hệ trong D đều ở dạng chuẩn 3NF. Tiếp tới chúng ta sẽ xem đến một thuật toán đảm bảo một sự phân tách về 3NF mà vẫn bảo toàn phụ thuộc hàm.

Thuật toán kiểm tra tính bảo toàn phụ thuộc hàm

Algorithm Preserve

```
// đầu vào: một phân tách D= (R1, R2, ..., Rk), một tập các phụ thuộc hàm F, một phụ thuộc hàm //X → Y
//đầu ra: true nếu D được bảo toàn F, false nếu trái lại
Preserve (D , F, X → Y)
    Z = X;
    while (những thay đổi tới Z xảy ra) do
        for i = 1 to k do // có k lược đồ trong D
            Z = Z ∪ ( (Z ∩ Ri)+ ∩ Ri)
        endfor;
    endwhile;
    if Y ⊆ Z
        then return true; // Z ⊨ X → Y
    else return false;
end.
```

Sự hoạt động của thuật toán trên như sau: tập Z được tính về có bǎn như sau $G = \bigcup_{i=1}^k F[R_i]$

Lưu ý là G không được tính trên thực tế mà cũng hiếm khi được kiểm tra xem thực sự G có phủ F hay không. Để kiểm tra xem G có phủ F hay không chúng ta cần xét mỗi phụ thuộc hàm $X \rightarrow Y$ trong F và xác định xem X^+ có chứa Y hay không. Vì vậy kỹ thuật này là tính mà không cần G có sẵn bằng cách lặp đi lặp lại xem xét bao đóng của F với các kết quả phép chiêu của F lên các R_i khác nhau. Một ví dụ chạy thuật toán trên:

Let $R = (A, B, C, D)$ với $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$ và phép tách $D = \{(AB), (BC), (CD)\}; G = F[AB] \cup F[BC] \cup F[CD]$ $Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$

Kiểm tra cho mỗi phụ thuộc hàm trong F.

Kiểm tra cho $A \rightarrow B$

$$\begin{aligned} Z &= A, \\ &= \{A\} \cup ((A \cap AB)^+ \cap AB) \\ &= \{A\} \cup ((A)^+ \cap AB) \\ &= \{A\} \cup (ABCD \cap AB) \\ &= \{A\} \cup \{AB\} \\ &= \{\mathbf{AB}\} \\ Z &= \{AB\} \\ &= \{AB\} \cup ((AB \cap BC)^+ \cap BC) \\ &= \{AB\} \cup ((B)^+ \cap BC) \\ &= \{AB\} \cup (BCDA \cap BC) \\ &= \{AB\} \cup \{BC\} \\ &= \{\mathbf{ABC}\} \\ Z &= \{ABC\} \\ &= \{ABC\} \cup ((ABC \cap CD)^+ \cap CD) \\ &= \{ABC\} \cup ((C)^+ \cap CD) \\ &= \{ABC\} \cup (CDAB \cap CD) \\ &= \{ABC\} \cup \{CD\} \\ &= \{\mathbf{ABCD}\} \end{aligned}$$

Vì vậy G chứa $A \rightarrow B$

Kiểm tra $B \rightarrow C$

$$\begin{aligned} Z &= B, \\ &= \{B\} \cup ((B \cap AB)^+ \cap AB) \\ &= \{B\} \cup ((B)^+ \cap AB) \\ &= \{B\} \cup (BCDA \cap AB) \\ &= \{B\} \cup \{AB\} \\ &= \{\mathbf{AB}\} \\ Z &= \{AB\} \\ &= \{AB\} \cup ((AB \cap BC)^+ \cap BC) \\ &= \{AB\} \cup ((B)^+ \cap BC) \\ &= \{AB\} \cup (BCDA \cap BC) \\ &= \{AB\} \cup \{BC\} \\ &= \{\mathbf{ABC}\} \\ Z &= \{ABC\} \\ &= \{ABC\} \cup ((ABC \cap CD)^+ \cap CD) \\ &= \{ABC\} \cup ((C)^+ \cap CD) \\ &= \{ABC\} \cup (CDAB \cap CD) \\ &= \{ABC\} \cup \{CD\} \\ &= \{\mathbf{ABC}\} \end{aligned}$$

Vì vậy G chứa $B \rightarrow C$

Kiểm tra C→D

$$Z = C,$$

$$= \{C\} \cup ((C \cap AB)^+ \cap AB)$$

$$= \{C\} \cup ((\emptyset)^+ \cap AB)$$

$$= \{C\} \cup (\emptyset)$$

$$= \{\mathbf{C}\}$$

$$Z = \{C\}$$

$$= \{C\} \cup ((C \cap BC)^+ \cap BC)$$

$$= \{C\} \cup ((C)^+ \cap BC)$$

$$= \{C\} \cup (CDAB \cap BC)$$

$$= \{C\} \cup \{BC\}$$

$$= \{\mathbf{BC}\}$$

$$Z = \{BC\}$$

$$= \{BC\} \cup ((BC \cap CD)^+ \cap CD)$$

$$= \{BC\} \cup ((C)^+ \cap CD)$$

$$= \{BC\} \cup (CDAB \cap CD)$$

$$= \{BC\} \cup \{CD\}$$

$$= \{\mathbf{BCD}\}$$

Vì vậy **G** chứa **C → D**

Kiểm tra D→A

$$Z = D,$$

$$= \{D\} \cup ((D \cap AB)^+ \cap AB)$$

$$= \{D\} \cup ((\emptyset)^+ \cap AB)$$

$$= \{D\} \cup (\emptyset)$$

$$= \{\mathbf{D}\}$$

$$Z = \{D\}$$

$$= \{D\} \cup ((D \cap BC)^+ \cap BC)$$

$$= \{D\} \cup ((\emptyset)^+ \cap BC)$$

$$= \{D\} \cup (\emptyset)$$

$$= \{\mathbf{D}\}$$

$$Z = \{D\}$$

$$= \{D\} \cup ((D \cap CD)^+ \cap CD)$$

$$= \{D\} \cup ((D)^+ \cap CD)$$

$$= \{D\} \cup (DABC \cap CD)$$

$$= \{D\} \cup \{CD\}$$

$$= \{\mathbf{DC}\}$$

G có thay đổi vì vậy tiếp tục.

Kiểm tra D→A tiếp tục ở lượt thứ hai qua D.

$$Z = DC,$$

$$= \{DC\} \cup ((DC \cap AB)^+ \cap AB)$$

$$= \{DC\} \cup ((\emptyset)^+ \cap AB)$$

$$= \{DC\} \cup (\emptyset)$$

$$= \{\mathbf{DC}\}$$

$$Z = \{DC\}$$

$$= \{DC\} \cup ((DC \cap BC)^+ \cap BC)$$

$$= \{DC\} \cup ((C)^+ \cap BC)$$

$$\begin{aligned}
 &= \{D\} \cup (CDAB \cap BC) \\
 &= \{D\} \cup (BC) \\
 &= \{\mathbf{DBC}\} \\
 Z &= \{DBC\} \\
 &= \{DBC\} \cup ((DBC \cap CD)^+ \cap CD) \\
 &= \{DBC\} \cup ((CD)^+ \cap CD) \\
 &= \{DBC\} \cup (CDAB \cap CD) \\
 &= \{DBC\} \cup \{CD\} \\
 &= \{\mathbf{DBC}\} \quad G có thay đổi vì vậy tiếp tục.
 \end{aligned}$$

Kiểm tra cho $D \rightarrow A$ tiếp tục trên lượt thứ ba qua D.

$$\begin{aligned}
 Z &= DBC, \\
 &= \{DBC\} \cup ((DBC \cap AB)^+ \cap AB) \\
 &= \{DBC\} \cup ((B)^+ \cap AB) \\
 &= \{DBC\} \cup (BCDA \cap AB) \\
 &= \{DBC\} \cup (AB) \\
 &= \{\mathbf{DBCA}\}
 \end{aligned}$$

Cuối cùng, chúng ta đưa mọi thuộc tính vào trong R.

Vì vậy, G chứa $D \rightarrow A$. Vì vậy, D bảo toàn phụ thuộc hàm trong F.

***** Bài tập thực hành ở nhà: Kiểm tra xem liệu D có bảo toàn phụ thuộc hàm trong F không với: $R = (C, S, Z)$

$$\begin{aligned}
 F &= \{CS \rightarrow Z, Z \rightarrow C\} \\
 D &= \{(SZ), (CZ)\}
 \end{aligned}$$

Thuật toán kiểm tra tính kết nối không tồn thắt thông tin

Algorithm Lossless

//đầu vào: một lược đồ quan hệ $R = (A_1, A_2, \dots, A_n)$, một tập các phụ thuộc hàm F, một lược đồ //phân tách $D = \{R_1, R_2, \dots, R_k\}$

//đầu ra: true nếu D có tính kết nối không tồn thắt thông, false nếu trái lại
Lossless (R, F, D)

Tạo một ma trận có n cột và k hàng với cột y liên quan tới một thuộc tính A_y ($1 \leq y \leq n$) và hàng x liên quan tới lược đồ quan hệ R_x ($1 \leq x \leq k$). Gọi ma trận này là T.

Điền vào ma trận theo cách sau: tại T_{xy} điền vào kí hiệu a_y nếu A_y nằm trong R_x và kí hiệu b_{xy} nếu không phải vậy.

Lặp lại các công việc sau với mỗi phụ thuộc hàm $X \rightarrow Y$ trong F cho đến khi không còn thay đổi đối với T.

Mỗi khi một phụ thuộc hàm được xét đến, tìm kiếm các hàng trong T đồng nhất trong tất cả các cột liên quan tới các thuộc tính trong A. Chuyển giá trị của tất cả các ô trên các hàng có cùng giá trị X thành giá trị của Y theo cách sau: nếu bắt đầu một biểu tượng Y nào là a_y thì chuyển tất cả chúng thành a_y , nếu trong chúng không có a_y thì cho chúng thành một giá trị bất kỳ trong số các giá trị b_{xy} .

Nếu sau khi thực hiện tất cả những thay đổi có thể đối với T một trong các hàng trở thành $a_1 a_2 \dots a_n$ thì trả về yes, trái lại trả về no.

end.

Xét một ví dụ chạy thuật toán này: Cho $R = (A, B, C, D, E)$; $F = \{A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A\}$; $D = \{(AD), (AB), (BE), (CDE), (AE)\}$
 ma trận khởi tạo T:

	A	B	C	D	E
(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{23}	b_{24}	b_{25}
(BE)	b_{31}	a_2	b_{33}	b_{34}	a_5
(CDE)	b_{41}	b_{42}	a_3	a_4	a_5
(AE)	a_1	b_{52}	b_{53}	b_{54}	a_5

Xem xét mỗi phụ thuộc hàm trong F cho đến khi không có thay đổi nào được thực hiện với ma trận.

$A \rightarrow C$: đồng hóa b_{13}, b_{23}, b_{53} một cách tùy ý chọn một giá trị cho chúng như bảng dưới đây

	A	B	C	D	E
(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{13}	b_{24}	b_{25}
(BE)	b_{31}	a_2	b_{33}	b_{34}	a_5
(CDE)	b_{41}	b_{42}	a_3	a_4	a_5
(AE)	a_1	b_{52}	b_{13}	b_{54}	a_5

$B \rightarrow C$: đồng hóa b_{13}, b_{33} một cách tùy ý chọn một giá trị cho chúng như bảng dưới đây

	A	B	C	D	E
(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{13}	b_{24}	b_{25}

(BE)	b_{31}	a_2	b_{13}	b_{34}	a_5
(CDE)	b_{41}	b_{42}	a_3	a_4	a_5
(AE)	a_1	b_{52}	b_{13}	b_{54}	a_5

C→D: đồng hoá $a_4, b_{24}, b_{34}, b_{54}$ một cách tùy ý chọn một giá trị cho chúng như bảng dưới đây

	A	B	C	D	E
(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{13}	a_4	b_{25}
(BE)	b_{31}	a_2	b_{13}	a_4	a_5
(CDE)	b_{41}	b_{42}	a_3	a_4	a_5
(AE)	a_1	b_{52}	b_{13}	a_4	a_5

DE→C: đồng hoá a_3, b_{13} một cách tùy ý chọn một giá trị cho chúng như bảng dưới đây

	A	B	C	D	E
(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{13}	a_4	b_{25}
(BE)	b_{31}	a_2	a_3	a_4	a_5
(CDE)	b_{41}	b_{42}	a_3	a_4	a_5
(AE)	a_1	b_{52}	a_3	a_4	a_5

CE→A: đồng hoá a_1, b_{31}, b_{41} một cách tùy ý chọn một giá trị cho chúng như bảng dưới đây

	A	B	C	D	E

(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{13}	a_4	b_{25}
(BE)	$\textcolor{red}{a}_1$	a_2	a_3	a_4	a_5
(CDE)	$\textcolor{red}{a}_1$	b_{42}	a_3	a_4	a_5
(AE)	$\textcolor{red}{a}_1$	b_{52}	a_3	a_4	a_5

Lượt đầu tiên duyệt F đã hoàn thành, tuy nhiên hàng (BE) tất cả đã trở thành a_i vì vậy kết thúc và trả về true, sự phân tách này thoả mãn kết nối không tồn thất thông tin.

	A	B	C	D	E
(AD)	a_1	b_{12}	b_{13}	a_4	b_{15}
(AB)	a_1	a_2	b_{13}	a_4	b_{25}
(BE)	a_1	a_2	a_3	a_4	a_5
(CDE)	a_1	b_{42}	a_3	a_4	a_5
(AE)	a_1	b_{52}	a_3	a_4	a_5

Bài 12: Giới thiệu về chuẩn hoá-phần 3

Thuật toán số 1 cho việc phân tách về 3NF

Algorithm 3NF.1

// đầu vào: một lược đồ quan hệ $R = (A_1, A_2, \dots, A_n)$, một tập các phụ thuộc hàm F , một tập các khoá dự bị K .

// đầu ra: một phân tách về 3NF của R , được gọi là D , thoả mãn tính kết nối không mất mát thông tin và bảo toàn các phụ thuộc hàm.

3NF.1 (R, F, K)

$a = 0;$

for each fd $X \rightarrow Y$ in F do

$a = a + 1;$

$R_a = XY;$

endfor

```

if [không có lược đồ  $R_b$  ( $1 \leq b \leq a$ ) nào chứa một khoá dự bị của  $R$ ] then
     $a = a + 1;$ 
     $R_a$  = một khoá dự bị nào đó của  $R$ 
endif
if [ $\bigcup_{b=1}^a R_b \neq R$ ] then //bị thiếu một số thuộc tính
     $R_{a+1} = R - \bigcup_{b=1}^a R_b$ 
    return  $D = \{R_1, R_2, \dots, R_{a+1}\}$ 
end.

```

Một ví dụ sử dụng thuật toán 3NF.1

Cho lược đồ quan hệ $R = (A, B, C, D, E)$ với $K = \{AB, AC\}$ và

$$F = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow B, C \rightarrow D, B \rightarrow E\}$$

Bước 1: $D = \{(ABCDE), (ACBDE), (BC), (CB), (CD), (BE)\}$

Tối giản hóa: $D = \{(ABC), (BC), (CD), (BE)\}$

Bước 2: Kiểm tra xem D có chứa một khoá dự bị của R không? Có, trong (ABC)

Bước 3: Tất cả các thuộc tính của R có được chứa trong D ? Có

Trả về D là $\{(ABC), (BC), (CD), (BE)\}$

Thuật toán số 2 cho việc phân tách về 3NF

Algorithm 3NF.2

// đầu vào: một lược đồ quan hệ $R = (A_1, A_2, \dots, A_n)$, một tập các phụ thuộc hàm F , một tập các khoá dự bị K .

// đầu ra: một phân tách về 3NF của R , được gọi là D , không đảm bảo thoả mãn cả tính kết nối không mất mát thông tin và bảo toàn các phụ thuộc hàm.

// Thuật toán này dựa trên ý tưởng loại bỏ các phụ thuộc hàm bắc cầu.

3NF.2 (R, F, K)

do

if [$K \rightarrow Y \rightarrow A$ với A là một thuộc tính không khoá và không phải là một thành phần của K hoặc y] then phân tách R thành: $R_1 = \{R - A\}$ với $K_1 = \{K\}$ và $R_2 = \{YA\}$

với $K_2 = \{Y\}$.

repeat until không tồn tại các phụ thuộc hàm bắc cầu trong bất kể một lược đồ nào

$D = \text{Hợp của tất cả các lược đồ 3NF được sinh ra ở trên.}$

kiểm tra tính kết nối không mất mát thông tin

kiểm tra tính bảo toàn phụ thuộc hàm

end.

Một ví dụ về sử dụng thuật toán 3NF.2

Cho lược đồ quan hệ $R = (A, B, C, D, E)$ với $K = \{AB, AC\}$ và

$$F = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow B, C \rightarrow D, B \rightarrow E\}$$

Bước 1: R không ở 3NF vì $AB \rightarrow C \rightarrow D$

Phân tách thành: $R_1 = (A, B, C, E)$ với $K_1 = K = \{AB, AC\}$ và $R_2 = (C, D)$ with $K_2 = \{C\}$

Bước 2: R_2 ở 3NF. R_1 không ở 3NF vì $AB \rightarrow B \rightarrow E$

Phân tách R_1 thành: $R_{11} = (A, B, C)$ với $K_{11} = K_1 = K = \{AB, AC\}$

$R_{12} = (B, E)$ với $K_{12} = \{B\}$

Bước 3: R_2 , R_{11} , và R_{12} đều ở 3NF

Bước 4: Kiểm tra tính kết nối không mất mát thông tin như sau

$AB \rightarrow CDE$: (lần thứ nhất: không đồng nhất ô nào cả)

$AC \rightarrow BDE$: (lần thứ nhất: không đồng nhất ô nào cả)

$B \rightarrow C$: (lần thứ nhất: đồng nhất a_3 & b_{33})

$C \rightarrow B$: (lần thứ nhất: đồng nhất a_2 & b_{12})

$C \rightarrow D$: (lần thứ nhất: đồng nhất b_{14} , b_{24} , b_{34}) – dừng vì hàng thứ hai tất cả trở thành a

$B \rightarrow E$: (lần thứ nhất: đồng nhất a_5 , b_{15} , b_{25})

Phân tách có thuộc tính kết nối không tồn thất thông tin

	A	B	C	D	E
(CD)	b_{11}	a_2	a_3	a_4	b_{15}
(ABC)	a_1	a_2	a_3	a_4	b_{15}
(BE)	b_{31}	a_2	a_3	a_4	a_5

Bước 5: Kiểm tra xem có bảo toàn phụ thuộc hàm

Cho $R = (A, B, C, D, E)$

$$F = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow B, C \rightarrow D, B \rightarrow E\}$$

$$D = \{(CD), (ABC), (BE)\}$$

$$G = F[CD] \cup F[ABC] \cup F[BE]$$

$$Z = Z \cup ((Z \cap R_i^+) \cap R_i^-)$$

Kiểm tra cho $AB \rightarrow CDE$

$$Z = AB,$$

$$= \{AB\} \cup ((AB \cap CD)^+ \cap CD) = \{AB\} \cup ((\emptyset)^+ \cap CD)$$

$$= \{AB\} \cup (\emptyset \cap CD) = \{AB\} \cup (\emptyset)$$

$$= \{\mathbf{AB}\}$$

$$Z = \{AB\} \cup ((AB \cap ABC)^+ \cap ABC) = \{AB\} \cup ((AB)^+ \cap ABC)$$

$$= \{AB\} \cup (ABCDE \cap ABC) = \{AB\} \cup (ABC)$$

$$= \{\mathbf{ABC}\}$$

$$= \{ABC\} \cup ((ABC \cap BE)^+ \cap BE) = \{ABC\} \cup ((B)^+ \cap BE)$$

$$= \{ABC\} \cup (BCDE \cap BE) = \{ABC\} \cup (BE)$$

$$= \{\mathbf{ABCE}\}$$

$$Z = \{ABCE\} \cup ((ABCE \cap CD)^+ \cap CD) = \{ABCE\} \cup ((C)^+ \cap CD)$$

$$= \{ABCE\} \cup (CBDE \cap CD) = \{ABCE\} \cup (CD)$$

$$= \{\mathbf{ABCDE}\} \text{ vì vậy } \mathbf{AB} \rightarrow \mathbf{CDE} \text{ được bảo toàn}$$

Kiểm tra cho $\mathbf{AC} \rightarrow \mathbf{BDE}$

$$Z = AC$$

$$= \{AC\} \cup ((AC \cap CD)^+ \cap CD) = \{AC\} \cup ((C)^+ \cap CD)$$

$$= \{AC\} \cup (CBDE \cap CD) = \{AC\} \cup (CD)$$

$$= \{\mathbf{ACD}\}$$

$$= \{ACD\} \cup ((ACD \cap ABC)^+ \cap ABC) = \{ACD\} \cup ((AC)^+ \cap ABC)$$

$$= \{ACD\} \cup (ACBDE \cap ABC) = \{ACD\} \cup (ABC)$$

$$= \{\mathbf{ACD}\}$$

$$Z = \{ACD\} \cup ((ACD \cap BE)^+ \cap BE) = \{ACD\} \cup ((B)^+ \cap BE)$$

$$= \{ACD\} \cup (BCDE \cap BE) = \{ACD\} \cup (BE)$$

$$= \{\mathbf{ABCDE}\} \text{ vì vậy } \mathbf{AC} \rightarrow \mathbf{BDE} \text{ được bảo toàn}$$

Kiểm tra cho $\mathbf{B} \rightarrow \mathbf{C}$

$$Z = B$$

$$= \{B\} \cup ((B \cap CD)^+ \cap CD) = \{B\} \cup ((C)^+ \cap CD)$$

$$\begin{aligned}
 &= \{B\} \cup (CBDE \cap CD) && = \{B\} \cup (CD) \\
 &= \{\textbf{BCD}\} \text{ vì vậy } \mathbf{B \rightarrow C} \text{ được bảo toàn}
 \end{aligned}$$

Kiểm tra cho $\mathbf{C \rightarrow B}$

$$\begin{aligned}
 Z &= C \\
 &= \{C\} \cup ((C \cap CD)^+ \cap CD) && = \{C\} \cup ((C)^+ \cap CD) \\
 &= \{C\} \cup (CBDE \cap CD) && = \{C\} \cup (CD) \\
 &= \{\textbf{CD}\} \\
 Z &= \{CD\} \cup ((CD \cap ABC)^+ \cap ABC) && = \{CD\} \cup ((C)^+ \cap ABC) \\
 &= \{CD\} \cup (CBDE \cap ABC) && = \{CD\} \cup (BC) \\
 &= \{\textbf{BCD}\} \text{ vì vậy } \mathbf{C \rightarrow B} \text{ được bảo toàn}
 \end{aligned}$$

Kiểm tra cho $\mathbf{C \rightarrow D}$

$$\begin{aligned}
 Z &= C \\
 &= \{C\} \cup ((C \cap CD)^+ \cap CD) && = \{C\} \cup ((C)^+ \cap CD) \\
 &= \{C\} \cup (CBDE \cap CD) && = \{C\} \cup (CD) \\
 &= \{\textbf{CD}\} \text{ vì vậy } \mathbf{C \rightarrow D} \text{ được bảo toàn}
 \end{aligned}$$

Kiểm tra cho $\mathbf{B \rightarrow E}$

$$\begin{aligned}
 Z &= B \\
 &= \{B\} \cup ((B \cap CD)^+ \cap CD) && = \{B\} \cup ((\emptyset)^+ \cap CD) \\
 &= \{B\} \cup (\emptyset) && = \{\textbf{B}\} \\
 Z &= \{B\} \cup ((B \cap ABC)^+ \cap ABC) && = \{B\} \cup ((B)^+ \cap ABC) \\
 &= \{B\} \cup (BCDE \cap ABC) && = \{BC\} \cup (BC) \\
 &= \{\textbf{BC}\} \\
 Z &= \{BC\} \\
 &= \{BC\} \cup ((BC \cap ABC)^+ \cap ABC) && = \{BC\} \cup ((C)^+ \cap ABC) \\
 &= \{BC\} \cup (CBDE \cap ABC) && = \{BC\} \cup (BC) \\
 &= \{\textbf{BC}\} \\
 Z &= \{BC\}
 \end{aligned}$$

$$\begin{aligned} &= \{BC\} \cup ((BC \cap BE)^+ \cap BE) = \{BC\} \cup ((B)^+ \cap BE) \\ &= \{BC\} \cup (BCDE \cap BE) = \{BC\} \cup (BE) \\ &= \{\text{BCE}\} \text{ vì vậy, } B \rightarrow E \text{ được bảo toàn} \end{aligned}$$

Tại sao lại sử dụng 3NF.2 mà không sử dụng 3NF.1

Bạn biết rằng thuật toán 3NF.1 sẽ đảm bảo việc phân tách về 3NF thoả mãn cả hai thuộc tính quan trọng là kết nối không tồn thất thông tin và bảo toàn phụ thuộc hàm, trong khi thuật toán 3NF.2 không đảm bảo cả hai thuộc tính đó, vậy tại sao bạn lại cần sử dụng thuật toán 3NF.2 mà không sử dụng 3NF.1 trong một số trường hợp.

Câu trả lời rất đơn giản, Thuật toán 3NF.2 sẽ sinh ra ít lược đồ quan hệ hơn thuật toán 3NF.1. Mặc dù cả hai thuộc tính bảo toàn thông tin và bảo toàn phụ thuộc hàm đều cần phải được kiểm tra một cách độc lập khi sử dụng thuật toán 3NF.2

Thuật toán số 3 để phân tách một lược đồ quan hệ về 3NF

Algorithm 3NF.3

// đầu vào: một lược đồ quan hệ $R = (A_1, A_2, \dots, A_n)$, một tập các phụ thuộc hàm F .
// đầu ra: một phân tách về 3NF của R , được gọi là D , thoả mãn tính kết nối không mất mát thông tin và bảo toàn các phụ thuộc hàm.
// Thuật toán này dựa trên một phủ tối thiểu của F (xem bài giảng số 9).

3NF.3 (R, F)

tìm phủ tối thiểu của F , gọi phủ này là G
for mỗi vế trái X xuất hiện trong G do
 tạo một lược đồ quan $\{X \cup A_1 \cup A_2 \cup \dots \cup A_m\}$ với $A_i (1 \leq i \leq m)$ thể hiện
 tất cả vế phải của các phụ thuộc hàm trong G với vế trái là X .
 đưa toàn bộ các thuộc tính còn lại, nếu có, vào một lược đồ
 nếu không có lược đồ nào chứa một khoá của R thì tạo thêm một lược đồ mới chứa mọi
 khoá dự bị của R .
end.

Thuật toán 3NF.3 rất giống với thuật toán 3NF.1, chỉ khác về cách tạo ra các lược đồ phân tách

- Ở thuật toán 3NF.1, các lược đồ được tạo ra trực tiếp từ F
- Ở thuật toán 3NF.2, các lược đồ được tạo ra từ một phủ tối thiểu của F .

Nhìn chung, thuật toán 3NF.3 sẽ tạo ra ít lược đồ quan hệ hơn thuật toán 3NF.1.

Giới thiệu một kỹ thuật khác để kiểm tra tính bảo toàn phụ thuộc hàm

Chúng ta đã có một thuật toán kiểm tra tính bảo toàn phụ thuộc hàm cho một lược đồ phân tách về dạng chuẩn 3NF có độ phức tạp tính toán khá hiệu quả nhưng thuật toán này vẫn có xu hướng phải thực hiện bằng tay. Trong phần này chúng ta sẽ giải quyết cùng bài toán ví dụ đã được giới thiệu trong bài 11, nhưng sử dụng một kỹ thuật khác dựa trên khái niệm về phủ.

Kỹ thuật mới ở đây là ta sẽ sinh ra phần của G^+ mà cho phép chúng ta biết rằng G phủ F .
 Với D, R , và F cho trước, nếu $D = \{R_1, R_2, \dots, R_n\}$ thì $G = F[R_1] \cup F[R_2] \cup F[R_3] \cup \dots \cup F[R_n]$ và nếu mọi phụ thuộc hàm trong F được suy diễn ra từ G thì G phủ F .

Xét ví dụ: Cho $R = (A, B, C, D)$ với $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$ và $D = \{(AB), (BC), (CD)\}$

$$G = F[AB] \cup F[BC] \cup F[CD]$$

Chiếu lên lược đồ (AB)

$$\begin{aligned} F[AB] &= A^+ \cup B^+ \cup (AB)^+ \\ &= \{ABCD\} \cup \{ABCD\} \cup \{ABCD\} \\ &= \{AB\} \cup \{AB\} \cup \{AB\} = \{\mathbf{AB}\}, \mathbf{A \rightarrow B} \text{ được phủ} \end{aligned}$$

Chiếu lên lược đồ (BC)

$$\begin{aligned} F[BC] &= B^+ \cup C^+ \cup (BC)^+ \\ &= \{BCDA\} \cup \{CDAB\} \cup \{BCDA\} \\ &= \{BC\} \cup \{BC\} \cup \{BC\} = \{\mathbf{BC}\}, \mathbf{C \rightarrow C} \text{ được phủ} \end{aligned}$$

Chiếu lên lược đồ (CD)

$$\begin{aligned} F[CD] &= C^+ \cup D^+ \cup (CD)^+ \\ &= \{CDAB\} \cup \{DABC\} \cup \{CDAB\} \\ &= \{CD\} \cup \{CD\} \cup \{CD\} = \{\mathbf{CD}\}, \mathbf{C \rightarrow D} \text{ được phủ} \end{aligned}$$

áp dụng phép chiếu phủ

Vì vậy, các phép chiếu sẽ bao phủ mọi phụ thuộc hàm trong F trừ $D \rightarrow A$. Vì vậy, vấn đề

trở thành việc xác định xem G có suy diễn được $D \rightarrow A$?

- Cách làm như sau: Sinh ra D^+ (sử dụng các phụ thuộc hàm từ G) và nếu A thuộc bao đóng này thì câu trả lời là có suy diễn được.
- Ta có $D_G^+ = \{D, C, B \rightarrow A\}$, $G \models D \rightarrow A$

Bài 13: Giới thiệu về ngôn ngữ truy vấn có cấu trúc SQL

Lịch sử phát triển của ngôn ngữ SQL

SQL là viết tắt của từ tiếng Anh structural query language đã trở thành một ngôn ngữ tuân thủ chuẩn de facto cho việc tạo ra và truy vấn các cơ sở dữ liệu quan hệ. Nó được chấp nhận bởi viện tiêu chuẩn quốc gia Hoa Kỳ ANSI và tổ chức tiêu chuẩn quốc tế ISO cũng như thoả mãn các tiêu chuẩn xử lý thông tin của liên bang FIPS.

Giữa những năm 1974 và 1979, các nhân viên làm việc trong phòng nghiên cứu thí nghiệm của công ty IBM tại San Jose, bang California, Hoa Kỳ đã tiến hành phát triển hệ thống có tên là R, ngay sau khi bài báo truyền thống định nghĩa cơ sở dữ liệu quan hệ được công bố. Mục tiêu của hệ thống R là chứng minh tính khả thi việc cài đặt mô hình quan hệ trong một hệ quản trị cơ sở dữ liệu. Họ sử dụng một ngôn ngữ có tên là SEQUEL (Structured English Query Language), là một ngôn ngữ nối tiếp của SQUARE (Specifying Queries as Relational Expressions), cả hai đều được phát triển tại IBM ở San Jose. Sau đó SEQUEL được đổi tên thành SQL trong dự án xây dựng hệ thống R này.

Bản thân hệ thống R chưa bao giờ được đưa ra thương mại hoá nhưng nó trực tiếp đưa đến sự phát triển của SQL/DS (bản SQL chạy trên hệ điều hành DOS năm 1981, và trên VMS OS một phiên bản máy ảo vào năm 1982) là hệ quản trị cơ sở dữ liệu quan hệ được thương mại hoá đầu tiên của IBM.

Tuy nhiên IBM không phải là công ty đưa ra phiên bản cài đặt thương mại đầu tiên cho một hệ quản trị cơ sở dữ liệu quan hệ mà vinh dự đó được chuyển cho Oracle (cho ra đời phần mềm quan hệ) năm 1979. Ngày nay hệ thống quản trị cơ sở dữ liệu quan hệ của tất cả các công ty sản xuất đều dựa trên SQL. Mỗi nhà sản xuất cung cấp tất cả các đặc tính chuẩn của SQL nhưng hầu hết mỗi nhà sản xuất đều cung cấp thêm những đặc tính riêng biệt của mình, được gọi là mở rộng của SQL chuẩn. Những mở rộng này dẫn đến các vấn đề về khả năng dùng lẫn khi đem các ứng dụng dựa trên SQL sang các hệ quản trị cơ sở dữ liệu khác. Các nhà sản xuất cố gắng phân biệt các phiên bản SQL của họ thông qua những mở rộng này.

Phiên bản hiện thời của chuẩn ANSO cho SQL là SQL-99 (hay còn được biết đến như SQL3). Chuẩn này cũng được chấp nhận bởi ISO. Mặc dù rất nhiều bản mở rộng của SQL tồn tại, chúng ta sẽ xem xét bản chất bên trong của SQL, là nội dung sẽ tìm thấy trong mọi hệ quản trị cơ sở dữ liệu. Cho dù bạn sử dụng Oracle, Microsoft SQL Server, IBM's DB2, Microsoft Access hay MySQL hoặc bất kể hệ quản trị cơ sở dữ liệu được xây dựng tốt nào, bạn đều có thể tìm hiểu hệ thống đó một cách nhanh chóng với những thông tin sẽ được đề cập tới trong bài giảng này.

Giới thiệu về ngôn ngữ SQL

SQL là một ngôn ngữ cơ sở dữ liệu quan hệ đầy đủ vì nó bao gồm cả hai loại ngôn ngữ: ngôn ngữ định nghĩa dữ liệu DDL và ngôn ngữ thao tác dữ liệu DML. Chúng ta sẽ tìm hiểu các thành phần của cả hai loại ngôn ngữ này của SQL. Nếu bạn dùng Access, ví dụ như vậy, bạn sẽ cần biết ít về DDL hơn khi bạn dùng Oracle9i hoặc MySQL. Bảng dưới đây sẽ tóm tắt các câu lệnh trong phân DDL của SQL. Mỗi mục vào của bảng không thể hiện trật tự bạn sử dụng các câu lệnh này, đơn giản chỉ đưa ra cho bạn một danh sách tóm tắt các lệnh có sẵn của DDL nhưng cũng không phải là một danh sách đầy đủ.

Câu lệnh hoặc lựa chọn	Mô tả
CREATE SCHEMA AUTHORIZATION	Tạo ra một lược đồ cơ sở dữ liệu
CREATE TABLE	Tạo ra một bảng mới trong lược đồ cơ sở dữ liệu của người dùng
NOT NULL	Ràng buộc đảm bảo một cột sẽ không có giá trị rỗng
UNIQUE	Ràng buộc đảm bảo một cột sẽ không có giá trị trùng lặp
PRIMARY KEY	Định nghĩa một khoá chính cho một bảng
FOREIGN KEY	Định nghĩa một khoá ngoại cho một bảng
DEFAULT	Định nghĩa một giá trị ngầm định cho một cột (khi không đưa một giá trị nào đó vào)
CHECK	Ràng buộc được sử dụng để kiểm tra tính đúng đắn của dữ liệu trong một cột
CREATE INDEX	Tạo chỉ mục cho một bảng
CREATE VIEW	Tạo một tập con động cần thiết các hàng/cột từ một hoặc nhiều bảng
ALTER TABLE	Thay đổi định nghĩa của một bảng: thêm vào/xoá bỏ/cập nhật các thuộc tính hoặc ràng buộc
DROP TABLE	Xoá vĩnh viễn một bảng (cùng với dữ liệu của nó) từ một lược đồ cơ sở dữ liệu

DROP INDEX	Xoá vĩnh viễn một tập chỉ mục
DROP VIEW	Xoá vĩnh viễn một khung nhìn

Trước khi bạn có thể sử dụng một hệ quản trị cơ sở dữ liệu, hai công việc cần được thực hiện đó là: (1) tạo một cấu trúc cơ sở dữ liệu và (2) tạo các bảng để lưu trữ dữ liệu của người dùng cuối. Hoàn thành công việc thứ nhất liên quan tới việc kiến tạo ra các tệp vật lý để lưu cơ sở dữ liệu. Hệ quản trị cơ sở dữ liệu sẽ tự động tạo ra các bảng từ điển dữ liệu và tạo ra một người quản trị cơ sở dữ liệu ngầm định (DBA). Việc tạo ra các tệp vật lý đòi hỏi sự tương tác giữa hệ điều hành và hệ quản trị cơ sở dữ liệu. Vì vậy, tạo ra cấu trúc cơ sở dữ liệu là một đặc tính có sự khác nhau từ một hệ quản trị cơ sở dữ liệu này sang hệ khác.

Với một ngoại lệ là có thể tạo ra cơ sở dữ liệu, hầu hết các nhà cung cấp hệ thống quản trị cơ sở dữ liệu sử dụng bản SQL có khác một chút với bản SQL chuẩn của ANSI. Mặc dù vậy nhưng cũng chỉ thỉnh thoảng bạn mới gặp những sự khác nhau nhỏ trong cú pháp của các câu lệnh SQL. Ví dụ, hầu hết để yêu cầu mọi câu lệnh SQL được kết thúc bởi dấu chấm phẩy (;) tuy nhiên một số bản cài đặt SQL không sử dụng dấu (;). Hầu hết những sự khác nhau chung về cú pháp sẽ được chỉ ra trong bài giảng này hoặc ít nhất cũng liệt kê những sự khác nhau do người viết nhận thức được.

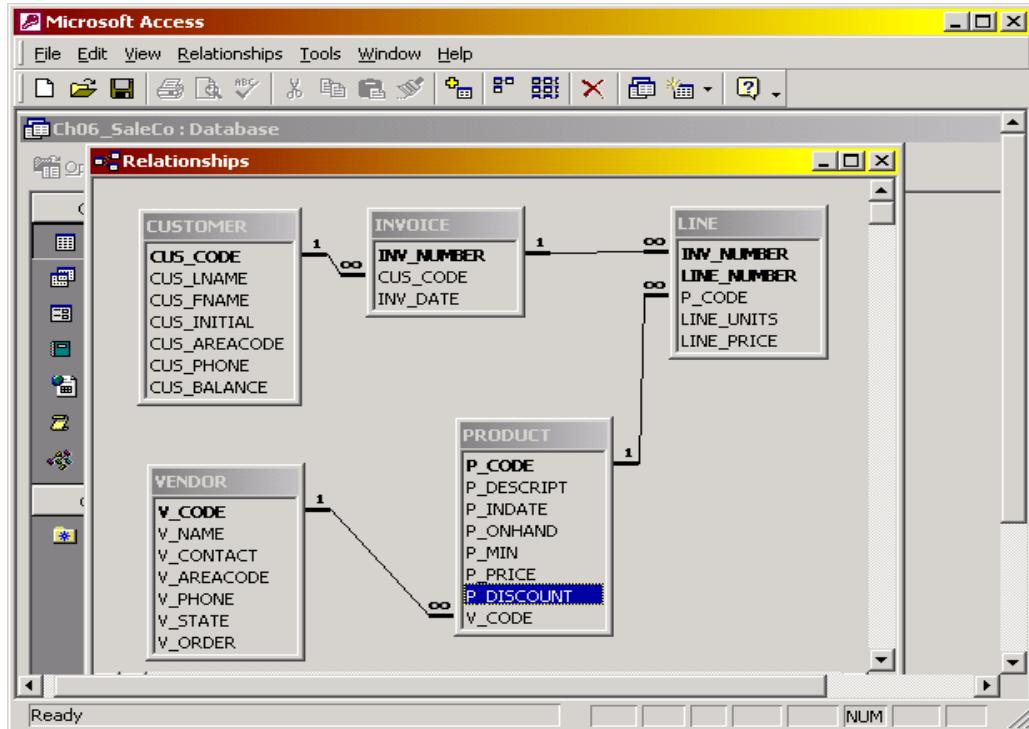
Cách sử dụng các lệnh của ngôn ngữ định nghĩa dữ liệu DDL trong SQL

Chúng ta sẽ dùng cơ sở dữ liệu dưới đây để mô tả cho cách sử dụng của các lệnh DDL của SQL. Cơ sở dữ liệu này liên quan một chút tới cơ sở dữ liệu về nhà cung cấp-linh kiện-công việc-vận chuyển. Các qui định về nghiệp vụ của hệ thống này như sau:

- 1- Một khách hàng có thể có nhiều yêu cầu trả tiền. mỗi bản yêu cầu trả tiền (invoice) chỉ được tạo ra bởi một khách hàng
- 2- Một invoice chứa một hoặc nhiều dòng. mỗi dòng của invoice liên quan tới một invoice
- 3- Mỗi dòng invoice là cho một mặt hàng. Một mặt hàng có thể tìm thấy ở nhiều dòng khác nhau. Một nhà cung cấp có thể cung cấp nhiều mặt hàng. Một vài nhà cung cấp có thể không cung cấp mặt hàng nào cả.
- 4- Nếu một mặt hàng được cung cấp bởi một nhà cung cấp, thì mặt hàng đó chỉ được cung cấp bởi duy nhất nhà cung cấp đó.
- 5- một số mặt hàng không được cung cấp bởi nhà cung cấp nào cả mà chúng được công ty tự sản xuất (in-house) hoặc được cung cấp qua những cách khác.

Cơ sở dữ liệu của hệ thống ví dụ trên được thiết kế nhanh trong Access và được thể hiện trong hình vẽ dưới đây trong đó

Customer để chỉ thực thể khách hàng; Invoice để thể hiện các yêu cầu trả tiền của khách hàng; Line là bảng để lưu thông tin các dòng của các invoice; Vendor để lưu thông tin của các nhà cung cấp; Product để lưu thông tin của các mặt hàng.



Các qui định về ký pháp cho câu lệnh SQL được trình bày trong bài giảng
được thể hiện trong bảng sau đây

Ký pháp	Mô tả
Viết hoa	từ khoá cần thiết cho câu lệnh SQL
<i>viết nghiêng</i>	Một tham số do người dùng cung cấp- thường là cần thiết
{a b ... }	Một tham số bắt buộc, sử dụng một trong số danh sách lựa chọn
[...]	Một tham số lựa chọn-mọi thứ trong ngoặc vuông đều là lựa chọn
<i>tablename</i>	Tên của bảng
<i>column</i>	Tên của một thuộc tính trong một bảng

<i>data type</i>	Một định nghĩa kiểu dữ liệu đúng
<i>constraint</i>	Một định nghĩa ràng buộc đúng cách
<i>condition</i>	Một biểu thức điều kiện đúng cách-trả về giá trị đúng hoặc sai
<i>columnlist</i>	Một hoặc nhiều tên cột hoặc biểu thức được phân cách nhau bởi dấu phẩy
<i>tablelist</i>	Một hoặc nhiều tên bảng được phân cách nhau bởi dấu phẩy
<i>conditionlist</i>	Một hoặc nhiều biểu thức điều kiện được phân cách nhau bởi dấu phẩy
<i>expression</i>	Một giá trị đơn (ví dụ 76 or ‘married’) hoặc một công thức (ví dụ., price-10)

Câu lệnh tạo cấu trúc bảng trong SQL

Câu lệnh CREATE TABLE có cú pháp như sau

```
CREATE TABLE tablename (
    column1 data type [constraint] [, 
    column2 data type [constraint] ] [, 
    PRIMARY KEY (column1 [,column2]) ] [, 
    FOREIGN KEY (column1 [,column2]) REFERENCES tablename ] [, 
    CONSTRAINT constraint] );
```

Ví dụ tạo một bảng VENDOR của cơ sở dữ liệu ví dụ được mô tả ở trên

```
CREATE TABLE VENDOR (
    V_CODE          INTEGER      NOT NULL      UNIQUE,
    V_NAME          VARCHAR(35)  NOT NULL,
    V_CONTACT       VARCHAR(15)   NOT NULL,
    V_AREACODE     CHAR(3)      NOT NULL,
    V_PHONE         CHAR(8)      NOT NULL,
    V_STATE         CHAR(2)      NOT NULL,
    V_ORDER         CHAR(1)      NOT NULL,
    PRIMARY KEY ( V_CODE));
```

trong câu lệnh trên ta đã dùng kỹ pháp để tạo khoá chính V_CODE là PRIMARY KEY

Sau đó ta sẽ tạo tiếp đến bảng PRODUCT như sau

```
CREATE TABLE PRODUCT (
    P_CODE          VARCHAR(10)      NOT NULL      UNIQUE,
    P_DESCRPT       VARCHAR(35)      NOT NULL,
    P_INDATE        DATE           NOT NULL,
    P_ONHAND        SMALLINT        NOT NULL,
    P_MIN           SMALLINT        NOT NULL,
    P_PRICE          NUMBER(8,2)     NOT NULL,
    P_DISCOUNT      NUMBER(4,2)     NOT NULL,
    V_CODE           INTEGER,
PRIMARY KEY ( P_CODE),
FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);
```

Trong ví dụ trên khoá ngoại V-CODE được tạo ra bằng cách dùng từ khoá FOREIGN KEY, nó được tham chiếu tới bảng VENDOR bằng cách dùng từ khoá REFERENCE với ràng buộc ON UPDATE CASCADE (sẽ được mô tả sau)

Ví dụ tiếp theo là tạo ra bảng CUSTOMER như sau

- Now let's create the CUSTOMER table as described on page 11.

```
CREATE TABLE CUSTOMER (
    CUS_CODE         NUMBER          PRIMARY KEY,
    CUS_LNAME        VARCHAR(15)     NOT NULL,
    CUS_FNAME        VARCHAR(15)     NOT NULL, ràng buộc cột
    CUS_INITIAL      CHAR(1),
    CUS_AREACODE    CHAR(3)        DEFAULT '615' NOT NULL
                                CHECK (CUS_AREACODE IN ('615', '713', '931')),
    CUS_PHONE        CHAR(8)         NOT NULL,
    CUS_BALANCE      NUMBER(9,2)     DEFAULT 0.00,
CONSTRAINT CUS_UI1 UNIQUE (CUS_LNAME, CUS_FNAME));
```

trong ví dụ này ta dùng thêm các cách để tạo ràng buộc cho mỗi cột riêng rẽ, tạo giá trị ngầm định bằng từ khoá DEFAULT, tạo ràng buộc kiểm tra tính đúng đắn của dữ liệu nhập vào bằng việc dùng hàm có từ khoá CHECK (kiểm tra các giá trị nhập vào của

CUS_AREACODE có nằm trong tập 3 giá trị ('615','713','931'). Tạo ràng buộc cho toàn bảng dùng từ khoá CONSTRAINT.

Bây giờ là tạo bảng INVOICE với câu lệnh như sau

```
CREATE TABLE INVOICE (
    INV_NUMBER      NUMBER          PRIMARY KEY,
    CUS_CODE        NUMBER          NOT NULL, REFERENCES CUSTOMER(CUS_CODE),
    INV_DATE        DATE           DEFAULT SYSDATE NOT NULL,
    CONSTRAINT INV_CK1 CHECK (INV_DATE > TO_DATE('01-JAN-2002', 'DD-MON-YYYY')));

```

Một cách khác để định nghĩa khóa ngoại

Hàm đặc biệt để trả về ngày hiện tại

trong ví dụ này ràng buộc CHECK được sử dụng để kiểm tra tính hợp lệ của ngày invoice có lớn hơn 1/1/2002 không. Hàm TO_DATE cần hai tham số bao gồm ngày cụ thể và định dạng ngày được sử dụng

Cuối cùng chúng ta tạo bảng LINE như sau

```
CREATE TABLE LINE (
    INV_NUMBER      NUMBER          NOTNULL,
    LINE_NUMBER     NUMBER(2,0)      NOTNULL,
    P_CODE          VARCHAR(10)     NOTNULL,
    LINE_UNITS      NUMBER(9,2)      DEFAULT 0.00 NOTNULL,
    LINE_PRICE      NUMBER(9,2)      DEFAULT 0.00 NOTNULL,
    PRIMARY KEY (INV_NUMBER, LINE_NUMBER),
    FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE ON DELETE CASCADE
    FOREIGN KEY (P_CODE) REFERENCES PRODUCT(P_CODE),
    CONSTRAINT LINE_UI1 UNIQUE(INV_NUMBER, P_CODE));

```

trong ví dụ này một ràng buộc trên toàn bảng được tạo ra để ngăn chặn không có hai dòng trong một invoice giống nhau. Ngoài ra việc sử dụng ON DELETE CASCADE được khuyến cáo nên dùng cho các thực thể yếu để đảm bảo rằng việc xoá một dòng (tương ứng với một mã invoice) trong thực thể chính (khoẻ) sẽ gây ra việc xoá tự động các dòng tương ứng (các dòng thể hiện các mặt hàng của mã invoice bị xoá) trong thực thể yếu phụ thuộc vào thực thể chính đó.

Một số các lưu ý đối với việc tạo bảng

- Đối với cơ sở dữ liệu ví dụ trên, bảng PRODUCT chứa một khóa ngoại tham chiếu tới bảng VENDOR. Vì vậy, bảng VENDOR phải được tạo trước. Nói chung, các

bảng nằm bên phía lực lượng 1 của một quan hệ 1-nhiều phải được tạo trước khi bảng bên phía lực lượng nhiều có thể được tạo ra.

- Với hệ thống Oracle9i nếu bạn sử dụng cách định nghĩa chính bằng từ khoá PRIMARY KEY bạn không cần đưa yêu cầu NOT NULL và UNIQUE vào câu lệnh tạo bảng nữa. Thực tế, bạn sẽ nhận được một thông báo lỗi nếu bạn làm như vậy
- Ràng buộc ON UPDATE CASCADE là một phần của chuẩn ANSI nhưng nhiều hệ quản trị cơ sở dữ liệu không hỗ trợ nps. Oracle là một trong số những hệ thống quản trị không hỗ trợ tính năng này.
- Nếu khoá chính là một khoá ghép, tất cả các thuộc tính của khoá được chứa trong một dấu ngoặc đơn và được phân tách nhau bởi dấu phẩy. Ví dụ, bảng LINE có khoá chính được định nghĩa như sau

PRIMARY KEY (inv_number, line_number)

- Hỗ trợ ràng buộc tham chiếu rất đa dạng, thay đổi từ hệ quản trị này sang hệ quản trị khác
 - o MS Access, SQL Server và Oracle hỗ trợ ON DELETE CASCADE (để tránh trường hợp dị thường khi xoá bô)
 - o MS Access, SQL Server hỗ trợ ON UPDATE CASCADE (để tránh dị thường khi cập nhật)
 - o Oracle không hỗ trợ ON UPDATE CASCADE
 - o MS Access, SQL Server không hỗ trợ SET NULL
 - o Oracle hỗ trợ SET NULL
- MS Access không hỗ trợ ON DELETE CASCADE hoặc ON UPDATE CASCADE tại mức câu lệnh SQL tuy nhiên nó lại hỗ trợ thông quan giao diện cửa sổ quan hệ.

Chúng ta tạm dừng xem xét cách sử dụng các câu DDL của SQL ở đây để chuyển sang giới thiệu về phần ngôn ngữ thao tác dữ liệu DML của SQL trước, sau đó sẽ quay lại DDL với những câu lệnh phức tạp hơn

Ngôn ngữ thao tác dữ liệu DML của SQL

Ngôn ngữ thao tác dữ liệu của SQL có thể được chia ra làm hai phần tách riêng nhưng vẫn chung nhau ở một số phạm vi nào đó. Hai phần này là các câu lệnh DML không truy vấn và các câu lệnh truy vấn dữ liệu

Ngôn ngữ thao tác không truy vấn cho phép bạn thêm dữ liệu vào bảng (INSERT), sửa đổi dữ liệu (UPDATE), xoá dữ liệu từ các bảng (DELETE) và thực hiện những thay đổi vĩnh viễn (COMMIT) và huỷ những thay đổi (tới một mức độ nào đó với ROLLBACK).

Các câu lệnh truy vấn DML chắc chắn bao gồm câu lệnh đơn SELECT với rất nhiều các mệnh đề lựa chọn khác nhau. Chúng ta sẽ xem xét các câu lệnh không truy vấn của DML trước.

Tóm tắt các câu lệnh DML của SQL được mô tả trong bảng dưới đây

Câu lệnh hoặc lựa chọn	Mô tả
INSERT	Chèn thêm một (các) hàng vào trong một bảng
SELECT	Lựa chọn các thuộc tính từ các hàng trong một hoặc nhiều bảng hoặc khung nhìn
WHERE	Hạn chế việc lựa chọn các hàng dựa trên một biểu thức điều kiện
GROUP BY	Gộp nhóm các hàng đã được chọn ra dựa trên một hoặc nhiều thuộc tính
HAVING	Hạn chế sự lựa chọn các hàng để gộp nhóm dựa trên một điều kiện
ORDER BY	Xếp thứ tự các hàng được chọn
UPDATE	Sửa đổi giá trị thuộc tính của một hoặc nhiều hàng của một bảng
DELETE	Xoá một hoặc nhiều hàng từ một bảng
COMMIT	Lưu trữ vĩnh viễn những thay đổi về dữ liệu
ROLLBACK	Phục hồi dữ liệu về những giá trị ban đầu của chúng
<i>Các phép toán so sánh</i>	
=, <, >, <=, >=, <>	Được sử dụng trong các biểu thức điều kiện
<i>Các phép toán logic</i>	
AND, OR, NOT	Được sử dụng trong các biểu thức điều kiện
Câu lệnh hoặc lựa chọn	Mô tả

<i>Các phép toán đặc biệt</i>	<i>được sử dụng trong các biểu thức điều kiện</i>
BETWEEN	Kiểm tra xem các giá trị của một thuộc tính có nằm trong một khoảng
IS NULL	Kiểm tra xem giá trị của một thuộc tính có là trống không
LIKE	Kiểm tra xem giá trị của một thuộc tính có giống với một kiểu chuỗi ký tự cho trước
IN	Kiểm tra xem giá trị của một thuộc tính có nằm trong một danh sách các giá trị
EXISTS	Kiểm tra xem một truy vấn con có trả về hàng dữ liệu nào không
DISTINCT	Hạn chế các giá trị tối đa những giá trị duy nhất, hay loại bỏ những giá trị trùng lặp
<i>Hàm tổng hợp (thống kê)</i>	<i>được sử dụng với SELECT để trả về những giá trị tổng hợp trên các cột</i>
COUNT	Trả về số lượng các hàng với các giá trị không rỗng cho một cột nào đó
MIN	Trả về giá trị nhỏ nhất của một thuộc tính được tìm thấy trong một cột nào đó
MAX	Trả về giá trị lớn nhất của một thuộc tính được tìm thấy trong một cột nào đó
SUM	Trả về tổng của tất cả các giá trị của một cột nào đó
AVG	Trả về giá trị trung bình của tất cả các giá trị của một cột nào đó

Câu lệnh thêm bản ghi mới vào bảng: SQL dùng câu lệnh INSERT để thêm dữ liệu mới vào một bảng. Cú pháp của câu lệnh này như sau:

```
INSERT INTO tablename
VALUES (value1, value2, ..., value n);
```

Ví dụ: Thêm hai bản ghi mới vào bảng VENDOR chúng ta cần thực hiện hai câu lệnh SQL dưới đây

INSERT INTO VENDOR

```
VALUES (21225, 'Bryson, Inc.', 'Smithson', '615', '223-3234', 'TN', 'Y');
```

INSERT INTO VENDOR

```
VALUES (21226, 'SuperLoo, Inc.', 'Flushing', '904', '215-8995', 'FL', 'N');
```

Nếu một thuộc tính của một bản ghi không có giá trị (hay có giá trị là null) bạn sẽ sử dụng cú pháp sau đây để thêm một hàng vào bảng

INSERT INTO PRODUCT

```
VALUES ('23114-AA', 'Sledge hammer, 12 lb.', '02-Jan-02', 8, 5, 14.40, 0.05,  
NULL);
```

Trong một số trường hợp, nhiều hơn một thuộc tính có thể nhận giá trị null hoặc không. Thay vì khai báo các thuộc tính là NULL trong lệnh INSERT, bạn có thể chỉ cần mô tả các thuộc tính cần có giá trị nhập, không cần quan tâm tới các thuộc tính nhận giá trị NULL.

Việc này được thực hiện bằng cách liệt kê tên của các thuộc tính mà giá trị của chúng được đưa vào bên trong dấu ngoặc đơn ngay sau tên của bảng. Xét ví dụ dưới đây, giả sử rằng chỉ P_CODE và P_DESCRIP cần nhập giá trị vào trong bảng PRODUCT. Hai cách nhập sau đây đều đúng:

INSERT INTO PRODUCT

```
VALUES ('23114-AA', 'Sledge hammer, 12 lb.', NULL, NULL, NULL, NULL,  
NULL, NULL);
```

hoặc

INSERT INTO PRODUCT(P_CODE, P_DESCRIP)

```
VALUES('23114-AA', 'Sledge hammer, 12 lb.');
```

Câu lệnh xoá các bản ghi (hàng) từ một bảng: dễ dàng sử dụng SQL để xoá một hàng từ một bảng, thông qua câu lệnh DELETE. Cú pháp của câu lệnh này như sau:

DELETE FROM *tablename*

```
[WHERE conditionlist];
```

Để xoá một hàng từ một bảng dựa trên giá trị của khoá chính, bạn có thể sử dụng câu lệnh như sau:

DELETE FROM PRODUCT

```
WHERE P_CODE = '23114-AA';
```

Lệnh DELETE cũng có thể xoá bỏ nhiều hàng từ một bảng. Ví dụ bạn muốn xoá các sản phẩm từ bảng PRODUCT mà giá trị P_MIN =5. Để thực hiện điều này bạn có thể dùng câu lệnh:

DELETE FROM PRODUCT

WHERE P_MIN = 5;

Lưu ý câu lệnh DELETE là một câu lệnh hướng tập hợp. Có nghĩa là điều kiện ở câu lệnh WHERE là có thể có hoặc không, nếu điều kiện đó không được chỉ rõ thì tất cả các hàng của bảng sẽ được xoá.

Cập nhật dữ liệu cho các hàng của một bảng: Để thay đổi dữ liệu trong một bảng, câu lệnh UPDATE được sử dụng. Cú pháp của câu lệnh này như sau:

UPDATE *tablename*

SET *columnname* = *expression* [, *columnname* = *expression*]
[WHERE *conditionlist*];

Lưu ý rằng điều kiện WHERE là có thể có hoặc không trong câu lệnh UPDATE. Nếu điều kiện không được xác định, thì câu lệnh UPDATE sẽ thực hiện trên tất cả các hàng của bảng đó.

Ví dụ bạn muốn thay đổi P_INDATE từ 13/12/2003 thành 18/1/2004 trong hàng thứ hai của bảng PRODUCT. Bạn cần sử dụng giá trị của khoá chính 13-Q2/P2 để xác định đúng hàng trong bảng cần thay đổi, câu lệnh tương ứng như sau

UPDATE PRODUCT

SET P_INDATE = '18-Jan-2004'
WHERE P_CODE = '13-Q2/P2';

Nếu nhiều hơn một thuộc tính cần được thay đổi giá trị trong một hàng, các câu lệnh UPDATE sẽ được phân cách nhau bởi dấu phẩy

UPDATE PRODUCT

SET P_INDATE = '18-JAN-2004', P_PRICE = 16.99, P_MIN = 10
WHERE P_CODE = '13-Q2/P2';

Lưu trữ các thay đổi tới một bảng

Mọi thay đổi được thực hiện tới nội dung của bảng sẽ không được lưu trữ một cách vật lý trong một bảng vật lý (là một tệp trong hệ thống) cho tới khi một câu lệnh COMMIT được thực thi.

Phụ thuộc vào sự phức tạp của hệ thống bạn đang làm việc, nếu năng lượng điện bị ngắt trong quá trình cập nhật một bảng (hoặc một cơ sở dữ liệu nói chung), trước khi câu lệnh COMMIT được thực thi, tất cả các thay đổi bạn thực hiện trước đó sẽ bị mất. Những hệ thống phức tạp hơn sẽ có khả năng phục hồi dữ liệu sau những sự cố như vật nhưng các hệ thống máy tính cá nhân thì tốt nhất bạn nên sử dụng bộ lưu điện UPS để hạn chế sự cố này.

Cú pháp của câu lệnh COMMIT như sau

COMMIT [*tablename*]; // lưu tất cả các thay đổi cho một bảng hoặc

COMMIT; // lưu tất cả những thay đổi được thực hiện trong tất cả các bảng

Nếu bạn chưa sử dụng câu lệnh COMMIT để lưu trữ vĩnh viễn những thay đổi trong cơ sở dữ liệu, bạn có thể phục hồi cơ sở dữ liệu về trạng thái trước đó (kết quả của lần cuối thực hiện câu lệnh COMMIT) bằng việc sử dụng câu lệnh ROLLBACK.

Câu lệnh ROLLBACK phục hồi lại những thay đổi được thực hiện và trả lại dữ liệu những giá trị cũ của nó trước khi những thay đổi này được thực hiện. Cú pháp của câu lệnh như sau

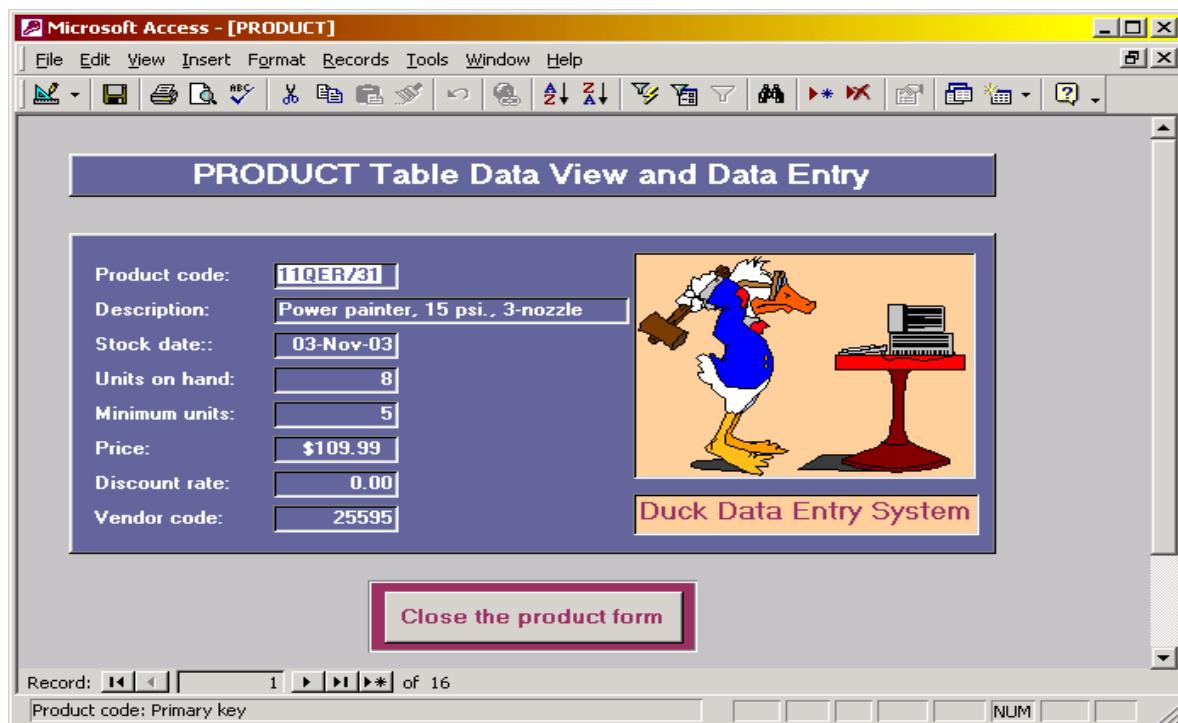
ROLLBACK;

Lưu ý là MS Access không hỗ trợ câu lệnh ROLLBACK. Một số hệ quản trị cơ sở dữ liệu như Oracle tự động COMMIT những thay đổi dữ liệu khi thực hiện các câu lệnh DDL vì vậy ROLLBACK sẽ không làm được gì đối với những hệ thống loại này.

ROLLBACK sẽ phục hồi tất cả những thay đổi kể từ lần hệ thống COMMIT cuối cùng, điều này có nghĩa là thậm chí những thay đổi mà bạn không muốn phục hồi cũng sẽ được khôi phục lại với giá trị cũ nếu không một câu lệnh COMMIT nào được thực thi

Tóm tắt các câu lệnh DML không truy vấn của SQL

Nhìn hình vẽ một biểu mẫu nhập dữ liệu dưới đây, mỗi mục vào trong SQL cũng tương đối phức tạp. Các ứng dụng của người dùng cuối được tạo ra tốt nhất với các tiện ích mà tạo ra các màn hình nhập liệu dễ dàng và hấp dẫn. Bạn sẽ thấy MS Access xử lý việc nhập liệu tốt hơn nhiều so với SQL thuần túy.



Câu lệnh truy vấn của DML của SQL

Câu lệnh truy vấn của DML bao gồm duy nhất một câu lệnh đơn được gọi là lệnh SELECT. Cú pháp của câu lệnh như sau:

SELECT [ALL | DISTINCT] columnlist

FROM tablelist

[WHERE condition]

[GROUP BY columnlist]

[HAVING condition]

[ORDER BY columnlist];

Chúng ta sẽ tìm hiểu hầu hết các đặc tính của câu lệnh SELECT, bắt đầu từ các câu truy vấn đơn giản tới những câu truy vấn phức tạp hơn, vẫn sử dụng cùng một cơ sở dữ liệu mà chúng ta đang xây dựng để minh họa.

Câu lệnh lựa chọn đơn giản trong SQL: đây có lẽ là câu lệnh đơn giản nhất có thể, lấy ra các hàng từ một bảng cụ thể nào đó.

Ví dụ, bạn muốn liệt kê tất cả các thuộc tính của mọi bản ghi của bảng PRODUCT. Nói cách khác, để xem bảng này thì dùng câu lệnh sau:

```
SELECT      P_CODE,    P_DESCRIFT,   P_INDATE,   P_ONHAND,   P_MIN,  
P_PRICE,  P_DISCOUNT, V_CODE  
FROM PRODUCT;
```

Hoặc

```
SELECT *  
FROM PRODUCT;
```

Ở đây dùng kí tự thay thế * để thể hiện tất cả các thuộc tính của một bảng

Sử dụng một câu truy vấn con SELECT để thêm các hàng vào một bảng

Mặc dù câu lệnh INSERT là một thao tác dữ liệu không truy vấn, nhưng nó cũng có thể bao gồm một câu truy vấn. Ví dụ sau sẽ thể hiện điều đó trước khi chúng ta chuyển sang các biểu thức truy vấn phức tạp hơn. SQL cho phép bạn có thể thêm các hàng của một bảng từ dữ liệu lấy từ một bảng khác. Cú pháp thực hiện loại câu lệnh INSERT này như sau:

```
INSERT INTO tablename  
SELECT columnlist  
FROM tablename;
```

Ở đây câu lệnh SELECT là câu lệnh truy vấn con (hoặc truy vấn lồng nhau). Với loại câu lệnh này thì các câu truy vấn bên trong được thực hiện trước bởi hệ quản trị cơ sở dữ liệu, kết quả của nó sẽ được dùng như đầu vào cho câu lệnh bên ngoài sát với câu lệnh đó (ở loại câu lệnh đang xét thì câu lệnh SELECT là loại bên trong, và câu lệnh INSERT là câu lệnh ngoài). Các giá trị được trả về bởi câu lệnh bên trong phải phù hợp với các thuộc tính và kiểu dữ liệu của bảng trong câu lệnh INSERT.

Câu lệnh truy vấn lựa chọn với sự hạn chế của điều kiện

Bạn có thể chọn một phần nội dung của bảng bằng cách thiết lập giới hạn trên các bản ghi kết quả. Điều này được tiến hành với việc dùng mệnh đề WHERE cú pháp như sau

```
SELECT columnlist  
FROM tablelist  
WHERE conditionlist ;
```

Lệnh SELECT sẽ lấy tất cả các hàng thỏa mãn điều kiện được thể hiện ở mệnh đề WHERE.

Ví dụ:

```
SELECT      P_DESCRIFT,   P_INDATE,   P_PRICE,  
V_CODE  
FROM PRODUCT  
WHERE V_CODE = 21344;
```

Cấu trúc của câu lệnh SQL cung cấp hầu hết tính linh hoạt trong truy vấn không hạn chế. Các giới hạn điều kiện số có thể được dùng cho nội dung các bảng được lựa chọn. Trừ việc kiểm tra giá trị các thuộc tính là NULL, SQL không trả về những hàng mà giá trị thuộc tính được lựa chọn là NULL trong kết quả.

Xem ví dụ trong truy vấn dưới đây

```
SELECT P_DESCRIFT, P_INDATE, P_PRICE, V_CODE
```

**FROM PRODUCT
WHERE V_CODE <> 21344;**

Bảng PRODUCT được thể hiện trong hình vẽ dưới đây chứa kết quả của truy vấn trên, đê ý là hàng thứ 10 và 13 trong bảng PRODUCT không xuất hiện trong các kết quả của truy vấn này.

The screenshot shows the Microsoft Access interface with the 'PRODUCT : Table' open. The table has columns: P_CODE, P_DESCRPT, P_INDATE, P_OIHAND, P_MIH, P_PRICE, P_DISCOUNT, and V_CODE. Rows 10 and 13 are highlighted with red boxes. Row 10 contains values: P_CODE '13-Q2/P2', P_DESCRPT '7.25-in. pwr. saw blade', P_INDATE '13-Dec-03', P_OIHAND '32', P_MIH '15', P_PRICE '\$14.99', P_DISCOUNT '0.05', and V_CODE '21344'. Row 13 contains values: P_CODE '1546-QQ2', P_DESCRPT 'Hrd. cloth, 1/4-in., 2x50', P_INDATE '15-Jan-04', P_OIHAND '15', P_MIH '8', P_PRICE '\$39.95', P_DISCOUNT '0.00', and V_CODE '23119'. A green box at the bottom states: 'Hai hàng này không xuất hiện trong kết quả' (These two items do not appear in the results).

Kết quả của truy vấn không thể hiện trong hình vẽ dưới đây

The screenshot shows the Microsoft Access interface with the 'qryFig6-07 : Select Query' open. The table has columns: P_DESCRPT, P_INDATE, P_PRICE, and V_CODE. The data is identical to the first screenshot, except it does not include the two highlighted rows from the original query. A green box at the bottom states: 'Product description'.

Các thủ tục về ngày tháng thường cụ thể cho từng phần mềm hơn các thủ tục SQL khác. Ví dụ, truy vấn để liệt kê tất cả các hàng trong đó ngày lưu kho từ 20/1/2004, sẽ như sau:

```
SELECT P_DESCRIP, P_ONHAND, P_MIN, P_PRICE, P_INDATE  
FROM PRODUCT  
WHERE P_INDATE >= '20-Jan-2004';
```

Lưu ý trong Access ký tự phân biệt dùng cho ngày tháng là # vì vậy câu truy vấn trên trong Access sẽ như sau

```
SELECT P_DESCRIP, P_ONHAND, P_MIN, P_PRICE, P_INDATE  
FROM PRODUCT  
WHERE P_INDATE >= #20-Jan-2004#;
```

Sử dụng các cột tính toán và đổi tên các cột

Giả sử rằng câu truy vấn của bạn cần xác định một giá trị mà không được lưu trữ vật lý trong cơ sở dữ liệu nhưng lại được tính toán từ dữ liệu trong cơ sở dữ liệu. Ví dụ bạn muốn tính tổng giá trị của các mặt hàng hiện tại có trong kho. Về logic, tổng số tiền này bằng tích của số lượng mỗi mặt hàng hiện đang có nhân với giá bán hiện tại của mặt hàng đó. Câu truy vấn SQL cho yêu cầu này được thể hiện như sau

```
SELECT P_DESCRIP, P_ONHAND, P_PRICE, P_ONHAND * P_PRICE AS  
TOTVALUE  
FROM PRODUCT
```

SQL sẽ chấp nhận mọi biểu thức hợp lệ (như P_ONHAND * P_PRICE ở mệnh đề SELECT trong câu truy vấn trên) trong các cột tính toán, sử dụng các thuộc tính của các bảng được liệt kê trong mệnh đề FROM. Chú ý Acces sẽ tự động thêm một nhãn Expr tới tất cả các cột tính toán. Oracle sử dụng biểu thức thực tế làm nhãn của các cột tính toán này.

SQL chuẩn cho phép sử dụng biệt danh (alias-một tên khác) cho bất kể các cột nào trong mệnh đề SELECT. Biệt danh cho mỗi cột được bắt đầu bởi từ khoá AS.

Kết quả của câu lệnh SELECT trên được thể hiện trong hình vẽ dưới đây trong đó cột tính toán được biểu diễn bởi biệt danh của nó.

P_DESCRPT	P_OHID	P_PRICE	TOTVALUE
Power painter, 15 psi, 3-nozzle	8	\$109.99	\$879.92
7.25-in. pwr. saw blade	32	\$14.99	\$479.68
9.00-in. pwr. saw blade	18	\$17.49	\$314.82
Hrd. cloth, 1/4-in., 2x50	15	\$39.95	\$599.25
Hrd. cloth, 1/2-in., 3x50	23	\$43.99	\$1,011.77
B&D jigsaw, 12-in. blade	8	\$109.92	\$879.36
B&D jigsaw, 8-in. blade	6	\$99.87	\$599.22
B&D cordless drill, 1/2-in.	12	\$38.95	\$467.40
Claw hammer	23	\$9.95	\$228.85
Sledge hammer, 12 lb.	8	\$14.40	\$115.20
Rat-tail file, 1/8-in. fine	43	\$4.99	\$214.57
Hicut chain saw, 16 in.	11	\$256.99	\$2,826.89
PVC pipe, 3.5-in., 8-ft	188	\$5.87	\$1,103.56
1.25-in. metal screw, 25	172	\$6.99	\$1,202.28
2.5-in. vvd. screw, 50	237	\$8.45	\$2,002.65
Steel matting, 4'x8'x1/8", .5" mesh	18	\$119.95	\$2,159.10
*	0	\$0.00	

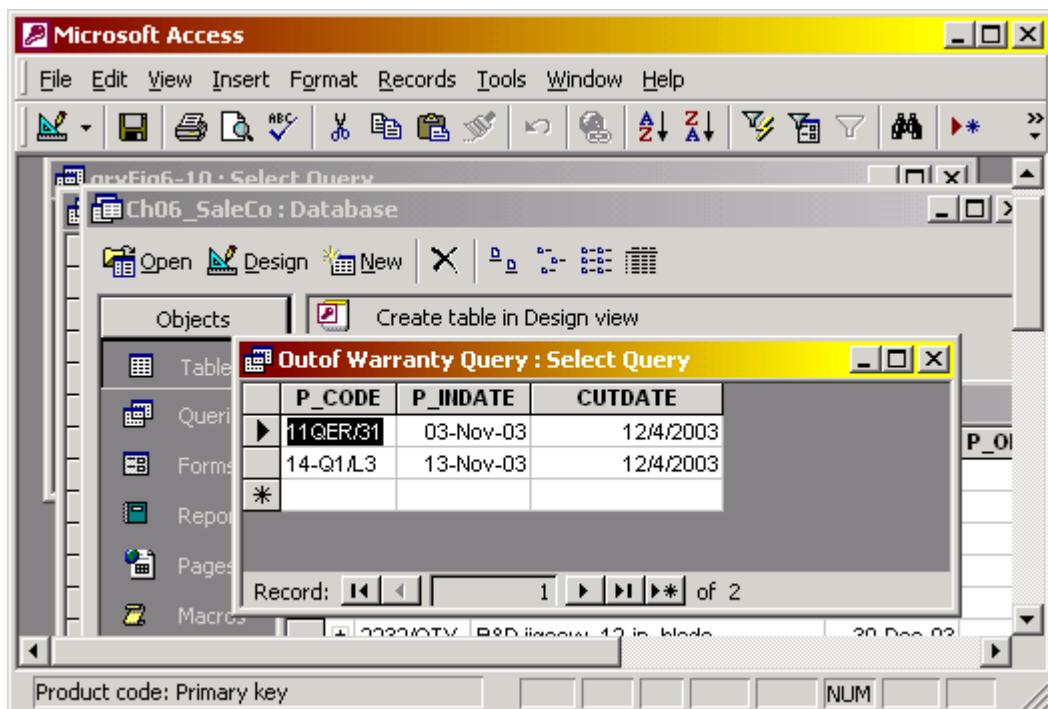
Giả sử chúng ta muốn liệt kê một danh sách các sản phẩm không còn bảo hành nữa. Trong trường hợp này, giả thiết rằng những sản phẩm không còn bảo hành là những sản phẩm được nhập kho quá 90 ngày. Vì thế, P_INDATE phải nhỏ hơn ngày hiện tại ít nhất là 90 ngày. Phiên bản Access của câu truy vấn này được thể hiện dưới đây, tiếp đó là phiên bản của Oracle với kết quả được thể hiện ở màn hình sau đó.

Access Version

```
SELECT P_CODE, P_INDATE, DATE() - 90 AS CUTDATE
FROM PRODUCT
WHERE P_INDATE <= DATE() - 90;
```

Oracle Version

```
SELECT P_CODE, P_INDATE, SYSDATE - 90 AS CUTDATE
FROM PRODUCT
WHERE P_INDATE <= SYSDATE - 90;
```



Trong thực tế, việc tìm kiếm dữ liệu thường liên quan tới nhiều điều kiện. SQL cho phép bạn có thể thể hiện nhiều điều kiện trong một câu truy vấn đơn bằng cách dùng các phép toán logic. Các phép toán logic mà SQL hỗ trợ là AND, OR và NOT.

Ví dụ: giả sử bạn muốn liệt kê một danh sách các sản phẩm từ bảng PRODUCT mà V_CODE=21344 hoặc =24288. Câu truy vấn như sau

```
SELECT P_DESCRIP, P_INDATE, P_PRICE, V_CODE  
FROM PRODUCT  
WHERE V_CODE = 21344 OR V_CODE = 24288;
```

Bài 14: Giới thiệu về ngôn ngữ SQL -phần 2

Các toán tử đặc biệt trong SQL

SQL theo chuẩn ANSI cho phép sử dụng các toán tử đặc biệt trong biểu thức nằm trong mệnh đề WHERE. Các toán tử này bao gồm:

BETWEEN- được dùng để kiểm tra xem giá trị một thuộc tính có nằm trong một khoảng nào đó không

IS NULL- được dùng để xác định liệu một thuộc tính có nhận giá trị NULL không

LIKE- được dùng để gắn giá trị một thuộc tính với một kiểu chuỗi ký tự. Nhiều ký tự thay thế được sẵn có để sử dụng với toán tử này.

IN- được sử dụng để xác định liệu giá trị một thuộc tính có nằm trong một danh sách giá trị không

EXISTS- được dùng để xác định liệu một truy vấn con có trả về một tập rỗng hay không

Các bảng ảo - tạo các khung nhìn

Nhắc lại rằng kết quả của một phép toán quan hệ (kiểu như SELECT trong SQL) là các quan hệ khác (hoặc bảng khác). Trong bài này chúng ta tiếp tục sử dụng cơ sở dữ liệu xuyên suốt bài giảng từ trước để thể hiện các ví dụ minh họa cho ngôn ngữ SQL.

Giả sử rằng cuối ngày làm việc, chúng ta muốn có danh sách của tất cả các mặt hàng được đặt hàng lại, tức là tất cả các mặt hàng có số lượng sẵn có trong kho nhỏ hơn một giá trị ngưỡng nào đó (số lượng nhỏ nhất để phân phối mặt hàng này). Thay vì cứ gõ cùng một câu truy vấn hàng ngày, chúng ta nên lưu trữ câu truy vấn này dài hạn trong cơ sở dữ liệu để dùng đi dùng lại nhiều lần.

Khung nhìn quan hệ là một chức năng để có thể làm được việc này. Trong SQL, một khung nhìn là một bảng dựa trên một câu truy vấn SELECT. Câu truy vấn này có thể chứa các cột, các cột là kết quả tính toán, các cột biệt danh và các hàm tổng hợp từ một hoặc nhiều bảng. Các bảng mà dựa trên đó một khung nhìn được thiết lập được gọi là bảng cơ sở (base table). Các khung nhìn được tạo ra trong SQL sử dụng lệnh CREATE VIEW với cú pháp như sau:

CREATE VIEW *viewname* AS SELECT *query*

Câu lệnh CREATE VIEW là một lệnh DDL nhưng chứa một câu truy vấn con được sử dụng để tạo ra một bảng ảo trong từ điển dữ liệu. Ví dụ:

CREATE VIEW PRODUCT_3 AS

SELECT P_DESCRIPTOR, P_ONHAND, P_PRICE

FROM PRODUCT

WHERE P_PRICE > 50.00;

Lưu ý: Câu lệnh CREATE VIEW không được hỗ trợ trực tiếp trong Access. Để tạo một khung nhìn trong Access, bạn cần tạo một truy vấn SQL và sau đó lưu trữ nó.

Một khung nhìn quan hệ có một số đặc tính đặc biệt sau:

- 1- Bạn có thể sử dụng tên của một khung nhìn ở bất kể vị trí nào mà tên bảng có thể được đặt vào trong câu lệnh SQL
- 2- Các khung nhìn được cập nhật động. Có nghĩa là khung nhìn được tạo lại mỗi khi cần nó liên quan tới công việc đang thực hiện
- 3- Các khung nhìn cung cấp một mức độ bảo mật trong cơ sở dữ liệu bởi vì khung nhìn có thể hạn chế người dùng chỉ truy nhập tới các cột cụ thể và các hàng cụ thể trong một bảng

- 4- Các khung nhìn cũng có thể được sử dụng làm nền tảng cho các báo cáo. Khung nhìn được tạo ra sau đây thể hiện một bảng tóm tắt về tổng cho phí các sản phẩm và thống kê số lượng sản phẩm có trong kho được gộp nhóm theo từng nhà cung cấp

CREATE VIEW SUMPRDXVEN AS

```
SELECT V_CODE, SUM(P_ONHAND*P_PRICE) AS TOTCOST,  
MAX(P_ONHAND) AS MAXQTY, MIN(P_OHAND) AS MINQTY,  
AVG(P_ONHAND) AS AVGQTY  
FROM PRODUCT  
GROUP BY V_CODE;
```

Kết nối các bảng trong cơ sở dữ liệu

Khả năng kết nối các bảng thông qua các thuộc tính chung có lẽ là một đặc tính nổi bật quan trọng nhất của cơ sở dữ liệu quan hệ so với các loại cơ sở dữ liệu khác. Trong SQL, một phép kết nối được thực hiện khi dữ liệu được lấy ra từ nhiều hơn một bảng tại một thời điểm.

Để kết nối các bảng, đơn giản bạn chỉ cần liệt kê các bảng trong mệnh đề FROM của câu lệnh SELECT. Hệ quản trị cơ sở dữ liệu sẽ tạo ra tích Đề các của mọi bảng được liệt kê trong mệnh đề FROM. Để thực hiện phép kết nối tự nhiên, bạn phải chỉ ra cụ thể sự liên kết trên các thuộc tính chung trong mệnh đề WHERE. Sự liên kết này được gọi là **điều kiện kết nối**.

Điều kiện kết nối thường được cấu thành bởi một phép so sánh bằng giữa khoá ngoài và khoá chính của các bảng liên quan. Giả sử rằng chúng ta muốn kết nối hai bảng VENDOR và PRODUCT. V_CODE là khoá ngoài trong bảng PRODUCT và là khoá chính trong bảng VENDOR, điều kiện kết nối sẽ được thể hiện qua thuộc tính này trong câu lệnh sau

```
SELECT PRODUCT.P_DESCRPT, PRODUCT.P_PRICE, VENDOR.V_NAME  
VENDOR.V_CONTACT, VENDOR.V_AREACODE, VENDOR.V_PHONE  
FROM PRODUCT, VENDOR  
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```

Nếu bạn không xác định cụ thể một điều kiện kết nối tại mệnh đề WHERE thì kết quả của câu lệnh sẽ là một tích đề các. Sử dụng cùng một cơ sở dữ liệu, bảng PRODUCT bao gồm 16 bộ (hàng) và bảng VENDOR bao gồm 11 bộ, sẽ cho kết quả một tích đề các gồm $16 \times 11 = 176$ bộ. Hầu hết các bộ này (xem kết quả trong hình vẽ dưới đây) đều không sử dụng đến (không phục vụ cho mục đích câu truy vấn).

Khi kết nối ba hoặc nhiều bảng, bạn cần xác định một điều kiện kết nối cho mỗi cặp bảng. Số lượng các điều kiện kết nối sẽ luôn là N-1 trong đó N là số lượng các bảng được liệt kê trong mệnh đề FROM. Nên cần thận để tránh tạo các điều kiện kết nối vòng tròn. Ví dụ nếu bảng A kết nối với bảng B, bảng B kết nối với bảng C, và bảng C cũng có liên kết với bảng A, lúc đó chỉ tạo hai điều kiện kết nối: A với B và B với C. Không tạo kết nối C với A !

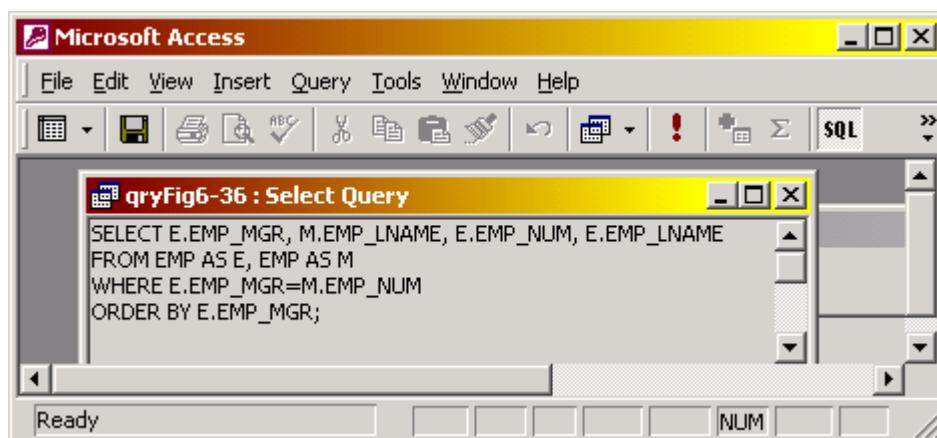
Kết nối đệ quy: là kết nối một bảng với chính bảng đó.

Một biệt danh có thể được sử dụng để xác định bảng nguồn từ đó dữ liệu được lấy ra cho câu truy vấn. Ví dụ:

```
SELECT P_DESCRIP, P_PRICE, V_NAME, V_CONTACT, V_AREACODE,  
V_PHONE  
  
FROM PRODUCT P, VENDOR V  
  
WHERE P.V_CODE = V.V_CODE  
  
ORDER BY P_PRICE;
```

Trong câu truy vấn trên một biệt danh được sử dụng cho bảng PRODUCT và bảng VENDOR. Lưu ý trong Access cần phải thêm từ khoá AS trước biệt danh.

Một biệt danh đặc biệt rất hữu ích khi một bảng phải kết nối với chính nó, cái được gọi là kết nối đệ quy. Ví dụ, sử dụng bảng EMPLOYEE chúng ta muốn tạo ra danh sách tất cả các nhân viên cùng với tên của người quản lý của họ. Nếu không sử dụng một biệt danh cho bảng thì truy vấn này là không thể, bởi vì tên của các thuộc tính cần thiết là không duy nhất để lấy ra, sẽ gây nhầm lẫn cho hệ quản trị cơ sở dữ liệu. Sử dụng biệt danh, câu truy vấn tương ứng trong Access được thể hiện ở hình vẽ dưới đây và kết quả của truy vấn được thể hiện trong hình vẽ sau đó.



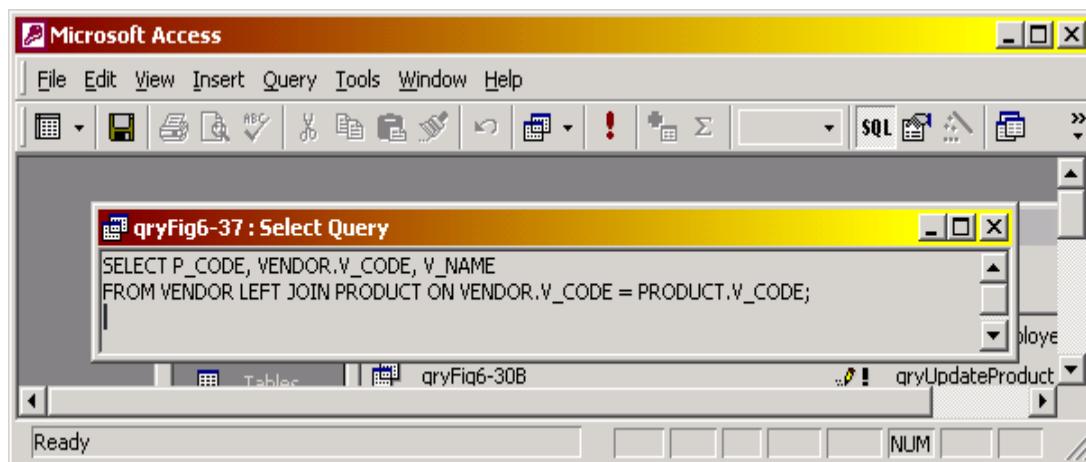
	EMP_MGR	M.EMP_LNAME	EMP_NUM	E.EMP_LNAME
▶	100	Kolmycz	112	Johnson
	100	Kolmycz	103	Jones
	100	Kolmycz	102	Vandam
	100	Kolmycz	101	Lewis
	105	Williams	115	Saranda
	105	Williams	113	Smythe
	105	Williams	111	Washington
	105	Williams	107	Diante
	105	Williams	106	Smith
	105	Williams	104	Lange
	108	Wiesenbach	116	Smith
	108	Wiesenbach	114	Brandon
	108	Wiesenbach	110	Genkazi
	108	Wiesenbach	109	Smith

Kết nối ngoài

Kết quả câu truy vấn cho kết nối tự nhiên trình bày ở trên giữa hai bảng PRODUCT và VENDOR có 14 dòng được liệt kê. Nếu bạn so sánh kết quả này với bản thân các bản ghi của bảng PRODUCT sẽ thấy thiếu mất hai sản phẩm. Lý do là hai sản phẩm bị thiếu đó có các giá trị rỗng ở thuộc tính V_CODE trong bảng PRODUCT. Bởi vì không có giá trị nào trong bảng VENDOR có thể kết nối với giá trị NULL trong PRODUCT nên chúng không xuất hiện trong kết quả cuối cùng của phép kết nối.

Để đưa hai bản ghi thiếu trong trường hợp trên vào kết quả cuối cùng của phép kết nối, chúng ta cần sử dụng một phép kết nối ngoài. Nhắc lại rằng có ba loại kết nối ngoài, kết nối bên trái, bên phải và kết nối đầy đủ cả hai bên. Cho trước hai bảng A và B, A kết nối ngoài bên trái với B cho kết quả là tất cả những bản ghi thoả mãn điều kiện kết nối cộng thêm tất cả các bản ghi không thoả mãn của A. A kết nối ngoài bên phải với B cho kết quả là tất cả các bản ghi thoả mãn điều kiện kết nối cộng thêm tất cả những bản ghi không thoả mãn trong B. Chúng ta sẽ xem xét phép kết nối ngoài đầy đủ sau.

Để thêm bộ có giá trị NULL của V_CODE từ bảng PRODUCT vào kết quả cuối cùng, chúng ta sẽ sử dụng câu truy vấn sau đây



Lưu ý là từ kết nối ngoài không xuất hiện trong câu truy vấn, nó được ngầm hiểu khi thể hiện từ khoá LEFT là kết nối ngoài bên trái, hay RIGHT là kết nối ngoài bên phải.

Kết quả của câu lệnh được thể hiện trong hình vẽ dưới đây: đưa ra tất cả các hàng trong bảng VENDOR thoả mãn điều kiện kết nối trái với bảng PRODUCT.

The image shows three windows from Microsoft Access:

- Projection on PRODUCT :** A table view showing columns P_CODE and V_CODE. Data includes rows like 11QER/31, 25595; 13-Q2/P2, 21344; etc.
- qryFig6-37 : Select Query**: A query result window showing columns P_CODE, V_CODE, and V_NAME. It lists various vendor names and their codes.
- VENDOR : Table**: A table view showing columns V_CODE and V_NAME. Data includes rows like 21225, Bryson, Inc.; 21226, SuperLoo, Inc.; etc.

Bảng VENDOR có một số bản ghi mà V-CODE không phù hợp với bất kể bản ghi nào xuất hiện trong bảng PRODUCT. Nếu thực hiện kết nối ngoài bên phải thể hiện trong hình sau

```

qryFig6-38 : Select Query
SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR RIGHT JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
    
```

Kết quả của truy vấn được thể hiện trong hình vẽ dưới đây trong đó đưa ra tất cả những hàng của PRODUCT phù hợp và không phù hợp với VENDOR

The image shows three Microsoft Access query windows side-by-side:

- Projection on PRODU**: Shows a list of products with their codes and vendor codes.
- qryFig6-38 : Select Query**: A join query showing product codes, vendor codes, and vendor names.
- VENDOR : Table**: Shows a list of vendors with their codes and names.

The join query (middle window) displays the following data:

P_CODE	V_CODE	V_NAME
11QER/31	25595	
13-Q2/P2	21344	
14-Q1/L3	21344	
1546-QQ2	23119	
1558-QWV1	23119	
2232/QTY	24288	
2232/QWE	24288	
2238/QPD	25595	
23109-HB	21225	Bryson, Inc.
23114-AA		
54778-2T	21344	Gomez Bros.
89-WRE-Q	24288	Gomez Bros.
PVC23DRT		
SM-18277	21225	Randsets Ltd.
SW-23116	21231	D&E Supply
WR3/TT3	25595	Rubicon Systems

Các toán tử tập hợp quan hệ

Nhắc lại rằng đại số quan hệ có định hướng tập hợp và bao gồm rất nhiều các toán tử như toán tử hợp, giao, trừ và các thuật ngữ tập hợp, bảng và quan hệ có thể dùng lẫn với nhau trong phạm vi bài giảng về cơ sở dữ liệu quan hệ.

Như với đại số quan hệ thuần túy, các toán tử tập hợp chỉ làm việc với những quan hệ khả hợp. Trong SQL, điều này có nghĩa là tên của các thuộc tính phải giống nhau và kiểu dữ liệu của chúng cũng phải như nhau. Đây là một lĩnh vực mà các hệ quản trị cơ sở dữ liệu quan hệ khác nhau thì rất khác nhau về định nghĩa khái niệm khả hợp. Ví dụ, một số các hệ quản trị cơ sở dữ liệu quan hệ sẽ coi kiểu dữ liệu VARCHAR(35) và VARCHAR(15) là khả hợp bởi vì mặc dù chúng có độ dài khác nhau nhưng đều dựa trên một kiểu dữ liệu cơ bản. Các hệ quản trị khác sẽ không coi hai kiểu dữ liệu này là khả hợp. Bạn có thể thử với hệ quản trị của bạn đang tiến hành bài tập lớn để xem cái gì được gọi là khả hợp và cái gì là không khả hợp.

Toán tử hợp

Xem xét ngữ cảnh sau đây. Giả sử công ty của chúng ta đã mua lại một công ty khác và người quản lý muốn chắc chắn rằng danh sách khách hàng của công ty được trộn với danh sách khách hàng hiện tại của công ty. Vì có thể một vài khách hàng có thể mua hàng từ cả hai công ty nên hai danh sách khách hàng có thể có những khách hàng chung. Người quản lý không muốn có bất kỳ khách hàng nào bị trùng lặp trong danh sách.

Câu lệnh UNION của SQL sẽ tự động loại bỏ những bản ghi trùng lặp từ hai quan hệ toán hạng. nếu bạn muốn bao gồm cả những bản ghi trùng lặp thì dùng câu lệnh UNION ALL.

Cú pháp của câu lệnh truy vấn UNION là *query UNION query*

Về cơ bản câu lệnh UNION kết hợp hai đầu ra của hai câu lệnh truy vấn SELECT. Nhớ rằng đầu ra của hai câu truy vấn này phải khả hợp. Để mô tả truy vấn UNION, hãy kết hợp danh sách các khách hàng hiện tại voi danh sách khách hàng mới như sau

	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
▶	10010	Ramas	Alfred	A	615	844-2573	\$0.00
▶	10011	Dunne	Leona	K	713	894-1238	\$0.00
▶	10012	Smith	Kathy	W	615	894-2285	\$345.86
▶	10013	Ołowski	Paul	F	615	894-2180	\$536.75
▶	10014	Orlando	Myron		615	222-1672	\$0.00
▶	10015	O'Brian	Amy	B	713	442-3381	\$0.00
▶	10016	Brown	James	G	615	297-1228	\$221.19
▶	10017	Williams	George		615	290-2556	\$768.93
▶	10018	Farris	Anne	G	713	382-7185	\$216.55
▶	10019	Smith	Olette	K	615	297-3809	\$0.00
*	0						\$0.00

	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
▶	345	Terrell	Justine	H	615	322-9870
	347	Ołowski	Paul	F	615	894-2180
	351	Hernandez	Carlos	J	723	123-7654
	352	McDowell	George		723	123-7768
	365	Tirpin	Khaleed	G	723	123-9876
	368	Lewis	Marie	J	734	332-1789
	369	Dunne	Leona	K	713	894-1238
*	0					

Kết quả hợp hai bảng như sau

	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
▶	Brown	James	G	615	297-1228
	Dunne	Leona	K	713	894-1238
	Farris	Anne	G	713	382-7185
	Hernandez	Carlos	J	723	123-7654
	Lewis	Marie	J	734	332-1789
	McDowell	George		723	123-7768
	O'Brian	Amy	B	713	442-3381
	Ołowski	Paul	F	615	894-2180
	Orlando	Myron		615	222-1672
	Ramas	Alfred	A	615	844-2573
	Smith	Kathy	W	615	894-2285
	Smith	Olette	K	615	297-3809
	Terrell	Justine	H	615	322-9870
	Tirpin	Khaleed	G	723	123-9876
	Williams	George		615	290-2556
*	0				

Trong kết quả này khách hàng có tên là Dunne và Ołowski xuất hiện trong cả hai bảng vì vậy chỉ xuất hiện một lần trong bảng kết quả. Còn nếu sử dụng UNION ALL thì hai khách hàng đó sẽ xuất hiện hai lần trong bảng kết quả.

Toán tử giao có cú pháp như sau *query INTERSECT query*

Access không hỗ trợ toán tử giao. Để thực hiện phép toán này trong Access bạn cần sử dụng toán tử IN như sau

```

SELECT CUS_CODE
FROM CUSTOMER
WHERE CUS_AREACODE = '615' AND
      CUS_CODE IN (SELECT DISTINCT CUS_CODE
                    FROM INVOICE);
  
```

Một ví dụ về phép giao hai bảng CUSTOMER và INVOICE

	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
▶	10010	Ramas	Alfred	A	615	844-2573	\$0.00
▶	10011	Dunne	Leona	K	713	894-1238	\$0.00
▶	10012	Smith	Kathy	W	615	894-2285	\$345.86
▶	10013	Olowski	Paul	F	615	894-2180	\$536.75
▶	10014	Orlando	Myron		615	222-1672	\$0.00
▶	10015	O'Brian	Amy	B	713	442-3381	\$0.00
▶	10016	Brown	James	G	615	297-1228	\$221.19
▶	10017	Williams	George		615	290-2556	\$768.93
▶	10018	Farriss	Anne	G	713	382-7185	\$216.55
▶	10019	Smith	Olette	K	615	297-3809	\$0.00
*	0						\$0.00

	INV_NUMBER	CUS_CODE	INV_DATE
▶	1001	10014	16-Jan-04
▶	1002	10011	16-Jan-04
▶	1003	10012	16-Jan-04
▶	1004	10011	17-Jan-04
▶	1005	10018	17-Jan-04
▶	1006	10014	17-Jan-04
▶	1007	10015	17-Jan-04
▶	1008	10011	17-Jan-04
*	0	0	

	CUS_CODE
▶	10012
▶	10014
*	0

Toán tử trừ có cú pháp *query MINUS query*

Access cũng không hỗ trợ phép toán này. Để thực hiện nó, bạn cần sử dụng toán tử NOT IN. Hầu hết các hệ quản trị đều để tên phép toán này là EXCEPT.

```

SELECT CUS_CODE, CUS_LNAME
FROM CUSTOMER
WHERE CUS_AREACODE = '615' AND
CUS_CODE NOT IN (SELECT DISTINCT CUS_CODE
FROM INVOICE);

```

Một ví dụ về phép trừ được thể hiện trong hình vẽ dưới đây

	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
▶	10010	Ramas	Alfred	A	615	844-2573	\$0.00
▶	10011	Dunne	Leona	K	713	894-1238	\$0.00
▶	10012	Smith	Kathy	W	615	894-2285	\$345.86
▶	10013	Olowski	Paul	F	615	894-2180	\$536.75
▶	10014	Orlando	Myron		615	222-1672	\$0.00
▶	10015	O'Brian	Amy	B	713	442-3381	\$0.00
▶	10016	Brown	James	G	615	297-1228	\$221.19
▶	10017	Williams	George		615	290-2556	\$768.93
▶	10018	Farris	Anne	G	713	382-7185	\$216.55
▶	10019	Smith	Olette	K	615	297-3809	\$0.00
*	0						\$0.00

	INV_NUMBER	CUS_CODE	INV_DATE
▶	1001	10014	16-Jan-04
▶	1002	10011	16-Jan-04
▶	1003	10012	16-Jan-04
▶	1004	10011	17-Jan-04
▶	1005	10018	17-Jan-04
▶	1006	10014	17-Jan-04
▶	1007	10015	17-Jan-04
▶	1008	10011	17-Jan-04
*	0	0	

	CUS_CODE	CUS_LNAME
▶	10010	Ramas
	10013	Olowski
	10016	Brown
	10017	Williams
	10019	Smith
*	0	

Các phép kết nối của SQL

Các phép toán kết nối của SQL trộn các hàng từ hai bảng và trả về các hàng mà

1. Có các giá trị giống nhau cho các cột chung (kết nối tự nhiên) hoặc
2. Thoả mãn một điều kiện kết nối nào đó (bằng nhau hoặc không bằng nhau) hoặc
3. Có các giá trị giống nhau tại các cột chung hoặc có cả các giá trị không bằng nhau

Chúng ta đã xem xét các dạng cơ bản của một kết nối SQL trong đó có hai bảng được liệt kê tại mệnh đề FROM và mệnh đề WHERE thể hiện điều kiện kết nối. Một ví dụ cho dạng cơ bản của phép kết nối này được thể hiện sau đây

```
SELECT P_CODE, P_DESCRIP, P_PRICE, V_NAME
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE
```

Mệnh đề FROM thể hiện các bảng được kết nối. Nếu ba hoặc nhiều hơn 3 bảng được kết nối thì các phép kết nối sẽ thực hiện hai bảng tại một thời điểm, bắt đầu từ trái sang phải. Điều kiện kết nối được thể hiện trong mệnh đề WHERE, với ví dụ trên một kết nối tự nhiên được thể hiện trên thuộc tính V_CODE. Cú pháp của phép kết nối được thể hiện ở trên đôi khi được cho là một phép kết nối được viết theo kiểu cỏ điển. Bảng tóm tắt cá phép toán kết nối được thể hiện dưới đây

Phân loại kết nối	Kiểu kết nối	Ví dụ về cú pháp SQL	Mô tả
Cross	CROSS JOIN	<code>SELECT * FROM T1, T2;</code>	Kiểu cỏ điển. Trả về tích Đè các của T1 và T2
		<code>SELECT * FROM T1 CROSS JOIN T2;</code>	Kiểu mới. Trả về tích Đè các của T1 và T2
Inner	Old Style JOIN	<code>SELECT * FROM T1, T2 WHERE T1.C1 = T2.C1</code>	Trả về các hàng thoả mãn điều kiện kết nối trong mệnh đề WHERE-kiểu cỏ điển. Chỉ những hàng có giá trị thích hợp mới được chọn.
	NATURAL JOIN	<code>SELECT * FROM T1 NATURAL JOIN T2</code>	Trả về các hàng có giá trị thích hợp tại các cột giống nhau. Các cột tương thích nhau phải có cùng tên và cùng kiểu dữ liệu.
	JOIN USING	<code>SELECT * FROM T1 JOIN T2 USING (C1)</code>	Chỉ trả về các hàng có giá trị thích hợp tại các cột được thể hiện trong mệnh đề USING.
	JOIN ON	<code>SELECT * FROM T1 JOIN T2 ON T1.C1 = T2.C1</code>	Chỉ trả về các hàng thoả mãn điều kiện kết nối được thể hiện trong mệnh đề ON.
Outer	LEFT JOIN	<code>SELECT * FROM T1 LEFT</code>	Trả về các hàng có giá trị thích hợp và bao gồm cả những hàng không thích

		OUTER JOIN T2 ON T1.C1= T2.C1	hợp trong bảng bên trái (T1).
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1= T2.C1	Trả về các hàng có giá trị thích hợp và bao gồm cả những hàng không thích hợp trong bảng bên phải (T2).
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1= T2.C1	. Trả về các hàng có giá trị thích hợp và bao gồm cả những hàng không thích hợp trong cả hai bảng bên phải và bên trái

Phép CROSS JOIN (kết nối chéo) trong SQL tương đương với một tích đề các trong đại số quan hệ chuẩn. Cú pháp của phép kết nối này như sau:

Theo kiểu cũ điển

```
SELECT column-list  
FROM table1, table2;
```

Cú pháp của phép kết nối tự nhiên như sau

Theo kiểu cũ điển

```
SELECT column-list  
FROM table1, table2  
WHERE table1.C1 = table2.C2;
```

Theo kiểu mới

```
SELECT column-list  
FROM table1 CROSS JOIN table2;
```

Theo kiểu mới

```
SELECT column-list  
FROM table1 NATURAL JOIN table2;
```

Phép kết nối tự nhiên sẽ thực hiện các nhiệm vụ sau đây

- Xác định các thuộc tính chung bằng cách tìm kiếm các thuộc tính có cùng tên và có kiểu dữ liệu tương thích.
- Chỉ chọn những hàng có chung giá trị tại các thuộc tính chung.
- Nếu không có các thuộc tính chung, trả về kết nối chéo của hai bảng.

Một sự khác nhau quan trọng giữa kết nối tự nhiên và cú pháp cổ điển là kết nối tự nhiên không đòi hỏi việc sử dụng định danh bảng trong các thuộc tính chung.

Cách thứ hai để thể hiện một kết nối là thông qua từ khoá USING. Câu truy vấn này sẽ trả về các hàng có giá trị tương ứng bằng nhau tại các cột được chỉ ra trong mệnh đề USING. Các cột được liệt kê trong mệnh đề USING phải xuất hiện trong cả hai bảng.

Cú pháp của câu lệnh này như sau

**SELECT *column-list*
FROM *table1* JOIN *table2* USING (*common-column*);**

Ví dụ:

**SELECT INV_NUMBER, P_CODE, P_DESCRPT, LINE_UNITS, LINE_PRICE
FROM INVOICE JOIN LINE USING (INV_NUMBER) JOIN PRODUCT USING
(P_CODE);**

Cũng như đối với câu lệnh kết nối tự nhiên, câu lệnh này không đòi hỏi phải có định danh tên bảng cho các thuộc tính. Trên thực tế, Oracle 9i sẽ thông báo lỗi nếu bạn đưa tên bảng vào mệnh đề USING.

Cả câu lệnh kết nối tự nhiên NATURAL JOIN và JOIN USING đều sử dụng tên của các thuộc tính chung trong các bảng cần kết nối. Một cách khác để thể hiện một kết nối khi các bảng không có thuộc tính chung là sử dụng câu lệnh JOIN ON. Câu truy vấn này sẽ trả về những bản ghi thoả mãn điều kiện. Điều kiện kết nối sẽ thường bao gồm một biểu thức so sánh bằng của hai cột. Các cột có thể có chung tên hoặc không nhưng hiển nhiên là phải có kiểu dữ liệu tương thích nhau. Cú pháp của câu lệnh này như sau:

**SELECT *column-list*
FROM *table1* JOIN *table2* ON *join-condition*;**

Một ví dụ

**SELECT INVOICE.INV_NUMBER, P_CODE, P_DESCRPT, LINE_UNITS,
LINE_PRICE

FROM INVOICE JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER
JOIN PRODUCT ON LINE.P_CODE = PRODUCT.P_CODE;**

Chú ý trong ví dụ trên, không giống như NATURAL JOIN và JOIN USING, mệnh đề JOIN ON đòi hỏi sử dụng định danh bảng cho các thuộc tính chung. Nếu bạn không đưa định danh bảng vào bạn sẽ nhận được một thông báo lỗi “column ambiguously defined” (cột được xác định không rõ ràng).

Luôn nhớ rằng cú pháp của JOIN ON cho phép bạn thực hiện một phép kết nối thậm chí khi các bảng không có tên thuộc tính chung. Ví dụ để tạo ra một danh sách tất cả các nhân viên cùng với tên của người quản lý bạn có thể sử dụng truy vấn để quy dùng mệnh đề JOIN ON như dưới đây:

**SELECT E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME
FROM EMP E JOIN EMP M ON E.EMP_MGR = M.EMP_NUM**

ORDER BY E.EMP_MGR;

Các phép kết nối ngoài OUTER JOIN

Chúng ta đã xem xét các dạng của kết nối ngoài bên trái và kết nối ngoài bên phải trong các bài giảng trước. Ngoài ra còn có phép kết nối ngoài đầy đủ trong SQL. Một phép kết nối ngoài đầy đủ sẽ trả về không chỉ các hàng thoả mãn điều kiện kết nối (đó là những bản ghi có các giá trị bằng nhau tại những cột chung) nhưng cũng gồm tất cả các hàng với giá trị không bằng nhau ở cả hai bảng bên trái và bên phải. Cú pháp của phép kết nối ngoài đầy đủ như sau

```
SELECT column-list
FROM table1 FULL [OUTER] JOIN table2 ON join-condition;
```

Ví dụ sau đây của phép kết nối ngoài đầy đủ sẽ liệt kê mã sản phẩm, mã người cung cấp và tên người cung cấp của tất cả các sản phẩm và bao gồm cả những hàng (sản phẩm không của các nhà cung cấp) và tất cả những hàng của nhà cung cấp (mà không có sản phẩm nào cả).

```
SELECT P_CODE, VENDOR.V_CODE, V_NAME
FROM VENDOR FULL OUTER JOIN PRODUCT
ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

Truy vấn con và truy vấn tương hỗ

Việc sử dụng các phép kết nối cho phép một hệ quản trị cơ sở dữ liệu có thể lấy thông tin từ hai hoặc nhiều bảng. Dữ liệu lấy từ các bảng được xử lý đồng thời.

Trên thực tế, thông thường các truy vấn hay cần xử lý dữ liệu dựa trên các dữ liệu đã được xử lý trước. Ví dụ, giả sử bạn muốn tạo ra một danh sách các nhà cung cấp có cung cấp các sản phẩm. (Nhắc lại rằng không phải tất cả các nhà cung cấp trong bảng VENDOR đều cung cấp sản phẩm-một vài trong số đó chỉ là những nhà cung cấp trong tương lai). Câu truy vấn thực hiện yêu cầu này như sau:

```
SELECT V_CODE, V_NAME
FROM VENDOR
WHERE V_CODE NOT IN (SELECT V_CODE FROM PRODUCT);
```

Một truy vấn con là một truy vấn dạng câu lệnh SELECT bên trong một truy vấn khác. Nó thường được thể hiện bên trong một dấu ngoặc. Câu truy vấn đầu tiên trong câu lệnh SQL được biết đến là câu truy vấn bên ngoài, câu truy vấn thứ hai là câu truy vấn bên trong. Câu truy vấn bên trong bao giờ cũng được thực thi đầu tiên, kết quả của câu truy vấn bên trong được sử dụng như đầu vào cho câu truy vấn bên ngoài. Toàn bộ câu lệnh SQL kiểu như vậy

được biết đến là truy vấn lồng nhau.Câu truy vấn bên trong còn được gọi là câu truy vấn con.

Một truy vấn con có thể trả về một trong các loại kết quả sau đây:

- 1- Một giá trị đơn (một cột hoặc một hàng). Câu truy vấn con này có thể được dùng ở bất kỳ nơi nào một giá trị đơn được sử dụng. Ví dụ, ngay ở bên phải của biểu thức so sánh.
- 2- Một danh sách các giá trị (một cột hoặc nhiều hàng). Kiểu truy vấn con này có thể được dùng ở bất kỳ nơi nào mà một danh sách các giá trị được sử dụng. Ví dụ khi dùng mệnh đề IN.
- 3- Một bảng ảo (nhiều cột, nhiều hàng gồm tập các giá trị). Câu truy vấn con loại này có thể được sử dụng ở bất kể nơi nào cần một bảng trong câu lệnh. Ví dụ trong câu lệnh FROM
- 4- Không có giá trị nào cả hay giá trị NULL. Trong những trường hợp kiểu này đầu ra của truy vấn bên ngoài có thể cho kết quả là một thông báo lỗi hoặc một tập rỗng, phụ thuộc vào vị trí của câu truy vấn đó được sử dụng (trong một phép so sánh, một biểu thức hoặc một tập các bảng)

Loại truy vấn con phổ biến nhất sử dụng một câu truy vấn con SELECT bên trong ở bên phải của một biểu thức so sánh ở mệnh đề WHERE. Ví dụ, để tìm thấy tất cả các sản phẩm có giá lớn hơn hoặc bằng giá trung bình của một sản phẩm, câu truy vấn sau đây là cần thiết

```
SELECT P_CODE, P_PRICE  
FROM PRODUCT  
WHERE P_PRICE >= (SELECT AVG(P_PRICE)  
                    FROM PRODUCT);
```

Câu truy vấn con cũng có thể được sử dụng kết hợp với các phép kết nối. Câu truy vấn dưới đây liệt kê tất cả các khách hàng đặt hàng sản phẩm “Claw hammer”.

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME  
FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)  
              JOIN LINE USING (INV_NUMBER)  
              JOIN PRODUCT USING (P_CODE)  
WHERE P_CODE = (SELECT P_CODE  
                  FROM PRODUCT  
                  WHERE P_DESCRIP = “Claw hammer” );
```

Lưu ý rằng câu truy vấn trên cũng có thể được viết như sau:

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME  
FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)
```

JOIN LINE USING (INV_NUMBER)
JOIN PRODUCT USING (P_CODE)

WHERE P_DESCRIP = ‘Claw hammer’);

Tuy nhiên, điều gì sẽ xảy ra nếu hai hoặc nhiều hơn phần mô tả sản phẩm có chứa chuỗi “Claw hammer”. Bạn sẽ nhận được một thông báo lỗi bởi vì về phái của biểu thức so sánh chỉ có thể nhận một giá trị.

Để giải quyết vấn đề vừa nêu trên, chúng ta sử dụng toán hạng IN. Câu truy vấn dưới đây sẽ liệt kê tất cả các khách hàng đặt bất kể loại hàng nào là hammer hoặc saw

```
SELECT DISTINCT CUS_CODE, CUS_LNAME, CUYS_FNAME  
FROM CUSTOMER JOIN INVOICE USING (CUS_CODE)  
    JOIN LINE USING (INV_NUMBER)  
    JOIN PRODUCT USING (P_CODE)  
  
WHERE P_CODE IN (SELECT P_CODE  
    FROM PRODUCT  
    WHERE P_DESCRIP LIKE ‘%hammer%’  
    OR P_DESCRIP LIKE ‘%saw%’ );
```

Câu truy vấn con cũng có thể được sử dụng trong mệnh đề HAVING. Gợi nhớ rằng mệnh đề HAVING được sử dụng để hạn chế đầu ra của một truy vấn GROUP BY bằng cách áp dụng các tiêu chí điều kiện đối với các bản ghi đã được gộp nhóm. Ví dụ, truy vấn liệt kê tất cả các sản phẩm với tổng số lượng bán được lớn hơn lượng bán trung bình của các mặt hàng được thể hiện như dưới đây

```
SELECT DISTINCT P_CODE, SUM(LINE_UNITS)  
FROM LINE  
GROUP BY P_CODE  
  
HAVING SUM(LINE_UNITS) > (SELECT AVG(LINE_UNITS) FROM LINE);
```

Trong câu truy vấn con IN sử dụng một toán tử bằng, điều đó có nghĩa là nó chỉ chọn các hàng thích hợp với ít nhất một trong các giá trị trong danh sách. Điều gì sẽ xảy ra nếu bạn cần thực hiện một so sánh không bằng của một giá trị với một danh sách các giá trị?. Lấy ví dụ, bạn muốn biết sản phẩm nào có giá lớn hơn giá của mọi sản phẩm mà do nhà cung cấp ở Florida cung cấp. Câu truy vấn sau đây giải quyết điều đó

```
SELECT P_CODE, P_ONHAND*P_PRICE  
FROM PRODUCT  
WHERE P_ONHAND*P_PRICE > ALL (SELECT P_ONHAND*P_PRICE  
    FROM PRODUCT)
```

```
WHERE V_CODE IN (SELECT V_CODE  
                  FROM VENDOR  
                  WHERE V_STATE=  
'FL'));
```

Trong tất cả các trường hợp sử dụng truy vấn con mà chúng ta đã xem xét từ trước đến giờ, câu truy vấn con là một phần của biểu thức điều kiện và nó luôn xuất hiện ở về phải của một biểu thức. Đó là trường hợp của truy vấn con ở WHERE, HAVING, và IN cũng như toán tử ANY và ALL.

Nhắc lại rằng mệnh đề FROM dùng để liệt kê các bảng cần để lấy dữ liệu ra. Bởi vì đầu ra của một câu lệnh SELECT là một bảng khác (hoặc chính xác hơn là một bảng ảo khác), bạn có thể sử dụng câu truy vấn con SELECT trong mệnh đề FROM. Ví dụ, bạn muốn biết tất cả các khách hàng những người đã mua sản phẩm 13-Q2/P2 và 23109-HB. Vì tất cả việc mua bán các sản phẩm được lưu trong bảng LINE nên dễ dàng tìm thấy khách hàng mua bất kỳ một sản phẩm nào bằng cách tìm kiếm thuộc tính P_CODE trong bảng LINE. Tuy nhiên, trong trường hợp này, bạn muốn biết tất cả các khách hàng mua cả hai loại hàng chứ không chỉ một loại. Câu truy vấn được thể hiện như dưới đây:

```
SELECT DISTINCT CUSTOMER.CUS_CODE , CUSTOMER.LNAME  
FROM CUSTOMER, (SELECT INVOICE.CUS_CODE  
                  FROM INVOICE NATURAL JOIN LINE  
                  WHERE P_CODE = ' 13-Q2/P2' ) CP1,  
      (SELECT INVOICE.CUS_CODE  
                  FROM INVOICE NATURAL JOIN LINE  
                  WHERE P_CODE = '23109-HB' ) CP2  
WHERE CUSTOMER.CUS_CODE = CP1.CUS_CODE  
      AND CP1.CUS_CODE = CP2.CUS_CODE;
```