|  |
| --- |
| 4. PHỤ THUỘC HÀM |
| Chuẩn hoá 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 hoá  của một tập đoàn. Quá trình chuẩn hoá đầu tiên được phát triển bởi Codd năm 1972. Việc chuẩn hoá 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 hoá 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 ® B) nếu mỗi giá trị của A liên hệ tới duy nhất mọt 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. |
| 4.1. Mục đích của sự chuẩn hóa và 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 hoá 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 hoá cón hỗ trợ cho người thiết kế thông quan một chuỗi các kiểm thử mà có thể dùng cho mỗi quan hệ sao cho một lược đồ quan hệ có thể được chuẩn hoá 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.   |  | | --- | |  | |  | |  | |  | |
| 4.1.1. 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 riền 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 lcủa 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:   * Thuộc tính kết nối không tổn 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. * 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 bau đầu có bị vi phạm hay không. |
| 4.1.2. Thuộc tính kết nối không tổn thất thông tin |
| Xem xét lược đồ quan hệ SP sau đây và sự phân tách nó thành hai lược đồ S1 và S2.   |  | | --- | |  | |  | |
| 4.2. Bảo toàn các phụ thuộc hàm |
| Ví dụ R = (A, B, C) với  F = {AB → C, C → A} và  \fn_cm \gamma= {(B, C), (A, C)}. Rõ ràng là C → A được bảo toàn trong lược đồ quan hệ (A,C). Làm thế nào để AB → C được bảo toàn mà không cần kết nối hai lược đồ quan hệ  trong g. Câu trả lời là không thể vì thế các phụ thuộc hàm không được bảo toàn trong \fn_cm \gamma. |
| 4.2.1. 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, …., 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. |
| 4.2.2. 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. Mối 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 hàm 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. Đẻ 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 thieeys kế mức khái niệm. staff# → sname, position, salary, branch#, baddress branch# → baddress  baddress →  branch# branch#, position → salary baddress, position →  salary Thông thường trong  nhiều 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 *staff và branch* với các phụ thuộc hàm được xác định ở trên của quan hệ này được mô tả dưới đây   |  | | --- | |  | |  | |
| 4.2.3. 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 vế phải của phụ thuộc hàm là một tập con của vế 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:  { staff#, sname} → sname { staff#, sname} → staff#  Mặc dầu 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. |
| 4.2.4. 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. |
| 4.2.5. 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 ⊨ X → Y thể hiện rằng phụ thuộc hàm X → Y được suy diễn ra bởi tập phụ thuộc hàm F. Một cách chính thức,  F+ \fn_cm \equiv{X → Y | F ⊨ X → 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 \fn_cm \supseteq Y, thì X → Y Luật suy diễn IR2: luật tăng trưởng – nếu X → Y, thì XZ → YZ   Luật suy diễn IR3: luật bắc cầu – nếu X → Y và Y → Z, thì X → Z Luật suy diễn IR4: luật chiếu – nếu X → YZ, thì X → Y và X → Z Luật suy diễn IR5: luật cộng thêm – nếu X → Y và X → Z, thì X → YZ Luật suy diễn IR6: luật giả bắc cầu – nếu X → Y và YZ → W, thì XZ → W  Ba luật suy diễn đầu tiên được biết đến như hệ tiên đề Amstrong 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 → E, AG → J, BE → I, E → G, GI → H}.  Hỏi F ⊨ AB → GH ? **Chứng minh:**   1. AB → E, được cho trong F 2. AB → AB, dùng luật phản xạ IR1 3. AB → B, dùng luật chiếu IR4 từ bước 2 4. AB → BE, dùng luật cộng thêm IR5 từ các bước 1 và 3 5. BE → I, được cho sẵn trong F 6. AB → I, dùng luật IR3 từ bước 4 và 5 7. E → G, cho sẵn trong F 8. AB → G, dùng luật bắc cầu IR3 từ bước 1 và 7 9. AB → GI, dùng luật cộng thêm IR5 từ bước 6 và 8 10. GI → H, cho sẵn trong F 11. AB → H, dùng luật bắc cầu IR3 từ bước 9 và 10 12. AB → GH, dùng luật cộng thêm IR5 từ bước 8 và 11- kết quả được chứng minh |
| 4.3. Tính toán bao đóng |
| 4.3.1. 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 Amstrong 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 Amstrong 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 →C, C → 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 → 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 ⊨ X → Y không hoặc chính xác hơn là  X → 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 → 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. |
| 4.3.2. Thuật toán tính bao đóng |
| Đầu vào:  tập các thuộc tính X, và một tập các phụ thuộc hàm F Đầu ra: X+ trên  F Closure (X, F) {       X+ \fn_cm \leftarrow X;       repeat             oldX+ \fn_cm \leftarrowX+;             for mỗi phụ thuộc hàm W → Z trong F do                   if W \fn_cm \subseteq X+ then X+ \fn_cm \leftarrowX+ \fn_cm \cup Z;       until (oldX+ = X+); }  ***Ví dụ sử dụng thuật toán tính Bao đóng Closure***  Cho F = {A → D, AB → E, BI → E, CD → I, E → C}, Tìm (AE)+  vòng 1  X+ = {A, E} dùng A → D, A \fn_cm \subseteq  X+, vì vậy thêm D vào X+, X+ = {A, E, D} dùng AB → E, không  dùng BI → E, không dùng CD → I, không dùng E → C, EÍ 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 → D, có nhưng không có thay đổi dùng AB → E, không dùng BI → E, không dùng CD → I, CD \fn_cm \subseteqX+, vì vậy thêm I vào X+, X+ = {A, E, D, C, I} dùng E → 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 → D, có nhưng không có thay đổi  dùng AB → E, không  dùng BI → E, không dùng CD → I, có nhưng không có thay đổi dùng E → C, có nhưng không có thay đổi  Không có thay đổi xảy ra tới 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 → 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 vế 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 → Y Đầu ra: true nếu F⊨X→ Y, false nếu trái lại Member (F, X → Y) {      if Y \fn_cm \subseteqClosure(X,F)    then trả về true;      else trả về false; } |
| 4.4. 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 ® Y trong F. Nếu Y \fn_cm \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 dường 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). |
| 4.4.1. 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 \fn_cm \leftarrow G;      for mỗi phụ thuộc hàm X → Y \fn_cm \in G do           if Member(F – {X → Y}, X → Y)             then F ¬ F – {X → Y};       return (F); }  Ví dụ về việc sinh ra một phủ không dư thừa  Cho G = {A → B, B → A, B → C, A → C}, hãy  tìm một phủ không dư thừa của G. F \fn_cm \leftarrowG Member({B → A, B → C, A → C}, A → B)             Closure(A, {B → A, B → C, A → C})                         A+ = {A, C}, vì vậy A → B là không dư thừa Member({A → B, B → C, A → C}, B → A)             Closure(B, {A → B, B → C, A → C})                         B+ = {B, C}, vì vậy B → A là không dư thừa Member({A → B, B → A, A → C}, B → C)             Closure(B, {A → B, B → A, A → C})                         B+ = {B, A, C}, vì vậy B → C là dư thừa  F = F – {B → C}  Member({A → B, B → A}, A → C)             Closure(A, {A → B, B → A})                         A+ = {A, B}, vì vậy A → C là không dư thừa  **Return F = {A** → **B, B** → **A, A** → **C}**  Nếu G = {A → B, A → C, B → A, B → C}, giống hệt tập như trước nhưng với trật tự các phụ thuộc hàm khác. Một phủ khác sẽ được sinh ra!  F \fn_cm \leftarrowG Member({A → C, B → A, B → C}, A → B)             Closure(A, {A → C, B → A, B → C})                         A+ = {A, C}, vì vậy A → B là không dư thừa Member({A → B, B → A, B → C}, A → C)             Closure(A, {A → B, B → A, B → C})                         A+ = {A, B, C}, vì vậy A → C là dư thừa F = F – {A → C} Member({A → B, B → C}, B → A)             Closure(B, {A → B, B → C})                         B+ = {B, C}, vì vậy B → A là không dư thừa  Member({A → B, B → A}, B → C)             Closure(B, {A → B, B → A})                         B+ = {B, A}, vì vậy B → C là không dư thừa **Return F = {A** → **B, B** → **A, B** → **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 → B, B → A, B → C, A → C} thì F = {A → B, B → A, AB → C} là một phủ không dư thừa của G tuy nhiên F chứa những phụ thuộc hàm không thuộc G. |
| 4.4.2. 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 → Y \fn_cm \in F thì thuộc tính A được gọi là dư thừa trong X → Y \fn_cm \inF nếu:  X = AZ, X \fn_cm \neq Z và {F – {X → Y}} \fn_cm \cup {Z → Y} \fn_cm \equiv F, hoặc  Y = AW, Y \fn_cm \neqW và {F – {X → Y}} \fn_cm \cup  {X → W} \fn_cm \equivF  Nói một cách khác, một thuộc tính A là dư thừa trong X → 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 → BC, B → C, AB → D, thuộc tính C là dư thừa trong vế phải của A → BC vì  A+ = {A, B, C, D} khi F = F – {A → C}. Tương tự như vậy, B là thuộc tính dư thừa của vế trái của AB → D vì AB+ = {A, B, C, D} khi F= F – {AB → D}. |
| 4.4.3. 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 → Y \fn_cm \in F.  X → 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 → Y được gọi là tối giản phải nếu Y không chứa các thuộc tính dư thừa A. X → Y được gọi là tối giản nếu nó tối giản trái và tối giản phải và Y khác rỗng. Thuật toán tối giản về 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 \fn_cm \leftarrowG;       for mỗi phụ thuộc hàm X → Y trong G do             for mỗi thuộc tính A trong X do                   if Member(F, (X-A) → Y)                         then loại bỏ A khỏi X trong X → Y của F        return(F); }  Thuật toán tối giản vế phải dưới đây sẽ sinh ra một tập các phụ thuộc hàm tối giản vế phải.  Thuật toán Right-Reduce  {trả về một bản tối giản vế phải của F} Đầu vào:  tập các phụ thuộc hàm G Đầu ra: một phủ tối giản phải của G  Right-Reduce (G) {       F \fn_cm \leftarrowG;       For mỗi phụ thuộc hàm X → Y trong G do             for mỗi thuộc tính A trong Y do                   if Member(F – {X → Y} \fn_cm \cup{X → (Y- A)}, X → A)                         then loại bỏ A khỏi Y từ X → 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 \fn_cm \leftarrowRight-Reduce( Left-Reduce(G));       Loại bỏ tất cả các phụ thuộc hàm có dạng X → null từ F  // nếu G chứa một phụ thuộc hàm dư thừa X → Y thì mọi thuộc tính trong Y sẽ là dư thừa vì vậy sẽ tối giản tới X → 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 vế trái trước rồi mới tối giản vế phải. Ví dụ sau mô tả những gì xảy ra nếu trật tự này bị vi phạm. Ví dụ: Cho G = {B → A , D → A , BA → D}, G là tối giản phải nhưng không phải tối giản trái. Nếu chúng ta tối giản trái G để sinh ra F = {B → A , D → A , B → D}, chúng ta có F sẽ tối giản trái nhưng không tối giản phải. B → A là dư thừa bên vế phải vì B → D  → A. |
| 4.4.4. 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 vế phải là một thuộc tính 2. F là không dư thừa 3. Không có phụ thuộc hàm nào dạng X ® A có thể được thay thế bởi một trong dạng Y → A với Y \fn_cm \subseteq X và vẫn là một tập tương đương, nói cách khác F là tối giản trái.   Ví dụ: G = {A → BCE,  AB → DE,  BI → J} thì một phủ tối thiểu của G là F = {A → B, A → C, A → D, A → E, BI → 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 \fn_cm \leftarrow F;      Thay thế mỗi phụ thuộc hàm X → A1A2...An trong G bởi n phụ thuộc hàm X → A1, X → A2,..., X → An       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. |