|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Họ tên** | **MSSV** | **Hình ảnh** | **Ghi chú** |
| 1 | Hà Thị Phương Thảo | 0912430 | Description: 0912430 |  |
| 2 | Nguyễn Thị Thanh Thảo | 0912431 | Description: 0912431 |  |
| 3 | Trương Nguyễn Thủy Tiên | 0912463 | Description: 0912463 |  |
| 4 | Nguyễn Văn Tiến | 0912469 | Description: 0912469 | Nhóm trưởng |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Nội dung công việc** | **Thành viên 1**  **0912430** | **Thành viên 2**  **0912431** | **Thành viên 3**  **0912463** | **Thành viên 4**  **0912469** |
| 1 | Dịch tài liệu |  |  |  |  |
| 2 | Tổng hợp kết quả, viết tài liệu |  |  |  |  |

### Topic #8b – Cơ sở dữ liệu hướng đối tượng

Mục lục

[I. DẪN NHẬP: 5](#_Toc309861610)

[II. CÁC LOẠI HỆ THỐNG CƠ SỞ DỮ LIỆU ĐỐI TƯỢNG:.](#_Toc309861611) 5

[III. Nội dung trình bày: 6](#_Toc309861617)

[1. Minh họa lý do tại sao lại mở rộng mô hình quan hệ: 6](#_Toc309861619)

1.[1. Kiểu dữ liệu mới.](#_Toc309861620) 7

[1.2 Thao tác với các loại dữ liệu mới 8](#_Toc309861619)

[2. Thảo luận về kiểu dữ liệu trừu tượng, định nghĩa và thao tác. 1](#_Toc309861620)0

2.[1. Định nghĩa phướng thức của kiểu dữ liệu trừu tượng 1](#_Toc309861619)2

[3. Cách thức và cấu trúc của kiểu dữ liệu trừu tượng. 1](#_Toc309861620)2

3.[1. Thao tác ở kiểu dữ liệu có cấu trúc 1](#_Toc309861619)2

[4. Xem xét và nhận dạng đối tượng.. 1](#_Toc309861620)2

4.[1. Khái niệm về Equality 1](#_Toc309861619)3

4.[2. Bất tham chiếu đến các loại tham chiếu. 1](#_Toc309861620)4

[5. Sự kế thừa và phân cấp các loại. 1](#_Toc309861619)4

[5.1. Định nghĩa các loại với sự kế thừa. 1](#_Toc309861620)4

5.[2. Tính ràng buộc của những phương thức 1](#_Toc309861619)5

[5.3. Tập các hệ thống phân cấp, loại mở rộng, và truy vấn. 1](#_Toc309861620)5

[6. Xem xét làm thế nào để tận dụng lợi thế của khái niệm hướng đối tượng mới để thiết kế cơ sở dữ liệu ORDBMS 1](#_Toc309861619)6

[6.1. Loại có cấu trúc và loại trừu tượng. 1](#_Toc309861620)6

[6.2. Định danh đối tượng 1](#_Toc309861620)7

[6.2. Mở rộng mô hình thực thể kết hợp (ER) 1](#_Toc309861619)8

[6.3. Sử dụng tập Nested(Lồng nhau). 1](#_Toc309861620)9

[7. Thảo luận một số thách thức mới đặt ra bởi hệ thống quan hệ đối tượng. 1](#_Toc309861619)9

[7.1. Lưu trữ và truy cập phương thức 1](#_Toc309861620)9

7.[2. Chỉ mục trên loại dữ liệu mới 1](#_Toc309861619)9

7.[3. Xử lý truy vấn](#_Toc309861620) 20

7.4[. Tối ưu truy vấn 21](#_Toc309861619)

[8. Thảo luận về ODL và OQL, các tiểu chuẩn cho OODBMSs. 2](#_Toc309861620)2

[8.1. Mô hình dữ liệu ODMG và ODL 2](#_Toc309861619)2

[8.2. OQL 2](#_Toc309861620)2

[9. Trình bày so sánh ngắn gọn của ORDBMSs và OODBMs 3](#_Toc309861619)3

[9.1. So sánh RDBMS và ORDBMS 2](#_Toc309861620)3

9.[2. So sánh OODBMS và ORDBMS 2](#_Toc309861619)3

9.[3. So sánh OODBMS và ORDBMS 2](#_Toc309861620)3

**CƠ SỞ DỮ LIỆU ĐỐI TƯỢNG**

**Nội dung trình bày:**

* Từ phần 25.1 đến 25.5 tạo động cơ và giới thiệu khái niệm về đối tượng. Các khải niệm được thảo luận trong các phần phổ biến cho cả OODBMSs và ORDBMSs, mặc dù cứ pháp tương tự SQL: 1999. Cụ thể:
  + Phần 25.1: Minh họa lý do tại sao lại mở rộng mô hình quan hệ.
  + Phần 25.2: Thảo luận về kiểu dữ liệu trừu tượng, định nghĩa và thao tác.
  + Phần 25.3: Cách thức và cấu trúc của kiểu dữ liệu trừu tượng.
  + Phần 25.4: Xem xét và nhận dạng đối tượng.
  + Phần 25.5: Sự kế thừa và phân cấp các loại.
* Phần 25.6: Xem xét làm thế nào để tận dụng lợi thế của khái niệm hướng đối tượng mới để thiết kế cơ sở dữ liệu ORDBMS.
* Phần 25.7: Thảo luận một số thách thức mới đặt ra bởi hệ thống quan hệ đối tượng.
* Phần 25.8: Thảo luận về ODL và OQL, các tiểu chuẩn cho OODBMSs.
* Phần 25.9: Trình bày so sánh ngắn gọn của ORDBMSs và OODBMs.

1. **DẪN NHẬP:**

* Với mô hình, hệ thống cơ sở dữ liệu quan hệ hỗ trợ rất tốt các kiểu dữ liệu nhỏ, các tập cố định như (integer, dates, strings), Trong đó có cho thấy hỗ trợ đầy đủ cho các lĩnh vực ứng dụng truyền thống như xử lý dữ liệu hành chính. Trong nhiều lĩnh vực ứng dụng, tuy nhiên, các loại dữ liệu phức tạp hơn nhiều cần được xử lý. Thông thường các loại dữ liệu phức tạp được lưu trữ trong một hệ thống file lưu trữ. Ví dụ về các lĩnh vực với các dữ liệu phức tạp bao gồm hỗ trợ thiết kế và mô hình hóa máy tính (CAD / CAM), kho lưu trữ đa phương tiện, và quản lý tài liệu.
* Với một lượng lớn dữ liệu ngày càng tăng nhanh chóng, một số tính năng từ DBMS. Ví dụ: giảm thời gian phát triển ứng dụng, đồng thời kiểm soát và phục hồi dữ liệu, hỗ trợ lập chỉ mục, và khả năng truy vấn ngày càng trở nên cần thiết và hấp dẫn. Để hỗ trợ các ứng dụng như vậy, một DBMS phải hỗ trợ các kiểu dữ liệu phức tạp. Khái niệm hướng đối tượng có ảnh hướng mạnh mẻ đến những nổ lực tăng cường hỗ trợ cơ sở liệu phức tạp và dẫn đến sự phát triển của hệ thống cơ sở dữ liệu đối tượng.

1. **CÁC LOẠI HỆ THỐNG CƠ SỞ DỮ LIỆU ĐỐI TƯỢNG:**

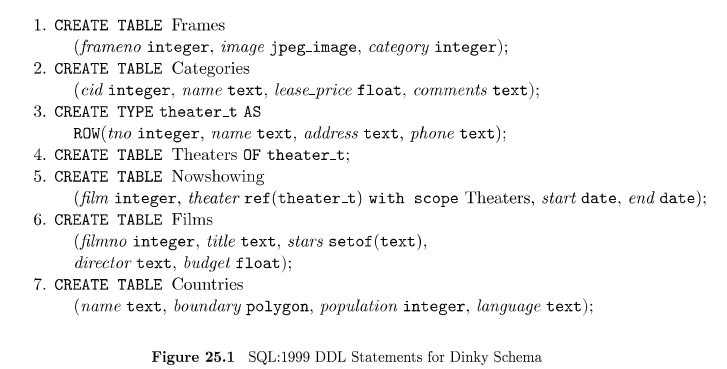
* Hệ thống cơ sở dữ liệu đối tượng có 2 hướng phát triển khác nhau:
  + Hệ thống cơ sở dữ liệu hướng đối đối tượng: hệ thống cơ sở dữ liệu hướng đối tượng là những lĩnh vực nơi các đối tượng phức tạp đóng một vai trò trung tâm. Cách tiếp cận này bị ảnh hưởng nhiều bởi các ngôn ngữ lập trình hướng đối tượng và có thể được hiểu như là một cố gắng để thêm chức năng  DBMS một môi trường ngôn ngữ lập trình.
  + Hệ thống cơ sở dữ liệu quan hệ đối tượng:  Nó có thể được coi là một nỗ lực để mở rộng hệ thống cơ sở dữ liệu quan hệ với các chức năng cần thiết để hỗ trợ một class rộng lớn hơn của các ứng dụng, và bằng nhiều cách, cung cấp một cầu nối giữa các mô hình quan hệ và hướng đối tượng.

🡪Trong seminar này sẽ tập trung vào cơ sở dữ liệu quan hệ đối tượng và nhấn mạnh làm thế nào nó có thể được xem như là một sự phát triển của RDMSs, chứ không phải là một mô hình hoàn toàn khác nhau.

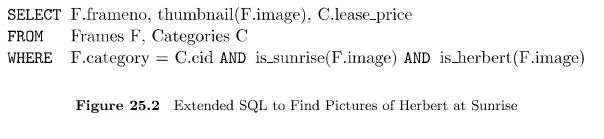
* SQL: tiêu chuẩn năm 1999 là dựa trên mô hình ORDBMS, chứ không phải là mô hình OODBMS. Tiêu chuẩn này bao gồm hỗ trợ cho rất nhiều các tính năng kiểu dữ liệu phức tạp được thảo luận trong chương này. Bài seminar này tập trung vào việc phát triển các khái niệm cơ bản, chứ không phải là trình bày SQL: 1999, một số trong những tính năng mà chúng tôi thảo luận là không bao gồm trong SQL: 1999. Chúng tôi đã cố gắng để phù hợp với SQL: năm 1999 với ký hiệu, mặc dù chúng tôi có đôi khi tách ra một chút cho rõ ràng cho rõ ràng. Điều quan trọng là nhận ra rằng các khái niệm chính được thảo luận phổ biến cho cả hai ORDBMSs và OODBMSs, và chúng tôi thảo luận làm thế nào họ được hỗ trợ trong tiêu chuẩn ODL / OQL được đề xuất cho OODBMSs tại Mục VIII.
* Các nhà cung cấp RDBMS bao gồm cả IBM, Informix, Oracle, có thêm chức năng ORDBMS (mức độ khác nhau) trong các sản phẩm của họ, và điều quan trọng là nhận ra làm thế nào các cơ quan hiện có của kiến thức về thiết kế và thực hiện các cơ sở dữ liệu quan hệ có thể được thừa hưởng để giải quyết các ORDBMS phần mở rộng. Nó cũng rất quan trọng để hiểu những thách thức và cơ hội mà các phần mở rộng trình bày cho người sử dụng cơ sở dữ liệu, thiết kế, và người thực hiện.

1. **Nội dung trình bày:**
2. **Minh họa lý do tại sao lại mở rộng mô hình quan hệ:**

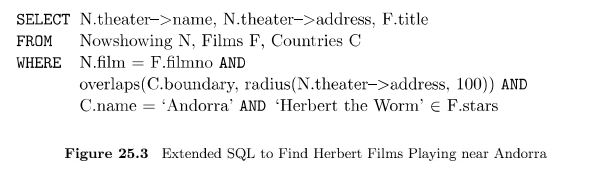
* Như một ví dụ cụ thể về sự cần thiết cho các hệ thống quan hệ đối tượng, chúng tôi tập trung vào một vấn đề kinh doanh xử lý dữ liệu mới khó hơn và (theo quan điểm của tác giả ) giải trí nhiều hơn so với đồng đô la và xu sổ sách kế toán của thập kỷ trước.
* Ngày nay, các công ty trong các ngành công nghiệp như giải trí trong kinh doanh bán dữ liệu số, tài sản cơ bản của công ty của họ không phải là sản phẩm hữu hình, nhưng thay vì các đồ tạo tác phần mềm như video và âm thanh.
* Chúng tôi xem xét Công ty tưởng tượng về Giải trí dinky, một tập đoàn lớn Hollywood có tài sản chính là một bộ sưu tập của các nhân vật hoạt hình, đặc biệt là Herbert âu yếm và quốc tế yêu quý Worm.Dinky có một số bộ phim Herbert Worm, nhiều trong số đó đang được thể hiện trong các rạp chiếu phim trên toàn thế giới tại bất kỳ thời điểm nào. Dinky cũng làm cho một thỏa thuận tốt về tiền bạc cấp giấy phép Herbert của hình ảnh, giọng nói, và đoạn băng video cho các mục đích khác nhau, con số hành động, trò chơi video, xác nhận sản phẩm, và như vậy. Cơ sở dữ liệu của Dinky được sử dụng để quản lý việc bán hàng và hồ sơ cho thuê cho các sản phẩm khác nhau liên quan đến Herbert, cũng như video và âm thanh tạo nên nhiều bộ phim của Herbert.
  1. **Kiểu dữ liệu mới**
* Một vấn đề cơ bản phải đối mặt khi thiết kế cơ sở dữ liệu của Dinky là cần hỗ trợ các loại dữ liệu phong phú hơn đáng kể hơn là có sẳn trong một DBMS:
  + **Người dùng định nghĩa kiểu dữ liệu trừu tượng (ADTs):** Những dữ liệu của Dinky bao gồm hình ảnh, giọng nói, video của Herbert và chúng phải được lưu trữ trong cơ sở dữ liệu. Hơn nữa chúng ta cần các chức năng đặc biệt để thao tác với các đối tượng này. Vi dụ như chúng ta có thể viết các chức năng tạo ra bản nén của một hình ảnh hoặc hình ảnh có độ phân giải thấp hơn.
  + **Kiểu dữ liệu có cấu trúc:** Trong các ứng dụng quả thực chúng ta cần xây dựng loại mới dựa vào các hàm dựng như sets, tuples, arrays, sequences.
  + **Tính kế thừa:** Khi số lượng các loại dữ liệu phát triển, điều quan trọng là phải nhận ra sự phổ biến giữa các loại và tận dụng lợi thế của nó. Ví dụ nén hình ảnh và hình ảnh có độ phân giải thấp là 2 loại, nhưng ở mức độ nào đó, chỉ cần 1 loại là đủ. Do đó, mong muốn kế thừa một số tính năng của các đối tượng hình ảnh khi xác định các đối tượng hình ảnh nén và các đối tượng có độ phân giải thấp.
* Làm thế nào chúng ta có thể giải quyết những vấn đề này trong một RDBMS? Chúng ta có thể lưu trữ hình ảnh, video, và như vậy là BLOB trong hệ thống hiện tại quan hệ, một nhị phân lớn đối tượng (BLOB) là chỉ một dòng dài byte, và việc của DBMS hỗ trợ lưu trữ và lấy BLOB trong cách một ví dụ mà người dùng không phải lo lắng về kích thước của BLOB, một BLOB có thể chiều dài một vài trang, không giống như một thuộc tính truyền thống. Tất cả các quá trình gia công, chế biến thêm BLOB đã được thực hiện bằng chương trình ứng dụng của người sử dụng, trong ngôn ngữ máy chủ lưu trữ trong đó các mã SQL được nhúng vào. Giải pháp này không hiệu quả bởi vì chúng ta buộc phải lấy tất cả các BLOB trong một bộ sưu tập ngay cả trong số họ có thể được lọc ra câu trả lời bằng cách áp dụng chức năng người dùng định nghĩa (trong DBMS). Nó là không thỏa đáng từ một quan điểm nhất quán dữ liệu, bởi vì ngữ nghĩa của dữ liệu là phụ thuộc nhiều vào mã ứng dụng ngôn ngữ sở tại và không thể được thi hành bởi DBMS
* Đối với các loại cấu trúc và thừa kế, chỉ đơn giản là không có hỗ trợ trong mô hình quan hệ. Chúng ta buộc phải ánh xạ dữ liệu với cấu trúc phức tạp như vậy vào một bộ sưu tập các bảng phẳng. (Chúng ta đã thấy ví dụ của ánh xạ như vậy khi chúng tôi thảo luận về các bản dịch từ sơ đồ ER, thừa kế cho mối quan hệ tại Chương 2.)
* Ứng dụng này rõ ràng đòi hỏi các tính năng không có sẵn trong mô hình quan hệ. Là một minh họa các tính năng, hình 25,1 hiện nay SQL: 1999 DDL báo cáo cho phần của giản đồ ORDBMS Dinky của mà sẽ được sử dụng trong ví dụ tiếp theo Mặc dù DDL là một hệ thống quan hệ truyền thống, nó có một số khác biệt quan trọng mà làm nổi bật việc dữ liệu mới mô hình khả năng của một ORDBMS.Nhìn lướt qua các câu lệnh DDL là đủ cho bây giờ, chúng tôi sẽ nghiên cứu chi tiết trong phần tiếp theo, sau khi trình bày một số các khái niệm cơ bản mà ứng dụng mẫu của chúng tôi cho thấy là cần thiết trong một DBMS thế hệ tiếp theo
  1. **Thao tác với các loại dữ liệu mới**
* Như vậy đến thời điểm này, chúng ta đã mô tả các loại dữ liệu phải được lưu trữ trong cơ sở dữ liệu dinky. Ở đây ta không nói bất cứ điều gì về cách sử dụng các loại mới trong các truy vấn, vì vậy chúng ta hãy nghiên cứu hai truy vấn dữ liệu của dinky cần phải hỗ trợ. Cú pháp của các truy vấn không quan trọng, nó là đủ để hiểu những gì họ thể hiện.Chúng tôi sẽ trở về đặc biệt của cú pháp truy vấn như chúng ta tiến hành



* Thách thức đầu tiên  đến từ công ty Clog cung cấp ngũ cốc ăn sáng. Clog sản xuất ngũ cốc được gọi là Delirios, và muốn thuê một hình ảnh của Herbert Worm ở phía trước của mặt trời mọc, để kết hợp trong thiết kế hộp Delirios. Một truy vấn để trình bày một bộ sưu tập các hình ảnh có thời điểm này có và giá cho thuê có thể được thể hiện trong SQL-giống như cú pháp như trong hình25,2. Dinky có một phương pháp số được viết bằng một ngôn ngữ bắt buộc như Java và đăng ký với hệ thống cơ sở dữ liệu. Những phương pháp này có thể được sử dụng trong các truy vấn trong cùng xây dựng trong các phương pháp, chẳng hạn như =, +, -, được sử dụng trong một ngôn ngữ quan hệ như SQL. Các phương pháp hình ảnh thu nhỏ trong mệnh đề Select sản xuất một phiên bản nhỏ của hình ảnh đầu vào đầy đủ kích thước của nó.Các phương pháp is\_sunrise là một chức năng boolean phân tích một hình ảnh và trả về true nếu ảnh có chứa một mặt trời mọc;phương pháp is\_herbert trả về true nếu ảnh có chứa một hình ảnh của Herbert.Truy vấn sản xuất mã số khung, thumhnail hình ảnh, và giá cho tất cả các khung có chứa Herbert và mặt trời mọc.



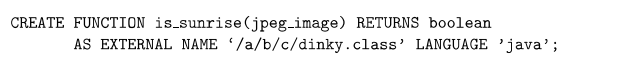
* Thách thức thứ hai đến từ giám đốc điều hành dinky.Họ biết Delirios là cực kỳ phổ biến ở đất nước nhỏ bé của Andorra, vì vậy họ muốn đảm bảo rằng một số bộ phim Herbert đang chơi tại nhà hát gần Andorra khi cung cấp ngũ cốc việc kệ hàng. Để kiểm tra về hiện trạng của vấn đề, việc giám đốc điều hành muốn tìm tên của tất cả các rạp chiếu phim cho thấy có bộ phim Herbert trong vòng 100km Andorra. Hình 25,3 cho có truy vấn này trong một cú pháp SQL:



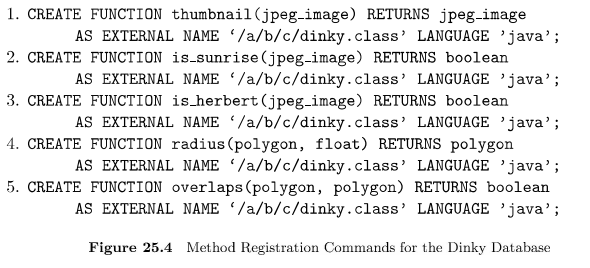
* Các thuộc tính rạp chiếu phim của bảng Nowshowing là một tham chiếu đến một đối tượng trong bảng khác, trong đó có các thuộc tính name, address, và location. Tham chiếu đối tượng này cho phép việc ký hiệu N.theater-> name và N.theater-> address, trong đó đề cập các thuộc tính của đối tượng theater\_t tham chiếu có trong hàng Nowshowing N. Thuộc tính có ngôi sao của bảng bộ phim là một tập hợp các tên mỗi ngôi sao của bộ phim. Phương pháp radius trả về một vòng có trung tâm ở số đầu tiên của nó với radius tương đương với số thứ hai của có. Phương pháp overlaps kiểm tra chồng chéo lên nhau không gian. Vì vậy, Nowshowing và Films là sự tham gia của các điều khoản kết bằng, trong khi Nowshowing và các nước tham gia củaviệc khoản chồng chéo không có. Các lựa chọn "Andorra và phim ảnh có chứa ‘Hertbert Worm’ hoàn chỉnh với câu truy vấn.
* Hai câu truy vấn trên tương tự như truy vấn SQL-92 nhưng có một số tính năng không bình thường:
  + Phương thức do người dùng định nghĩa: Người dùng tự định nghĩa các kiểu dữ liệu trừu tượng để thực thi cho các phương thức của họ ví dụ is\_heabert.
  + Các toán tử có cầu trúc: Cùng với các loại cấu trúc có sẵn trong mô hình dữ liệu, ORDBMSs cung cấp việc phương pháp tự nhiên cho các loại dữ liệu đó. Ví dụ, các thiết lập của chuẩn của các loại…
  + Toán tử liên kết: tính tham chiếu trong loại được thế hiện bằng toán tử -> (arrow).

1. **Thảo luận về kiểu dữ liệu trừu tượng, định nghĩa và thao tác**

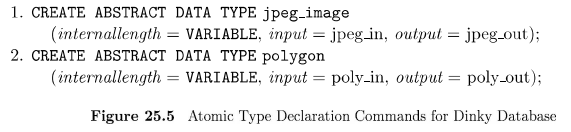
* Xem xét bảng Frames - 25,1. Nó có một cột hình ảnh của loại jpeg\_image, dùng để lưu trữ một hình ảnh nén đại diện cho một Frame duy nhất của một bộ phim. Các loại jpeg\_image không nằm trong nhân DBMS được xây dựng trong các loại và được xác định bởi một người dùng cho việc ứng dụng dinky để lưu trữ dữ liệu hình ảnh nhất nén bằng cách sử dụng tiêu chuẩn JPEG. Một ví dụ khác, bảng Countries xác định trong dòng 6 của hình 25,1 có một ranh giới cột kiểu của đa giác, trong đó có đại diện của các hình dạng của các Countries vạch ra trên bản đồ thế giới.
* Cho phép người dùng định nghĩa kiểu dữ liệu tùy ý mới là một tính năng quan trọng của ORDBMSs. DBMS cho phép người dùng lưu trữ và lấy đối tượng jpeg\_image loại, giống như một đối tượng của bất kỳ loại khác, chẳng hạn như số nguyên. Các kiểu dữ liệu nguyên tử thường cần phải có các loại hoạt động cụ thể quy định bởi người sử dụng tạo ra chúng. Ví dụ, người ta có thể xác định các hoạt động trên một kiểu dữ liệu hình ảnh và các cộng sự của nó phương pháp được gọi là một kiểu dữ liệu trừu tượng, hoặc ADT. Truyền thống SQL đi các với xây dựng trong ADTs, chẳng hạn như số nguyên (với các phương pháp số học có liên quan), hoặc các chuỗi (với sự bình đẳng, so sánh, và phương thức tương tự). Hệ thống quan hệ đối tượng bao gồm các ADTs và cũng cho phép người dùng xác định ADTs của riêng mình.
* Nhãn "abstract" được áp dụng cho các loại dữ liệu vì hệ thống cơ sở dữ liệu không cần phải biết làm thế nào dữ liệu của ADT được lưu trữ cũng như phương thức làm việc của ADT. Nó chỉ đơn thuần là cần phải biết những phương thức có sẵn và các loại đầu vào và ouput cho các phương thức. Cất giấu bên trong ADT được gọi là đóng gói. Lưu ý rằng ngay cả trong một hệ thống quan hệ, các loại nguyên tử chẳng hạn như số nguyên có liên quan đến phương pháp được đóng gói vào ADTs. Trong trường hợp các số nguyên, phương pháp tiêu chuẩn cho ADT là các nhà khai thác số liệu thông thường và so sánh. Để đánh giá các nhà điều hành bổ liệu vào số nguyên, hệ thống cơ sở dữ liệu không cần phải hiểu những quy định của - luật bổ sung - nó chỉ cần biết làm thế nào để gọi phương thức đoạn mã khai thác và loại dữ liệu để mong đợi trong trả về.
* Trong một hệ thống quan hệ đối tượng, đơn giản hóa do sự đóng gói là rất quan trọng bởi vì nó che giấu bất kỳ sự phân biệt nội dung giữa các loại dữ liệu và cho phép một ORDBMS được thực hiện mà không cần dự đoán các loại và phương thức mà người dùng có thể muốn thêm. Ví dụ, hình ảnh bổ sung thêm các số nguyên và phủ có nhất được xử lý đồng nhất của hệ thống, với sự phân biệt đáng kể duy nhất được rằng mã khác nhau trong gọi cho hai hoạt động và đối tượng khác nhau đã nhập được dự kiến sẽ được trả lại từ mã đó.
  1. **Định nghĩa phướng thức của kiểu dữ liệu trừu tượng**
* Ở mức tối thiểu, đối với từng loại nguyên tử mới, một người sử dụng phải xác định các phương pháp cho phép các DBMS để đọc và để tạo ra các đối tượng của loại hình này và để tính toán lượng lưu trữ cần thiết để giữ đối tượng.Người dùng tạo ra một loại mới nguyên tử phải đăng ký các phương thức sau đây với các DBMS:
  + **Size:** Trả về số byte cần thiết dùng để lưu trữ đối tượng này hoặc một giá trị động đặc biệt, nếu như nó có kích thước động.
  + **Import:** Là phương thức cho phép tạo ra một đối tượng mới dữ liệu đầu vào.
  + **Export:** Ánh xạ các đối tượng để thích hợp cho việc in ấn, hoặc sử dụng trong các chương trình ứng dụng.
* Để đăng ký một phương pháp mới cho một loại nguyên tử, người sử dụng phải viết mã cho phương pháp này và sau đó thông báo cho hệ thống cơ sở dữ liệu về phương pháp.Các mã được viết phụ thuộc vào các ngôn ngữ được hỗ trợ bởi các DBMS, và có thể là hệ điều hành trong câu hỏi.Ví dụ, ORDBMSS có thể xử lý mã Java trong hệ điều hành Linux.Trong trường hợp này, các phương pháp mã phải được viết bằng Java và biên dịch vào một tập tin bytecode Java được lưu trữ trong một hệ thống tập tin Linux.Sau đó, một phong cách phương pháp lệnh SQL đăng ký được trao cho ORDBMS để nó nhận ra phương pháp mới:



* Các khai báo này là những điểm đặc trưng của phương thức: các loại ADT liên quan, kiểu trả về, vị trị của đoạn mã này. Khi phương thức này được đăng ký, DBMS sử dụng một máy ảo Java để thực thi mã:



* Những khai báo định nghĩa kiểu dữ liệu nguyên tử do người dùng định nghĩa để sử dụng trong lược đồ của Dinky



1. **Cách thức và cấu trúc của kiểu dữ liệu trừu tượng**

* Các loại nguyên tử và các loại do người dùng định nghĩa có thể kết hợp để mô tả cấu trúc phức tạp hơn bằng cách sử dụng cac loại hàm dựng. Ví dụ dòng thứ 6 của hình 25.1 xác định cột ngôi sao của loại setof(text), mối mục trong cột đó là một tập cac chuỗi văn bản, đại diện cho các ngôi sao trong bộ phim. Các cú pháp setof là một ví dụ của một hàm dựng. Các hàm dựng phổ biến khác bao gồm:
  + **ROW**(n­1 t1, …, nn tn): Một loại trình bày cho một dòng, hoặc một bộ của n cột dữ liệu với n1,…,nn thuộc loại t1,…,tn tương ứng.
  + **Listof(base):** Một kiểu đại diện cho một list các phần tử cơ sở.
  + **ARRAY(basse):** Một kiểu đại diện cho một mảng các phần tử cơ sở.
  + **setof(base):** Một kiểu đại diện cho một tập (set) các phần tử cơ sở **.** Một set thì không có 2 phần tử trùng nhau.
  + **bagof(base):** Một kiểu đại diện cho một bag bao gồm các phần tử cơ sở hoặc các set.
* Đánh giá đầy đủ khả năng của các hàm dựng, ví dụ ARRAY (ROW (age: integer, sal: integer)). Các loại được xác định bằng cách sử dụng các hàm dựng của các kiểu được gọi là kiểu có cấu trúc. Ngoài ra có thể sử dụng listof, ARRAY, bagof, hoặc setof như là các hàm dựng sau cùng đôi khí được gọi là tập hợp các loại, hoặc dữ liệu với số lượng lớn.
  1. **Thao tác ở kiểu dữ liệu có cấu trúc**
* DBMS cung cấp phương thức được xây dựng cho các loại hỗ trợ thông qua hàm dựng loại. Những phương pháp này tương tự như trong hoạt động xây dựng như phép cộng và phép nhân đối với các loại nguyên tử chẳng hạn như số nguyên.Trong phần này chúng tôi trình bày các phương pháp để xây dựng loại hình khác nhau và minh họa các truy vấn SQL có thể tạo ra và thao tác giá trị với các loại cấu trúc.
* **ROWS:** Với mỗi item có loại là ROW(n1 t1, …, nn tn) phương pháp lấy các field cho phép chúng ta truy cập đến một nk sử dụng ký hiệu i.nk truyền thống các hàm dựng được lồng trong một kiểu định nghĩa, dấu (.) có thể được lồng vào nhau để truy cập vào các field của hàng lồng nhau.
* Ký hiệu các chấm lồng nhau này thường được gọi là một biểu thức đường dẫn bởi vì nó mô tả một đường thông qua các cấu trúc lông nhau.
* **Sets và multisets:** Một tập đối tượng có thể so sánh được với nhau sử dụng các toán tử truyền thống của tập (các toán tử tập hợp). Một phần tử trong tập setof(foo) có thể so sánh với các tập khác sử dụng phương thức thuộc tập phương thức. Hai tập đốio tượng (các phần tử có cùng kiểu) có thể tổ hợp để tạo nên đối tượng mới sử dựng (giao, hợp, hiệu).
* **Lists:** Danh sách bình thường sẽ bao gồm head, trả về yếu tố đầu tiên là tail, trả về danh sách thu được bằng cách bỏ các yếu tố đầu tiên, thêm vào trước, trong đó có một phần tử và chèn nó vào như là các yếu tố đầu tiên trong một list và thêm vào một danh sách khác.
* **Arrays:** Mảng loại hỗ trợ một phương thức ‘mảng chỉ số’ cho phép truy cập vào các mảng tại một thiết lập cụ thể. Có có dấu ngoặc vuông phía sau dùng để truy cập đến phần tử nào.

1. **Xem xét và nhận dạng đối tượng.**

* Trong HQT CSDL hướng đối tượng, đối tượng dữ liệu có thể dc gán 1 giá trị object identifier (oid), xác định duy nhất trong dữ liệu trong suốt thời gian sống của nó. DBMS chịu trách nhiệm tạo ra giá trị này và phải đảm bảo nó luôn xd dc 1 đối tượng duy nhất trong suốt thời gian sống. ở 1 số hqt tất cả các bộ dc lưu trong bất cứ bảng nào thì cũng đều là đối tượng và he qt sẽ tự đống đánh cho chúng 1 giá trị oid. ; 1 số hqt khách, ng dùng có thể chỉ ra các bảng nào có oid.
* 1 oid của đối tượng dc dùng để chỉ tới đối tượng đó mà có dữ liệu dc lưu ở nơi khác. Nó như 1 giá con trỏ tham chiếu có kiểu là loại mà nó tham chiếu đến. ref(base) là 1 hàm dựng kiểu là con trỏ trỏ đến object kiểu base. Ref có thể dc chèn vào trong hàm dựng khác.
* So sánh url với oid:

|  |  |
| --- | --- |
| **Oid** | **URL** |
| Xác định duy nhất 1 đối tượng trong suốt thời gian | Tài nguyên web dc trỏ bởi 1 URL có thể thay đổi theo thời gian |
| Đơn giản, nó không mang ý nghĩa vật lý (không chứa thông tin vật lý của đối tượng mà nó trỏ đến) - dẫn đến nếu thay đổi địa chỉ lưu trữ của đối tượng thì ta khộng phải sửa oid này. | Chứa cả địa chỉ network và tên file system, khi tài nguyên mà url đó trỏ đến thay đổi địa chỉ lưu hoặc địa chỉ network, đường dẫn url đến tài nguyên đó bị sai hoặc phải yêu cầu cơ chế forwarding. |
| Oid được phát sinh tự động bởi DBMS | Người dùng tạo ra |

* 1. **Khái niệm về Equality**
* Sự khác biệt giửa các loại tham chiếu với các loại cấu trúc tham chiếu tự do (reference-free structured types) làm nảy sinh 1 vấn đề khác: định nghĩa đẳng thức. 2 object có cùng loại được xem là deep equal khi và chỉ khi:
* Các Đối tượng là kiểu nguyên tử và có cùng giá trị.
* Các đối tượng là kiểu tham chiếu và
* Đối tượng là loại cấu trúc, toán tử deep equal đúng với tất cả thành phần tương ứng của 2 đối tượng.
* 2 đối tượng có cùng loại tham chiếu được định nghĩa là shallow equal nếu cả 2 chỉ đến cùng đối tượng. định nghĩa shallow equality có thể dc mở rộng cho đối tượng có loại tuỳ ý bằng cách lấy định nghĩa của deep equality và thay deep equals thành shallow equals.
* 2 đối tượng thoả shallow equal thì thoả deep equal, ngược lại không đúng
  1. **Bất tham chiếu đến các loại tham chiếu**
* Loại tham chiếu ref(foo) không giống như loại foo nó trỏ đến. để truy cập foo dc tham chiếu, có phương thức deref() cung cấp song song với hàm dựng loại ref. giống như trong ng lập trình def() = \*a. ví dụ cần lấy 1 bộ trong bảng Nowhowing, lấy giá trị name của theater: Noshowing.deref(theater).name. vì tham chiếu đến các loại bộ thì giống nhau, nên 1 số hqt cung cập 1 toán tử mũi tên kết hợp với hậu tố của toán tử dereference operator . đó là: Nowshowing.theater🡪name.

1. **Sự kế thừa và phân cấp các loại.**
   1. **Định nghĩa các loại với sự kế thừa**

* Trong csdl Dinky, ta mô hình hoá rạp chiếu phim thành kiểu theater\_t. nhưng Dinky muốn kinh doanh thêm loại rạp chiếu-café, rapchiếu-café sẽ cung cấp pizza và các laoị thức ăn uống khi xem phim. Vì vậy trong db cần hỗ trợ lưu trữ các thông tin này, theater-café giống như theater nhưng có thêm các loại thuộc tính thực đơn. T dùng kế thừa, đoạn lệnh DDL sau diễn tả điều đó:
* Create type theatercafe\_t under theater\_t (menu text):
* Trong lệnh này tạo ra theatcafe và tất cả các thuộc tính và phương thức của theater\_t đều dc theatercafe dùng lại nhưng điểu ngược lại thì k. ta nói theatercafe\_t kế thừa các thuộc tính của theater\_t.
* Lưu ý cơ chế kế thừa k phải là macro làm ngắn gọn lệnh create. Nó tạo ra quan hệ rõ ràng trong dữ liệu giữa subtype (theatercafe\_t) và supertype (theater\_t): 1 object của subtype ccũng dc xét là 1 object của supertype.
* The substitution Principle: cho 1 supertype A và 1 subtype B, luôn luôn có thể thay thế 1 đối tượng loại B thành 1 biểu diễn hợp lý được viết bởi object laoị A, ngoại trừ việc tạo ra type errors.
* Note: kế thừa cũng có thể dc dùng cho kiểu nguyên tử, cho cả kiểu row.
  1. **Tính ràng buộc của những phương thức**
* Ta có thể viết lại phương thức của supertype trong subtype. Viết 1 phương thức mới với phương thức cũ gọi là overloading tên phương thức.
* Do việc overloading này, hệ thống phải hiểu phương thức nào dc dành cho 1 biểu diễn cụ thể. Vd, khi sys cần gọi pthức display() trong object kiểu jpeg\_image\_t, nó dùng phương thức display riêng biệt. khi cần vọi display cảu image\_t (k phải loại subtyped) thì n1o gọi pthuc display chuẩn. quá trình (process) quyết định phương thức nào dc gọi lên dc gọi là liên kết phương thức với đối tượng (binding to the object). Binding dc thực hiện khi 1 biểu thức dc phân tách (early binding), 1 số trường hợp
* Early binding : Khi quá trình binding được thực thi trước khi chương trình chạy(by compiler & linker). Vấn đề chính của early binding là compiler không biết chính xác method nào để gọi khi chỉ có một tham khảo tới đối tượng Instrument. C compilers chỉ có một lời gọi method là early binding.
* Late binding : ra đời để giải quyết vấn đề của early binding, nó sẽ binding ở thời điểm chạy, nó có cơ chế xác định đối tượng nào ngay tại thời điểm gọi và gọi method của đối tượng đó.
* Late binding là cái core của Tính đa hình (Polymorphism). Nhờ late binding mà tính đa hình trở thành công cụ mạnh mẽ trong Lập trình hướng đối tượng.
  1. **Tập các hệ thống phân cấp, loại mở rộng, và truy vấn**
* Có 1 chút khác biệt giữa kế thừa trong topic này với ngôn ngữ lập trình hướng đối tượng
* Tuy nhiên vì ORDBMS cung ấp ngôn gnữ truy vấn trên tập dữ liệu dạng bảng, nên các cơ chế từ các ngôn gnữ lập trình dc tăng cường trong CSDL đối tượng để xử lý với bảng và truy vấn tốt. một cách riêng, trong ob-relational sys ta ca thể định nghĩa 1 bảng chứa cá các đối tượng có kiểu cụ thể, như bảng theater trong lược đồ Dinky. Cho 1 subtype như theater-café, nếu mốn tạo bảng theater-café khác để lưu thong tin về theater cafes. Nhưng khi viết 1 truy vấn trên bảng theaters, đôi khi muốn 1 câu truy vấn tượng tự trong bảng theate-cafes, sau hết, nếu ta chiếu các cột, 1 ví dụ của bảng theater-café có thể dc quan tam như 1 ví dụ của bảng.
* Không phải yêu cầu ng dùng chỉ rõ 1 câu query riêng lẽ cho mỗi bảng, ta có thể thông báo cho hệ thống là 1 bảng mới thuộc subtype dc xem như 1 phần của bảng supertype:
* Create table theater\_cafes of type theater\_cafe\_t under theaters;
* Truy vấn này nói cho hệ thống biết các lệnh query trên bảng theaters phải dc chạy trong các bộ trên cả bảng theaters và theater\_cafes. Nếu subtype definition gọi phương thức overlaoding, thì late-binding dc dùng để chắc ràng phước thức đó dc gọi cho các bộ hợp lý.
* Nhìn chung, mệnh để clause dc dùng để tạo 1 cây tuỳ ý của bảng gọi 1 tập phân cấp. lệnh query trên 1 bảng riêng biệt T trong phân cấp dc chạy trên tất cả bộ trong T và các node con của nó. Đôi khi, 1 ng dùng có thể muốn truy vấn chạy chỉ trên T, và k trên các con, thì thêm cú pháp only.
* Một số hệ thống tự động tạo các bảng đạc biệt cho mỗi kiểu, mà chứa tham chiếu đến các instance của type mà tồn tại trong db. Các bảng này dc gọi là type extents và cho phép truy vấn trên tất cả đối tượng của type cho, k kể đến nơi các đối tượng cư trú trong db. Type extents hình thành 1 collection hierarchy mà song song với kiểu phân cấp.

1. **Xem xét làm thế nào để tận dụng lợi thế của khái niệm hướng đối tượng mới để thiết kế cơ sở dữ liệu ORDBMS**

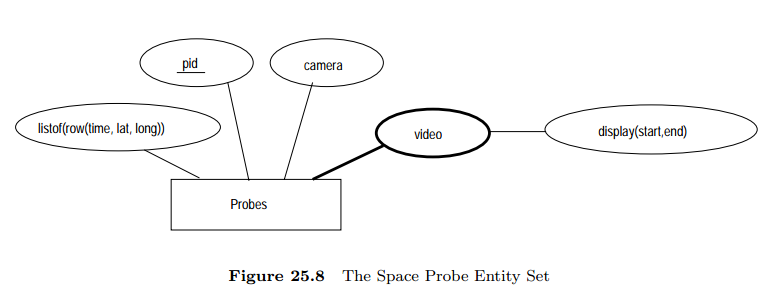
* Sự đa dạng về trạng thái của loại dữ liệu trong ORDBMS đã mang lại cho các nhà thiết kế cơ sở dữ liệu nhiều cơ hội cho việc thiết kế hiệu quả.
  1. **Loại có cấu trúc và loại trừu tượng**
* Ví dụ đầu tiên của chúng ta liên quan đến tàu thăm dò vũ trụ (probe), với mỗi probe sẽ có một loại camera xác định, sẽ ghi lại một đoạn video.
* Mỗi video sẽ được thu lại trong một khoảng thời gian cụ thể và ở vị trí với vĩ độ và kinh độ xác định.
* Do đó thông tin liên quan đến một tàu thăm dò sẽ có 3 phần chủ yếu: probe\_id, video stream, <time, location>
* **Thiết kế CSDL với RDBMS:**
  + Với RDBMS, chúng ta phải lưu video stream với kiểu dữ liệu BLOB và mỗi vị trí tuần tự chúng ta lưu thành một bộ trong bảng.
    - Probe (**pid: integer, time: timestamp, latitude: real, longitude: real**, camera: string, video: BLOB)
    - Vơi một bảng Probes, và có một dòng cho một probe.
    - Và chúng ta có 2 phụ thuộc hàm
      * PTLN🡪 CV
      * P🡪CV
    - Và với 2 phụ thuộc hàm này cấu trúc của chúng ta không đạt dạng chuẩn BCNF, vì vậy tách ra làm 2 bảng như sau:
    - Probe\_Loc (**pid: integer, time: timestamp, latitude: real, longitude: real**)
    - Probe (**pid: integer,** camera: string, video: BLOB)
  + Thiết kế này thì tốt khi được thực hiện trên RDBMS. Tuy nhiên, có nhiều hạn chế:
    - Đối với câu truy vấn “Với probe 10, hiển thị đoạn video được thu từ 1:10pm đến 1:15pm vào ngày 10, 1996” 🡪 Chúng ta phải truy vấn toàn bộ các đối tượng video của probe 10 được thu trong hàng giờ để ra được kết quả trên.
    - Sự thật mà mỗi probe có một dòng các location không rõ nghĩa, và thông tin nối tiếp liên quan tới probe bị phân tán qua các bộ.
    - Chúng ta bắt buộc phải chia thông tin liên tiếp của một probe ra. Điều này không hiệu quả đối với các truy vấn có liên quan đến tất cả thông tin của probe. Ví dụ “Với mỗi probe, hãy in thời gian sớm nhất được thu lại và loại camera”.
* **Thiết kế CSDL với ORDBMS:**
  + Một ORDBMS hỗ trợ nhiều giải pháp tốt hơn.
    - Thứ nhất, chúng ta có thể lưu trữ video dưới dạng một đối tượng ADT và viết các phương thức để chụp hay xử lý đặc biệt như chúng ta mong muốn.
    - Thứ hai, bởi vì chúng ta có thể lưu trữ với các kiểu dữ liệu có cấu trúc như là list, chúng ta sẽ dùng list để lưu trữ vị trí của probe trên một bộ đơn cùng với đoạn video stream.
    - Với ORDBMS ta có một quan hệ duy nhất như sau:
      * Probes\_AllInfo (pid: integer, locseq: location\_seq, camera: string, video: mpeg\_stream)
      * Và khi viết theo cách này chúng ta có thể có các câu truy vấn sau:

**SELECT** display (P.video, 1:10pm, May 10 1996, 1:15pm, May 10 1996)

**FROM** Probes\_Allinfo P

**WHERE** P.id = 1

* 1. **Định danh đối tượng**
* Chúng ta sẽ thảo luận hệ quả của việc sử dụng reference hoặc oid. Việc sử dụng oid đặc biệt có ý nghĩa khi các đối tượng có kích thước lơn, bởi vì các đối tượng đó là một cấu trúc dữ liệu hoặc một đối tượng lớn như một hình ảnh.
* Giải thích các thao tác thường sử dụng:
  + Xóa: các đối tượng tham chiếu bị ảnh hưởng, thường nó sẽ chuyển thành giá trị null
  + Cập nhật: Thay đổi trên tất cả các đối tượng tham chiếu -> không cần phải cập nhật thủ công cho từng đối tượng như đối tượng không tham chiếu
  + Chia sẻ với sao chép: một đối tượng có thể bị tham chiếu bởi nhiều đối tượng khác, do đó một sự cập nhật sẽ được phản chiếu ở nhiều nơi. Trong khi đó, đối tượng cơ sở dữ liệu quan hệ đòi hỏi phải cập nhật ở từng bản copy
  + Chi phí lưu trữ: lưu trữ các bản copy của đối tượng sẽ tốn nhiều không gian hơn lưu một giá trị và tham chiếu đến nó. Điều này có thể ảnh hưởng đến việc sử dụng đĩa và quản lí bộ đệm (đặc biệt khi nhiều bản copy được truy vấn cùng lúc)
  + Phân nhóm: Những phần nhỏ trong đối tượng có cấu trúc thường lưu trữ gần nhau trên đĩa. Do đó các đối tượng nếu có tham chiếu quá xa sẽ khiến đĩa truy xuất lâu hơn, vì vậy đối tượng có cấu trúc sẽ hiệu quả hơn các loại tham chiếu nếu chúng thường được truy vấn theo tổng số
* Nhưng cần lưu ý, trong thiết kế cơ sở dữ liệu, sự lựa chọn giữa kiểu cấu trúc và kiểu tham chiếu cần phải xem xét dựa trên chi phí lưu trữ, vấn đề phân nhóm, và ảnh hưởng của việc cập nhật
* **Dùng định danh đối tượng hay Foreign Keys:**
  + Một vấn đề nữa đặt ra là nên sử dụng định danh đối tượng (Oid) hay khóa ngoại.
  + Oid: một oid có thể trỏ đến một đối tượng được lưu ở bất cứ đâu trong cơ sở dữ liệu.
  + Khóa ngoại: là một ràng buộc trỏ đến đối tượng trong một quan hệ tham chiếu cụ thể.
  + Do đặc điểm trên mà DBMS dễ dàng cung cấp nhiều hỗ trợ cho ràng buộc tham chiếu hơn là con trỏ Oid. Chú ý khi xóa.
  + Chúng ta nên cẩn trọng khi sử dụng oid, và nên sử dụng khóa ngoại thay thế bất cứ khi nào có thể.
  1. **Mở rộng mô hình thực thể kết hợp (ER)**
* Với mô hình ER mô tả ở chương 2 không đầy đủ cho thiết kế ORDBMS. Chúng ta sẽ sử dụng mô hình ER mở rộng và hỗ trợ các giá trị có cấu trúc. Phân biệt giữa một thực thể và oid của tối tượng , và mô hình cho phép mô tả các phương thức của đối tượng:

****

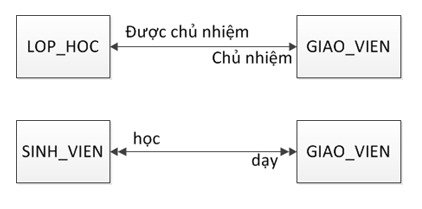
* Cuối cùng, chúng ta cần phải ghi chú các mở rộng có ý nghĩa của mô hình ER là yêu cầu hỗ trợ từ nhà thiết kế nằm từ những tập lồng nhau. Ví dụ như về cách lưu trữ vị trí của một phi thuyền.
  1. **Sử dụng tập Nested(Lồng nhau)**
* **Tập hợp lồng nhau cung cấp sức mạnh mô hình hóa lớn, tuy nhien cũng tạo ra nhiều trở ngại trong quyết định thiết kế:**
  + Probes1 (pid: integer, locseq: location\_seq)
  + Câu truy vấn: Với mỗi probe, hãy xuất ra thời gian sớm nhất được ghi lại và loại camera
  + Probes2 (pid: integer, time:timestamp, latitude: real, longitude: real)
  + Câu truy vấn: Tìm thời điểm sớm nhất được ghi ở latitude =5, và longitude =10
  + Sự chọn lựa trong thiết kế lược đồ phải phụ thuộc vào lượng workload mong muốn
  + Tập hợp lồng nhau thích hợp ở trường hợp cụ thể do đó khi sử dụng cần cẩn trọng, tránh lạm dụng

1. **Thảo luận một số thách thức mới đặt ra bởi hệ thống quan hệ đối tượng.**

* Cơ sở dữ liệu hướng đối tượng ra đời là đã giải quyết được một số khó khăn nhưng đồng thời cũng mang theo các thách thức:
  1. **Lưu trữ và truy cập phương thức**
* Dữ liệu người dùng tự định nghĩa ADT khá lớn. Trong một số trường hợp, nó sẽ lớn hơn page, ví dụ như BLOB. Khi ADT lớn hơn page memory, sư dụng các disk-base pointer được duy trì từ các bộ đến đối tượng mà nó chứa.
* Solution: các con trỏ dựa trên đĩa (disk-based pointer) được duy trì từ các bộ đến đối tượng mà nó chứa
* Đối tượng cấu trúc không giống ADT, có kích thước biến thiên trong vòng đời của cơ sở dữ liệu. Ví dụ: đối với thuộc tính star, qua nhiều năm khi các diễn viên nhỏ trở nên nổi tiếng, Dinky muốn quảng bá họ đã từng tham gia bộ film đó thì sẽ đòi hỏi thểm họ vào thuộc tính star. Bởi vì các thuộc tính này có thể tăng tùy ý, do đó đòi hỏi cơ chế sắp xếp đĩa linh hoạt.
* Đối với loại dữ liệu mảng, thông thường sẽ được lưu trữ liên tiếp trên đĩa theo kiểu từng dòng. Tuy nhiên đối với các truy vấn chỉ truy xuất một phần của mảng, điều này sẽ đòi hỏi chi phí I/O cao. Vì vậy để giảm chi phí mảng thường được chia nhỏ ra thành những khối nhỏ (chunk) và các chunk không cần lưu theo từng dòng hay từng cột
  1. **Chỉ mục trên loại dữ liệu mới**
* Cấu trúc chỉ mục RDBMS chuẩn chỉ hỗ trợ điều kiện bằng (cây B+ và hash index) và điều kiện phạm vi (cây B+).
* Nhiều cấu trúc chỉ mục đã được đưa ra cho từng loại ứng dụng như: vẽ bản đồ, nghiên cứu di truyền, lưu trữ đa phương tiên, web search … nhưng một ORDBMS không thể cài đặt tất cả các chỉ mục được sáng tạo ra. Do đó tập cấu trúc chỉ mục ORDBMS phải linh hoạt đối với người dùng
* Giải pháp thứ nhất: xây dựng một interface phương thức truy vấn cho phép người dùng cài đặt các cấu trúc chỉ mục bên ngoài DBMS. Trong đó chỉ mục và dữ liệu có thể được lưu trong file system và DBMS chỉ đưa ra các yêu cầu lặp đi lặp lại là open, next, close đến đoạn mã chỉ mục bên ngoài của người dùng. Tuy nhiên phần dữ liệu ở chỉ mục bên ngoài không được bảo vệ bởi DBMS cho tính đồng thời và phục hồi.
* Giải pháp thứ hai: xây dựng một template cấu trúc chỉ mục chung có thể đáp ứng tổng quát hầu hết các cấu trúc chỉ mục mà người dùng tạo ra. Bởi vì cấu trúc này sẽ được cài đặt trong DBMS nên nó có thể khắc phục vấn đề đồng thời và phục hồi. => Generalised Search Tree (GiST) là một cấu trúc như vậy. Nó là một template dựa trên cây B+, cho phép hầu hết các cấu trúc chỉ mục có thể cài đặt với vài dòng ADT code do người dùng định nghĩa
  1. **Xử lý truy vấn**
* Hầu hết các ORDBMS cho phép người dùng định nghĩa hàm kết hợp mới. Để định nghĩa một hàm kết hợp, người dùng phải cài đặt 3 phương thức sau:
  + Initialize: khởi tạo trạng thái bên trong cho sự kết hợp
  + Iterate: cập nhật trạng thái cho mỗi bộ nhìn thấy
  + Terminate: tính toán kết quả kết hợp dựa trên trạng thái cuối cùng và sau đó dọn dẹp
* Ví dụ: tìm giá trị cao thứ hai của một thuộc tính:
  + Initialize: xác định lưu trữ cho hai giá trị cao nhất
  + Iterate: so sánh giá trị của bộ hiện tại với hai giá trị cao nhất và cập nhật hai giá trị cao nhất lại nếu cần
  + Terminate: xóa lưu trữ của hai giá trị và trả về bản copy của giá trị cao thứ 2
  1. **Tối ưu truy vấn**
* **Method caching**
  + Đối với phương thức được sử dụng nhiều lần, chúng ta có thể duy trì cache của phương thức input và những output trùng khớp như một bảng trong cơ sở dữ liệu. Sau đó để tìm kiếm giá trị của phương thức với một input cụ thể, chúng ta có thể kết bộ input với cache table. Đối với một truy vấn đơn, có thể tránh gọi phương thức hai lần trên cùng giá trị của một cột bằng cách sắp xếp bảng trên cột đó hoặc sử dụng lược đồ dựa trên hash giống như đã dùng trong chương 12.7. Chúng ta có thể kết hợp cả hai phương pháp trên với nhau
* **Pointer Swizzling**
  + Trong một vài ứng dụng, đối tượng đưa vào bộ nhớ và truy cập thường xuyên thông qua oid. Một số hệ thống duy trì một bảng oid trong bộ nhớ. Khi đối tượng O được đưa vào bộ nhớ, chúng kiểm tra oid trong đối tượng O và thay thế oid của đối tượng trong bộ nhớ bằng con trỏ đến các đối tượng đó. Kỹ thuật này được gọi là pointer swizzling. Tuy nhiên khi một đối tượng page out, thì tham chiếu trong bộ nhớ đến nó cũng không còn tác dụng nữa và sẽ bị thay thế với oid của nó
  1. **Tinh chỉnh truy vấn**
* **Khi chỉ mục mới được thêm vào hệ thống, bộ tinh chỉnh phải được thông báo về sự có mặt của nó và chi phí truy vấn.:**
  + Bộ tinh chỉnh phải biết được
  + Điều kiện gì trong mệnh đề WHERE trùng khớp với chỉ mục
  + Chi phí nạp một bộ đối với chỉ mục là bao nhiêu
  + Dựa vào các thông tin đó mà bộ tinh chỉnh có thể sử dụng bất kì cấu trúc chỉ mục nào trong việc xây dựng kế hoạch thực thi.
* **Hệ số giảm và tính toán chi phí cho phương thức ADT**
  + Việc tính toán hệ số giảm đối với điều kiện do người dùng định nghĩa là một vấn đề khó và vẫn còn đang được nghiên cứu.
  + Cách tiếp cận thông thường hiện nay là để người dùng tự tính toán, người dùng một khi có thể tạo ra một phương thức thì cũng có tạo một hàm bổ trọ việc tính toán hệ số giảm của phương thức đó. Nếu như hàm tính toán đó không được đăng kí thì bộ tinh chỉnh sử dụng một số tùy ý như 1/10.
  + Người dùng có thể tính toán bằng chi phí I/O trong hệ thống. Ngoài ra ORDBMS có thể thực thi phương thức trên nhiều đối tượng có kích thước khác nhau để cố gắng tính toán chi phí một cách tự động, nhưng cách này vẫn chưa được xem xét chi tiết.
* **Tinh chỉnh các phép chọn chi phí cao**
  + Trong cơ sở dữ liệu quan hệ, phép chọn được xem như một thao tác không tốn thời gian. Ví dụ emp.salary không cần I/O và chỉ tốn một vài chu kỳ CPU, nhưng đối với is\_herbert (Frames.image) thì khá đắt đỏ vì nó phải nạp một lượng lớn đối tượng từ đĩa vào và xử lí chúng trong bộ bằng nhiều cách phức tạp.
  + Do đó mà bộ tinh chỉnh DBMS phải xem xét cẩn thận để sắp xếp thứ tự các điều kiện chọn.
  + Ví dụ Frames.frameno < 100 is\_herbert (Frames.image)
  + Một cách tổng quát, thứ tự tốt nhất trong phép chọn là một hàm của chi phí và hệ số giảm, rank = (hệ số giảm -1 )/ cost. Nếu rank trong một truy vấn nhiều bảng lớn thì nên hoãn phép chọn lại sau khi đã thực hiện phép kết. Điều này trái ngược với quan niệm thực hiện phép chọn càng sớm càng tốt trong cơ sở dữ liệu quan hệ

1. **Thảo luận về ODL và OQL, các tiểu chuẩn cho OODBMSs.**

* Trong chương này chúng ta định nghĩa một OODBMS như một ngôn ngữ lập trình với sự hỗ trợ của các đối tượng lâu dài. Trong khi định nghĩa này phản ánh nguồn gốc của OODBMSs chính xác, và đến một mức độ nhất định tập trung thực hiện OODBMSs, thực tế là OODBMS hỗ trợ một tập ngôn ngữ truy vấn truy trên dữ liệu. Và một tiêu chuẩn được phát triển bởi ODMG và được gọi là OQL.
* OQL thì tương tự SQL, với cú pháp SELECT-FROM-WHERE (và GROUP BY, HAVING, ORDER BY cũng được hỗ trợ) và dựa trên sự mở rộng của SQL:1999.
* Ngoài ra dựa trên DDL (ngôn ngữ định nghĩa đối tượng) cho OODBMSs (Object Data Language, or ODL) thì ODL sẽ được thêm vào OODBMS, để định nghĩa đối tượng.
  1. **Mô hình dữ liệu ODMG và ODL**
* Mô hình dữ liệu ODMG là cơ bản cho OODBMS, giống như mô hình CSDL quan hệ là cơ bản cho RDBMS. Một CSDL chứa các tập của đối tượng, mà tương tự như các thực thể trong mô hình ER. Mỗi đối tượng đều có một oid duy nhất, và CSDL chứa các tập đối tượng này với các thuộc tính, mỗi tập như vậy được gọi là class.
* Các thuộc tính của class có thì theo quy định sử dụng ODL và gồm 3 loại sau: thuộc tính, mối quan hệ, và phương thức. Các định nghĩa của các loại này tương đương các định nghĩa bên lập trình hướng đối tượng.
* Mối kết hợp: có nhiều loại một-một, một-nhiều, nhiều-nhiều, ví dụ:

****

* Phương thức là các hoạt động của đối tượng, không giống với store procedure trong mô hình quan hệ.
* **ODL:**
  + Hỗ trợ các khái niệm ngữ nghĩa ngữ nghĩa của OODBMS.
  + Không phụ thuộc vào ngôn ngữ lập trình.
  + Dùng để tạo các đặc điểm của đối tượng.
  + Dùng để tạo các đặc điểm của đối tượng.
  + Không dùng để xử lý cơ sở dữ liệu
  1. **OQL**
* Ngôn ngữ truy vấn OQL của ODMG được cố tình thiết kế để có cú pháp tương tự SQL, để làm cho nó dễ dàng cho người sử dụng quen thuộc với SQL để tìm hiểu ODL. Cùng xem qua ODL bằng các ví dụ:
  + **SELECT *mname*: M.movieName, *tname*: T.theaterName**

**FROM Movies M, M.shownAt T**

**WHERE T.numshowing() > 1**

* **SELECT T.ticketPrice,*avgNum*: AVG(SELECT P.T.numshowing() FROM *partition* P)**

**FROM Theaters T**

**GROUP BY T.ticketPrice**

1. **Trình bày so sánh ngắn gọn của ORDBMSs và OODBMs**
   1. **So sánh RDBMS và ORDBMS**

* RDBMS không hỗ trợ phần mở rộng của ORDBMS.
* RDBMS dễ sử dụng hơn vì nó có ít tính năng hơn.
* RDBMS ít linh hoạt hơn ORDBMS.
  1. **So sánh OODBMS và ORDBMS: Sự giống nhau**
* Hỗ trợ các kiểu có cấu trúc, kiểu đối tượng, ADTS và kế thừa.
* Hỗ trợ ngôn ngữ truy vấn để thao tác trên kiểu collection:
  + ORDBMS: phần mở rộng của SQL.
  + OODBMS: ODL/OQL.
  + ORDBMS thêm các tính năng của OODBMS vào RDBMS.
  + OODBMS phát triển ngôn ngữ truy vấn dựa trên ngôn ngữ truy vấn quan hệ.
  1. **So sánh OODBMS và ORDBMS: Sự khác nhau**
* OODBMS thêm ngôn ngữ lập trình vào DBMS, trong khi ORDBMS thêm các kiểu dữ liệu phong phú vào RDBMS.
* OODBMS có mục tiêu là tích hợp liền mạch với ngôn ngữ lập trình, trong khi đây không phải là mục tiêu quan trọng của ORDBMS.\
* OQL không được hỗ trợ hiệu quả trong hầu hết OODBMS, ngược lại chúng là trung tâm của ORDBMS.
* OODBMS đã được tối ưu hóa để hỗ trợ cho các ứng dụng HĐT và các ngôn ngữ OO.
* ORDBMSs được hỗ trợ bở hầu hết các HQTCSDL hiện nay.

## TOPIC #8C CHUẨN NGÔN NGỮ - THIẾT KẾ CSDL HƯỚNG ĐỐI TƯỢNG

Mục lục

[1. TỔNG QUAN VỀ MÔ HÌNH ĐỐI TƯỢNG CỦA ODMG 26](#_Toc309861610)

1.[1. Object và Literal](#_Toc309861620) 26

[1.2 Xây dựng Interface cho tập các đối tượng 28](#_Toc309861619)

1.[3. Atomic (User-Defined) Objects](#_Toc309861620) 28

[1.4 Interfaces, Classes và Inheritance 31](#_Toc309861619)

[1.5 Extents, Keys và Factory Objects 32](#_Toc309861619)

[2. OBJECT DEFINITION LANGUAGE ODL.](#_Toc309861611) 34

[3. OBJECT QUERY LANGUAGE OQL 37](#_Toc309861617)

3.[1. Simple OQL Queries, database entry points, iterator variables 37](#_Toc309861619)

3.[2. Query Results and Path Expressions](#_Toc309861620) 38

3.[3. Other Features of OQL 39](#_Toc309861619)

3.[3.1. Views as Named Queries](#_Toc309861620) 40

3.[3.2. Single Elements from Singleton Collections](#_Toc309861620) 40

3.[3.3. Collection Operators (Aggregate Functions, Quantifiers) 40](#_Toc309861619)

3.[3.4. Ordered (Indexed) Collection Expressions](#_Toc309861620) 41

[3.3.5 The Grouping Operator 42](#_Toc309861619)

[4. TỔNG QUAN VỀ C++ LANGUAGE BINDING. 4](#_Toc309861620)2

5[. OBJECT DATABASE CONCEPTUAL DESIGN 4](#_Toc309861619)3

[5.1. Sự khác nhau giữa thiết kế quan niệm của ODB và RDB. 4](#_Toc309861620)3

5.[2. Chuyển đổi từ lược đồ EER sang ODB 4](#_Toc309861619)5

1. **TỔNG QUAN VỀ MÔ HÌNH ĐỐI TƯỢNG CỦA ODMG**

Mô hình đối tượng ODMG là mô hình mà ngôn ngữ định nghĩa đối tượng (ODL) và ngôn ngữ truy vấn đối tượng (OQL) là nền tảng. Mô hình này cung cấp các kiểu dữ liệu, hàm dựng kiểu và những khác niệm khác có thể được sử dụng trong ODL để xác định lược đồ CSDL đối tượng. Vì thế, mô hình ODMG là mô hình dữ liệu chuẩn cho CSDL hướng đối tượng, giống như SQL mô tả mô hình dữ liệu chuẩn cho CSDL quan hệ.

* 1. **Object và Literal**

Object và literal là các yếu tố cơ bản của mô hình đối tượng. Object có cả ID nhận biết đối tượng và giá trị, trong khi literal chỉ có giá trị mà không có ID. Giá trị của đối tượng có thể có cấu trúc phức tạp và có thể thay đổi. Tuy nhiên giá trị của literal là cố định và không thay đổi, có thể có kiểu cấu trúc.

* Object có 4 đặc điểm:
  + Identifier: là định danh duy nhất trong hệ thống (OBJECT\_ID).
  + Name: duy nhất trong một CSDL hay một chương trình. Tên của đối tượng là tùy ý. Thông thường, những tên này được sử dụng như là một entry points trong CSDL; bằng cách xác định các đối tượng này theo tên, người sử dụng có thể định vị được các đối tượng khác được tham chiếu từ những đối tượng này.
  + Lifetime: cho biết đây là *persistent object* (database object) hay *transient object* (đối tượng trong quá trình thực thi chương trình và sẽ biến mất sau khi chương trình kết thúc).
  + Structure: cho biết một đối tượng được được xây dựng bằng hàm dựng kiểu như thế nào và nó có phải là đối tượng nguyên tử hay bộ sưu tập (atomic hay collection).
* Literal:
  + Có giá trị nhưng không có ID.
  + Có ba kiểu dữ liệu:
    - Atomic: đã được định nghĩa, giá trị kiểu dữ liệu cơ bản. (short, long, float, boolean, char…)
    - Collection: một bộ sưu tập (array…) của các giá trị hay đối tượng nhưng không có Object\_id. (set<T>, bag<T>, list<T>, array<T> với T là kiểu của các đối tượng hay giá trị, Dictionary<K,V> với K là khóa, V là giá trị)
    - Structured: các giá trị được xây dựng bởi các hàm dựng kiểu (date, các biến kiểu struct …)
* Sau đây là một ví dụ đơn giản về mô hình đơn giản. Mô hình ODMG sử dụng từ khóa interface để định nghĩa một lớp. Trong thực tế, từ khóa *interface* thích hợp hơn bởi vì nó mô tả interface của các loại đối tượng, cụ thể là các thuộc tính, mối quan hệ và các phương thức. Chúng được tạo ra nhằm mục đích phục vụ cho việc định nghĩa các phương thức có thể được **kế thừa (inherited)** bởi các đối tượng do người dùng định nghĩa cho các ứng dụng riêng biệt.

interface Object {

…

boolean same\_as (in Object other\_object);

Object copy();

void delete();

};

interface Date : Object {

enum Weekday

{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

enum Month

{January, February, March, April, May, June, July, August, September, October, November, December};

unsigned short year();

unsigned short month();

unsigned short day();

…

boolean is\_equal (in Date other\_Date);

boolean is\_greater (in Date other\_Date);

…

};

Interface Collection : Object {

…

exception ElementNotFound {any element};

unsigned long cardinality();

boolean is\_empty();

…

boolean contains\_element (in any element);

void insert\_element (in any element);

void remove\_element (in any element) raises(ElementNotFound);

Iterator create\_iterator(in boolean stable);

…

};

interface Iterator {

exception NoMoreElements();

…

boolean is\_stable();

boolean at\_end();

void reset();

any get\_element() raises(NoMoreElement);

void next\_position() raises(NoMoreElement);

…

};

interface Set : Collection {

Set creation\_union(in Set other\_set);

…

boolean is\_subset\_of(in Set other\_set);

…

};

interface Bag : Collection {

unsigned long occurrences\_of(in any element);

Bag create\_union(in Bag other\_bag);

…

};

* Các phương thức được sử dụng cho các đối tượng thông qua dấu chấm.

Vd: trong lớp Object để sử dụng phương thức để so sánh với đối tượng p

**o.same\_as(p)**

hay sao chép đối tượng cho đối tượng p

**p = o.copy()**

* Ngoài ra có thể sử dụng dấu [->]: **o->same\_as(p)** hay **o->copy()**
  1. **Xây dựng Interface cho tập các đối tượng**

Một đối tượng bộ sưu tập kế thừa những phương thức và đặc điểm cơ bản từ Collection interface, ví dụ như:

* O.cardinality() trả về số phần tử trong một bộ.
* O.is\_empty() trả về true nếu bộ O là rỗng.
* O.insert\_element(E) thêm một phần tử E vào bộ O.
* O.remover\_element(E) xóa một phần tử ra khỏi bộ O.
* O.contains\_element(E) trả về true nếu bộ O có chứa phần tử E.
* I = O.create\_iterator() tạo một bản sao đối tượng I từ đối tượng O.
  1. **Atomic (User-Defined) Objects**
* Trong phần này, chúng ta sẽ biết cách tạo lớp đối tượng cho các đối tượng nguyên tử (*atomic object*). Trong ODL, chúng ta sẽ sử dụng từ khóa **class.**
* Trong mô hình đối tượng, bất kì đối tượng người dùng tự định nghĩa mà không phải là đối tượng bộ sưu tập được gọi là đối tượng nguyên tử (**atomic object**). Ví dụ, trong CSDL UNIVERSITY, ta có thể định nghĩa một lớp đối tượng cho các đối tượng Student.
* Hầu hết các đối tượng là các đối tượng có cấu trúc (structured object). Ví dụ, một đối tượng Student sẽ có cấu trúc phức tạp với nhiều thuộc tính, mối quan hệt và phương thức, nhưng vẫn được xem là nguyên tử vì nó không phải là một bộ sưu tập.
* Một đối tượng nguyên tử được định nghĩa như một lớp bằng cách xác định các đặc điểm (**properties**) và các phương thức (**operations**).
* VD:

**class** Employee

( **extent** all\_employees **key** ssn)

{

**attribute** **string** name;

**attribute** **string** ssn;

**attribute** **date** birthdate;

**attribute** **enum** Gender{M, F} sex;

**attribute** **short** age;

**relationship** Department works\_for **inverse** Department::has\_emps;

void reassign\_emp(**in** **string** new\_dname) **raises**(dname, valid);

};

**class** Department

( **extent** all\_departments **key** dname, dnumber)

{

**attribute** **string** dname;

**attribute** **short** dnumber;

**attribute** **struct** Dept\_Mgr {Employee manager, **date** startdate}

mgf;

**attribute** **set**<**string**> location;

**attribute** **struct** Projs {**string** projname, **time** weekly\_hours} projs;

**relationship set**<Employee> has\_emps **inverse**

Employee::works\_for;

**void** add\_emp(**in** **string** new\_enamel **raises**(ename\_not\_valid);

**void** change\_manager(**in** **string** new\_mgr\_name; **in** **date** startdate);

}

* + Properties:
    - Thuộc tính (attributes):
      * Một thuộc tính mô tả một số khía cạnh nào đó của đối tượng. Các thuộc tính có giá trị, thường là các literal có giá trị đơn hay cấu trúc phức tạp và được lưu trong đối tượng. Tuy nhiên, các giá trị thuộc tính có thể là Object\_Ids của các đối tượng khác. Giá trị thuộc tính còn được xác định thông qua các phương thức được sử dụng để tính giá trị thuộc tính.
      * VD: trong ví dụ trên, các thuộc tính của lớp Employee là ***name***, ***ssn***, bir***t***hdate, ***sex*** và ***age***. Các thuộc tính của Department là ***dname***, ***dnumber***, ***mgr***, ***locations*** và ***projs***. Các thuộc tính ***mgr*** và ***projs*** của ***Department*** có cấu trúc phức tạp và được định nghĩa thông qua ***struct***. Vì thế giá trị của ***mgr*** trong mỗi đối tượng ***Department*** sẽ có hai phần: ***manager*** – có giá trị là ***Object***\_Id tham chiếu từ đối tượng E***m***ployee, là người quản lý ***Department***, và ***startdate*** có giá trị là ***date***. Thuộc tính vị trí của ***Department*** được định nghĩa thông qua kiểu ***set***, vì mỗi đối tượng Department có thể có nhiều vị trí.
    - Quan hệ (Relationships):
      * Một quan hệ cho biết hai đối tượng trong CSDL có quan hệ với nhau. Trong mô hình ODMG, chỉ có mối quan hệ nhị phân (mối quan hệ đơn) và mỗi mối quan hệ được biểu diễn bằng một cặp tham chiếu (**pair of inverse references**) thông qua từ khóa ***relationship***.
      * VD: trong ví dụ trên, có một một quan hệ liên quan giữa Employee với Department là nơi nhân viên làm việc – mối quan hệ ***works\_for*** của Employee. Ngược lại, mỗi Department có quan hệ với một tập các Employee làm việc trong Department - mối quan hệ ***has\_emps*** của Department.
      * Từ khóa ***inverse*** xác định mối quan hệ đơn tương ứng ngược chiều. Bằng cách xác định mối quan hệ ngược này, hệ CSDL có thể duy trì tính toàn vẹn tham chiếu của mối quan hệ một cách tự động. Ví dụ, nếu giá trị của works\_for cho Employee e tham chiếu Department d, thì giá trị của has\_emps của Department d phải chứa một tham chiếu đến e trong tập tham chiếu đến Employee. Nếu người thiết kế CSDL muốn có mối quan hệ thể hiện trong một hướng – *only one direction*, thì nó phải được tạo như một thuộc tính hay phương thức. Một ví dụ là giá trị *manager component* của thuộc tính ***mgr*** trong Department.
  + Operations:
    - Mỗi đối tượng có thể có một số phương thức, mỗi phương thức có tên, các kiểu đối số, giá trị trả về. Tên của phương thức là duy nhất trong mỗi lớp đối tượng, nhưng vẫn có thể đặt tên giống cho các phương thức trong các lớp đối tượng khác nhau. Phương thức còn có thể xác định tên của trường hợp ngoại lệ (**exception**) có thể xảy ra trong quá trình thực thi phương thức. Ví dụ, trong lớp Employee có phương thức ***reassign\_emp*** và lớp Department có hai phương thức ***add\_emp*** và ***change\_manager***.
  1. **Interfaces, Classes và Inheritance**

Trong mô hình ODMG, có hai khái niệm tồn tại để xác định các kiểu đối tượng: **interface** và **class**. Ngoài ra, còn có hai kiểu của mối quan hệ kế thừa. Trong phần này, chúng ta sẽ thảo luận sự giống và khác nhau giữa những khái niệm này. Dựa vào các thuật ngữ ODMG, ta sử dụng từ **behavior** để thể hiện cho phương thức *operations*, và **state** để thể hiện cho các đặc tính *properties* (thuộc tính và mối quan hệ).

* **Interface** mô tả hành vi trừu tượng (**abstract behavior**)của một loại đối tượng, cho biết phương thức của đối tượng đó. Mặc dù interface có thể có các đặc tính như thuộc tính và phương thức, nhưng các đặc điểm này sẽ không được kế thừa. Interface còn có tính chất **noninstantiable** – có nghĩa là không thể tạo các đối tượng tương ứng từ định nghĩa interface. Được dùng chủ yếu cho việc xác định các phương thức trừu tượng có thể được kế thừa bởi các lớp và các interface khác, được gọi là **behavior inheritance** và sử dụng kí hiệu **“:”**. Vì thế trong mô hình đối tượng ODMG, behavior inheritance yêu cầu **supertype là interface**, **subtype là class hoặc interface**.
* Class mô tả các hành vi và đặc điểm trừu tượng của một kiểu đối tượng, có tính chất **instantianle –** có nghĩa là có thể tạo đối tượng riêng rẽ tương ứng từ định nghĩa class. Một loại quan hệ kế thừa khác gọi là **EXTENDS** và được xác định bằng từ khóa **extends**, được dùng để **kế thừa cả state và behavior** giữa các class. Trong sự kế thừa **EXTENDS**, cả supertype và subtype đều phải là class. Đa kế thừa thông qua EXTENDS không đượng chấp nhận. Tuy nhiên, đa kế thừa được cho phép cho kế thừa hành vi (behavior) thông qua dấu **“:”**. Vì thế, interface có thể kế thừa hành vi từ một số interface khác. Một class cũng có thể kế thừa hành vi từ một số interface khác thông qua dấu “:”, ngoài ra có thể kế thừa behavior và state từ một class khác thông qua EXTENDS.
  1. **Extents, Keys và Factory Objects**
* ***Extent:*** Trong mô hình ODMG, người thiết kế CSDL có thể thêm phần mở rộng **extent** cho bất kỳ loại đối tượng nào được định nghĩa bằng **class**. Phần mở rộng được đặt tên, và sẽ chứa tất cả các thể hiện đối tượng của lớp. Vì thế, phần mở rộng được xem là một tập các đối tượng chứa tất cả các thể hiện của lớp. Trong ví dụ phần 1.3 ta thấy các lớp ***Employee*** và ***Department*** có phần mở rộng gọi là ***all\_employees*** và ***all\_departments***. Phần mở rộng còn được sử dụng để tự động tuân thủ mối quan hệ set/subset giữa phần mở rộng của supertype và subtype của nó. Nếu hai lớp A và B có phần mở rộng là all\_A và all\_B, và lớp B là subtype của lớp A (có nghĩa là lớp B EXTENDS lớp A) thì bộ sưu tập của các đối tượng trong all\_B phải là subset của tất cả các đối tượng trong all\_A tại mọi thời điểm. Ràng buộc này tự động được tuân thủ bởi hệ CSDL.
* ***Key:*** Lớp với phần mở rộng có thể có một hoặc nhiều khóa. Một key chứa một hay nhiều thuộc tính hay mối quan hệ mà giá trị của nó chứa phải là duy nhất với mỗi đối tượng trong phần mở rộng. Trong phần mở rộng mục 1.3, lớp Employee có thuộc tính ***ssn*** là khóa, lớp Department có hai khóa: ***dname*** và ***dnumber***. Với những khóa kết hợp có nhiều thuộc tính hay mối quan hệ, các đặc tính đó được chứa trong dấu **()**.
* ***Factory Object***: một đối tượng có thể được dùng để phát sinh hay tạo các đối tượng riêng lẽ từ các phương thức của nó. Một số interface của factory object là một phần của mô hình đối tượng ODMG. Interface ***ObjectFactory*** có một phương thức đơn, ***new()***, sẽ trả về một đối tượng mới với một **Object\_Id*.*** Bằng cách kế thừa interface này, người dùng có thể tạo các factory interface cho mỗi kiểu đối tượng nguyên tử, và người lập trình có thể thực thi phương thức ***new*** một cách khác nhau cho mỗi kiểu đối tượng. Ví dụ bên dưới cho thấy một interface DataFactory, có phương thức tạo một **calendar\_date** mới, và một phương thức tạo một đối tượng mà giá trị của nó là **current\_date**… Ta có thể thấy, factory object cơ bản cung cấp phương thức dựng (**constructor operations**) cho các đối tượng mới.
* Ví dụ:

interface ObjectFactory{

Object new();

};

interface DateFactory : ObjectFactory{

exception InvalidDate{};

…

Date calendar\_date ( in unsigned short year,

in unsigned short month,

in unsigned short day)

raises(InvalidDate);

…

Date current();

interface DatabaseFactory{

Database new();

};

interface Database{

void open(in string database\_name);

void close()l

void bind(in any some\_object, in string object\_name);

Object unbind(in string name);

Object lookup(in string object\_name)

raises(ElementNotFound);

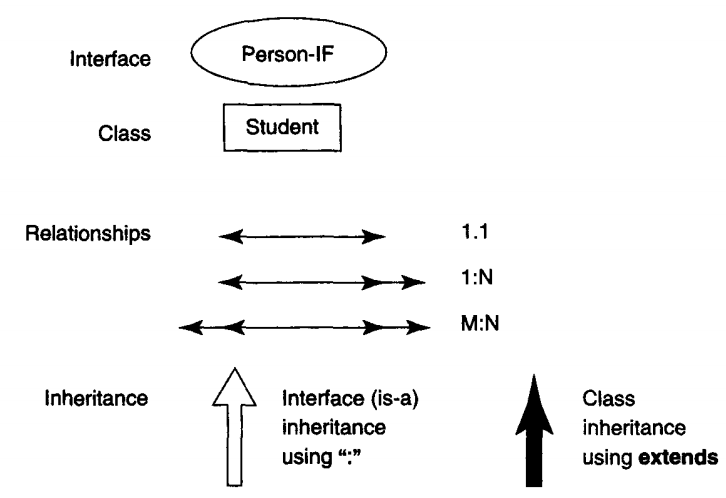
…

};

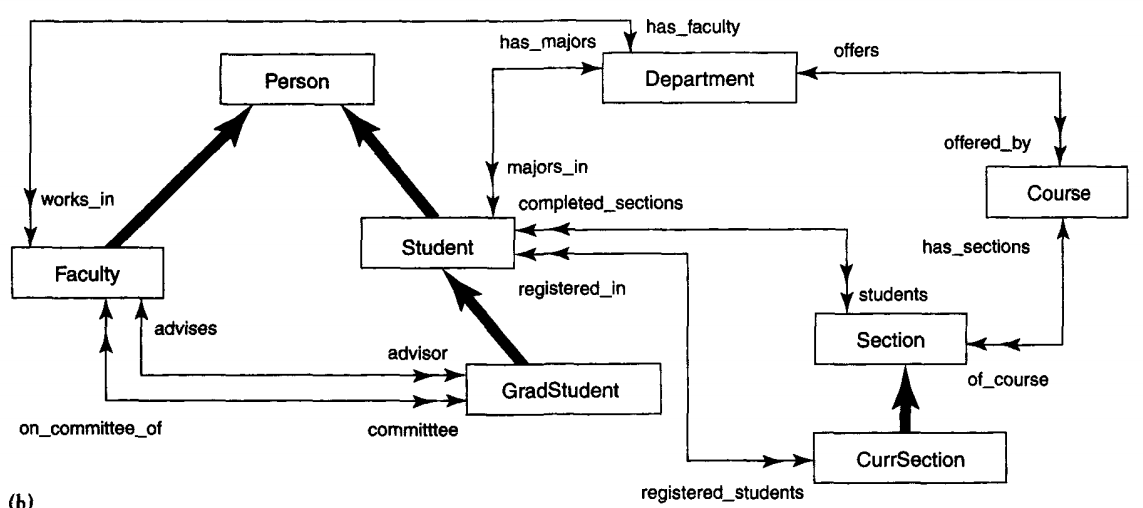
* ***Database***: bởi vì ODBMS có thể tạo nhiều CSDL khác nhau, với những lược đồ khác nhau, nên mô hình dữ liệu ODMG có các interface cho các đối tượng **DatabaseFactory** và **Database**. Mỗi CSDL có một tên CSDL riêng, phương thức **bind** có thể được dùng để gán các tên duy nhất riêng biệt vào các thể hiện đối tượng trong một CSDL cụ thể. Phương thức **lookup** trả về một đối tượng từ CSDL có tên **object\_name**, và phương thức unbind lấy tên ra khỏi các thể hiện đối tượng đã được đặt tên từ CSDL.

1. **OBJECT DEFINITION LANGUAGE ODL**

* ODL được thiết kế để hỗ trợ các khái niệm ngữ nghĩa của mô hình ODMG và không phụ thuộc vào bất kì ngôn ngữ lập trình nào. Mục đích chính của nó là dùng để tạo các đối tượng – class và interface. Vì thế ODL không phải là một ngôn ngữ lập trình. Người dùng có thể xác định lược đồ trong ODL mà không cần phụ thuộc vào ngôn ngữ lập trình, và sử dụng ngôn ngữ liên kết để xác định các khái niệm ODL có thể **map** với các khái niệm trong ngôn ngữ lập trình như C++, SMALLTALK và JAVA.
* Các kí hiệu trong ODL:



* Ví dụ mô hình CSDL ***UNIVERSITY***:



* Ví dụ tạo một số đối tượng bằng class trong CSDL UNIVERSITY bằng ODL:

**class** Person

(**extent** persons **key** ssn)

{

**attribute** **struct** Pname {**struct** fname,

**string** mname, **string** Inme} name;

**attribute** **string** ssn;

**attribute** **date** birthdate;

**attribute** **enum** Gender{M, F} sex;

…

**Short** age();

};

**class** Faculty **extends** Person

(**extent** faculty)

{

**attribute** **string** rank;

**attribute** **float** salary;

**attribute** **string** phone;

…

**relationship** Department works\_in

**inverse** Department::has\_faculty;

**relationship** **set**<GradStudent> advises

**inverse** GradStudent::advisor;

**void** give\_raise (**in** **float** raise);

**void** promote (**in** **string** new\_rank);

};

**class** Department

( **extent** departments **key** dname)

{

**attribute** **string** dname;

**attribute** **string** dphone;

**attribute** **string** doffice;

**attribute** **string** college;

**attribute** Faculty chair;

**ralatlonshlp** **set**<Faculty> has\_faculty **inverse** Faculty::works\_in;

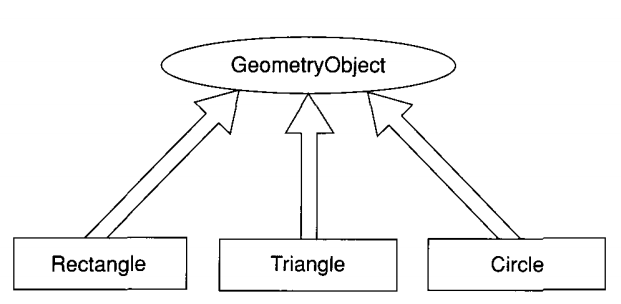
**relationship** **set**<Student> has\_majors **inverse** Student::majors\_in;

**relationship** **set**<Course> offers **inverse** Course::offered\_by;

};

…

* Ví dụ thể hiện interface và kế thừa:



* Ví dụ tạo các đối tượng trong ví dụ interface và kế thừa:

**interface** GeometryObject {

**attribute enum** Shape {Rectangle, Triangle, Circle, …} shape;

**attribute** **struct** Point{**short** x, **short** y} reference\_point;

**float** perimeter ();

**float** area();

**void** translate (**in short** x\_translation; **in short** y\_translation);

**void** rotate (**in float** angle\_of\_rotation)l

};

**class** Triangle : GeometryObject

(**extent** triangles) {

**attribute** **struct** Point{**short** x, **short** y} reference\_point;

**attribute** **short** length;

**attribute** **short** height;

attribute float orientation\_angle;

};

* Đa kế thừa (multiple inheritance) của interface bởi class được cho phép, giống như việc đa kế thừa của interface từ một interface. Tuy nhiên, với kế thừa EXTENDS (class), đa kế thừa không được sử dụng. Vì thế, một class có thể kế thừa bằng EXTNEDS từ một class.

1. **OBJECT QUERY LANGUAGE OQL**

OQL là ngôn ngữ truy vấn được đưa ra cho mô hình đối tượng ODMG. Nó được thiết kế để làm việc gần gũi hơn với ngôn ngữ lập trình mà liên kết ODMG đã được định nghĩa, như C++, SMALLTALK và JAVA. Vì thế, truy vấn OQL được thêm vào ngôn ngữ lập trình có thể trả về các đối tượng có kiểu thích hợp với kiểu của ngôn ngữ lập trình. Các phương thức thực thi của class có thể được viết bằng mã trong ngôn ngữ lập trình. Cú phá OQL tương tự như cú pháp của chuẩn SQL, thêm vào các khái niệm ODMG như object identity, complex object, operation, inheritance, polymorphism và relationship.

**3.1. Simple OQL Queries, database entry points, iterator variables**

* Cú pháp cơ bản của OQL là **select… from… where…**, giống như của SQL.

VD: Q0:

**SELECT** d.dname

**FROM** d **in** department

**WHERE** d.college = ‘Engineering’;

* Thông thường, entry point đến CSDL là cần thiết cho mỗi câu truy vấn, là tên đại diện cho đối tượng - ***named persistent object***. Với nhiều câu truy vấn, entry point là tên của phần **extent** của class. Tên phần mở rộng được xem là tên của một thể hiện đối tượng mà kiểu của nó là một bộ sưu tập của các đối tượng trong class. Ví dụ, tên của đối tượng **departments** là kiểu **set**<**Department**>; **persons** là kiểu **set<Person>…**
* Việc sử dụng tên mở rộng – **departments** trong Q0 – như một entry point để thể hiện một tập thể hiện của các đối tượng. Bất cứ khi nào một bộ sưu tập được tham chiếu đến trong OQL, chúng ta nên định nghĩa một biến **iterator variable – d**  trong Q0 – duyệt qua mỗi đối tượng trong bộ sưu tập. Trong nhiều trường hợp, câu truy vấn sẽ chọn các đối tượng từ bộ sưu tập, dựa trên các điều kiện đã được chỉ định trong WHERE. Trong Q0, chỉ các đối tượng d trong bộ sưu tập của **department** thỏa điều kiện **d.college = ‘Engineering’** được chọn cho kết quả. Với mỗi đối tượng được chọn d, giá trị **d.name** được gọi trong kết quả. Vì thế, kiểu của kết quả cho Q0 là **bag<string>** bởi vì kiểu của mỗi giá trị **dname** là string. Trong thực hết, kết quả của truy vấn sẽ là kiểu **bag** cho **select… from…** và kiểu **set** cho **select distinct… from…**.
* Sử dụng ví dụ trong Q0, có 3 cú pháp để định nghĩa iterator variables:
  + d **in** department
  + department d
  + department as d

**3.2. Query Results and Path Expressions**

* Kết quả truy vấn có thể là bất kì kiểu gì có thể được thể hiện trong mô hình ODMG. Một câu truy vấn không nhất thiết phải theo cấu trúc select… from… where; trong trường hợp đơn giản nhất, bất kỳ tên đại diện nào cũng là câu truy vấn, kết quả được tham chiếu đến thể hiện đối tượng đó. Ví dụ:

**Q1: departments;**

* Khi entry points được sử dụng, khái niệm **path expression** có thể được sử dụng để xác định đường dẫn đến các thuộc tính đối tượng liên quan. Path expression thông thường bắt đầu tại *persistent object name* hay tại iterator variable để duyệt qua các đối tượng trong bộ sưu tập. Theo sau tên là dấu chấm (.) và tên của thuộc tính hay quan hệ có liên quan. Ví dụ:
  + **Q2: csdepartment.chair;**
  + **Q2a: csdepartment.chair.rank;**
  + **Q2b: csdepartment.has\_faculty;**
* Trong Q2 và Q2a, kết quả trả về sẽ là những giá trị đơn, truy nhiên kết quả ở Q2b lại là một bộ sưu tập có kiểu set.
* Ví dụ trả về ranks của khoa computer science:
  + **Q3: csdepartment.has\_faculty.rank;**
  + **Q3a: select f.rank**

**from f in csdepartment.has\_faculty;**

* + **Q3b: select distinct f.rank**

**from f in csdepartment.has\_faculty;**

* Q3 và Q3a trả về kiểu bag, Q3b trả về set.
* Ví dụ trả về bộ sưu tập của các sinh viên tốt nghiệp trong khoa KHMT:
  + Q4: csdepartment.chair.advises;
  + Q4a: **select struct** (name: s**t**ruct (last\_name: s.name.Iname,

first\_name:s.name.fname),

degrees: (**select** **struct** (deg: d.degree,

yr: d.year, college: d.college)

**from** d **in** s.degrees))

**from** s **in** csdepartment.chair.advises;

* Trong câu truy vấn Q4a đã có sử dụng cấu trúc phức tạp trong kết quả trả về. Cụ thể là tên và bằng cấp của sinh viên.
* **Lưu ý:** OQL trực giao với với việc xác định path expression. (OQL is *orthogonal* with respect to specifying path expressions)
  + Tên các thuộc tính, mối quan hệ và phương thức có thể được sử dụng thay thế lẫn nhau.
  + Ví dụ: gọi điểm trung bình của tất cả sinh viên năm cuối trong khoa KHMT, với kết quả được sắp xếp theo gpa.

Q5a:

**Select** **struct** (last\_name: s.name.lname, first\_name: s.name.fname, gpa: s.gpa)

**From** s **in** csdepartment.has\_majors

**Where** s.class = ‘senior’

**Order** **by** gpa **desc**, last\_name **asc**, first\_name **asc**;

Q5b:

**Select** **struct** (last\_name: s.name.lname, first\_name: s.name.fname, gpa: s.gpa)

**From** s **in** students

**Where** s.majors\_in.dname = ‘Computer Science’ **and** s.class = ‘senior’

**Order** **by** gpa **desc**, last\_name **asc**, first\_name **asc**;

**3.3. Other Features of OQL**

**3.3.1. Views as Named Queries**

- Từ khóa **define** trong OQL dùng để xác định định danh cho **named query**.

- Tên phải là duy nhất. Nếu không, kết quả hiện tại sẽ thay thế cho named query đang tồn tại.

- Khi một định nghĩa quey được tạo, nó sẽ tồn tại cho đến khi bị xóa hoặc định nghĩa lại.

- Một định nghĩa khung nhìn có thể bao gồm các đối số.

- Ví dụ: một view gồm các sinh viên trong một khoa có minor

**Define** has\_minor(dept\_name) **as**

**Select** s

**From** s **in** students

**Where** s.minor\_in.dname = dept\_name

* Trong đó, **has\_minor** là tên của view, **dept\_name** là đối số.
* has\_minor có thể được sử dụng như câu truy vấn.
* Kết quả trả về một tập có kiểu bag các sinh viên minoring trong khoa KHMT:

**has\_minors(‘Computer Science’);**

**3.3.2. Single Elements from Singleton Collections**

- Câu truy vấn OQL trả về kiểu bộ sưu tập.

- Toán tử element của OQL có thể được sử dụng để trả về các phần tử đơn từ một bộ sưu tập đơn chứa một phần tử:

Q6: **element** (**select** d

**from** d **in** departments

**where** d.dname = 'Computer Science');

- Kiểu của kết quả: **d:Department**

**-** Nếu ***d*** là rỗng hay chứa hơn một phần tử, **exception** sẽ được thực thi.

**3.3.3.** **Collection Operators (Aggregate Functions, Quantifiers)**

**-** OQL hỗ trợ một số toán tử tập hợp có thể áp dụng cho câu truy vấn.

**-** Các toán tử tập hợp và tính toán trên bộ sưu tập bao gồm: ***min, max, count, sum, avg.***

**-** **Count** trả về kết quả có kiểu interger; những toán tử khác trả về kết quả có kiểu giống với kiểu của bộ sưu tập.

- Ví dụ toán tử tập hợp: tính trung bình GPA của tất cả các chuyên ngành cao cấp trong Business:

Q8: **avg** (**select** s.gpa

**from** s **in** students

**where** s.majors\_in.dname = ‘Business’ **and** s.class = 'senior');

**- Membership và Quantification:**

* OQL hỗ trợ các toán tử membership và quantification như sau:
  + (e **in** c) là true nếu e thuộc bộ sưu tập c.
  + (**for** **all** e **in** c: b) là true nếu tất cả các phần tử e của bộ sưu tập c thỏa b.
  + (**exists** e **in** c: b) là true nếu ít nhất có một e trong bộ sưu tập c thỏa b.
* Ví dụ: gọi tên của tất cả các sinh viên đã hoàn thành khóa học có tên là ‘Database Systems’

**select** s.name.lname, s.name.fname

**from** s **in** students

**where** 'Database Systems' **in**

(**select** c.cname

**from** c **in** s.completed\_sections.section.of\_course);

* Kiểu của kết quả: **bag<struct(string, string)>**

**3.3.4. Ordered (Indexed) Collection Expressions**

**-** Các bộ sưu tập có kiểu list hoặc array cho phép gọi các phần tử đầu tiên, cuối cùng hay ở bất kì vị trí nào.

**-** OQL cung cấp các phương thức để trích một sub-collectuon hay nối hai list lại.

**-** OQL cũng cung cấp các phương thức để sắp xếp các kết quả.

**-** Ví dụ: lấy last name của thành viên có lương cao nhất:

**Q14:** **first** ( **select** **struct**(faculty: f.name.lname, salary: f.salary)

**from** f **in** faculty

**order** **by** f.salary **desc**);

- Ví dụ: lấy 3 môn trong khoa KHMT đầu tiên dựa vào giá trị gpa:

Q15: (**select struct**( last\_name: s.name.lname,

first\_name: s.name.fname, gpa: s.gpa)

**from** s **in** csdepartment.has\_majors

**order** **by** gpa **desc**) **[0:2]**;

**3.3.5. The Grouping Operator**

**-** OQL còn hỗ trợ phương thức gom nhóm **group by**.

**-** Ví dụ: lấy trung bình GPA của các môn trong mỗi khoa có hơn 100 môn

**Select** deptname., avg\_gpa: **avg** (**select** p.s.gpa **from** p **in** **partition**)

**From** s **in** students

**Group** **by** deptname: s.majors\_in.dname

**Having** **count**(**partition**) > 100;

1. **TỔNG QUAN VỀ C++ LANGUAGE BINDING**

* Ngôn ngữ liên kết C++ cho biết cấu trúc OQL tương thích như thế nào với cấu trúc của C++, bao gồm:
  + Thư viện lớp C++: cung cấp các lớp và các phương thức thực thi cấu trúc ODL.
  + Ngôn ngữ xử lý dữ liệu – Data Manipulation Language (ODL/OML): cho biết các đối tượng CSDL được gọi và tính toán như thế nào trong ngôn ngữ C++, là cơ sở cho cú pháp và ngữ nghĩa cho ngôn ngữ lập trình C++.
  + Một tập các cấu trúc gọi là physical pragmas (cho phép người lập trình một số điều khiển trên các lưu trữ vật lý).
* Lớp thư viện thêm C++ vào chuẩn ODMG để sử dụng tiếp đầu ngữ d\_ cho các khai báo lớp làm việc với CSDL quan niệm. Người lập trình nên biết rằng chỉ có một ngôn ngữ đang được sử dụng, không phải là hai ngôn ngữ riêng biệt. Với người lập trình làm việc với các đối tượng CSDL trong chương trình**, lớp d\_Ref<T>** được định nghĩa cho mỗi lớp cơ sở dữ liệu T trong lược đồ. Do đó, các biến chương trình kiểu d\_Ref<T> có thể tham chiếu đến các đối tượng liên tục (persistent object) hay tạm thời (transient object) của lớp T. Để sử dụng nhiều kiểu dựng trong mô hình đối tượng ODMG như collection type, nhiều template class được định nghĩa trong thư viện. Ví dụ như lớp trừu tượng d\_Object<T> xác định phương thức được kế thừa bởi tất cả các đối tượng. Tương tự như vậy, một lớp trừu tượng d\_Collection<T> xác định các phương thức của bộ sưu tập. Những lớp này không tạo đối tượng, chúng chỉ quy định các thao tác mà được kế thừa từ các đối tượng và đối tượng collection tương ứng.
* Template class: hỗ trợ các kiểu bộ sưu tập trong ODMG gồm **d\_Set<T>, d\_List<T>, d\_Bag<T>, d\_Varray<T>, d\_Dictionary<T>**. Vì thế người lập trình có thể tạo các class của các kiểu như **d\_Set<d\_Ref<Student>>** mà giá trị của nó là một tập các tham chiếu đến đối tượng **Student**.
* C++ ODL cho phép người dùng xác định các lớp của lược đồ CSDL sử dụng cấu trúc của C++ cũng như cấu trúc được hỗ trợ bởi thư viện CSDL đối tượng. Để xác định các kiểu dữ liệu của thuộc tính, các kiểu cơ sở như **d\_Short (short interger), d\_UShort (unsigned short integer), d\_Long (long integer) và d\_Float (floating point number)** đều được hỗ trợ. Ngoài ra, còn một số kiểu literal có cấu trúc cũng được hỗ trợ như **d\_String, d\_Interval, d\_Date, d\_Time và d\_Timestamp**.
* Để xác định các mối quan hệ, tiền tố Rel\_ được sử dụng với hậu tố là tên các kiểu như **d\_Rel\_Ref<Dept, has\_majors> majors\_in.** Liên kết C++ còn cho phép tạo extents thông qua sử dụng lớp thư viện d\_Extent: **d\_Extent<Person> All\_persons (DB1).**

1. **OBJECT DATABASE CONCEPTUAL DESIGN**

**5.1. Sự khác nhau giữa thiết kế quan niệm của ODB và RDB**

* Sự khác nhau cơ bản là các mối quan hệ và kế thừa. Các phương thức trong ODB được thể hiện vì chúng là một phần trong định nghĩa của class.
* Mối quan hệ trong ODB:
  + Các mối quan hệ được xây dựng bằng các thuộc tính tham chiếu bao gồm OID của các đối tượng liên quan.
  + Đối tượng đơn hay bộ đều được cho phép sử dụng để tham chiếu.
  + Mối quan hệ nhị phân có thể được thể hiện đơn chiều hay hai chiều thông qua toán tử **inverse**.
* Mối quan hệ trong RDB:
  + Mối quan hệ giữa các bộ được xác định bằng các thuộc tính có giá trị tương thích với nhau (thông quan khóa ngoại).
  + Khóa ngoại là giá trị đơn.
  + Mối quan hệ nhiều – nhiều phải được thể hiện thông qua một quan hệ riêng (bảng).
* Kế thừa:
  + Cấu trúc thừa kế được xâu dựng trong ODB (được sử dụng thông qua dấu “:” và toán tử **extends**).
  + RDB không hỗ trợ mối quan hệ kế thừa, tuy nhiên có một số lựa chọn để chuyển mối quan hệ kế thừa sang RDB.
* Một sự khác nhau chính nữa giữa ODB và RDB là các phương thức xử lý:
  + ODB: các phương thức được xác định trong quá trình thiết kế (là một phần của lớp).
  + RDB: các phương thức bị hoãn cho đến khi thực thi.

**5.2. Chuyển đổi từ lược đồ EER sang ODB**

* Chuyển từ lược đồ EER sang ODB rất đơn giản vì lược đồ ODB hỗ trợ cho các mối quan hệ kế thừa.
* Khi chuyển đổi được hoàn thành, các phương thức phải được thêm vào lược đồ ODB bởi vì lược đồ EER không chứa các phương thức.
* Các bước chuyển đổi:
  + B1: tạo một lớp ODL cho mỗi thực thể EER hay phân lớp.
    - Thuộc tính đa trị:
      * List: giá trị thuộc tính được sắp xếp.
      * Bag: cho phép các giá trị trùng nhau.
      * Set: các trường hợp còn lại.
    - Các thuộc tính kết hợp: sử dụng kiểu tuple.
    - Các thuộc tính khóa là khóa của phần mở rộng extent.
    - Xác định phần mở rộng cho mỗi class và các thuộc tính khóa.
  + B2: thêm mối quan hệ hay các thuộc tính tham chiếu cho các mối quan hệ nhị phân trong các lớp ODL:
    - Các kiểu mối quan hệ: giá trị đơn cho 1 – 1 và N – 1, tập giá trị cho 1 – N và quan hệ 2 chiều cho M – N.Mối quan hệ nhị phân: thêm các đặc điểm mối quan hệ hay tham chiếu vào các lớp ODL có tham gia vào MQH. Tham chiếu mối quan hệ hai chiều nếu có với các class khác.
    - Thuộc tính mối quan hệ: tạo thông qua hàm dựng bộ (tuple constructors).
  + B3: thêm các phương thức vào mỗi lớp
    - Các phương thức không có trong lược đồ EER; các yêu cầu ban đầu phải được xem xét.
    - Các hàm dựng và hủy thích hợp phải được thêm vào.
  + B4: xác định các mối quan hệ kế thừa thông qua **extends**
    - Lớp ODL tương ứng với một sub-class trong lược đồ EER kế thừa kiểu và phương thức của super-class của nó trong lược đồ ODL.
    - Các thuộc tính khác của sub-class được thêm vào như bước 1 – 3.
  + B5: nối những kiểu thực thể yếu giống như những thực thể bình thường.
    - Các thực thể yếu không tham gia vào bất kì mối quan hệ nào có thể được biểu diễn như thuộc tính đa giá trị kết hợp (composite multi-values attribute) của thực thể chủ: set<struct<. . . >> hay list<struct<. . .>>.
  + B6: nối các category (union type) vào ODL
    - Tiến trình không phức tạp.
    - Có thể theo các nối tương tự được sử dụng cho mô hình quan hệ EER:
      * Xác định class đại diện cho category.
      * Định nghĩa mối quan hệ 1:1 giữa category và mỗi super-class của nó.
    - B7:mối quan hệ nhiều nhiều
      * Mỗi mối quan hệ được chuyển thành một lớp riêng biệt với MQH tham chiếu đến các lớp tham gia.