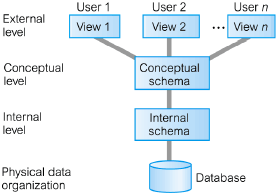
**Chương 1**

1. **Roles in the Database Evironment**:
   * 1. ***Database Administrator*** (***DBA***): cho phép truy cập vào DB, điều phối & giám sát việc sử dụng nó & cho việc mua phần mềm & tài nguyên phần cứng khi cần thiết.
     2. ***Database Designers***: xác định các data đc lưu vào DB, chọn các cấu trúc phù hợp để thể hiện & lưu trữ data.
     3. Application Programmers
     4. End Users
2. **Characteristics of the Database Approach**:
   * 1. Tự mô tả tính chất của hệ CSDL
     2. Tách biệt giữa programs & data, trừu tượng hóa data.
        1. Program-data independence + Program-operation independence = Data abstraction
        2. A data model là 1 kiểu data abstraction
     3. Support multiple views of the data
     4. Chia sẽ data & quá trình transaction của multi-user
3. **Mục tiêu của Three-Schema Architecture**:
   * 1. Tất cả user đều có thể truy xuất cùng data
     2. Người dung ko cần phải biết về chi tiết lưu trữ vật lý
     3. DBA có khả năng thay đổi cấu trúc lưu trữ mà ko ảnh hưởng đến users’ views.
     4. Cấu trúc bên trong của database ko nên bị ảnh hưởng bởi việc thay đổi vật lý của việc lưu trữ
     5. DBA có khả năng thay đổi cấu trúc ý niệm của database mà ko ảnh hưởng đến tất cả users.
4. **Three-Schema Architecture**:

****

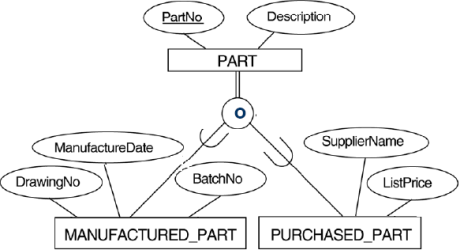
* 1. ***External Level***:
     1. Users’ view of the database
     2. Miêu tả 1 phần của database thích hợp với 1 user cụ thể
  2. ***Conceptual Level***:
     1. Cộng đồng hiển thị database
     2. Dữ liệu mô tả những gì đc luu trữ trong CSDL & các mối quan hệ giữa các dữ liệu
  3. ***Internal Level***:
     1. Dữ liệu vật lý của database trên 1 máy tính
     2. Mô tả cách mà data lưu trữ trên database

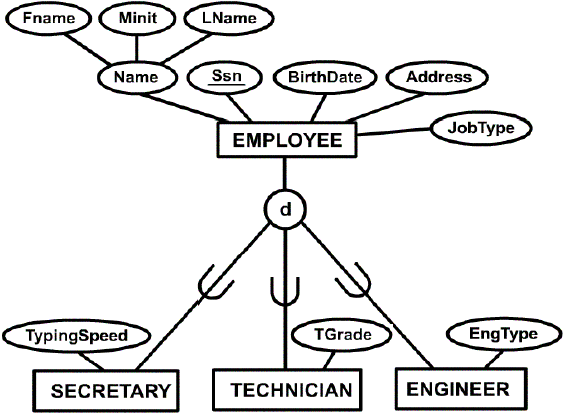
1. **Data Independence**: Độc lập dữ liệu là khả năng thay đổi CSDL tại 1 cấp độ of hệ CSDL mà ko cần thay đổi sơ đồ ở cấp độ cao hơn kế tiếp
   * 1. ***Logical Data Independence***:
        1. Đề cập đến khả năng ko bị ảnh hưởng của external schemas khi thay đổi conceptual schema.
        2. Conceptual schema thay đổi ko yêu cầu phải thay đổi external schema hay rewrites of application programs
     2. ***Physical Data Independence***:
        1. Đề cập đến khả năng ko bị ảnh hưởng của conceptual schema khi thay đổi internal schema.
        2. Internal schema thay đổi (ex: dùng cách tổ chúc file khác, cấu trúc lưu trữ/thiết bị) ko cần thay đổi conceptual/external schemas
2. **Database Languages**:
   * 1. ***Data Definition Language*** (***DDL***) cho phép DBA/user mô tả & các tổ chức, tên, thuộc tính & mối quan hệ cần thiết cho các ứng dụng thêm vào bất kỳ liên kết toàn vẹn & ràng buộc bảo mật.
     2. ***Data Manipulation Language*** (***DML***) cung cấp các tác vụ cơ bản thao tác lên dữ liệu lưu trong database.
     3. ***Data Control Language*** (***DCL***) xác định các hoạt động mà ko có trong phân loại DDL & DML, như gán quyền cho users.

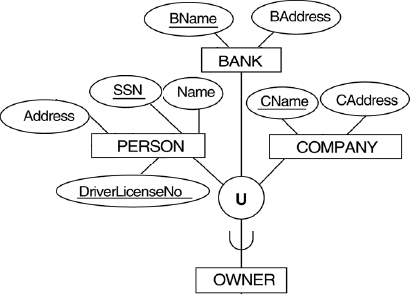
Chương: **Conceptual DB Design: ER & EER Models**

1. **ER Model**:
   * 1. ER model là dữ liệu đc tổ chức luận lý bên trong hệ CSDL
     2. ER model dựa trên mô hình dữ liệu quan hệ
     3. Dùng ER data model vì:
        1. Yêu cầu của người dung có thể đc chỉ định chính thức & rỏ ràng
        2. Conceptual data model là độc lập với bất kỳ DBMS cụ thể
        3. Nó ko bao hàm bất kỳ chi tiết vật lý hay hiện thực
        4. Người dung bình thường có thể dễ dàng hiểu nó
        5. Nó cung cấp 1 cầu nối hiệu quả giữa các yêu cầu người dùng ko chính thức & thiết kế CSDL hợp lý & thực tiển
2. **Entities & Attributes**:
   * 1. Thực thể là những object cụ thể hay các vật gì đó trong thế giới thu nhỏ miêu tả database.
     2. Attributes là các thuộc tính dung mô tả 1 entity
     3. Một entity cụ thể sẽ có 1 giá trị cho mỗi thuộc tính của nó
     4. Mỗi attribute có 1 tập các giá trị (data type) tương ứng với nó
     5. ***Type of Attribute***:
        1. ***Simple***: mỗi entity có 1 kiểu giá trị đơn vị cho attribute
        2. ***Composite***: attribute có thể đc kết hợp bởi vài thành phần. Vd: Address(Apt#, House#, Street, City, State, ZipCode, Country)
        3. ***Multi-valued***: 1 entity có thể có nhiều values cho 1 attribute. Vd: Color của 1 Car.
     6. Composite & các attribute đa trị có thể đc lồng nhau tùy ý đến bất kỳ số lượng level, mặc dù điều này là hiếm
     7. Thuộc tính kế thừa
3. **ERR: *Specialization & Generalization***:
   1. ***Disjointess Constraint***: 1 entity có thể là thể hiện của nhiều nhất là 1 subclasses. Ký hiệu là **d** trong EER diagram.
   2. ***Overlap Constraint***: 1 entity có thể là thể hiện của nhiều hơn 1 subclass. Ký hiệu là **o** trong EER diagram
   3. ***Complete***: 1 thể hiện của superclass phải là subclass. Ký hiệu là đường kép trong EER diagram
   4. ***Incomplete***: 1 thể hiện của superclass có thể ko phụ thuộc vào bất kỳ subclasses nào. Ký hiệu là 1 đường đơn.
   5. ***Category or Union type***: T là class con của n supeclasses D1, D2, … Dn, n>1:

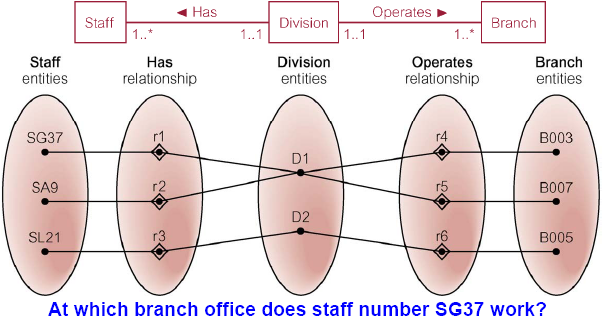
Với pi là thuộc tính của T.



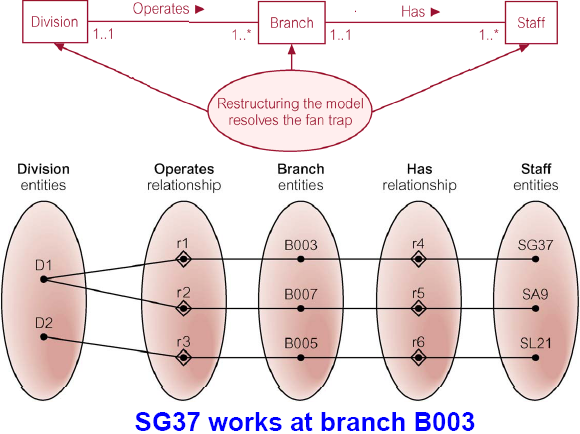




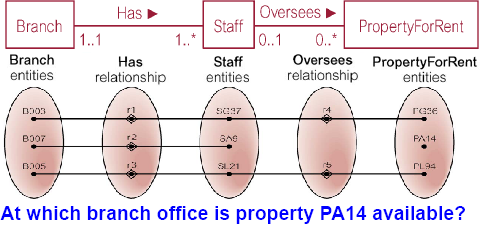
1. **Problems with ER Models**: Vấn đề có thể phát sinh khi thiết kế 1 mô hình dữ liệu khái niệm, gọi là connection trap. Điều này xảy ra thường là do sự giải thích sai về ý nghĩa của mối quan hệ nhất định. Có 2 kiệu chính của connection traps là:
   1. **Fan Trap**: Trường hợp 1 mô hình đại diện cho 1 mối quan hệ giữa các kiểu thực thể, nhưng đường giữa thực thể xuất hiện. Thông thường: 2 hay nhiều quan hệ 1:N xuất fát từ cùng 1 entity.



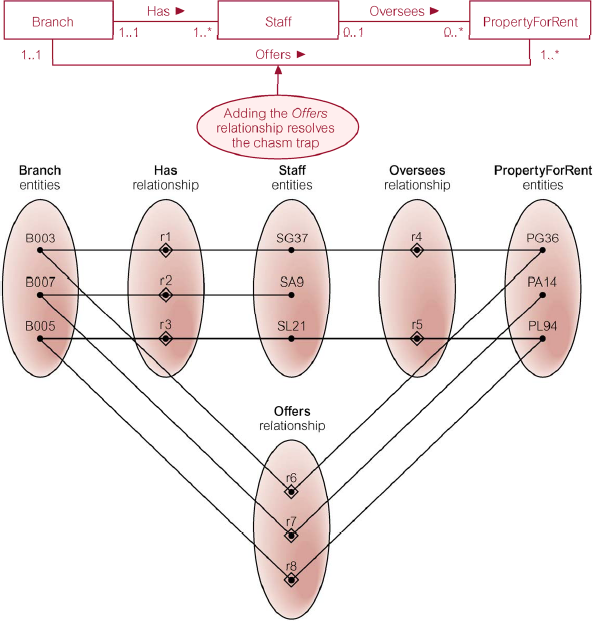
***Giải quyết***: tái cấu trúc lại ER model để bỏ đi Fan Trap



* 1. **Chasm Trap**: Trường hợp 1 mô hình gợi ý sự tồn tại của 1 mối quan hệ giữa các kiểu thực thể, nhưng đường nối ko tồn tại giữa các thực thể trong một số trường hợp



***Giải pháp***: Thêm quan hệ



**Logical DB Design & Relational Model**

1. **Ràng buộc toàn vẹn quan hệ**: Ràng buộc là 1 điều kiện mà tất cả quan hệ phải thỏa. Có 3 kiểu ràng buộc chính là:
   1. ***Key, primary key & foreign key***:
      1. ***Superkey***: là tập các thuộc tính của R mà ko có 2 dòng nào trong các quan hệ hợp lệ có cùng trị superkey.
      2. ***Key***: là tập superkey nhỏ nhất. Nếu bỏ đi 1 attribute nào trong key thì nó sẽ ko còn là superkey nữa
      3. Nếu quan hệ có vài khóa dự tuyển, 1 trong số đó sẽ đc chọn tùy ý làm Key. Khóa chính đc gạch chân
   2. ***Ràng buộc toàn vẹn thực thể***:
   3. ***Toàn vẹn tham khảo***:
      1. Ràng buộc toàn vẹn thực thể liên quan đến 2 quan hệ, dùng để xác định 1 mối quan hệ giữa các dòng của 2 quan hệ: referencing relation & referenced relation.
      2. Các dòng trong referencing relation R1 có foreign key FK, FK tham khảo đến khóa chính PK của referenced relation R2. Dòng t1 trong R1 tham khảo t2 trong R2 khi: t1[FK]=t2[PK]
      3. Trị của FK trong R1 có thể là:
         1. 1 giá trị mà tồn tại trong các giá trị khóa chính trong R2
         2. Là giá trị NULL: trong trường hợp này thì FK trong R1 ko nên là 1 phần của khóa chính R1.

Ngoài ra còn có:

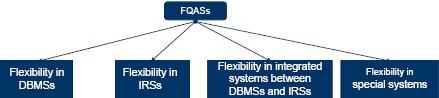
* 1. ***Null***: biểu diễn trị của 1 attribute mà hiện tại ko biết, ko dùng đc cho 1 dòng. Có thể tương ứng với incomplete hay trị ngoại lệ; hay thể hiện trị chưa có & nó ko giống với trị 0 hay khoản trắng
  2. ***Ràng buộc toàn vẹn ngữ nghĩa***: Dựa trên ngữ nghĩa của ứng dụng & ko thể mô tả trong model
  3. State/static constraints
  4. Transition/dynamic constraints

1. **ER & ERR to Relational Mapping**:
   1. ***ER***:
      1. Step 1: Mapping of Regular Entity Types
      2. Step 2: Mapping of Weak Entity Types
      3. Step 3: Mapping of Binary 1:1 Relationship Types
      4. Step 4: Mapping of Binary 1:N Relationship Types
      5. Step 5: Mapping of Binary M:N Relationship Types
      6. Step 6: Mapping of Multivalued attributes
      7. Step 7: Mapping of Relationship Types
   2. ***ERR***:
      1. Step 8: Options for Mapping Specialization or Generalization
      2. Step 9: Mapping of Union Type (Categories)
2. ***Multiple Inferitance***:
   1. ***Shared subclass*** (như Student\_Assistant) là 1 subclass của nhiều class, là đa thừa kế. Các class này phải có cùng khóa chính; nếu ko thì shared subclass sẽ là mô hình như ***Category***.
   2. ***Union Types*** (***Categories***) khi các superclass ko có cùng khóa chính, ta phải tìm 1 khóa mới (khóa thay thế) & dung nó trong các quan hệ tương ứng.

**Advanced Indexing Techniques**

**& Flexible Query Answering in DBs**

1. **Main problem**:
   * 1. Vấn đề cơ bản là cách đánh địa chỉ hiệu quả & linh động khi tập kết quả truy vấn trả về từ DBMSs là rổng.
     2. ***Flexible Query Answering Systems*** (***FQASs***) là khả năng thỏa mãn yêu cầu thông tin của người dùng (ở một level cụ thể nào đó)



* + 1. Mặc dù FQASs linh hoạt ở mức độ khác nhau, trong ngữ cảnh này chúng ta có thể suy ra khả năng cung cấp truy cập dễ dàng, thông tin, truy xuất dữ liệu trực quan & nhiều thông tin cho mỗi kiểu dự liệu cần thiết. Để đạt đc mục tiêu này trong RDBMSs, chúng ta phải mở rộng & tạo điều kiện cho họ có khả năng truy cập linh hoạt, mà ko có sẵn cho hầu hết các RDBMSs hiện có.

1. **Main challenges**:
   * 1. ***Modeling the system*** (i.e FQAS) chức năng, biểu diễn ngữ nghĩa của dữ liệu, phương thức hiện thực tốt.
     2. ***Phát triển hệ thống linh hoạt***: ko chỉ truy vấn linh hoạt mà chính nó cũng phải linh hoạt
     3. ***Xây dựng hệ thống trong suốt & bảo đảm***: sửa đổi của bất kỳ phần nào của hệ CSDL sẽ ko gây ra vấn đề cho người dung & đúng đắn của các kết quả trả về phải đc đảm bảo



1. **R-Tree**:
   1. ***Tính chất***:
      1. Cây cân bằng chiều cao như B-Tree.
      2. Node lá: (I, tuple-identifier)
         1. ***I = (I0, I1, … In)***: n là số chiều của hình chữ nhật bao không gian đối tượng
         2. ***tuple-identifier***: xác định 1 dòng trong database
      3. Ko phải node lá: (I, child-pointer)
         1. ***I*** là hình chữ nhật bảo phủ node ở mức thấp hơn
         2. ***Child-pointer***: chỉ đến địa chỉ của node thấp hơn
      4. Nếu **M** là số phần tử (entry) lớn nhất trong 1 node, thì **m≤M/2** là số phần tử nhỏ nhất có trong 1 node
      5. Mỗi node chứa số node từ m đến M trừ node root
      6. Nút root có ít nhất 2 con, nếu ko nó là nút lá
      7. Tất cả các nút lá đều có cùng 1 bậc
      8. Với mỗi index record(I, tuple-identifier) ở mỗi nút lá, I là hình chữ nhật nhỏ nhất bao phủ đối tượng dữ liệu n chiều đó
      9. Mỗi entry(I, child-pointer) trong nút ko phải lá, I là hình chữ nhật nhỏ nhất mà bao phủ hình chữ nhật chứa các node con.
   2. ***Searching***:
      1. EI: phần hình chữ nhật của 1 index entry
      2. EP: the tuple-identifier or child-pointer part
      3. Algorithm Search: cho R-Tree với T là gốc, S là hình chữ nhật cần tìm kiếm.
         1. T ko là nút lá: kiểm tra các phần tử E để xác định EI có overlap với S ko. Với mỗi phần tử overlapping, thực hiện tìm kiếm với root node là EP.
         2. Nếu T là lá, kiểm tra các phần tử E để xác định EI có overlap với S ko. Nếu có thì E là record cần tìm.
   3. ***Insertion***: insert a new index entry E
      1. Chọn node lá L sẽ là node để thêm E vào (chọn hình chữ nhật nhỏ nhất có thể chứa phần tử E này)
      2. Nếu L có đủ chổ cho 1 entry mới, thêm E vào. Nếu ko thì ***SplitNode*** để L & LL chứa E & các entries trước đây của L
      3. Thực hiện ***AdjustTree*** (điều chỉnh cây từ node L lên root, cần split hay ko) trên L & cũng như truyền LL nếu có split node
      4. Nếu split lan truyền làm cả root cũng split thì tạo root mới chứa 2 node con của kết quả của split.
   4. ***Deletion***: delete node lá E
      1. Tìm node lá L chứa E, dừng lại nếu cây ko chứa node lá E
      2. Xóa E khỏi L
      3. Thực hiện duyệt ngược lên từ node lá L:
         1. Q là tập node loại bỏ; P là node cha của L; EN là phần tử trong P gắn với L; gán N=L;
         2. Nếu L có ít hơn m entries thì xóa EN trong P, đưa L vào Q
         3. Nếu L ko bị loại bỏ thì điều chỉnh các hình chữ nhật ENI bao tất cả phần tử trong N.
         4. Gán N=P, nếu N ko là root thì quay lại bước 2.
         5. Re-insert tất cả các phần tử trong Q vào cây. Lá vào lá, node ở level nào vô level đó trong cây.
      4. Nếu node gốc chỉ có 1 con sau khi duyệt cây, tạo root mới bằng cách xóa root cũ & lấy node con làm root mới
   5. ***Node Splitting***:
      1. ***Exhaustive Algorithm***:
         1. Try all possible combination & choose the best
         2. Optimal results but Bad running time
      2. ***Quadratic-cost Algorithm***:
         1. Áp dụng giải thuật ***PickSeeds*** để chọn 2 entry đầu tiên trong các nhóm. Gán mỗi entry vào 1 group.

Giải thuật ***PickSeed***

* + - * + Chọn các cặp entries E1, E2, xác định hình chữ nhật J bao phủ E1I & E2I.
        + Tính d = area(J) – area(E1I) – area(E1I)
        + Chọn cặp entries có d lớn nhất
      1. Nếu tất cả các entry đã đc gán thì dừng lại. Nếu 1 nhóm có ít phần tử thì phải gán phần tử vào nó để nó có nhiều hơn m phần tử
      2. Thực hiện giải thuật ***PickNext*** để chọn phần tử kế đế gán vào. Thêm nó vào group mà hình chữ nhật của group sẽ đủ lớn để bảo phủ phần tử mới này. Giải quyết vấn đề này bằng cách chọn group có area nhỏ nhất, group có ít phần tử hơn. Lặp lại bước 2

Giải thuật ***PickNext***:

* + - * + Mỗi entry E chưa trong nhóm, tính d1 là diện tích cần tăng lên để bao phủ Group1 & EI. Tính d2 cho Group2
        + Chọn bất kỳ entry với sự khác biệt lớn nhất giữa d1 & d2
    1. ***LinearPickSeed Algorithm***

1. **R\*-Tree**:
   1. ***Tính chất***:
      1. Là tối ứu của R-Tree
      2. R-Tree xem xét đến các tham số area => có nhiều tham số cần phải quan tâm (area, overlap, margin in different combinaions)
   2. ***Insertion***:
      1. Các node ở level trên (leaf - 1) sẽ giống như R-Tree
      2. Các node ở level (leaf-1), chọn cây con với overlap nhỏ nhất (giải quyết bằng cách chọn hình chữ nhật cần mở rộng ít nhất)
         1. Overlap(E, node)=sumOfArea(Eentry) for all entry in node
         2. Only marginally better
   3. ***Split***:
      1. Với mỗi chiều, sort giảm dần M+1 values, dung upder value để break ties.
      2. Nhóm thứ nhất chứa m-1+k phần tử, nhóm 2 chứa M+2-m-k phần tử. Với k trong [1, M-2m+2]
      3. Xác định area-value, margin-value & overlap-value cho mỗ split point.
   4. ***Forced reinserts***:
      1. Nếu split xảy ra ở level k (not level root) & là lần đầu tiên overflow xảy ra trong level này
      2. Nếu ko thì split như trước
      3. Điều này giảm overlap & đưa ra cấu trúc tốt hơn.
   5. ***Benefits***:
      1. Forced reinserts thay đổi entries giữa các node láng giềng & do đó tăng overlap
      2. Hiệu ứng lề, lưu trữ sử dụng đc cải thiện.
      3. Do cơ cấu lại nhiều hơn nên ít split xảy ra hơn
      4. Khi hình chữ nhật bên ngoài node đc thêm vào, hình chữ nhật chứa sẽ càng vuông hơn

**Object Oriented Data Modeling**

1. **Object**: thuộc tính:
   * 1. *State*: thuộc tính kiểu & trị
     2. *Behavior*: cách mà object tác động & phản ứng lại
     3. *Identity*: mỗi object có 1 unique identity ngay cả khi tất cả các thuộc tính của chúng giống nhau. 1 object duy trì id trong suốt thời gian sống của nó.
2. **Object class**: tập hợp các đối tượng có cùng 1 cấu trúc & hành vi
3. **Class diagram**: mô tả cấu trúc tĩnh của 1 mô hình dự liệu hướng đối tượng: object classes, internal structures & relationships.
4. **Object diagram**: mô tả các thể hiện tương ứng với 1 diagram cho trước
5. **Operations**: là 1 hàm/dịch vụ mà sẽ đc cung cấp bởi tất cả các thể hiện của 1 class. Operations đc hiện thực thành các behavior.

Các kiểu của operations:

* + 1. *Constructor*: tạo ra 1 thể hiện mới của 1 class
    2. *Query*: lấy giá trị trạng thái (state) of object mà ko thay đổi nó
    3. *Update*: thay đổi trị của 1 trạng thái của object

1. **Associations**: là quan hệ giửa các object classes.
   1. ***Vai trò của associations***:
      1. Vai trò của 1 object trong association
      2. Kết thúc của 1 association là nơi mà nó kết hợp với 1 class
   2. ***Multiplicity*** (***lượng số***) số lượng các objects tham gia vào 1 association. Trong 1 class diagram, multiplicity cụ thể là:

Lowerbound … Upperbound

Vd: 0…\* 0…1 1…\* 1 \*

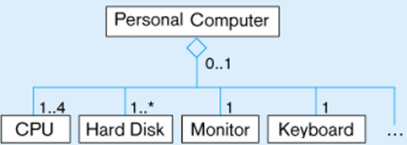
2…6 1,3,5,7

* 1. Một binary association là 2 hướng. Tên của mỗi hướng ảnh hưởng lên 1 chiều
  2. ***Biểu diễn lớp kết hợp***: 1 association có các attributes hay operations of riêng nó hay tham gia vào quan hệ với các class ≠

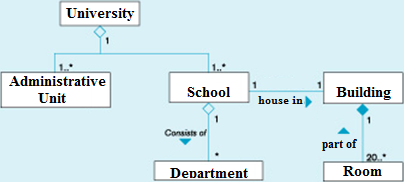
*Tên của class kết hợp*:

* + 1. Khi 1 kết hợp chỉ có các attribute nhưng ko có các operations hay ko tham gia vào các kết hợp khác => đặt tên trên đường kết hợp (association path) để nhất mạnh nó là “***association nature***”
    2. Khi 1 kết hợp có các operations của riêng nó => nên đặt tên của nó trong hình chữ nhật class để nhấn mạnh nó là “***class nature***”

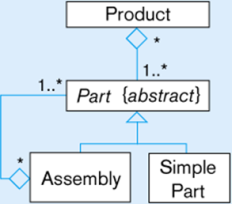
1. **Derived Attributes, Derived Associations & Derived Roles**: chúng là một & có thể nhận đc từ các attributes, associations, roles tương ứng. Phần tử kế thừa có dấu **/** phía trước tên của phần tử đó
2. ***Generalization/Specialization***:
   * 1. Subclass, superclass
     2. Common attributes, relationships & operations
     3. ***Disjoint*** >< ***Overlapping***:
        1. ***Disjoint***: nếu 1 instance of superclass là thành viên của 1 kiểu con & ko thể đồng thời là thành viên của kiểu con nào khác
        2. ***Overlapping***: nếu 1 instance of superclass có thể là thành viên của nhiều hơn 1 kiểu con.
        3. ***Thuộc tính phân biệt kiểu con***: là một thuộc tính của siêu kiểu mà trị của nó có thể xác định kiểu con
     4. ***Complete*** (total specialization) >< ***Incomplete*** (partial specialization)
        1. ***Complete***: superclass chia thành n class con, 1 instance của superclass phải là 1 trong n class con này.
        2. ***Incomplete***: superclass chia thành n class con, 1 instance của superclass có thể ko là 1 trong n class con này.
     5. ***Dynamic***: superclass chia thành n class con, 1 instance có thể thay đổi subclass theo thời gian
     6. ***Abstract class***: no direct instances
     7. ***Concrete class***: direct instances
3. **Class-level attribute**:
   * 1. Xác định 1 giá trị chung cho class, hơn là xác định 1 giá trị cụ thể cho 1 instance.
     2. Biểu diễn bằng underlining
     3. “=” là giá trị khởi tạo, giá trị mặc định
4. **Polymorphism**:
   * 1. ***Abstract Operation***: định nghĩa form hay protocol của operation, nhưng ko hiện thực nó
     2. ***Method***: hiện thực của 1 operation
     3. ***Polymorphism***: cùng operation có thể áp dụng vào nhiều class bằng nhiều cách khác nhau.
5. **Overriding Inheritance**: quá trình hiện thực các phương thức của superclass thành các hàm cụ thể của các class con
   * 1. For Extension: add code
     2. For Restriction: giới hạn phương thức
     3. For Optimization: cải thiện mã do khai thác hạn chế áp đặt bởi các lớp con
6. **Multiple Inheritance**:
   * 1. ***Multiple Classification***: 1 object là 1 thể hiện of n` hơn 1 class
     2. ***Multiple Inheritance***: 1 class kế thừa các thuộc tính từ nhiều hơn 1 supperclass
7. **Aggregation** (mối liên kết thu gộp)
   * 1. ***Aggregation***: object thu gộp quan hệ với nhiều component objects



* + 1. ***Composition***: 1 dạng mạnh của thu gộp. Object là 1 phần của object tổng thể, nó tồn tại phụ thuộc vào object tổng thể tồn tại. Tương ứng với thực thể yếu & thực thể mạnh



* + 1. ***Recursive Aggregation***: object tổng thể đc tổng hợp từ các object tổng thể khác hay object đơn vị (object nhỏ nhất)



1. **Class Diagram => RD Schema**:
   1. Chuyển 1 class trong class diagram thành table
   2. Chuyển 1 attribute của 1 class thành table column
   3. Trong trường hợp ko có khai báo id tường minh, tạo khóa chính (thông thường kiểu integer) cho table tương ứng với class
   4. Mỗi table tương ứng với 1 subclass, dung khóa chính của superclass của nó như khóa chính của nó.
   5. Mỗi table tương ứng với associate class, tạo khóa chính của nó. Sau đó thêm khóa chính của các table tương ứng với các class tham gia kết hợp vào làm khóa ngoại
   6. Với 1 component class trong liên kết thu gộp, table của nó lấy khóa chính của table tương ứng với class thu gộp làm 1 phần của khóa chính.
   7. Chuyển kết hợp 2 ngôi hay 1 ngôi thành 1 lược đồ quan hệ mạnh giống với chuyển trong mô hình EER
   8. Chuyển các operation của 1 class thành các stored procedure hay function. Các procedure/function này chứa các lệnh SQL
   9. Ko có cách tốt nhất để biểu diễn đa kế thừa trong lược đồ quan hệ. Chúng ta phải tái cấu trúc lại class diagram để tránh trường hợp này.