# Bài tập Chương 2

## Bài tập 2.1

1. Các mảnh được tạo ra là:

* EMP1 = σ\_TITLE<"Programmer"(EMP)

| ENO | ENAME | TITLE |
| --- | --- | --- |
| E1 | J. Doe | Elect. Eng. |
| E3 | A. Lee | Mech. Eng. |
| E6 | L. Chu | Elect. Eng. |
| E7 | R. Davis | Mech. Eng. |

* EMP2 = σ\_TITLE>"Programmer"(EMP)

| ENO | ENAME | TITLE |
| --- | --- | --- |
| E2 | M. Smith | Syst. Anal. |
| E5 | B. Casey | Syst. Anal. |
| E8 | J. Jones | Syst. Anal. |

1. Phân mảnh này không đúng vì vi phạm quy tắc **tính đầy đủ (completeness)**.
2. Các vị từ được sửa đổi là: p1\_new: TITLE < "Programmer", p2\_new: TITLE > "Programmer", p3\_new: TITLE = "Programmer". Các mảnh mới bao gồm EMP1\_new và EMP2\_new (giống câu a) và:

* EMP3\_new = σ\_p3\_new(EMP)

| ENO | ENAME | TITLE |
| --- | --- | --- |
| E4 | J. Miller | Programmer |

Phân mảnh này thỏa mãn cả ba quy tắc đúng đắn (completeness, reconstruction, disjointness).

## Bài tập 2.2

Các mảnh ngang không rỗng của quan hệ ASG dựa trên yêu cầu của hai ứng dụng là:

* ASG1 (RESP = "Manager" AND DUR < 20): { (E1, P1, Manager, 12) }
* ASG2 (RESP = "Manager" AND DUR ≥ 20): { (E5, P2, Manager, 24), (E6, P4, Manager, 48), (E8, P3, Manager, 40) }
* ASG3 (RESP = "Consultant" AND DUR < 20): { (E3, P3, Consultant, 10) }
* ASG4 (RESP = "Engineer" AND DUR ≥ 20): { (E3, P4, Engineer, 48), (E7, P3, Engineer, 36) }
* ASG5 (RESP = "Programmer" AND DUR < 20): { (E4, P2, Programmer, 18) }
* ASG6 (RESP = "Analyst" AND DUR < 20): { (E2, P2, Analyst, 6) }
* ASG7 (RESP = "Analyst" AND DUR ≥ 20): { (E2, P1, Analyst, 24) }

## Bài tập 2.3

Đồ thị kết nối ban đầu là một **đồ thị phân hoạch (partitioned)**. Để sửa đổi thành đồ thị đơn giản, ta phân mảnh EMP dẫn xuất theo PAY:

* EMP\_new1 = EMP ⋉ PAY1

| ENO | ENAME | TITLE |
| --- | --- | --- |
| E1 | J. Doe | Elect. Eng. |
| E2 | M. Smith | Syst. Anal. |
| E5 | B. Casey | Syst. Anal. |
| E6 | L. Chu | Elect. Eng. |
| E8 | J. Jones | Syst. Anal. |

* EMP\_new2 = EMP ⋉ PAY2

| ENO | ENAME | TITLE |
| --- | --- | --- |
| E3 | A. Lee | Mech. Eng. |
| E4 | J. Miller | Programmer |
| E7 | R. Davis | Mech. Eng. |

Đồ thị kết nối mới (PAY1 → EMP\_new1, PAY2 → EMP\_new2) là **đồ thị đơn giản (simple)**.

## Bài tập 2.4

Một ví dụ ma trận CA có điểm phân chia ở giữa:

|  | A1 | A3 | A4 | A2 | A5 |
| --- | --- | --- | --- | --- | --- |
| **A1** | - | 10 | 8 | 5 | 4 |
| **A3** | 10 | - | 100 | 90 | 8 |
| **A4** | 8 | 100 | - | 95 | 9 |
| **A2** | 5 | 90 | 95 | - | 6 |
| **A5** | 4 | 8 | 9 | 6 | - |

Trong trường hợp này, cụm {A3, A4, A2} có độ kề cao nằm ở giữa. Để tìm điểm phân chia tối ưu, thuật toán SPLIT cần thực hiện các thao tác SHIFT. Số thao tác SHIFT cần thiết để đảm bảo tìm được điểm tối ưu có thể lên tới n-1 lần, với n là số thuộc tính.

## Bài tập 2.5

Các mảnh của PAY là:

* PAY1 = σ\_SAL≥30000(PAY): {("Elect. Eng.", 40000), ("Syst. Anal.", 34000)}
* PAY2 = σ\_SAL<30000(PAY): {("Mech. Eng.", 27000), ("Programmer", 24000)}

Các mảnh dẫn xuất của EMP là:

* EMP1 = EMP ⋉ PAY1: Bao gồm các nhân viên {E1, E2, E5, E6, E8}.
* EMP2 = EMP ⋉ PAY2: Bao gồm các nhân viên {E3, E4, E7}.

Phân mảnh của EMP thỏa mãn 3 quy tắc đúng đắn:

* **Completeness:** Đúng.
* **Reconstruction:** Đúng.
* **Disjointness:** Đúng.

## Bài tập 2.6

Ma trận kề thuộc tính (AA) được tính từ dữ liệu đã cho:

|  | A1 | A2 | A3 | A4 | A5 |
| --- | --- | --- | --- | --- | --- |
| **A1** | - | 30 | 70 | 40 | 55 |
| **A2** | 30 | - | 60 | 0 | 45 |
| **A3** | 70 | 60 | - | 40 | 85 |
| **A4** | 40 | 0 | 40 | - | 40 |
| **A5** | 55 | 45 | 85 | 40 | - |

Sau khi áp dụng BEA và SPLIT, kết quả phân mảnh dọc (với A1 là khóa) là:

* Fragment 1 = {A1, A2, A4}
* Fragment 2 = {A1, A3, A5}

## Bài tập 2.7

Algorithm: DerivedHorizontalFragmentation  
Input:   
 R: target relation.  
 F\_S = {S1, S2, ..., Sw}: set of horizontal fragments of source relation S.  
 join\_predicate: semijoin predicate between R and S.  
Output:   
 F\_R = {R1, R2, ..., Rw}: set of derived horizontal fragments of R.  
  
begin  
 F\_R ← ∅;  
 for each Si in F\_S do  
 Ri ← R ⋉\_join\_predicate Si;  
 F\_R ← F\_R ∪ {Ri};  
 end for;  
 return F\_R;  
end

## Bài tập 2.8

Ma trận use(qi, Aj) cho các thuộc tính của EMP và ASG:

|  | ENO | ENAME | PNO | RESP | DUR |
| --- | --- | --- | --- | --- | --- |
| **q1** | 1 | 1 | 1 | 1 | 1 |
| **q2** | 1 | 0 | 0 | 0 | 1 |

Ma trận kề (Affinity Matrix) cho tất cả thuộc tính:

|  | ENO | ENAME | PNO | RESP | DUR |
| --- | --- | --- | --- | --- | --- |
| **ENO** | - | 30 | 30 | 30 | 60 |
| **ENAME** | 30 | - | 30 | 30 | 30 |
| **PNO** | 30 | 30 | - | 30 | 30 |
| **RESP** | 30 | 30 | 30 | - | 30 |
| **DUR** | 60 | 30 | 30 | 30 | - |

Sau khi biến đổi bằng BEA, ma trận sẽ nhóm các thuộc tính có độ kề cao, có thể hình thành 2 cụm: {ENO, DUR} và {ENAME, PNO, RESP}.

## Bài tập 2.9

Định nghĩa 3 quy tắc đúng đắn cho phân mảnh ngang dẫn xuất:

1. **Completeness:** ∪Ri = R. Mọi tuple của quan hệ đích R phải nằm trong một mảnh nào đó.
2. **Reconstruction:** Quan hệ R có thể được tái tạo bằng phép hợp các mảnh.
3. **Disjointness:** Các mảnh Ri và Rj (với i ≠ j) không được có tuple chung.

## Bài tập 2.10

1. Kết quả của PHF sẽ là hai mảnh:

* R1 = σ\_(R.A=10 AND R.B=15)(R)
* R2 = σ\_¬(R.A=10 AND R.B=15)(R)

1. Không, thuật toán COM\_MIN sẽ không tạo ra một tập vị từ đầy đủ và tối thiểu vì chỉ có một ứng dụng duy nhất, do đó không có vị từ nào được coi là “liên quan” theo định nghĩa của thuật toán.

## Bài tập 2.11

Thuật toán BEA tạo ra cùng kết quả khi hoạt động trên hàng hoặc cột vì:

1. Ma trận kề thuộc tính (AA) là đối xứng.
2. Hàm bond(Ax, Ay) được sử dụng trong thuật toán cũng có tính đối xứng. Do đó, việc tối ưu hóa dựa trên các cột hay các hàng đều dẫn đến cùng một thứ tự tương đối của các thuộc tính.

## Bài tập 2.12

Thuật toán SPLIT để phân mảnh n-chiều có thể được cài đặt đệ quy:

Algorithm: N\_WAY\_SPLIT\_Recursive(CA, R, n\_way)  
begin  
 if n\_way = 1 then return {R};  
   
 // Tìm điểm phân chia 2 chiều tốt nhất  
 best\_split\_point ← find\_best\_2\_way\_split(CA, R);  
   
 // Chia quan hệ và ma trận  
 R\_left, R\_right ← split(R, best\_split\_point);  
 CA\_left, CA\_right ← split(CA, best\_split\_point);  
   
 // Phân chia số mảnh và gọi đệ quy  
 k ← floor(n\_way / 2);  
 F\_left ← N\_WAY\_SPLIT\_Recursive(CA\_left, R\_left, k);  
 F\_right ← N\_WAY\_SPLIT\_Recursive(CA\_right, R\_right, n\_way - k);  
   
 return F\_left ∪ F\_right;  
end

Độ phức tạp của thuật toán này là O(n\_way \* n^2), với n là số thuộc tính.

## Bài tập 2.13

Định nghĩa 3 quy tắc đúng đắn cho phân mảnh hỗn hợp (hybrid):

1. **Completeness:** Mỗi thuộc tính của mỗi tuple trong quan hệ gốc phải tồn tại trong ít nhất một mảnh lá.
2. **Reconstruction:** Quan hệ gốc có thể được tái tạo lại bằng cách áp dụng các phép UNION và JOIN theo thứ tự ngược với quá trình phân mảnh.
3. **Disjointness:** Các thuộc tính không phải khóa của một tuple bất kỳ chỉ được lưu trữ trong một mảnh lá duy nhất.

## Bài tập 2.14

Có, thứ tự áp dụng các loại phân mảnh (ngang và dọc) ảnh hưởng đến tập các mảnh cuối cùng. Phân mảnh ngang trước rồi dọc sau (H-V) tạo ra các mảnh khác với phân mảnh dọc trước rồi ngang sau (V-H), vì các quyết định phân mảnh ở bước sau phụ thuộc vào kết quả của bước trước.

## Bài tập 2.15

Trong mô hình bài toán phân bổ CSDL: a. Mối quan hệ giữa các mảnh: Được mô hình hóa gián tiếp qua chi phí xử lý các truy vấn có phép nối giữa các mảnh. b. Xử lý truy vấn: Được mô hình hóa qua các thành phần chi phí: AC (truy cập), IE (toàn vẹn), CC (tương tranh) và TC (truyền thông). c. Thực thi toàn vẹn: Được mô hình hóa như một phần của chi phí xử lý (IEi). d. Điều khiển tương tranh: Được mô hình hóa như một phần của chi phí xử lý (CCi).

## Bài tập 2.16

1. Các tiêu chí so sánh heuristic: Chất lượng lời giải (độ gần tối ưu), độ phức tạp tính toán, tính tổng quát, và tính ổn định.
2. So sánh: Các thuật toán như network flow hay branch-and-bound cho chất lượng lời giải tốt hơn nhưng chậm hơn, trong khi các thuật toán greedy hay knapsack thì nhanh hơn nhưng chất lượng lời giải thường thấp hơn.

## Bài tập 2.17

Đây là bài tập lập trình. Một thuật toán heuristic tham lam (greedy) có thể được cài đặt với hai bước:

1. Phân bổ không sao chép ban đầu: Gán mỗi mảnh vào site có tổng chi phí truy cập thấp nhất.
2. Thêm bản sao lặp đi lặp lại: Trong mỗi vòng lặp, tìm vị trí đặt bản sao mới mang lại lợi ích ròng (giảm chi phí truy vấn trừ đi tăng chi phí lưu trữ/cập nhật) lớn nhất và thực hiện nó. Dừng lại khi không còn vị trí nào mang lại lợi ích.

## Bài tập 2.18

Phân mảnh và phân bổ tối ưu cho EMP và ASG:

* Phân mảnh:
  + ASG1 = σ\_DUR=24(ASG)
  + ASG2 = σ\_DUR≠24(ASG)
  + EMP1 = EMP ⋉ ASG1
  + EMP2 = EMP ⋉ ASG2
* Phân bổ:
  + ASG1 và EMP1: Đặt tại **Site 2**, sao chép sang **Site 1**.
  + ASG2: Đặt tại **Site 2**, sao chép sang **Site 3**. Đây là một phương án cân bằng giữa việc cục bộ hóa truy vấn và chi phí cập nhật bản sao.