Chương 2. SỰ TRONG SUỐT PHÂN TÁN

Mục tiêu

Chương này giới thiệu về các vấn đề sau:

1. Khái niệm về phân tán dữ liệu, các kiểu phân mảnh, quy tắc phân mảnh.
2. Sự trong suốt phân tán trong việc truy xuất, cập nhật dữ liệu

2.1 Các kiểu phân đoạn dữ liệu

Trong cơ sở dữ liệu phân tán, chúng ta đã biết các quan hệ trong lược đồ cơ sở dữ liệu thường được phân ra thành các mảnh nhỏ hơn nhưng chưa đưa ra lý do hoặc chi tiết nào về quá trình này. Phần này đề cập đến các chi tiết đó.

Các câu hỏi sau đây sẽ bao quát toàn bộ vấn đề:

- Tại sao cần phải phân mảnh?
- Làm thế nào để thực hiện phân mảnh?
- Phân mảnh nên thực hiện đến mức độ nào?
- Có cách gì kiểm tra tính đúng đắn của phân mảnh hay không?
- Chúng ta sẽ cấp phát như thế nào?

Những thông tin nào cần thiết cho việc cấp phát?

2.1.1 Các lý do phân mảnh

Đối với phân mảnh, điều quan trọng là có được một đơn vị phân mảnh thích hợp. Một quan hệ không phải là một đơn vị đáp ứng được yêu cầu đó.

- Trước tiên, khung nhìn của các ứng dụng thường chỉ là tập con của quan hệ. Vì thế đơn vị truy xuất không phải là toàn bộ quan hệ mà chỉ là các tập con của quan hệ. Kết quả là *xem tập con của các quan hệ là đơn vị phân tán sẽ là điều thích hợp nhất*.
- Thứ hai là nếu các ứng dụng có các khung nhìn được định nghĩa trên một quan hệ cho trước lại nằm tại những vị trí khác nhau thì có hai cách chọn lựa với đơn vị phân tán là : hoặc *quan hệ được lưu ở một vị trí* hoặc *quan hệ được nhân bản cho tất cả hay một số vị trí có chạy ứng dụng*. Chọn lựa đầu gây ra một số lượng lớn các truy xuất đến dữ liệu ở xa. Còn chọn lựa sau thực hiện nhân bản không cần thiết, gây ra nhiều vấn đề khi cập nhật và có thể gây ra lãng phí không gian lưu trữ. Vì thế, việc phân rã một quan hệ thành nhiều mảnh, mỗi mảnh được xử lý như một đơn vị, sẽ cho phép thực hiện nhiều giao tác đồng thời. Ngoài ra, việc phân mảnh các quan hệ sẽ cho phép thực hiện song song một câu vấn tin bằng cách chia nó thành một tập các câu vấn tin con hoạt tác trên từng mảnh. Do đó việc phân

mảnh sẽ làm tăng mức độ hoạt động song hành và như thế làm tăng lưu lượng hoạt động của hệ thống.

- Tuy nhiên cũng cần chỉ rõ những khiếm khuyết của việc phân mảnh:

- Nếu ứng dụng cần phải truy xuất dữ liệu từ hai mảnh rồi nối hoặc hợp chúng lại thì chi phí rất cao.

- Vấn đề thứ hai liên quan đến tính toàn vẹn dữ liệu: do kết quả của việc phân mảnh, các thuộc tính tham gia vào một phụ thuộc hàm có thể bị phân rã vào các mảnh khác nhau và được cấp phát cho những vị trí khác nhau. Trong trường hợp này việc kiểm tra các phụ thuộc hàm cũng phải thực hiện truy tìm dữ liệu ở nhiều vị trí.

2.1.2 Các kiểu phân mảnh

Có ba kiểu phân mảnh khác nhau là phân mảnh theo chiều dọc, phân mảnh theo chiều ngang và phân mảnh hỗn hợp sẽ được trình bày chi tiết ở phần sau.

2.1.3 Mức độ phân mảnh

Phân mảnh cơ sở dữ liệu đến mức độ nào là một quyết định rất quan trọng, nó có ảnh hưởng đến hiệu suất vấn tin. Mức độ phân mảnh có thể đi từ thái cực không phân mảnh đến phân mảnh thành từng bộ (trong trường hợp phân mảnh ngang) hoặc từng thuộc tính (trường hợp phân mảnh dọc).

Các đơn vị phân mảnh quá lớn hoặc quá nhỏ cũng gây ảnh hưởng xấu đến hệ thống. Vì thế điều cần tìm là mức độ phân mảnh thích hợp. Các phương pháp phân mảnh sẽ được đề cập đến nó trong phần 2.1.6.

2.1.4 Các quy tắc phân mảnh

Trong quá trình phân mảnh chúng ta phải *tuân thủ ba qui tắc để bảo đảm cơ sở dữ liệu sẽ không thay đổi về mặt ngữ nghĩa*.

1. **Tính đầy đủ** (completeness): Nếu một quan hệ R được phân rã thành các mảnh $R_1, R_2, \ldots, R_n$ thì mỗi mục dữ liệu trong R cũng có thể có trong một hay nhiều mảnh R_i. Đặc tính này nói lên sự phân mảnh mà không mất thông tin.

2. **Tính tái thiết** được (reconstruction): Nếu một quan hệ R được phân rã thành các mảnh $R_1, R_2, \ldots, R_n$ thì cần định nghĩa một phép toán quan hệ ∇ sao cho: $R = \nabla R_i \quad i \in [1,n]$

Phép toán ∇ thay đổi tùy theo loại phân mảnh. Khả năng tái thiết một quan hệ từ các mảnh của nó bảo đảm rằng các ràng buộc được định nghĩa trên dữ liệu dưới dạng phụ thuộc hàm sẽ được bảo toàn.

3. **Tính tách biệt** (disjointness): Nếu một quan hệ R được phân rã thành các mảnh $R_1, R_2, \ldots, R_n$ và mục dữ liệu d_i nằm trong mảnh R_j, thì nó sẽ không nằm trong một mảnh R_k khác ($k \neq j$). Tiêu chuẩn này đảm bảo các mảnh ngang sẽ tách biệt. Nếu quan hệ được phân rã dọc, các thuộc tính khoá chính phải được lặp lại trong mỗi mảnh.

2.1.5 Các kiểu cấp phát :

Giả sử cơ sở dữ liệu được phân mảnh thích hợp và cần phải quyết định cấp phát các mảnh cho các vị trí trên mạng. Khi dữ liệu được cấp phát, nó có thể được nhân bản hoặc chỉ giữ một bản duy nhất. Lý do cần phải nhân bản nhằm đảm bảo độ tin cậy

và hiệu quả cho các câu truy vấn chỉ đọc. *Nếu có nhiều bản sao thì chúng ta vẫn có cơ hội truy xuất được dữ liệu đó ngay khi hệ thống xảy ra sự cố.* Hơn nữa các câu truy vấn chỉ đọc truy xuất đến cùng một mục dữ liệu có thể cho thực hiện song song bởi vì các bản sao có mặt tại nhiều vị trí. *Ngược lại câu vấn tin cập nhật có thể gây ra nhiều rắc rối bởi vì hệ thống phải đảm bảo rằng tất cả các bản sao phải được cập nhật chính xác*. Vì vậy quyết định nhân bản cần phải cân nhắc và phụ thuộc vào tỷ lệ giữa các câu vấn tin chỉ đọc và các câu vấn tin cập nhật. Quyết định này hầu như có ảnh hưởng đến tất cả các thuật toán của hệ quản trị cơ sở dữ liệu phân tán và các chức năng kiểm soát khác

	Nhân bản hoàn toàn	**Nhân bản 1 phần**	**Phân hoạch**
Xử lý vấn tin	Dễ	Khó ⟷	
Quản lý từ điển dữ liệu	Dễ	Khó ⟷	
Điều khiển đồng thời	Khó	Vừa phải	Dễ
Độ khả tín	Rất cao	Cao	Thấp
Tính thực tế	Thực tế	Thực tế	Thực tế

Hình 2.1 So sánh các chọn lựa nhân bản

Một cơ sở dữ liệu không nhân bản (thường gọi là cơ sở dữ liệu phân hoạch) có chứa các mảnh được cấp phát cho các vị trí, trong đó chỉ tồn tại một bản sao duy nhất cho mỗi mảnh trên mạng. Trong trường hợp nhân bản, hoặc toàn bộ cơ sở dữ liệu đều tồn tại ở mọi vị trí (cơ sở dữ liệu nhân bản hoàn toàn) hoặc các mảnh được phân tán đến các vị trí, trong đó một mảnh có thể có nhiều bản sao tại nhiều vị trí (cơ sở dữ liệu nhân bản một phần). Hình 2.1 so sánh ba lựa chọn nhân bản.

2.1.6 Các phương pháp phân mảnh:

2.1.6.1 Phân mảnh ngang

Phân mảnh ngang chia một quan hệ theo các bộ. Vì vậy mỗi mảnh là một tập con của quan hệ. Có hai loại phân mảnh ngang: phân mảnh ngang nguyên thủy và phân mảnh ngang dẫn xuất.

- Phân mảnh ngang nguyên thủy (Primary Horizontal Fragmentation) là sự phân mảnh một quan hệ dựa trên một vị từ được định nghĩa trên một quan hệ.

- Phân mảnh ngang dẫn xuất (Derived Horizontal Fragmentation) là phân rã một quan hệ dựa vào các vị từ được định nghĩa trên một quan hệ khác.

Trước khi trình bày thuật toán hình thức cho phân mảnh ngang, chúng ta sẽ thảo luận một cách trực quan về quá trình phân mảnh.

Cho quan hệ R, các mảnh ngang R_i là :

$$R_i = \sigma_{Fi}(R)$$

Trong đó F_i là công thức chọn để có được mảnh R_i.

Ví dụ : Cho lược đồ quan hệ toàn cục :

SUPPLIER (SNUM, NAME, CITY)

Chúng ta có thể có hai phân mảnh ngang sau:

$$SUPPLIER_1 = \sigma_{CITY= \text{“SF”}}(SUPPLIER)$$

$$SUPPLIER_2 = \sigma_{CITY= \text{“LA”}}(SUPPLIER)$$

- Sự phân mảnh trên thỏa điều kiện đầy đủ nếu “SF” và “LA” chỉ là các giá trị có thể có của thuộc tính CITY; ngược lại chúng ta sẽ không biết những mảnh nào với các giá trị CITY khác.

- Điều kiện tái thiết được kiểm tra dễ dàng vì chúng ta luôn luôn có thể tái thiết lại quan hệ toàn cục SUPPLIER bằng phép toán hội:

$$SUPPLIER = SUPPLIER_1 \ U \ SUPPLIER_2$$

- Điều kiện tách biệt cũng được kiểm tra một cách rõ ràng.

2.1.6.2 Phân mảnh ngang dẫn xuất

Trong một số trường hợp sự phân mảnh ngang được dẫn ra từ một phân mảnh ngang của một quan hệ khác.

Ví dụ: Một quan hệ toàn cục

SUPPLY (SNUM, PNUM, DEPTNUM, QUAN)

Với SNUM là mã số người cung cấp. Chúng ta muốn phân chia quan hệ này sao cho một mảnh chứa các bộ cho những người cung cấp ở một thành phố cho trước. Tuy nhiên thành phố không phải là thuộc tính của quan hệ SUPPLY mà là thuộc tính của quan hệ SUPPLIER. Vì thế chúng ta cần thực hiện phép nữa kết để xác định các bộ của SUPPLY tướng ứng với những người cung cấp

trong một thành phố cho trước. Sự phân mảnh dẫn xuất của SUPPLY có thể được định nghĩa như sau:

$SUPPLY_1 =$ SUPPLY SJ $SUPPLIER_1$

$SUPPLY_2 =$ SUPPLY SJ $SUPPLIER_2$

Với SJ là phép toán nửa kết (Semi Join)

- Tính tái thiết quan hệ toàn cục SUPPLY có thể được thể hiện qua phép hội.

- Rõ ràng với phép kết như vậy thì một bộ trong quan hệ SUPPLY chỉ có thể thuộc 1 trong 2 mảnh SUPPLY1 hoặc SUPPLY2 , do đó, nó thỏa tính đầy đủ và tách biệt .

2.1.6.3 Phân mảnh dọc

Sự phân mảnh dọc của một quan hệ toàn cục là việc chia các thuộc tính vào nhiều nhóm; các mảnh nhận được từ phép chiếu quan hệ toàn cục trên mỗi nhóm. Sự phân mảnh này sẽ đúng đắn nếu mỗi thuộc tính được ánh xạ vào ít nhất vào một thuộc tính của các phân mảnh; hơn nữa, nó phải có khả năng tái thiết lại quan hệ nguyên thủy bằng cách kết nối các phân mảnh lại với nhau. *Để có khả năng tái thiết lại quan hệ nguyên thủy thì mỗi phân mảnh dọc phải chứa khóa chính của quan hệ nguyên thủy đó*.

Ví dụ : Xét quan hệ toàn cục

EMP (EMPNUM, NAME, SAL, TAX, MNRNUM, DEPTNUM)

Một sự phân mảnh dọc của quan hệ này được định nghĩa:

$EMP_1 = \pi_{EMPNUM,\ NAME,\ MGRNUM,\ DETPNUM}\ (EMP)$

$EMP_2 = \pi_{EMPNUM,\ SAL,\ TAX}\ (EMP)$

Phân mảnh này phản ánh lương và thuế của các nhân viên được quản lý riêng. Việc tái thiết lại quan hệ EMP có thể nhận được từ :

$EMP = EMP_1$ JNN EMP_2

(với JNN là phép kết nối tự nhiên hai quan hệ)

Từ đó chúng ta thấy sự phân mảnh cũng thỏa tính đầy đủ và tính tách biệt.

2.1.6.4 Sự phân mảnh hỗn hợp

Các phân mảnh nhận được bởi các phép phân mảnh trên là các quan hệ, vì thế chúng ta có thể áp dụng các phép toán phân mảnh một cách đệ qui. Việc tái thiết quan hệ thực hiện được bằng cách áp dụng các luật tái thiết theo thứ tự ngược. Các biểu thức mà định nghĩa các phân mảnh trong trường hợp này sẽ phức tạp hơn.

Ví dụ: Xét quan hệ toàn cục:

EMP (EMPNUM, NAME, SAL, TAX, MNRNUM, DEPTNUM)

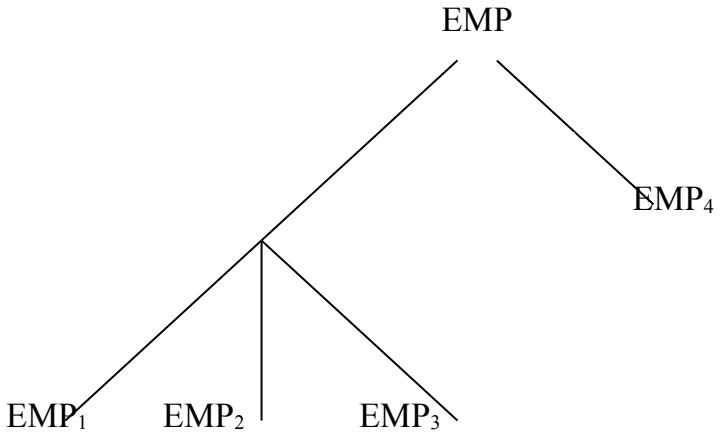
Dưới đây là một sự phân mảnh hỗn hợp bằng cách áp dụng sự phân mảnh dọc rồi sau đó áp dụng phân mảnh ngang trên DEPTNUM:

$EMP_1 = \sigma_{deptnum<=10}\ (\pi_{empnum,\ name,\ mgrnum,\ deptnum}\ (EMP))$

$EMP_2 = \sigma_{10<deptnum<=20}(\pi_{empnum,\ name,\ mgrnum,\ deptnum}\ (EMP))$

$EMP_3 = \sigma_{deptnum>20}\ (\pi_{empnum,\ name,\ mgrnum,\ deptnum}\ (EMP))$

$EMP_4 = \pi_{empnum,\ name,\ sal,\ tax}(EMP)$



Hình 2.2 Cây phân mảnh của quan hệ EMP

Việc tái thiết lại quan hệ EMP được định nghĩa bởi biểu thức:

$$EMP = U(EMP_1, EMP_2, EMP_3)\ JNN\ \pi_{empnum,\ sal,\ tax}(EMP_4)$$

Sự phân mảnh hỗn hợp có thể được biểu diễn bởi cây phân mảnh. Trong cây phân mảnh, gốc tương ứng với quan hệ toàn cục, các lá tương ứng với các phân mảnh và các nút trung gian tương ứng với các kết quả trung gian của các biểu thức phân mảnh. Tập các nút con của một nút thể hiện sự phân rã của nút này bởi một phép toán phân mảnh ngang hoặc dọc. Hình 2.2 minh họa cây phân mảnh của quan hệ EMP.

2.2 Sự trong suốt phân tán

2.2.1 Sự trong suốt phân tán của ứng dụng chỉ đọc:

Để hiểu được sự trong suốt phân tán của ứng dụng chỉ đọc, chúng ta sẽ xem xét các ví dụ được minh họa bằng ngôn ngữ tựa Pascal có nhúng ngôn ngữ SQL. Trong các ví dụ này chúng ta có một số chú ý sau:

- Các biến có kiểu chuỗi ký tự. (Declare @manv nvarchar(20))
- Nhập xuất được thực hiện qua các thủ tục chuẩn: Read(filename, variable)

(**Set** @bien = giatri hoặc **gởi giá trị cho tham số** của SP) , write(filename, variable) (**Select @bien**). Nếu nhập xuất được thực hiện tại một trạm thì filename sẽ là "terminal".

- Trong các phát biểu SQL, các biến của ngôn ngữ Pascal sẽ bắt đầu bằng ký tự '$'.

Ví dụ: Xét câu truy vấn : Tìm tên của người cung cấp khi biết mã số người cung cấp.

SET $NAME = (Select NAME From SUPPLIER Where SNUM = $SNUM)

SET $CITY = (Select CITY From SUPPLIER Where SNUM = $SNUM)

Select $NAME =NAME, $CITY =CITY From SUPPLIER Where SNUM = $SNUM

Trong câu truy vấn này $NAME là biến xuất còn $SNUM là biến nhập.

- Các biến của Pascal sử dụng để liên hệ với hệ quản trị cơ sở dữ liệu phân tán bắt đầu bằng ký tự '#'.

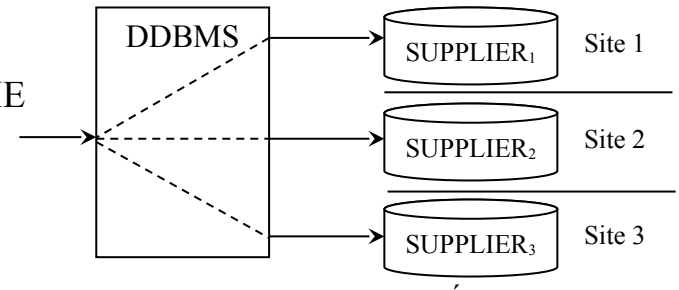
Ví dụ: Sau khi truy vấn một câu SQL, biến luận lý #FOUND = TRUE nếu kết quả trả về khác rỗng (@@ROWCOUNT>0). Biến #OK = TRUE (@@ERROR =0) nếu phép toán thực hiện đúng bởi hệ quản trị cơ sở dữ liệu.

Các mức độ trong suốt sẽ được xét từ cao đến thấp qua hai trường hợp sau.

a. Xét trên một đồ quan hệ phổ quát

Mức 1: Sự trong suốt phân tán

```
Read (terminal, $SNUM)
        Select NAME into $NAME
        From SUPPLIER
        Where SNUM = $SNUM
Write(terminal, $NAME)
```

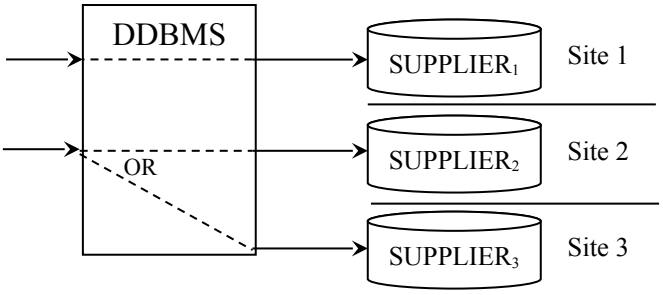


Hình 2.3a Sự trong suốt phân tán

Nhận xét: Theo hình 2.3a và đoạn lệnh ở trên, chúng ta thấy câu truy vấn trên tương tự như câu truy vấn cục bộ, không cần chỉ ra các phân mảnh cũng như các vị trí cấp phát cho các phân mảnh đó. Khi đó người sử dụng không hề có cảm giác là đang thao tác trên một câu truy vấn phân tán.

Mức 2: Sự trong suốt vị trí (location)

```
Read (terminal, $SNUM)
        Select NAME into $NAME
        From SUPPLIER1
        Where SNUM = $SNUM
If not #Found then
        Select NAME into $NAME
        From SUPPLIER2
        Where SNUM = $SNUM
Write(terminal, $NAME)
```



Hình 2.3b Sự trong suốt vị trí

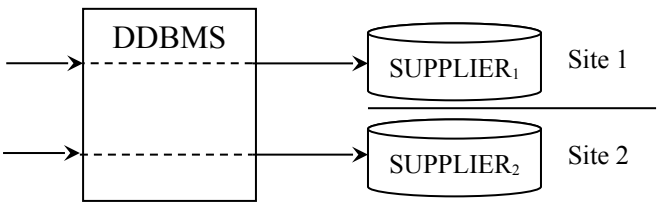
Nhận xét: Người sử dụng phải cung cấp tên các phân mảnh cụ thể cho câu truy vấn nhưng không cần chỉ ra vị trí của các phân mảnh.

Sự trong suốt vị trí ở hình 2.3b có thể được viết như sau:

```
Read(Terminal, $SNUM)
Read(Terminal, $CITY)
Case $CITY Of
  "SF": Select NAME into $NAME
        From SUPPLIER1
        Where SNUM = $SNUM;
  "LA": Select NAME into $NAME
        From SUPPLIER2
        Where SNUM = $SNUM
End;
Write(Terminal, $NAME)
```

Mức 3: Sự trong suốt ánh xạ cục bộ

```
Read (terminal, $SNUM)
        Select NAME into $NAME
        From SUPPLIER
        Where SNUM = $SNUM
If not #Found then
        Select NAME into $NAME
```



Hình 2.3c Sự trong suốt ánh xạ cục bộ

From SUPPLIER AT SITE 2

Where SNUM = $SNUM

Write(terminal, $NAME)

Nhận xét: tại mức trong suốt này người sử dụng phải cung cấp tên các phân mảnh và vị trí cấp phát của chúng.

Mức 4: Không trong suốt

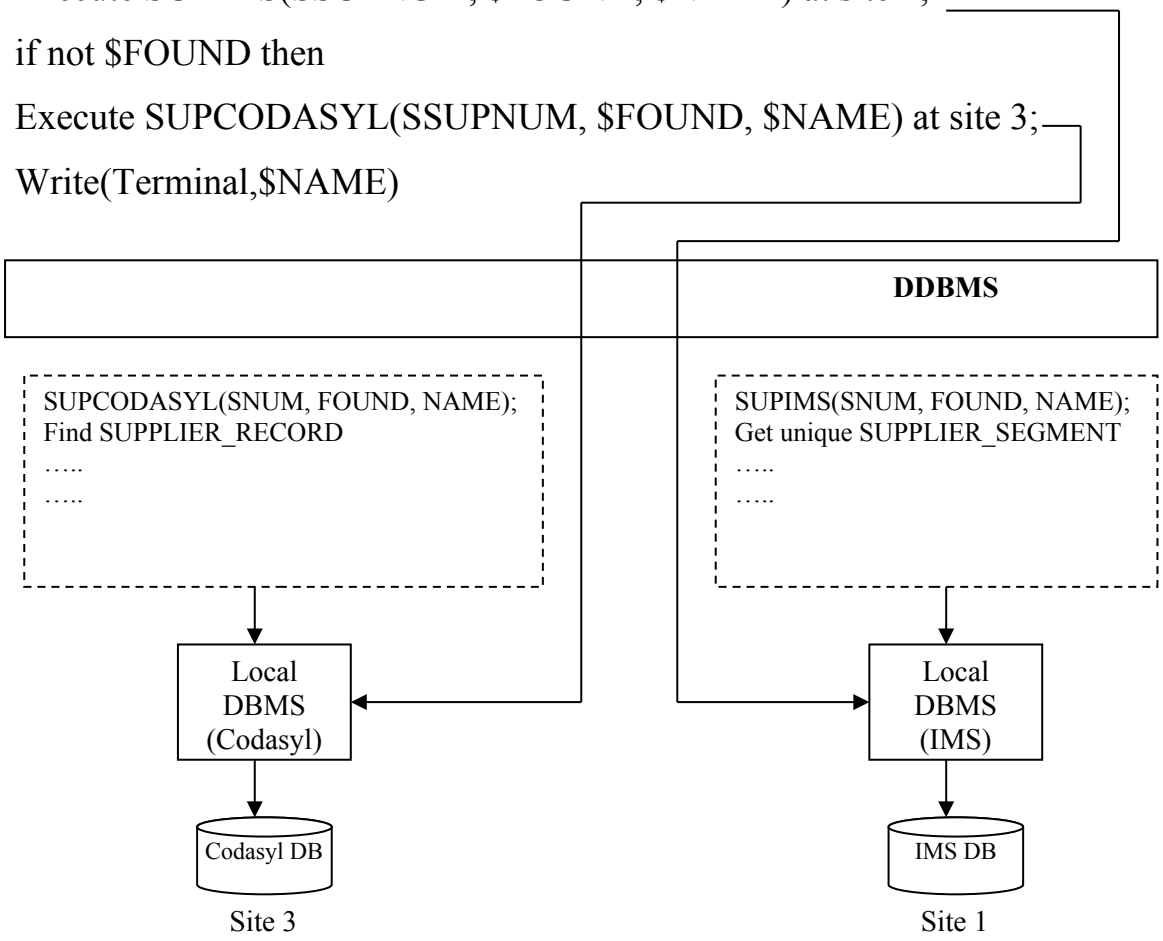
Read(Terminal,SSUPNUM)

Execute SUPIMS(SSUPNUM, $FOUND, $NAME) at site 1;

if not $FOUND then

Execute SUPCODASYL(SSUPNUM, $FOUND, $NAME) at site 3;

Write(Terminal,$NAME)



Hình 2.4 Một ứng dụng trên cơ sở dữ liệu phân tán không đồng nhất và không trong suốt

Nhận xét: Tại mức thấp nhất này chúng ta cần phải viết lệnh theo hệ quản trị cơ sở dữ liệu tương ứng.

b. Xét trên hai lược đồ quan hệ phổ quát

Chúng ta hãy xét một ví dụ phức tạp hơn. Giả sử cần tìm tên những nhà cung cấp mặt hàng có mã số (mặt hàng) cho trước.

Mức 1: Trong suốt phân tán

Read(Terminal, $PNUM)
Select @NAME =NAME

```
from SUPPLIER, SUPPLY
where SUPPLIER.SNUM = SUPPLY.SNUM and
        SUPPLY.PNUM = $PNUM
write(Terminal, $NAME)
```

Nhận xét:

Mức 2: Trong suốt vị trí

```
Read(Terminal, $PNUM)
Select NAME into $NAME
from SUPPLIER1, SUPPLY1
where SUPPLIER1.SNUM = SUPPLY1.SNUM and
        SUPPLY1.PNUM = $PNUM
if not #FOUND then
 Select NAME into $NAME
from SUPPLIER2, SUPPLY2
where SUPPLIER2.SNUM = SUPPLY2.SNUM and
        SUPPLY2.PNUM = $PNUM
write(Terminal, $NAME)
```

Nhận xét:

Mức 3: Trong suốt ánh xạ cục bộ

Giả sử các sơ đồ cấp phát các phân mảnh của quan hệ SUPPLY và SUPPLIER như sau:

$SUPPLIER_1$: Tại site 1

$SUPPLIER_2$: Tại site 2

$SUPPLY_1$: Tại site 3

$SUPPLY_2$: Tại site 4

```
Read(Terminal, $PNUM)
Select SNUM into $SNUM
from SUPPLY1 at site 3
where SUPPLY1.PNUM = $PNUM
if   #FOUND then
begin
send $SNUM from site 3 to site 1
 Select NAME into $NAME
from SUPPLIER1 at site 1
where SUPPLIER1.SNUM = $SNUM
end
else  begin
Select SNUM into $SNUM
from SUPPLY2 at site 4
where SUPPLY2.PNUM = $PNUM
send $SNUM from site 4 to site 2
 Select NAME into $NAME
from SUPPLIER2 at site 2
where SUPPLIER2.SNUM = $SNUM
```

end;
write(Terminal, $NAME)

Nhận xét:

2.2.2 Sự trong suốt phân tán đối với các ứng dụng cập nhật

Trong phần trước, chúng ta chỉ xét các ứng dụng tìm kiếm dữ liệu trong cơ sở dữ liệu phân tán . Phần này chúng ta sẽ xem xét các ứng dụng cập nhật dữ liệu trong cơ sở dữ liệu phân tán. Bài toán cập nhật ở đây chỉ xét dưới khía cạnh trong suốt phân tán đối với các lập trình viên, trong khi bài toán bảo đảm tính nguyên tử của các giao tác cập nhật sẽ được xem xét ở chương sau.

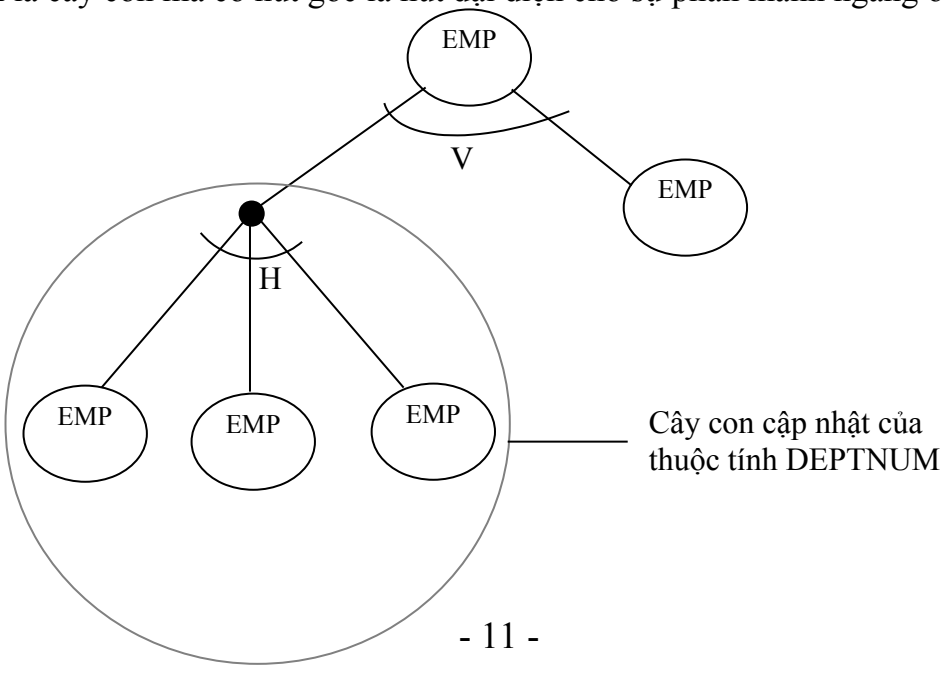
Các mức trong suốt phân tán cũng được phân tích như trong các ứng dụng chỉ đọc. Tuy nhiên một phép cập nhật phải được thực hiện trên tất cả các bản sao của một mục dữ liệu trong khi phép tìm kiếm chỉ cần thực hiện trên một bản sao. Điều này có nghĩa là nếu hệ quản trị cơ sở dữ liệu không hỗ trợ sự trong suốt vị trí và sự trong suốt nhân bản thì lập trình viên chịu trách nhiệm thực hiện mọi cập nhật được yêu cầu.

Trong việc hỗ trợ sự trong suốt phân tán cho các ứng dụng cập nhật có một vấn đề khá phức tạp hơn việc cập nhật tất cả các bản sao của một mục dữ liệu. Đó là vấn đề di chuyển dữ liệu sau khi cập nhật.

Xét ví dụ sau: Điều gì xảy ra khi cập nhật lại giá trị của thuộc tính CITY trong quan hệ SUPPLIER. Rõ ràng, các bộ Supplier phải được chuyển từ một phân mảnh này đến phân mảnh khác. Hơn nữa, như trong ví dụ ở phần 2.2.1, các bộ của quan hệ SUPPLY mà tham khảo đến cùng Supplier cũng phải thay đổi phân mảnh vì quan hệ SUPPLY có một phân mảnh dẫn xuất. Một cách trực quan chúng ta dễ dàng thấy việc thay đổi giá trị của một thuộc tính mà chúng ta định nghĩa trong lược đồ phân mảnh có dẫn đến các hệ quả phức tạp. Mức độ mà các hệ quản trị cơ sở dữ liệu phân tán quản lý các hệ quả đó chính là đặc trưng cho các mức trong suốt phân tán cho sự cập nhật.

Để minh họa cho các phép toán di chuyển dữ liệu trong việc cập nhật dữ liệu phân tán người ta sử dụng cây con cập nhật.

Xét một thuộc tính A được sử dụng trong vị từ phân mảnh ngang. Cây con cập nhật của A là cây con mà có nút gốc là nút đại diện cho sự phân mảnh ngang ở trên.



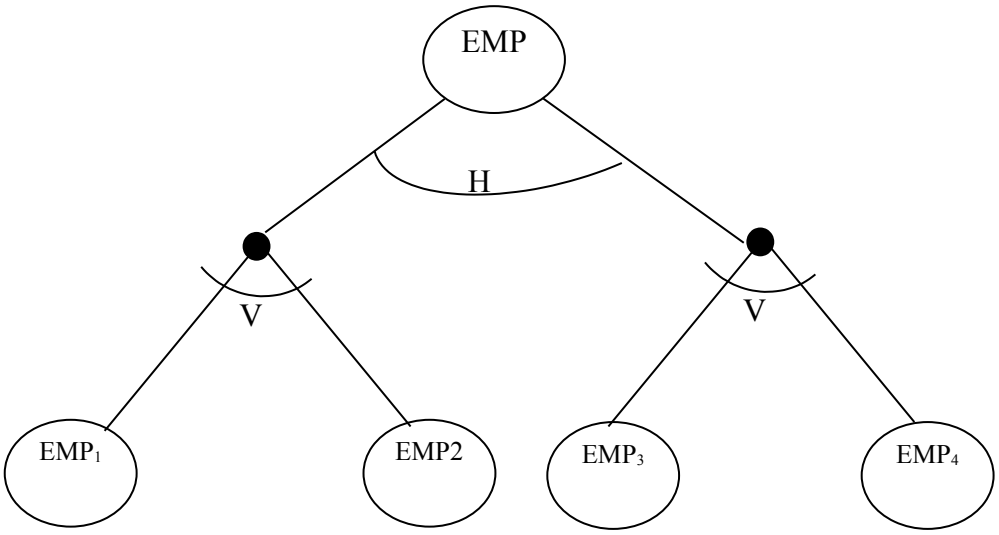
V: Phép toán phân mảnh dọc H: Sự phân mảnh ngang

Hình 2.5 Cây con cập nhật của thuộc tính DEPTNUM trong cây phân mảnh của quan hệ EMP

- Các ảnh hưởng của sự thay đổi giá trị của một thuộc tính chỉ giới hạn trong các phân mảnh ở nút lá của cây con cập nhật.

Ví dụ: Một sự thay đổi giá trị của thuộc tính DEPTNUM chỉ ảnh hưởng đến EMP_1, EMP_2 và EMP_3. Một bộ có thể di chuyển giữa hai trong ba phân mảnh trên.

Xét một ví dụ khác phức tạp hơn. Giả sử quan hệ EMP có cây phân mảnh như hình 2.6a.



$EMP_1 = PJ_{EMPNUM, NAME, SAL, TAX}SL_{DEPTNUM <10}(EMP)$

$EMP_2 = PJ_{EMPNUM, MGRNUM, DEPTNUM}SL_{DEPTNUM <10}(EMP)$

$EMP_3 = PJ_{EMPNUM, NAME, DEPTNUM}SL_{DEPTNUM >=10}(EMP)$

$EMP_4 = PJ_{EMPNUM, SAL, TAX, MGRNUM}SL_{DEPTNUM >=10}(EMP)$

Hình 2.6a Cây phân mảnh khác của quan hệ EMP

EMP_1

EMPNUM	NAME	SAL	TAX
100	SMITH	10000	1000

EMP_2

EMPNUM	MGRNUM	DEPTNUM
100	20	**3**

Trước khi cập nhật

Sau khi cập nhật

EMP_3

EMPNUM	NAME	DEPTNUM

EMP_4

EMPNU	SAL	TAX	MGRNUM

100	SMITH	**15**

M			
100	10000	1000	20

Hình 2.6b Hệ quả của việc cập nhật DEPTNUM của EMPNUM=100 từ 3 sang 15

Trong trường hợp này cây con cập nhật của thuộc tính DEPTNUM tương tự như cây phân mảnh. Sự ảnh hưởng của việc thay đổi giá trị của DEPTNUM của bộ có EMPNUM=100 từ 3 thành 15 được minh họa ở hình 2.6b. Sự cập nhật của bộ này chỉ thuộc về cây con trái, sau khi cập nhật nó trở thành một phần cây con phải. Chúng ta nhận thấy không chỉ dữ liệu được chuyển giữa các mảnh mà bộ này cũng được tổ hợp lại theo một cách khác.

Bây giờ chúng ta sẽ xem xét các mức trong suốt phân tán cho một ứng dụng cập nhật đơn giản.

Mức 1: Sự trong suốt phân tán

Mức này minh hoạ chương trình ứng dụng cập nhật dữ liệu như trong một cơ sở dữ liệu không phân tán. Bởi thế các lập trình viên không cần biết thuộc tính nào được dùng để phân mảnh. Để thay đổi giá trị DEPTNUM của employee có EMPNUM=100, đoạn chương trình được viết như sau:

```
Update EMP
        set DETPNUM=15
        where EMPNUM =100
```

Mức 2: Sự trong suốt vị trí

Tại mức này, lập trình viên phải làm việc với các phân mảnh một cách tường minh.

Đoạn chương trình được viết như sau:

```
Select NAME, SAL, TAX into $NAME, $SAL, $TAX
        from EMP1 where EMPNUM=100;
Select MGRNUM into $MGRNUM
        from EMP2 where EMPNUM=100;
Insert into EMP3 (EMPNUM, NAME, DEPTNUM)
        Values (100, $NAME, 15);
Insert into EMP4 (EMPNUM, SAL, TAX, MGRNUM)
        Values (100, $SAL, $TAX, $MGRNUM);
Delete EMP1 where EMPNUM = 100;
Delete EMP2 where EMPNUM = 100;
```

Mức 3: Sự trong suốt ánh xạ cục bộ

Tại mức này ứng dụng phải giải quyết vị trí của các phân mảnh một cách tường minh. Giả sử các phân mảnh của quan hệ EMP được cấp phát như sau:

EMP1 : site1 và 5
EMP2 : site2 và 6
EMP3 : site3 và 7
EMP4 : site4 và 8

Đoạn chương trình được viết như sau:

Select NAME, SAL, TAX into $NAME, $SAL, $TAX
from EMP_1 at site 1 where EMPNUM=100;
Select MGRNUM into $MGRNUM
from EMP_2 at site 2 where EMPNUM=100;
Insert into EMP_3 (EMPNUM, NAME, DEPTNUM) at site 3
Values (100, $NAME, 15);
Insert into EMP_3 (EMPNUM, NAME, DEPTNUM) at site 7
Values (100, $NAME, 15);
Insert into EMP_4 (EMPNUM, SAL, TAX, MGRNUM) at site 4
Values (100, $SAL, $TAX, $MGRNUM);
Insert into EMP_4 (EMPNUM, SAL, TAX, MGRNUM) at site 8
Values (100, $SAL, $TAX, $MGRNUM);
Delete EMP_1 at site 1 where EMPNUM = 100;
Delete EMP_1 at site 5 where EMPNUM = 100;
Delete EMP_2 at site 2 where EMPNUM = 100;
Delete EMP_2 at site 6 where EMPNUM = 100;

2.3 Các nguyên tắc truy xuất cơ sở dữ liệu phân tán

Trong các ví dụ ở phần trước chúng ta chỉ xét các truy vấn dữ liệu chỉ trả về một giá trị. Tuy nhiên trong trường hợp tổng quát, một câu truy vấn có thể trả về một quan hệ. Khi đó chúng ta sẽ qui ước dùng tham số có tiếp vị ngữ "REL", được xem như một file, để nhận kết quả trả về.

Ví dụ: Cho câu SQL sau:

Select EMPNUM, NAME INTO **$EMP_REL($EMPNUM, $NAME)**
from EMP

Bây giờ chúng ta sẽ xem xét các cách truy xuất cơ sở dữ liệu và đánh giá hiệu quả của các cách đó theo yêu cầu sau:

Cho biết danh sách các nhà cung cấp đã cung cấp sản phẩm (được nhập)

1: Cơ sở dữ liệu được truy xuất với mỗi giá trị $SNUM

```
Repeat
        read(terminal, $SNUM);
        select PNUM into $PNUM_REL($PNUM)
        from SUPPLY where SNUM = $SNUM;
        repeat
              read($PNUM_REL, $PNUM)
              write(terminal, $PNUM)
        until END-OF-$PNUM_REL
until END-OF-TERMINAL-INPUT
```

Cách 2: Cơ sở dữ liệu được truy xuất sau khi tất cả giá trị của $SNUM được nhập

```
Repeat
    read(terminal, $SNUM);
    write($SNUM_REL($SNUM), $SNUM)
Until END-OF-TERMINAL-INPUT
select PNUM into $PNUM_REL($PNUM)
```

```
        from SUPPLY, $SNUM_REL
        where SUPPLY.SNUM = $SNUM_REL.$SNUM;
    Repeat
        read($PNUM_REL, $PNUM)
        write(terminal, $PNUM)
    Until END-OF-$PNUM_REL
```

Cách 3: Cơ sở dữ liệu được truy xuất trước khi nhập giá trị $SNUM

```
Select PNUM, SNUM into $TEMP_REL($TEMP_PNUM, $TEMP_SNUM)
from SUPPLY;
Repeat
        read(terminal, $SNUM);
        select $TEMP_PNUM into $TEMP2_REL($TEMP2_PNUM)
        from $TEMP_REL  where $TEMP_SNUM = $SNUM;
        Repeat
                read($TEMP2_REL, $TEMP2_PNUM);
                write(terminal, $TEMP2_PNUM);
        until END-OF-$TEMP2_PNUM
Until END-OF-TERMINAL-INPUT
```

Nhận xét về hiêu quả của ba cách trên?