

An toàn và An ninh Dữ liệu

Nội dung chi tiết

* *An toàn Dữ liệu*

» Phân loại sự cố

- ❖ Nhật ký Giao tác
- ❖ Điểm kiểm tra
- ❖ Undo logging
- ❖ Redo logging
- ❖ Undo/Redo logging

* *An ninh Dữ liệu*

- ❖ Cơ chế phân quyền
- ❖ Cơ chế mã hóa

✱ *Cơ sở Dữ liệu luôn cần phải ở trạng thái nhất quán, đảm bảo tất cả ràng buộc toàn vẹn*

✱ *Làm thế nào mà ràng buộc bị vi phạm?*

❖ **Lỗi do sự cố**

- ❖ Lỗi lập trình của các giao tác (transaction bug)
- ❖ Lỗi lập trình của DBMS (DBMS bug) hay OS
- ❖ Hư hỏng phần cứng (hardware failure) và các sự cố khác

❖ **Chia sẻ dữ liệu (data sharing)**

✱ *Làm thế nào để sửa lỗi và khôi phục?*

- ❖ Để giải quyết lỗi do chia sẻ dữ liệu : Các kỹ thuật quản lý giao tác và xử lý đồng thời
- ❖ Để giải quyết lỗi do sự cố : Các kỹ thuật khôi phục sự cố

Đối tượng của chương này

Phân loại sự cố

✱ *Phân loại theo nguyên nhân*

- ❖ Sự cố do nhập liệu sai
- ❖ Sự cố trên thiết bị lưu trữ – Media failure
- ❖ Sự cố của giao dịch – Transaction failure
- ❖ Sự cố liên quan đến hệ thống – System failure (lỗi phần cứng, lỗi phần mềm)

✱ *Nhận xét*

- ❖ Mọi sự cố khác đều cũng dẫn đến sự cố giao tác

✱ *Phân loại theo tính chất*



* **Bao gồm :**

- ❖ **Dữ liệu sai hiển nhiên :** Là sự nhập sai dữ liệu mà máy tính có thể phát hiện được

- ⊗ Vd : Nhập thiếu 1 số trong dãy số điện thoại, nhập sai khóa ngoại, nhập chuỗi tràn...

- ❖ **Dữ liệu sai không hiển nhiên :** Là sự nhập sai dữ liệu liên quan đến ngữ nghĩa mà máy tính khó có thể tự nó phát hiện được

- ⊗ Vd : Nhập sai 1 số trong dãy số điện thoại

* **Giải quyết : Hệ quản trị CSDL cung cấp các cơ chế cho phép phát hiện lỗi**

- ❖ Ràng buộc khóa chính, khóa ngoại
- ❖ Ràng buộc miền giá trị
- ❖ Trigger

Sự cố trên thiết bị lưu trữ

* **Là những sự cố :**

- ❖ Là những sự cố gây nên việc mất hay không thể truy xuất dữ liệu ở bộ nhớ ngoài (ổ cứng, CD, băng từ...)

- ⊗ Vd : Cháy nổ gây phá hủy thiết bị lưu trữ,...

- ⊗ Vd : Đầu đọc của đĩa cứng hư, sector trên đĩa cứng hư, ...

- ❖ Đây là loại sự cố nguy hiểm nhất, khó khôi phục trọn vẹn

* **Giải quyết**

- ❖ Phải backup thường xuyên (toàn bộ hoặc chỉ phần thay đổi), chu kỳ không được quá thưa

- ❖ Chạy nhiều bản CSDL song hành (1 bản chính – primary và nhiều bản phụ – minor) và thực hiện đồng bộ tức thì

- ⊗ Tồn thiết bị lưu trữ và đòi hỏi phần cứng rất mạnh

- ⊗ Kiểm soát tốc độ hệ thống

- ⊗ Bản minor phải đặt ở vị trí địa lý khác bản primary

- * ***Sự cố làm cho 1 giao tác kết thúc không bình thường (không đến được lệnh commit hay lệnh rollback của chính nó)***
- * ***Ví dụ***
 - ❖ Chia cho không
 - ❖ Giao tác bị hủy
 - ❖ Dữ liệu nhập sai
 - ❖ Tràn số
- * ***Giải quyết : Khi giao tác T bị sự cố, DBMS sẽ***
 - ❖ Hủy T và các giao tác bị quay lui dây chuyền theo nó
 - ❖ Tra lock-table và giải phóng các khóa mà các giao tác này đang giữ
 - ❖ Reset lại các giá trị mà các giao tác này đã ghi
 - ❖ Thực hiện lại tất cả các giao tác này

Sự cố hệ thống

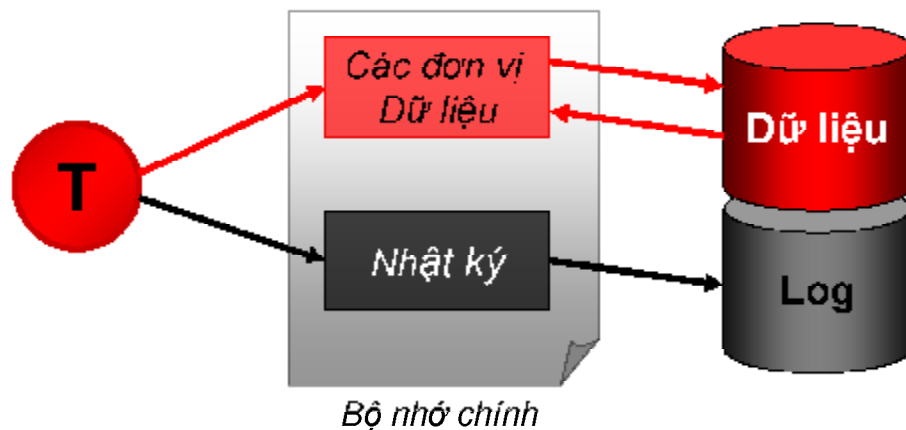
- * ***Là những sự cố gây nên bởi***
 - ❖ Lỗi phần cứng
 - ⊗ Ví dụ
 - ▶ Cúp điện
 - ▶ Hư bộ nhớ trong
 - ▶ Hư CPU
 - ❖ Lỗi phần mềm
 - ⊗ Ví dụ
 - ▶ Lỗi hệ điều hành
 - ▶ Lỗi DBMS
- * ***Giải quyết : Hệ quản trị CSDL cần cứu chữa và phục hồi dữ liệu***
 - ❖ Nhật ký giao tác (transaction log)

- * ***Đưa CSDL về trạng thái nhất quán sau cùng nhất trước khi xảy ra sự cố***
- * ***Đảm bảo 2 tính chất của giao tác***
 - ❖ Nguyên tử (atomic)
 - ❖ Bền vững (durability)

Nội dung chi tiết

- * ***An toàn Dữ liệu***
 - ❖ Phân loại sự cố
 - ❖ Nhật ký Giao tác
 - ❖ Điểm kiểm tra
 - ❖ Undo logging
 - ❖ Redo logging
 - ❖ Undo/Redo logging
- * ***An ninh Dữ liệu***
 - ❖ Cơ chế phân quyền
 - ❖ Cơ chế mã hóa

- * **Nhật ký giao tác là một chuỗi các mẫu tin (log record) ghi nhận lại các hành động của DBMS**
 - ❖ Một mẫu tin cho biết một giao tác nào đó đã làm những gì
- * **Nhật ký là một tập tin tuần tự được lưu trữ trên bộ nhớ chính, và sẽ được ghi xuống đĩa ngay khi có thể**



Nhật ký giao tác (tt)

- * **Mẫu tin nhật ký gồm có**
 - ❖ **<start T>**
 - ⊛ Ghi nhận giao tác T bắt đầu hoạt động
 - ❖ **<commit T>**
 - ⊛ Ghi nhận giao tác T đã hoàn tất
 - ❖ **<abort T>**
 - ⊛ Ghi nhận giao tác T bị hủy
 - ❖ **<T, X, v, w>**
 - ⊛ Ghi nhận giao tác T cập nhật lên đơn vị dữ liệu X
 - ⊛ X có giá trị trước khi cập nhật là v và sau khi cập nhật là w

*** Khi sự cố hệ thống xảy ra**

- ❖ DBMS sẽ tra cứu nhật ký giao tác để khôi phục những gì mà các giao tác đã làm

*** Để sửa chữa các sự cố**

- ❖ Một vài giao tác sẽ phải thực hiện lại (redo)
 - ⊗ Những giá trị mà các giao tác này đã cập nhật xuống CSDL sẽ phải cập nhật lần nữa
- ❖ Một vài giao tác không cần phải thực hiện lại (undo)
 - ⊗ CSDL sẽ được khôi phục về lại trạng thái trước khi các giao tác này được thực hiện

Nội dung chi tiết

*** An toàn Dữ liệu**

- ❖ Phân loại sự cố
- ❖ Nhật ký Giao tác
- ❖ Điểm kiểm tra
- ❖ Undo logging
- ❖ Redo logging
- ❖ Undo/Redo logging

*** An ninh Dữ liệu**

- ❖ Cơ chế phân quyền
- ❖ Cơ chế mã hóa

✱ **Khi cần, DBMS không thể tra cứu toàn bộ nhật ký vì :**

- ❖ Nhật ký tích lũy thông tin về tất cả các hành động của một giai đoạn rất dài
- ❖ Quá trình tra cứu nhật ký
 - ⊛ Phải quét hết tập tin nhật ký
 - ⊛ Vì vậy mất nhiều thời gian
- ❖ Thực hiện lại các giao tác đã được ghi xuống đĩa làm cho việc phục hồi phải lặp lại, mất thời gian vô ích

✱ **Giải pháp : dùng Điểm kiểm tra - Checkpoint**

- ❖ Nhật ký giao tác có thêm mẫu tin <checkpoint> hay <ckpt>
- ❖ Mẫu tin <checkpoint> sẽ được ghi xuống nhật ký định kỳ vào thời điểm mà DBMS ghi tất cả những gì thay đổi của CSDL từ vùng đệm xuống đĩa

Điểm kiểm tra đơn giản

✱ **Khi đến điểm kiểm tra, DBMS sẽ**

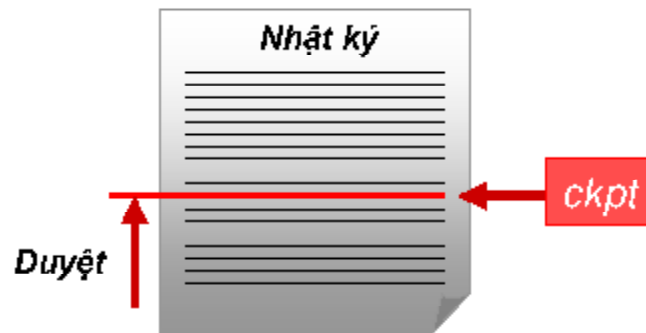
- ❖ Tạm dừng tiếp nhận các giao tác mới
- ❖ Đợi các giao tác đang thực hiện
 - ⊛ Hoặc là hoàn tất (commit)
 - ⊛ Hoặc là hủy bỏ (abort)
 - ⊛ và ghi mẫu tin <commit T> hay <abort T> vào nhật ký
- ❖ Tiến hành ghi dữ liệu và nhật ký từ vùng đệm xuống đĩa
- ❖ Tạo 1 mẫu tin <checkpoint> và ghi xuống đĩa
- ❖ Trở về công việc bình thường (tiếp tục nhận các giao tác mới)

* **Khi có sự cố, DBMS dùng nhật ký để khôi phục :**

- ❖ Các giao tác ở phía trước điểm kiểm tra mới nhất là những giao tác đã kết thúc → không cần làm lại
- ❖ Các giao tác ở phía sau điểm kiểm tra là những giao tác chưa thực hiện xong → cần khôi phục

* **Như vậy, DBMS sẽ :**

- ❖ Không phải duyệt hết nhật ký
- ❖ Chỉ duyệt ngược từ cuối nhật ký đến điểm kiểm tra



Điểm kiểm tra linh động

* **Khi đến điểm kiểm tra, DBMS sẽ**

- ❖ Tạm ngưng mọi hoạt động
- ❖ Chờ các giao tác hoàn tất hoặc hủy bỏ
- ❖ Cho phép tiếp nhận các giao tác mới trong quá trình checkpoint
- ❖ Mẫu tin ckpt trong *Điểm kiểm tra đơn giản* nay được chia thành 2 mẫu tin :
 - ⊛ Mẫu tin $\langle \text{start ckpt} (T_1, T_2, \dots, T_k) \rangle$: Bắt đầu checkpoint
 - ▶ T_1, T_2, \dots, T_k là các giao tác đang dở dang khi bắt đầu checkpoint
 - ⊛ Mẫu tin $\langle \text{end ckpt} \rangle$: Kết thúc checkpoint

✱ **Khi đến điểm kiểm tra, DBMS sẽ**

- ❖ Tạo và ghi mẫu tin <start ckpt (T_1, T_2, \dots, T_k)>
 - ⊛ T_1, T_2, \dots, T_k là những giao tác đang thực thi khi ấy
- ❖ Chờ cho đến khi T_1, T_2, \dots, T_k hoàn tất hay hủy bỏ
- ❖ Trong khi ấy không ngăn các giao tác mới bắt đầu
- ❖ Khi T_1, T_2, \dots, T_k kết thúc, tạo mẫu tin <end ckpt> và ghi xuống đĩa

Nội dung chi tiết

✱ **An toàn Dữ liệu**

- ❖ Phân loại sự cố
- ❖ Nhật ký Giao tác
- ❖ Điểm kiểm tra
- ❖ Undo logging
- ❖ Redo logging
- ❖ Undo/Redo logging

✱ **An ninh Dữ liệu**

- ❖ Cơ chế phân quyền
- ❖ Cơ chế mã hóa

* Qui tắc

- ❖ Một thao tác T khi cập nhật một đơn vị dữ liệu X sẽ phát sinh ra 1 mẫu tin nhật ký
 - ⊛ Mẫu tin của thao tác cập nhật chỉ ghi nhận lại giá trị cũ
 - ⊛ Cấu trúc mẫu tin : $\langle T, X, v \rangle$ với v là giá trị cũ của X
 - ⊛ Ghi mẫu tin này xuống đĩa trước khi cập nhật X xuống đĩa
- ❖ Trước khi mẫu tin $\langle \text{commit}, T \rangle$ được ghi xuống đĩa, tất cả các cập nhật của T đã phải được ghi xuống đĩa
 - ⊛ Flush-log: Là hành động ghi xuống đĩa những block mẫu tin nhật ký mới được phát sinh trên bộ nhớ mà chưa ghi xuống đĩa, không ghi những block mẫu tin nhật ký đã ghi trước đó

Ví dụ

| Bước | Giao tác T | t | Mem A | Mem B | Disk A | Disk B | Mem Log |
|------|------------------|----|-------|-------|--------|--------|------------------------------------|
| 1 | | | | | | | $\langle \text{start } T \rangle$ |
| 2 | Read(A,t) | 8 | 8 | | 8 | 8 | |
| 3 | $t = t * 2$ | 16 | 8 | | 8 | 8 | |
| 4 | Write(A,t) | 16 | 16 | | 8 | 8 | $\langle T, A, 8 \rangle$ |
| 5 | Read(B,t) | 8 | 16 | 8 | 8 | 8 | |
| 6 | $t = t * 2$ | 16 | 16 | 8 | 8 | 8 | |
| 7 | Write(B,t) | 16 | 16 | 16 | 8 | 8 | $\langle T, B, 8 \rangle$ |
| 8 | Flush log | | | | | | |
| 9 | Output(A) | 16 | 16 | 16 | 16 | 8 | |
| 10 | Output(B) | 16 | 16 | 16 | 16 | 16 | |
| 11 | | | | | | | $\langle \text{commit } T \rangle$ |
| 12 | Flush log | | | | | | |

* **Khôi phục**

❖ Gọi S là tập các giao tác chưa kết thúc

✧ Có $\langle \text{start } T_i \rangle$ trong nhật ký nhưng

✧ Không có $\langle \text{commit } T_i \rangle$ hay $\langle \text{abort } T_i \rangle$ trong nhật ký

❖ Với mỗi mẫu tin $\langle T_i, X, v \rangle$ trong nhật ký (theo thứ tự cuối tập tin đến đầu tập tin)

✧ Nếu $T_i \in S$ thì

▶ $\text{Write}(X, v)$


▶ $\text{Output}(X)$

❖ Với mỗi $T_i \in S$


✧ Ghi mẫu tin $\langle \text{abort } T_i \rangle$ lên nhật ký

Ví dụ


| Bước | Giao tác T | t | Mem A | Mem B | Disk A | Disk B | Mem Log |
|------|------------------|----|-------|-------|--------|--------|------------------------------------|
| 1 | | | | | | | $\langle \text{start } T \rangle$ |
| 2 | Read(A,t) | 8 | 8 | | 8 | 8 | |
| 3 | $t := t * 2$ | 16 | 8 | | 8 | 8 | |
| 4 | Write(A,t) | 16 | 16 | | 8 | 8 | $\langle T, A, 8 \rangle$ |
| 5 | Read(B,t) | 8 | 16 | 8 | 8 | 8 | |
| 6 | $t := t * 2$ | 16 | 16 | 8 | 8 | 8 | |
| 7 | Write(B,t) | 16 | 16 | 16 | 8 | 8 | $\langle T, B, 8 \rangle$ |
| 8 | Flush log | | | | | | |
| 9 | Output(A) | 16 | 16 | 16 | 16 | 8 | |
| 10 | Output(B) | 16 | 16 | 16 | 16 | 16 | |
| 11 | | | | | | | $\langle \text{commit } T \rangle$ |
| 12 | Flush log | | | | | | |



Chưa ghi A và B ra đĩa, không khôi phục



Khôi phục A=8 & B=8



Không cần khôi phục A và B

* Điểm kiểm tra

- ❖ Vì T1 và T2 đang thực thi nên chờ
- ❖ Sau khi T1 và T2 hoàn tất hoặc hủy bỏ
- ❖ Ghi mẫu tin <checkpoint> lên nhật ký

```
<start T1>
<T1, A, 5>
<start T2>
<T2, B, 10>
```

Checkpoint đơn giản

* Khôi phục

❖ T3 chưa kết thúc

- ⦿ <T3, F, 30>
 - ▶ Khôi phục F=30
- ⦿ <T3, E, 25>
 - ▶ Khôi phục E=25

❖ <checkpoint>

- ⦿ Dừng
- ⦿ Ghi abort(T3)

```
<start T1>
<T1, A, 5>
<start T2>
<T2, B, 10>
<T2, C, 15>
<T2, D, 20>
<commit T1>
<commit T2>
<checkpoint>
<start T3>
<T3, E, 25>
<T3, F, 30>
```

Quét
đến đây
thì dừng

Undo-Logging & Checkpoint linh động

* Điểm kiểm tra, DBMS sẽ

- ❖ Vì T1 và T2 đang thực thi nên tạo <start ckpt (T1,T2)>
- ❖ Trong khi chờ T1 và T2 kết thúc, DBMS vẫn tiếp nhận các giao tác mới
- ❖ Sau khi T1 và T2 kết thúc, ghi <end ckpt> lên nhật ký

```
<start T1>
<T1, A, 5>
<start T2>
<T2, B, 10>
```

Checkpoint Linh động

* *Khôi phục*

❖ **<T3, F, 30>**

- ⊗ T3 chưa kết thúc
- ⊗ Khôi phục F=30

❖ **<end ckpt>**

- ⊗ Những giao tác bắt đầu trước <start ckpt> đã hoàn tất
- ⊗ T1 và T2 đã hoàn tất

❖ **<T3, E, 25>**

- ⊗ Khôi phục E=25

❖ **<start ckpt (T1, T2)>**

- ⊗ Dừng
- ⊗ Ghi abort(T3)

```
<start T1>
<T1, A, 5>
<start T2>
<T2, B, 10>
<start ckpt (T1,T2)>
<T2, C, 15>
<start T3>
<T1, D, 20>
<commit T1>
<T3, E, 25>
<commit T2>
<end ckpt>
<T3, F, 30>
```

Quét
đến đây
thì dừng

Undo-Logging & Checkpoint linh động (tt)

* *Khôi phục khi thiếu <end ckpt>*

❖ **<T3, E, 25>**

- ⊗ T3 chưa kết thúc → khôi phục E=25

❖ **<commit T1>**

- ⊗ T1 bắt đầu trước <start ckpt> và đã hoàn tất

❖ **<T2, C, 15>**

- ⊗ T2 bắt đầu trước <start ckpt> và chưa kết thúc
- ⊗ Khôi phục C=15

❖ **<start ckpt (T1, T2)>**

- ⊗ Chỉ có T1 và T2 bắt đầu trước đó

❖ **<T2, B, 10>**

- ⊗ Khôi phục B=10

❖ **<start T2>**

- ⊗ Dừng, ghi abort(T3) và ghi abort(T2)

```
<start, T1>
<T1, A, 5>
<start, T2>
<T2, B, 10>
<start ckpt (T1,T2)>
<T2, C, 15>
<start, T3>
<T1, D, 20>
<commit T1>
<T3, E, 25>
```

Quét
đến đây
thì dừng

✱ *An toàn Dữ liệu*

- ❖ Phân loại sự cố
- ❖ Nhật ký Giao tác
- ❖ Điểm kiểm tra
- ❖ Undo logging
- ❖ Redo logging
- ❖ Undo/Redo logging

✱ *An ninh Dữ liệu*

- ❖ Cơ chế phân quyền
- ❖ Cơ chế mã hóa

Phương pháp Redo-Logging

✱ *Qui tắc*

- ❖ Một thao tác T muốn cập nhật đơn vị dữ liệu X sẽ phát sinh ra 1 mẫu tin nhật ký
 - ⊛ Mẫu tin của thao tác cập nhật chỉ ghi nhận lại giá trị mới
 - ⊛ Cấu trúc mẫu tin $\langle T, X, w \rangle$ với w là giá trị mới của X
- ❖ Trước khi X được cập nhật xuống đĩa
 - ⊛ Tất cả các mẫu tin nhật ký $\langle T_i, X, w \rangle$ của các giao tác T_i cập nhật X đã phải có trên đĩa
 - ⊛ Kể cả các mẫu tin hoàn tất $\langle \text{commit } T_i \rangle$ của các T_i cũng đã phải được ghi xuống đĩa

| Bước | Hành động | t | Mem A | Mem B | Disk A | Disk B | Mem Log |
|------|------------------|----|-------|-------|--------|--------|------------|
| 1 | | | | | | | <start T> |
| 2 | Read(A,t) | 8 | 8 | | 8 | 8 | |
| 3 | $t = t * 2$ | 16 | 8 | | 8 | 8 | |
| 4 | Write(A,t) | 16 | 16 | | 8 | 8 | <T, A, 16> |
| 5 | Read(B,t) | 8 | 16 | 8 | 8 | 8 | |
| 6 | $t = t * 2$ | 16 | 16 | 8 | 8 | 8 | |
| 7 | Write(B,t) | 16 | 16 | 16 | 8 | 8 | <T, B, 16> |
| 8 | | | | | | | <commit T> |
| 9 | Flush log | | | | | | |
| 10 | Output(A) | 16 | 16 | 16 | 16 | 8 | |
| 11 | Output(B) | 16 | 16 | 16 | 16 | 16 | |

Phương pháp Redo-Logging (tt)

* *Khôi phục*

❖ Gọi S là tập các giao tác hoàn tất

⊛ Có mẫu tin <commit T_i > trong nhật ký

❖ Với mỗi mẫu tin < T_i , X, w> trong nhật ký (theo thứ tự cuối tập tin đến đầu tập tin)

⊛ Nếu $T_i \in S$ thì

▶ *Write(X, w)*

▶ *Output(X)*

❖ Với mỗi $T_j \notin S$

⊛ Ghi mẫu tin <abort T_j > lên nhật ký

| Bước | Hành động | t | Mem A | Mem B | Disk A | Disk B | Mem Log |
|------|------------|----|-------|-------|--------|--------|---|
| 1 | | | | | | | <start T> |
| 2 | Read(A,t) | 8 | 8 | | 8 | 8 | |
| 3 | t:=t*2 | 16 | 8 | | 8 | 8 | |
| 4 | Write(A,t) | 16 | 16 | | 8 | 8 | <T, A, 16> |
| 5 | Read(B,t) | 8 | 16 | 8 | 8 | 8 | |
| 6 | t:=t*2 | 16 | 16 | 8 | 8 | 8 | |
| 7 | Write(B,t) | 16 | 16 | 16 | 8 | 8 | <T, B, 16> |
| 8 | | | | | | | <commit T> |
| 9 | Flush log | | | | | | Xem như T chưa hoàn tất, A và B không có thay đổi |
| 10 | Output(A) | 16 | 16 | 16 | 16 | 8 | |
| 11 | Output(B) | 16 | 16 | 16 | 16 | 16 | Thực hiện lại T, ghi A=16 và B=16 |

Redo-Logging & Checkpoint linh động

* Đến điểm kiểm tra, DBMS

- ❖ Tạo mẫu tin <start ckpt (T1, T2,..., Tk)> và ghi xuống đĩa
 - ⊗ T1, T2, ..., Tk là những giao tác đang thực thi lúc ấy
- ❖ Ghi xuống đĩa những dữ liệu trên vùng đệm của các giao tác đã commit
- ❖ Tạo mẫu tin <end ckpt> và ghi xuống đĩa
 - ⊗ Không cần đợi T1, T2,..., Tk hoàn tất → Khác với quy tắc chung của checkpoint linh động

* Ví dụ 1

❖ T1 đã hoàn tất trước <start ckpt>

- Có thể đã được ghi xuống đĩa
- Nếu chưa thì trước khi <end ckpt> cũng được ghi xuống đĩa

❖ Sau <start ckpt>

- T2 đang thực thi
- T3 bắt đầu

❖ Sau <end ckpt>

- T2 và T3 hoàn tất

```
<start T1>
<T1, A, 5>
<start T2>
<commit T1>
<T2, B, 10>
<start ckpt (T2)>
<T2, C, 15>
<start T3>
<T3, D, 20>
<end ckpt>
<commit T2>
<commit T3>
```

Redo-Logging & Checkpoint linh động (tt)

* Ví dụ 1

❖ T2 và T3 commit sau <end ckpt> nên xét T2 và T3

❖ <commit T2>

- Thực hiện lại T2
- Ghi C=15 và B=10

❖ <commit T3>

- Thực hiện lại T3
- Ghi D=20

```
<start T1>
<T1, A, 5>
<start T2>
<commit T1>
<T2, B, 10>
<start ckpt (T2)>
<T2, C, 15>
<start T3>
<T3, D, 20>
<end ckpt>
<commit T2>
<commit T3>
```

Quét
đến đây
thì dừng

✱ **Ví dụ 2 (khôi phục khi thiếu <end ckpt>)**

❖ T1 đã hoàn tất trước <start ckpt>...

nhưng vì thiếu <end ckpt> nên
có thể T1 chưa ghi đĩa

- ⊗ Thực hiện lại T1
- ⊗ Ghi A=5

❖ T2 và T3 chưa hoàn tất

- ⊗ Không thực hiện lại
- ⊗ Ghi abort(T2)
- ⊗ Ghi abort(T3)

```
<start T1>
<T1, A, 5>
<start T2>
<commit T1>
<T2, B, 10>
<start ckpt (T2)>
<T2, C, 15>
<start T3>
<T3, D, 20>
```

Nhận xét

✱ **Undo-logging (immediate modification)**

- ❖ Khi giao tác kết thúc, dữ liệu được ghi xuống đĩa ngay lập tức
- ❖ Truy xuất đĩa nhiều nhưng không chiếm dụng nhiều bộ nhớ

✱ **Redo-logging (deferred modification)**

- ❖ Giữ lại các cập nhật trên vùng đệm cho đến khi giao tác hoàn tất và mẫu tin nhật ký được ghi xuống đĩa
- ❖ Tốn nhiều bộ nhớ nhưng giảm tần suất truy xuất đĩa

✳ *An toàn Dữ liệu*

- ❖ Phân loại sự cố
- ❖ Nhật ký Giao tác
- ❖ Điểm kiểm tra
- ❖ Undo logging
- ❖ Redo logging

✳ *Undo/Redo logging*

✳ *An ninh Dữ liệu*

- ❖ Cơ chế phân quyền
- ❖ Cơ chế mã hóa

Phương pháp Undo/Redo-Logging

✳ *Qui tắc*

- ❖ Khi 1 giao tác T muốn ghi đơn vị dữ liệu X thì phát sinh ra 1 mẫu tin nhật ký
 - ⊗ Mẫu tin của thao tác cập nhật ghi nhận giá trị cũ và mới của đơn vị dữ liệu X
 - ⊗ Cấu trúc mẫu tin : $\langle T, X, v, w \rangle$ với v là giá trị cũ và w là giá trị mới
- ❖ Trước khi X được cập nhật xuống đĩa, các mẫu tin cập nhật $\langle T, X, v, w \rangle$ đã phải có trên đĩa
- ❖ Khi T hoàn tất (nhưng không nhất thiết đã ghi hết dữ liệu cập nhật ra đĩa), tạo mẫu tin $\langle \text{commit } T \rangle$ trên nhật ký và ghi xuống đĩa

| Bước | Hành động | t | Mem A | Mem B | Disk A | Disk B | Mem Log |
|------|------------------|----|-------|-------|--------|--------|---------------|
| 1 | | | | | | | <start T> |
| 2 | Read(A,t) | 8 | 8 | | 8 | 8 | |
| 3 | $t := t * 2$ | 16 | 8 | | 8 | 8 | |
| 4 | Write(A,t) | 16 | 16 | | 8 | 8 | <T, A, 8, 16> |
| 5 | Read(B,t) | 8 | 16 | 8 | 8 | 8 | |
| 6 | $t := t * 2$ | 16 | 16 | 8 | 8 | 8 | |
| 7 | Write(B,t) | 16 | 16 | 16 | 8 | 8 | <T, B, 8, 16> |
| 8 | Flush log | | | | | | |
| 9 | Output(A) | 16 | 16 | 16 | 16 | 8 | |
| 10 | | | | | | | <commit T> |
| 11 | Output(B) | 16 | 16 | 16 | 16 | 16 | |

Phương pháp Undo/Redo-Logging (tt)

* *Khôi phục*

❖ Gọi S là tập các giao tác hoàn tất

✪ Có mẫu tin <commit Ti> trong nhật ký

❖ Gọi S' là phần bù của S

❖ Với mỗi mẫu tin <Ti, X, v, w> trong nhật ký (theo thứ tự cuối tập tin đến đầu tập tin)

✪ Nếu $T_i \in S$ thì Redo T

✪ Ngược lại thì Undo T

❖ Với mỗi $T_j \in S'$

$T_j \notin S$

✪ Ghi mẫu tin <abort Tj> lên nhật ký

| Bước | Hành động | t | Mem A | Mem B | Disk A | Disk B | Mem Log |
|------|------------------|----|-------|-------|--------|--------|---------------|
| 1 | | | | | | | <start T> |
| 2 | Read(A,t) | 8 | 8 | | 8 | 8 | |
| 3 | t:=t*2 | 16 | 8 | | 8 | 8 | |
| 4 | Write(A,t) | 16 | 16 | | 8 | 8 | <T, A, 8, 16> |
| 5 | Read(B,t) | 8 | 16 | 8 | 8 | 8 | |
| 6 | t:=t*2 | 16 | 16 | 8 | 8 | 8 | |
| 7 | Write(B,t) | 16 | 16 | 16 | 8 | 8 | <T, B, 8, 16> |
| 8 | Flush log | | | | | | |
| 9 | Output(A) | 16 | 16 | 16 | 16 | 8 | |
| 10 | | | | | | | <commit T> |
| 11 | Output(B) | 16 | 16 | 16 | 16 | 16 | |

T chưa kết thúc, khôi phục A=8

<commit T> đã được ghi xuống đĩa, thực hiện lại T, A=16 và B=16

Undo/Redo-Logging & Checkpoint

* Khi đến điểm kiểm tra, DBMS

❖ Tạo mẫu tin <start ckpt (T1, T2,..., Tk)> và ghi xuống đĩa

⊙ T1, T2, ..., Tk là những giao tác đang thực thi

❖ Ghi xuống đĩa những dữ liệu đang nằm trên vùng đệm

* (⊙ Những đơn vị dữ liệu được cập nhật bởi các giao tác trước khi bắt đầu điểm kiểm tra, bất kể đã hoàn tất hay chưa

❖ Tạo mẫu tin <end ckpt> trong nhật ký và ghi xuống đĩa

⊙ Không cần đợi T1, T2,..., Tk hoàn tất → Khác với quy tắc chung của checkpoint linh động

✓ Trước <Start Ckpt (T_i)>

* Ví dụ 1

❖ T1 đã hoàn tất trước <start ckpt>

⊙ Có thể đã được ghi xuống đĩa

⊙ Nếu chưa thì trước khi <end ckpt> cũng được ghi xuống đĩa

❖ Giá trị B=10 đã được ghi xuống đĩa

```
<start T1>
<T1, A, 4, 5>
<start T2>
<commit T1>
<T2, B, 9, 10>
<start ckpt (T2)>
<T2, C, 14, 15>
<start T3>
<T3, D, 19, 20>
<end ckpt>
<commit T2>
<commit T3>
```

→ đang dở

Undo/Redo-Logging & Checkpoint (tt)

* Ví dụ 1

❖ Tìm thấy <end ckpt>

⊙ T1 không cần thực hiện lại

❖ Xét T2 và T3

❖ <commit T2>

⊙ Thực hiện lại T2 và ghi C=15

⊙ Không cần ghi B

❖ <commit T3>

⊙ Thực hiện lại T3 và ghi D=20

```
<start T1>
<T1, A, 4, 5>
<start T2>
<commit T1>
<T2, B, 9, 10>
<start ckpt (T2)>
<T2, C, 14, 15>
<start T3>
<T3, D, 19, 20>
<end ckpt>
<commit T2>
<commit T3>
```

Quét đến đây thì dừng

* Ví dụ 2

❖ Tìm thấy <end ckpt>

- ☉ T1 không cần thực hiện lại

❖ Xét T2 và T3

❖ <commit T2>

- ☉ Thực hiện lại T2 và ghi C=15
- ☉ Không cần ghi B

❖ T3 chưa kết thúc

- ☉ Khôi phục D=19 và E=6
- ☉ Ghi abort(T3)

```
<start T1>
<T1, A, 4, 5>
<start T2>
<commit T1>
<start T3>
<T2, B, 9, 10>
<T3, E, 6, 7>
<start ckpt (T2, T3)>
<T2, C, 14, 15>
<T3, D, 19, 20>
<end ckpt>
<commit T2>
```

Quét
đến đây
thì dừng

Nhận xét

* Các kỹ thuật

- ❖ Undo-Logging
- ❖ Redo-Logging
- ❖ Undo-Redo-Logging

* Điều không quan tâm vấn đề quay lui dây chuyền

- ❖ Do đó Bộ lập lịch cần đảm bảo lịch thao tác là một lịch không quay lui dây chuyền

*** An toàn Dữ liệu**

- ❖ Phân loại sự cố
- ❖ Nhật ký Giao tác
- ❖ Điểm kiểm tra
- ❖ Undo logging
- ❖ Redo logging
- ❖ Undo/Redo logging

*** An ninh Dữ liệu**

- ❖ Cơ chế phân quyền
- ❖ Cơ chế mã hóa

An ninh Dữ liệu

*** Thực hiện hai bài toán :**

❖ Bài toán phân quyền

- ⊗ Quản lý tốt việc truy xuất Dữ liệu của các đối tượng người dùng hợp pháp → Bảo mật Dữ liệu
- ⊗ Thông qua 2 cơ chế
 - ▶ **Cơ chế chứng thực**
 - ▶ **Cơ chế phân quyền**
 - » Quan điểm phân quyền cụ thể
 - » Quan điểm phân cấp mức độ MẬT

❖ Bài toán mã hóa

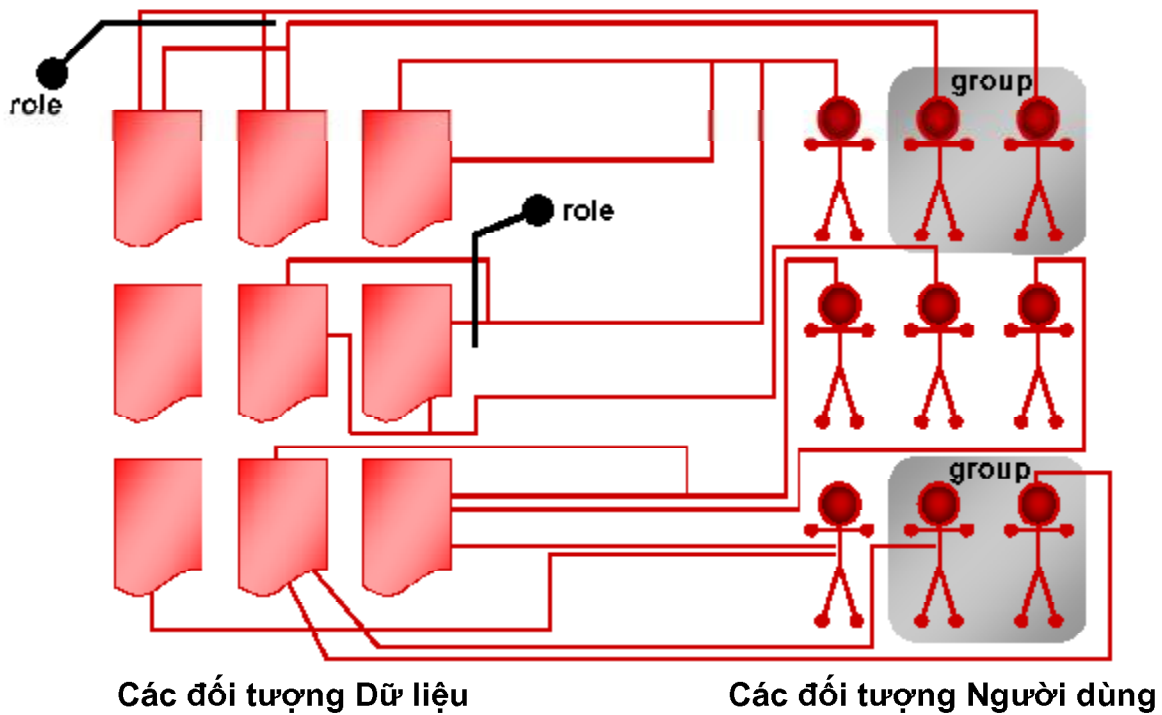
- ⊗ Ngăn chặn hiệu quả sự tấn công, xâm nhập của các đối tượng tin tặc → An ninh Dữ liệu

- * *Mỗi người dùng DBMS được xác định bởi*
 - ❖ Một tên đăng nhập – user name
 - ❖ Một mật mã đăng nhập – password
- * *Thông tin về user name và password*
 - ❖ Không được lưu trữ tường minh trong dữ liệu
 - ❖ User name và password của DBMS và của OS có thể tách bạch nhau hay dùng chung cho nhau là tùy hệ thống
 - ⊙ Vd : Mixed-mode của Microsoft SQL Server

Cơ chế phân quyền

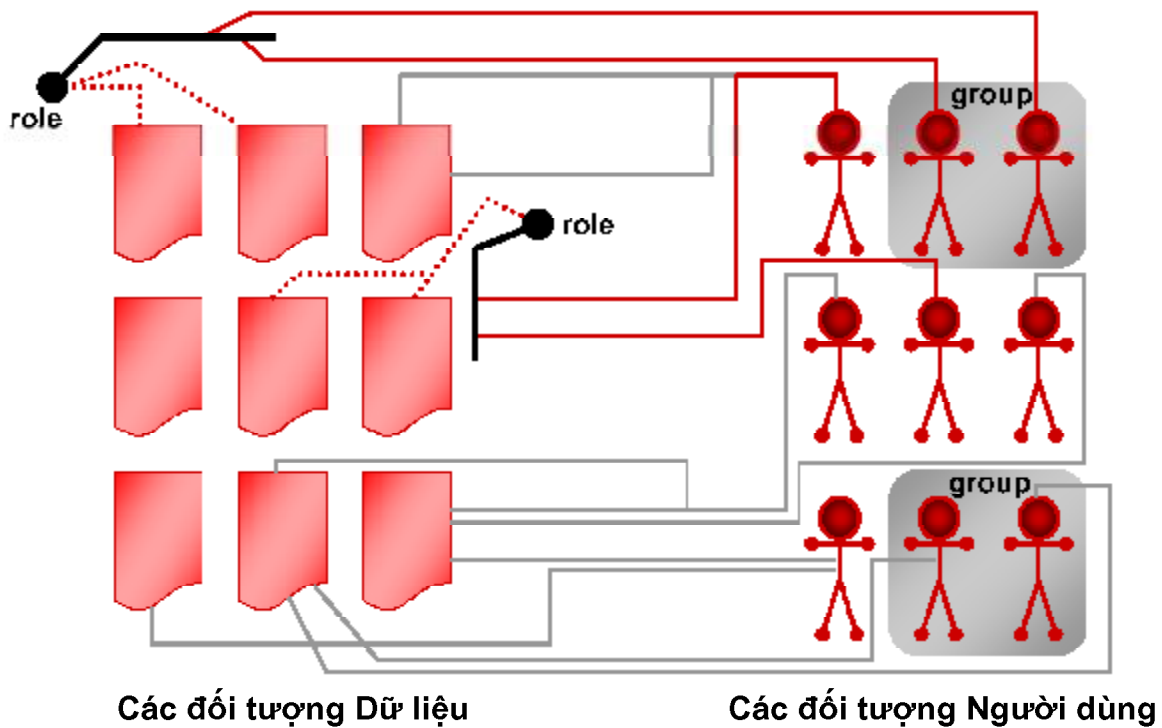
- * *Một tài khoản chứng thực*
 - ❖ Được phép đăng nhập vào hệ thống DBMS
 - ❖ Được nhìn thấy các CSDL
 - ❖ Chưa được phép truy xuất các đối tượng trong các CSDL
- * *Tài khoản chứng thực muốn truy xuất các đối tượng dữ liệu thì cần được phân quyền cụ thể chi tiết trên các đối tượng dữ liệu đó*

* Quan điểm phân quyền cụ thể



Cơ chế phân quyền (tt)

* Quan điểm phân quyền cụ thể



* **Quan điểm phân cấp mức độ MẬT**

- ❖ Các đối tượng Dữ liệu được phân ra các cấp độ bảo mật khác nhau

⊗ Vd :

- ▶ Cấp 3 : Dành cho tài liệu tuyệt mật
- ▶ Cấp 2 : Dành cho tài liệu mật
- ▶ Cấp 1 : Dành cho tài liệu công khai

- ❖ Các đối tượng Người dùng cũng được phân ra các cấp độ bảo mật khác nhau

⊗ Vd :

- ▶ Cấp 3 : Dành cho ban giám đốc
- ▶ Cấp 2 : Dành cho các trưởng phòng
- ▶ Cấp 1 : Dành cho nhân viên

- ❖ Khó khăn : Làm sao phân cấp cho hợp lý (♣)

Cơ chế phân quyền (tt)

* **Quan điểm phân cấp mức độ MẬT**

❖ Phân quyền

- ⊗ Quyền đọc dữ liệu : Người dùng cấp i được đọc các tài liệu cấp i trở xuống

- ⊗ Quyền ghi dữ liệu : (♣♣)

- ▶ Ban giám đốc đọc các tài liệu mật nhưng tài liệu ấy không nhất định do họ tạo ra, thông thường lại do nhân viên tạo ra
- ▶ Người dùng cấp i được ghi tài liệu cấp i trở xuống
- ▶ Nếu người dùng X thuộc cấp i tạo ra tài liệu A thuộc cấp j (với $j > i$) thì chỉ có X được đọc A trong khi các X' cùng cấp không được đọc A

- ❖ Vì (♣) và (♣♣) nên quan điểm này gặp nhiều thách thức và ít được ứng dụng trong các DBMS thương mại

*** Bất chấp cơ chế phân quyền, nhiều đối tượng người dùng bất hợp pháp vẫn có thể xâm nhập vào CSDL**

❖ Ví dụ :

- ❖ Thâm nhập từ mức Hệ điều hành để chép các file dữ liệu của DBMS (như file *.mdf và *.ndf của SQL Server)
- ❖ Chặn trên đường truyền mạng để hứng lấy dữ liệu luân chuyển giữa Client và Server

*** Giải pháp : Mã hóa thông tin trước lưu trữ hoặc truyền trên đường truyền**

- ❖ Tin tặc lấy được file hay dữ liệu cũng không hiểu được
- ❖ Việc mã hóa không được xung đột với hệ thống index → thách thức
- ❖ Thuật toán mã hóa được chọn sao cho việc giải mã của tin tặc là khó khăn nhất

Hết chương IV

