

Phụ lục 5

**TRƯỜNG ĐẠI HỌC TRÀ VINH
KHOA KỸ THUẬT VÀ CÔNG NGHỆ**



**TÀI LIỆU GIẢNG DẠY
MÔN HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU**

GV biên soạn: Phan Thị Phương Nam

Trà Vinh, Tháng 5 năm 2015

Lưu hành nội bộ



KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN

TÌNH TRẠNG PHÊ DUYỆT TÀI LIỆU GIẢNG DẠY

Tên tài liệu giảng dạy: **HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU**

Ngày hoàn chỉnh: **10/06/2015**

Tác giả biên soạn: **PHAN THỊ PHƯƠNG NAM**

Đơn vị công tác: Bộ Môn Công Nghệ Thông Tin

Địa chỉ liên lạc: Bộ Môn Công Nghệ Thông Tin, Khoa Kỹ thuật và Công nghệ

Trà Vinh, ngày 10 tháng 06 năm 2015

Tác giả

Phan Thị Phương Nam

PHÊ DUYỆT CỦA BỘ MÔN

Đồng ý sử dụng tài liệu giảng dạy
do biên soạn để giảng dạy
môn.....

Trà Vinh, ngày tháng năm.....

TRƯỞNG BỘ MÔN

PHÊ DUYỆT CỦA KHOA

Trà Vinh, ngày tháng năm 20.....

TRƯỞNG KHOA

MỤC LỤC

Nội dung	Trang
CHƯƠNG 1 GIỚI THIỆU HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU	1
I.1 ĐỊNH NGHĨA HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU.....	1
I.2 CHỨC NĂNG CỦA HQTCSQL.....	2
<i>I.2.1 Lưu trữ, truy xuất và cập nhật dữ liệu.....</i>	2
<i>I.2.2 Danh mục hệ thống.....</i>	2
<i>I.2.3 An toàn dữ liệu</i>	2
<i>I.2.4 Toàn vẹn dữ liệu</i>	3
<i>I.2.5 Điều khiển cạnh tranh</i>	3
<i>I.2.6 Phục hồi CSDL.....</i>	3
<i>I.2.7 Các tiện ích khác</i>	3
I.3 CÁC THÀNH PHẦN CỦA MỘT HQTCSQL	4
I.4. KIẾN TRÚC HỆ CSDL ĐA NGƯỜI DÙNG	6
<i>I.4.1 Kiến trúc xử lý từ xa</i>	6
<i>I.4.2. Kiến trúc máy chủ tập tin (Tập tin-server).....</i>	6
<i>I.4.3. Kiến trúc Client-server hai tầng.....</i>	7
<i>I.4.4. Kiến trúc Client-server ba tầng</i>	8
I.5 NHÀ QUẢN TRỊ CSDL	9
<i>I.5.1 Vai trò, nhiệm vụ của nhà quản trị CSDL</i>	9
<i>I.5.2 Các tiêu chí đòi hỏi ở một DBA.....</i>	10
<i>I.5.3 Các vai trò quản trị khác nhau</i>	10
I.6 QUÁ TRÌNH PHÁT TRIỂN CỦA HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU.	11
CHƯƠNG 2 GIAO TÁC (TRANSACTION)	13
II.1 GIAO TÁC	13
<i>II.1.1 Định nghĩa giao tác.....</i>	13
<i>II.1.2 Các tính chất của giao tác</i>	13
<i>II.1.3 Các trạng thái của giao tác</i>	16
II.2 LỊCH THAO TÁC (SCHEDULE).....	18
<i>II.2.1 Điều khiển cạnh tranh.....</i>	18
<i>II.2.2 Lịch tuần tự (Serial schedule)</i>	19
<i>II.2.3 Lịch khả tuần tự (Serializable schedule)</i>	20
<i>II.2.3.1 Khả tuần tự xung đột (Conflict-Serializability)</i>	22

II.2.3.2 Kiểm tra khả tuần tự xung đột (Conflict-Serializability)	26
II.2.4 Khả tuần tự View (View Serializability)	29
CHƯƠNG 3 ĐIỀU KHIỂN ĐỒNG THỜI	34
III.1 CÁC VẤN ĐỀ TRONG ĐIỀU KHIỂN ĐỒNG THỜI	34
III.1.1 Mất dữ liệu đã cập nhật (lost updated)	34
III.1.2 Không thể đọc lại (unrepeatable read)	35
III.1.3 “Bóng ma” (phantom)	35
III.1.4 Đọc dữ liệu chưa chính xác (dirty read)	36
III.2 KỸ THUẬT KHÓA (LOCK).....	36
III.2.1 Giới thiệu	36
III.2.2 Qui tắc	37
III.2.3 Khóa 2 giai đoạn (Two Phase Lock - 2PL).....	40
III.2.4 Kỹ thuật khóa đọc viết	42
III.2.5 Kỹ thuật khóa đa hạt	46
III.3 KỸ THUẬT NHÃN THỜI GIAN (TIMESTAMP)	48
III.3.1 Giới thiệu	48
III.3.2 Nhãn thời gian toàn phần	49
III.3.3 Nhãn thời gian riêng phần	50
III.3.4 Nhãn thời gian nhiều phiên bản	57
CHƯƠNG 4 KHÔI PHỤC DỮ LIỆU	60
IV.1 BẢO MẬT DỮ LIỆU / AN TOÀN DỮ LIỆU TRONG SQL SERVER.....	61
IV.1.1 Khái niệm cơ bản về bảo mật.....	61
IV.1.2 Lựa chọn bảo mật	62
IV.1.3 Quyền người dùng và quản trị quyền người dùng	64
IV.1.4 Vai trò của người sử dụng trong SQL Server và cơ sở dữ liệu	68
IV.2 KHÔI PHỤC DỮ LIỆU.....	70
IV.2.1 Mục tiêu của khôi phục sự cố	70
IV.2.2 Nhật ký giao tác	70
IV.2.3 Điểm lưu trữ	71
IV.2.4 Phương pháp khôi phục	73
CHƯƠNG 5 CẤU TRÚC LUU TRỮ DỮ LIỆU TRÊN ĐĨA	88
V.1 CÁC THÀNH PHẦN LIÊN QUAN ĐẾN VIỆC QUẢN LÝ VÀ TRUY XUẤT DỮ LIỆU	88
V.2 ĐĨA TÙ	90

V.2.1	<i>Đặc trưng vật lý của đĩa</i>	90
V.2.2	<i>Đo lường hiệu năng của đĩa</i>	91
V.2.3	<i>Tối ưu hóa truy xuất khối đĩa (disk-block)</i>	92
V.2.4	<i>RAID (Redundant Arrays of Inexpensive Disks)</i>	93
V.3	TRUY XUẤT LUƯ TRỮ	97
V.4	TỔ CHỨC TẬP TIN	100
V.4.1	<i>Mẫu tin độ dài cố định (Fixed-Length Records)</i>	100
V.4.2	<i>Mẫu tin độ dài thay đổi (Variable-Length Records)</i>	102
V.5	TỔ CHỨC CÁC MẪU TIN TRONG TẬP TIN	105
V.5.1	<i>Tổ chức tập tin tuần tự</i>	106
V.5.2	<i>Tổ chức tập tin cụm</i>	107
V.5.3	<i>Lưu trữ tự điển dữ liệu</i>	109
V.5.4	<i>Chỉ mục được sắp</i>	111
V.5.5	<i>Tập tin chỉ mục B^+-cây (B^+-tree index file)</i>	116
V.5.6	<i>Băm (HASHING)</i>	120
CHƯƠNG 6	ƯỚC LƯỢNG CHI PHÍ THỰC HIỆN CÂU TRUY VĂN	126
VI.1	XỬ LÝ TRUY VĂN (QUERY PROCESSING)	126
VI.1.1	<i>Phân tích câu truy vấn</i>	127
VI.1.2	<i>Biến đổi sang đại số quan hệ</i>	129
VI.2	TỐI ƯU CÂU TRUY VĂN	133
VI.2.1	<i>Qui tắc: Kết tự nhiên, tích cartesian, hôi</i>	133
VI.2.2	<i>Qui tắc: Phép chọn σ</i>	133
VI.2.3	<i>Qui tắc: σ</i>	134
VI.2.4	<i>Qui tắc: σ, \cup và $\sigma, -$</i>	134
VI.2.5	<i>Qui tắc: Phép chiếu π</i>	134
VI.2.6	<i>Qui tắc: π</i>	134
VI.2.7	<i>Qui tắc: σ, π</i>	134
VI.2.8	<i>Qui tắc: σ, π</i>	135
VI.2.9	<i>Qui tắc: x</i>	135
VI.2.10	<i>Qui tắc: δ</i>	135
VI.2.11	<i>Qui tắc: γ</i>	136
VI.3	ƯỚC LƯỢNG CHI PHÍ	138
VI.3.1	<i>Thống kê cơ sở dữ liệu</i>	138

VI.3.2 Uớc lượng chi phí thực hiện	139
TÀI LIỆU THAM KHẢO	145
PHỤ LỤC 1:	1
PHỤ LỤC 2: VIEW	22
PHỤ LỤC 3: THỦ TỤC LUU TRU (STORED PROCEDURE)	31
PHỤ LỤC 4: TRIGGER	43
PHỤ LỤC 5: GIAO TÁC	49
PHỤ LỤC 6: BÀI TẬP THỰC HÀNH SỐ 1	54
PHỤ LỤC 7: BÀI TẬP THỰC HÀNH SỐ 2	59

DANH MỤC HÌNH

STT	Tên hình	Trang
1	Hình 1.1: Hệ cơ sở dữ liệu	1
2	Hình 1.2: Các thành phần của một HQTCSDL	4
3	Hình 1.3: Các thành phần của bộ quản lý CSDL	5
4	Hình 1.4: Kiến trúc xử lý từ xa	6
5	Hình 1.5: Kiến trúc máy chủ tập tin	7
6	Hình 1.6: Kiến trúc Client-Server	7
7	Hình 1.7: Kiến trúc Client-server hai tầng	8
8	Hình 1.8: Kiến trúc Client-server ba tầng	9
9	Hình 1.9: Trách nhiệm của DA, DBA và SA	11
10	Hình 2.1: Giao dịch chuyển tiền từ tài khoản A sang tài khoản B	14
11	Hình 2.2: Sơ đồ trạng thái của giao dịch	17
12	Hình 2.3: Lịch giao dịch	19
13	Hình 2.4: Lịch tuần tự	19
14	Hình 2.5: Lịch tuần tự	20
15	Hình 2.6: Lịch khả tuần tự	20
16	Hình 2.7: Lịch khả tuần tự	21
17	Hình 2.8: Lịch không khả tuần tự	21
18	Hình 2.9: Lịch khả tuần tự	22
19	Hình 2.10: Hai hành động xung đột nhau	23
20	Hình 2.11: Lịch giao dịch có các hành động xung đột nhau	23
21	Hình 2.12: Đổi chỗ hai hành động xung đột nhau	24
22	Hình 2.13: Lịch trình tuần tự S' tương đương với lịch trình ban đầu S ₆	24
23	Hình 2.14: Lịch trình không tương đương xung đột	25

STT	Tên hình	Trang
24	Hình 2.15: Lịch trình khả tuần tự nhưng không khả tuần tự xung đột	25
25	Hình 2.16: Lịch trình tuần tự	26
26	Hình 2.17: Lịch trình khả tuần tự nhưng không khả tuần tự xung đột	26
27	Hình 2.18: Hai lịch trình có cùng đồ thị trình tự nhưng không tương đương xung đột	27
28	Hình 2.19: Sắp xếp topo	27
29	Hình 2.20: Thứ tự tuyến tính trong sắp xếp topo	28
30	Hình 2.21: Đồ thị trình tự P(S) không có chu trình	28
31	Hình 2.22: Đồ thị trình tự P(S) có chu trình	28
32	Hình 2.23: Cung đi từ T_j đến T_i nếu có $r_i(X)$ với gốc là T_j	30
33	Hình 2.24: Vẽ cung cho trường hợp(2a)	31
34	Hình 2.25a: Vẽ cung cho trường hợp (2b)	31
35	Hình 2.25b: Vẽ cung cho trường hợp (2c)	31
36	Hình 3.1: Hai giao tác độc lập	34
37	Hình 3.2: Hai giao tác thực hiện đồng thời	34
38	Hình 3.3: Không thể đọc lại dữ liệu	35
39	Hình 3.4: “Bóng ma”	35
40	Hình 3.5: Đọc dữ liệu rác	36
41	Hình 3.6: Bộ lập lịch với cơ chế khóa	36
42	Hình 3.7: Giao tác yêu cầu khóa	37
43	Hình 3.8: Giao tác đúng, lịch hợp lệ	38
44	Hình 3.9: Lịch hợp lệ, nhưng không khả tuần tự	38
45	Hình 3.10: Đồ thị G không có chu trình	39
46	Hình 3.11: Giao tác có đồ thị G có chu trình	40

STT	Tên hình	Trang
47	Hình 3.12: Chu kỳ của khóa 2PL	41
48	Hình 3.13: Các giao tác thỏa 2PL	41
49	Hình 3.14: Các giao tác không thỏa 2PL	41
50	Hình 3.15: Khóa chét trong 2PL	42
51	Hình 3.16: Ma trận tương thích kỹ thuật khóa đọc viết	44
52	Hình 3.17: Ma trận tương thích kỹ thuật khóa cập nhật	45
53	Hình 3.18: Cây phân cấp dữ liệu	46
54	Hình 3.19: Cây phân cấp dữ liệu với trạng thái hiện hành của các nút được khóa	47
55	Hình 3.20: Ma trận tương thích kỹ thuật khóa đa hạt	47
56	Hình 3.21: Phương thức khóa các nút của T_i	47
57	Hình 3.22: Đọc dữ liệu quá trễ	51
58	Hình 3.23: Ghi dữ liệu quá trễ	52
59	Hình 3.24: Đọc dữ liệu rác	52
60	Hình 3.25: Qui tắc ghi Thomas	53
61	Hình 3.26: Giao tác ghi dữ liệu bị hủy	53
62	Hình 4.1: Nhật ký giao tác	70
63	Hình 4.2: Điểm lưu trữ đơn giản	72
64	Hình 4.3: Điểm lưu trữ linh động	72
65	Hình 4.4: Khôi phục theo phương pháp Undo-Logging	74
66	Hình 4.5: Undo-logging và checkpoint đơn giản	75
67	Hình 4.6: Undo-logging và checkpoint linh động	76
68	Hình 4.7: Khôi phục theo phương pháp Redo-Logging	79
69	Hình 4.8: Khôi phục theo phương pháp Undo/Redo-Logging	83

STT	Tên hình	Trang
70	Hình 5.1: Phân cấp thiết bị lưu trữ	89
71	Hình 5.2: Trình bày các đĩa được nối với một hệ thống máy tính	90
72	Hình 5.3: Mẫu tin có chiều dài cố định	101
73	Hình 5.4: Biểu diễn chuỗi byte của các mẫu tin độ dài thay đổi.	103
74	Hình 5.5: Phương pháp không gian lưu trữ	104
75	Hình 5.6a: Phương pháp con trỏ	105
76	Hình 5.6b: Tổ chức tập tin tuần tự	106
77	Hình 5.7: Chèn mẫu tin mới vào tập tin tuần tự	107
78	Hình 5.8: Tập tin tuần tự các mẫu tin account.	112
79	Hình 5.9: Chỉ mục nhiều mức	114
80	Hình 5.10: Chỉ mục thứ cấp	115
81	Hình 5.11: Cây chỉ mục B-Tree	117
82	Hình 5.12: Cấu trúc băm có thể mở rộng tổng quát	123
83	Hình 6.1: Các giai đoạn trong quá trình xử lý câu truy vấn	126
84	Hình 6.2: Cây phân tích	127
85	Hình 6.3: Cây phân tích cho câu truy vấn lồng	128
86	Hình 6.4: Cây phân tích cho câu truy vấn đơn	128
87	Hình 6.5: Cây truy vấn	129
88	Hình 6.6: Cây truy vấn được chuyển từ câu truy vấn	130
89	Hình 6.7a: Cây truy vấn cho câu truy vấn lồng	130
90	Hình 6.7b: Cây truy vấn lồng	131
91	Hình 6.8: Biến đổi phép chọn 2 biến	132
92	Hình 6.9a: Cây truy vấn	132
93	Hình 6.9b: Cây truy vấn	133

CHƯƠNG 1

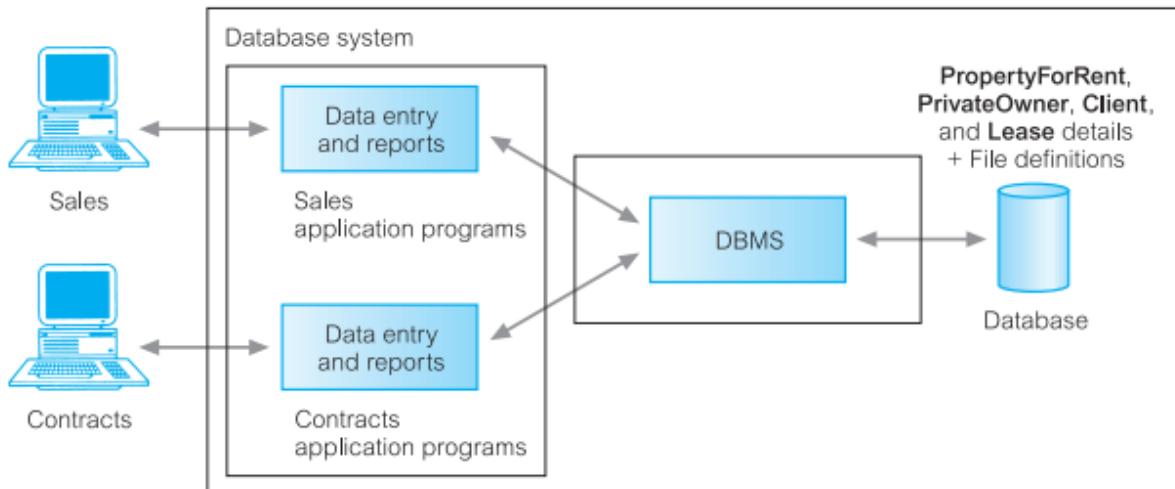
GIỚI THIỆU HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU (DATABASE MANAGEMENT SYSTEM)

❖ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

- Định nghĩa hệ quản trị cơ sở dữ liệu (HQTCSDL)
- Trình bày các chức năng, thành phần của một HQTCSDL
- Mô tả kiến trúc của HQTCSDL đa người dùng
- Xác định vai trò, nhiệm vụ của nhà quản trị CSDL

I.1 ĐỊNH NGHĨA HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Hệ quản trị cơ sở dữ liệu (**HQTCSDL**) là một hệ thống phần mềm cho phép người dùng định nghĩa, tạo và duy trì cơ sở dữ liệu (CSDL), đồng thời cung cấp dịch vụ truy cập đến CSDL này một cách có quản lý. Chúng ta cần lưu ý là HQTCSDL chỉ là một phần của hệ thống CSDL (database system). Như minh họa tại *Hình 1.1* bên dưới, ngoài HQTCSDL, một hệ thống CSDL còn bao gồm các CSDL và các chương trình ứng dụng phục vụ cho người dùng cuối trong hệ thống.



Hình 1.1: Một hệ thống cơ sở dữ liệu

Các HQTCSDL phổ biến hiện nay gồm: Oracle của tập đoàn Oracle, DB2 của IBM, MS SQL Server của MicroSofts. Ngoài ra còn có MS Access của MicroSofts và các HQTCSDL mã nguồn mở như MySQL, PostGreSQL,...

I.2 CHỨC NĂNG CỦA HQTCSDL

Một HQTCSDL có các chức năng sau:

I.2.1 Lưu trữ, truy xuất và cập nhật dữ liệu

Đây là chức năng cơ bản của một HQTCSDL. Người dùng có thể lưu trữ, truy xuất và cập nhật dữ liệu thông qua việc sử dụng các ngôn ngữ định (Data Definition Language – DDL), ngôn ngữ thao tác dữ liệu (Data Manipulation Language – DML) trong đó bao hàm ngôn ngữ truy vấn dữ liệu (Structured Query Language – SQL). HQTCSDL phải hỗ trợ khả năng thực hiện các chức năng này từ một máy tính truy cập từ xa vào CSDL qua mạng.

I.2.2 Danh mục hệ thống

Người dùng có thể truy cập vào danh mục chứa thông tin mô tả về các dữ liệu được lưu trữ trong CSDL gọi là danh mục hệ thống (system catalog) hay còn gọi là từ điển dữ liệu (data dictionary). Dữ liệu này còn được gọi là siêu dữ liệu (metadata) hay dữ liệu về dữ liệu, tùy theo HQTCSDL mà chúng có thể bao gồm:

- Tên, kiểu và kích cỡ của các mục dữ liệu
- Tên các mối quan hệ giữa các dữ liệu
- Ràng buộc toàn vẹn trên dữ liệu
- Tên người dùng được quyền truy cập vào dữ liệu
- Mục dữ liệu và kiểu truy cập mà mỗi (loại) người dùng có quyền
- Lược đồ ngoài, lược đồ quan niệm, lược đồ trong và ánh xạ giữa các lược đồ này.
- Các con số thống kê sử dụng như tần suất của giao tác, số lược truy cập đến một đối tượng trong CSDL.

I.2.3 An toàn dữ liệu

An toàn dữ liệu là sự bảo vệ CSDL khỏi những đe dọa có chủ ý hay không chủ ý thông qua các biện pháp có sử dụng máy tính hay không sử dụng máy tính. Việc xem xét an toàn không chỉ áp dụng cho dữ liệu trong CSDL mà còn bao gồm cả phần cứng, phần mềm và con người. Vì vậy các HQTCSDL phải cung cấp nhiều tiện ích để đảm bảo sự an toàn cho dữ liệu như ngăn chặn truy cập đối với người dùng không được phép

through qua việc quản lý và cấp quyền người dùng (user authorization) hoặc chỉ truy cập dữ liệu thông qua view được cấp quyền. Các quyền này sẽ được đề cập chi tiết hơn ở *chương 4*.

I.2.4 Toàn vẹn dữ liệu

Dữ liệu với ý nghĩa trong thực tế luôn phải thỏa mãn các điều kiện ràng buộc toàn vẹn nào đó về mặt giá trị, ví dụ như điểm thi phải nằm trong miền giá trị thực từ 0 đến 10,... Các ràng buộc này cũng góp phần duy trì một hệ CSDL an toàn bằng cách ngăn không cho dữ liệu thành không hợp lệ để tránh dẫn đến kết quả sai. Các HQTCSQL giúp duy trì tính nhất quán của dữ liệu thông qua việc hỗ trợ các phương tiện đảm bảo ràng buộc trên dữ liệu (data constraints); cơ chế quản lý giao tác (transaction management). Cơ chế quản lý giao tác sẽ được đề cập trong *chương 2*.

I.2.5 Điều khiển cạnh tranh

Trong tình huống CSDL có nhiều người dùng cùng lúc sẽ xảy ra tình huống cập nhật làm cho dữ liệu mất tính nhất quán. Các HQTCSQL phải cung cấp cơ chế đảm bảo CSDL được cập nhật đúng đắn khi có nhiều người dùng cùng lúc cập nhật một cơ sở dữ liệu. Các cơ chế điều khiển cạnh tranh sẽ được đề cập trong *chương 3*.

I.2.6 Phục hồi CSDL

Để đảm bảo tính an toàn cho hệ thống CSDL ngay cả khi có sự cố xảy ra, các HQTCSQL cần cung cấp cơ chế cho phép phục hồi CSDL về một trạng thái nhất quán sau sự cố làm CSDL bị hỏng theo bất cứ kiểu nào. Cơ chế về sao lưu và phục hồi sẽ được đề cập trong *chương 4*.

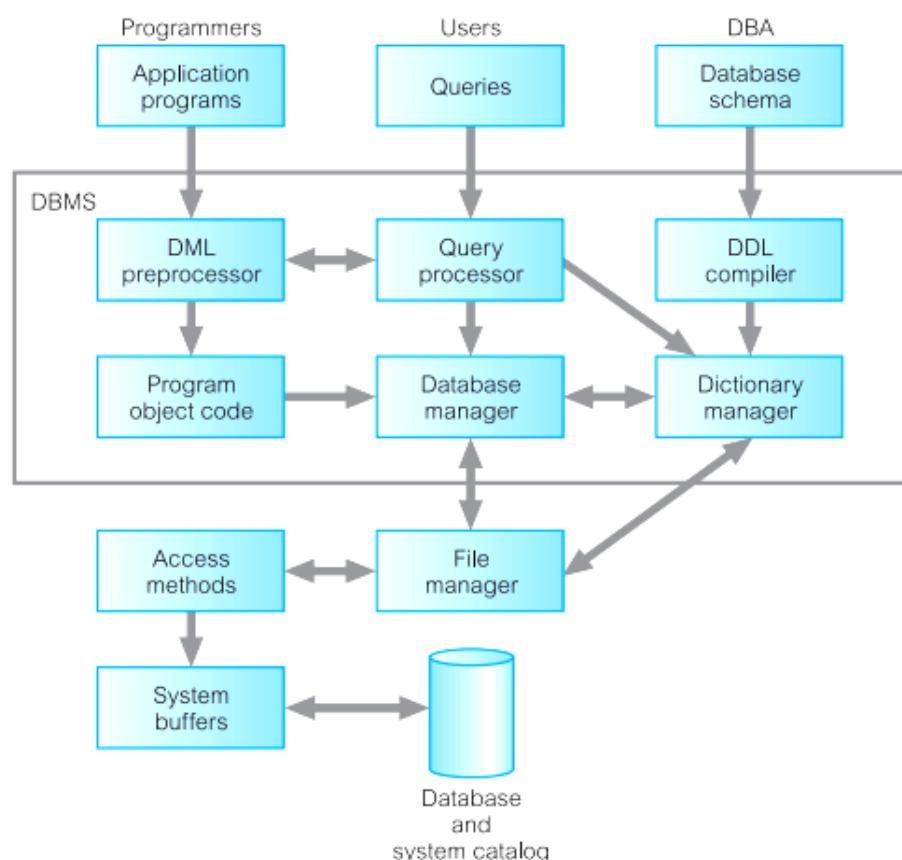
I.2.7 Các tiện ích khác

Ngoài các chức năng trên, HQTCSQL thường hỗ trợ thêm các tiện ích khác như sau:

- Nhập/xuất dữ liệu: cho phép tải dữ liệu từ các tập tin văn bản hay từ CSDL của các HQTCSQL khác và ngược lại.
- Giám sát: Giám sát việc sử dụng và thao tác trên cơ sở dữ liệu.
- Chương trình phân tích thống kê: dùng để khảo sát hiệu suất và thống kê việc sử dụng.
- Tạo chức chỉ mục.

I.3 CÁC THÀNH PHẦN CỦA MỘT HQTCSDL

Để đáp ứng được các chức năng nêu trên, các HQTCSDL là các phần mềm rất phức tạp và mỗi HQTCSDL đều có kiến trúc riêng nên khó đưa ra kiến trúc chung. *Hình 1.2* bên dưới mô tả kiến trúc của HQTCSDL được đề xuất bởi [Thomas 2005].



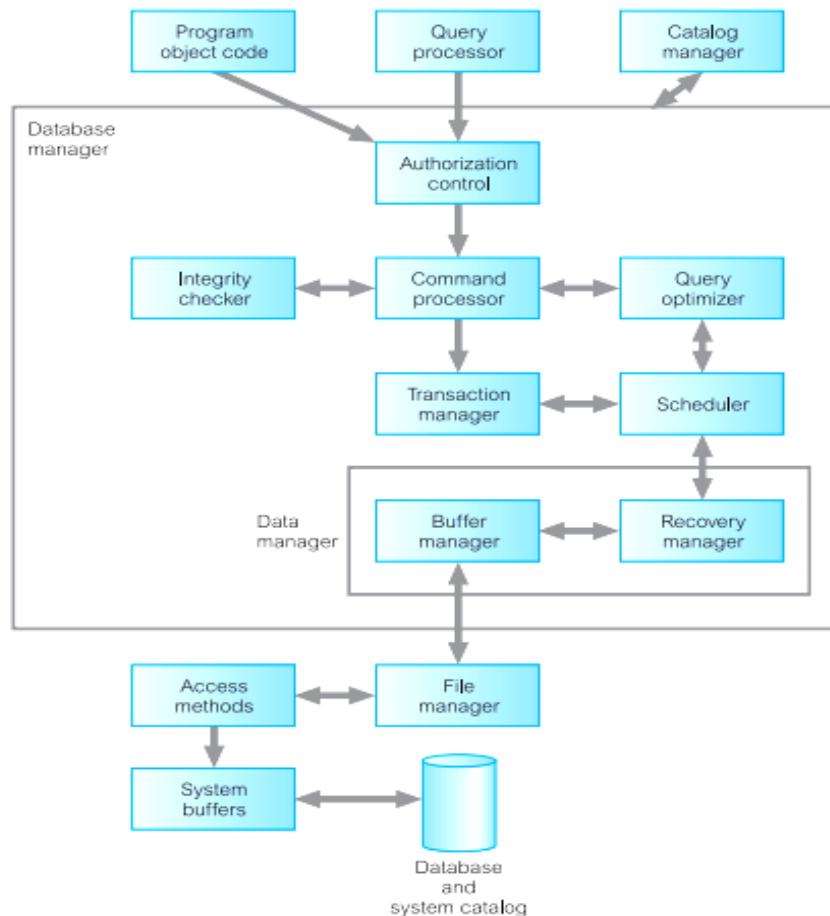
Hình 1.2: Các thành phần của một HQTCSDL [Thomas 2005]

Một HQCSLD có thể phân thành sáu thành phần chính như trong *Hình 1.2*. Mỗi thành phần đảm nhận một công việc cụ thể.

- **Bộ xử lý truy vấn (Query processor):** Đây là thành phần chủ yếu của một HQTCSDL để biến đổi các truy vấn thành một tập lệnh cấp thấp chuyển đến bộ quản lý CSDL.
- **Bộ quản lý CSDL (Database Manager):** nhận vào các mã đối tượng chương trình và các truy vấn, đổi chiểu với lược đồ ngoài và lược đồ quan niệm để xác định các mẫu tin quan niệm cần thiết để trả lời truy vấn. Bộ phận này được chia nhỏ thành các thành phần nhỏ hơn như trong *Hình 1.3*.

- **Bộ tiền xử lý ngôn ngữ DML (DML preprocessor):** Bộ phận này biến đổi các lệnh thao tác dữ liệu như insert, delete, update,... thành các lời gọi hàm chuẩn. bộ phận này phải tương tác với bộ xử lý truy vấn để sinh mã phù hợp.
- **Trình biên dịch (DDL compiler):** chuyển các lệnh định nghĩa dữ liệu thành một tập các bảng chứa siêu dữ liệu. Các bảng này sau đó được lưu trong từ điển hệ thống.
- **Bộ quản lý từ điển (Dictionary manager):** quản lý sự truy cập và duy trì từ điển hệ thống.

Các thành phần chính của bộ quản lý CSDL như *Hình 1.3*, gồm:



Hình 1.3: Các thành phần của bộ quản lý CSDL

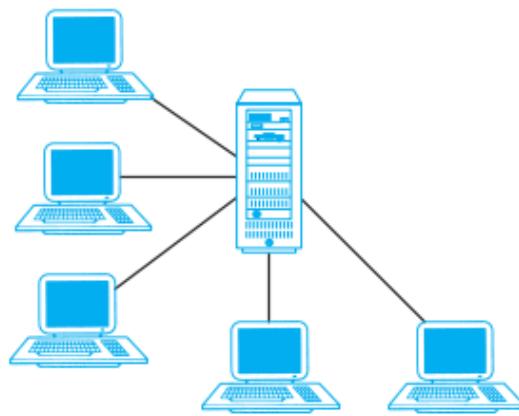
- **Điều khiển quyền (Authorization control):** kiểm tra xem người dùng có đủ các quyền cần thiết để thực hiện thao tác yêu cầu không.
- **Bộ xử lý lệnh (Command processor):** xử lý câu lệnh sau khi xác nhận là đủ quyền

- **Bộ kiểm tra toàn vẹn (Integrity checker):** đối với các thao tác làm thay đổi CSDL, bộ phận này sẽ kiểm tra xem thao tác này có thỏa mãn tất cả các ràng buộc toàn vẹn hay không.
- **Bộ tối ưu truy vấn (Query Optimizer):** xác định một chiến lược tối ưu để truy vấn.
- **Bộ quản lý giao tác (Transaction manager):** thực hiện các xử lý cần thiết cho các thao tác nhận được từ các giao tác.
- **Bộ lập lịch (Scheduler):** Bộ phận này có trách nhiệm đảm bảo các thao tác cạnh tranh trên CSDL được thực hiện mà không gây ra mâu thuẫn. Nó điều khiển trật tự thực thi tương đối của các thao tác trong từng giao tác.

I.4. KIẾN TRÚC HỆ CSDL ĐA NGƯỜI DÙNG

I.4.1 Kiến trúc xử lý từ xa

Đây là một kiến trúc truyền thống. Một máy tính rất mạnh (mainframe) kết nối với nhiều máy đầu cuối (terminal) như minh họa tại [Hình 1.4](#). Toàn bộ tài (lưu trữ dữ liệu, HQTCSDL và ứng dụng) đều thực hiện trên mainframe. Một xu hướng được nhìn nhận hiện nay là thay thế mainframe đặt tiền bằng một tập hợp các máy tính cá nhân nối mạng nhằm đạt được khả năng tính toán mạnh như mainframe.

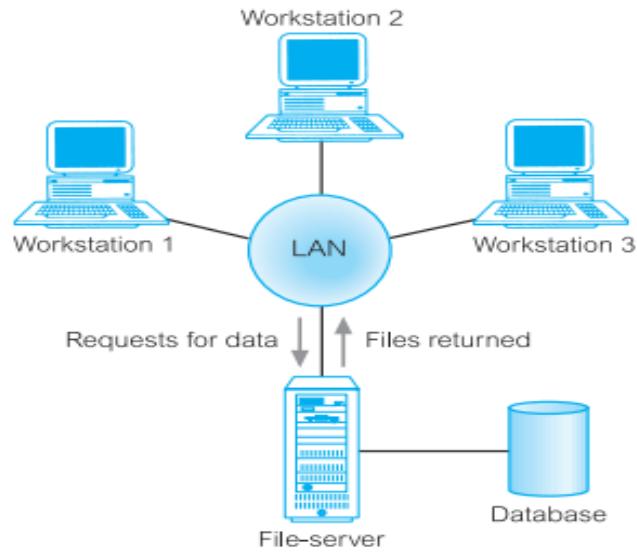


Hình 1.4: Kiến trúc xử lý từ xa

I.4.2. Kiến trúc máy chủ tập tin (Tập tin-server)

CSDL sẽ được lưu trên một máy chủ, HQTCSDL sẽ được cài đặt trên các máy trạm (workstation) nối kết với máy chủ thông qua mạng cục bộ như minh họa tại [Hình 1.5](#). Kiến trúc này giúp phân chia tải trên nhiều máy. Tuy nhiên có khuyết điểm là ánh

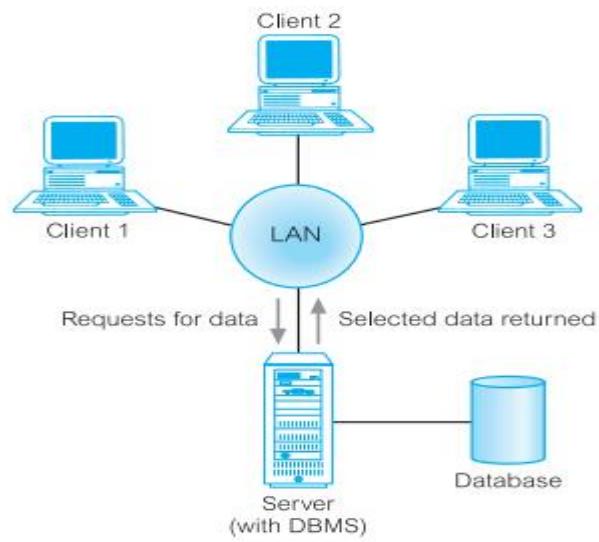
hướng đến lưu thông mạng, mỗi máy trạm điều phải cài HQTCSQL nên khả năng quản lý cạnh tranh, phục hồi và toàn vẹn dữ liệu phức tạp hơn.



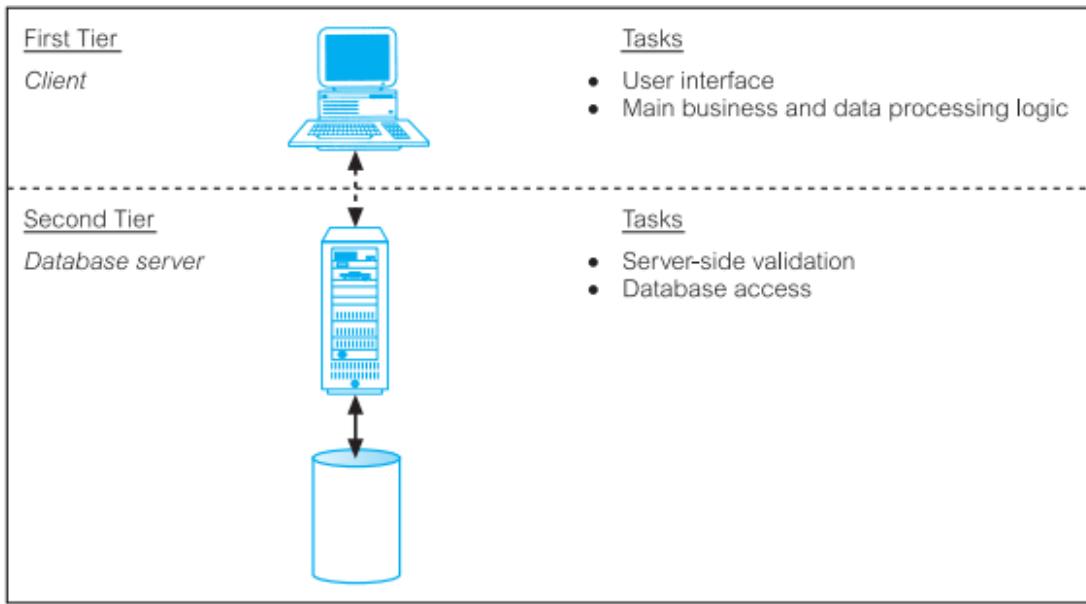
Hình 1.5: Kiến trúc máy chủ tập tin

I.4.3. Kiến trúc Client-server hai tầng

Kiến trúc này gồm hai tầng: tầng client sẽ cài và chạy chương trình ứng dụng, tầng server cài HQTCSQL và CSDL như minh họa tại [Hình 1.6](#). Thể hiện cụ thể sự phân tách giữa hai tầng tại [Hình 1.7](#)



Hình 1.6: Kiến trúc Client-Server

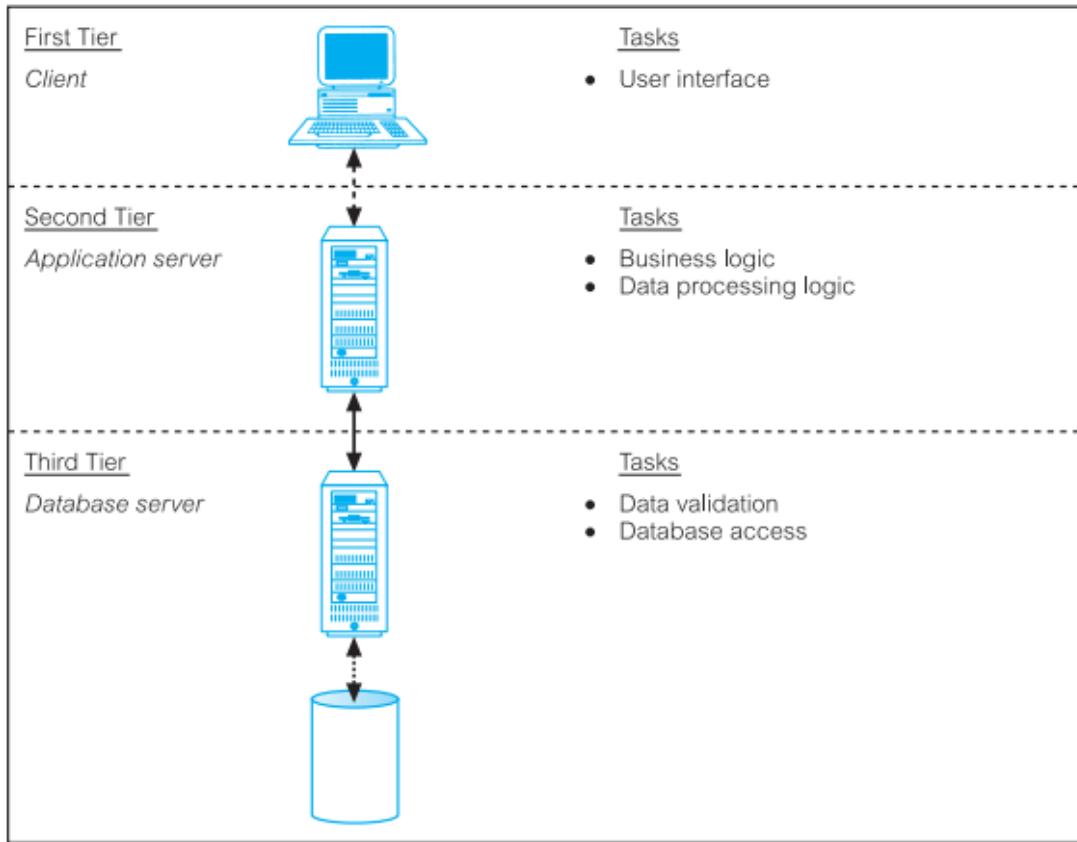


Hình 1.7: Kiến trúc Client-server hai tầng

Nhờ CSDL được quản lý tập trung bởi một HQTCSDL tại server nên kiến trúc này có ưu điểm là khả năng truy cập đến CSDL được mở rộng hơn, tăng hiệu quả hoạt động, có thể giảm chi phí phần cứng, giảm chi phí truyền thông và tính nhán quán được bảo đảm. Tuy nhiên phía client này sinh hai khó khăn khi chương trình ứng dụng trở nên lớn. Một là đòi hỏi nhiều tài nguyên hơn trên máy client để có thể chạy ứng dụng một cách hiệu quả, hai là đòi hỏi nhiều công việc quản trị phía client.

I.4.4. Kiến trúc Client-server ba tầng

Đây là kiến trúc phát triển từ kiến trúc Client-server 2 tầng. Trong đó, chương trình ứng dụng sẽ không còn chạy ở các máy client mà sẽ được cài đặt ở một server nằm ở tầng giữa, gọi là máy chủ ứng dụng như minh họa tại [Hình 1.8](#). Ưu điểm của kiến trúc này là client “nhỏ” chỉ làm nhiệm vụ giao diện nên không cần phần cứng đắt tiền; việc bảo trì ứng dụng diễn ra tập trung tại máy chủ ứng dụng; dễ thay đổi hoặc thay thế một tầng mà không làm ảnh hưởng đến tầng khác, dễ dàng cân bằng tải thông qua việc gia tăng số lượng máy chủ ứng dụng giữa các tầng. Kiến trúc này ánh xạ khá tự nhiên vào môi trường Web. Đây là kiến trúc được sử dụng khá phổ biến hiện nay.



Hình 1.8: Kiến trúc Client-server ba tầng

I.5 NHÀ QUẢN TRỊ CSDL

I.5.1 Vai trò, nhiệm vụ của nhà quản trị CSDL

Nhà quản trị CSDL (Database administrator - DBA) chịu trách nhiệm về việc đảm bảo cho sự hoạt động thông suốt của các chức năng, tính hiệu quả của các CSDL và ứng dụng truy cập vào các CSDL này của tổ chức. Khi tổ chức cần phát triển một ứng dụng mới, DBA với vai trò là người am hiểu nhất về dữ liệu của tổ chức, tham gia vào tất cả các giai đoạn của đề án từ đặc tả yêu cầu, phân tích, thiết kế, cài đặt và kiểm thử ứng dụng nhằm đảm bảo ứng dụng có được sự duy trì chính xác, hiệu quả đến dữ liệu của tổ chức. Khi ứng dụng chuẩn bị đưa vào sử dụng DBA phải đảm bảo rằng HQTCSL đã được sẵn sàng cho tải mới. Điều này bao gồm cả cài đặt các biện pháp an ninh, đo đạc, điều chỉnh nhu cầu về lưu trữ và bộ nhớ của ứng dụng mới, đồng thời dự đoán sự ảnh hưởng của tải mới này đối với CSDL và ứng dụng trong hệ thống đang vận hành.

DBA cũng có trách nhiệm đưa dữ liệu mới từ môi trường thử nghiệm vào hoạt động. Khi ứng dụng đã được đưa vào sử dụng, DBA phải đảm bảo tính sẵn sàng, an

ninh, toàn vẹn của hệ thống; giám sát hiệu suất và điều chỉnh hệ thống, sao lưu và phục hồi hệ thống. Cuối cùng khi ứng dụng không còn được sử dụng nữa, DBA phải giúp xác định tình trạng cuối cùng của dữ liệu sử dụng bởi ứng dụng để liệu rằng dữ liệu có thể được sử dụng cho ứng khác hay cần phải lưu trữ theo một quy định nào đó,...

Các công việc cơ bản mà một DBA đảm trách trong quá trình CSDL đã được đưa vào sử dụng gồm:

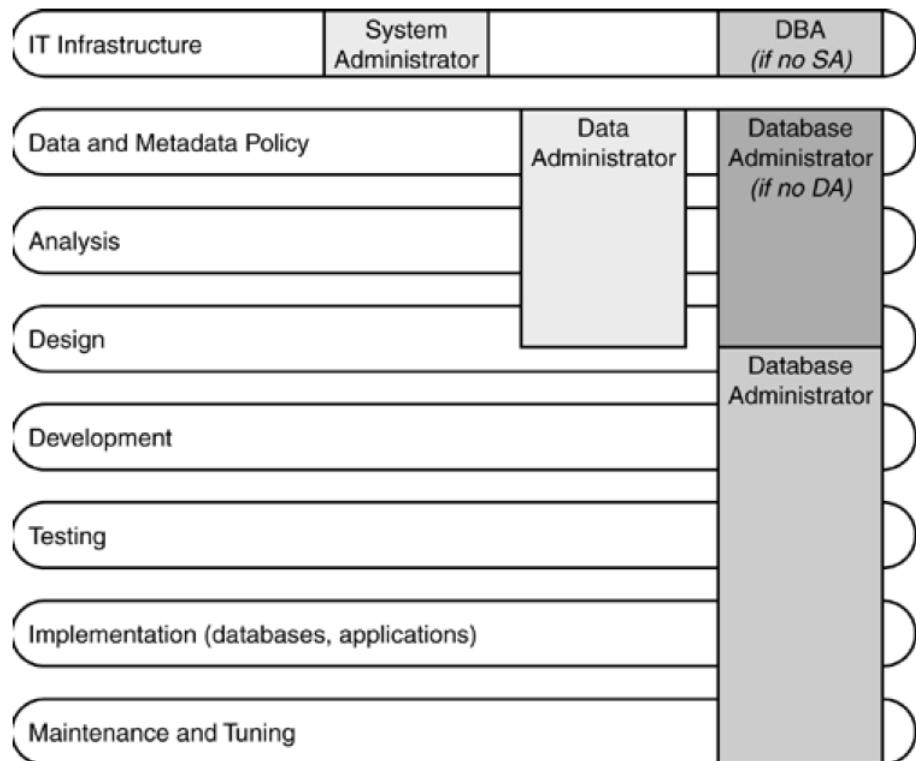
- Các thao tác cơ bản trên CSDL: Kết nối, quản lý CSDL, nhập/xuất CSDL
- Cài đặt các biện pháp bảo vệ CSDL: quản lý, cấp quyền người dùng
- Đảm bảo tính sẵn sàng của hệ thống: sao lưu, phục hồi
- Đảo bảo tính toàn vẹn dữ liệu: Cài đặt các ràng buộc trên CSDL

I.5.2 Các tiêu chí đòi hỏi ở một DBA

Để có thể thực hiện tốt các nhiệm vụ trên, một DBA không những có kiến thức tốt về các công nghệ CSDL mà về tất cả những gì kết nối đến CSDL. DBA cũng cần có các kỹ năng giao tiếp đặc biệt để có thể hợp tác tốt với các nhân viên khác nhau từ kỹ thuật viên, lập trình viên, người dùng cuối, khách hàng đến các nhà lãnh đạo. DBA còn là người thích đương đầu với thử thách, với các vấn đề mới và giải quyết các sự cố. Ngoài các yêu cầu về chuyên môn kỹ thuật. DBA cũng cần có kỹ năng quản lý để hoàn thành tốt nhiệm vụ. DBA không nên thụ động chỉ giải quyết các vấn đề sau khi nó xảy ra, mà còn chủ động phát triển và cài đặt một kế hoạch thực hiện chiến lược để triển khai CSDL trong tổ chức.

I.5.3 Các vai trò quản trị khác nhau

Trong các tổ chức với các hệ thống thông tin lớn, người ta có thể thuê nhiều quản trị viên với các vai trò quản trị khác nhau, bao gồm: quản trị CSDL (DBA), quản trị dữ liệu (DA) và quản trị hệ thống (SA) như trong **Hình 1.9**



Hình 1.9: Trách nhiệm của DA, DBA và SA

Trong [Hình 1.9](#) cho thấy trong khi SA phụ trách về hạ tầng CNTT, DA liên quan nhiều đến quá trình phân tích, thiết kế, đề xuất các chính sách cho dữ liệu và siêu dữ liệu, còn DBA liên quan nhiều đến các giai đoạn phát triển, kiểm thử, cài đặt, duy trì hệ thống và điều chỉnh các hoạt động hơn.

Ngoài ra, một số tổ chức với các hệ thống cực lớn cũng có các DBA chuyên trách cho công việc như: DBA phụ trách sao lưu phục hồi, DBA phụ trách phân tích hiệu suất hệ thống, DBA phụ trách kho dữ liệu (Data warehousse). Đây là các chuyên gia dày kinh nghiệm và có thể đảm trách các công việc rất quan trọng.

I.6 QUÁ TRÌNH PHÁT TRIỂN CỦA HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU.

Quá trình phát triển của DBMS như sau:

- Flat files: 1960s – 1980s
- Hierarchical: 1970s – 1990s
- Network : 1970s – 1990s
- Relational: 1980s – đến nay
- Object-oriented: 1990s – đến nay

- Object-relational: 1990s – đến nay
- Data warehousing: 1980s – đến nay
- Web-enabled: 1990s – đến nay

❖ **Câu hỏi (bài tập) củng cố:**

1. Định nghĩa DBMS
2. Trình bày các chức năng của một HQTCSDL
3. Trình bày các thành phần của một HQTCSDL
4. Trình bày các thành phần và chức năng của bộ quản lý CSDL
5. Kể tên các kiểu kiến trúc của HQTCSDL đa người dùng
6. Kể tên các tầng trong kiến trúc Client-Server ba tầng
7. Trình bày chức năng của mỗi tầng trong kiến trúc Client-Server ba tầng
8. Trình bày các công việc của DBA, DA
9. Trình bày các vai trò quản trị khác nhau trong hệ thống

CHƯƠNG 2

GIAO TÁC (TRANSACTION)

❖ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

- Trình bày các tính chất của giao tác
- Nêu các trạng thái của giao tác
- Mô tả sơ đồ trạng thái các giao tác
- Kiểm tra lịch khả tuần tự
- Phân loại lịch giao tác

II.1 GIAO TÁC

II.1.1 Định nghĩa giao tác

Giao tác là một hành động hay một chuỗi các hành động được thực hiện bởi một người dùng hoặc một chương trình ứng dụng, trong đó có truy cập hoặc làm thay đổi nội dung của một CSDL. **Một giao tác là một đơn vị luận lý** của công việc trên CSDL. Nó có thể là toàn bộ chương trình, một phần của chương trình hoặc một lệnh đơn lẻ như INSERT hay UPDATE và nó có thể bao gồm nhiều thao tác trên CSDL.

II.1.2 Các tính chất của giao tác

Để đảm bảo tính toàn vẹn của dữ liệu, ta yêu cầu hệ CSDL duy trì các tính chất sau của giao tác:

- **Tính nguyên tử (Atomicity):** Hoặc toàn bộ các hoạt động của giao tác được phản ánh đúng đắn trong CSDL hoặc không có gì cả.
- **Tính nhất quán (consistency):** Sự thực hiện của một giao tác là cô lập (Không có giao tác khác thực hiện đồng thời) để bảo tồn tính nhất quán của CSDL.
- **Tính cô lập (Isolation):** Cho dù nhiều giao tác có thể thực hiện đồng thời, hệ thống phải đảm bảo rằng đối với mỗi cặp giao tác T_i, T_j , hoặc T_j kết thúc thực hiện trước khi T_i khởi động hoặc T_j bắt đầu sự thực hiện sau khi T_i kết thúc. Như vậy mỗi giao tác không cần biết đến các giao tác khác đang thực hiện đồng thời trong hệ thống.
- **Tính bền vững (Durability):** Sau một giao tác hoàn thành thành công, các thay đổi đã được tạo ra đối với CSDL vẫn còn ngay cả khi xảy ra sự cố hệ thống.

Ví dụ:

Ta xét một hệ thống nhà băng gồm một số tài khoản và một tập các giao tác truy xuất và cập nhật các tài khoản. Tại thời điểm hiện tại, ta giả thiết rằng CSDL nằm trên đĩa, nhưng một vài phần của nó đang nằm tạm thời trong bộ nhớ. Các truy xuất CSDL được thực hiện bởi hai hoạt động sau:

- **READ(X):** chuyển đơn vị dữ liệu X từ CSDL đến bộ đệm (buffer) của giao tác thực hiện hoạt động READ này.
- **WRITE(X):** chuyển đơn vị dữ liệu X từ bộ đệm của giao tác thực hiện WRITE đến CSDL.

Trong hệ CSDL thực, hoạt động WRITE không nhất thiết dẫn đến sự cập nhật trực tiếp dữ liệu trên đĩa; hoạt động WRITE có thể được lưu tạm thời trong bộ nhớ và được thực hiện trên đĩa muộn hơn. Trong ví dụ, ta giả thiết hoạt động WRITE cập nhật trực tiếp CSDL.

T_i là một giao tác chuyển 50\$ từ tài khoản A sang tài khoản B. Giao tác này có thể được xác định như sau:

T_i	READ (A)
	A:=A-50
	WRITE(A)
	READ(B)
	B:=B+50
	WRITE(B)

Hình 2.1: Giao tác chuyển tiền từ tài khoản A sang tài khoản B

Ta xem xét mỗi một trong các yêu cầu ACID:

Tính nhất quán: Đòi hỏi nhất quán ở đây là tổng của A và B là không thay đổi bởi sự thực hiện giao tác. Nếu không có yêu cầu nhất quán, tiền có thể được tạo ra hay bị phá huỷ bởi giao tác. Để dàng kiểm nghiệm rằng nếu CSDL nhất quán trước một thực hiện giao tác, nó vẫn nhất quán sau khi thực hiện giao tác. Đảm bảo tính nhất quán cho một giao tác là trách nhiệm của người lập trình ứng dụng người đã viết ra giao tác.

Nhiệm vụ này có thể được làm cho dễ dàng bởi kiểm thử tự động các ràng buộc toàn vẹn.

Tính nguyên tử: Giả sử rằng ngay trước khi thực hiện giao tác T_i , giá trị của các tài khoản A và B tương ứng là 1000 và 2000. Giả sử rằng trong khi thực hiện giao tác T_i , một sự cố xảy ra cản trở T_i hoàn tất thành công sự thực hiện của nó. Ta cũng giả sử rằng sự cố xảy ra sau khi hoạt động WRITE(A) đã được thực hiện, nhưng trước khi hoạt động WRITE(B) được thực hiện. Trong trường hợp này giá trị của tài khoản A và B là 950\$ và 2000\$. Ta đã phá huỷ 50\$. Tổng A+B không còn được bảo tồn.

Như vậy, kết quả của sự cố là trạng thái của hệ thống không còn phản ánh trạng thái của thế giới mà CSDL được giả thiết nắm giữ. Ta sẽ gọi trạng thái như vậy là trạng thái không nhất quán. Ta phải đảm bảo rằng tính không nhất quán này không xuất hiện trong một hệ CSDL. Chú ý rằng, cho dù thế nào tại một vài thời điểm, hệ thống cũng phải ở trong trạng thái không nhất quán. Ngay cả khi giao tác T_i , trong quá trình thực hiện cũng tồn tại thời điểm tại đó giá trị của tài khoản A là 950\$ và tài khoản B là 2000\$ - một trạng thái không nhất quán. Trạng thái này được thay thế bởi trạng thái nhất quán khi giao tác đã hoàn tất. Như vậy, nếu giao tác không bao giờ khởi động hoặc được đảm bảo sẽ hoàn tất, trạng thái không nhất quán sẽ không bao giờ xảy ra. Đó chính là lý do có yêu cầu về tính nguyên tử: Nếu tính chất nguyên tử được cung cấp, tất cả các hành động của giao tác được phản ánh trong CSDL hoặc không có gì cả. ý tưởng cơ sở để đảm bảo tính nguyên tử là như sau: hệ CSDL lưu vết (trên đĩa) các giá trị cũ của bất kỳ dữ liệu nào trên đó giao tác đang thực hiện viết, nếu giao tác không hoàn tất, giá trị cũ được khôi phục để đặt trạng thái của hệ thống trở lại trạng thái trước khi giao tác diễn ra. Đảm bảo tính nguyên tử là trách nhiệm của hệ CSDL, và được quản lý bởi một thành phần được gọi là thành phần quản trị giao tác (transaction-management component).

Tính bền vững: Tính chất bền vững đảm bảo rằng mỗi khi một giao tác hoàn tất, tất cả các cập nhật đã thực hiện trên cơ sở dữ liệu vẫn còn đó, ngay cả khi xảy ra sự cố hệ thống sau khi giao tác đã hoàn tất. Ta giả sử một sự cố hệ thống có thể gây ra việc mất dữ liệu trong bộ nhớ chính, nhưng dữ liệu trên đĩa thì không mất. Có thể đảm bảo tính bền vững bởi việc đảm bảo hoặc các cập nhật được thực hiện bởi giao tác đã được viết lên đĩa trước khi giao tác kết thúc hoặc thông tin về sự cập nhật được thực hiện bởi giao tác và được viết lên đĩa đủ cho phép CSDL xây dựng lại các cập nhật khi hệ CSDL được khởi động lại sau sự cố. Đảm bảo tính bền vững là trách nhiệm của một

thành phần của hệ CSDL được gọi là thành phần quản trị phục hồi (recovery-management component). Hai thành phần quản trị giao tác và quản trị phục hồi quan hệ mật thiết với nhau.

Tính cô lập: Ngay cả khi tính nhất quán và tính nguyên tử được đảm bảo cho mỗi giao tác, trạng thái không nhất quán vẫn có thể xảy ra nếu trong hệ thống có một số giao tác được thực hiện đồng thời và các hoạt động của chúng đan xen theo một cách không mong muốn. Ví dụ, CSDL là không nhất quán tạm thời trong khi giao tác chuyển khoản từ A sang B đang thực hiện, nếu một giao tác khác thực hiện đồng thời đọc A và B tại thời điểm trung gian này và tính A+B, nó đã tham khảo một giá trị không nhất quán, sau đó nó thực hiện cập nhật A và B dựa trên các giá trị không nhất quán này, như vậy CSDL có thể ở trạng thái không nhất quán ngay cả khi cả hai giao tác hoàn tất thành công. Một giải pháp cho vấn đề các giao tác thực hiện đồng thời là thực hiện tuần tự các giao tác, tuy nhiên giải pháp này làm giảm hiệu năng của hệ thống. Các giải pháp khác cho phép nhiều giao tác thực hiện cạnh tranh đã được phát triển ta sẽ thảo luận về chúng sau này. Tính cô lập của một giao tác đảm bảo rằng sự thực hiện đồng thời các giao tác dẫn đến một trạng thái hệ thống tương đương với một trạng thái có thể nhận được bởi thực hiện các giao tác này một tại một thời điểm theo một thứ nào đó. Đảm bảo tính cô lập là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị cạnh tranh (Concurrency-Control Component).

II.1.3 Các trạng thái của giao tác

Nếu không có sự cố, tất cả các giao tác đều hoàn tất thành công. Tuy nhiên, một giao tác trong thực tế có thể không thể hoàn tất sự thực hiện của nó. Giao tác như vậy được gọi là bị bỏ dở. Nếu ta đảm bảo được tính nguyên tử, một giao tác bị bỏ dở không được phép làm ảnh hưởng tới trạng thái của CSDL. Như vậy, bất kỳ thay đổi nào mà giao tác bị bỏ dở này phải bị huỷ bỏ. Mỗi khi các thay đổi do giao tác bị bỏ dở bị huỷ bỏ, ta nói rằng giao tác bị quay lui (rolled back). Việc này là trách nhiệm của bộ quản lý khôi phục nhằm quản trị các giao tác bị bỏ dở. Một giao tác hoàn tất thành công sự thực hiện của nó được gọi là được bàn giao (committed). Một giao tác được bàn giao (committed), thực hiện các cập nhật sẽ biến đổi CSDL sang một trạng thái nhất quán mới và nó là bền vững ngay cả khi có sự cố. Mỗi khi một giao tác là được bàn giao (committed), ta không thể huỷ bỏ các hiệu quả của nó bằng cách bỏ dở nó. Cách duy nhất để huỷ bỏ các hiệu quả của một giao tác được bàn giao (committed) là thực hiện một giao tác bù (compensating transaction); nhưng không phải luôn luôn có thể tạo ra

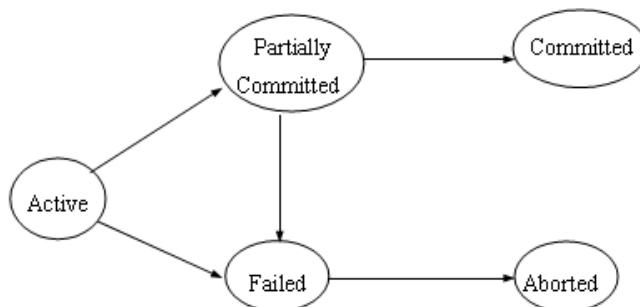
một giao tác bù. Do vậy trách nhiệm viết và thực hiện một giao tác bù thuộc về người sử dụng và không được quản lý bởi hệ CSDL.

Một giao tác phải ở trong một trong các trạng thái sau:

- **Hoạt động (Active):** Trạng thái khởi đầu, giao tác giữ trong trạng thái này trong khi nó đang thực hiện.
- **Được bàn giao bộ phận (Partially Committed):** Sau khi lệnh cuối cùng được thực hiện.
- **Thất bại (Failed):** Sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- **Bỏ dở (Aborted):** Sau khi giao tác đã bị quay lui và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao tác.
- **Được bàn giao (Committed):** Sau khi hoàn thành thành công giao tác.

Ta nói một giao tác đã được bàn giao (Committed) chỉ nếu nó đã đi vào trạng thái Committed, tương tự, một giao tác bị bỏ dở nếu nó đã đi vào trạng thái Aborted. Một giao tác được gọi là kết thúc nếu nó hoặc là Committed hoặc là Aborted. Một giao tác khởi đầu bởi trạng thái Active. Khi nó kết thúc lệnh sau cùng của nó, nó chuyển sang trạng thái Partially Committed. Tại thời điểm này, giao tác đã hoàn thành sự thực hiện của nó, nhưng nó vẫn có thể bị bỏ dở do đâu ra hiện tại vẫn có thể trú tạm thời trong bộ nhớ chính và như thế một sự cố phần cứng vẫn có thể ngăn cản sự hoàn tất của giao tác. Hệ CSDL khi đó đã kịp viết lên đĩa đầy đủ thông tin giúp việc tái tạo các cập nhật đã được thực hiện trong quá trình thực hiện giao tác, khi hệ thống tái khởi động sau sự cố. Sau khi các thông tin sau cùng này được viết lên đĩa, giao tác chuyển sang trạng thái Committed.

Sơ đồ trạng thái tương ứng với một giao tác như sau:



Hình 2.2: Sơ đồ trạng thái của giao tác

Với giả thiết sự cố hệ thống không gây ra sự mất dữ liệu trên đĩa. Một giao tác đi vào trạng thái Failed sau khi hệ thống xác định rằng giao tác không thể tiến triển bình thường được nữa (do lỗi phần cứng hoặc phần mềm). Như vậy, giao tác phải được quay lui rồi chuyển sang trạng thái bỏ dỡ. Tại điểm này, hệ thống có hai lựa chọn:

♦ **Khởi động lại giao tác:** nhưng chỉ nếu giao tác bị bỏ dỡ là do lỗi phần cứng hoặc phần mềm nào đó không liên quan đến logic bên trong của giao tác. Giao tác được khởi động lại được xem là một giao tác mới.

♦ **Giết giao tác:** thường được tiến hành hoặc do lỗi logic bên trong giao tác, lỗi này cần được chỉnh sửa bởi viết lại chương trình ứng dụng hoặc do đầu vào xấu hoặc do dữ liệu mong muốn không tìm thấy trong CSDL.

II.2 LỊCH THAO TÁC (SCHEDULE)

II.2.1 Điều khiển cạnh tranh

Hệ thống xử lý giao tác thường cho phép nhiều giao tác thực hiện đồng thời. Việc cho phép nhiều giao tác cập nhật dữ liệu đồng thời gây ra những khó khăn trong việc bảo đảm sự nhất quán dữ liệu. Bảo đảm sự nhất quán dữ liệu mà không quan tâm tới sự thực hiện cạnh tranh các giao tác sẽ cần thêm các công việc phụ. Một phương pháp dễ tiến hành là cho các giao tác thực hiện tuần tự: *đảm bảo rằng một giao tác khởi động chỉ sau khi giao tác trước đã hoàn tất*. Tuy nhiên có hai lý do hợp lý để thực hiện cạnh tranh là:

♦ Một giao tác gồm nhiều bước. Một vài bước liên quan tới hoạt động đọc/viết; các bước khác liên quan đến hoạt động CPU. CPU và các đĩa trong một hệ thống có thể hoạt động song song. Sự song song của hệ thống CPU và đọc/viết có thể được khai thác để chạy nhiều giao tác song song. Trong khi một giao tác tiến hành một hoạt động đọc/viết trên một đĩa, một giao tác khác có thể đang chạy trong CPU, một giao tác thứ ba có thể thực hiện đọc/viết trên một đĩa khác ... như vậy sẽ tăng lượng đầu vào hệ thống có nghĩa là tăng số lượng giao tác có thể được thực hiện trong một lượng thời gian đã cho, cũng có nghĩa là hiệu suất sử dụng bộ xử lý và đĩa tăng lên.

♦ Có thể có sự thực hiện đồng thời các giao tác đang chạy trong hệ thống, có giao tác ngắn, có giao tác dài. Nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến trước hoàn tất, điều đó dẫn đến một sự trì hoãn không lường trước được trong việc thực hiện một giao tác. Nếu các giao tác đang hoạt động trên các phần khác nhau của CSDL, sẽ tốt hơn nếu ta cho chúng thực hiện đồng thời, chia sẻ các

chu kỳ CPU và truy xuất đĩa giữa chúng. Thực hiện cạnh tranh làm giảm sự trì hoãn không lường trước trong việc chạy các giao tác, đồng thời làm giảm thời gian đáp ứng trung bình (thời gian để một giao tác được hoàn tất sau khi đã được đệ trình).

Động cơ để sử dụng thực hiện cạnh tranh trong CSDL cũng giống như động cơ để thực hiện đa chương trong hệ điều hành. Khi một vài giao tác thực hiện đồng thời, tính nhất quán CSDL có thể bị phá huỷ cho dù mỗi giao tác là đúng. Một giải pháp để giải quyết vấn đề này là sử dụng định thời. Hệ CSDL phải điều khiển sự trao đổi giữa các giao tác cạnh tranh để ngăn ngừa chúng phá hủy sự nhất quán của CSDL. Các cơ chế cho điều đó được gọi là sơ đồ điều khiển cạnh tranh (Concurrency-control Scheme).

Ví dụ: cho lịch trình như [Hình 2.3](#)

Giả sử ràng buộc nhất quán trên CSDL là $A=B$

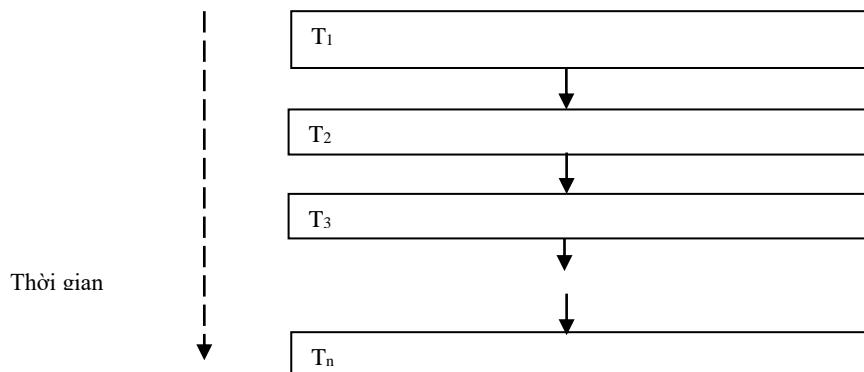
Từng giao tác thực hiện riêng lẻ thì tính nhất quán sẽ được bảo toàn

T_1	T_2
Read(A,t)	Read(A,s)
$t:=t+100$	$s:=s^2$
Write(A,t)	Write(A,s)
Read(B,t)	Read(B,s)
$t:=t+100$	$s:=s^2$
Write(B,t)	Write(B,s)

Hình 2.3: Lịch giao tác

II.2.2 Lịch tuần tự (Serial schedule)

Một lịch S được gọi là tuần tự nếu các hành động của các giao tác T_i ($i=1..n$) được thực hiện liên tiếp nhau được minh họa như [Hình 2.4](#)



Hình 2.4: Lịch tuần tự

Ví dụ:

Lịch S_1 và S_2 là hai lịch tuần tự

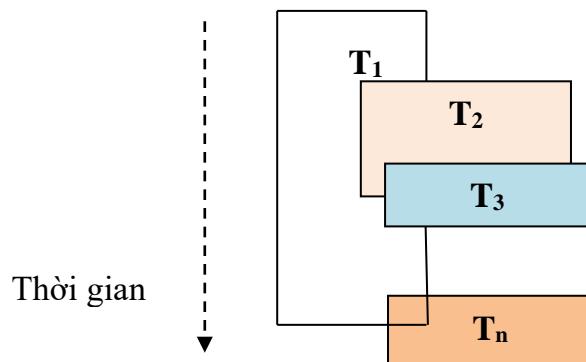
S₁	T₁	T₂	A	B
			25	25
Read(A,t)				
t:=t+100				
Write(A,t)		125		
Read(B,t)				
t:=t+100				
Write(B,t)			125	
	Read(A,s)			
	s:=s*2			
	Write(A,s)	250		
	Read(B,s)			
	s:=s*2			
	Write(B,s)		250	

S₂	T₁	T₂	A	B
			25	25
	Read(A,s)			
	s:=s*2			
	Write(A,s)	50		
	Read(B,s)			
	s:=s*2			
	Write(B,s)			50
	Read(A,t)			
	t:=t+100			
	Write(A,t)			150
	Read(B,t)			
	t:=t+100			
	Write(B,t)			150

Hình 2.5: Lịch tuần tự

II.2.3 Lịch khả tuần tự (Serializable schedule)

Một lịch S được lập từ n giao tác T_1, T_2, \dots, T_n xử lý đồng thời được gọi là khả tuần tự nếu nó cho cùng kết quả với 1 lịch tuần tự nào đó được lập từ n giao tác này.



Hình 2.6 : Lịch khả tuần tự

Ví dụ:

Cho lịch S₃, S₄ và S₅ như sau:

S3	T ₁	T ₂	A	B
			25	25
Read(A,t)				
t:=t+100				
Write(A,t)		125		
	Read(A,s)			
	s:=s*2			
	Write(A,s)	250		
Read(B,t)				
t:=t+100				
Write(B,t)			125	
	Read(B,s)			
	s:=s*2			
	Write(B,s)		250	

S4	T ₁	T ₂	A	B
			25	25
Read(A,t)				
t:=t+100				
Write(A,t)			125	
	Read(A,s)			
	s:=s*2			
	Write(A,s)		250	
Read(B,s)				
s:=s*2				
Write(B,s)			50	
Read(B,t)				
t:=t+100				
Write(B,t)				150

Hình 2.7: Lịch khả tuần tự

Lịch S₅ (Trang sau)

Hình 2.8: Lịch không khả tuần tự

S5	T ₁	T ₂	A	B
			25	25
Read(A,t)				
t:=t+100				
Write(A,t)		125		
	Read(A,s)			
	s:=s*1			
	Write(A,s)	125		
	Read(B,s)			
	s:=s*1			
	Write(B,s)		25	
Read(B,t)				
t:=t+100				
Write(B,t)				125

Hình 2.9 : Lịch khả tuần tự

II.2.3.1 Khả tuần tự xung đột (Conflict-Serializability)

- Xét 2 hành động liên tiếp nhau trong 1 lịch thao tác
 - Nếu thứ tự của chúng được đổi cho nhau
 - Thì hoạt động của ít nhất 1 giao tác có thể thay đổi.
- Cho lịch S có 2 giao tác T_i và T_j, xét các trường hợp
 - r_i(X) ; r_j(Y)
 - Không bao giờ có xung đột, ngay cả khi X=Y
 - Cả 2 thao tác không làm thay đổi giá trị của đơn vị dữ liệu X, Y
 - r_i(X) ; w_j(Y)
 - Không xung đột khi X≠Y
 - T_j ghi Y sau khi T_i đọc X, giá trị của X không bị thay đổi
 - T_i đọc X không ảnh hưởng gì đến T_j ghi giá trị của Y
 - w_i(X) ; r_j(Y)
 - Không xung đột khi X≠Y

- $w_i(X) ; w_j(Y)$
 - Không xung đột khi $X \neq Y$
- Hai hành động xung đột nếu thỏa ba điều kiện sau:
 - Thuộc 2 giao tác khác nhau
 - Truy xuất đến cùng 1 đơn vị dữ liệu
 - Có ít nhất một hành động ghi (write)

Nếu hai hành động xung đột nhau thì không thể hoán vị thứ tự cho nhau. Hình 2.9 minh họa cho các trường hợp của hai hành động xung đột nhau

T_i	T_j	T_i	T_j	T_i	T_j
Write(A)		Read(A)		Write(A)	
	Write(A)		Write(A)		Read(A)

Hình 2.10: Hai hành động xung đột nhau

Ví dụ:

Hình 2.11 minh họa cho các hành động xung đột.

S_6	T_1	T_2
	Read (A)	
	Write (A)	
		Read (A)
		Write (A)
	Read (B)	
	Write (B)	
		Read (B)
		Write (B)

Hình 2.11: Lịch giao tác có các hành động xung đột nhau

Hành động **Write(A)** trong T_1 xung đột với **Read(A)** trong T_2 . Tuy nhiên, hành động Write(A) trong T_2 không xung đột với hành động Read(B) trong T_1 do các hành động này truy xuất các đơn vị dữ liệu khác nhau.

Gọi I_i và I_j là hai hành động liên tiếp trong lịch trình S. Nếu I_i và I_j là các hành động của các giao tác khác nhau và không xung đột, khi đó ta có thể đổi thứ

tự của chúng mà không làm ảnh hưởng gì đến kết quả xử lý và như vậy ta nhận được một lịch trình mới S' tương đương với S . Trong *Hình 2.11*, do hành động Write(A) của T_2 không xung đột với hành động Read(B) của T_1 , ta có thể đổi chỗ các hành động này để được một lịch trình tương đương như *Hình 2.12* dưới đây:

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
Read (B)	
	Write (A)
Write (B)	
	Read (B)
	Write (B)

Hình 2.12: Đổi chỗ hai hành động xung đột nhau

Ta tiếp tục đổi chỗ các hành động không xung đột như sau:

- ♦ Đổi chỗ hành động Read(B) của T_1 với hành động Read(A) của T_2 .
- ♦ Đổi chỗ hành động Write(B) của T_1 với hành động Write(A) của T_2 .
- ♦ Đổi chỗ hành động Write(B) của T_1 với hành động Read(A) của T_2 .

Kết quả cuối cùng của các bước đổi chỗ này là một lịch trình mới (*Hình 2.13*)

S_6'	T_1	T_2
	Read (A)	
	Write (A)	
	Read (B)	
	Write (B)	
		Read (A)
		Write (A)
		Read (B)
		Write (B)

Hình 2.13: Lịch trình tuần tự S_6' tương đương với lịch trình ban đầu S_6

Sự tương đương này cho ta thấy: bất chấp trạng thái hệ thống ban đầu, lịch S tại *Hình 2.11* sẽ sinh ra cùng trạng thái cuối như một lịch trình tuần tự nào đó.

Nếu một lịch trình S có thể biến đổi thành một lịch trình S' bởi một dãy các đổi chỗ các hành động không xung đột, ta nói S và S' là **tương đương xung đột** (conflict equivalent).

Khái niệm tương đương xung đột dẫn đến khái niệm khả tuần tự xung đột. Ta nói một lịch trình S là **khả tuần tự xung đột** (conflict serializable) *nếu nó tương đương xung đột với một lịch trình tuần tự*.

Như vậy, lịch S_6 tại *Hình 2.11* là khả tuần tự xung đột. Lịch trình S_7 dưới đây không tương đương xung đột với một lịch trình tuần tự nào. Do vậy nó không là khả tuần tự xung đột:

S_7	T_3	T_4
	Read(A)	
		Write(A)
	Write(A)	

Hình 2.14: Lịch trình không tương đương xung đột

Có thể có hai lịch trình sinh ra cùng kết quả, nhưng không tương đương xung đột.

● Xét lại lịch S_5

S_5	T_1	T_2	A	B
			25	25
	Read(A,t)			
	t:=t+100			
	Write(A,t)		125	
		Read(A,s)		
		s:=s*1		
		Write(A,s)	125	
		Read(B,s)		
		s:=s*1		
		Write(B,s)		25
	Read(B,t)			
	t:=t+100			
	Write(B,t)			125

Hình 2.15: Lịch trình khả tuần tự nhưng không khả tuần tự xung đột

● Xét trường hợp

S ₈	T ₁	T ₂	T ₃
Write(Y)			
Write(X)			
	Write(Y)		
	Write(X)		
		Write(X)	

Hình 2.16: Lịch trình tuần tự

S _{8'}	T ₁	T ₂	T ₃
Write(Y)			
	Write(Y)		
	Write(X)		
Write(X)			
		Write(X)	

Hình 2.17: Lịch trình khả tuần tự nhưng không khả tuần tự xung đột

II.2.3.2 Kiểm tra khả tuần tự xung đột (Conflict-Serializability)

Giả sử S là một lịch trình. Ta xây dựng một đồ thị định hướng, được gọi là đồ thị trình tự từ S (Precedence Graph – P(S)). Đồ thị gồm một cặp (V, E) trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh bao gồm tất cả các giao tác tham gia vào lịch trình.

Tập các cung bao gồm tất cả các cung dạng $T_i \rightarrow T_j$ sao cho một trong các điều kiện sau được thoả mãn:

1/- T_i thực hiện **Write(X)** trước khi T_j thực hiện **Read(X)**.

2/- T_i thực hiện **Read(X)** trước khi T_j thực hiện **Write(X)**.

3/- T_i thực hiện **Write(X)** trước khi T_j thực hiện **Write(X)**.

Nếu một cung $T_i \rightarrow T_j$ tồn tại trong đồ thị trình tự, thì trong bất kỳ lịch trình tuần tự S' nào tương đương với S, T_i phải xuất hiện trước T_j .

- Hai lịch S₁ và S₂ là tương đương xung đột thì chúng có đồ thị trình tự giống nhau

$$S_1, S_2 \text{ tương đương xung đột} \Rightarrow P(S_1) = P(S_2)$$

- Nhưng hai đồ thị trình tự giống nhau thì không đảm bảo lịch trình của chúng là tương đương xung đột.

$$P(S_1) = P(S_2) \not\Rightarrow S_1, S_2 \text{ tương đương xung đột}$$

Ví dụ:

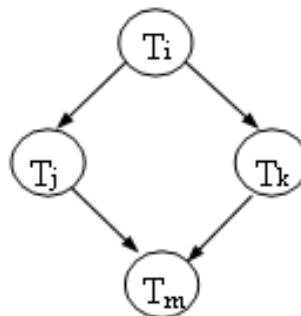
Xét 2 trường hợp như hình *Hình 2.18*:

S ₉	T ₁	T ₂	S _{9'}	T ₁	T ₂
	Write(A)				Read(A)
		Read(A)		Write(A)	
		Write(B)		Read(B)	
	Read(B)				Write(B)



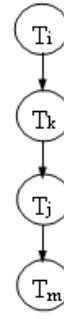
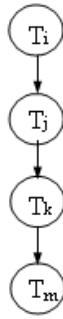
Hình 2.18: Hai lịch trình có cùng đồ thị trình tự nhưng không tương đương xung đột

Nếu đồ thị trình tự đối với S có chu trình, khi đó lịch trình S không là khả tuần tự xung đột. Nếu đồ thị không chứa chu trình, khi đó lịch trình S là khả tuần tự xung đột. Thứ tự khả tuần tự có thể nhận được thông qua sắp xếp topo (topological sorting), nó xác định một thứ tự tuyến tính nhất quán với thứ tự bộ phận của đồ thị trình tự. Nói chung, có một vài thứ tự tuyến tính có thể nhận được qua sắp xếp topo. Ví dụ, đồ thị sau:



Hình 2.19: Sắp xếp topo

Có hai thứ tự tuyến tính chấp nhận được là (*Hình 2.20*):

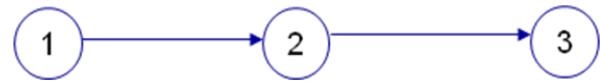


Hình 2.20: Thứ tự tuyến tính trong sắp xếp topo

Như vậy, để kiểm thử tính khả tuần tự xung đột, ta cần xây dựng đồ thị trình tự và gọi thuật toán phát hiện chu trình. Ta nhận được một sơ đồ thực nghiệm để xác định tính khả tuần tự xung đột.

Ví dụ: Cho lịch S_{10} như sau và có đồ thị trình tự như tại *Hình 2.21*

S ₁₀	T ₁	T ₂	T ₃
		Read(A)	
	Read(B)		
		Write(A)	
			Read(A)
	Write(B)		
			Write(A)
		Read(B)	
			Write(B)

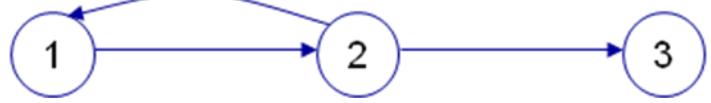


Hình 2.21: Đồ thị trình tự P(S) không có chu trình

Do P(S) không có chu trình nên S khả tuần tự xung đột theo thứ tự T₁, T₂, T₃

Cho lịch S_{11} có đồ thị trình tự như tại *Hình 2.22*

S ₁₁	T ₁	T ₂	T ₃
		Read(A)	
	Read(B)		
		Write(A)	
		Read(B)	
			Read(A)
	Write(B)		
			Write(A)
		Write(B)	



Hình 2.22: Đồ thị trình tự P(S) có chu trình

Do P(S) có chu trình nên lịch S không khả tuần tự xung đột

II.2.4 Khả tuân tự View (View Serializability)

Xét hai lịch trình S và S', trong đó cùng một tập hợp các giao tác tham gia vào cả hai lịch trình. Các lịch trình S và S' được gọi là tương đương view (view-equivalent) nếu thỏa ba điều kiện sau:

1/- Đối với mỗi đơn vị dữ liệu A, nếu giao tác Ti đọc giá trị khởi đầu của A trong lịch trình S, thì giao tác Ti phải cũng đọc giá trị khởi đầu của A trong lịch trình S'.

2/- Đối với mỗi đơn vị dữ liệu A, nếu giao tác Ti thực hiện Read(A) trong lịch trình S và giá trị đó được sản sinh ra bởi giao tác Tj thì Ti cũng phải đọc giá trị của A được sinh ra bởi giao tác Tj trong S'.

3/- Đối với mỗi đơn vị dữ liệu A, giao tác thực hiện hoạt động Write(A) sau cùng trong lịch trình S, phải thực hiện hoạt động Write(A) sau cùng trong lịch trình S'.

Điều kiện 1 và 2 đảm bảo mỗi giao tác đọc cùng các giá trị trong cả hai lịch trình và do vậy thực hiện cùng tính toán. Điều kiện 3 đi cặp với các điều kiện 1 và 2 đảm bảo cả hai lịch trình cho ra kết quả là trạng thái cuối cùng của hệ thống như nhau.

Quan niệm tương đương view đưa đến quan niệm khả tuân tự view. Ta nói lịch trình S là **khả tuân tự view** (view serializable) nếu nó **tương đương view với một lịch trình tuân tự**. Ta xét lịch trình sau:

S'12	T ₃	T ₄	T ₅
	Read(A)		
		Write(A)	
	Write(A)		
			Write(A)

Nó tương đương view với lịch trình tuân tự < T₃, T₄, T₅ > do hành động Read(A) đọc giá trị khởi đầu của A trong cả hai lịch trình và T₆ thực hiện Write sau cùng trong cả hai lịch trình như vậy S₁₂ khả tuân tự view.

Mỗi lịch trình khả tuân tự xung đột là khả tuân tự view, nhưng có những lịch trình khả tuân tự view không khả tuân tự xung đột (ví dụ S₁₂).

Trong S₁₂ các giao tác T₄ và T₅ thực hiện các hoạt động Write(A) mà không thực hiện hoạt động Read(A), Các Write dạng này được gọi là các Write mù (blind

write). Các Write mù xuất hiện trong bất kỳ lịch trình khả tuần tự view không khả tuần tự xung đột.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ VIEW

- Cho 1 lịch thao tác S
- **Thêm 1 giao tác cuối T_f** vào trong S sao cho T_f thực hiện việc **đọc hết tất cả** đơn vị dữ liệu ở trong S, bỏ qua điều kiện thứ 3 của định nghĩa tương đương view.

$$S = \dots W_1(A) \dots \dots \dots W_2(A) R_f(A)$$

- **Thêm 1 giao tác đầu tiên T_b** vào trong S sao cho T_b thực hiện việc **ghi các giá trị ban đầu** cho các đơn vị dữ liệu, bỏ qua điều kiện thứ 2 của định nghĩa tương đương view)

$$S = W_b(A) \dots W_1(A) \dots \dots \dots W_2(A) \dots$$

- Vẽ đồ thị trình tự gán nhãn cho S, ký hiệu $G(S)$,

- Đính là các giao tác T_i (bao gồm T_b và T_f)
- Cung

(1) Nếu có **$R_i(X)$ với gốc là T_j** ($w_j(X) \dots r_i(X)$) thì vẽ cung đi từ T_j đến T_i



Hình 2.23: **Cung đi từ T_j đến T_i nếu có $R_i(X)$ với gốc là T_j**

(2) Với mỗi $W_j(X) \dots R_i(X)$, xét $W_k(X)$ sao cho T_k không chèn vào giữa T_j và T_i

- (2a) Nếu $T_j \neq T_b$ và $T_i \neq T_f$ thì vẽ cung $T_k \rightarrow T_j$ và $T_i \rightarrow T_k$

T_k	T_j	T_i
Write(X)		
	Write(X)	
		Read(X)

T_j	T_i	T_k
Write(X)		
	Read(X)	
		Write(X)



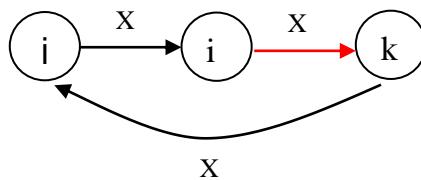
Hình 2.24: Vẽ cung cho trường hợp (2a)

Chọn 1 cung vừa tạo, sao cho đồ thị không có chu trình

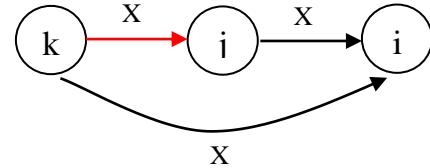
- (2b) Nếu $T_j = T_b$ thì vẽ cung $T_i \rightarrow T_k$
- (2c) Nếu $T_i = T_f$ thì vẽ cung $T_k \rightarrow T_j$

$T_j = T_b$	T_i	T_k
Write(X)		
	Read(X)	
		Write(X)

T_k	T_j	$T_i = T_f$
Write(X)		
	Write(X)	
		Read(X)



Hình 2.25a: Vẽ cung cho trường hợp (2b)



Hình 2.25b: Vẽ cung cho trường hợp (2c)

II.3 GIAO TÁC TRONG SQL

Giao tác SQL được định nghĩa dựa trên các câu lệnh xử lý giao tác sau đây:

- BEGIN TRANSACTION: Bắt đầu một giao tác
- SAVE TRANSACTION: Đánh dấu một vị trí trong giao tác (gọi là điểm đánh dấu).
- ROLLBACK TRANSACTION: Quay lui trở lại đầu giao tác hoặc một điểm đánh dấu trước đó trong giao tác.
- COMMIT TRANSACTION: Đánh dấu điểm kết thúc một giao tác. Khi câu lệnh này thực thi cũng có nghĩa là giao tác đã thực hiện thành công.
- ROLLBACK [WORK]: Quay lui trở lại đầu giao tác.
- COMMIT [WORK]: Đánh dấu kết thúc giao tác.

❖ Câu hỏi (bài tập) củng cố:

1. Giao tác là gì?
2. Trình bày các tính chất của giao tác?
3. Trình bày các trạng thái của giao tác?
4. Cho biết lịch tuần tự là gì?
5. Cho biết lịch khả tuần tự là gì?
6. Thể nào là hai hành động xung đột nhau?
7. Cách kiểm tra lịch khả tuần tự?
8. Thể nào là khả tuần tự View?
9. Cách kiểm tra lịch khả tuần tự View?
10. Kiểm tra tính khả tuần tự cho các lịch sau?
 - a. Cho lịch S₁₃ như sau:

	T ₁	T ₂	T ₃
1		Read(A)	
2	Read (B)		
3		Write(B)	
4			Read(A)
5	Write(B)		
6		Write(A)	
7			Write(A)
8	Read (B)		
9		Read (B)	
10		Write(A)	
11	Read(A)		

- b. Cho lịch S₁₄ như sau:

	T ₁	T ₂	T ₃	T ₄
1	Write A)			
2		Re d(A)		
3			Read(A)	
4				Write(A)

❖ Câu hỏi (bài tập) củng cố:

c. Cho lịch S₁₅ như sau:

	T1	T2	T3	T4
1		Read(A)		
2			Read (A)	
3		Write(B)		
4			Write(A)	
5	Read(B)			
6				Read (B)
7	Read (A)			
8	Write(C)			
9				Write(A)

11. Cho các lịch thao tác:

- a) r₁(A); r₂(A); r₃(B); w₁(A); r₂(C); r₂(B); w₂(B); w₁(C);
- b) r₁(A); w₁(B); r₂(B); w₂(C); r₃(C); w₃(A);
- c) w₃(A); r₁(A); w₁(B); r₂(B); w₂(C); r₃(C);
- d) r₁(A); r₂(A); w₁(B); w₂(B); r₁(B); r₂(B); w₂(C); w₁(D);
- e) r₁(A); r₂(A); r₁(B); r₂(B); r₃(A); r₄(B); w₁(A); w₂(B);

Hãy thực hiện các yêu cầu sau:

11.1 Vẽ đồ thị trình tự của các lịch thao tác trên.

11.2 Chúng có khả tuần tự xung đột không? Nếu có, cho biết chúng khả tuần tự theo thứ tự nào?

11.3 Có những lịch nào tương đương nhau nhưng không tương đương xung đột?

12. Hãy vẽ đồ thị trình tự gán nhãn cho các lịch thao tác sau và tìm xem có những lịch nào là khả tuần tự view:

- a) r₁(A); r₂(A); r₃(A); w₁(B); w₂(B); w₃(B);
- b) r₁(A); r₂(A); r₃(A); r₄(A); w₁(B); w₂(B); w₃(B); w₄(B);
- c) r₁(A); r₃(D); w₁(B); r₂(B); w₃(B); r₄(B); w₂(C); r₅(C); w₄(E); r₅(E); w₅(B);
- d) w₁(A); r₂(A); w₃(A); r₄(A); w₅(A); r₆(A);

13. Thực hành trên máy tính phần giao tác (Xem tại **PHỤC LỤC 5**)

CHƯƠNG 3

ĐIỀU KHIỂN ĐỒNG THỜI

❖ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

- Trình bày các khái niệm của kỹ thuật khóa và kỹ thuật nhãn thời gian
- Phân tích các kỹ thuật điều khiển đồng thời: kỹ thuật khóa, nhãn thời gian
- So sánh các kỹ thuật điều khiển đồng thời: kỹ thuật khóa, nhãn thời gian

III.1 CÁC VẤN ĐỀ TRONG ĐIỀU KHIỂN ĐỒNG THỜI

III.1.1 Mất dữ liệu đã cập nhật (*lost updated*)

Xét hai giao tác

T ₁	T ₂
Read (A)	Read (A)
A:=A+10	A:=A+20
Write (A)	Write (A)

Hình 3.1: Hai giao tác độc lập

Giả sử T₁ và T₂ được thực hiện đồng thời

A=50	T ₁	T ₂
t ₁	Read (A)	
t ₂		Read (A)
t ₃	A:=A+10	
t ₄		A:=A+20
t ₅	Write (A)	
t ₆		Write (A)

A=60 A=70

Hình 3.2: Hai giao tác thực hiện đồng thời

Ta thấy dữ liệu đã cập nhật **tại t₅ của T₁** bị mất vì đã bị ghi **chồng lên** ở thời điểm t₆

III.1.2 Không thể đọc lại (unrepeatable read)

Xét hai giao tác

T ₁	T ₂
Read (A)	Read (A)
A:=A+10	Print (A)
Write (A)	Read (A)

Giả sử T₁ và T₂ được thực hiện đồng thời

A=50	T ₁	T ₂	
t ₁	Read (A)		
t ₂		Read (A)	A=50
t ₃	A:=A+10		
t ₄		Print (A)	A=50
t ₅	Write (A)		
t ₆		Read (A)	A=60
t ₇		Print (A)	A=60

Hình 3.3: Không thể đọc lại dữ liệu

Ta thấy T₂ tiến hành đọc A hai lần thì cho hai kết quả khác nhau

III.1.3 “Bóng ma” (phantom)

Xét 2 giao tác T₁ và T₂ được xử lý đồng thời

- A và B là 2 tài khoản
- T₁ rút 1 số tiền ở tài khoản A rồi đưa vào tài khoản B
- T₂ kiểm tra đã nhận đủ tiền hay chưa?

A=70, B=50	T ₁	T ₂	
t ₁	Read (A)		A=50
t ₂	A:=A - 50		
t ₃	Write (A)		A=20
t ₄		Read (A)	A=20
t ₅		Read (B)	B=50
t ₆		Print (A+B)	A+B=70
t ₇	Read(B)		
t ₈	B=B+50		
t ₉	Write(B)		

Mất 50 ???

Hình 3.4: “Bóng ma”

III.1.4 Đọc dữ liệu chưa chính xác (dirty read)

Xét 2 giao tác T_1 và T_2 được xử lý đồng thời như *Hình 3.5*:

	T₁	T₂
t ₁	Read (A)	
t ₂	A:=A+10	
t ₃	Write (A)	
t ₄		Read (A)
t ₅		Print (A)
t ₆	Abort	

Hình 3.5: Đọc dữ liệu rác

T_2 đã đọc dữ liệu được ghi bởi T_1 nhưng sau đó T_1 yêu cầu **hủy việc ghi**

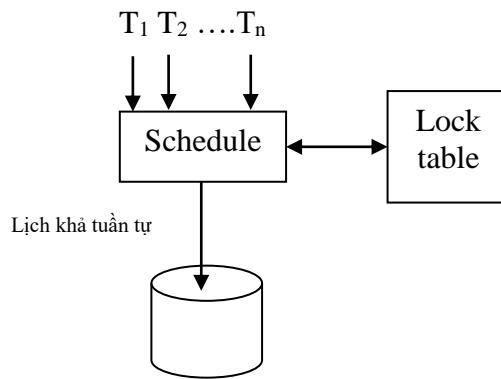
III.2 KỸ THUẬT KHÓA (LOCK)

III.2.1 Giới thiệu

- Làm thế nào để bộ lập lịch ép buộc một lịch phải khả thi?
- Bộ lập lịch với cơ chế khóa (locking scheduler)

Có thêm 2 hành động

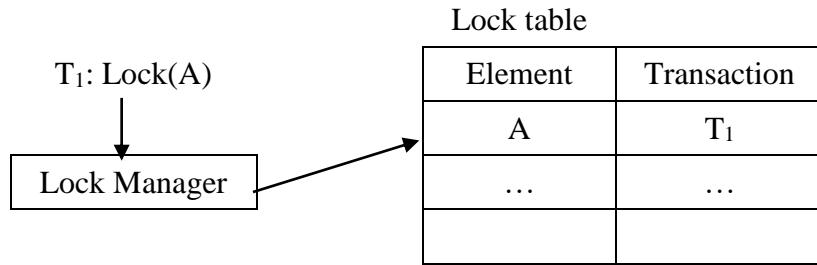
- o Lock
- o Unlock



Hình 3.6: Bộ lập lịch với cơ chế khóa

- Các giao tác trước khi muôn **đọc/viết** lên 1 đơn vị dữ liệu phải phát ra 1 yêu cầu **xin khóa** (lock) đơn vị dữ liệu đó: **Lock(A)** hay **l(A)**

- Yêu cầu này được bộ phận *quản lý khóa* xử lý: Nếu yêu cầu được chấp thuận thì giao tác mới được phép *đọc/ghi* lên đơn vị dữ liệu



Hình 3.7: Giao tác yêu cầu khóa

- Sau khi thao tác xong thì giao tác phải phát ra lệnh *giải phóng đơn vị dữ liệu*: *Unlock(A)* hay *u(A)*

III.2.2 Qui tắc

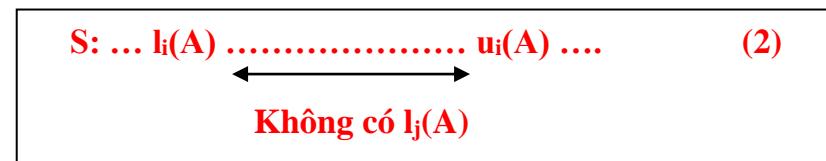
III.2.2.1 Giao tác đúng đắn

Giao tác T_i trước khi muốn đọc/viết lên một đơn vị dữ liệu nào thì phải phát ra yêu cầu khóa đơn vị dữ liệu đó. Sau khi đọc/viết xong thì phải giải phóng cho đơn vị dữ liệu đã bị khóa.

$T_i: \dots l(A) \dots r(A) / w(A) \dots u(A) \dots$ (1)

III.2.2.2 Lịch thao tác hợp lệ

Trong khi giao tác thứ i khóa một đơn vị dữ liệu nào đó cho đến khi i giải phóng cho đơn vị dữ liệu đó thì không có một giao tác thứ j nào được xin khóa trên đơn vị dữ liệu đó.



Ví dụ

Xét lịch S_{15} như sau

S ₁₅	T ₁	T ₂
	Lock(A)	
	Read(A,t)	
	t:= t+100	
	Write(A,t)	
	Unlock(A)	
		Lock(A)
		Read(A,s)
		s:= s*2
		Write(A,s)
		Unlock(A)
		Lock(B)
		Read(B,s)
		s:= s*2
		Write(B,s)
		Unlock(B)
	Lock(B)	
	Read(B,t)	
	t:= t+100	
	Write(B,t)	
	Unlock(B)	

Hình 3.8: Giao tác đúng, lịch hợp lệ

III.2.2.3 Kiểm tra tính khả tuần tự

Nếu lịch S₁₆ hợp lệ thì S₁₆ có khả tuần tự không?

S ₁₆	T ₁	T ₂	A	B
			25	25
	Lock(A); Read(A,t)			
	t:= t+100			
	Write(A,t); Unlock(A)		125	
		Lock(A); Read(A,s)		
		s:= s*2		
		Write(A,s); Unlock(A)	250	
		Lock(B); Read(B,s)		
		s:= s*2		
		Write(B,s); Unlock(B)		50
	Lock(B); Read(B,t)			
	t:= t+100			
	Write(B,t); Unlock(B)			150

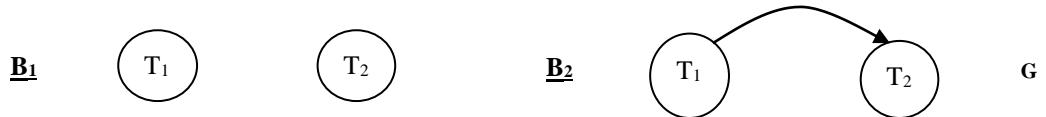
Hình 3.9: Lịch hợp lệ, nhưng không khả tuần tự

Kiểm tra tính khả tuần tự

- Input : Lịch S được lập từ n giao tác xử lý đồng thời
 T_1, T_2, \dots, T_n theo kỹ thuật khóa đơn giản
 - Output : S khả tuần tự hay không?
- Xây dựng 1 đồ thị có hướng G
- Mỗi giao tác T_i là 1 đỉnh của đồ thị
 - Nếu một giao tác T_j phát ra **Lock(A)** sau một giao tác T_i phát ra **Unlock(A)** thì sẽ vẽ cung từ $T_i \rightarrow T_j$, $i \neq j$
 - S khả tuần tự nếu G không có chu trình

Ví dụ 1: Xét lịch S_{17} như sau:

S_{17}	T_1	T_2
Lock(A)		
Read(A)		
$A=A-10$		
Write(A)		
UnLock(A)		
	Lock(A)	
	Read(A)	
	Print(A)	
	Read(A)	
	Print(A)	
	UnLock(A)	

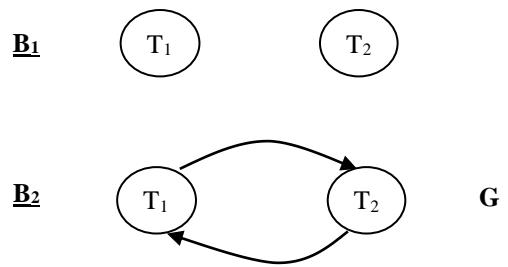


B3: G không có chu trình nên S khả tuần tự theo thứ tự $T_1 \rightarrow T_2$

Hình 3.10: Đồ thị G không có chu trình

Ví dụ 2: Xét lịch S₁₈ như sau:

S ₁₈	T ₁	T ₂
	Lock(A)	
	Read(A,t)	
	t := t + 100	
	Write(A,t)	
	Unlock(A)	
		Lock(A)
		Read(A,s)
		s := s * 2
		Write(A,s)
		Unlock(A)
		Lock(B)
		Read(B,s)
		s := s * 2
		Write(B,s)
		Unlock(B)
	Lock(B)	
	Read(B,t)	
	t := t + 100	
	Write(B,t)	
	Unlock(B)	



B3: G có chu trình nên S không khả thi

Hình 3.11: Giao tác có đồ thị G có chu trình

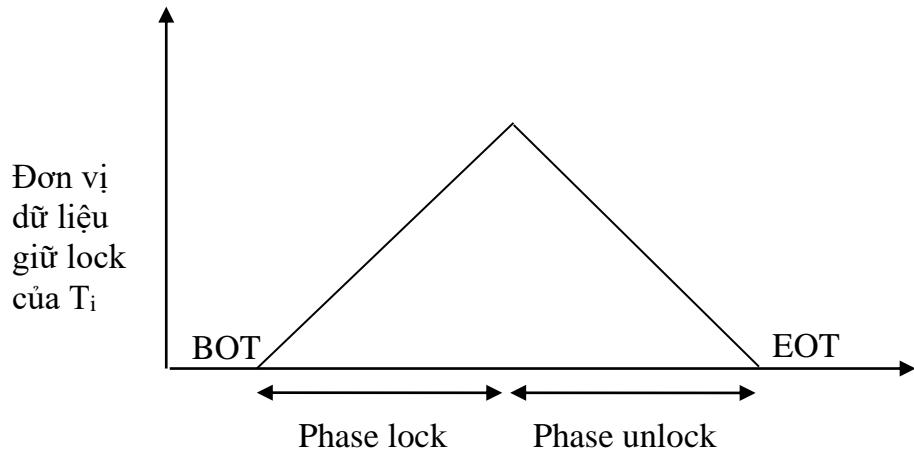
III.2.3 Khóa 2 giai đoạn (Two Phase Lock - 2PL)

- Qui tắc

(3) Giao tác 2PL



Thực hiện xong hết tất cả các yêu cầu lock rồi mới tiến hành unlock. Chu kỳ lock và unlock được minh họa như *Hình 3.12*



Hình 3.12: Chu kỳ của khóa 2PL

Ví dụ 1:

Các giao tác T_1, T_2 là các giao tác thỏa 2PL, T_3 và T_4 không thỏa 2PL

T_1
Lock(A)
Read(A)
Lock(B)
Read(B)
$B:=B+A$
Write(B)
Unlock(A)
Unlock(B)

T_2
Lock(B)
Read(B)
Lock(A)
Read(A)
Unlock(B)
$A:=A+B$
Write(A)
Unlock(A)

T_3
<i>Lock(B)</i>
Read(B)
$B:=B - 50$
Write(B)
Unlock(B)
<i>Lock(A)</i>
Read (A)
$A:=A+50$
Write(A)
<i>Unlock(A)</i>

T_4
<i>Lock(A)</i>
Read(A)
UnLock(A)
<i>Lock(B)</i>
<i>Read(B)</i>
<i>Unlock(B)</i>
Print(A+b)

Hình 3.13: Các giao tác thỏa 2PL

Hình 3.14: Các giao tác không thỏa 2PL

- **Các lịch S thỏa qui tắc (1), (2), (3) \Rightarrow S khả tuần tự xung đột**
- **Chú ý trong kỹ thuật khóa 2PL**

S ₁₇	T ₁	T ₂
	Lock(A)	
	Read(A,t)	Lock(B)
		Read(B,s)
	t:=t+100	
		s:=s*2

S ₁₇	T₁	T₂
	Write(A,t)	
		Write(B,s)
	Lock(B) ↓	
	Không xin được lock → chờ	Lock(A) ↓
		Không xin được lock → chờ

Hình 3.15: Khóa chết trong 2PL

III.2.4 Kỹ thuật khóa đọc viết

III.2.4.1 Giới thiệu

S ₁₈	T_i	T_j
	Lock(A)	
	Read(A)	
	Unlock(A)	
		Lock(A)
		Read(A)
		Unlock(A)

- Bộ lập lịch có các hành động
 - Khóa đọc (Read lock, Shared lock)
 - RLock(A) hay rl(A)
 - Khóa ghi (Write lock, Exclusive lock)
 - WLock(A) hay wl(A)
 - Giải phóng khóa
 - Unlock(A) hay u(A)
- Cho 1 đơn vị dữ liệu A bất kỳ
 - WLock(A)
 - Hoặc có 1 khóa ghi duy nhất lên A
 - Hoặc không có khóa ghi nào lên A

- RLock(A)
 - Có thể có nhiều khóa đọc được thiết lập lên A
- Giao tác muốn Write(A)
 - Yêu cầu WLock(A)
 - WLock(A) sẽ được chấp thuận nếu A tự do
 - Sẽ không có giao tác nào nhận được WLock(A) hay RLock(A)
- Giao tác muốn Read(A)
 - Yêu cầu RLock(A) hoặc WLock(A)
 - RLock(A) sẽ được chấp thuận nếu A không đang giữ một WLock nào
 - Không ngăn chặn các thao tác khác cùng xin RLock(A)
 - Các giao tác không cần phải chờ nhau khi đọc A
- Sau khi thao tác xong thì giao tác phải giải phóng khóa trên đơn vị dữ liệu A
 - U(A)

III.2.4.2 Qui tắc giao tác đúng đắn

- Giao tác đúng đắn

T_i: ... rl(A) ... r(A) ... u(A) ...

(1)

T_i: ... wl(A) ... w(A) ... u(A) ...

III.2.4.3 Các vấn đề cần lưu ý

- Các giao tác đọc và ghi trên cùng 1 đơn vị dữ liệu thì giải quyết như thế nào, như trong trường hợp sau:

T _i
Read(B)
Write(B)?

- Giải quyết
 - Cách 1: yêu cầu khóa độc quyền
 $T_i: \dots wl(A) \dots r(A) \dots w(A) \dots u(A) \dots$
 - Cách 2: nâng cấp khóa
 $T_i: \dots rl(A) \dots r(A) \dots wl(A) \dots w(A) \dots u(A) \dots$
- Hãy suy nghĩ và cho biết cách nào là hợp lý
 - Xin khóa thứ 2 cho đơn vị dữ liệu muốn ghi?
 - Xin khóa độc quyền ngay từ đầu?

III.2.4.4 Qui tắc lịch hợp lệ

$S: \dots rl_i(A) \dots \dots \dots u_i(A) \dots$ <div style="text-align: center; margin-top: 10px;"> không có $wl_j(A)$ </div>
--

(2)

$S: \dots wl_i(A) \dots \dots \dots u_i(A) \dots$ <div style="text-align: center; margin-top: 10px;"> không có $wl_j(A)$ không có $rl_j(A)$ </div>
--

III.2.4.5 Ma trận tương thích (compatibility matrices)

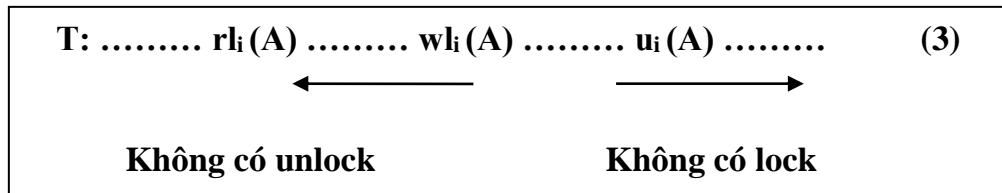
Yêu cầu lock		
Trạng thái hiện hành		
	Share	eXclusive
Share	Yes	No
eXclusive	No	No

Hình 3.16: Ma trận tương thích kỹ thuật khóa đọc viết

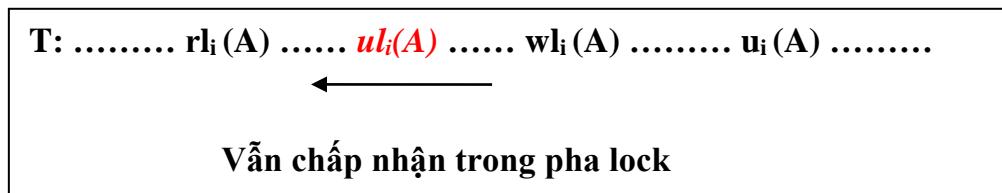
III.2.4.6 Qui tắc giao tác 2PL

- Ngoại trừ trường hợp nâng cấp khóa, các trường hợp còn lại đều giống với nghị thức khóa.

- Nâng cấp xin nhiều khóa hơn



- Nâng cấp giải phóng khóa đọc



- Định lý

S thỏa qui tắc (1), (2), (3) \Rightarrow S khả tuần tự xung đột của khóa đọc viết

Khóa cập nhật (update lock)

- ULock(A)
- Giao tác muốn **read(A)** và sau đó cũng muốn **write(A)**
 - Yêu cầu ULock(A)
 - ULock(A) được chấp thuận khi A tự do hoặc đang giữ RLock
 - Khi đó, không có giao tác nào nhận được RLock(A), WLock(A) hay ULock(A)
- Ma trận tương thích

Yêu cầu lock

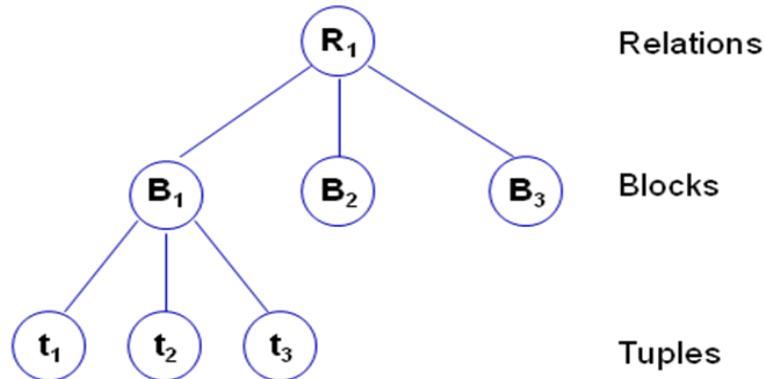
Trạng thái hiện hành	Share	eXclusive	Update
	Share	Yes	No
eXclusive	No	No	No
Update	No	No	No

Hình 3.17: Ma trận tương thích kỹ thuật khóa cập nhật

III.2.5 Kỹ thuật khóa đa hạt

III.2.5.1 Phân cấp dữ liệu

- Quan hệ (Relations) là đơn vị dữ liệu khóa lớn nhất
- Một quan hệ gồm một hoặc nhiều khối (blocks/pages)
- Một khối gồm một hoặc nhiều bộ (tuples)



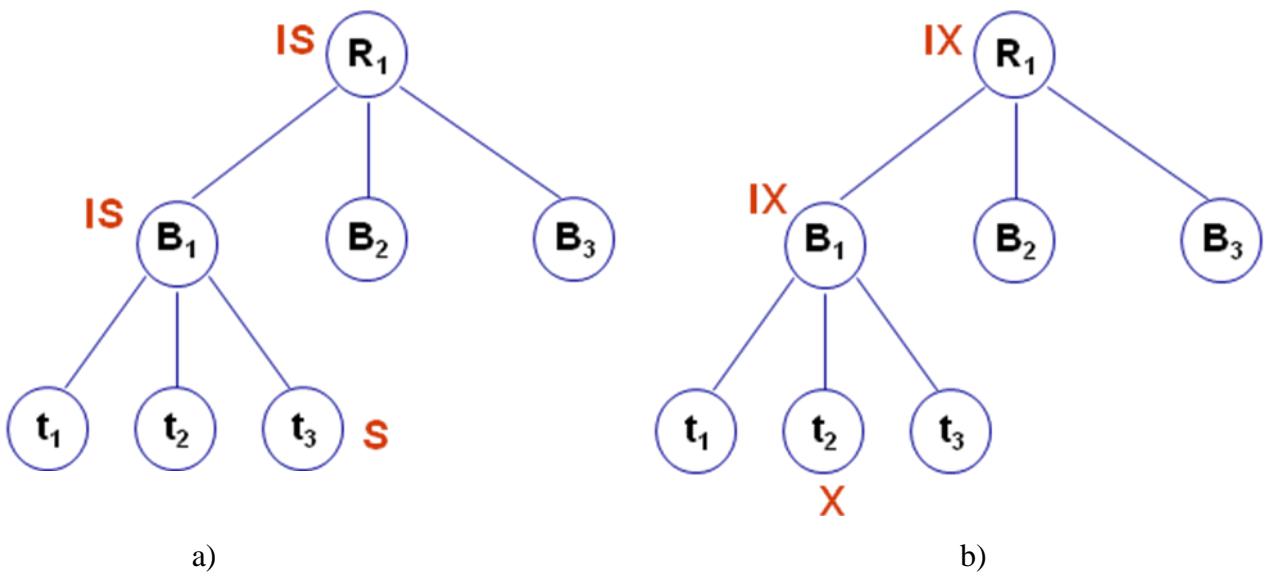
Hình 3.18: Cây phân cấp dữ liệu

III.2.5.2 Kỹ thuật khóa đa hạt

- Gồm các khóa
 - Khóa thông thường
 - Shared lock: S
 - Exclusive lock: X
 - Khóa cảnh báo (warning lock)
 - Warning (intention to) shared lock: IS
 - Warning (intention to) exclusive lock: IX

Ví dụ:

Hình 3.19 minh họa cây phân cấp dữ liệu với trạng thái hiện hành của các nút được khóa



Hình 3.19: Cây phân cấp dữ liệu với trạng thái hiện hành của các nút được khóa

- #### - Ma trận tương thích

Yêu cầu lock

	IS	IX	S	X
Trạng thái hiện hành	IS	Yes	Yes	Yes
	IX	Yes	Yes	No
	S	Yes	No	Yes
	X	No	No	No

Hình 3.20: Ma trận tương thích kỹ thuật khóa đa hat

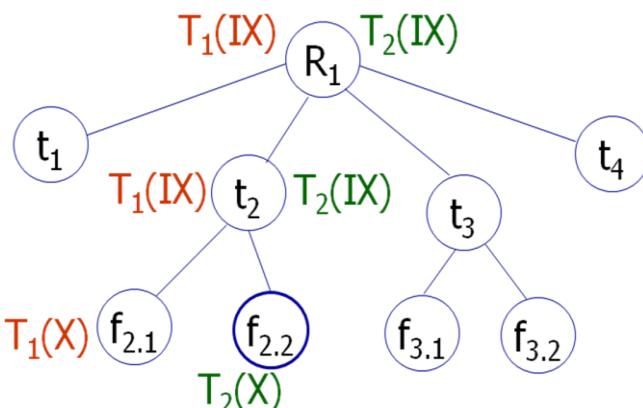
Nút cha đã khóa bằng phương thức	Nút con có thể khóa bằng các phương thức
IS	IS, S
IX	IS, S, IX,X
S	[S, IS] không cần thiết
X	Không có

Hình 3.21: Phương thức khóa các nút của T_i

Mỗi giao tác T_i có thể khóa một nút Q theo các quy tắc sau:

- (1) Thỏa mã trận tương thích
- (2) Khóa nút gốc của cây trước
- (3) Nút Q có thể được khóa bởi T_i bằng S hay IS khi cha (Q) đã bị khóa bởi T_i bằng IX hay IS
- (4) Nút Q có thể được khóa bởi T_i bằng X hay IX khi cha (Q) đã bị khóa bởi T_i bằng IX
- (5) T_i thỏa 2PL
- (6) T_i có thể giải phóng nút Q khi không có nút con nào của Q bị khóa bởi T_i

Ví dụ



Giải thích tại sao T_2 có thể truy xuất $f_{2.2}$ bằng khóa X được?

III.3 KỸ THUẬT NHÃN THỜI GIAN (TIMESTAMP)

III.3.1 Giới thiệu

- Ý tưởng
 - Giả sử không có hành động nào vi phạm tính khả thi
 - Nhưng nếu có, hủy giao tác có hành động đó và thực hiện lại giao tác
- Chọn một *thứ tự thực hiện* nào đó cho các giao tác bằng cách gán nhãn thời gian (timestamping)
 - Mỗi giao tác T sẽ có một nhãn, ký hiệu $TS(T)$
 - Tại thời điểm giao tác bắt đầu

- Thứ tự của các nhãn tăng dần
 - Giao tác bắt đầu trễ thì sẽ có nhãn thời gian lớn hơn các giao tác bắt đầu sớm

Có hai phương pháp đơn giản để thực hiện sự gán nhãn này:

1/- Sử dụng giá trị của đồng hồ hệ thống như nhãn thời gian: Một nhãn thời gian của một giao tác bằng giá trị của đồng hồ khi giao tác đi vào hệ thống.

2/- Sử dụng bộ đếm logic: bộ đếm được tăng lên mỗi khi một nhãn thời gian đã được gán, nhãn thời gian của một giao tác bằng với giá trị của bộ đếm khi giao tác đi vào hệ thống.

Nhãn thời gian của các giao tác xác định thứ tự khả tuần tự. Như vậy, nếu $TS(T_i) < TS(T_j)$, hệ thống phải đảm bảo rằng lịch trình được sinh ra là tương đương với một lịch trình tuần tự trong đó T_i xuất hiện **trước** T_j .

III.3.2 Nhãn thời gian toàn phần

- Mỗi giao tác T khi phát sinh sẽ được gán 1 nhãn $TS(T)$ ghi nhận lại thời điểm phát sinh của T
- Mỗi đơn vị dữ liệu X cũng có 1 nhãn thời $TS(X)$, nhãn này ghi lại $TS(T)$ của giao tác T đã thao tác read/write thành công sau cùng lên X
- Khi đến lượt giao tác T thao tác trên dữ liệu X, so sánh $TS(T)$ và $TS(X)$

Giải thuật

Read(T,X) <pre> If TS(X) <= TS(T) Read(X) ; // Cho phép đọc x TS(X) := TS(T); Else Abort {T} // Hủy bỏ T // Khởi tạo lại TS </pre>	Write(T,X) <pre> If TS(X) <= TS(T) Write(X) ; // Cho phép ghi x TS(X) := TS(T); Else Abort {T} // Hủy bỏ T // Khởi tạo lại TS </pre>
--	--

Ví dụ:

	T₁ TS(T ₁)=100	T₂ TS(T ₁)=200	A TS(A)=0	B TS(B)=0	Ghi chú
1	Read(A)		TS(A)=100		TS(A)<=TS(T1): T1 đọc được A
2		Read(B)		TS(A)=200	TS(B)<=TS(T2): T2 đọc được B
	A:=A*2				
3	Write(A)		TS(A)=100		TS(A)<=TS(T1): T1 ghi lên A được
		B:=B+20			
4		Write(B)		TS(A)=200	TS(B)<=TS(T2): T2 ghi lên B được B
5	Read(B) ↓ (Abort)				TS(B)<=TS(T1): Không đọc được B

Khởi tạo lại với nhãn thời gian mới, lịch khả thi tuần tự theo thứ T2 → T1

Lưu ý: Trong các trường hợp sau:

T₁ TS(T ₁)=100	T₂ TS(T ₂)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Write(A)	TS(A)=120
Write(A)		
↓ T ₁ bị hủy và bắt đầu thực hiện lại với nhãn thời gian mới		

T₁ TS(T ₁)=100	T₂ TS(T ₂)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Read(A)	TS(A)=120
Read(A)		
↓ T ₁ bị hủy và bắt đầu thực hiện lại với nhãn thời gian mới		

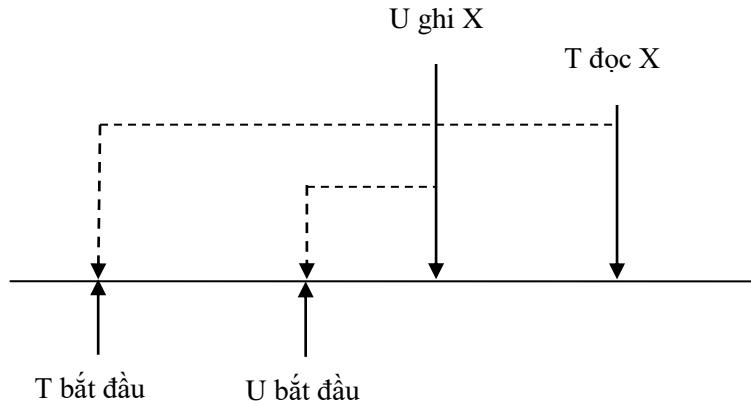
Không phân biệt tính chất của thao tác là đọc hay viết → T₁ vẫn bị hủy và làm
lại từ đầu với một nhãn thời gian mới.

III.3.3 Nhãn thời gian riêng phần

- Nhãn của đơn vị dữ liệu X được tách ra thành 2 phần

- RT(X) - read
 - Ghi nhận TS(T) gần nhất đọc X thành công
- WT(X) - write
 - Ghi nhận TS(T) gần nhất ghi X thành công
- Công việc của bộ lập lịch
 - Gán nhãn RT(X), WT(X)
 - Kiểm tra thao tác đọc/ghi xuất hiện vào lúc nào
 - Có xảy ra tình huống
 - Đọc quá trễ
 - Ghi quá trễ
 - Đọc dữ liệu rác (dirty read)
 - Qui tắc ghi Thomas

III.3.3.1 Đọc quá trễ

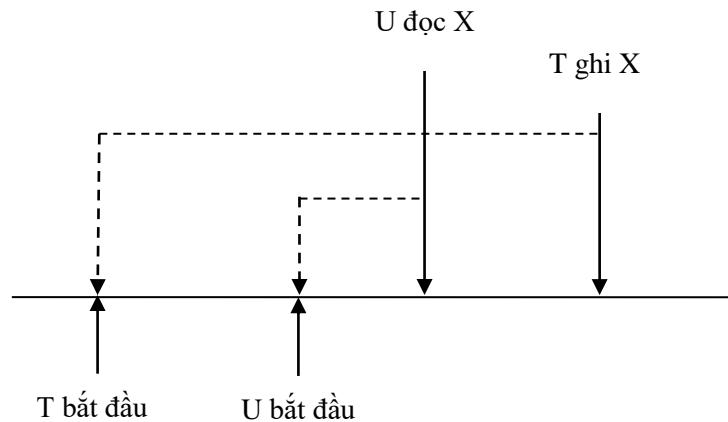


Hình 3.22: Đọc dữ liệu quá trễ

Các giao tác được thực thi như hình *Hình 3.22* thì:

- $TS(T) < TS(U)$
- U ghi X trước, T đọc X sau,
 - $TS(T) < WT(X)$
 - T không thể đọc X sau U \rightarrow Hủy T

III.3.3.2 Ghi quá trễ

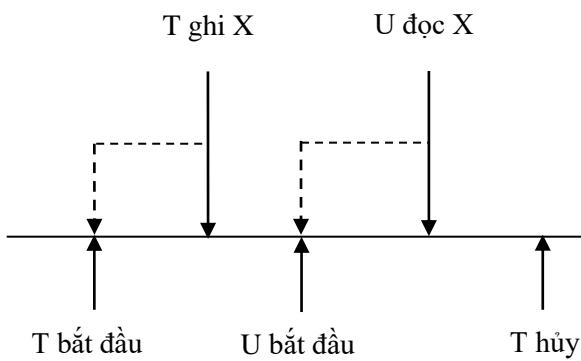


Hình 3.23: Ghi dữ liệu quá trễ

Các giao tác được thực thi như hình *Hình 3.23* thì:

- $TS(T) < TS(U)$
- U đọc X trước, T ghi X sau
 - $WT(X) < TS(T) < RT(X)$
 - U phải đọc được giá trị X được ghi bởi $T \rightarrow$ Hủy T

III.3.3.3 Đọc dữ liệu rác



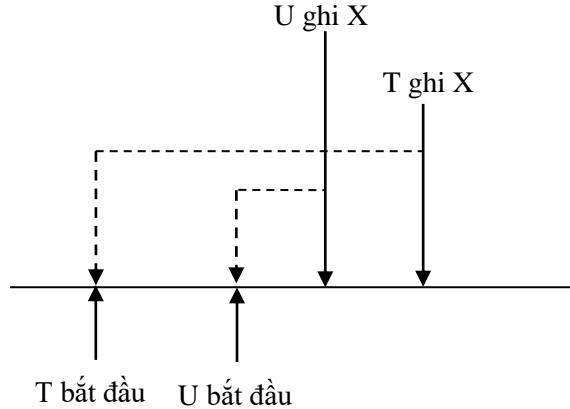
Hình 3.24: Đọc dữ liệu rác

Các giao tác được thực thi như hình *Hình 3.24* thì:

- $ST(T) < ST(U)$
- T ghi X trước, U đọc X sau
 - Thao tác bình thường

- Nhưng T hủy
- U đọc X sau khi T commit
- U đọc X sau khi T abort

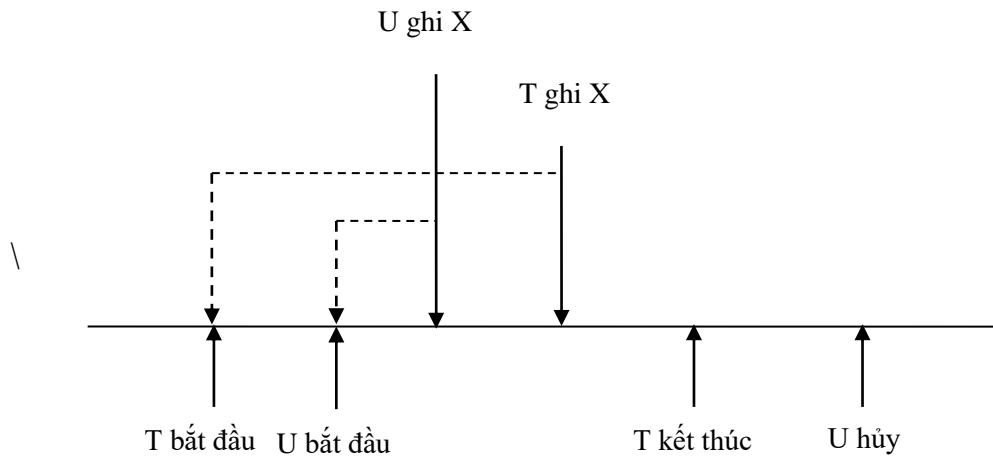
III.3.3.4 Qui tắc ghi Thomas



Hình 3.25: Qui tắc ghi Thomas

- $TS(T) < TS(U)$
- U ghi X trước, T ghi X sau
 - $TS(T) < WT(X)$
 - T ghi X xong thì không làm được gì
 - Không có giao tác nào đọc được giá trị X của T (nếu có cũng bị hủy do đọc quá trễ)
 - Các giao tác đọc sau T và U thì mong muốn đọc giá trị X của U

→ Bỏ qua thao tác ghi của T



Hình 3.26: Giao tác ghi dữ liệu bị hủy

- Nhưng U hủy
 - Giá trị của X được ghi bởi U bị mất
 - Cần khôi phục lại giá trị X trước đó
- Và T đã kết thúc
 - X có thể khôi phục từ thao tác ghi của T
- Do qui tắc ghi Thomas
 - Thao tác ghi đã được bỏ qua
 - Quá trễ để khôi phục X
- Khi T muốn ghi
 - Cho T thử ghi và sẽ bỏ nếu T hủy
 - Sao lưu giá trị cũ của X và nhãn WT(X) trước đó
- Tóm lại
 - Khi có yêu cầu đọc và ghi từ giao tác T
 - Bộ lập lịch sẽ
 - Đáp ứng yêu cầu
 - Hủy T và khởi tạo lại T với 1 timestamp mới
 - T rollback
 - Trì hoãn T, sau đó mới quyết định phải hủy T hoặc đáp ứng yêu cầu

Read(T,X)

If $WT(X) \leq TS(T)$

 Read(X); // Cho phép đọc x

$RT(X) := \max(RT(X), TS(T));$

Else

 Rollback{T};

 // Hủy bỏ T và khởi tạo lại TS

Write(T,X)	If RT(X) <= TS(T) If WT(X) <= TS(T) Write(X) ; // Cho phép ghi x WT(X):=TS(T); Else Rollback{T}; // Hủy bỏ T và khởi tạo lại TS
-------------------	--

Ví dụ 1:

	T₁ TS(T₁)=100	T₂ TS(T₂)=200	A RT(A)=0 WT(A)=0	B RT(B)=0 WT(B)=0	C RT(C)=0 WT(C)=0	Ghi chú
1	Read(A)		RT(A)=100 WT(A)=0			WT(A)<TS(T1): T1 đọc được A
2		Read(B)		RT(B)=200 WT(B)=0		WT(B)<TS(T2): T2 đọc được B
3	Write(A)		RT(A)=100 WT(A)=100			
4		Write(B)		RT(B)=200 WT(B)=200		
5		Read(C)			RT(C)=200 WT(C)=0	WT(C)<TS(T2): T2 đọc được C
6	Read(C)				RT(C)=200 WT(C)=0	WT(C)<TS(T1): T1 đọc được C
7	Write(C) ↓ Abort					RT(C)<TS(T1): T1 không ghi lên C được

Ví dụ 2:

	T₁ TS(T ₁)=200	T₂ TS(T ₂)=150	T₃ TS(T ₃)=175	A RT(A)=0 WT(A)=0	B RT(B)=0 WT(B)=0	C RT(C)=0 WT(C)=0
1	Read(B)				RT(B)=200 WT(B)=0	
2		Read(A)		RT(B)=150 WT(B)=0		
3			Read(C)			RT(C)=175 WT(C)=0
4	Write(B)				RT(B)=200 WT(B)=200	
5	Write(A)			RT(A)=150 WT(A)=200		
6		Write(C)				
7		↓ ↓	Write(A) ↓			

Rollback

Giá trị của A đã sao lưu bởi T1

→ T3 không bị rollback và không cần ghi A

Ví dụ 3:

	T₁ TS(T ₁)=150	T₂ TS(T ₂)=200	T₃ TS(T ₃)=175	T₄ TS(T ₄)=255	A RT(A)=0 WT(A)=0
1	Read(A)				RT(B)=150 WT(B)=0
2	Write(A)				RT(B)=150 WT(B)=150
3		Read(A)			RT(B)=200 WT(B)=150
4		Write(A)			RT(B)=200 WT(B)=200
5			Read(A)		
6			↓ ↓	Read(A)	RT(B)=255 WT(B)=200
7			↓ Rollback		

❖ Nhận xét

- Thao tác $\text{read}_3(A)$ làm cho giao tác T_3 bị hủy
- T_3 đọc giá trị của A sẽ được ghi đè trong tương lai bởi T_2
- Giả sử nếu T_3 đọc được giá trị của A do T_1 ghi thì sẽ không bị hủy

III.3.4 Nhận thời gian nhiều phiên bản

- Ý tưởng
 - Cho phép thao tác $\text{read}_3(A)$ thực hiện
- Bên cạnh việc lưu trữ giá trị hiện hành của A , ta giữ lại các giá trị được sao lưu trước kia của A (phiên bản của A)
- Giao tác T sẽ đọc được giá trị của A ở 1 phiên bản thích hợp nào đó
- Mỗi phiên bản của 1 đơn vị dữ liệu X có
 - $\text{RT}(X)$
 - Ghi nhận lại giao tác sau cùng đọc X thành công
 - $\text{WT}(X)$
 - Ghi nhận lại giao tác sau cùng ghi X thành công
- Khi giao tác T phát ra yêu cầu thao tác lên X
 - Tìm 1 phiên bản thích hợp của X
 - Đảm bảo tính khả thi tự
- Một phiên bản mới của X sẽ được tạo khi hành động ghi X thành công

Giải thuật

Read(T,X)

i = “số thứ tự phiên bản sau cùng nhất của X”

While $\text{WT}(X_i) > \text{TS}(T)$

$i := i-1;$

$\text{Read}(X_i);$

$\text{RT}(X_i) := \max(\text{RT}(X_i), \text{TS}(T));$

Write(T,X)

i = “số thứ tự phiên bản sau cùng nhất của X”

While $WT(X_i) > TS(T)$

i := i-1;

If $RT(X_i) > TS(T)$

Rollback

Else

Tạo phiên bản X_{i+1} ;

Write (X_{i+1});

$RT(X_{i+1}) = 0$;

$WT(X_{i+1}) = TS(T)$;

Ví dụ:

	T₁ TS(T₁)=150	T₂ TS(T₂)=200	T₃ TS(T₃)=175	T₄ TS(T₄)=255	A₀ RT=0 WT=0	A₁	A₂
1	Read(A)				RT=150 WT=0		
2	Write(A)					RT=0 WT=150	
3		Read(A)				RT=200 WT=150	
4		Write(A)					RT=0 WT=200
5			Read(A)			RT=200 WT=150	
6				Read(A)			RT=255 WT=200

❖ **Nhận xét**

● Thao tác đọc

- Giao tác T chỉ đọc giá trị của phiên bản do T hay những giao tác trước T cập nhật
- T không đọc giá trị của các phiên bản do các giao tác sau T cập nhật

→ Thao tác đọc không bị rollback

● Thao tác ghi

- Thực hiện được bằng cách chèn thêm phiên bản mới
- Không thực hiện được thì rollback

- Tốn nhiều chi phí tìm kiếm, tốn bộ nhớ
- Nên giải phóng các phiên bản quá cũ không còn được các giao tác sử dụng

❖ Câu hỏi (bài tập) củng cố:

Bài 1: Cho lịch thao tác S₁₉ sau:

	T ₁	T ₂	T ₃	T ₄
1		Read(A)		
2			Read (A)	
3		Write(B)		
4			Write(A)	
5	Read(B)			
6				Read (B)
7	Read (A)			
8	Write(C)			
9				Write(A)

Dùng kỹ thuật timestamp **riêng phần** để điều khiển truy xuất đồng thời của 4 giao tác trên, với timestamp của các giao tác T₁, T₂, T₃, T₄ lần lượt là:

- 300, 310, 320, 330
- 250, 200, 210, 275

Bài 2: Cho lịch thao tác S₂₀ sau:

	T ₁	T ₂	T ₃	T ₄	T ₅
1.	Read (A)				
2.			Write (A)		
3.		Read (B)			
4.				Read (A)	
5.					Write (B)
6.		Write (A)			
7.	Write (C)				
8.		Read (C)			
9.			Read (C)		
10.				Write (B)	
11.					Write (C)

Dùng kỹ thuật timestamp **riêng phần** để điều khiển việc truy xuất đồng thời của các giao tác biết các nhãn thời gian của các giao tác là T₁=100, T₂=300, T₃=200, T₄=400, T₅=500.

Bài 3: Cho lịch thao tác S₂₁ sau:

	T ₁	T ₂	T ₃	T ₄
1.	Rlock A			
2.		Rlock A		
3.			Wlock B	
4.				Wlock A
5.	Rlock B			
6.				Wlock C
7.		Wlock B		
8.			Wlock A	

Biết các nhãn thời gian của các giao tác là T₁ = 100, T₂ = 200, T₃ = 300, T₄ = 400.

Hãy điều khiển việc truy xuất đồng thời của các giao tác dùng:

- a. Kỹ thuật nhãn thời gian **toàn phần**
- b. Kỹ thuật nhãn thời gian **riêng phần**

Bài 4: Cho lịch thao tác S₂₂ sau:

	T ₁	T ₂	T ₃	T ₄
1.	RL(A)			
2.			RL(B)	
3.	RL(C)			
4.		WL(C)		
5.				WL(C)
6.				WL(A)
7.			WL(A)	
8.		RL(B)		
9.			WL(B)	

Thực hiện với kỹ thuật nhãn thời gian **nhiều phiên bản** với

Các nhãn thời gian của các giao tác như sau: T₁=100, T₂=300, T₃=200, T₄=400

CHƯƠNG 4

KHÔI PHỤC DỮ LIỆU

❖ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

- Trình bày khái niệm về an toàn dữ liệu
- Trình bày các chiến lược khôi phục dữ liệu
- Cài đặt chế độ an toàn dữ liệu

IV.1 BẢO MẬT DỮ LIỆU / AN TOÀN DỮ LIỆU TRONG SQL SERVER

IV.1.1 Khái niệm cơ bản về bảo mật

Nhằm bảo vệ hệ thống CSDL không bị xâm nhập, người quản trị cơ sở dữ liệu phải quyết định cho phép hay không cho phép người dùng truy cập và thao tác trên cơ sở dữ liệu dựa vào nhiệm vụ của người dùng trên hệ CSDL. Người quản trị thường dựa trên nền tảng lý thuyết bảo mật của hệ CSDL đa người dùng, nhằm tìm ra phương pháp bảo mật theo đúng với nhu cầu của bảo mật dữ liệu.

Với mục đích tăng tính bảo mật dữ liệu, SQL Server hỗ trợ các tính năng cho phép người quản trị thiết lập cơ chế bảo vệ CSDL trong môi trường đa người dùng, bao gồm các yếu tố chính sau:

- o Vai trò của người dùng trong hệ thống và cơ sở dữ liệu.*
- o Quyền sử dụng các ứng dụng cơ sở dữ liệu trong SQL Server.*
- o Quyền tạo và sửa đổi cấu trúc các đối tượng CSDL.*
- o Quyền truy cập, xử lý dữ liệu.*

Khi đăng nhập vào một hệ thống CSDL đa người dùng, người sử dụng cần phải cung cấp tài khoản (UserID) và mật khẩu (Password). Dựa trên UserID hệ thống có khả năng kiểm soát tất cả các hành vi của người sử dụng trên CSDL SQL Server.

Để thực hiện được chức năng này, người quản trị CSDL cần phải thiết lập các quyền xử lý và truy cập vào CSDL khi tạo ra UserID, ngoài ra còn có một số thuộc tính khác của SQL Server như quyền backup dữ liệu, trao đổi dữ liệu với các ứng dụng CSDL khác,...

Khi nói đến bảo mật, người quản trị cần quan tâm đến các thông tin sau của người dùng:

- o Một người dùng chỉ có một UserID và một Password.
- o Thời gian có hiệu lực của mật khẩu.
- o Giới hạn chiều dài của mật khẩu.
- o Giới hạn người sử dụng theo license hay mở rộng.
- o Thông tin về người sử dụng.

Khi tạo người sử dụng, tên tài khoản cần rõ ràng, dễ hiểu dễ gọi nhớ, và không cho phép các ký tự đặc biệt, không nên có khoảng trắng.

IV.1.2 Lựa chọn bảo mật

Khi tạo ra một người dùng (login user) trong SQL Server, có 3 cách để tăng tính bảo mật cho người sử dụng đó:

- o Giao tiếp với hệ điều hành: sử dụng UserID và Password của hệ điều hành Windows để đăng nhập SQL Server. Với loại bảo mật này, người dùng truy cập vào mạng và có thể sử dụng CSDL SQL Server, đồng thời một người dùng có UserID và Password có thể sử dụng tài nguyên trên mạng.
- o Bảo mật chuẩn: với loại này, người sử dụng có UserID và Password tách rời với hệ điều hành mạng, ứng với loại bảo mật này người sử dụng chỉ có hiệu lực trong CSDL SQL Server, không thể sử dụng tài nguyên trên mạng.
- o Tổng hợp cả hai trường hợp trên: một số người dùng sử dụng quyền sử dụng trên hệ điều hành và SQL Server, một số khác chỉ sử dụng quyền truy cập vào SQL Server.

Lưu ý: Tài khoản người dùng có giá trị trên SQL Server hiện hành, khi sang một SQL Server khác phải tạo ra tài khoản người dùng trên server đó.

o SQL Server cung cấp các chức năng hay các thủ tục tạo mới và quản trị người dùng CSDL SQL Server như sau:

Sử dụng thủ tục **sp_addlogin**.

Sử dụng công cụ Enterprise Manager.

• Sử dụng thủ tục **sp_addlogin**

Cú pháp

```
sp_addlogin [ @loginame = ] 'login'  
[ , [ @passwd = ] 'password' ]  
[ , [ @defdb = ] 'database' ]  
[ , [ @deflanguage = ] 'language' ]  
[ , [ @sid = ] sid ]  
[ , [ @encryptopt = ] 'encryption_option' ]
```

Trong đó các tham số có ý nghĩa như sau:

- **@loginame:** tên tài khoản sẽ tạo.
- **@passwd:** mật khẩu cho người dùng có tài khoản trên.
- **@defdb:** CSDL mặc định khi người dùng đăng nhập vào SQL Server.
- **@deflanguage:** ngôn ngữ mặc nhiên cho người dùng SQL Server.
- **@sid:** số nhận dạng hệ thống khi người dùng đăng nhập vào.
- **@encryptopt:** khi tạo tài khoản người dùng trong CSDL, các thông tin về tài khoản, mật khẩu được lưu trữ trong bảng sysusers của CSDL Master, nếu bạn cung cấp tham số skip_encription thì không mã hoá mật khẩu trước khi lưu vào bảng sysusers, nếu không cung cấp tham số hay để trống, SQL Server sẽ mã hoá mật khẩu trước khi lưu trữ.

Ví dụ:

Tạo người dùng có tên ‘nam’, mật khẩu ‘123’, cơ sở dữ liệu mặc định ‘QLSV’

Exec sp_addlogin ‘Nam’, ‘123’, ‘QLSV’

- Thay đổi mật khẩu

```
sp_password [[ @old = ] 'old_password' ,]  
{ [ @new = ] 'new_password' }  
[ , [ @loginame = ] 'login' ]
```

IV.1.3 Quyền người dùng và quản trị quyền người dùng

Quyền của người dùng được định nghĩa như mức độ người dùng có thể hay không thể thực thi trên CSDL, quyền được chia thành 4 loại như sau:

- o Quyền truy cập vào SQL Server.
- o Quyền truy xuất vào CSDL.
- o Quyền thực hiện trên các đối tượng của CSDL.
- o Quyền xử lý dữ liệu.

Cấp phát quyền truy cập vào CSDL

Cú pháp:

```
Use db_name
Go
sp_grantdbaccess [@loginame =] 'login'
,[[@name_in_db =] 'name_in_db' [OUTPUT]]
```

Các tham số:

- **@loginame:** tài khoản của người sử dụng đăng nhập vào SQL Server
- **@name_in_db:** tạo bí danh (tên khác) của tài khoản người dùng khi truy cập vào CSDL db_name được chỉ định, nếu không chỉ rõ CSDL muốn cho phép người dùng truy cập thì người dùng được cấp quyền trên CSDL hiện hành.

Loại bỏ quyền truy cập vào CSDL db_name của người dùng

Cú pháp:

```
Use db_name
Go
sp_revokedbaccess [@loginame =] 'login'
```

Cấp phát quyền thực thi trên cơ sở dữ liệu

Sau khi cấp phát quyền cho người dùng truy cập vào CSDL, kế tiếp cho phép người dùng đó có quyền truy cập và xử lý các đối tượng trong CSDL cũng như xử lý dữ liệu trên các đối tượng đó.

Các quyền truy cập trên các đối tượng trong một CSDL:

STT	Quyền	Điễn giải
1	SELECT	Cho phép người sử dụng nhìn thấy dữ liệu, nếu người sử dụng có quyền này thì họ chỉ có thể thực thi những phát biểu select để truy vấn dữ liệu trên các bảng hay các view được cho phép.
2	INSERT	Cho phép người sử dụng thêm dữ liệu, nếu người sử dụng có quyền này, họ có thể thực hiện phát biểu Insert, đối với một số hệ thống CSDL khác, muốn thực thi phát biểu Insert, người sử dụng phải có quyền Select, trong SQL Server quyền Insert không liên quan đến quyền truy vấn Select.
3	UPDATE	Quyền này cho phép người sử dụng chỉnh sửa dữ liệu bằng phát biểu Update.
4	DELETE	Quyền này cho phép người sử dụng xóa dữ liệu bằng phát biểu Delete.
5	REFERENCE	Cho phép người sử dụng thêm dữ liệu vào bảng có khóa ngoại bằng phát biểu Insert, trong SQL Server quyền Insert không liên quan đến quyền truy vấn Select.
6	EXECUTE	Quyền này cho phép người sử dụng thực thi các thủ tục (SP) trong CSDL.

Thủ tục cấp quyền GRANT

Cú pháp:

```
GRANT ALL | < PERMISSION> [, ...n]  
ON  
<table hoặc view name> [(<column name> [, ... n])]  
| <stored hoặc extension procedure name>  
TO  
<login or role name> [, ... n]  
[WITH GRANT OPTION]  
[AS <role name>]
```

Trong đó, các tham số như sau:

- **ALL** cho phép người sử dụng có tất cả các quyền.
- **PERMISSION** là một trong các quyền: SELECT, INSERT, UPDATE, DELETE, REFERENCE, EXECUTE. Chỉ rõ những bảng dữ liệu, view hoặc thủ tục nào cho phép người dùng truy cập và xử lý.

Từ chối quyền truy vấn và xử lý dữ liệu

Cú pháp:

```
DENY ALL | < PERMISSION> [, ...n]  
ON  
<table hoặc view name> [(<column name> [, ... n])]  
| <stored hoặc extension procedure name>  
TO  
<login or role name> [, ... n]  
[Cascade]
```

Loại bỏ quyền truy vấn và xử lý dữ liệu

```
REVOKE [ GRANT OPTION FOR ]  
    { ALL [ PRIVILEGES ] | permission [ ,...n ] }  
    {  
        [ ( column [ ,...n ] ) ] ON { table | view }  
        | ON { table | view } [ ( column [ ,...n ] ) ]  
        | ON { stored_procedure | extended_procedure }  
        | ON { user_defined_function }  
    }  
    { TO | FROM }  
    security_account [ ,...n ]  
    [ CASCADE ]  
[ AS { group | role } ]
```

Quyền tạo đối tượng trong CSDL

Trong CSDL có một số đối tượng và các chức năng như sao lưu dữ liệu mà mỗi người sử dụng trên CSDL tùy theo chức năng và nhiệm vụ cụ thể được phép hay không được phép tạo các đối tượng như table, view, stored procedure, ... và tạo CSDL.

Các quyền tạo các đối tượng như sau:

- *Create Database.*
- *Create Table.*
- *Create View.*
- *Create Procedure.*
- *Create Rule.*
- *Create Default.*
- *Backup Database.*
- *Backup Log.*

Để phân quyền tạo đối tượng trong CSDL cho người dùng, trong SQL Server có thể sử dụng thủ tục **GRANT** như sau:

GRANT < ALL | Statement [, ... n]> TO <login ID> [, ... n]

IV.1.4 Vai trò của người sử dụng trong SQL Server và cơ sở dữ liệu

IV.1.4.1 Vai trò trên SQL Server

Vai trò (Role)	Diễn giải
sysadmin	Có các quyền tương đương với sa.
serveradmin	Cấu hình một số tham số và tắt server.
setupadmin	Bị giới hạn bót một số chức năng liên kết server và khởi động một số thủ tục.
securityadmin	Quản lý người dùng và tạo CSDL.
processadmin	Được phép dùng các giao tác đang thực hiện trên CSDL và một số quá trình thực hiện khác của SQL Server.
dbcreator	Được phép tạo CSDL.
Diskadmin	Quản lý các tập tin liên quan đến CSDL SQL Server.

IV.1.4.2 Vai trò trên CSDL

Vai trò	Diễn giải
db_owner	Với vai trò này, người sử dụng (NSD) thuộc nhóm sở hữu CSDL mới có thể truy cập vào CSDL.
db_accessadmin	Thực hiện các chức năng giống như securityadmin
db_datareader	NSD được phép select trên các bảng dữ liệu của các người dùng khác trong CSDL.
db_datawriter	NSD được phép insert, update, delete trên các bảng dữ liệu của các người dùng khác trong CSDL.
db_ddladmin	NSD có thể thêm hay chỉnh sửa các đối tượng của CSDL.
db_securityadmin	NSD có quyền tương đương với quyền của

Vai trò	Diễn giải
	securityadmin.
db_backupoperator	NSD có thể thực hiện chức năng backup dữ liệu.
db_denydareader	Không cho phép sử dụng phát biểu SELECT trên tất cả các bảng dữ liệu của CSDL.
db_denydawriter	Không cho phép sử dụng phát biểu INSERT, UPDATE, DELETE trên tất cả các bảng dữ liệu của CSDL.

Sử dụng các thủ tục hệ thống để tạo một role mới, thêm một người dùng vào một role, loại bỏ người sử dụng ra khỏi một role.

- **Tạo một role**

Cú pháp:

```
sp_addrole [ @rolename = ] 'role_name'
[ , [ @ownername = ] 'owner' ]
```

Trong đó:

@rolename: tên role mới

@ownername: chủ sở hữu của role mới, mặc định là dbo

Sau khi tạo role mới cần phải gán một số quyền truy cập và xử lý trên các bảng dữ liệu nào đó trong CSDL cho role mới đó.

- **Thêm người sử dụng vào Role**

Cú pháp:

```
sp_addrolemember [ @rolename = ] 'role_name' ,
[ @membername = ] 'login_ID'
```

- **Loại bỏ người sử dụng ra khỏi một role**

Cú pháp:

```
sp_droprolemember [ @rolename = ] 'role_name' ,
[ @membername = ] 'login_ID '
```

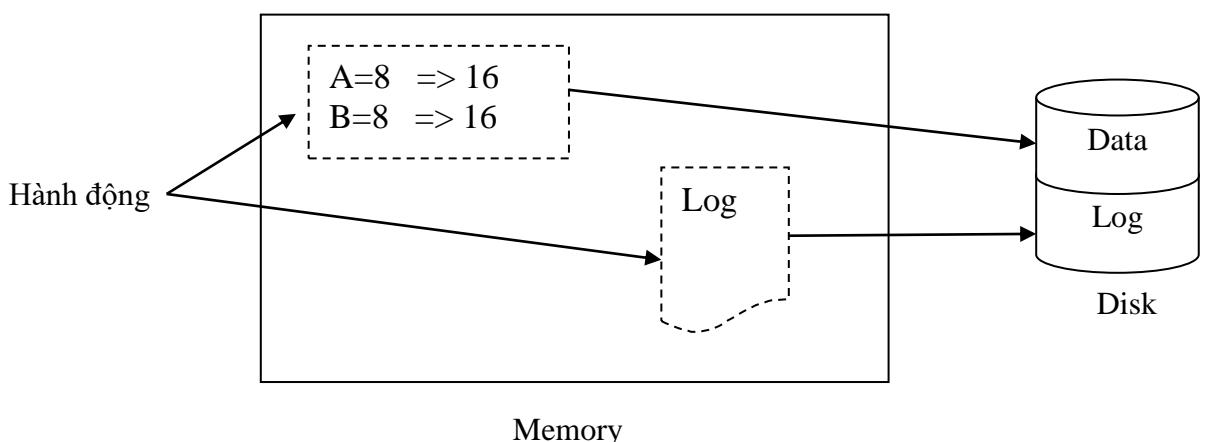
IV.2 KHÔI PHỤC DỮ LIỆU

IV.2.1 Mục tiêu của khôi phục sự cố

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

IV.2.2 Nhật ký giao tác

- 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.



Hình 4.1: Nhật ký giao tác

- 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 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 thực hiện

IV.2.3 Điểm lưu trữ

- Quá trình tra cứu nhật ký mất nhiều thời gian
 - Do phải quét hết tập tin nhật ký
- Thực hiện lại các giao tác đã được ghi xuống đĩa làm cho việc phục hồi diễn ra lâu hơn do đó cần có điểm check point
 - 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 tắt cả những gì thay đổi của CSDL từ vùng đệm xuống đĩa.

IV.2.3.1 Điểm lưu trữ đơn giản

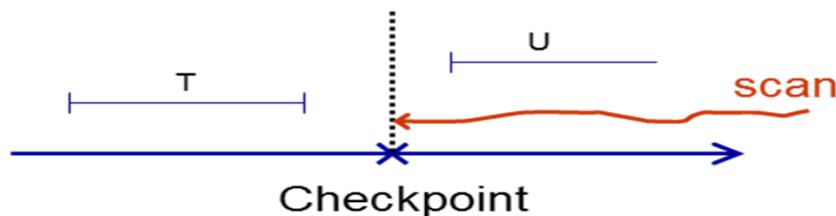
- Khi đến điểm lưu trữ, DBMS
 - (1) Tạm dừng tiếp nhận các giao tác mới
 - (2) Đợi các giao tác đang thực hiện
 - Hoặc là hoàn tất (commit)
 - Hoặc là hủy bỏ (abort)
 - và ghi mẫu tin <commit T> hay <abort T> vào nhật ký
 - (3) Tiến hành ghi nhật ký từ vùng đệm xuống đĩa

(4) Tạo 1 mẫu tin <checkpoint> và ghi xuống đĩa

(5) Tiếp tục nhận các giao tác mới

- Các giao tác ở phía trước điểm lưu trữ là những giao tác đã kết thúc
→ không cần làm lại
- Và sau điểm lưu trữ là những giao tác chưa thực hiện xong → cần khôi phục
- Không phải duyệt hết nhật ký

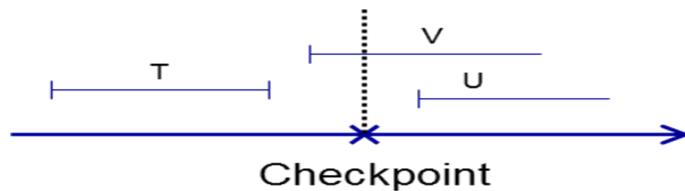
Duyệt từ cuối nhật ký đến điểm lưu trữ



Hình 4.2: Điểm lưu trữ đơn giản

IV.2.3.2 Điểm lưu trữ linh động (Nonquiescent checkpoint)

- Trong thời gian checkpoint hệ thống gần như tạm ngưng hoạt động để chờ các giao tác hoàn tất hoặc hủy bỏ. Nhằm tăng tốc độ xử lý dữ liệu thì cần *Nonquiescent checkpoint*.
- Cho phép tiếp nhận các giao tác mới trong quá trình checkpoint.
- Mẫu tin <start ckpt (T₁, T₂, ..., T_k)>
- Mẫu tin <end ckpt>



Hình 4.3: Điểm lưu trữ linh động

- Khi đến điểm lưu trữ, DBMS

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

- T₁, T₂, ..., T_k là những giao tác đang thực thi

(2) Chờ cho đến khi T_1, T_2, \dots, T_k hoàn tất hay hủy bỏ, nhưng không ngăn các giao tác mới bắt đầu

(3) Khi T_1, T_2, \dots, T_k thực hiện xong, tạo mẫu tin `<end ckpt>` và ghi xuống đĩa

IV.2.4 Phương pháp khôi phục

IV.2.4.1 Phương pháp Undo-Logging

❖ Qui tắc

(1) Một thao tác phát sinh ra 1 mẫu tin nhật ký

- Mẫu tin của thao tác cập nhật chỉ ghi nhận lại giá trị cũ
- $\langle T, X, v \rangle$

(2) Trước khi X được cập nhật xuống đĩa, mẫu tin $\langle T, X, v \rangle$ đã phải có trên đĩa

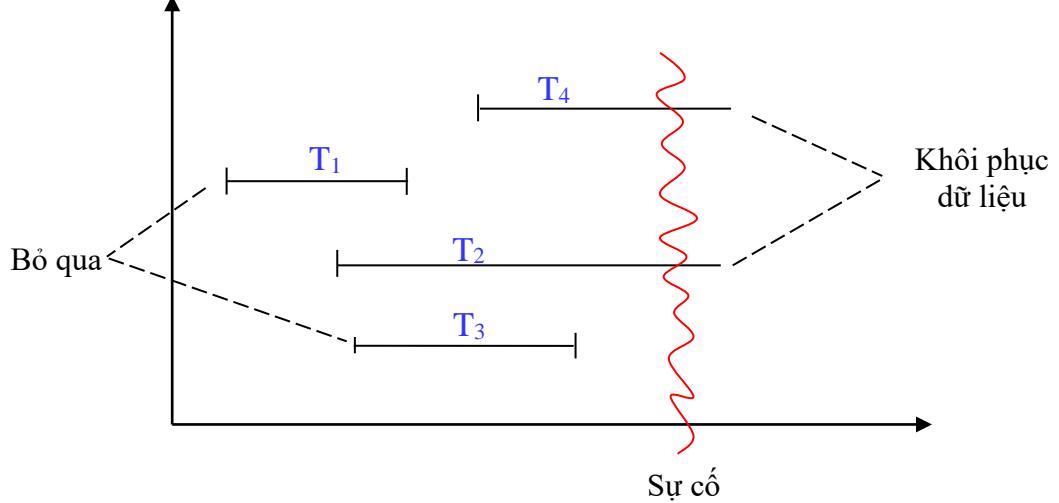
(3) Trước khi mẫu tin $\langle \text{commit}, T \rangle$ được **ghi** xuống đĩa, tất cả các cập nhật của T đã được phản ánh lên đĩa

- **Flush-log:** chỉ chép những block mẫu tin nhật ký mới chưa được chép trước đó

Ví dụ

Bước	Hành động	T	MemA	MemB	Disk A	Disk B	Mem Log
1							<code><start T></code>
2	Read(A,t)	8	8		8	8	
3	$t := t^2$	16	8		8	8	
4	Write(A,t)	16	16		8	8	<code><T,A,8></code>
5	Read(B,t)	8	8	8	8	8	
6	$t := t^2$	16	8	8	8	8	
7	Write(B,t)	16	16	16	8	8	<code><T,B,8></code>
8	Flush log						
9	Output(A)	16	16	16	16	8	
10	Output(B)	16	16	16	16	16	
11							<code><Commit></code>
12	Flush log						

- Khôi phục
 - (1) 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
 - (2) 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
 - Write(X, v)
 - Output(X)
 - (3) Với mỗi $T_i \in S$
 - Ghi mẫu tin $\langle \text{abort } T_i \rangle$ lên nhật ký
- Khi có sự cố
 - T_1 và T_3 đã hoàn tất
 - T_2 và T_4 chưa kết thúc



Hình 4.4: Khôi phục theo phương pháp Undo-Logging

Ví dụ:

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>
5	Read(B,t)	8	8	8	8	8	
6	t:=t*2	16	8	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T,B,8>
8	Flush log						A và B không thay đổi nên không cần khôi phục
9	Output(A)	16	16	16	16	8	
10	Output(B)	16	16	16	16	16	
11							<Commit>
12	Flush log						Không cần khôi phục A và B

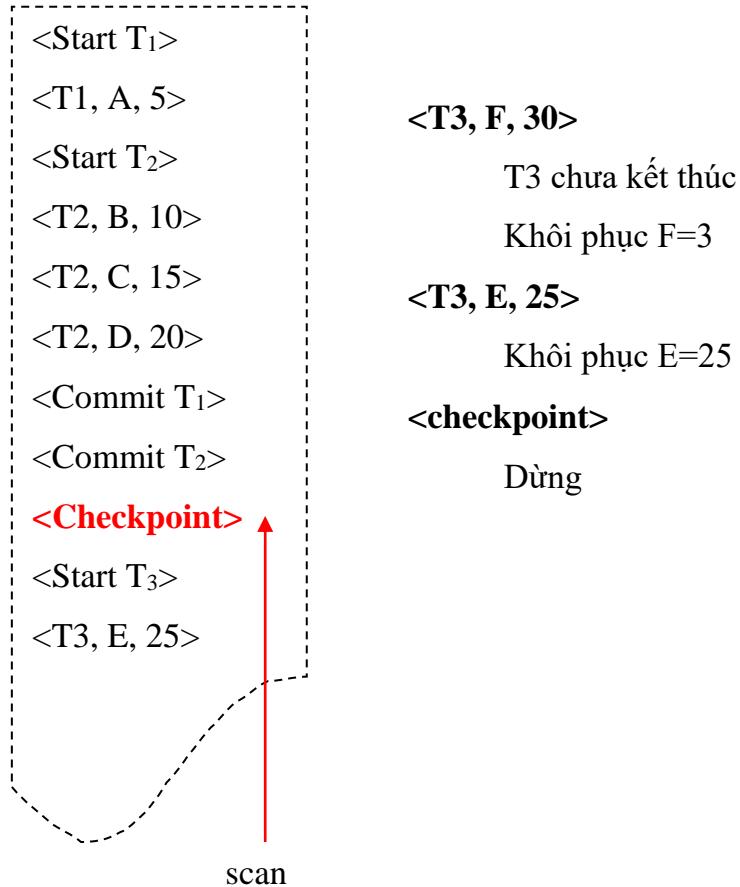
❖ Undo-Logging & Checkpoint

<start T₁>
 <T₁, A, 5>
 <start T₂>
 <T₁, B, 10>
 Checkpoint →

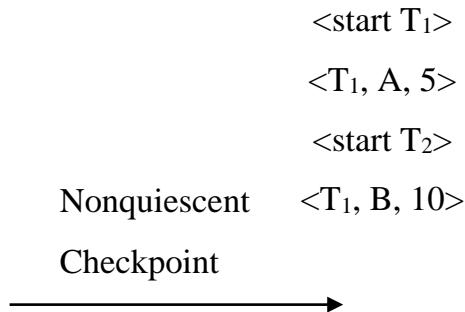
Hình 4.5: Undo-logging và checkpoint đơn giản

- 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

Ví dụ



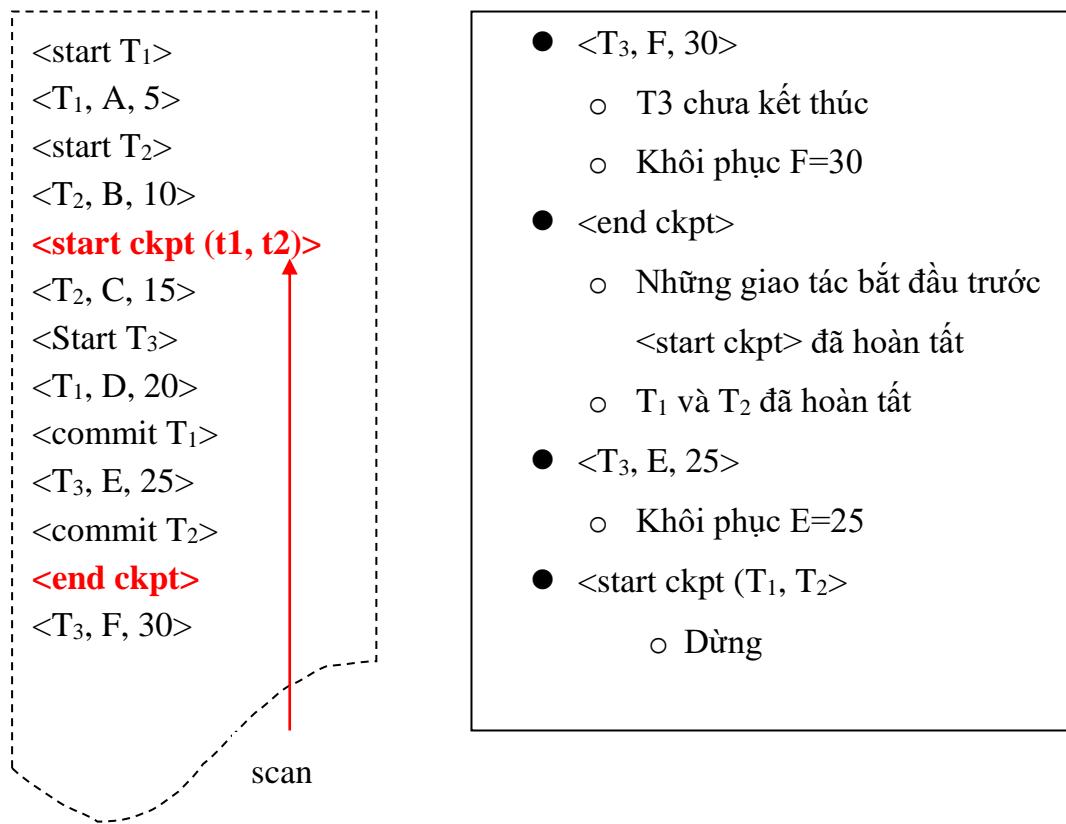
❖ Undo-logging và checkpoint linh động



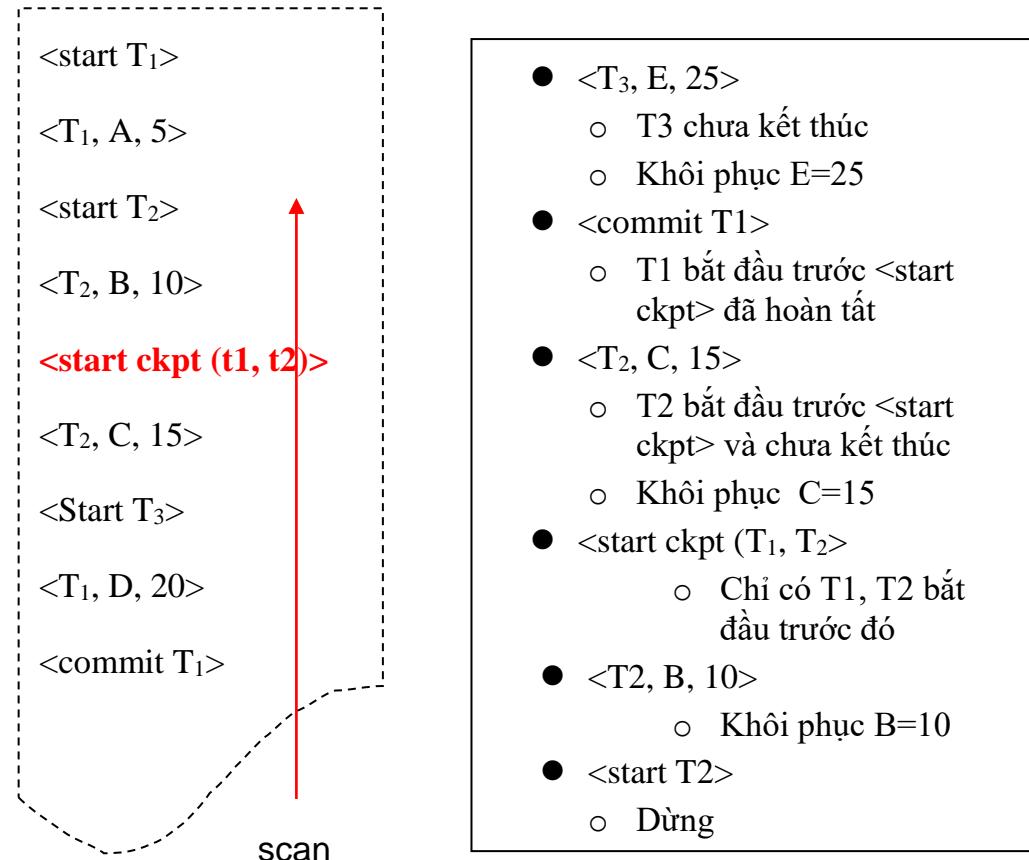
Hình 4.6: Undo-logging và checkpoint linh động

- 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

Ví dụ 1



Ví dụ 2



IV.2.4.2 Phương pháp Redo-Logging

- Qui tắc

(1) Một thao tác phát sinh ra 1 mẫu tin nhật ký

- Mẫu tin của thao tác cập nhật chỉ ghi nhận lại giá trị mới
- $\langle T, X, w \rangle$

(2) Trước khi X được cập nhật xuống đĩa, tất cả các mẫu tin nhật ký của giao tác cập nhật X đã phải có trên đĩa

- Bao gồm mẫu tin cập nhật $\langle T, X, w \rangle$ và $\langle \text{commit } T \rangle$

(3) Khi T hoàn tất, tiến hành ghi nhật ký xuống đĩa

- **Flush-log**: chỉ chép những block mẫu tin nhật ký mới chưa được chép trước đó

Ví dụ

Bước	Hành động	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	$\langle T, A, 16 \rangle$
5	Read(B,t)	8	8	8	8	8	
6	$t := t^2$	16	8	8	8	8	
7	Write(B,t)	16	16	16	8	8	$\langle T, B, 16 \rangle$
8							$\langle \text{Commit} \rangle$
9	Flush log						
10	Output(A)	16	16	16	16	8	
11	Output(B)	16	16	16	16	16	

- Khôi phục

(1) Gọi S là tập các giao tác hoàn tất

Có mẫu tin $\langle \text{commit } T_i \rangle$ trong nhật ký

(2) Với mỗi mẫu tin $\langle T_i, X, w \rangle$ trong nhật ký

(theo thứ tự cuối tập tin đến đầu tập tin)

Nếu $T_i \in S$ thì

- Write(X, w)

- Output(X)

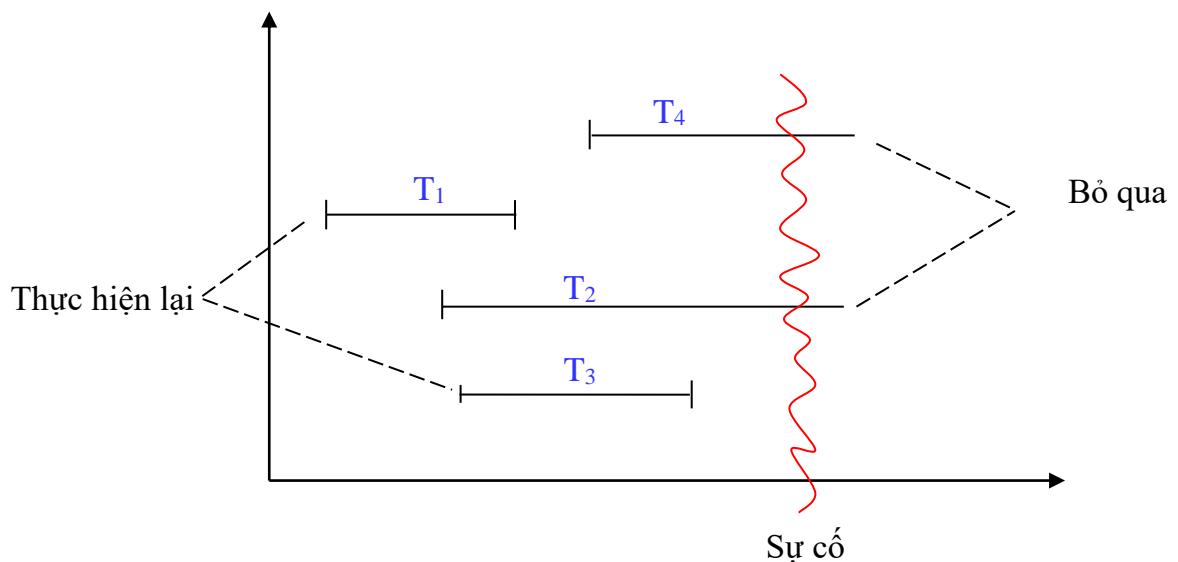
(3) Với mỗi $T_j \notin S$

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

- Khi có sự cố

o T_1 và T_3 đã hoàn tất

o T_2 và T_4 chưa kết thúc



Hình 4.7: Khôi phục theo phương pháp Redo-Logging

Ví dụ:

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	8	8	8	8	
6	t:=t*2	16	8	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T,B,16>
8							<Commit>
9	Flush log						
10	Output(A)	16	16	16	16	8	Thực hiện lại T, ghi A=16; B=16
11	Output(B)	16	16	16	16	16	Thực hiện lại T, ghi A=16; B=16

Redo-Logging & Checkpoint

- Nhận xét

- Phương pháp Redo thực hiện ghi dữ liệu trễ so với thời điểm hoàn tất của các giao tác
- <start ckpt>
 - Thực hiện ghi xuống đĩa những dữ liệu đã hoàn tất mà trước đó chưa được ghi
- <end ckpt>
- Mẫu tin <end ckpt> được ghi vào nhật ký mà không phải đợi các giao tác hoàn tất (commit) hoặc hủy bỏ (abort)

- Đến điểm lưu trữ, DBMS

- (1) Tạo mẫu tin <start ckpt (T₁, T₂, ..., T_k)> và ghi xuống đĩa
 - T₁, T₂, ..., T_k là những giao tác đang thực thi

- o (2) Ghi xuống đĩa những dữ liệu của các giao tác đã hoàn tất trên vùng đệm
- o (3) Tạo mẫu tin `<end ckpt>` và ghi xuống đĩa

Ví dụ 1:

```

<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>

```

- T₁ đã hoàn tất trước `<Start ckpt>`
 - o Có thể đã được ghi xuống đĩa
 - o Nếu chưa thì trước khi `<end ckpt>` cũng được ghi xuống đĩa
- Sau `<Start ckpt>`
 - o T₂ đang thực thi
 - o T₃ bắt đầu
- Sau `<End ckpt>`
 - o T₂ và T₃ hoàn tất

Ví dụ 2:

```

<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>

```

- Tìm thấy `<end ckpt>`
- Chỉ xét T₂ và T₃
- `<commit T2>`
 - o Thực hiện lại T₂
 - o Ghi C= 15 và B=10
- `<commit T3>`
 - o Thực hiện lại T₃
 - o Ghi D=20

↑
scan

IV.2.4.3 Phương pháp Undo/Redo-Logging

❖ Qui tắc

(1) Một thao tác phát sinh ra 1 mẫu tin nhật ký

- Mẫu tin của thao tác cập nhật ghi nhận giá trị cũ và mới của một đơn vị dữ liệu
- $\langle T, X, v, w \rangle$

(2) Trước khi X được cập nhật xuống đĩa, các mẫu tin cập nhật $\langle T, X, v, w \rangle$ đã phải có trên đĩa

(3) Khi T hoàn tất, tạo mẫu tin $\langle \text{commit } T \rangle$ trên nhật ký và ghi xuống đĩa

Ví dụ

Bước	Hành động	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	$\langle T, A, 8, 16 \rangle$
5	Read(B,t)	8	8	8	8	8	
6	$t := t^2$	16	8	8	8	8	
7	Write(B,t)	16	16	16	8	8	$\langle T, B, 8, 16 \rangle$
8	Flush log						
9	Output(A)	16	16	16	16	8	
10							$\langle \text{Commit} \rangle$
11	Output(B)	16	16	16	16	16	

● Khôi phục

(1) Khôi phục lại (undo) những giao tác chưa kết thúc

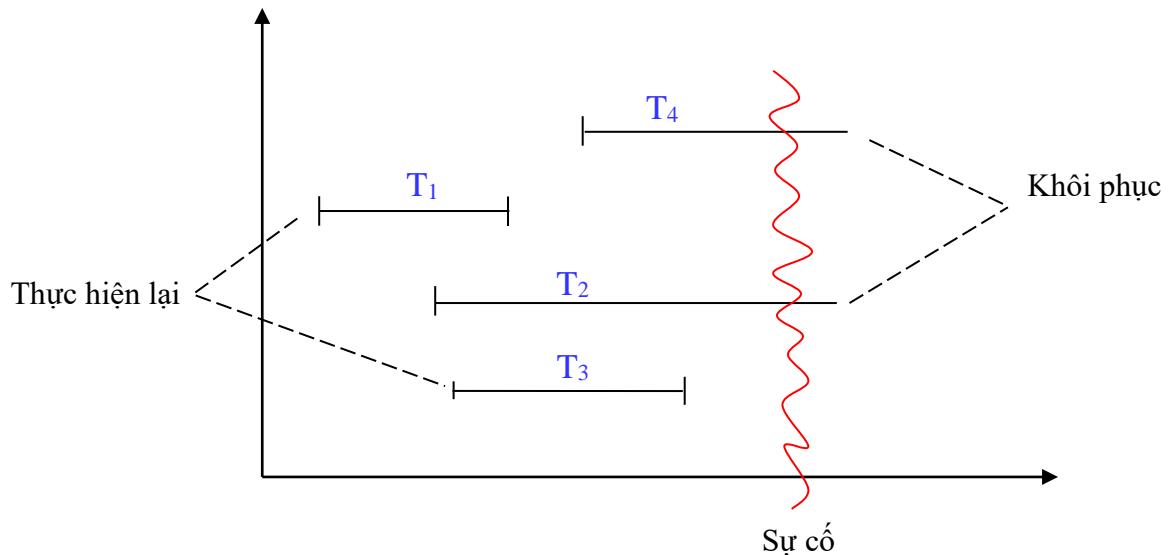
- Theo thứ tự từ cuối nhật ký đến đầu nhật ký

(2) Thực hiện lại (redo) những giao tác đã hoàn tất

- Theo thứ tự từ đầu nhật ký đến cuối nhật ký

● Khi gặp sự cố

- T_1 và T_3 đã hoàn tất
- T_2 và T_4 chưa kết thúc



Hình 4.8: Khôi phục theo phương pháp Undo/Redo-Logging

Ví dụ:

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	8	8	8	8	
6	$t:=t^2$	16	8	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	T Chưa kết thúc khôi phục A=8
10							<commit>
11	Output(B)	16	16	16	16	16	<commit T> đã được ghi xuống đĩ a thực hiện lại T, A=16 và B=16

❖ Undo/Redo-Logging & Checkpoint

- Khi đến điểm lưu trữ, DBMS

(1) Tạo mẫu tin $\langle \text{start ckpt} (T_1, T_2, \dots, T_k) \rangle$ và ghi xuống đĩa

- T_1, T_2, \dots, T_k là những giao tác đang thực thi

(2) Ghi xuống đĩa những dữ liệu đang nằm trên vùng đệm

- Những đơn vị dữ liệu được cập nhật bởi các giao tác

(3) Tạo mẫu tin $\langle \text{end ckpt} \rangle$ trong nhật ký và 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>
```

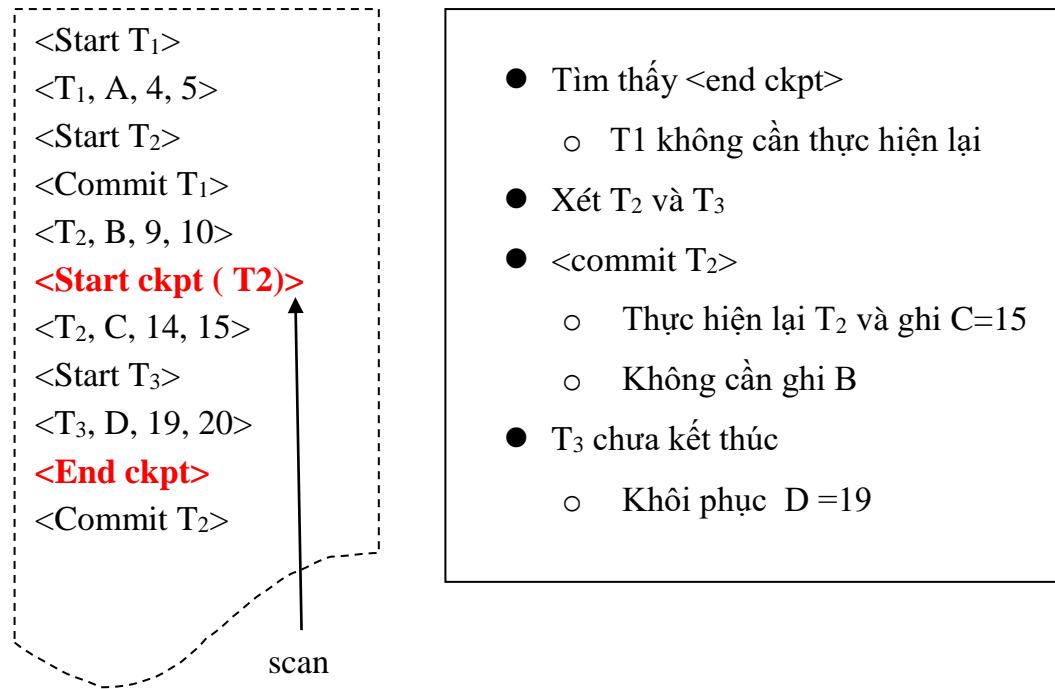
- T_1 đã hoàn tất trước $\langle \text{start ckpt} \rangle$
 - Có thể đã được ghi xuống đĩa
 - Nếu chưa thì trước khi $\langle \text{end ckpt} \rangle$ cũng được ghi xuống đĩa
- Giá trị $B=10$ đã được ghi xuống đĩa

Ví dụ 1

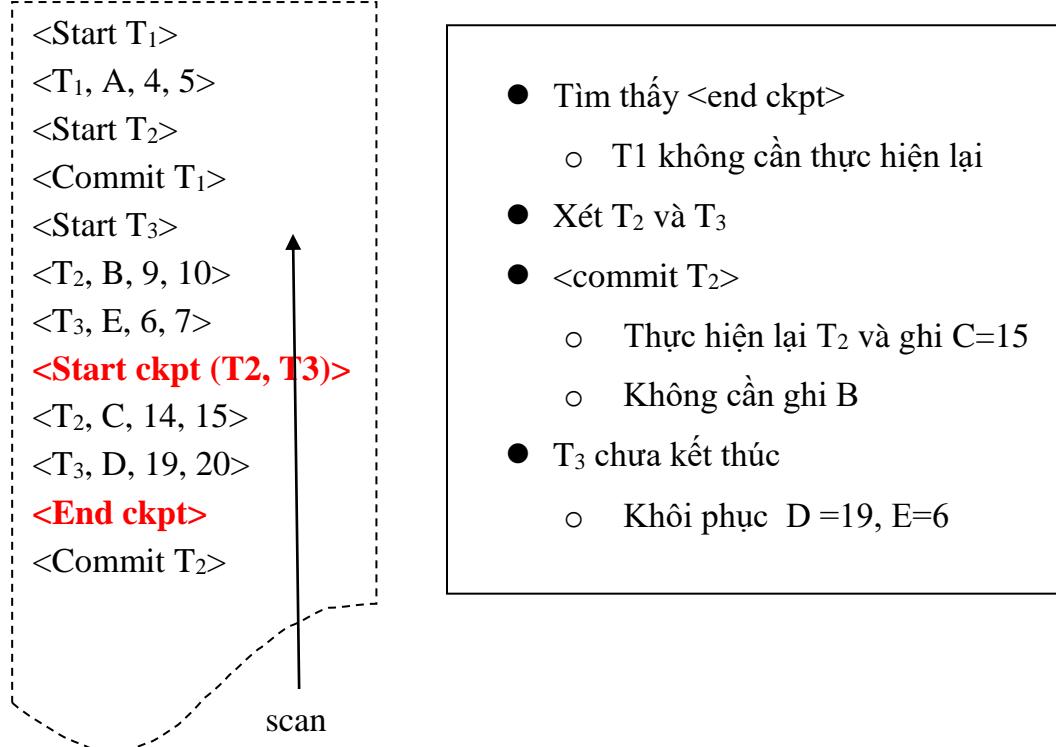
- Tìm thấy $\langle \text{end ckpt} \rangle$
 - T_1 không cần thực hiện lại
- Xét T_2 và T_3
- $\langle \text{commit } T_2 \rangle$
 - Thực hiện lại T_2 và ghi $C=15$
 - Không cần ghi B
- $\langle \text{commit } T_3 \rangle$
 - Thực hiện lại T_3 và ghi $D=20$

```
<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>
```

Ví dụ 2:



Ví dụ 3:



❖ Câu hỏi (bài tập) củng cố:

1. Trình bày khái niệm về an toàn dữ liệu
2. Trình bày các chiến lược khôi phục dữ liệu
3. Hãy nhận xét về việc ghi và truy xuất dữ liệu của hai phương pháp Undo -Logging và Redo – Logging.
4. Giả sử sau khi sự cố hệ thống xảy ra, DBMS được khởi động lại với tập tin nhật ký như sau:

```
<start T1>
<T1, A, 30>
<T1, B, 20>
<start T2>
<T2, C, 10>
<start T3>
<T3, D, 10>
<commit T3>
<T2, C, 15>
<T2, D, 20>
```

Hãy mô tả tiến trình khôi phục của DBMS dựa trên tập tin nhật ký này theo phương pháp

- a. Undo logging
- b. Redo logging.

5. Giả sử sau khi sự cố hệ thống xảy ra, DBMS được khởi động lại với tập tin nhật ký như sau:

```
<start T1>
<T1, A, 30,40>
<T1, B, 20, 10>
<start T2>
<T2, C, 10, 15>
<start T3>
<T3, D, 10, 20>
<commit T3>
<T2, C, 15, 40>
<T2, D, 20, 40>
<commit T1>
```

Hãy mô tả tiến trình khôi phục của DBMS dựa trên tập tin nhật ký này theo phương pháp **Undo/Redo logging**.

6. Giả sử sau khi sự cố hệ thống xảy ra, DBMS được khởi động lại với tập tin nhật ký như sau:

```
<start ckpt (T1, T2,T3)>
    <T1, A, 10>
    <T2, B, 20>
    <T3, C, 30>
    <commit T2>
    <T3, B, 40>
    <T1, D, 50>
<commit T3>
```

Hãy mô tả tiến trình khôi phục của DBMS dựa trên tập tin nhật ký này theo phương pháp: **Undo logging; Redo logging**

❖ Câu hỏi (bài tập) củng cố:

7. Giả sử sau khi sự cố hệ thống xảy ra, DBMS được khởi động lại với tập tin nhật ký như sau:

```
<start T1>
<T1, A, 30, 40>
<start T2>
<T2, B, 40, 60>
<T1, C, 20, 30>
<commit T2>
<start ckpt (T1)>
<start T3>
<T3, B, 60, 50>
<end ckpt>
<commit T3>
```

Hãy mô tả tiến trình khôi phục của DBMS dựa trên tập tin nhật ký này theo phương pháp **Undo/Redo logging**.

8. Thực hiện các nội dung thực hành trên máy tính đính kèm tại **PHỤ LỤC 1 →7**

CHƯƠNG 5

CẤU TRÚC LƯU TRỮ DỮ LIỆU TRÊN ĐĨA

❖ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

- Mô tả cấu trúc lưu trữ dữ liệu trên đĩa
- Trình bày các kiểu tổ chức dữ liệu trên tập tin (file)

V.1 CÁC THÀNH PHẦN LIÊN QUAN ĐẾN VIỆC QUẢN LÝ VÀ TRUY XUẤT DỮ LIỆU

Có một số kiểu lưu trữ dữ liệu trong các hệ thống máy tính. Các phương tiện lưu trữ được phân lớp theo tốc độ truy xuất, theo giá cả và theo độ tin cậy của phương tiện. Các phương tiện hiện có là:

- **Cache:** Là dạng lưu trữ nhanh nhất và cũng đắt nhất trong các phương tiện lưu trữ. Bộ nhớ cache nhỏ; sự sử dụng nó được quản trị bởi hệ điều hành.
- **Bộ nhớ chính (main memory):** Phương tiện lưu trữ dùng để lưu trữ dữ liệu sẵn sàng được thực hiện. Các chỉ thị máy mục đích chung (general-purpose) hoạt động trên bộ nhớ chính. Mặc dù bộ nhớ chính có thể chứa nhiều megabytes dữ liệu, nó vẫn là quá nhỏ (và quá đắt giá) để lưu trữ toàn bộ một cơ sở dữ liệu. Nội dung trong bộ nhớ chính thường bị mất khi mất cấp nguồn.
- **Bộ nhớ Flash:** Được biết như bộ nhớ chỉ đọc có thể lập trình, có thể xoá (EEPROM: Electrically Erasable Programmable Read-Only Memory). Bộ nhớ Flash khác bộ nhớ chính ở chỗ dữ liệu còn tồn tại trong bộ nhớ Flash khi không cấp nguồn. Đọc dữ liệu từ bộ nhớ flash mất ít hơn 100 ns, nhanh như đọc dữ liệu từ bộ nhớ chính. Tuy nhiên, viết dữ liệu vào bộ nhớ Flash phức tạp hơn nhiều. Dữ liệu được viết (một lần mất khoảng 4 đến 10 µs) nhưng không thể viết đè trực tiếp. Để viết đè bộ nhớ đã được viết, ta phải xoá trắng toàn bộ bộ nhớ sau đó mới có thể viết lên nó.
- **Lưu trữ đĩa từ (magnetic-disk):** Phương tiện căn bản để lưu trữ dữ liệu trực tuyến, lâu dài. Thường toàn bộ cơ sở dữ liệu được lưu trữ trên đĩa từ. Dữ liệu phải được chuyển từ đĩa vào bộ nhớ chính trước khi được truy nhập. Khi dữ liệu trong bộ nhớ chính này bị sửa đổi, nó phải được viết lên đĩa. Lưu trữ đĩa được xem là truy xuất trực tiếp vì có thể đọc dữ liệu trên đĩa theo một thứ tự bất kỳ. Lưu trữ đĩa vẫn tồn tại khi mất cấp nguồn. Lưu trữ đĩa có thể bị hỏng hóc, tuy không thường xuyên.

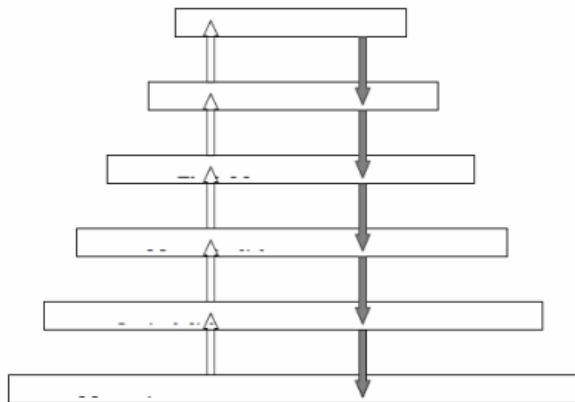
- **Lưu trữ quang (Optical storage):** Dạng quen thuộc nhất của đĩa quang học là loại đĩa CD-ROM : Compact-Disk Read-Only Memory. Dữ liệu được lưu trữ trên các đĩa quang học được đọc bởi laser. Các đĩa quang học CD-ROM chỉ có thể đọc. Các phiên bản khác của chúng là loại đĩa quang học: viết một lần, đọc nhiều lần (write-once, read-many: WORM) cho phép viết dữ liệu lên đĩa một lần, không cho phép xoá và viết lại, và các đĩa có thể viết lại (rewritable) v...v

- **Lưu trữ băng từ (Tape Storage):** Lưu trữ băng từ thường dùng để sao chép dự phòng (backup) dữ liệu. Băng từ rẻ hơn đĩa, truy xuất dữ liệu chậm hơn (vì phải truy xuất tuần tự). Băng từ thường có dung lượng rất lớn.

Các phương tiện lưu trữ có thể được tổ chức phân cấp theo tốc độ truy xuất và giá cả. Mức cao nhất là nhanh nhất nhưng cũng là đắt nhất, giảm dần xuống các mức thấp hơn.

Các phương tiện lưu trữ nhanh (cache, bộ nhớ chính) được xem như là lưu trữ sơ cấp (primary storage), các thiết bị lưu trữ ở mức thấp hơn như đĩa từ được xem như lưu trữ thứ cấp hay lưu trữ trực tuyến (on-line storage), còn các thiết bị lưu trữ ở mức thấp nhất và gần thấp nhất như đĩa quang học, băng từ kể cả các đĩa mềm được xếp vào lưu trữ tam cấp hay lưu trữ không trực tuyến (off-line).

Bên cạnh vấn đề tốc độ và giá cả, ta còn phải xét đến tính lâu bền của các phương tiện lưu trữ.



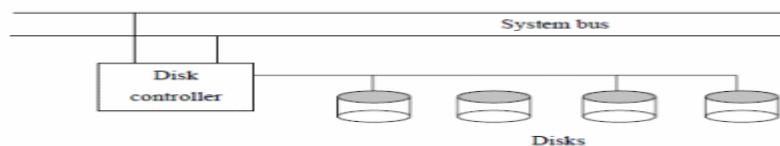
Hình 5.1: Phân cấp thiết bị lưu trữ

V.2 ĐĨA TÙ

V.2.1 Đặc trưng vật lý của đĩa

Mỗi tấm đĩa có dạng hình tròn, hai mặt của nó được phủ bởi vật liệu từ tính, thông tin được ghi trên bề mặt đĩa. Đĩa gồm nhiều tấm đĩa. Ta sẽ sử dụng thuật ngữ đĩa để chỉ các đĩa cứng.

Khi đĩa được sử dụng, một động cơ ổ đĩa làm quay nó ở một tốc độ không đổi. Một đầu đọc-viết được định vị trên bề mặt của tấm đĩa. Bề mặt tấm đĩa được chia logic thành các rãnh, mỗi rãnh lại được chia thành các sector, một sector là một đơn vị thông tin nhỏ có thể được đọc, viết lên đĩa. Tuỳ thuộc vào kiểu đĩa, sector thay đổi từ 32 bytes đến 4095 bytes, thông thường là 512 bytes. Có từ 4 đến 32 sectors trên một rãnh, từ 20 đến 1500 rãnh trên một bề mặt. Mỗi bề mặt của một tấm đĩa có một đầu đọc viết, nó có thể chạy dọc theo bán kính đĩa để truy cập đến các rãnh khác nhau. Một đĩa gồm nhiều tấm đĩa, các đầu đọc-viết của tất cả các rãnh được gắn vào một bộ được gọi là cánh tay đĩa, di chuyển cùng nhau. Các tấm đĩa được gắn vào một trục quay. Vì các đầu đọc-viết trên các tấm đĩa di chuyển cùng nhau, nên khi đầu đọc-viết trên một tấm đĩa đang ở rãnh thứ i thì các đầu đọc-viết của các tấm đĩa khác cũng ở rãnh thứ i, do vậy các rãnh thứ i của tất cả các tấm đĩa được gọi là trụ (cylinder) thứ i. Một bộ điều khiển đĩa chấp nhận các lệnh mức cao để đọc và viết một sector và khởi động các hành động như di chuyển cánh tay đĩa đến các rãnh đúng và đọc viết dữ liệu. Bộ điều khiển đĩa cũng tham gia vào checksum mỗi sector được viết. Checksum được tính từ dữ liệu được viết lên sector. Khi sector được đọc lại, checksum được tính lại từ dữ liệu được lấy ra và so sánh với checksum đã lưu trữ. Nếu dữ liệu bị sai lạc, checksum được tính sẽ không khớp với checksum đã lưu trữ. Nếu lỗi như vậy xảy ra, bộ điều khiển sẽ lặp lại việc đọc vài lần, nếu lỗi vẫn xảy ra, bộ điều khiển sẽ thông báo việc đọc thất bại. Bộ điều khiển đĩa còn có chức năng tái ánh xạ các sector xấu: ánh xạ các sector xấu đến một vị trí vật lý khác. *Hình 5.2* trình bày các đĩa được nối với một hệ thống máy tính:



Hình 5.2:Trình bày các đĩa được nối với một hệ thống máy tính

Các đĩa được nối với một hệ thống máy tính hoặc một bộ điều khiển đĩa qua một sự hợp nhất tốc độ cao. Hợp nhất hệ thống máy tính nhỏ (Small Computer-System Interconnect: SCSI) thường được sử dụng để kết nối các đĩa với các máy tính cá nhân

và workstation. Mainframe và các hệ thống server thường có các bus nhanh hơn và đắt hơn để nối với các đĩa.

Các đầu đọc-viết được giữ sát với bề mặt đĩa như có thể để tăng độ dày đặc (density).

Đĩa đầu cố định (Fixed-head) có một đầu riêng biệt cho mỗi rãnh, sự sắp xếp này cho phép máy tính chuyển từ rãnh này sang rãnh khác mau chóng, không phải di chuyển đầu đọc-viết. Tuy nhiên, cần một số rất lớn đầu đọc-viết, điều này làm nâng giá của thiết bị.

V.2.2 Đo lường hiệu năng của đĩa

Các tiêu chuẩn đo lường chất lượng chính của đĩa là dung lượng, thời gian truy xuất, tốc độ truyền dữ liệu và độ tin cậy.

- **Thời gian truy xuất (Access Time):** là khoảng thời gian từ khi yêu cầu đọc/viết được phát đi đến khi bắt đầu truyền dữ liệu. Để truy xuất dữ liệu trên một sector đã cho của một đĩa, đầu tiên cánh tay đĩa phải di chuyển đến rãnh đúng, sau đó phải chờ sector xuất hiện dưới nó, thời gian để định vị cánh tay được gọi là thời gian tìm kiếm (seek time), nó tỷ lệ với khoảng cách mà cánh tay phải di chuyển, thời gian tìm kiếm nằm trong khoảng 2...30 ms tùy thuộc vào rãnh xa hay gần vị trí cánh tay hiện tại.

- **Thời gian tìm kiếm trung bình (Average Seek Time):** Thời gian tìm kiếm trung bình là trung bình của thời gian tìm kiếm, được đo lường trên một dãy các yêu cầu ngẫu nhiên (phân phối đều) và bằng khoảng 1/3 thời gian tìm kiếm trong trường hợp xấu nhất.

- **Thời gian tiềm ẩn luân chuyển (Rotational Latency Time):** Thời gian chờ sector được truy xuất xuất hiện dưới đầu đọc/viết. Tốc độ quay của đĩa nằm trong khoảng 60...120 vòng quay/giây, trung bình cần nửa vòng quay để sector cần thiết nằm dưới đầu đọc/viết. Như vậy, thời gian tiềm ẩn trung bình (Average Latency Time) bằng nửa thời gian quay một vòng đĩa.

Thời gian truy xuất bằng tổng của thời gian tìm kiếm và thời gian tiềm ẩn và nằm trong khoảng 10...40 ms.

♦ **Tốc độ truyền dữ liệu:** Là tốc độ dữ liệu có thể được lấy ra từ đĩa hoặc được lưu trữ vào đĩa. Hiện nay tốc này vào khoảng 1...5 Mbps.

♦ **Thời gian trung bình không sự cố (Mean Time to Failure):** Lượng thời gian trung bình hệ thống chạy liên tục không có bất kỳ sự cố nào. Các đĩa hiện nay có thời gian không sự cố trung bình khoảng 30000 ... 800000 giờ nghĩa là khoảng từ 3,4 đến 91 năm.

V.2.3 Tối ưu hóa truy xuất khối đĩa (disk-block)

Yêu cầu I/O đĩa được sinh ra cả bởi hệ thống tập tin lẫn bộ quản trị bộ nhớ ảo trong hầu hết các hệ điều hành. Mỗi yêu cầu xác định địa chỉ trên đĩa được tham khảo, địa chỉ này ở dạng số khồi. Một khồi là một dãy các sector kè nhau trên một rãnh. Kích cỡ khồi trong khoảng 512 bytes đến một vài Kbytes. Dữ liệu được truyền giữa đĩa và bộ nhớ chính theo đơn vị khồi. Mức thấp hơn của bộ quản trị hệ thống tập tin sẽ chuyển đổi địa chỉ khồi sang số của trụ, của mặt và của sector ở mức phần cứng.

Truy xuất dữ liệu trên đĩa chậm hơn nhiều so với truy xuất dữ liệu trong bộ nhớ chính, do vậy cần thiết một chiến lược nhằm nâng cao tốc độ truy xuất khồi đĩa. Dưới đây ta sẽ thảo luận một vài kỹ thuật nhằm vào mục đích đó.

- **Lập lịch biểu (Scheduling):** Nếu một vài khồi của một trụ cần được truyền từ đĩa vào bộ nhớ chính, ta có thể tiết kiệm thời gian truy xuất bởi yêu cầu các khồi theo thứ tự mà nó chạy qua dưới đầu đọc/viết. Nếu các khồi mong muốn ở trên các trụ khác nhau, ta yêu cầu các khồi theo thứ tự sao cho làm tối thiểu sự di chuyển cánh tay đĩa. Các thuật toán lập lịch biểu cánh tay đĩa (Disk-arm-scheduling) nhằm lập thứ tự truy xuất các rãnh theo cách làm tăng số truy xuất có thể được xử lý. Một thuật toán thường dùng là thuật toán thang máy (elevator algorithm): Giả sử ban đầu cánh tay di chuyển từ rãnh trong nhất hướng ra phía ngoài đĩa, đối với mỗi rãnh có yêu cầu truy xuất, nó dừng lại, phục vụ yêu cầu đối với rãnh này, sau đó tiếp tục di chuyển ra phía ngoài đến tận khi không có yêu cầu nào chờ các rãnh xa hơn phía ngoài. Tại điểm này, cánh tay đổi hướng, di chuyển vào phía trong, lại dừng lại trên các rãnh được yêu cầu, và cứ như vậy đến tận khi không còn rãnh nào ở trong hơn được yêu cầu, rồi lại đổi hướng, v.v... Bộ điều khiển đĩa thường làm nhiệm vụ sắp xếp lại các yêu cầu đọc để cải thiện hiệu năng.

- **Tổ chức tập tin (Tập tin):** Để suy giảm thời gian truy xuất khồi, ta có thể tổ chức các khồi trên đĩa theo cách tương ứng gần nhất với cách mà dữ liệu được truy xuất. Ví dụ: Nếu ta muốn một tập tin được truy xuất tuần tự, khi đó ta bố trí các khồi của tập tin một cách tuần tự trên các trụ kè nhau. Tuy nhiên việc phân bổ các khồi lưu trữ kè

nhau này sẽ bị phá vỡ trong quá trình phát triển của tập tin dẫn đến tập tin không thể được phân bố trên các khối kề nhau được nữa, hiện tượng này được gọi là sự phân mảnh (Fragmentation). Nhiều hệ điều hành cung cấp tiện ích giúp suy giảm sự phân mảnh này (Defragmentation) nhằm làm tăng hiệu năng truy xuất tập tin.

- **Các bộ đệm (buffers) viết không hay đổi:** Vì nội dung của bộ nhớ chính bị mất khi mất nguồn, các thông tin về cơ sở dữ liệu cập nhật phải được ghi lên đĩa nhằm đề phòng sự cố. Hiệu năng của các ứng dụng cập nhật cường độ cao phụ thuộc mạnh vào tốc độ viết đĩa. Ta có thể sử dụng bộ nhớ truy xuất ngẫu nhiên không hay đổi (nonvolatile RAM) để nâng tốc độ viết đĩa. Nội dung của nonvolatile RAM không bị mất khi mất nguồn. Một phương pháp chung để thực hiện nonvolatile RAM là sử dụng RAM pin dự phòng (battery-back-up RAM). Khi cơ sở dữ liệu yêu cầu viết một khối lên đĩa, bộ điều khiển đĩa viết khối này lên buffer nonvolatile RAM, và thông báo ngay cho hệ điều hành là việc viết đã thành công. Bộ điều khiển sẽ viết dữ liệu đến đích của nó trên đĩa, mỗi khi đĩa rảnh hoặc buffer nonvolatile RAM đầy. Khi hệ cơ sở dữ liệu yêu cầu một viết khối, nó chỉ chịu một khoảng lặng chờ đợi khi buffer nonvolatile RAM đầy.

- **Đĩa log (log disk):** Một cách tiếp cận khác để làm suy giảm tiềm năng viết là sử dụng log-disk: Một đĩa được tận hiến cho việc viết một log tuần tự. Tất cả các truy xuất đến log-disk là tuần tự, nhằm loại bỏ thời gian tìm kiếm, và một vài khối kề có thể được viết một lần, tạo cho việc viết vào log-disk nhanh hơn viết ngẫu nhiên vài lần. Cũng như trong trường hợp sử dụng nonvolatile RAM, dữ liệu phải được viết vào vị trí hiện thời của chúng trên đĩa, nhưng việc viết này có thể được tiến hành mà hệ cơ sở dữ liệu không cần thiết phải chờ nó hoàn tất. Log-disk có thể được sử dụng để khôi phục dữ liệu. Hệ thống tập tin dựa trên log là một phiên bản của cách tiếp cận log-disk: Dữ liệu không được viết lại lên đích gốc của nó trên đĩa; thay vào đó, hệ thống tập tin lưu vết nơi các khối được viết mới nhất trên log-disk, và hoàn lại chúng từ vị trí này. Log-disk được "cô đặc" lại (compacting) theo một định kỳ. Cách tiếp cận này cải tiến hiệu năng viết, song sinh ra sự phân mảnh đối với các tập tin được cập nhật thường xuyên.

V.2.4 RAID (Redundant Arrays of Inexpensive Disks)

Trong một hệ thống có nhiều đĩa, ta có thể cải tiến tốc độ đọc viết dữ liệu nếu cho chúng hoạt động song song. Mặt khác, hệ thống nhiều đĩa còn giúp tăng độ tin cậy lưu trữ bằng cách lưu trữ dữ liệu thông tin trên các đĩa khác nhau, nếu một đĩa có sự cố dữ liệu cũng không bị mất. Một sự đa dạng các kỹ thuật tổ chức đĩa, được gọi là RAID

(Redundant Arrays of Inexpensive Disks), được đề nghị nhằm vào vấn đề tăng cường hiệu năng và độ tin cậy.

V.2.4.1 Cải tiến độ tin cậy thông qua sự dư thừa

Giải pháp cho vấn đề độ tin cậy là đưa vào sự dư thừa: lưu trữ thông tin phụ, bình thường không cần thiết, nhưng nó có thể được sử dụng để tái tạo thông tin bị mất khi gặp sự cố hỏng hóc đĩa, như vậy thời gian trung bình không sự cố tăng lên (xét tổng thể trên hệ thống đĩa).

Đơn giản nhất, là làm bản sao cho mỗi đĩa. Kỹ thuật này được gọi là tạo đĩa ảnh (mirroring) hay sự tạo bóng (shadowing). Một đĩa logic khi đó bao gồm hai đĩa vật lý, và mỗi việc viết được thực hiện trên cả hai đĩa. Nếu một đĩa bị hư, dữ liệu có thể được đọc từ đĩa kia. Thời gian trung bình không sự cố của đĩa mirror phụ thuộc vào thời gian trung bình không sự cố của mỗi đĩa và phụ thuộc vào thời gian trung bình được sửa chữa (mean time to repair): thời gian trung bình để một đĩa bị hư được thay thế và phục hồi dữ liệu trên nó.

V.2.4.2 Cải tiến hiệu năng thông qua song song

Với đĩa phản chiếu (mirror disk), tốc độ đọc có thể tăng lên gấp đôi vì yêu cầu đọc có thể được gửi đến cả hai đĩa. Với nhiều đĩa, ta có thể cải tiến tốc độ truyền bởi phân nhỏ (striping data) dữ liệu qua nhiều đĩa. Dạng đơn giản nhất là tách các bít của một byte qua nhiều đĩa, sự phân nhỏ này được gọi là sự phân nhỏ mức bit (bit-level striping). Ví dụ, ta có một dàn 8 đĩa, ta viết bít thứ i của một byte lên đĩa thứ i. Dàn 8 đĩa này có thể được xử lý như một đĩa với các sector 8 lần lớn hơn kích cỡ thông thường, quan trọng hơn là tốc độ truy xuất tăng lên tám lần. Trong một tổ chức như vậy, mỗi đĩa tham gia vào mỗi truy xuất (đọc/viết), như vậy, số các truy xuất có thể được xử lý trong một giây là tương tự như trên một đĩa, nhưng mỗi truy xuất có thể đọc/viết nhiều dữ liệu hơn tám lần.

Phân nhỏ mức bit có thể được tổng quát cho số đĩa là bội hoặc ước của 8. Ví dụ, ta có một dàn 4 đĩa, ta sẽ phân phối bít thứ i và bít thứ $4+i$ vào đĩa thứ i. Hơn nữa, sự phân nhỏ không nhất thiết phải ở mức bit của một byte. Ví dụ, trong sự phân nhỏ mức khối, các khối của một tập tin được phân nhỏ qua nhiều đĩa, với n đĩa, khối thứ i có thể được phân phối qua đĩa $(i \bmod n) + 1$. Ta cũng có thể phân nhỏ ở mức byte, sector hoặc các sector của một khối. Hai đích song song trong một hệ thống đĩa là:

1/ Nạp nhiều truy xuất nhỏ cân bằng (truy xuất trang) sao cho lượng dữ liệu được nạp trong một đơn vị thời gian của truy xuất như vậy tăng lên.

2/ Song song hoá các truy xuất lớn sao cho thời gian trả lời các truy xuất lớn giảm.

V.2.4.3 Các mức RAID

Đĩa ảnh (mirroring) cung cấp độ tin cậy cao, nhưng đắt giá. Phân nhỏ cung cấp tốc độ truyền dữ liệu cao, nhưng không cải tiến được độ tin cậy. Nhiều sơ đồ cung cấp sự dư thừa với giá thấp bằng cách phối hợp ý tưởng của phân nhỏ với bit chẵn lẻ (parity bit). Các sơ đồ này có sự thoả hiệp giá-hiệu năng khác nhau và được phân lớp thành các mức được gọi là các mức RAID.

- **Mức RAID 0:** Liên quan đến các dàn đĩa với sự phân nhỏ mức khồi, nhưng không có một sự dư thừa nào.

- **Mức RAID 1:** Liên quan đến đĩa phản chiếu (mirror disk).

- **Mức RAID 2:** Cũng được biết dưới cái tên mã sửa lỗi kiểu bộ nhớ (Memory-style error-Correcting-Code: ECC). Hệ thống bộ nhớ thực hiện phát hiện lỗi bằng bit chẵn lẻ. Mỗi byte trong hệ thống bộ nhớ có thẻ có một bit chẵn lẻ kết hợp với nó. Sơ đồ sửa lỗi lưu hai hoặc nhiều hơn các bit phụ, và có thể dựng lại dữ liệu nếu một bit bị lỗi. Ý tưởng của mã sửa lỗi có thể được sử dụng trực tiếp trong dàn đĩa thông qua phân nhỏ byte qua các đĩa. Ví dụ, bit đầu tiên của mỗi byte có thẻ được lưu trên đĩa 1, bit thứ hai trên đĩa 2, và cứ như vậy, bit thứ 8 trên đĩa 8, các bit sửa lỗi được lưu trên các đĩa thêm vào. Nếu một trong các đĩa bị hư, các bit còn lại của byte và các bit sửa lỗi kết hợp được đọc từ các đĩa khác có thể giúp tái tạo bit bị mất trên đĩa hư, như vậy ta có thể dựng lại dữ liệu. Với một dàn 4 đĩa dữ liệu, RAID mức 2 chỉ cần thêm 3 đĩa để lưu các bit sửa lỗi (các đĩa thêm vào này được gọi là các đĩa overhead), so sánh với RAID mức 1, cần 4 đĩa overhead.

- **Mức RAID 3:** Còn được gọi là tổ chức đan xen bit chẵn lẻ (bit-interleaved parity). Bộ điều khiển đĩa có thể phát hiện một sector được đọc đúng hay sai, như vậy có thể sử dụng chỉ một bit chẵn lẻ để sửa lỗi: Nếu một trong các sector bị hư, ta biết chính xác đó là sector nào. Với mỗi bit trong sector này ta có thể hình dung nó là bit 1 hay bit 0 bằng cách tính *chẵn lẻ* của các bit tương ứng từ các sector trên các đĩa khác. Nếu *chẵn lẻ* của các bit còn lại bằng với *chẵn lẻ* được lưu, bit mất sẽ là 0, ngoài ra bit mất là 1. RAID mức 3 tốt như mức 2 nhưng ít tốn kém hơn (chỉ cần một đĩa overhead).

- **Mức RAID 4:** Còn được gọi là tổ chức đan xen khói chẵn lẻ (Block-interleaved parity), lưu trữ các khói đúng như trong các đĩa chính quy, không phân nhỏ chúng qua các đĩa nhưng lấy một khói chẵn lẻ trên một đĩa riêng biệt đối với các khói tương ứng từ N đĩa khác. Nếu một trong các đĩa bị hư, khói parity có thể được dùng với các khói tương ứng từ các đĩa khác để khôi phục khói của đĩa bị hư.

Một đọc khói chỉ truy xuất một đĩa, cho phép các yêu cầu khác được xử lý bởi các đĩa khác. Như vậy, tốc độ truyền dữ liệu đối với mỗi truy xuất chậm, nhưng nhiều truy xuất đọc có thể được xử lý song song, dẫn đến một tốc độ I/O tổng thể cao hơn. Tốc độ truyền đối với các đọc dữ liệu lớn (nhiều khói) cao do tất cả các đĩa có thể được đọc song song; các viết dữ liệu lớn (nhiều khói) cũng có tốc độ truyền cao vì dữ liệu và chẵn lẻ có thể được viết song song. Tuy nhiên, viết một khói đơn phải truy xuất đĩa trên đó khói được lưu trữ, và đĩa chẵn lẻ (do khói chẵn lẻ cũng phải được cập nhật). Như vậy, viết một khói đơn yêu cầu 4 truy xuất: hai để đọc hai khói cũ, và hai để viết lại hai khói.

- **Mức RAID 5 :** Còn gọi là phân bổ đan xen khói chẵn lẻ (Block-interleaved Distributed Parity), cải tiến của mức 4 bởi phân hoạch dữ liệu và chẵn lẻ giữa toàn bộ N+1 đĩa, thay vì lưu trữ dữ liệu trên N đĩa và chẵn lẻ trên một đĩa riêng biệt như trong RAID 4. Trong RAID 5, tất cả các đĩa có thể tham gia làm thỏa mãn các yêu cầu đọc, như vậy sẽ làm tăng tổng số yêu cầu có thể được đặt ra trong một đơn vị thời gian. Đối với mỗi khói, một đĩa lưu trữ parity, các đĩa khác lưu trữ dữ liệu. Ví dụ, với một dàn năm đĩa, chẵn lẻ đối với khói thứ n được lưu trên đĩa $(n \bmod 5)+1$. Các khói thứ n của 4 đĩa khác lưu trữ dữ liệu hiện hành của khói đó.

- **Mức RAID 6:** Còn được gọi là sơ đồ dư thừa P+Q (P+Q redundancy scheme), nó rất giống RAID 5 nhưng lưu trữ thông tin dư thừa phụ để canh chừng nhiều đĩa bị hư. Thay vì sử dụng chẵn lẻ, người ta sử dụng các mã sửa lỗi.

V.2.4.4 Chọn mức RAID đúng

Nếu đĩa bị hư, thời gian tái tạo dữ liệu của nó là đáng kể và thay đổi theo mức RAID được dùng. Sự tái tạo dễ dàng nhất đối với mức RAID 1. Đối với các mức khác, ta phải truy xuất tất cả các đĩa khác trong dàn đĩa để tái tạo dữ liệu trên đĩa bị hư. Hiệu năng tái tạo của một hệ thống RAID có thể là một nhân tố quan trọng nếu việc cung cấp dữ liệu liên tục được yêu cầu (thường xảy ra trong các hệ CSDL hiệu năng

cao hoặc trao đổi). Hơn nữa, hiệu năng tái tạo ảnh hưởng đến thời gian trung bình không sự cố.

Vì RAID mức 2 và 4 được gộp lại bởi RAID mức 3 và 5, Việc lựa chọn mức RAID thu hẹp lại trên các mức RAID còn lại. Mức RAID 0 được dùng trong các ứng dụng hiệu năng cao ở đó việc mất dữ liệu không có gì là trầm trọng cả. RAID mức 1 là thông dụng cho các ứng dụng lưu trữ các tập tin nhật ký (log-tập tin) trong hệ CSDL. Do mức 1 có overhead cao, mức 3 và 5 thường được ưa thích hơn đối với việc lưu trữ khối lượng dữ liệu lớn. Sự khác nhau giữa mức 3 và mức 5 là tốc độ truyền dữ liệu đối với tốc độ I/O tổng thể. Mức 3 được lựa chọn nhiều hơn nếu truyền dữ liệu cao được yêu cầu, mức 5 được ưa thích hơn nếu việc đọc ngẫu nhiên là quan trọng. Mức 6, tuy hiện nay ít được áp dụng, nhưng nó có độ tin cậy cao hơn mức 5.

V.2.4.5 *Mở rộng*

Các quan niệm của RAID được khái quát hóa cho các thiết bị lưu trữ khác, bao hàm các dàn băng, thậm chí đối với quảng bá dữ liệu trên các hệ thống không dây. Khi áp dụng RAID cho dàn băng, cấu trúc RAID cho khả năng khôi phục dữ liệu cả khi một trong các băng bị hư hại. Khi áp dụng đối với quảng bá dữ liệu, một khối dữ liệu được phân thành các đơn vị nhỏ và được quảng bá cùng với một đơn vị parity; nếu một trong các đơn vị này không nhận được, nó có thể được dựng lại từ các đơn vị còn lại.

V.3 TRUY XUẤT LUU TRU'

Một cơ sở dữ liệu được ánh xạ vào một số các tập tin khác nhau được duy trì bởi hệ điều hành nền. Các tập tin này lưu trú thường trực trên các đĩa với sao lưu dự phòng trên băng. Mỗi tập tin được phân hoạch thành các đơn vị lưu trữ độ dài cố định được gọi là khối - đơn vị cho cả cấp phát lưu trữ và truyền dữ liệu.

Một khối có thể chứa một vài đơn vị dữ liệu (data item). Ta giả thiết không một đơn vị dữ liệu nào trải ra trên hai khối. Mục tiêu nổi trội của hệ CSDL là tối thiểu hóa số khối truyền giữa đĩa và bộ nhớ. Một cách để giảm số truy xuất đĩa là giữ nhiều khối như có thể trong bộ nhớ chính. Mục đích là để khi một khối được truy xuất, nó đã nằm sẵn trong bộ nhớ chính và như vậy không cần một truy xuất đĩa nào cả.

Do không thể lưu tất cả các khối trong bộ nhớ chính, ta cần quản trị cấp phát không gian sẵn có trong bộ nhớ chính để lưu trữ các khối. Bộ đệm là một phần của bộ nhớ chính sẵn có để lưu trữ bản sao khối đĩa. Luôn có một bản sao trên đĩa cho mỗi khối, song các bản sao trên đĩa của các khối là các phiên bản cũ hơn so với phiên bản

trong buffer. Hệ thống con đảm trách cấp phát không gian buffer được gọi là bộ quản trị bộ đệm.

Bộ quản trị bộ đệm

Các chương trình trong một hệ CSDL đưa ra các yêu cầu cho bộ quản trị bộ đệm khi chúng cần một khối đĩa. Nếu khối này đã sẵn sàng trong bộ đệm, địa chỉ khối trong bộ nhớ chính được chuyển cho người yêu cầu. Nếu khối chưa có trong bộ đệm, bộ quản trị bộ đệm đầu tiên cấp phát không gian trong bộ đệm cho khối, rút ra một số khối khác, nếu cần thiết, để lấy không gian cho khối mới. Khối được rút ra chỉ được viết lại trên đĩa khi nó có bị sửa đổi kể từ lần được viết lên đĩa gần nhất. Sau đó bộ quản trị bộ đệm đọc khối từ đĩa vào bộ đệm, và chuyển địa chỉ của khối trong bộ nhớ chính cho người yêu cầu. Bộ quản trị bộ đệm không khác gì nhiều so với bộ quản trị bộ nhớ ảo, một điểm khác biệt là kích cỡ của một CSDL có thể rất lớn không đủ chứa toàn bộ trong bộ nhớ chính do vậy bộ quản trị bộ đệm phải sử dụng các kỹ thuật tinh vi hơn các sơ đồ quản trị bộ nhớ ảo kiểu mẫu.

- **Chiến lược thay thế:** Khi không có chỗ trong buffer, một khối phải được xoá khỏi buffer trước khi một khối mới được đọc vào. Thông thường, hệ điều hành sử dụng sơ đồ LRU (Least Recently Used) để viết lên đĩa khối ít được dùng gần đây nhất, xoá bỏ nó khỏi buffer. Cách tiếp cận này có thể được cải tiến đối với ứng dụng CSDL.
- **Khối chốt (pinned blocks):** Để hệ CSDL có thể khôi phục sau sự cố, cần thiết phải hạn chế thời gian khi viết lại lên đĩa một khối. Một khối không cho phép viết lại lên đĩa được gọi là khối chốt.
- **Xuất ra bắt buộc các khối (Forced output of blocks):** Có những tình huống trong đó cần phải viết lại một khối lên đĩa, cho dù không gian bộ đệm mà nó chiếm là không cần đến. Việc viết này được gọi là sự xuất ra bắt buộc của một khối. Lý do ngăn gon của yêu cầu xuất ra bắt buộc khối là nội dung của bộ nhớ chính bị mất khi có sự cố, ngược lại dữ liệu trên đĩa còn tồn tại sau sự cố.

Các đối sách thay thế bộ đệm (Buffer-Replacement Policies)

Mục đích của chiến lược thay thế khối trong bộ đệm là tối thiểu hoá các truy xuất đĩa. Các hệ điều hành thường sử dụng chiến lược LRU (*Least Recently Used*) để thay thế khối. Tuy nhiên, một hệ CSDL có thể dự đoán mẫu tham khảo tương lai. Yêu cầu của một người sử dụng đối với hệ CSDL bao gồm một số bước. Hệ CSDL có thể

xác định trước những khôi nào sẽ là cần thiết bằng cách xem xét mỗi một trong các bước được yêu cầu để thực hiện hoạt động được yêu cầu bởi người sử dụng. Như vậy, khác với hệ điều hành, hệ CSDL có thể có thông tin liên quan đến tương lai, ít nhất là tương lai gần. Trong nhiều trường hợp, chiến lược thay thế khôi tối ưu cho hệ CSDL lại là MRU (Most Recently Used): Khôi bị thay thế sẽ là khôi mới được dùng gần đây nhất!

Bộ quản trị bộ đệm có thể sử dụng thông tin thống kê liên quan đến xác suất mà một yêu cầu sẽ tham khảo một quan hệ riêng biệt nào đó. Tự điển dữ liệu là một trong những phần được truy xuất thường xuyên nhất của CSDL. Như vậy, bộ quản trị bộ đệm sẽ không nên xoá các khôi tự điển dữ liệu khỏi bộ nhớ chính trừ phi các nhân tố khác bức ché làm điều đó. Một chỉ mục (Index) đối với một tập tin được truy xuất thường xuyên hơn chính bản thân tập tin, vậy thì bộ quản trị bộ đệm cũng không nên xoá khôi chỉ mục khỏi bộ nhớ chính nếu có sự lựa chọn.

Chiến lược thay thế khôi CSDL lý tưởng cần hiểu biết về các hoạt động CSDL đang được thực hiện. Không một chiến lược đơn lẻ nào được biết nắm bắt được toàn bộ các viễn cảnh có thể. Tuy vậy, một điều đáng ngạc nhiên là phần lớn các hệ CSDL sử dụng LRU bát chấp các khuyết điểm của chiến lược đó.

Chiến lược được sử dụng bởi bộ quản trị buffer để thay thế khôi bị ảnh hưởng bởi các nhân tố khác hơn là nhân tố thời gian tại đó khôi được tham khảo trở lại. Nếu hệ thống đang xử lý các yêu cầu của một vài người sử dụng cạnh tranh, hệ thống (con) điều khiển cạnh tranh (concurrency-control subsystem) có thể phải làm trễ một số yêu cầu để đảm bảo tính nhất quán của CSDL. Nếu bộ quản trị bộ đệm được cho các thông tin từ hệ thống điều khiển cạnh tranh mà nó nêu rõ những yêu cầu nào đang bị làm trễ, nó có thể sử dụng các thông tin này để thay đổi chiến lược thay thế khôi của nó. Đặc biệt, các khôi cần thiết bởi các yêu cầu tích cực (active requests) có thể được giữ lại trong bộ đệm, toàn bộ các bất lợi do dồn lên các khôi cần thiết bởi các yêu cầu bị làm trễ.

Hệ thống (con) khôi phục (crash-recovery subsystem) áp đặt các ràng buộc nghiêm nhặt lên việc thay thế khôi. Nếu một khôi bị sửa đổi, bộ quản trị bộ đệm không được phép viết lại phiên bản mới của khôi trong bộ đệm lên đĩa vì điều này phá huỷ phiên bản cũ. Thay vào đó, bộ quản trị khôi phải tìm kiếm quyền từ hệ thống khôi phục trước khi viết khôi. Hệ thống khôi phục có thể đòi hỏi một số khôi nhất định khác là

xuất bắt buộc (forced output) trước khi cấp quyền cho bộ quản trị buffer để xuất ra khỏi được yêu cầu.

V.4 TÔ CHỨC TẬP TIN

Một tập tin được tổ chức logic như một dãy các mẫu tin. Các mẫu tin này được ánh xạ lên các khói đĩa. Tập tin được cung cấp như một xây dựng cơ sở trong hệ điều hành, như vậy ta sẽ giả thiết sự tồn tại của hệ thống tập tin nền.

Ta cần phải xét những phương pháp biểu diễn các mô hình dữ liệu logic trong thuật ngữ tập tin.

Các khói có kích cỡ cố định được xác định bởi tính chất vật lý của đĩa và bởi hệ điều hành, song kích cỡ của mẫu tin lại thay đổi. Trong CSDL quan hệ, các bộ của các quan hệ khác nhau nói chung có kích cỡ khác nhau.

Một tiếp cận để ánh xạ một CSDL đến các tập tin là sử dụng một số tập tin, và lưu trữ các mẫu tin thuộc chỉ một độ dài cố định vào một tập tin đã cho nào đó. Một cách khác là cấu trúc các tập tin sao cho ta có thể điều tiết nhiều độ dài cho các mẫu tin. Các tập tin của các mẫu tin độ dài cố định dễ dàng thực thi hơn tập tin của các mẫu tin độ dài thay đổi.

V.4.1 Mẫu tin độ dài cố định (Fixed-Length Records)

Xét một tập tin các mẫu tin account đối với CSDL ngân hàng, mỗi mẫu tin của tập tin này được xác định như sau:

```
type deposit = record
    account-number: char(10);
    branch-name: char(22);
    balance: real;
end
```

Giả sử mỗi một ký tự chiếm 1 byte và mỗi số thực chiếm 8 bytes, như vậy mẫu tin account có độ dài 40 bytes. Một cách tiếp cận đơn giản là sử dụng 40 bytes đầu tiên cho mẫu tin thứ nhất, 40 bytes kế tiếp cho mẫu tin thứ hai, ... Cách tiếp cận đơn giản này nảy sinh những vấn đề sau;

A-102	Perryridge	400
A-305	Round Hill	350
A-215	Mianus	700
A-101	Downtown	500
A-222	Redwood	700
A-201	Perryridge	900
A-217	Brighton	750
A-110	Downtown	600
A-218	Perryridge	700

Hình 5.3: Mẫu tin có chiều dài cố định

1/- Khó khăn khi xoá một mẫu tin từ cấu trúc này. Không gian bị chiếm bởi mẫu tin bị xoá phải được lấp đầy với mẫu tin khác của tập tin hoặc ta phải đánh dấu mẫu tin bị xoá.

2/- Trừ khi kích cỡ khối là bội của 40, nếu không một số mẫu tin sẽ bắt chéo qua biên khối, có nghĩa là một phần mẫu tin được lưu trong một khối, một phần khác được lưu trong một khối khác. Như vậy đòi hỏi phải truy xuất hai khối để đọc/viết một mẫu tin "bắc cầu" đó.

Khi một mẫu tin bị xoá, ta có thể di chuyển mẫu tin kề sau nó vào không gian bị chiếm một cách hình thức bởi mẫu tin bị xoá, rồi mẫu tin kề tiếp vào không gian bị chiếm của mẫu tin vừa được di chuyển, cứ như vậy cho đến khi mỗi mẫu tin đi sau mẫu tin bị xoá được dịch chuyển hướng về đầu. Cách tiếp cận này đòi hỏi phải di chuyển một số lớn các mẫu tin. Một cách tiếp cận khác đơn giản hơn là di chuyển mẫu tin cuối cùng vào không gian bị chiếm bởi mẫu tin bị xoá. Song cách tiếp cận này đòi hỏi phải truy xuất khôi bỏ xung. Vì hoạt động xen xẩy ra thường xuyên hơn hoạt động xoá, ta có thể chấp nhận việc để "ngó" không gian bị chiếm bởi mẫu tin bị xoá, và chờ một hoạt động xen đến sau để tái sử dụng không gian đó. Một dấu trên mẫu tin bị xoá là không đủ vì sẽ gây khó khăn cho việc tìm kiếm không gian "tự do" đó khi xen. Như vậy ta cần đưa vào cấu trúc bỏ xung. Ở đầu của tập tin, ta cấp phát một số byte nhất định làm header của tập tin. Header này sẽ chứa đựng thông tin về tập tin. Header chứa địa chỉ của mẫu tin bị xoá thứ nhất, trong nội dung của mẫu tin này có chứa địa chỉ của mẫu tin bị xoá thứ hai và cứ như vậy. Như vậy, các mẫu tin bị xoá sẽ tạo ra một danh sách liên kết được gọi là danh sách tự do (free list). Khi xen mẫu tin mới, ta sử dụng con trỏ đầu

danh sách được chứa trong header để xác định danh sách, nếu danh sách không rỗng ta xen mẫu tin mới vào vùng được trỏ bởi con trỏ đầu danh sách nếu không ta xen mẫu tin mới vào cuối tập tin.

Xen và xoá đối với tập tin mẫu tin độ dài cố định thực hiện đơn giản vì không gian được giải phóng bởi mẫu tin bị xoá đúng bằng không gian cần thiết để xen một mẫu tin. Đối với tập tin của các mẫu tin độ dài thay đổi vẫn đề trở nên phức tạp hơn nhiều.

V.4.2 Mẫu tin độ dài thay đổi (Variable-Length Records)

Mẫu tin độ dài thay đổi trong CSDL do bởi:

- Việc lưu trữ nhiều kiểu mẫu tin trong một tập tin.
- Kiểu mẫu tin cho phép độ dài trường thay đổi.
- Kiểu mẫu tin cho phép lặp lại các trường.

Có nhiều kỹ thuật để thực hiện mẫu tin độ dài thay đổi. Để minh họa ta sẽ xét các biểu diễn khác nhau trên các mẫu tin độ dài thay đổi có định dạng sau:

```
type account-list = record
    branch-name: char(22);
    account-info: array [1..n] of
        record
            account-number: char(10);
            balance: real;
        end
    end
```

Biểu diễn chuỗi byte (Byte-String Representation)

Một cách đơn giản để thực hiện các mẫu tin độ dài thay đổi là gắn một ký hiệu đặc biệt End-of-record (\perp) vào cuối mỗi record. Khi đó, ta có thể lưu mỗi mẫu tin như một chuỗi byte liên tiếp. Thay vì sử dụng một ký hiệu đặc biệt ở cuối của mỗi mẫu tin, một phiên bản của biểu diễn chuỗi byte lưu trữ độ dài mẫu tin ở bắt đầu của mỗi mẫu tin.

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥				
2	Mianus	A-101	800	⊥				
3	Downtown	A-211	500	A-222	600	⊥		
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥				

Hình 5.4: Biểu diễn chuỗi byte của các mẫu tin độ dài thay đổi.

Biểu diễn chuỗi byte có các bất lợi sau:

- Khó sử dụng không gian bị chiếm hình thức bởi một mẫu tin bị xoá, điều này dẫn đến một số lớn các mảnh nhỏ của lưu trữ đĩa bị lãng phí.
- Không có không gian cho sự phát triển các mẫu tin. Nếu một mẫu tin độ dài thay đổi dài ra, nó phải được di chuyển và sự di chuyển này là đắt giá nếu mẫu tin bị chốt.

Biểu diễn chuỗi byte không thường được sử dụng để thực hiện mẫu tin độ dài thay đổi, song một dạng sửa đổi của nó được gọi là cấu trúc khe-trang (slotted-page structure) thường được dùng để tổ chức mẫu tin trong một khối đơn.

Trong cấu trúc slotted-page, có một header ở bắt đầu của mỗi khối, chứa các thông tin sau:

- Số các đầu vào mẫu tin (record entries) trong header.
- Điểm cuối không gian tự do (End of Free Space) trong khối.
- Một mảng các đầu vào chứa vị trí và kích cỡ của mỗi mẫu tin.

Các mẫu tin hiện hành được cấp phát kè nhau trong khối, bắt đầu từ cuối khối. Không gian tự do trong khối là một vùng kè nhau, nằm giữa đầu vào cuối cùng trong mảng header và mẫu tin đầu tiên. Khi một mẫu tin được xen vào, không gian cấp phát cho nó ở cuối của không gian tự do, và đầu vào tương ứng với nó được thêm vào header.

Nếu một mẫu tin bị xoá, không gian bị chiếm bởi nó được giải phóng, đầu vào ứng với nó được đặt là bị xoá (kích cỡ của nó được đặt chặng hạn là -1). Sau đó, các mẫu tin trong khối trước mẫu tin bị xoá được di chuyển sao cho không gian tự do của khối lại là phần nằm giữa đầu vào cuối cùng của mảng header và mẫu tin đầu tiên. Con trỏ điểm cuối không gian tự do và các con trỏ ứng với mẫu tin bị di chuyển được cập

nhật. Sự lớn lên hay nhỏ đi của mẫu tin cũng sử dụng kỹ thuật tương tự (trong trường hợp khôi còn không gian cho sự lớn lên của mẫu tin). Cái giá phải trả cho sự di chuyển không quá cao vì các khôi có kích cỡ không lớn (thường 4Kbytes).

V.4.2.2 Biểu diễn độ dài cố định

Một cách khác để thực hiện mẫu tin độ dài thay đổi một cách hiệu quả trong một hệ thống tập tin là sử dụng một hoặc một vài mẫu tin độ dài cố định để biểu diễn một mẫu tin độ dài thay đổi.

Hai kỹ thuật thực hiện tập tin của các mẫu tin độ dài thay đổi sử dụng mẫu tin độ dài cố định là:

1/- Không gian dự trù (reserved space): Giả thiết rằng các mẫu tin có độ dài không vượt quá một ngưỡng (độ dài tối đa). Ta có thể sử dụng mẫu tin độ dài cố định (có độ dài tối đa), Phần không gian chưa dùng đến được lấp đầy bởi một ký tự đặc biệt: null hoặc End-of-record.

0	Perrynidge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥	⊥	⊥	⊥	⊥
2	Mianus	A-101	800	⊥	⊥	⊥	⊥	⊥
3	Downtown	A-211	500	A-222	600	⊥	⊥	⊥
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥	⊥	⊥	⊥	⊥

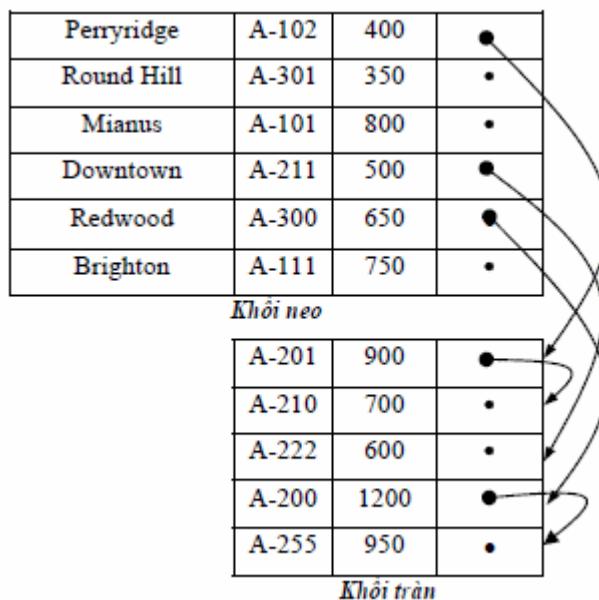
Hình 5.5: Phương pháp không gian lưu trữ

2/- Contrô (Pointers): Mẫu tin độ dài thay đổi được biểu diễn bởi một danh sách các mẫu tin độ dài cố định, được "móc xích" với nhau bởi các con trỏ.

Sự bất lợi của cấu trúc con trỏ là lãng phí không gian trong tất cả các mẫu tin ngoại trừ mẫu tin đầu tiên trong danh sách (mẫu tin đầu tiên cần trường branch_name, các mẫu tin sau trong danh sách không cần thiết có trường này!). Để giải quyết vấn đề này người ta đề nghị phân các khôi trong tập tin thành hai loại:

- ♦ **Khối neo (Anchor block):** chỉ chứa các mẫu tin đầu tiên trong danh sách.
- ♦ **Khối tràn (Overflow block):** chứa các mẫu tin còn lại của danh sách.

Như vậy, tất cả các mẫu tin trong một khối có cùng độ dài, cho dù tập tin có thể chứa các mẫu tin không cùng độ dài.



Hình 5.6a: Phương pháp con trò

V.5 TỔ CHỨC CÁC MẪU TIN TRONG TẬP TIN.

Ta đã xét làm thế nào để biểu diễn các mẫu tin trong một cấu trúc tập tin. Một thể hiện của một quan hệ là một tập hợp các mẫu tin. Đã cho một tập hợp các mẫu tin, vấn đề đặt ra là làm thế nào để tổ chức chúng trong một tập tin.

Có một số cách tổ chức sau:

- **Tổ chức tập tin đồng (Heap Tập tin Organization):** Trong tổ chức này, một mẫu tin bất kỳ có thể được lưu trữ ở bất kỳ nơi nào trong tập tin, ở đó có không gian cho nó. Không có thứ tự nào giữa các mẫu tin. Một tập tin cho một quan hệ.
- **Tổ chức tập tin tuần tự (Sequential Tập tin Organization):** Trong tổ chức này, các mẫu tin được lưu trữ thứ tự tuần tự, dựa trên giá trị của khoá tìm kiếm của mỗi mẫu tin.
- **Tổ chức tập tin băm (Hashed Tập tin Organization):** Trong tổ chức này, có một hàm băm được tính toán trên thuộc tính nào đó của mẫu tin. Kết quả của hàm băm xác định mẫu tin được bố trí trong khối nào trong tập tin. Tổ chức này liên hệ chặt chẽ với cấu trúc chỉ mục.

- **Tổ chức tập tin cụm (Clustering Tập tin Organization):** Trong tổ chức này, các mẫu tin của một vài quan hệ khác nhau có thể được lưu trữ trong cùng một tập tin. Các mẫu tin có liên hệ của các quan hệ khác nhau được lưu trữ trên cùng một khối sao cho một hoạt động I/O đem lại các mẫu tin có liên hệ từ tất cả các quan hệ.

V.5.1 Tổ chức tập tin tuần tự

Tổ chức tập tin tuần tự được thiết kế để xử lý hiệu quả các mẫu tin trong thứ tự được sắp dựa trên một khoá tìm kiếm (search key) nào đó. Để cho phép tìm lại nhanh chóng các mẫu tin theo thứ tự khoá tìm kiếm, ta "xích" các mẫu tin lại bởi các con trỏ. Con trỏ trong mỗi mẫu tin trỏ tới mẫu tin kế theo thứ tự khoá tìm kiếm. Hơn nữa, để tối ưu hoá số khối truy xuất trong xử lý tập tin tuần tự, ta lưu trữ vật lý các mẫu tin theo thứ tự khoá tìm kiếm hoặc gần với khoá tìm kiếm như có thể.

Tổ chức tập tin tuần tự cho phép đọc các mẫu tin theo thứ tự được sắp mà nó có thể hữu dụng cho mục đích trình bày cũng như cho các thuật toán xử lý vấn tin (query-processing algorithms).

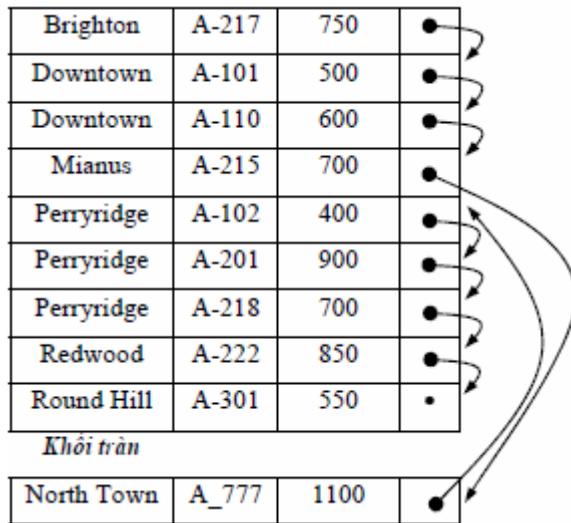
Brighton	A-217	750	•
Downtown	A-101	500	•
Downtown	A-110	600	•
Mianus	A-215	700	•
Perryridge	A-102	400	•
Perryridge	A-201	900	•
Perryridge	A-218	700	•
Redwood	A-222	850	•
Round Hill	A-301	550	•

Hình 5.6b: Tổ chức tập tin tuần tự

Khó khăn gặp phải của tổ chức này là việc duy trì thứ tự tuần tự vật lý của các mẫu tin khi xảy ra các hoạt động xen, xoá, do cái giá phải trả cho việc di chuyển các mẫu tin khi xen, xoá. Ta có thể quản trị vấn đề xoá bởi dùng dây chuyền các con trỏ như đã trình bày trước đây. Đối với xen, ta có thể áp dụng các quy tắc sau:

- 1/- Định vị mẫu tin trong tập tin mà nó đi trước mẫu tin được xen theo thứ tự khoá tìm kiếm.

2/- Nếu có mẫu tin tự do (không gian của mẫu tin bị xoá) trong cùng khói, xen mẫu tin vào khói này. Nếu không, xen mẫu tin mới vào một khói tràn. Trong cả hai trường hợp, điều chỉnh các con trỏ sao cho nó móc xích các mẫu tin theo thứ tự của khoá tìm kiếm.



Hình 5.7: Chèn mẫu tin mới vào tập tin tuần tự

V.5.2 Tổ chức tập tin cụm

Nhiều hệ CSDL quan hệ, mỗi quan hệ được lưu trữ trong một tập tin sao cho có thể lợi dụng được toàn bộ những cái mà hệ thống tập tin của điều hành cung cấp. Thông thường, các bộ của một quan hệ được biểu diễn như các mẫu tin độ dài cố định. Như vậy các quan hệ có thể ánh xạ vào một cấu trúc tập tin. Sự thực hiện đơn giản đó của một hệ CSDL quan hệ rất phù hợp với các hệ CSDL được thiết kế cho các máy tính cá nhân. Trong các hệ thống đó, kích cỡ của CSDL nhỏ. Hơn nữa, trong một số máy tính cá nhân, chủ yếu kích cỡ tổng thể mã đối tượng đối với hệ CSDL là nhỏ. Một cấu trúc tập tin đơn giản làm suy giảm lượng mã cần thiết để thực thi hệ thống.

Cách tiếp cận đơn giản này, để thực hiện CSDL quan hệ, không còn phù hợp khi kích cỡ của CSDL tăng lên. Ta sẽ thấy những điểm lợi về mặt hiệu năng từ việc gán một cách thận trọng các mẫu tin với các khói, và từ việc tổ chức kỹ lưỡng chính bản thân các khói. Như vậy, có vẻ như là một cấu trúc tập tin phức tạp hơn lại có lợi hơn, ngay cả trong trường hợp ta giữ nguyên chiến lược lưu trữ mỗi quan hệ trong một tập tin riêng biệt.

Tuy nhiên, nhiều hệ CSDL quy mô lớn không nhờ cậy trực tiếp vào hệ điều hành nền để quản trị tập tin. Thay vào đó, một tập tin hệ điều hành được cấp phát cho hệ

CSDL. Tất cả các quan hệ được lưu trữ trong một tập tin này, và sự quản trị tập tin này thuộc về hệ CSDL. Để thấy những điểm lợi của việc lưu trữ nhiều quan hệ trong cùng một tập tin, ta xét vần tin SQL sau:

```
SELECT account-number, customer-name, customer-street, customer-city  
FROM depositor, customer  
WHERE depositor.customer-name=customer.customer-name
```

Câu vần tin này tính một phép nối của các quan hệ depositor và customer. Như vậy, đối với mỗi bộ của depositor, hệ thống phải tìm bộ của customer có cùng giá trị customer_name. Một cách lý tưởng là việc tìm kiếm các mẫu tin này nhờ sự trợ giúp của chỉ mục. Bỏ qua việc tìm kiếm các mẫu tin như thế nào, ta chú ý vào việc truyền từ đĩa vào bộ nhớ. Trong trường hợp xấu nhất, mỗi mẫu tin ở trong một khối khác nhau, điều này buộc ta phải đọc một khối cho một mẫu tin được yêu cầu bởi câu vần tin. Ta sẽ trình bày một cấu trúc tập tin được thiết kế để thực hiện hiệu quả các câu vần tin liên quan đến depositor customer. Các bộ depositor đối với mỗi customer_name được lưu trữ gần bộ customer có cùng customer_name. Cấu trúc này trộn các bộ của hai quan hệ với nhau, nhưng cho phép xử lý hiệu quả phép nối. Khi một bộ của quan hệ customer được đọc, toàn bộ khối chứa bộ này được đọc từ đĩa vào trong bộ nhớ chính. Do các bộ tương ứng của depositor được lưu trữ trên đĩa gần bộ customer, khối chứa bộ customer chứa các bộ của quan hệ depositor cần cho xử lý câu vần tin. Nếu một customer có nhiều account đến nối các mẫu tin depositor không lấp đầy trong một khối, các mẫu tin còn lại xuất hiện trong khối kế cận. Cấu trúc tập tin này, được gọi là gom cụm (clustering), cho phép ta đọc nhiều mẫu tin được yêu cầu chỉ sử dụng một đọc khối, như vậy ta có thể xử lý câu vần tin đặc biệt này hiệu quả hơn.

Tuy nhiên, cấu trúc gom cụm trên lại tỏ ra không có lợi bằng tổ chức lưu mỗi quan hệ trong một tập tin riêng, đối với một số câu vần tin, chẳng hạn:

```
SELECT *  
FROM customer
```

Việc xác định khi nào thì gom cụm thường phụ thuộc vào kiểu câu vần tin mà người thiết kế CSDL nghĩ rằng nó xảy ra thường xuyên nhất. Sử dụng thận trọng gom cụm có thể cải thiện hiệu năng đáng kể trong việc xử lý câu vần tin.

V.5.3 Lưu trữ tự điển dữ liệu

Một hệ CSDL cần thiết duy trì dữ liệu về các quan hệ, như sơ đồ của các quan hệ. Thông tin này được gọi là tự điển dữ liệu (data dictionary) hay mục lục hệ thống (system catalog). Trong các kiểu thông tin mà hệ thống phải lưu trữ là:

- Các tên của các quan hệ.
- Các tên của các thuộc tính của mỗi quan hệ.
- Các miền (giá trị) và các độ dài của các thuộc tính.
- Các tên của các View được định nghĩa trên CSDL và định nghĩa của các view này.
- Các ràng buộc toàn vẹn.

Nhiều hệ thống còn lưu trữ các thông tin liên quan đến người sử dụng hệ thống:

- Tên của người sử dụng được phép.
- Giải trình thông tin về người sử dụng.

Các dữ liệu thống kê và mô tả về các quan hệ có thể cũng được lưu trữ:

- Số bộ trong mỗi quan hệ.
- Phương pháp lưu trữ được sử dụng cho mỗi quan hệ (cụm hay không).

Các thông tin về mỗi chỉ mục trên mỗi quan hệ cũng cần được lưu trữ:

- Tên của chỉ mục.
- Tên của quan hệ được chỉ mục.
- Các thuộc tính trên nó chỉ mục được định nghĩa.
- Kiểu của chỉ mục được tạo.

Toàn bộ các thông tin này trong thực tế bao hàm một CSDL nhỏ. Một số hệ CSDL sử dụng những cấu trúc dữ liệu và mã mục đích đặc biệt để lưu trữ các thông tin này. Nói chung, lưu trữ dữ liệu về CSDL trong chính CSDL vẫn được ưa chuộng hơn. Bằng cách sử dụng CSDL để lưu trữ dữ liệu hệ thống, ta đơn giản hóa cấu trúc tổng thể của hệ thống và cho phép sử dụng đầy đủ sức mạnh của CSDL trong việc truy xuất nhanh đến dữ liệu hệ thống.

Sự chọn lựa chính xác biểu diễn dữ liệu hệ thống sử dụng các quan hệ như thế nào là do người thiết kế hệ thống quyết định. Như một ví dụ, ta đề nghị sự biểu diễn sau:

```
System_catalog_schema = (relation_name, number_of_attributes)  
Attribute_schema = (attribute_name, relation_name, domain_type, position, length)  
User_schema = (user_name, encrypted_password, group)  
Index_schema = (index_name, relation_name, index_type, index_attributes)  
View_schema = (view_name, definition)
```

Chỉ mục

Ta xét hoạt động tìm sách trong một thư viện. Ví dụ ta muốn tìm một cuốn sách của một tác giả nào đó. Đầu tiên ta tra trong mục lục tác giả, một tấm thẻ trong mục lục này sẽ chỉ cho ta biết có thẻ tìm thấy cuốn sách đó ở đâu. Các thẻ trong một mục lục được thư viện sắp xếp thứ tự theo vần chữ cái, như vậy giúp ta có thể tìm đến thẻ cần tìm nhanh chóng không cần phải duyệt qua tất cả các thẻ. Chỉ mục của một tập tin trong các công việc hệ thống rất giống với một mục lục trong một thư viện. Tuy nhiên, chỉ mục được làm như mục lục được mô tả như trên, trong thực tế, sẽ quá lớn để được quản lý một cách hiệu quả. Thay vào đó, người ta sử dụng các kỹ thuật chỉ mục tinh tế hơn. Có hai kiểu chỉ mục:

- **Chỉ mục được sắp (Ordered indices):** được dựa trên một thứ tự sắp xếp theo các giá trị.
- **Chỉ mục băm (Hash indices):** được dựa trên các giá trị được phân phối đều qua các bucket. Bucket mà một giá trị được gán với nó được xác định bởi một hàm, được gọi là hàm băm (hash function).

Đối với cả hai kiểu này, ta sẽ nêu ra một vài kỹ thuật, đáng lưu ý là không kỹ thuật nào là tốt nhất. Mỗi kỹ thuật phù hợp với các ứng dụng CSDL riêng biệt. Mỗi kỹ thuật phải được đánh giá trên cơ sở của các nhân tố sau:

- **Kiểu truy xuất:** Các kiểu truy xuất được hỗ trợ hiệu quả. Các kiểu này bao hàm cả tìm kiếm mẫu tin với một giá trị thuộc tính cụ thể hoặc tìm các mẫu tin với giá trị thuộc tính nằm trong một khoảng xác định.
- **Thời gian truy xuất:** Thời gian để tìm kiếm một đơn vị dữ liệu hay một tập các hạng mục.

- **Thời gian xen:** Thời gian để xen một đơn vị dữ liệu mới. giá trị này bao hàm thời gian để tìm vị trí xen thích hợp và thời gian cập nhật cấu trúc chỉ mục.
- **Thời gian xoá:** Thời gian để xoá một đơn vị dữ liệu. giá trị này bao hàm thời gian tìm kiếm hạng mục cần xoá, thời gian cập nhật cấu trúc chỉ mục.
- **Tổng phí tổn không gian:** Không gian phụ bị chiếm bởi một cấu trúc chỉ mục.

Một tập tin thường đi kèm với một vài chỉ mục. Thuộc tính hoặc tập hợp các thuộc tính được dùng để tìm kiếm mẫu tin trong một tập tin được gọi là khoá tìm kiếm. Chú ý rằng định nghĩa này khác với định nghĩa khoá sơ cấp (primary key), khoá dự tuyển (candidate key), và siêu khoá (superkey). Như vậy, nếu có một vài chỉ mục trên một tập tin, có một vài khoá tìm kiếm tương ứng.

V.5.4 Chỉ mục được sắp

Một chỉ mục lưu trữ các giá trị khoá tìm kiếm trong thứ tự được sắp và kết hợp với mỗi khoá tìm kiếm, các mẫu tin chứa khoá tìm kiếm này. Các mẫu tin trong tập tin được chỉ mục có thể chính nó cũng được sắp. Một tập tin có thể có một vài chỉ mục trên những khoá tìm kiếm khác nhau.

Nếu tập tin chứa các mẫu tin được sắp tuần tự, chỉ mục trên khoá tìm kiếm xác định thứ tự này của tập tin được gọi chỉ mục sơ cấp (primary index). Các chỉ mục sơ cấp cũng được gọi là chỉ mục cụm (clustering index). Khoá tìm kiếm của chỉ mục sơ cấp thường là khoá sơ cấp (khoá chính). Các chỉ mục, khoá tìm kiếm của nó xác định một thứ tự khác với thứ tự của tập tin, được gọi là các chỉ mục thứ cấp (secondary indices) hay các chỉ mục không cụm (nonclustering indices).

Brighton	A-217	750	•
Downtown	A-101	500	•
Downtown	A-110	600	•
Mianus	A-215	700	•
Perryridge	A-102	400	•
Perryridge	A-201	900	•
Perryridge	A-218	700	•
Redwood	A-222	850	•
Round Hill	A-301	550	•

Hình 5.8: Tập tin tuần tự các mẫu tin account.

V.5.4.1 Chỉ mục sơ cấp

Trong phần này, ta giả thiết rằng tất cả các tập tin được sắp thứ tự tuần tự trên một khoá tìm kiếm nào đó. Các tập tin như vậy, với một chỉ mục sơ cấp trên khoá tìm kiếm này, được gọi là tập tin tuần tự chỉ mục (index-sequential tập tins). Chúng biểu diễn một trong các sơ đồ xưa nhất được dùng trong hệ CSDL. Chúng được thiết kế cho các ứng dụng đòi hỏi cả xử lý tuần tự toàn bộ tập tin lẫn truy xuất ngẫu nhiên đến một mẫu tin.

Chỉ mục đặc và chỉ mục thưa (Dense and Sparse Indices)

Chỉ mục đặc

Có hai loại chỉ mục được sắp:

♦ **Chỉ mục dày:** Mỗi mẫu tin chỉ mục (đầu vào chỉ mục/ index entry) xuất hiện đối với mỗi giá trị khoá tìm kiếm trong tập tin. mẫu tin chỉ mục chứa giá trị khoá tìm kiếm và một con trỏ tới mẫu tin dữ liệu đầu tiên với giá trị khoá tìm kiếm đó.

♦ **Chỉ mục thưa:** Một mẫu tin chỉ mục được tạo ra chỉ với một số giá trị. Cũng như với chỉ mục đặc, mỗi mẫu tin chỉ mục chứa một giá trị khoá tìm kiếm và một con trỏ tới mẫu tin dữ liệu đầu tiên với giá trị khoá tìm kiếm này. Để định vị một mẫu tin, ta tìm đầu vào chỉ mục với giá trị khoá tìm kiếm lớn nhất trong các giá trị khoá tìm kiếm nhỏ hơn hoặc bằng giá trị khoá tìm kiếm đang tìm. Ta bắt đầu từ mẫu tin được trỏ tới bởi đầu vào chỉ mục, và lần theo các con trỏ trong tập tin (dữ liệu) đến tận khi tìm thấy mẫu tin mong muốn.

Ví dụ:

Giả sử ta tìm các kiêm mẫu tin đối với chi nhánh Perryridge, sử dụng chỉ mục đặc. Đầu tiên, tìm Perryridge trong chỉ mục (tìm nhị phân!), đi theo con trỏ tương ứng đến mẫu tin dữ liệu (với Branch_name = Perryridge) đầu tiên, xử lý mẫu tin này, sau đó đi theo con trỏ trong mẫu tin này để định vị mẫu tin kế trong thứ tự khoá tìm kiêm, xử lý mẫu tin này, tiếp tục như vậy đến tận khi đạt tới mẫu tin có Branch_name khác với Perryridge.

Đối với chỉ mục thura, đầu tiên tìm trong chỉ mục, đầu vào có Branch_name lớn nhất trong các đầu vào có Branch_name nhỏ hơn hoặc bằng Perryridge, ta tìm được đầu vào với Mianus, lần theo con trỏ tương ứng đến mẫu tin dữ liệu, đi theo con trỏ trong mẫu tin Mianus để định vị mẫu tin kế trong thứ tự khoá tìm kiêm và cứ như vậy đến tận khi đạt tới mẫu tin dữ liệu Perryridge đầu tiên, sau đó xử lý bắt đầu từ điểm này.

Chỉ mục dày cho phép tìm kiêm mẫu tin nhanh hơn chỉ mục thura, song chỉ mục thura lại đòi hỏi ít không gian hơn chỉ mục đặc. Hơn nữa, chỉ mục thura yêu cầu một tốn phí duy trì nhỏ hơn đối với các hoạt động xen và xoá.

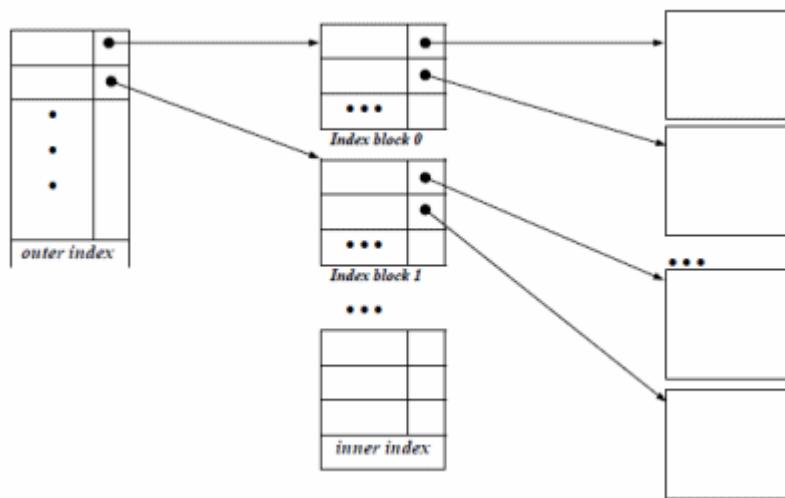
Người thiết kế hệ thống phải cân nhắc sự cân đối giữa thời truy xuất và tốn phí không gian. Một thoả hiệp tốt là có một chỉ mục thura với một đầu vào chỉ mục cho mỗi khối, vì như vậy cái giá nổi trội trong xử lý một yêu cầu CSDL là thời gian mang một khối từ đĩa vào bộ nhớ chính. Mỗi khi một khối được mang vào, thời gian quét toàn bộ khối là không đáng kể. Sử dụng chỉ mục thura, ta tìm khối chứa mẫu tin cần tìm. Như vậy, trừ phi mẫu tin nằm trên khối tràn, ta tối thiểu hoá được truy xuất khối, trong khi giữ được kích cỡ của chỉ mục nhỏ như có thể.

Chỉ mục nhiều mức

Chỉ mục có thể rất lớn, ngay cả khi sử dụng chỉ mục thura, và không thể chứa đủ trong bộ nhớ một lần. Tìm kiếm đầu vào chỉ mục đối với các chỉ mục như vậy đòi hỏi phải đọc vài khối đĩa. Tìm kiếm nhị phân có thể được sử dụng để tìm một đầu vào trên tập tin chỉ mục, song vẫn phải truy xuất khoảng $\lceil \log_2 B \rceil$ khối, với B là số khối đĩa chứa chỉ mục. Nếu B lớn, thời gian truy xuất này là đáng kể! Hơn nữa nếu sử dụng các khối tràn, tìm kiếm nhị phân không sử dụng được và như vậy việc tìm kiếm phải làm tuần tự. Nó đòi hỏi truy xuất lên đến B khối

Để giải quyết vấn đề này, Ta xem tập tin chỉ mục như một tập tin tuần tự và xây dựng chỉ mục thura cho nó. Để tìm đầu vào chỉ mục, ta tìm kiếm nhị phân trên chỉ mục "ngoài" để được mẫu tin có khoá tìm kiêm lớn nhất trong các mẫu tin có khoá tìm kiêm

nhỏ hơn hoặc bằng khoá muôn tìm. Con trỏ tương ứng trỏ tới khói của chỉ mục "trong". Trong khói này, tìm kiếm mẫu tin có khoá tìm kiếm lớn nhất trong các mẫu tin có khoá tìm kiếm nhỏ hơn hoặc bằng khoá muôn tìm, trường con trỏ của mẫu tin này trỏ đến khói chứa mẫu tin cần tìm. Vì chỉ mục ngoài nhỏ, có thể nằm sẵn trong bộ nhớ chính, nên một lần tìm kiếm chỉ cần một truy xuất khói chỉ mục. Ta có thể lặp lại quá trình xây dựng trên nhiều lần khi cần thiết. Chỉ mục với không ít hơn hai mức được gọi là chỉ mục nhiều mức. Với chỉ mục nhiều mức, việc tìm kiếm mẫu tin đòi hỏi truy xuất khói ít hơn đáng kể so với tìm kiếm nhị phân.



Hình 5.9: Chỉ mục nhiều mức

Cập nhật chỉ mục

Mỗi khi xen hoặc xoá một mẫu tin, bắt buộc phải cập nhật các chỉ mục kèm với tập tin chứa mẫu tin này. Dưới đây, ta mô tả các thuật toán cập nhật cho các chỉ mục một mức :

- **Xoá:** Để xoá một mẫu tin, đầu tiên phải tìm mẫu tin muôn xoá. Nếu mẫu tin bị xoá là mẫu tin đầu tiên trong dây chuyền các mẫu tin được xác định bởi con trỏ của đầu vào chỉ mục trong quá trình tìm kiếm, có hai trường hợp phải xét: nếu mẫu tin bị xoá là mẫu tin duy nhất trong dây chuyền, ta xoá đầu vào trong chỉ mục tương ứng, nếu không, ta thay thế khoá tìm kiếm trong đầu vào chỉ mục bởi khoá tìm kiếm của mẫu tin kè sau mẫu tin bị xoá trong dây chuyền, con trỏ bởi địa chỉ mẫu tin kè sau đó. Trong trường hợp khác, việc xoá mẫu tin không dẫn đến việc điều chỉnh chỉ mục.

- **Xen:** Trước tiên, tìm kiếm dựa trên khoá tìm kiếm của mẫu tin được xen. Nếu là chỉ mục đặc và giá trị khoá tìm kiếm không xuất hiện trong chỉ mục, xen giá trị khoá

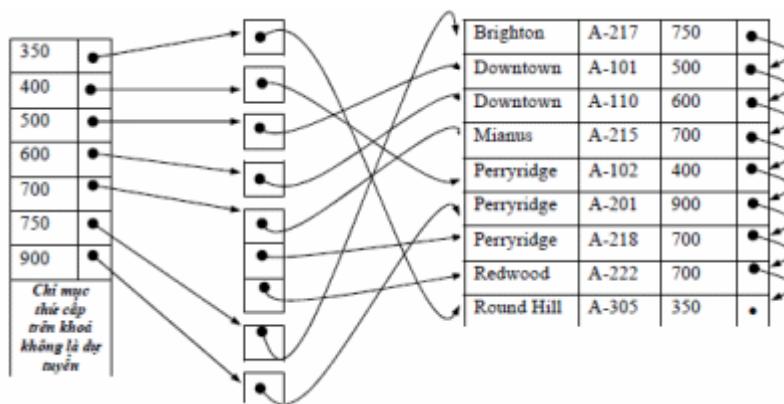
này và con trỏ tới mẫu tin vào chỉ mục. Nếu là chỉ mục thưa và lưu đầu vào cho mỗi khôi, không cần thiết phải thay đổi trừ phi khôi mới được tạo ra. Trong trường hợp đó, giá trị khoá tìm kiếm đầu tiên trong khôi mới được xen vào chỉ mục.

Giả thuật xen và xoá đối với chỉ mục nhiều mức là một mở rộng đơn giản của các giả thuật vừa được mô tả.

V.5.4.2 Chỉ mục thứ cấp

Chỉ mục thứ cấp trên một khoá dự tuyển giống như chỉ mục sơ cấp đặc ngoại trừ các mẫu tin được trả đến bởi các giá trị liên tiếp trong chỉ mục không được lưu trữ tuần tự. Nói chung, chỉ mục thứ cấp có thể được cấu trúc khác với chỉ mục sơ cấp. Nếu khoá tìm kiếm của chỉ mục sơ cấp không là khoá dự tuyển, chỉ mục chỉ cần trả đến mẫu tin đầu tiên với một giá trị khoá tìm kiếm riêng là đủ (các mẫu tin khác cùng giá trị khoá này có thể tìm lại được nhờ quét tuần tự tập tin).

Nếu khoá tìm kiếm của một chỉ mục thứ cấp không là khoá dự tuyển, việc trả tới mẫu tin đầu tiên với giá trị khoá tìm kiếm riêng không đủ, do các mẫu tin trong tập tin không còn được sắp tuần tự theo khoá tìm kiếm của chỉ mục thứ cấp, chúng có thể nằm ở bất kỳ vị trí nào trong tập tin. Bởi vậy, chỉ mục thứ cấp phải chứa tất cả các con trỏ tới mỗi mẫu tin. Ta có thể sử dụng mức phụ gián tiếp để thực hiện chỉ mục thứ cấp trên các khoá tìm kiếm không là khoá dự tuyển. Các con trỏ trong chỉ mục thứ cấp như vậy không trực tiếp trả tới mẫu tin mà trả tới một bucket chứa các con trỏ tới tập tin.



Hình 5.10: Chỉ mục thứ cấp

Chỉ mục thứ cấp phải là đặc, với một đầu vào chỉ mục cho mỗi mẫu tin. Chỉ mục thứ cấp cải thiện hiệu năng các vấn tin sử dụng khoá tìm kiếm không là khóa của chỉ mục sơ cấp, tuy nhiên nó lại đem lại một tổn phí sửa đổi CSDL đáng kể. Việc quyết

định các chỉ mục thứ cấp nào là cần thiết dựa trên đánh giá của nhà thiết kế CSDL về tần xuất vấn tin và sửa đổi.

V.5.5 Tập tin chỉ mục B^+ -cây (B^+ -tree index file)

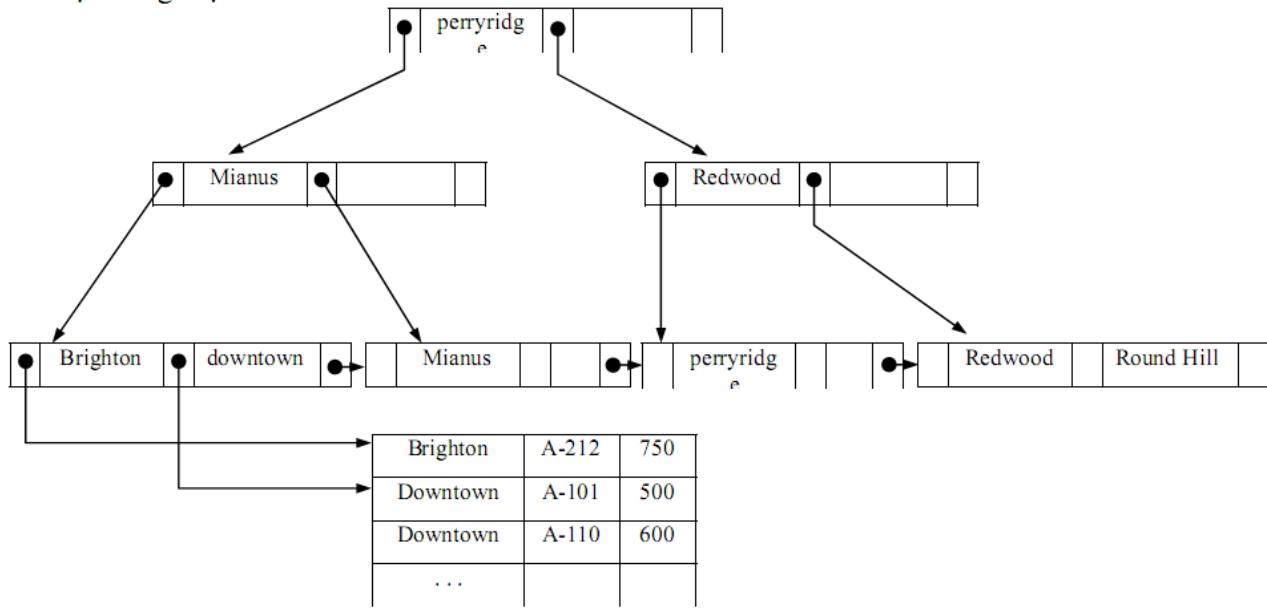
Tổ chức tập tin chỉ mục tuần tự có một nhược điểm chính là làm giảm hiệu năng khi tập tin lớn lên. Để khắc phục nhược điểm đó đòi hỏi phải tổ chức lại tập tin. Cấu trúc chỉ mục B^+ -cây là cấu trúc được sử dụng rộng rãi nhất trong các cấu trúc đảm bảo được tính hiệu quả của chúng bất chấp các hoạt động xen, xoá. Chỉ mục B^+ -cây là một dạng cây cân bằng (mọi đường dẫn từ gốc đến lá có cùng độ dài). Mỗi nút không lá có số con nằm trong khoảng giữa $[m/2]$ và m , trong đó m là một số cố định được gọi là bậc của B^+ -cây. Ta thấy rằng cấu trúc B^+ -cây cũng đòi hỏi một tổn phí hiệu năng trên xen và xoá cũng như trên không gian. Tuy nhiên, tổn phí này là chấp nhận được ngay cả đối với các tập tin có tần suất sửa đổi cao.

V.5.5.1 Cấu trúc của B^+ -cây

Một chỉ mục B^+ -cây là một chỉ mục nhiều mức, nhưng có cấu trúc khác với tập tin tuần tự chỉ mục nhiều mức (multilevel index-sequential). Một nút tiêu biểu của B^+ -cây chứa đến $n-1$ giá trị khoá tìm kiếm. K_1, K_2, \dots, K_{n-1} , và n con trỏ P_1, P_2, \dots, P_n , các giá trị khoá trong nút được sắp thứ tự: $i < j \Rightarrow K_i < K_j$.

P_1	K_1	P_2	K_2	...	P_{n-1}	K_{n-1}	P_n
-------	-------	-------	-------	-----	-----------	-----------	-------

Trước tiên, ta xét cấu trúc của nút lá. Đối với $i = 1, 2, \dots, n-1$, con trỏ P_i trỏ tới hoặc mẫu tin với giá trị khoá K_i hoặc tới một bucket các con trỏ mà mỗi một trong chúng trỏ tới một mẫu tin với giá trị khoá K_i . Cấu trúc bucket chỉ được sử dụng trong các trường hợp: hoặc khoá tìm kiếm không là khoá sơ cấp hoặc tập tin không được sắp theo khoá tìm kiếm. Con trỏ P_n được dùng vào mục đích đặc biệt: P_n được dùng để móc xích các nút lá lại theo thứ tự khoá tìm kiếm, điều này cho phép xử lý tuần tự tập tin hiệu quả. Nay giờ ta xem các giá trị khoá tìm kiếm được gắn với một nút lá như thế nào. Mỗi nút nút lá có thể chứa đến $n-1$ giá trị. Khoảng giá trị mà mỗi nút lá chứa là không chồng chéo. Như vậy, nếu L_i và L_j là hai nút lá với $i < j$ thì mỗi giá trị khoá trong nút L_i nhỏ hơn mọi giá trị khoá trong L_j . Nếu chỉ mục B^+ -cây là đặc, mỗi giá trị khoá tìm kiếm phải xuất hiện trong một nút lá nào đó.



Hình 5.11: Cây chỉ mục B-Tree

Các nút không lá của một B^+ -cây tạo ra một chỉ mục nhiều mức trên các nút lá. Cấu trúc của các nút không lá tương tự như cấu trúc nút lá ngoại trừ tất cả các con trỏ đều trỏ đến các nút của cây. Các nút không lá có thể chứa đến m con trỏ và phải chứa không ít hơn $\lceil m/2 \rceil$ con trỏ ngoại trừ nút gốc. Nút gốc được phép chứa ít nhất 2 con trỏ. Số con trỏ trong một nút được gọi là số fanout (fanout) của nút.

Con trỏ P_i của một nút không lá (chứa p con trỏ, $1 < i < p$) trỏ đến một cây con chứa các giá trị khoá tìm kiếm nhỏ hơn K_i và lớn hơn hoặc bằng K_{i-1} . Con trỏ P_1 trỏ đến cây con chứa các giá trị khoá tìm kiếm nhỏ hơn K_1 . Con trỏ P_p trỏ tới cây con chứa các khoá tìm kiếm lớn hơn K_{p-1} .

V.5.5.2 Các vấn tin trên B^+ -cây

Ta xét xử lý vấn tin sử dụng B^+ -cây như thế nào? Giả sử ta muốn tìm tất cả các mẫu tin với giá trị khoá tìm kiếm k . Đầu tiên, ta kiểm tra nút gốc, tìm giá trị khoá tìm kiếm nhỏ nhất lớn hơn k , giả sử giá trị khoá đó là K_i . Đi theo con trỏ P_i để di tới một nút khác. Nếu nút có p con trỏ và $k > K_{p-1}$, đi theo con trỏ P_p . Đến một nút tới, lặp lại quá trình tìm kiếm giá trị khoá tìm kiếm nhỏ nhất lớn hơn k và theo con trỏ tương ứng để di tới một nút khác và tiếp tục như vậy đến khi đạt tới một nút lá. Con trỏ tương ứng trong nút lá hướng ta tới mẫu tin/bucket mong muốn.

Số khối truy xuất không vượt quá: $\lceil \log_{m/2} K \rceil$

Trong đó K là số giá trị khoá tìm kiếm trong B^+ -cây, m là bậc của cây.

V.5.5.3 Cập nhật trên B⁺-cây

- Xen: Sử dụng cùng kỹ thuật như tìm kiếm, ta tìm nút lá trong đó giá trị khoá tìm kiếm cần xen sẽ xuất hiện. Nếu khoá tìm kiếm đã xuất hiện rồi trong nút lá, xen mẫu tin vào trong tập tin, thêm con trỏ tới mẫu tin vào trong bucket tương ứng. Nếu khoá tìm kiếm chưa hiện diện trong nút lá, ta xen mẫu tin vào trong tập tin rồi xen giá trị khoá tìm kiếm vào trong nút lá ở vị trí đúng (bảo tồn tính thứ tự), tạo một bucket mới với con trỏ tương ứng. Nếu nút lá không còn chỗ cho giá trị khoá mới, Một khối mới được yêu cầu từ hệ điều hành, các giá trị khoá trong nút lá được tách một nửa cho nút mới, giá trị khoá mới được xen vào vị trí đúng của nó vào một trong hai khối này. Điều này kéo theo việc xen giá trị khoá đầu khối mới và con trỏ tới khối mới vào nút cha. Việc xen cặp giá trị khoá và con trỏ vào nút cha này lại có thể dẫn đến việc tách nút ra làm hai. Quá trình này có thể dẫn đến tận nút gốc. Trong trường hợp nút gốc bị tách làm hai, một nút gốc mới được tạo ra và hai con của nó là hai nút được tách ra từ nút gốc cũ, chiều cao cây tăng lên một.

- Xoá: Sử dụng kỹ thuật tìm kiếm tìm mẫu tin cần xoá, xoá nó khỏi tập tin, xoá giá trị khoá tìm kiếm khỏi nút lá trong B⁺-cây nếu không có bucket kết hợp với giá trị khoá tìm kiếm hoặc bucket trỏ nên rỗng sau khi xoá con trỏ tương ứng trong nó. Việc xoá một giá trị khoá khỏi một nút của B⁺-cây có thể dẫn đến nút lá trỏ nên rỗng, phải trả lại, từ đó nút cha của nó có thể có số con nhỏ hơn ngưỡng cho phép, trong trường hợp đó hoặc phải chuyển một con từ nút anh em của nút cha đó sang nút cha nếu điều đó có thể (nút anh em của nút cha này còn số con $\geq [m/2]$ sau khi chuyển đi một con). Nếu không, phải gom nút cha này với một nút anh em của nó, điều này dẫn tới xoá một nút trong khối cây, rồi xoá khỏi nút cha của nó một hạng, ... quá trình này có thể dẫn đến tận gốc. Trong trường hợp nút gốc chỉ còn một con sau xoá, cây phải thay nút gốc cũ bởi nút con của nó, nút gốc cũ phải trả lại cho hệ thống, chiều cao cây giảm đi một.

V.5.5.4 Tổ chức tập tin B⁺-cây

Trong tổ chức tập tin B⁺-cây, các nút lá của cây lưu trữ các mẫu tin, thay cho các con trỏ tới tập tin. Vì mẫu tin thường lớn hơn con trỏ, số tối đa các mẫu tin được lưu trữ trong một khối lá ít hơn số con trỏ trong một nút không lá. Các nút lá vẫn được yêu cầu được lắp đầy ít nhất là một nửa.

Xen và xoá trong tổ chức tập tin B⁺-cây tương tự như trong chỉ mục B⁺-cây.

Khi B^+ -cây được sử dụng để tổ chức tập tin, việc sử dụng không gian là đặc biệt quan trọng, vì không gian bị chiếm bởi mẫu tin là lớn hơn nhiều so với không gian bị chiếm bởi (khoá, con trỏ). Ta có thể cải tiến sự sử dụng không gian trong B^+ -cây bằng cách bao hàm nhiều nút anh em hơn khi tái phân phôi trong khi tách và trộn. Khi xen, nếu một nút là đầy, ta thử phân phôi lại một số đầu vào đến một trong các nút kè để tạo không gian cho đầu vào mới. Nếu việc thử này thất bại, ta mới thực hiện tách nút và phân chia các đầu vào giữa một trong các nút kè và hai nút nhận được do tách nút. Khi xoá, nếu nút chứa ít hơn $\lfloor m/3 \rfloor$ đầu vào, ta thử mượn một đầu vào từ một trong hai nút anh em kè. Nếu cả hai đều có đúng $\lfloor m/3 \rfloor$ mẫu tin, ta phân phôi lại các đầu vào của nút cho hai nút anh em kè và xoá nút thứ 3. Nếu k nút được sử dụng trong tái phân phôi ($k-1$ nút anh em), mỗi nút đảm bảo chứa ít nhất $\lfloor (k-1)m/k \rfloor$ đầu vào. Tuy nhiên, cái giá phải trả cho cập nhật của cách tiếp cận này sẽ cao hơn.

V.5.5.5 Tập tin chỉ mục B-cây (B-Tree Index file)

Chỉ mục B-cây tương tự như chỉ mục B^+ -cây. Sự khác biệt là ở chỗ B-cây loại bỏ lưu trữ dữ liệu các giá trị khoá tìm kiếm. Trong B-cây, các giá trị khoá chỉ xuất hiện một lần. Do các khoá tìm kiếm xuất hiện trong các nút không lá không xuất hiện ở bất kỳ nơi nào khác nữa trong B-cây, ta phải thêm một trường con trỏ cho mỗi khoá tìm kiếm trong các nút không lá. Con trỏ thêm vào này trỏ tới hoặc mẫu tin trong tập tin hoặc bucket tương ứng.

Một nút lá B-cây tổng quát có dạng:

P ₁	K ₁	P ₂	K ₂	...	P _{m-1}	K _{m-1}	P _m
----------------	----------------	----------------	----------------	-----	------------------	------------------	----------------

Một nút không lá có dạng:

P ₁	B ₁	K ₁	P ₂	B ₂	K ₂	...	P ₁	B ₁	K ₁	P ₁
----------------	----------------	----------------	----------------	----------------	----------------	-----	----------------	----------------	----------------	----------------

Các con trỏ P_i là các con trỏ cây và được dùng như trong B^+ -cây. Các con trỏ B_i trong các nút không lá là các con trỏ mẫu tin hoặc con trỏ bucket. Rõ ràng là số giá trị khoá trong nút không lá nhỏ hơn số giá trị trong nút lá. Số nút được truy xuất trong quá trình tìm kiếm trong một B-cây phụ thuộc nơi khoá tìm kiếm được định vị.

Xoá trong một B-cây phức tạp hơn trong một B^+ -cây. Xoá một đầu vào xuất hiện ở một nút không lá kéo theo việc tuyển chọn một giá trị thích hợp trong cây con của nút chứa đầu vào bị xoá. Nếu khoá K_i bị xoá, khoá nhỏ nhất trong cây con được trỏ

bởi P_{i+1} phải được di chuyển vào vị trí của K_i . Nếu nút lá còn lại quá ít đầu vào, cần thiết các hoạt động bô xung.

V.5.6 Băm (HASHING)

Bất lợi của tổ chức tập tin tuần tự là ta phải truy xuất một cấu trúc chỉ mục để định vị dữ liệu, hoặc phải sử dụng tìm kiếm nhị phân và kết quả là có nhiều hoạt động I/O. Tổ chức tập tin dựa trên kỹ thuật băm cho phép ta tránh được truy xuất một cấu trúc chỉ mục. Băm cung cấp một phương pháp để xây dựng các chỉ mục.

Băm tĩnh (Static Hashing)

V.5.6.1 Tổ chức tập tin băm

Trong tổ chức tập tin băm, ta nhận được địa chỉ của khối đĩa chứa một mẫu tin mong muốn bởi tính toán một hàm trên giá trị khoá tìm kiếm của mẫu tin. thuật ngữ bucket được dùng để chỉ một đơn vị lưu trữ. Một bucket kiểu mẫu là một khối đĩa, nhưng có thể được chọn nhỏ hơn hoặc lớn hơn một khối đĩa.

K ký hiệu tập tất cả các giá trị khoá tìm kiếm, **B** ký hiệu tập tất cả các địa chỉ bucket. Một hàm băm h là một hàm từ **K** vào **B**: $K \rightarrow B$.

Xem một mẫu tin với giá trị khoá K vào trong tập tin: ta tính $h(K)$. Giá trị của $h(K)$ là địa chỉ của bucket sẽ chứa mẫu tin. Nếu có không gian trong bucket cho mẫu tin, mẫu tin được lưu trữ trong bucket.

Tìm kiếm một mẫu tin theo giá trị khoá K : đầu tiên tính $h(K)$, ta tìm được bucket tương ứng. sau đó tìm trong bucket này mẫu tin với giá trị khoá K mong muốn.

Xoá mẫu tin với giá trị khoá K : tính $h(K)$, tìm trong bucket tương ứng mẫu tin mong muốn, xoá nó khỏi bucket.

Hàm băm

Hàm băm xấu nhất là hàm ánh xạ tất cả các giá trị khoá vào cùng một bucket. Hàm băm lý tưởng là hàm phân phối đều các giá trị khoá vào các bucket, như vậy mỗi bucket chứa một số lượng mẫu tin như nhau. Ta muốn chọn một hàm băm thoả mãn các tiêu chuẩn sau:

- ♦ **Phân phối đều:** Mỗi bucket được gán cùng một số giá trị khoá tìm kiếm trong tập hợp tất cả các giá trị khoá có thể.

♦ **Phân phối ngẫu nhiên:** Trong trường hợp trung bình, các bucket được gán một số lượng giá trị khoá tìm kiếm gần bằng nhau.

Các hàm băm phải được thiết kế thận trọng. Một hàm băm xấu có thể dẫn đến việc tìm kiếm chiếm một thời gian tỷ lệ với số khoá tìm kiếm trong tập tin.

Điều khiển tràn bucket

Khi xen một mẫu tin, nếu bucket tương ứng còn chỗ, mẫu tin được xen vào bucket, nếu không sẽ xảy ra tràn bucket. Tràn bucket do các nguyên do sau:

♦ **Các bucket không đủ:** Số các bucket n_B phải thoả mãn $n_B > n_r / f_r$ trong đó n_r là tổng số mẫu tin sẽ lưu trữ, f_r là số mẫu tin có thể lấp đầy trong một bucket.

♦ **Sự lệch:** Một vài bucket được gán cho một số lượng mẫu tin nhiều hơn các bucket khác, như vậy một bucket có thể tràn trong khi các bucket khác vẫn còn không gian. Tình huống này được gọi là sự lệch bucket. Sự lệch xảy ra do hai nguyên nhân:

1/ Nhiều mẫu tin có cùng khoá tìm kiếm.

2/ Hàm băm được chọn phân phối các giá trị khoá không đều.

Ta quản lý tràn bucket bằng cách dùng các bucket tràn. Nếu một mẫu tin phải được xen vào bucket B nhưng bucket B đã đầy, khi đó một bucket tràn sẽ được cấp cho B và mẫu tin được xen vào bucket tràn này. Nếu bucket tràn cũng đầy một bucket tràn mới lại được cấp và cứ như vậy. Tất cả các bucket tràn của một bucket được "móc xích" với nhau thành một danh sách liên kết. Việc điều khiển tràn dùng danh sách liên kết như vậy được gọi là dây chuyền tràn. Đối với dây chuyền tràn, thuật toán tìm kiếm thay đổi chút ít: trước tiên ta cũng tính giá trị hàm băm trên khoá tìm kiếm, ta được bucket B, kiểm tra các mẫu tin, trong bucket B và tất cả các bucket tràn tương ứng, có giá trị khoá khớp với giá trị tìm không.

Một cách điều khiển tràn bucket khác là: Khi cần xen một mẫu tin vào một bucket nhưng nó đã đầy, thay vì cấp thêm một bucket tràn, ta sử dụng một hàm băm ké trong một dãy các hàm băm được chọn để tìm bucket khác cho mẫu tin, nếu bucket sau cũng đầy, ta lại sử dụng một hàm băm ké và cứ như vậy... Dãy các hàm băm thường được sử dụng là $\{h_i(K) = (h_{i-1}(K) + 1) \bmod n_B \text{ với } 1 \leq i \leq n_B - 1 \text{ và } h_0 \text{ là hàm băm cơ sở}\}$.

Dạng cấu trúc băm sử dụng dây chuyền bucket được gọi là băm mở. Dạng sử dụng dây các hàm băm được gọi là băm đóng. Trong các hệ CSDL, cấu trúc băm đóng thường được ưu dùng hơn.

Chỉ mục băm

Một chỉ mục băm tổ chức các khoá tìm kiếm cùng con trỏ kết hợp vào một cấu trúc tập tin băm như sau: áp dụng một hàm băm trên khoá tìm kiếm để định danh bucket sau đó lưu giá trị khoá và con trỏ kết hợp vào bucket này (hoặc vào các bucket tràn). Chỉ mục băm thường là chỉ mục thứ cấp.

Hàm băm trên số tài khoản được tính theo công thức:

$$h(\text{Account_number}) = (\text{tổng các chữ số trong số tài khoản}) \bmod 7$$

V.5.6.2 Băm động (Dynamic Hashing)

Trong kỹ thuật băm tĩnh (static hashing), tập **B** các địa chỉ bucket phải là cố định. Các CSDL phát triển lớn lên theo thời gian. Nếu ta sử dụng băm tĩnh cho CSDL, ta có ba lớp lựa chọn:

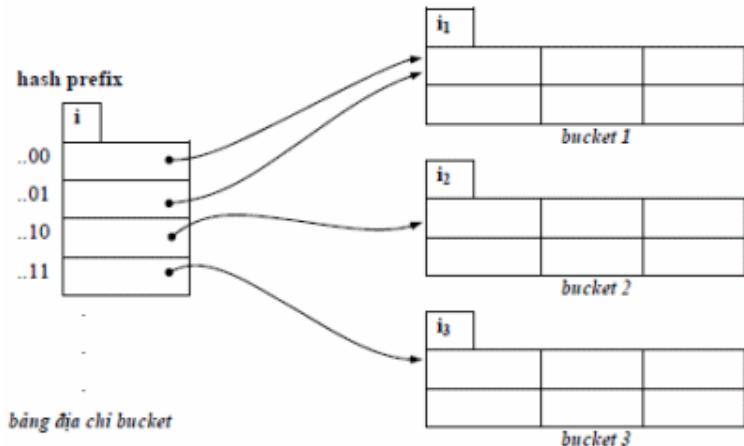
1/ Chọn một hàm băm dựa trên kích cỡ tập tin hiện hành. Sự lựa chọn này sẽ dẫn đến sự suy giảm hiệu năng khi CSDL lớn lên.

2/ Chọn một hàm băm dựa trên kích cỡ tập tin dự đoán trước cho một thời điểm nào đó trong tương lai. Mặc dù sự suy giảm hiệu năng được cải thiện, một lượng đáng kể không gian có thể bị lãng phí lúc khởi đầu.

3/ Tổ chức lại theo chu kỳ cấu trúc băm đáp ứng sự phát triển kích cỡ tập tin. Một sự tổ chức lại như vậy kéo theo việc lựa chọn một hàm băm mới, tính lại hàm băm trên mỗi mẫu tin trong tập tin và sinh ra các gán bucket mới. Tổ chức lại là một hoạt động tốn thời gian. Hơn nữa, nó đòi hỏi cảm truy xuất tập tin trong khi đang tổ chức lại tập tin.

Kỹ thuật băm động cho phép sửa đổi hàm băm để phù hợp với sự tăng hoặc giảm của CSDL. Một dạng băm động được gọi là băm có thể mở rộng (extendable hashing) được thực hiện như sau: Chọn một hàm băm h với các tính chất đều, ngẫu nhiên và có miền giá trị tương đối rộng, chẳng hạn, là một số nguyên b bit (b thường là 32). Khi khởi đầu ta không sử dụng toàn bộ b bit giá trị băm. Tại một thời điểm, ta chỉ sử dụng i bit $0 \leq i \leq b$. i bit này được dùng như một độ dời (offset) trong một bảng địa chỉ bucket phụ. giá trị i tăng lên hay giảm xuống tùy theo kích cỡ CSDL.

Số i xuất hiện bên trên bảng địa chỉ bucket chỉ ra rằng i bit của giá trị băm h(K) được đòi hỏi để xác định bucket đúng cho K, số này sẽ thay đổi khi kích cỡ tập tin thay đổi. Mặc dù i bit được đòi hỏi để tìm đầu vào đúng trong bảng địa chỉ bucket, một số đầu vào bảng kè nhau có thể trỏ đến cùng một bucket. Tất cả các như vậy có chung hash prefix chung, nhưng chiều dài của prefix này có thể nhỏ hơn i. Ta kết hợp một số nguyên chỉ độ dài của hash prefix chung này, ta sẽ ký hiệu số nguyên kết hợp với bucket j là i_j . Số các đầu vào bảng địa chỉ bucket trỏ đến bucket là $2^{(i-i_j)}$.



Hình 5.12: Cấu trúc băm có thể mở rộng tổng quát

Để định vị bucket chứa giá trị khoá tìm kiếm K, ta lấy i bit cao nhất tiên của h(K), tìm trong đầu vào bảng tương ứng với chuỗi bit này và lần theo con trỏ trong đầu vào bảng này. Để xen một mẫu tin với giá trị khoá tìm kiếm K, tiến hành thủ tục định vị trên, ta được bucket, giả sử là bucket j. Nếu còn cho cho mẫu tin, xen mẫu tin vào trong bucket đó. Nếu không, ta phải tách bucket ra và phân phối lại các mẫu tin hiện có cùng mẫu tin mới. Để tách bucket, đầu tiên ta xác định từ giá trị băm có cần tăng số bit lên hay không.

- ♦ Nếu $i = i_j$, chỉ có một đầu vào trong bảng địa chỉ bucket trỏ đến bucket j, ta cần tăng kích cỡ của bảng địa chỉ bucket sao cho ta có thể bao hàm các con trỏ đến hai bucket kết quả của việc tách bucket j bằng cách xét thêm một bit của giá trị băm tăng giá trị i lên một, như vậy kích cỡ của bảng địa chỉ bucket tăng lên gấp đôi. Mỗi một đầu vào được thay bởi hai đầu vào, cả hai cùng chứa con trỏ của đầu vào gốc. Bây giờ hai đầu vào trong bảng địa chỉ bucket trỏ tới bucket j. Ta định vị một bucket mới (bucket z), và đặt đầu vào thứ hai trỏ tới bucket mới, đặt i_j và i_z về i, tiếp theo đó mỗi một mẫu tin trong bucket j được băm lại, tùy thuộc vào i bit đầu tiên, sẽ hoặc ở lại bucket j hoặc được cấp phát cho bucket mới được tạo.

♦ Nếu $i > j$, khi đó nhiều hơn một đầu vào trong bảng địa chỉ bucket trả tới bucket j , như vậy ta có thể tách bucket j mà không cần tăng kích cỡ bảng địa chỉ bucket. Ta cấp phát một bucket mới (bucket z) và đặt i_j và i_z đến giá trị là kết quả của việc thêm 1 vào giá trị i_j gốc.

Kế đến, ta điều chỉnh các đầu vào trong bảng địa chỉ bucket trước đây trả tới bucket j . Ta để lại nửa đầu các đầu vào và đặt tất cả các đầu vào còn lại trả tới bucket mới tạo (z). Tiếp theo, mỗi mẫu tin trong bucket j được băm lại và được cấp phát cho hoặc vào bucket j hoặc bucket z .

Để xoá một mẫu tin với giá trị khoá K , trước tiên ta thực hiện thủ tục định vị, ta tìm được bucket tương ứng, gọi là j , ta xoá cả khoá tìm kiếm trong bucket lần mẫu tin mẫu tin trong tập tin. bucket cũng bị xoá, nếu nó trở nên rỗng. Chú ý rằng, tại điểm này, một số bucket có thể được kết hợp lại và kích cỡ của bảng địa chỉ bucket sẽ giảm đi một nửa.

Ưu điểm chính của băm có thể mở rộng là hiệu năng không bị suy giảm khi tập tin tăng kích cỡ, hơn nữa, tổng phí không gian là tối thiểu mặc dù phải thêm vào không gian cho bảng địa chỉ bucket. Một khuyết điểm của băm có thể mở rộng là việc tìm kiếm phải bao hàm một mức gián tiếp: ta phải truy xuất bảng địa chỉ bucket trước khi truy xuất đến bucket. Vì vậy, băm có thể mở rộng là một kỹ thuật rất hấp dẫn.

V.5.7 Chọn chỉ mục hay băm?

Ta đã xét qua các sơ đồ: chỉ mục thứ tự, băm. Ta có thể tổ chức tập tin các mẫu tin bởi hoặc sử dụng tổ chức tập tin tuần tự chỉ mục, hoặc sử dụng B+-cây, hoặc sử dụng băm ... Mỗi sơ đồ có những ưu điểm trong các tình huống nhất định. Một nhà thực thi hệ CSDL có thể cung cấp nhiều sơ đồ, để lại việc quyết định sử dụng sơ đồ nào cho nhà thiết kế CSDL.

Để có một sự lựa chọn khôn ngoan, nhà thực thi hoặc nhà thiết kế CSDL phải xét các yếu tố sau:

- ♦ Cái giá phải trả cho việc tổ chức lại theo định kỳ của chỉ mục hoặc băm có chấp nhận được hay không?
- ♦ Tần số tương đối của các hoạt động xen và xoá là bao nhiêu?
- ♦ Có nên tối ưu hoá thời gian truy xuất trung bình trong khi thời gian truy xuất thường hợp xấu nhất tăng lên hay không?

♦ Các kiểu văn tin mà các người sử dụng thích đặt ra là gì?

❖ Câu hỏi (bài tập) củng cố:

1. Mô tả cấu trúc lưu trữ dữ liệu trên đĩa
2. Trình bày các kỹ thuật của bộ quản trị Buffer
3. Trình bày các cách tổ chức file:
 - a. Mẫu tin có chiều dài cố định
 - b. Mẫu tin có chiều dài thay đổi
4. Trình bày các kiểu tổ chức dữ liệu trên file:
 - a. File tuần tự
 - b. File cụm
 - c. Chỉ mục
 - d. File chỉ mục B⁺ - cây
 - e. Băm

CHƯƠNG 6

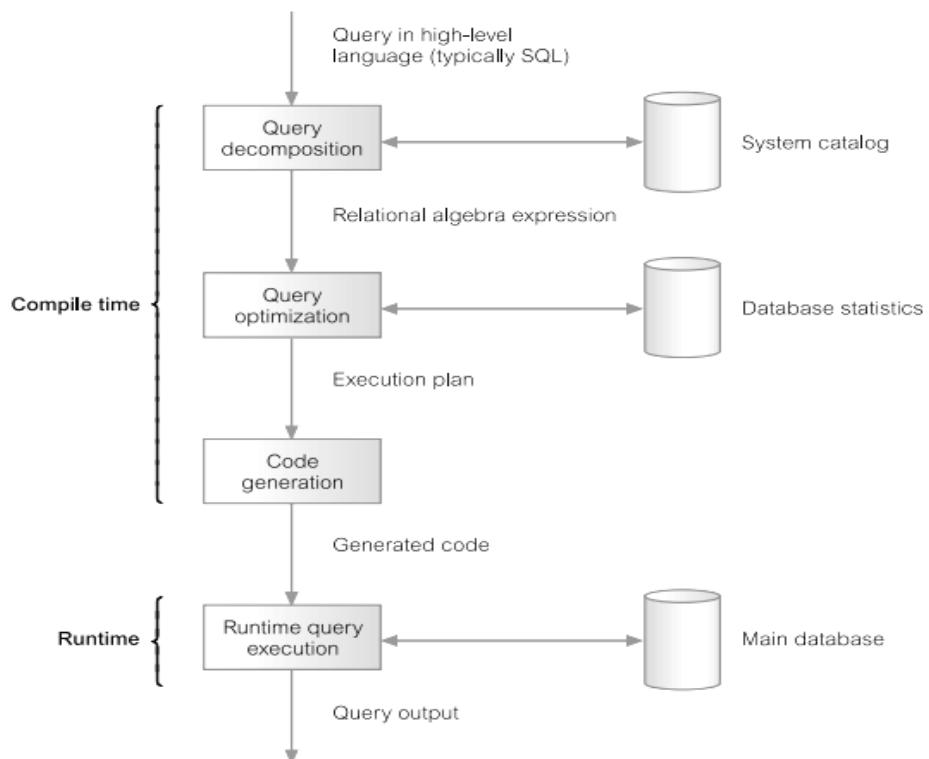
ƯỚC LƯỢNG CHI PHÍ THỰC HIỆN CÂU TRUY VẤN

- ❖ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:
- Ước lượng chi phí thực hiện phép toán trên **một** quan hệ
 - Ước lượng chi phí thực hiện phép toán trên **nhiều** quan hệ

VI.1 XỬ LÝ TRUY VẤN (QUERY PROCESSING)

Mục đích của xử lý truy vấn là để biến đổi một câu truy vấn được viết bằng một ngôn ngữ cấp cao (thường là SQL) sang cách thực hiện hiệu quả thể hiện bằng ngôn ngữ cấp thấp (thực thi bằng đại số quan hệ) và để thực hiện các chiến lược truy xuất dữ liệu cần thiết.

Quá trình xử lý truy vấn được chia thành bốn giai đoạn chính gồm: Phân tích (decomposition: gồm phân tích cú pháp (parsing) và xác nhận tính hợp lệ (validation)), tối ưu câu (optimization), sinh mã (Code generation) và thực thi câu truy vấn (Runtime query execution) như minh họa trong *Hình 6.1*.

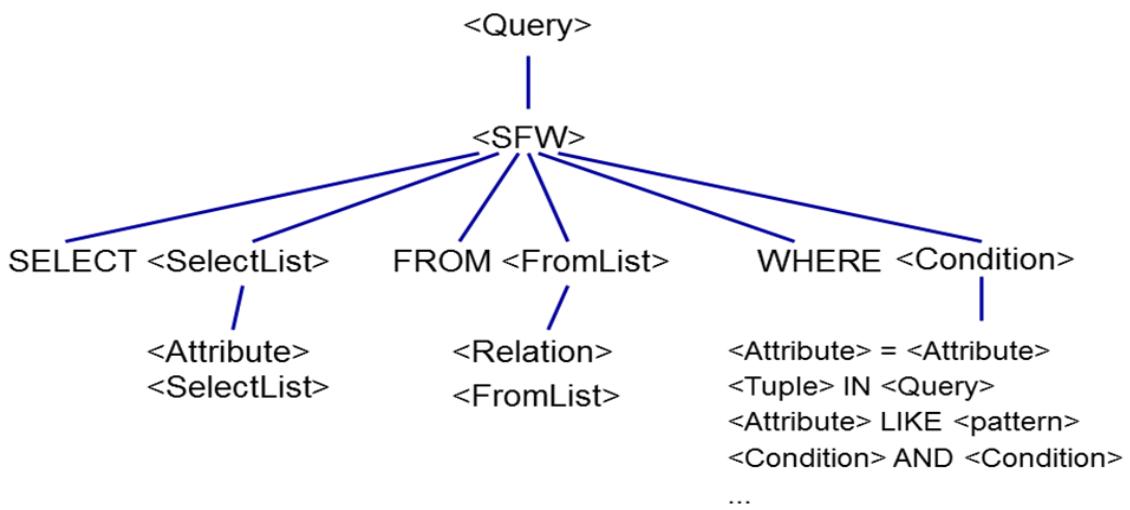


Hình 6.1: Các giai đoạn trong quá trình xử lý câu truy vấn

VI.1.1 Phân tích câu truy vấn

Phân tích truy vấn là giai đoạn đầu tiên của xử lý truy vấn. Mục đích phân tích truy vấn là để biến đổi một câu truy vấn được viết bằng ngôn ngữ cấp cao vào một truy vấn được viết bằng đại số quan hệ và để kiểm tra xem câu truy vấn có đúng cú pháp và ngữ nghĩa hay không. Các giai đoạn điển hình của phân tích truy vấn là: phân tích, chuẩn hóa (normalization), phân tích ngữ nghĩa, rút gọn (simplification) và sắp xếp lại câu truy vấn.

VI.1.1 .1 Cây phân tích



Hình 6.2: Cây phân tích

Cho hai quan hệ:

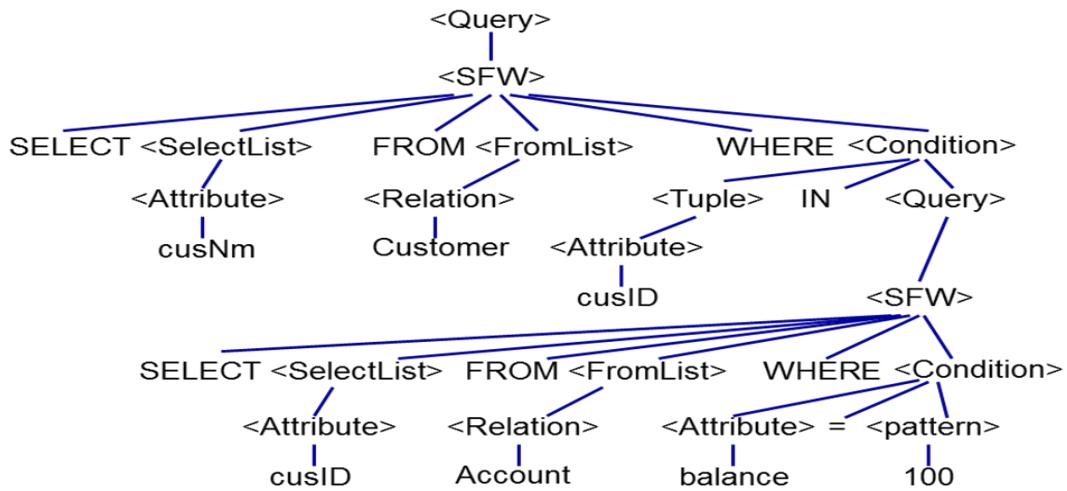
1. CUSTOMER (cusID, cusNm, cusStreet, cusCity)
2. ACCOUNT (accID, cusID, balance)

Ví dụ 1:

Cho câu truy vấn sau:

```
SELECT cusNm  
FROM CUSTOMER  
WHERE cusID IN  
(  
    SELECT cusID  
    FROM ACCOUNT  
    WHERE balance = 100  
)
```

Ta có cây phân tích như *Hình 6.3*



Hình 6.3: Cây phân tích cho câu truy vấn lồng

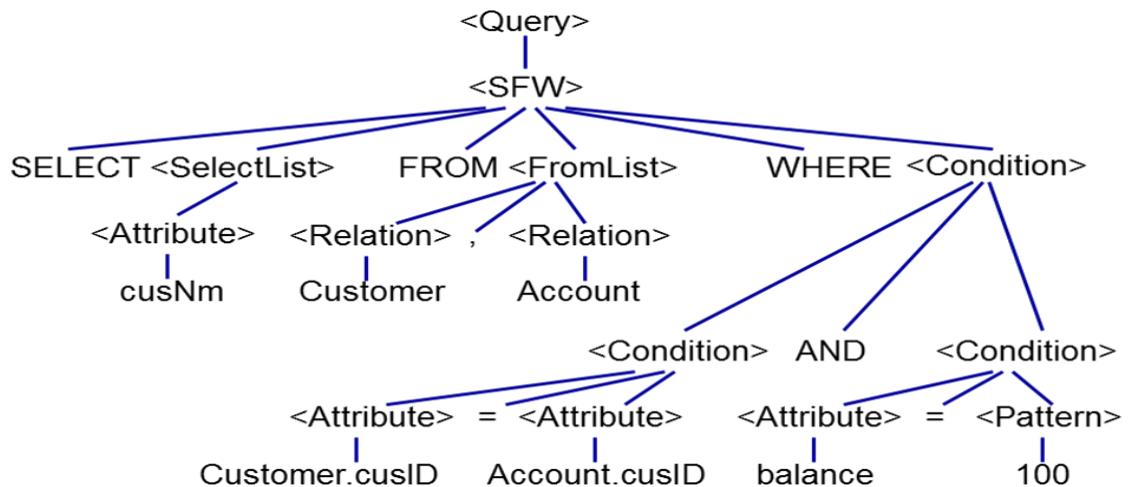
Ví dụ 2:

Cho câu truy vấn sau

```

SELECT cusNm
FROM CUSTOMER, ACCOUNT
WHERE Customer.cusID = Account.cusID
AND balance = 100
  
```

Ta có câu phân tích sau:



Hình 6.4: Cây phân tích cho câu truy vấn đơn

VI.1.1.2 Kiểm tra ngữ nghĩa

- Kiểm tra Quan hệ, thuộc tính trên mệnh đề **FROM** và **SELECT**
- Kiểm tra kiểu dữ liệu trên mệnh đề **WHERE**

Ví dụ:

Cho câu truy vấn sau:

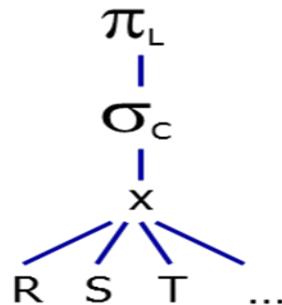
```
SELECT staffNumber  
FROM Staff  
WHERE position > 10;
```

- Kiểm tra thuộc tính **staffNumber** trong mệnh đề **SELECT** có phải thuộc tính được định nghĩa trong quan hệ **staff** ở mệnh đề **FROM** hay không.
- Trong mệnh đề **WHERE**, kiểm tra so sánh **> 10** có cùng kiểu dữ liệu với kiểu dữ liệu của thuộc tính **position** hay không.

VI.1.2 Biến đổi sang đại số quan hệ

Truy vấn đơn

- Xét cấu trúc **<SFW>**
 - Thay thế **<FromList>** thành các biến quan hệ
 - Sử dụng phép tích cartesian cho các biến quan hệ
 - Thay thế **<Condition>** thành phép chọn σ_C
 - Thay thế **<SelectList>** thành phép chiếu π_L

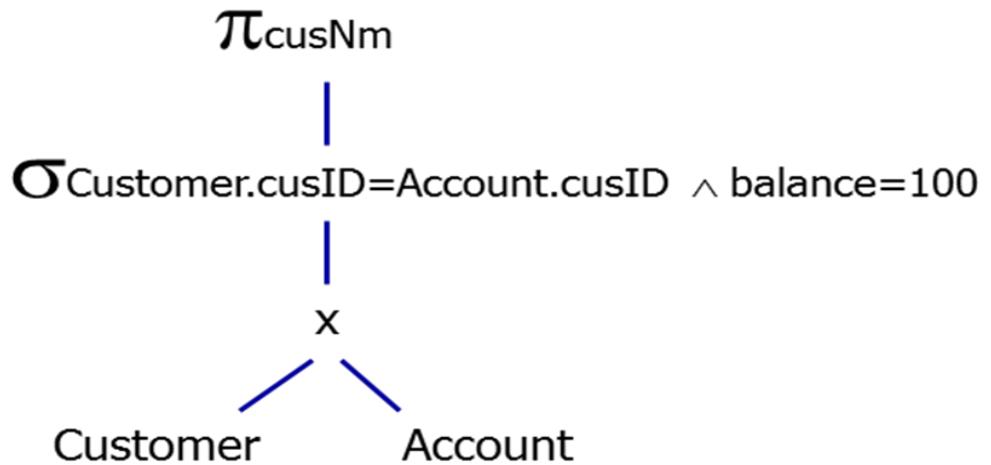


Hình 6.5: Cây truy vấn

Xét lại ví dụ 2:

```
SELECT cusNm  
FROM Customer, Account  
WHERE Customer.cusID = Account.cusID  
AND balance = 100
```

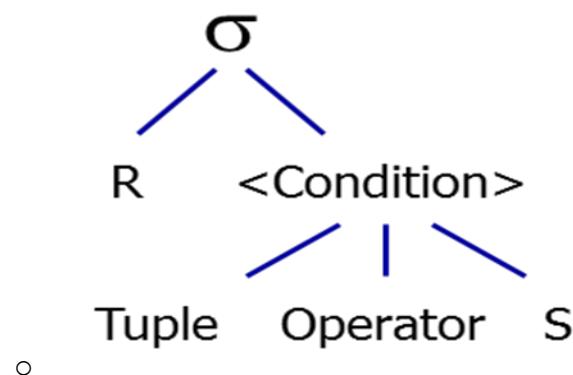
Áp dụng các quy tắc biến đổi ta được cây truy vấn như *Hình 6.6*



Hình 6.6: Cây truy vấn được chuyển từ câu truy vấn

- **Truy vấn lồng**

- Tồn tại câu truy vấn con S trong **<Condition>**
- Áp dụng qui tắc **<SFW>** cho truy vấn con
- Phép chọn 2 biến (two-argument selection)
 - Nút là phép chọn không có tham số
 - Nhánh con trái là biến quan hệ R
 - Nhánh con phải là **<condition>** áp dụng cho mỗi bộ trong S

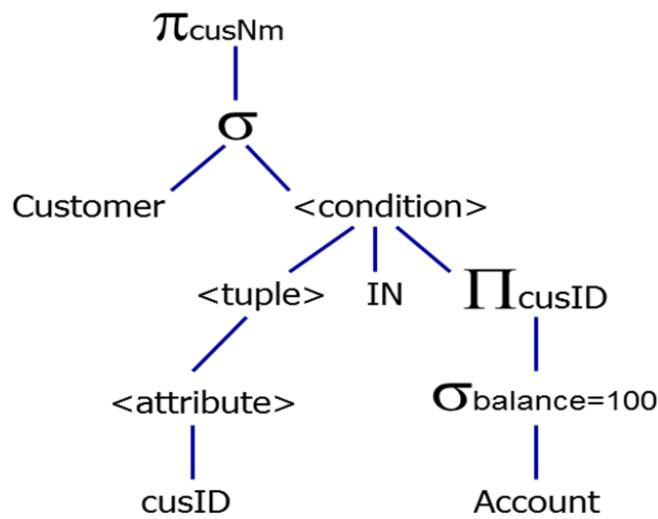


Hình 6.7a: Cây truy vấn cho câu truy vấn lồng

Xét ví dụ 1:

```
SELECT cusNm  
FROM Customer  
WHERE cusID IN (  
    SELECT cusID  
    FROM Account  
    WHERE balance = 100)
```

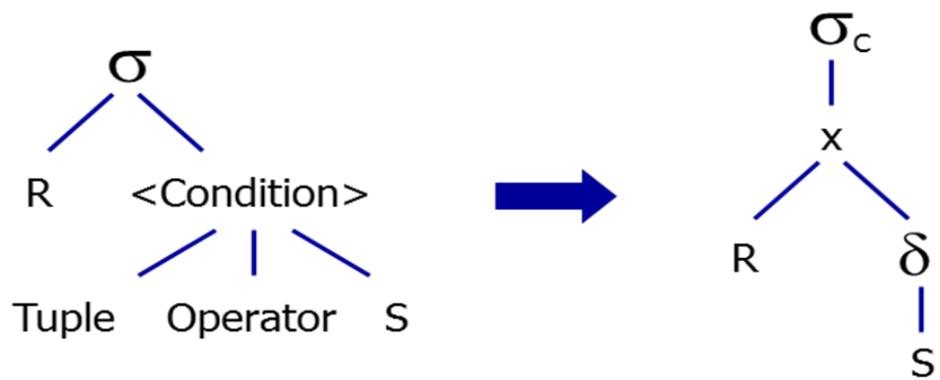
Áp dụng các quy tắc trên ta được câu truy vấn như *Hình 6.7b* sau:



Hình 6.7b: Cây truy vấn lồng

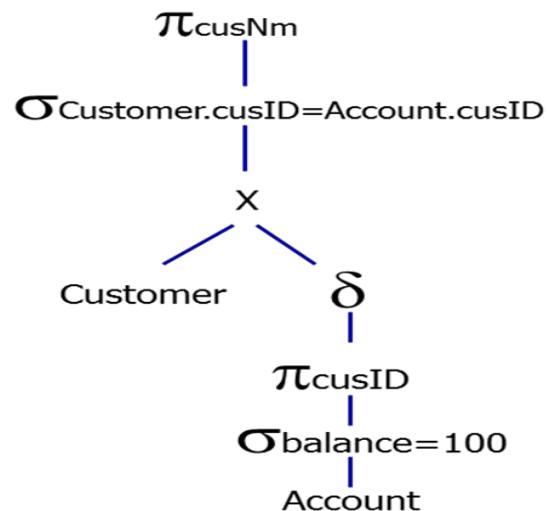
Biến đổi phép chọn 2 biến

- Thay thế <Condition> bằng 1 cây có gốc là S
 - Nếu S có các bộ trùng nhau thì phải lược bỏ bớt bộ trùng nhau đi
 - Sử dụng phép δ
- Thay thế phép chọn 2 biến thành σ_C
- σ_C là kết quả của phép cartesian của R và S

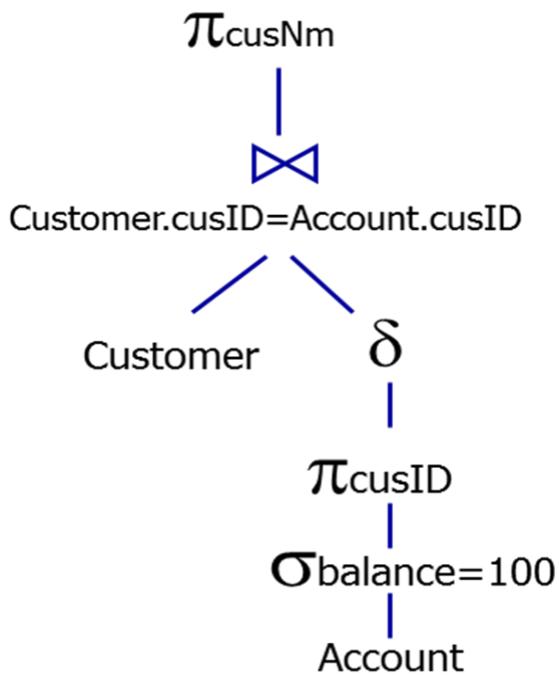


Hình 6.8: Biến đổi phép chọn 2 biến

Ta có cây truy vấn sau như *Hình 6.9a, b*



Hình 6.9a: Cây truy vấn



Hình 6.9b: Cây truy vấn

VI.2 TỐI ƯU CÂU TRUY VẤN

Áp dụng các luật để tối ưu hóa câu truy vấn:

VI.2.1 Qui tắc: Kết tự nhiên, tích cartesian, hội

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$R \times S = S \times R$$

$$(R \times S) \times T = R \times (S \times T)$$

$$R \cup S = S \cup R$$

$$(R \cup (S \cup T)) = (R \cup S) \cup T$$

VI.2.2 Qui tắc: Phép chọn σ

Cho

- a. p là vị từ chỉ có các thuộc tính của R
- b. q là vị từ chỉ có các thuộc tính của S
- c. m là vị từ có các thuộc tính của R và S

$$\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1}[\sigma_{p_2}(R)]$$

$$\sigma_{p_1 \vee p_2}(R) = [\sigma_{p_1}(R)] \cup [\sigma_{p_2}(R)]$$

VI.2.3 Qui tắc: σ ,

$$\sigma_p(R \diamondsuit S) = [\sigma_p(R)] \diamondsuit S$$

$$\sigma_q(R \diamondsuit S) = R \diamondsuit [\sigma_q(S)]$$

$$\sigma_{p \wedge q}(R \diamondsuit S) = [\sigma_p(R)] \diamondsuit [\sigma_q(S)]$$

$$\sigma_{p \wedge q \wedge m}(R \diamondsuit S) = \sigma_m[\sigma_p(R) \diamondsuit \sigma_q(S)]$$

$$\sigma_{p \vee q}(R \diamondsuit S) = [\sigma_p(R) \diamondsuit S] \cup [R \diamondsuit \sigma_q(S)]$$

VI.2.4 Qui tắc: σ , \cup và $\sigma, -$

$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$

$$\sigma_C(R - S) = \sigma_C(R) - S = \sigma_C(R) - \sigma_C(S)$$

VI.2.5 Qui tắc: Phép chiếu π

Cho

$$X = \text{tập thuộc tính con của } R$$

$$Y = \text{tập thuộc tính con của } R$$

Ta có

$$XY = X \cup Y$$

$$\pi_{XY}(R) \neq \pi_X[\pi_Y(R)]$$

VI.2.6 Qui tắc: π ,

Cho

$$X = \text{tập thuộc tính con của } R$$

$$Y = \text{tập thuộc tính con của } S$$

$$Z = \text{tập giao thuộc tính của } R \text{ và } S$$

$$\pi_{XY}(R \diamondsuit S) = \pi_{XY}[\pi_{XZ}(R) \diamondsuit \pi_{YZ}(S)]$$

VI.2.7 Qui tắc: σ, π

Cho

$$X = \text{tập thuộc tính con của } R$$

$Z = \text{tập thuộc tính con của } R \text{ xuất hiện trong vị từ } p$

$$\pi_X [(\sigma_p (R))] = \pi_X \{\sigma_p [\pi_{XZ} (R)]\}$$

VI.2.8 Qui tắc: σ, π

Cho

$X = \text{tập thuộc tính con của } R$

$Y = \text{tập thuộc tính con của } S$

$Z = \text{tập giao thuộc tính của } R \text{ và } S$

$Z' = Z \cup \{\text{các thuộc tính xuất hiện trong vị từ } p\}$

$$\pi_{XY} [\sigma_p (R \bowtie Z' S)] = \pi_{XY} \{\sigma_p [\pi_{XZ'} (R) \bowtie \pi_{YZ'} (S)]\}$$

Nhận xét: σ, π

- Bình thường
- Chiếu trước
- Nhung
- Giả sử A và B được cài đặt chỉ mục (index)
- Physical query plan dùng chỉ mục để chọn ra những bộ có $A=3$ và $B='a'$ trước
- Nếu thực hiện chiếu trước $\pi_{AB}(R)$ thì chỉ mục trên A và B là vô ích
- Chọn trước

→ Thông thường chọn trước tốt hơn

VI.2.9 Qui tắc: \times

$$[\sigma_C (R \bowtie S)] = R \bowtie_C S$$

$$R \times S = \pi_L [\sigma_C (R \times S)]$$

VI.2.10 Qui tắc: δ

$$\delta (R \bowtie S) = \delta (R) \bowtie \delta (S)$$

$$\delta (R \times S) = \delta (R) \times \delta (S)$$

$$\delta (\sigma_C (R)) = \sigma_C [\delta (R)]$$

$$\delta (R \cap_B S) = \delta (R) \cap_B S = R \cap_B \delta (S) = \delta (R) \cap_B \delta (S)$$

VI.2.11 Qui tắc: γ

- Cho

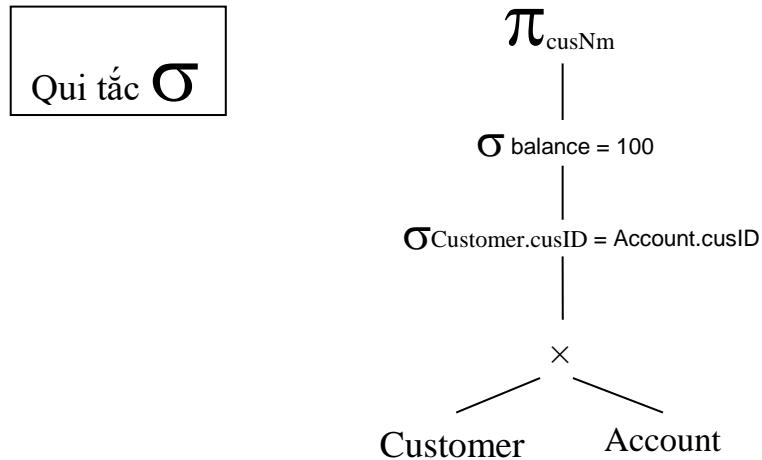
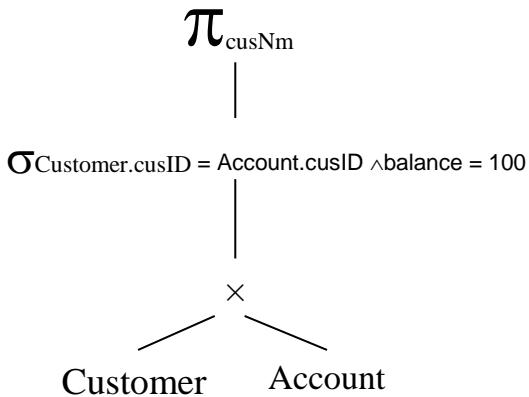
- $X = \text{tập thuộc tính trong } R \text{ được gom nhóm}$
- $Y = X \cup \{\text{một số thuộc tính khác của } R\}$

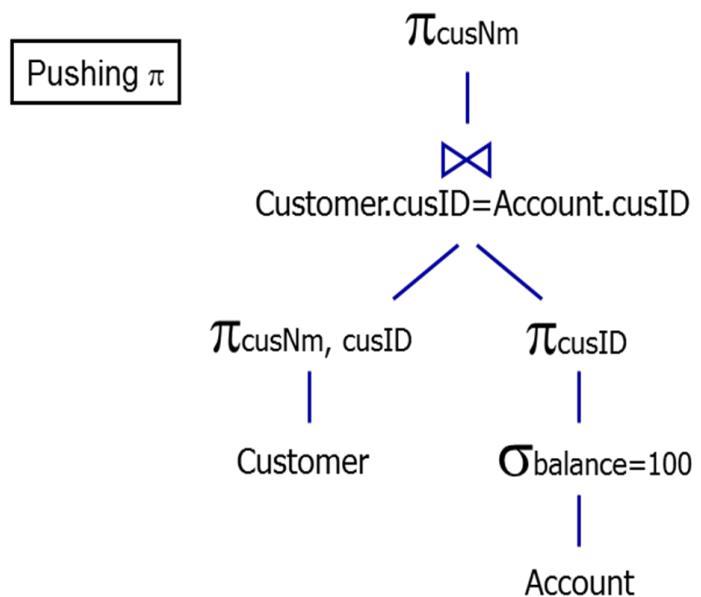
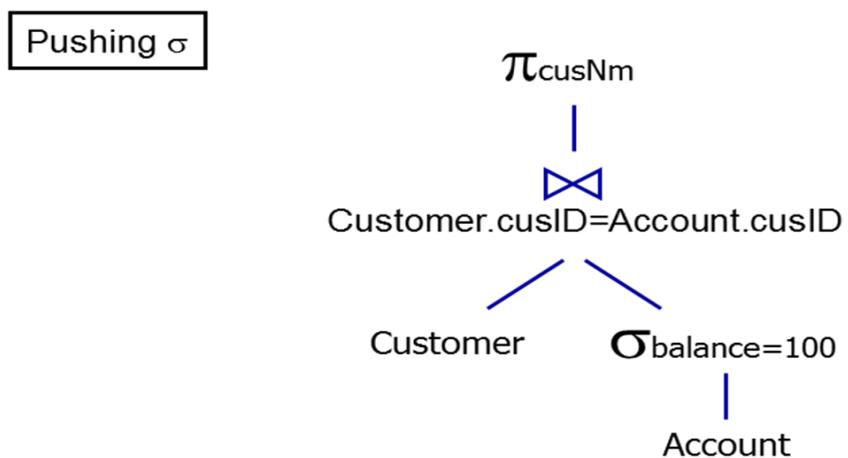
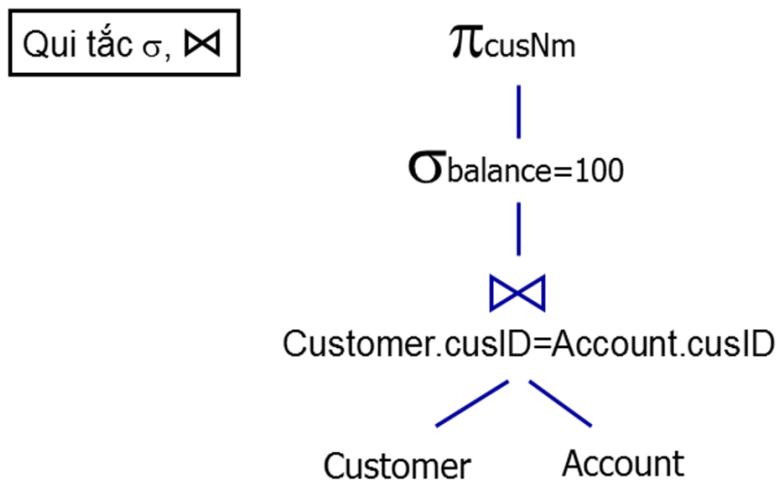
$$\delta [\gamma_X(R)] = \gamma_X(R)$$

$$\gamma_X(R) = \gamma_X[\pi_L(R)]$$

Xét lại ví dụ 2 có câu truy vấn sau

```
SELECT cusNm
FROM CUSTOMER, ACCOUNT
WHERE Customer.cusID = Account.cusID
AND balance = 100
```





VI.3 ƯỚC LƯỢNG CHI PHÍ

Sự thành công của việc ước tính kích thước và chi phí của các phép toán đại số quan hệ trung gian phụ thuộc vào số lượng và sự lưu hành của các thông tin thống kê mà các DBMS đang có.

VI.3.1 Thống kê cơ sở dữ liệu

- Thống kê quan hệ R

- $T(R)$: số bộ trong R
- $S(R)$: tổng số byte của 1 bộ trong R
- $B(R)$: tổng số block chứa tất cả các bộ của R
- $V(R, A)$: số giá trị khác nhau mà thuộc tính A trong R có thể có

Ví dụ:

Cho quan hệ R với các thể hiện như sau:

R	A	B	C	D
X	1	10	a	
X	1	20	b	
Y	1	30	a	
Y	1	40	c	
Z	1	50	d	

Trong đó:

- A: chuỗi 20 bytes
- B: số nguyên 4 bytes
- C: ngày 8 bytes
- D: chuỗi 68 bytes

1 block = 1024 bytes (block header: 24 bytes)

Tính được:

$$T(R) = 5$$

$$S(R) = 100$$

$$B(R) = 1$$

$$V(R, A) = 3$$

$$V(R, B) = 1$$

$$V(R, C) = 5$$

$$V(R, D) = 4$$

VI.3.2 Uớc lượng chi phí thực hiện

VI.3.2.1 Uớc lượng: $W = \sigma_{Z=val}(R)$

$$S(W) = S(R)$$

$$T(W) = T(R) / V(R, Z)$$

(Số bộ trung bình thỏa điều kiện $Z=val$)

VI.3.2.2 Uớc lượng: $W = \sigma_{Z \geq val}(R)$

Cách 1

$$T(W) = T(R) / 2$$

Cách 2

$$T(W) = T(R) / 3$$

Ví dụ:

Cho

- $R(A, B, C)$
- $T(R) = 10000$
- $V(R, A) = 50$
- Uớc lượng kích thước biểu thức $S = \sigma_{A=10 \wedge B<20}(R)$

$$T(S) = T(R) / (V(R, A) \times 3) = 10000 / (50 \times 3) = 67$$

VI.3.2.3 Uớc lượng: $W = R_1 \times R_2$

$$S(W) = S(R_1) + S(R_2)$$

$$T(W) = T(R_1) \times T(R_2)$$

VI.3.2.4 Uớc lượng: $W = R_1 \bowtie R_2$

● Cho

- $X = \text{tập thuộc tính của } R_1$

- $Y = \text{tập thuộc tính của } R_2$
- Xét trường hợp $X \cap Y = \emptyset$

$$T(W) = R_1 \times R_2$$

- Xét trường hợp $X \cap Y = A$

R_1	<table border="1"> <tr> <td>A</td><td>B</td><td>C</td></tr> <tr> <td></td><td></td><td></td></tr> </table>	A	B	C				R_2	<table border="1"> <tr> <td>A</td><td>D</td></tr> <tr> <td></td><td></td></tr> </table>	A	D		
A	B	C											
A	D												

- $V(R_1, A) \leq V(R_2, A)$
 - Mọi giá trị của A trong R_1 thì có trong R_2
 - $T(W) = T(R_1) \times [T(R_2) / V(R_2, A)]$
- $V(R_2, A) \leq V(R_1, A)$
 - Mọi giá trị của A có trong R_2 thì có trong R_1
 - $T(W) = T(R_2) \times [T(R_1) / V(R_1, A)]$
- 1 bộ trong R_1 sẽ thỏa với $T(R_2) / V(R_2, A)$ bộ trong R_2
 - $T(W) = T(R_1) \times [T(R_2) / V(R_2, A)]$
- Tổng quát
 - $T(W) = T(R_1) T(R_2) / \max\{V(R_1, A), V(R_2, A)\}$
- Tính $W(A, B, C, D)$
 - Các thuộc tính không tham gia vào phép kết thì số lượng các giá trị vẫn giữ nguyên
 - $V(W, A) = \min \{V(R_1, A), V(R_2, A)\}$
 - $V(W, B) = V(R_1, B)$
 - $V(W, C) = V(R_1, C)$
 - $V(W, D) = V(R_2, D)$

Ví dụ:

$$\text{Cho } Z = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$$

Và các giá trị trên R_1 , R_2 và R_3 như sau

R₁	R₂	R₃
$T(R_1) = 1000$ $V(R_1, A) = 50$ $V(R_1, B) = 100$	$T(R_2) = 2000$ $V(R_2, B) = 200$ $V(R_2, C) = 300$	$T(R_3) = 3000$ $V(R_3, C) = 90$ $V(R_3, D) = 500$

Giải

$$U = R_1(A, B) \triangleleft\triangleright R_2(B, C)$$

$$T(U) = (1000 \times 2000) / 200$$

$$V(U, A) = 50$$

$$V(U, B) = 100$$

$$V(U, C) = 300$$

$$Z = U \triangleleft\triangleright R_3(C, D)$$

$$T(Z) = (1000 \times 2000 \times 3000) / (200 \times 300)$$

$$V(Z, A) = 50$$

$$V(Z, B) = 100$$

$$V(Z, C) = 90$$

$$V(Z, D) = 500$$

VI.3.2.5 *Uớc lượng:* $W = R_1 \cup R_2$

- R_1 và R_2 là bag

$$T(W) = T(R_1) + T(R_2)$$

- R_1 và R_2 là set

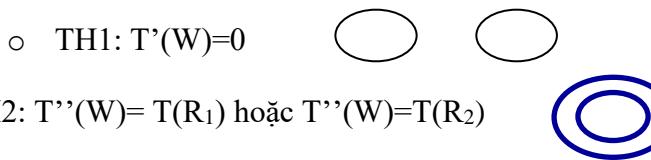
$$T'(W) = T(R_1) + T(R_2)$$

$$T''(W) \leq T(R_1) + T(R_2)$$

$$\rightarrow T(W) = [T'(W) + T''(W)] / 2$$

VI.3.2.6 *Uớc lượng:* $W = R_1 \cap R_2$

- Cách 1



$$\rightarrow T(W) = [T'(W) + T''(W)] / 2$$

● Cách 2

- Trường hợp đặc biệt của phép kết tự nhiên

Chỉ áp dụng cho \cap_s

$$T(W) = [T(R_1) T(R_2)] / \max\{V(R_1, Z), V(R_2, Z)\}$$

VI.3.2.7 *Ước lượng: $W = R_1 - R_2$*

- TH1: $T(W) = T(R_1)$
- TH2: $T(W) = T(R_1) - T(R_2)$

$$\rightarrow T(W) = T(R_1) - (\frac{1}{2}) * T(R_2)$$

VI.3.2.8 *Ước lượng: $W = \delta(R)$*

- TH1: $T(W) = 1$
 - Nếu trong R không có bộ nào thì $T(W)=0$
 - TH2: $T(W) = T(R)$
 - $R(a_1, a_2, \dots, a_n)$
 - Số bộ phân biệt tối đa của R là tích các $V(R, a_i)$, $i=1..n$
- $$\rightarrow T(W) = \min\{(1/2)*T(R), \text{tích các } V(R, a_i)\}$$

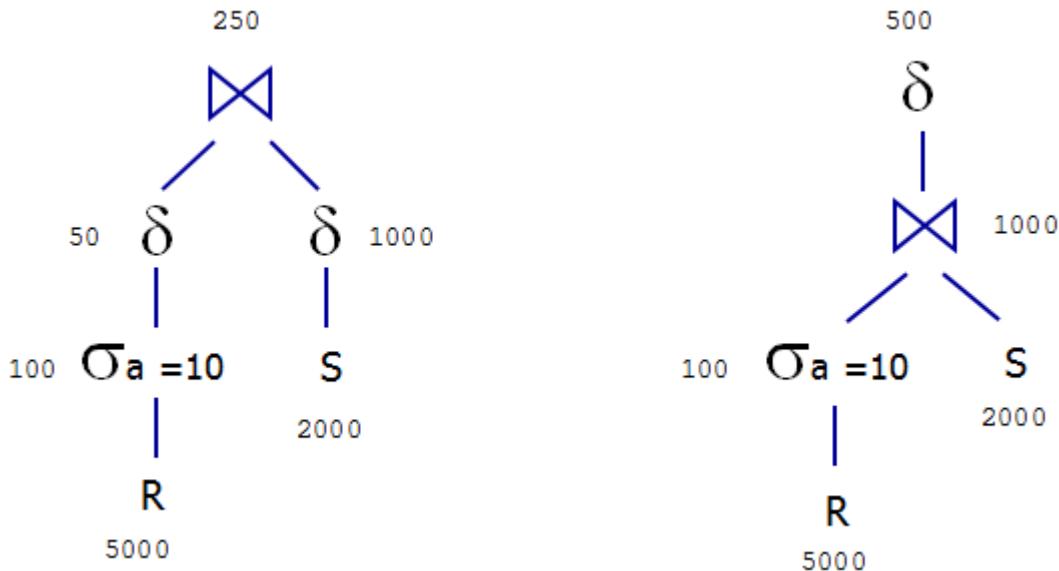
Ví dụ: Cho

● $R(a, b)$

- $T(R)=5000$
- $V(R, a)=50$
- $V(R, b)=100$

● $S(b, c)$

- $T(S)=2000$
- $V(S, b)=200$
- $V(S, c)=100$



- Cộng kích thước sau khi thực hiện các phép toán, ngoại trừ
 - Các nút lá
 - Nút gốc
 - (1): $100+50+1000=1150$
 - (2): $100+1000=1100$
- Phép lược bỏ trùng lắp thực hiện sau thì tốt hơn
- Các tham số thống kê
 - $B(R)$: tổng số block chứa tất cả các bộ của R
 - $S(R)$: số bộ tối đa trong mỗi block
 - M : số block trống trên bộ nhớ
- Quan tâm
 - Quan hệ R có được gom thành cụm không (clustered)?
 - Thuộc tính trong các phép toán có chỉ mục không (index)?
 - Chỉ mục có gom cụm không (clustering index)?
 - Kết quả cần được sắp thứ tự không?

❖ Câu hỏi (bài tập) củng cố:

1. Trình bày các giai đoạn của quá trình xử lý truy vấn?
2. Vẽ cây phân tích cho câu truy vấn sau
(a)

```
SELECT r.type, r.price
FROM Room r, Hotel h
WHERE r.hotel_number = h.hotel_number
      AND h.hotel_name = 'Grosvenor Hotel'
      AND r.type > 100;
```

(b)

```
SELECT g.guestNo, g.name
FROM Hotel h, Booking b, Guest g
WHERE h.hotelNo = b.hotelNo
      AND h.hotelName = 'Grosvenor Hotel';
```

(c)

```
SELECT r.roomNo, h.hotelNo
FROM Hotel h, Booking b, Room r
WHERE h.hotelNo = b.hotelNo
      AND h.hotelNo = 'H21'
      AND b.roomNo = r.roomNo
      AND type = 'S' AND b.hotelNo = 'H22';
```

3. Vẽ cây truy vấn cho các câu truy vấn (a), (b), (c) ở trên
4. Tối ưu hóa cây truy vấn vừa vẽ ở câu 3
5. Ước lượng kích thước của biểu thức sau

$$Z = R_1(B, C, D) \bowtie R_2(A, B, C) \bowtie R_3(D, E)$$

Và các giá trị trên R_1 , R_2 và R_3 như sau

R₁	R₂	R₃
$T(R_1) = 1000$	$T(R_2) = 2000$	$T(R_3) = 3000$
$V(R_1, B) = 50$	$V(R_2, A) = 200$	$V(R_3, D) = 90$
$V(R_1, C) = 100$	$V(R_2, B) = 300$	$V(R_3, E) = 500$
$V(R_1, D) = 300$	$V(R_2, C) = 300$	

TÀI LIỆU THAM KHẢO

❖ **TÀI LIỆU THAM KHẢO ĐỂ BIÊN SOẠN NỘI DUNG MÔN HỌC:**

- [1] Craig S. Mullin, *Database Administration The Complete Guide to DBA Practices and Procedures*, Second Edition 2013
- [2] Craig S. Mullins, *Database Administration: The Complete Guide to Practices and Procedures*, Publisher Addison Wesley, June 14, 2002
- [3] Nguyễn Thái Nghe, Trần Ngân Bình, Đăng Quốc Việt, *Giáo trình hệ quản trị cơ sở dữ liệu*, NXB Đại học Cần Thơ, 2014
- [4] Rob Vieira, *Professional Microsoft® SQL Server ® 2008 Programming*, Published by Wiley Publishing, 2009
- [5] Thomas, *Database Systems*, Printed and bound in the United States of America, Fourth edition published 2005
- [6] *Bài giảng Hệ quản trị Cơ sở dữ liệu*, Khoa Công nghệ Thông tin , Đại học Khoa học Tự nhiên TPHCM

❖ **TÀI LIỆU THAM KHẢO ĐỂ NGHỊ CHO HỌC VIÊN:**

- [7] Phần mềm SQL Server 2008
- [8] <http://www.cse.iitb.ac.in/~sudarsha/db-book/slide-dir/> để đọc tài liệu **Database System Concepts, Fifth Edition, Avi Silberschatz, Henry F. Korth, S. Sudarshan**
- [9] <http://pages.cs.wisc.edu/~dbbook/openAccess/secondEdition/solutions/answers2ed-odd.pdf>
- [10] <http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/solutions/ans3ed-oddonly.pdf>
- [11] [https://technet.microsoft.com/en-us/library/bb510424\(v=sql.100\).aspx](https://technet.microsoft.com/en-us/library/bb510424(v=sql.100).aspx)
- [12] http://www.techonthenet.com/sql_server/tables/alter_table.php

PHỤ LỤC 1: HƯỚNG DẪN SỬ DỤNG SQL SERVER 2008

1.1 Tổng quan về SQL Server 2008

SQL Server 2008 cung cấp công nghệ và khả năng mà các tổ chức hy vọng kiểm soát được các khó khăn thách thức đang ngày càng tăng đối với việc quản lý dữ liệu và cung cấp thông tin có giá trị kịp thời đến người dùng. SQL Server 2008 là một bộ phận trong toàn cảnh về nền tảng dữ liệu của Microsoft được thiết kế cho việc quản lý và làm việc với dữ liệu ngày nay là xa hơn nữa.

SQL Server 2008 là một phát hành quan trọng mang đến nhiều tính năng mới và những cải thiện quan trọng làm cho nó trở thành một phiên bản SQL Server toàn diện và mạnh mẽ với những tính năng cần thiết trong việc lưu trữ và quản lý dữ liệu bùng nổ như ngày nay.



Hình 1.1: Toàn cảnh nền tảng dữ liệu của Microsoft

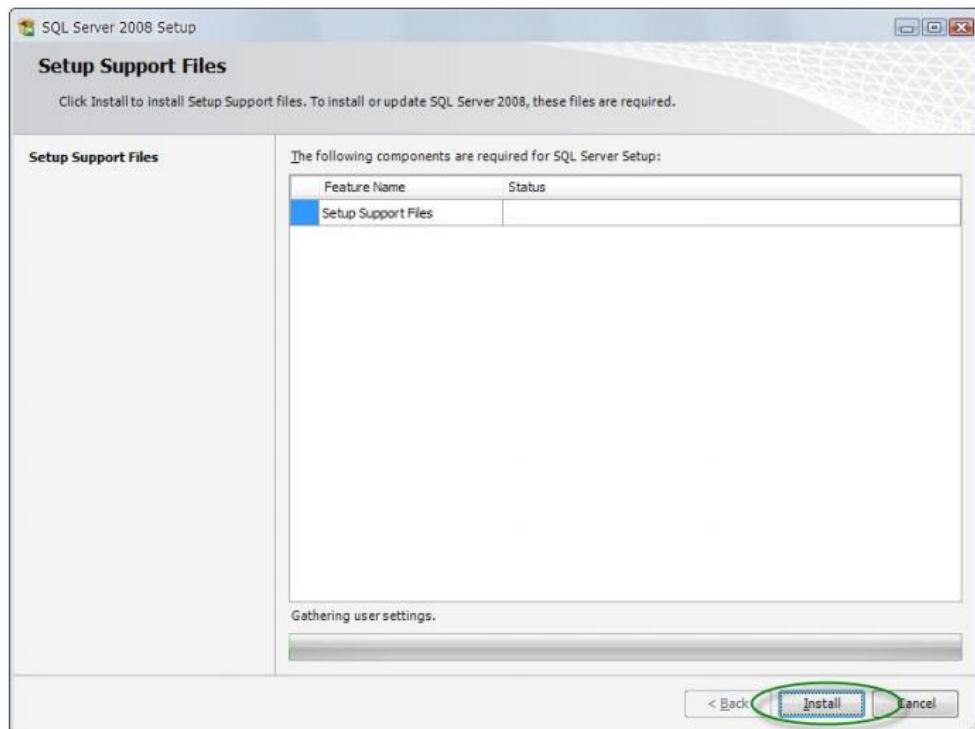
Các bước cài đặt SQL Server 2008

Bước 1: Chạy file setup.exe để cài đặt, chọn Installation -> New SQL Server stand-alone ...



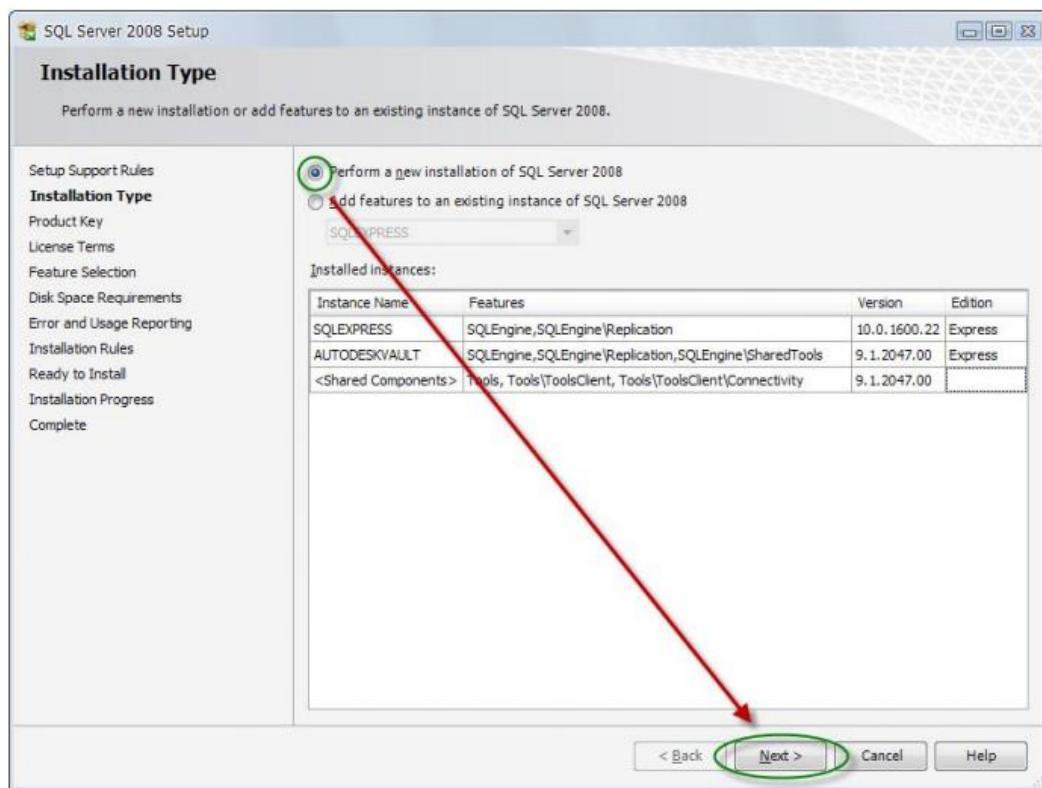
Hình 1.2: Giao diện SQL Server Installation Center

Bước 2: Chọn Ok -> Next



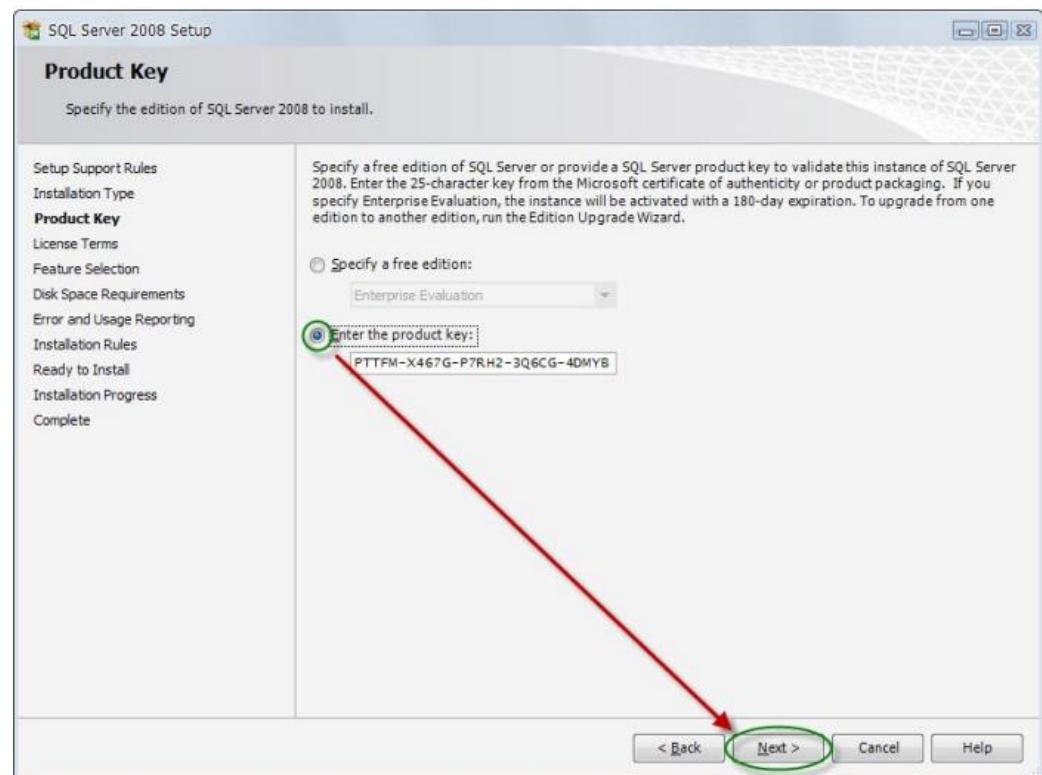
Hình 1.3: Giao diện Setup Support Files

Bước 3: Chọn kiểu cài đặt mới



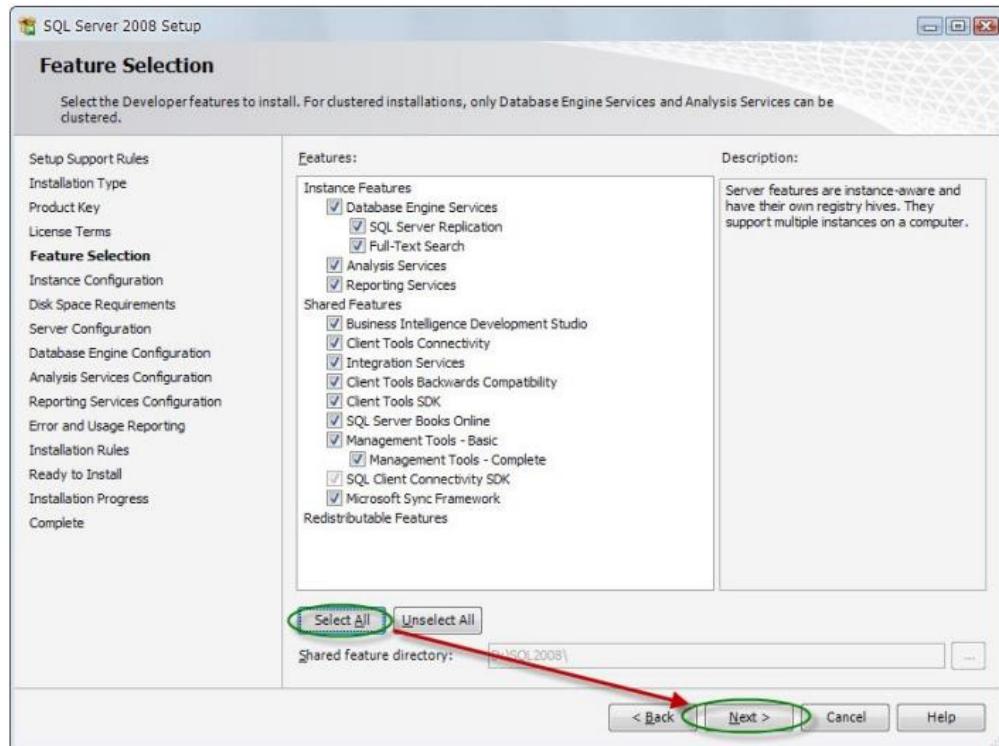
Hình 1.4: Giao diện Installation Type

Bước 4: Nhập product key



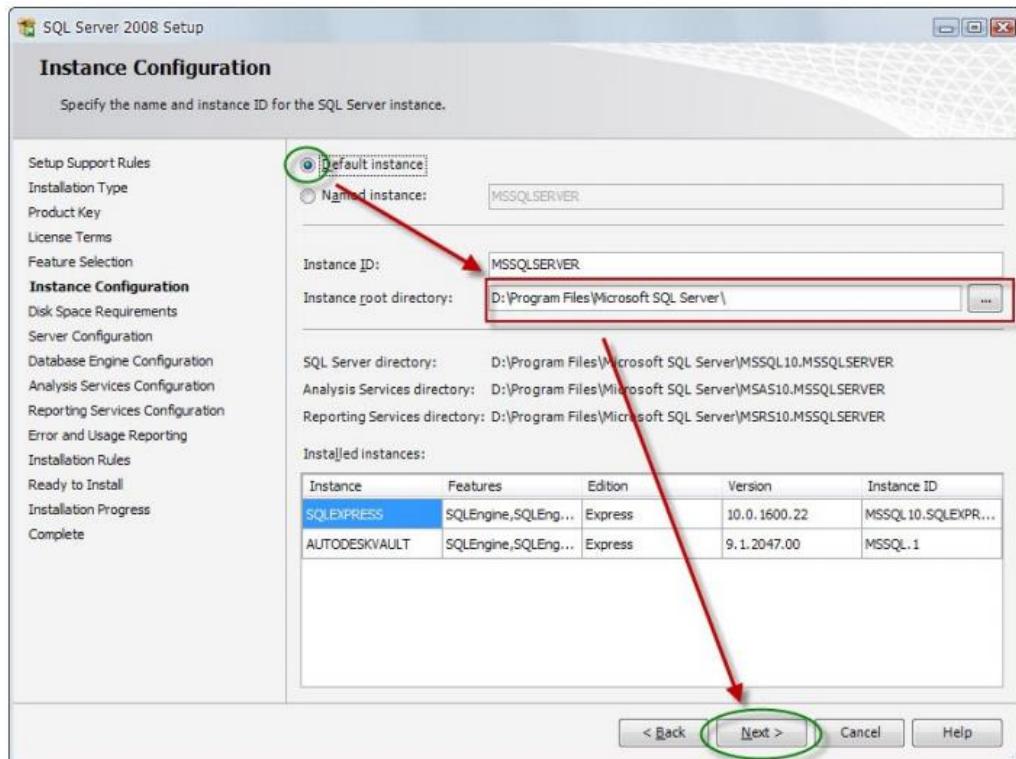
Hình 1.5: Giao diện Product Key

Bước 5: Sau khi đồng ý License Terms, chọn các thành phần cài đặt



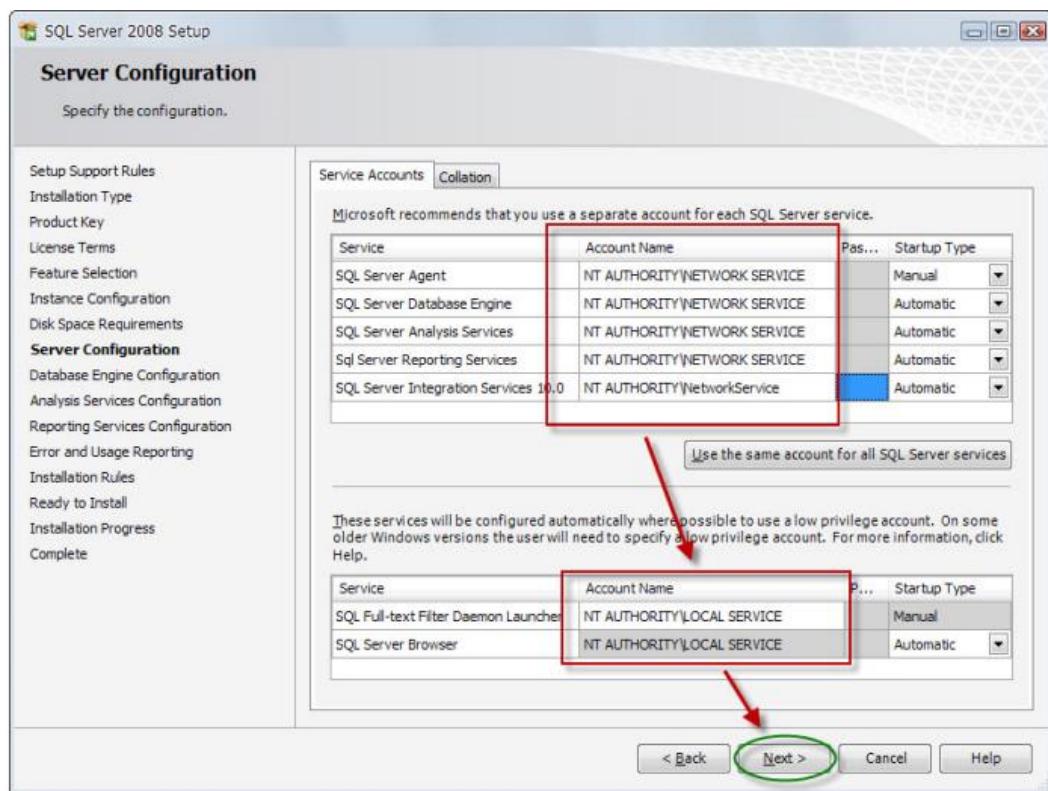
Hình 1.6: Giao diện Feature Selection

Bước 6: Thiết lập cài đặt chọn Default instance



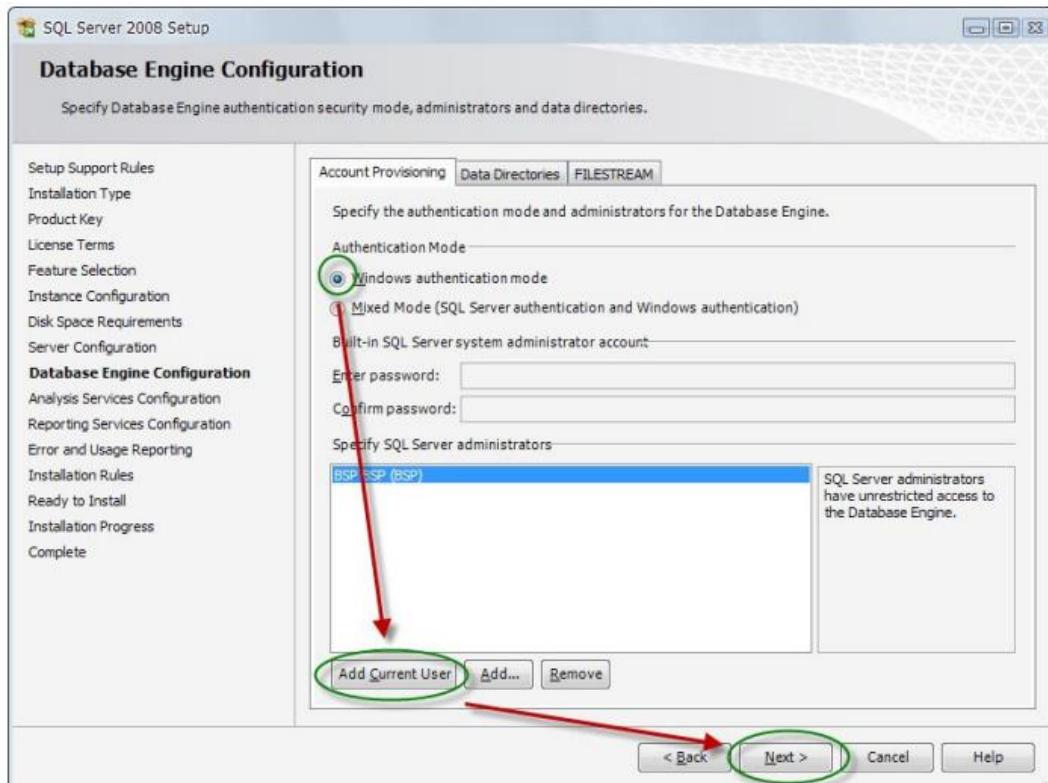
Hình 1.7: Giao diện Instance Configuration

Bước 7: Cấu hình server



Hình 1.8: Giao diện Server Configuration

Bước 8: Cấu hình dữ liệu như sau chọn Window Authentication và Add current User

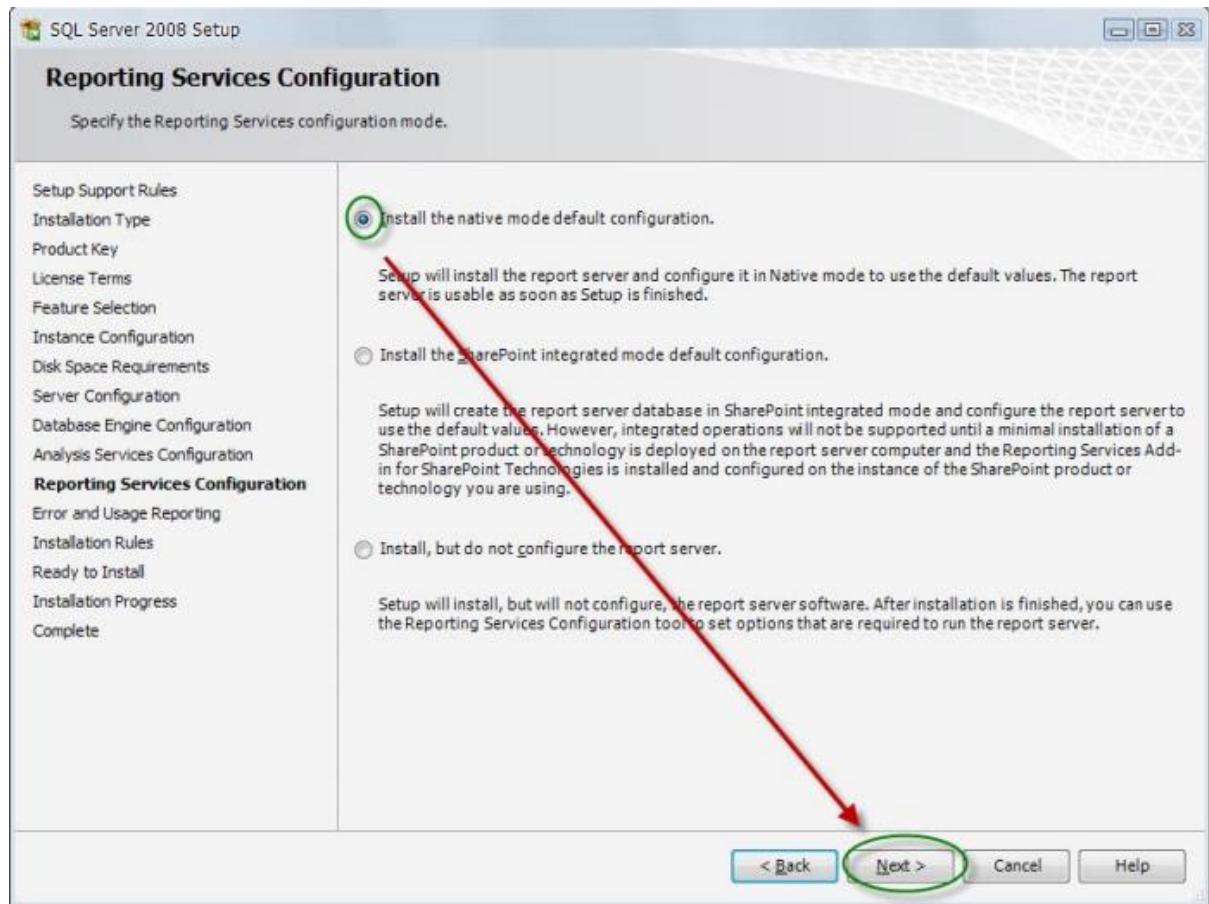


Hình 1.9: Giao diện Database Engine Configuration

Bước 9: Cấu hình analysis services Add Current User

Bước 10: Cấu hình report chọn option như hình nhấn Next, Next ...

Cho đến khi hoàn tất



Hình 1.10: Giao diện Reporting Services Configuration

1.2 Làm việc với SQL Server Management Studio

1.2.1 Mở SQL Server Management Studio:

Start → Program → Microsoft SQL Server 2008

→SQL Server Management Studio

Như Hình 1.11 (trang sau)

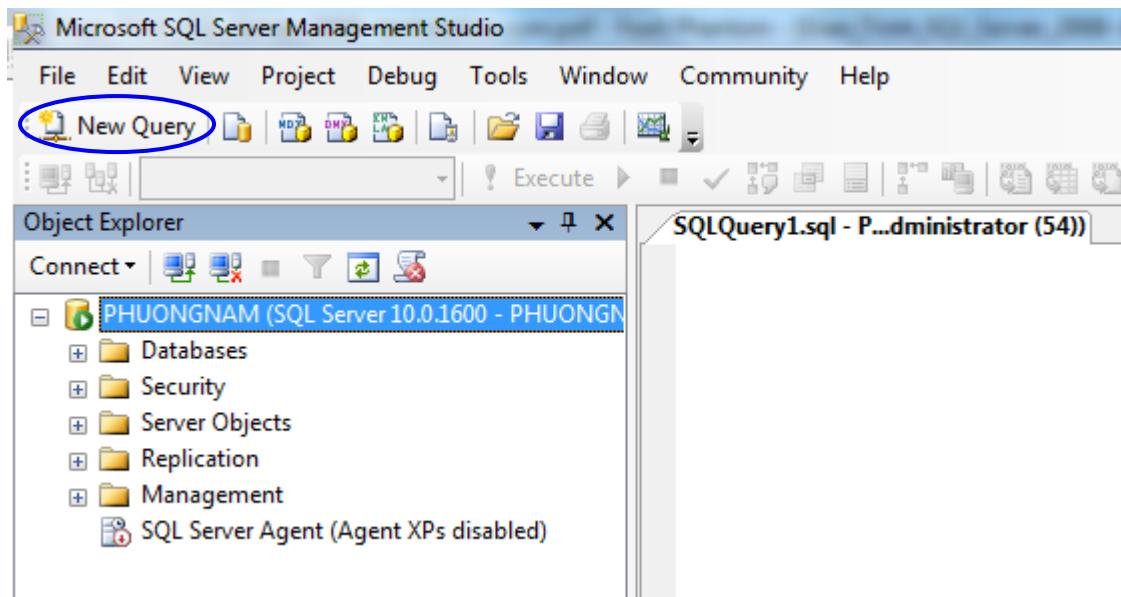


Hình 1.11: Kết nối vào SQL Server

Chú ý những thành phần trên hộp thoại sau:

- **Server Type:** Như ví dụ của cuốn sách này, cho phép server type là Database Engine. Các tùy chọn khác là kiểu dữ liệu khác nhau của servers nó sẽ hiển thị kết nối.
- **Server Name:** Hộp combo thứ 2 chứa 1 danh sách của SQL Server cài đặt mà bạn chọn. Trong hộp thoại hình 12, bạn sẽ thấy tên của máy tính được cài đặt trên local. Nếu bạn mở hộp Server name bạn có thể tìm kiếm nhiều server local hoặc network connection bằng cách chọn <Browse for more...>.
- **Authentication:** Combobox cuối cùng xác định các loại hình kết nối bạn muốn sử dụng. Trong giáo trình này chúng ta kết nối đến SQL Server sử dụng Windows Authentication. Nếu bạn cài đặt SQL Server với chế độ hỗn hợp(mix mode), thì bạn có thể thay đổi chọn lựa SQL Server authentication, thì nó sẽ mở hai hộp thoại và cho phép nhập username và password.

Sau khi nhấn nút *Connect* sẽ xuất hiện màn hình sau:



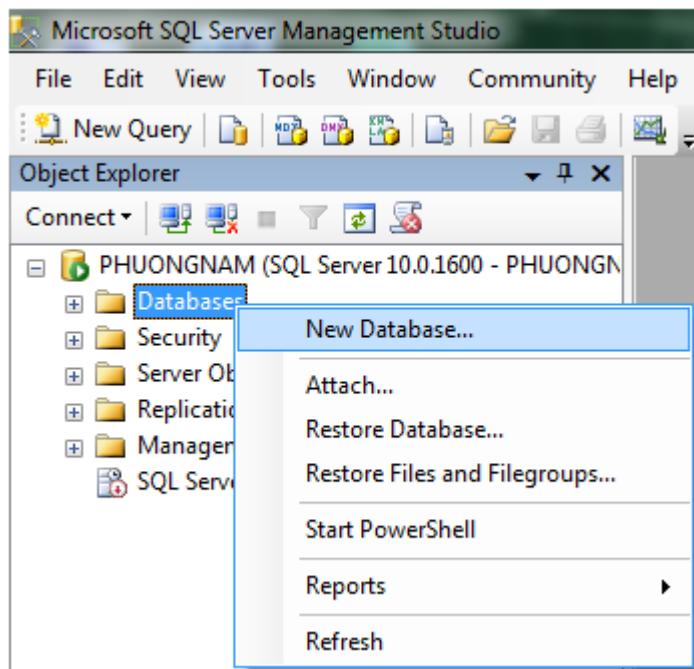
Hình 1.12: *SQL Server Management Studio*

Tại giao diện này ta chọn **New Query** để mở cửa sổ soạn thảo câu lệnh.

1.2.2 Tạo cơ sở dữ liệu (database)

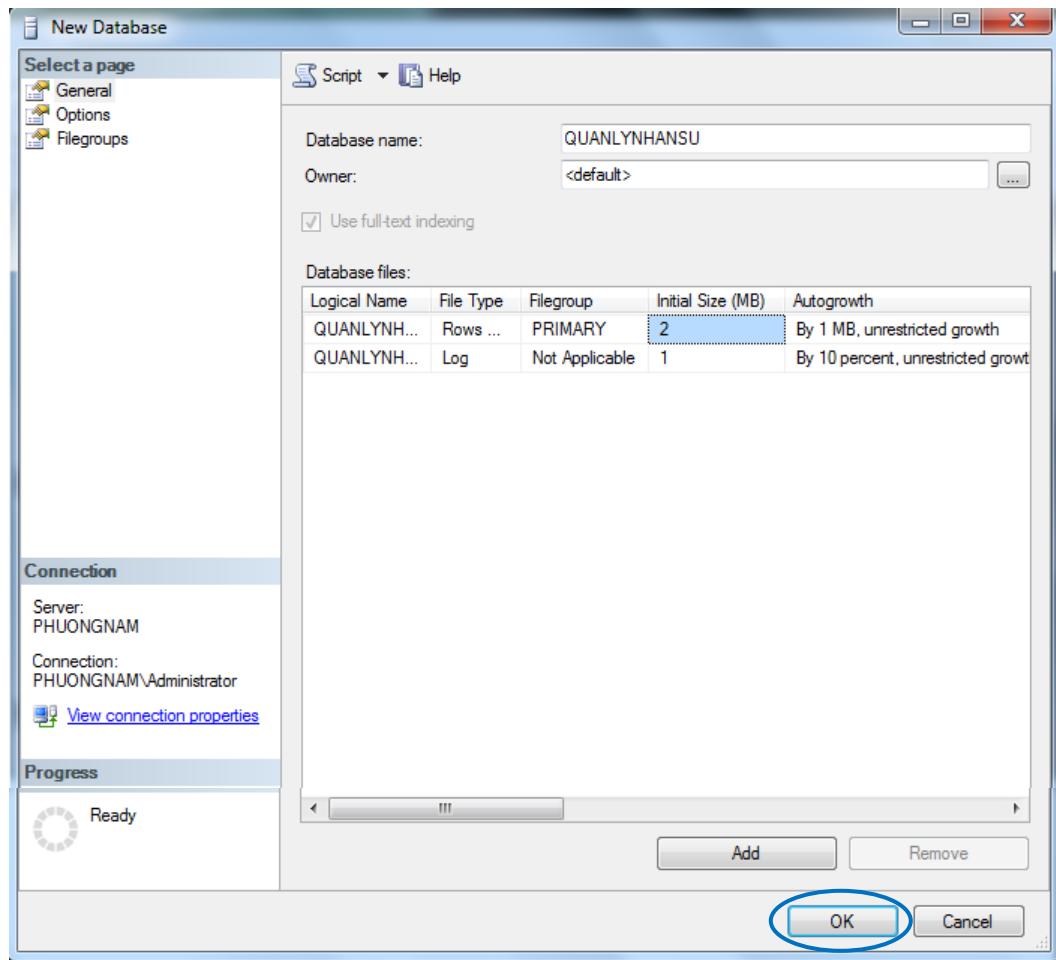
a. Thực hiện bằng công cụ

Chọn Database → Right Click → New Database...



Hình 1.13: *Hộp thoại Object Explorer*

Trong hộp thoại **New Database** đặt tên cho database name → **OK**



Hình 1.14: Giao diện New Database

b. Thực hiện bằng lệnh

```
CREATE DATABASE QUANLYNHANSU
```

Hình 1.15: Tạo cơ sở dữ liệu bằng lệnh

1.2.3 Tạo bảng (table)

Cú pháp lệnh tạo bảng như sau:

```
CREATE TABLE TableName  
  {  
    (columnName dataType [NOT NULL] [UNIQUE]  
     [DEFAULT defaultOption] [CHECK (searchCondition)] [, . . . ]}  
    [PRIMARY KEY (listOfColumns),]  
    {[UNIQUE (listOfColumns)] [, . . . ]}  
    {[FOREIGN KEY (listOfForeignKeyColumns)  
     REFERENCES ParentTableName [(listOfCandidateKeyColumns)]  
      [MATCH {PARTIAL | FULL}]  
      [ON UPDATE referentialAction]  
      [ON DELETE referentialAction]] [, . . . ]}  
    {[CHECK (searchCondition)] [, . . . ]})
```

Trong đó các từ khóa cơ bản được giải thích như sau:

- **CREATE TABLE:** Tạo table có tên là *TableName* gồm một hoặc nhiều cột (*columnName*) với các kiểu dữ liệu (*dataType*).
- **NOT NULL | UNIQUE:** là ràng buộc cho phép null/không cho phép null hoặc là duy nhất.
- **DEFAULT:** Hướng dẫn cung cấp các giá trị mặc định cho các cột riêng biệt. SQL sử dụng giá trị mặc định này khi câu lệnh INSERT bị lỗi.
- **PRIMARY KEY:** Chỉ định một cột hoặc một số cột làm khóa chính cho bảng. Mặc định là NOT NULL được quy định cho cột nào được chọn làm khóa chính. Dữ liệu cho cột / các cột làm khóa chính là duy nhất.
- **FOREIGN KEY:** Chỉ định một hoặc các thuộc tính nào đó là khóa ngoài và liên kết đến các bảng chứa khóa chính (bảng cha)
- **REFERENCES:** Tham chiếu đến bảng cha.

Ví dụ: Lệnh tạo bảng NHANVIEN như sau

```
CREATE TABLE NHANVIEN(  
  manv char(5) PRIMARY KEY not null,  
  ho nvarchar(20),  
  tenlot nvarchar(20),  
  ten nvarchar(20),  
  gioitinh char(3),  
  ngaysinh datetime,  
  hesoluong dec(4,2),  
  maphong char(3),  
  tel char(10),  
  ngaybc datetime,  
  FOREIGN KEY(maphong) REFERENCES PHONG(maphong))
```

1.2.4 Sửa đổi cấu trúc bảng

Lệnh ALTER TABLE được dùng để thêm, sửa, hoặc xóa cột trong bảng

Cú pháp:

```
ALTER TABLE table_name  
ADD COLUMN column_name column-definition;
```

ADD MULTIPLE COLUMNS IN TABLE: *thêm nhiều cột trong bảng*

Cú pháp:

```
ALTER TABLE table_name  
ADD column_1 column-definition,  
      column_2 column-definition,  
      ...  
      column_n column_definition;
```

MODIFY COLUMN IN TABLE: *Định nghĩa lại cột*

Cú pháp:

```
ALTER TABLE table_name  
ALTER COLUMN column_name column_type;
```

DROP COLUMN IN TABLE: *Xóa cột trong bảng*

Cú pháp:

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

RENAME COLUMN IN TABLE: *Đổi tên cột trong bảng*

Cú pháp:

```
sp_rename 'table_name'.old_column_name', 'new_column_name', 'COLUMN';
```

RENAME TABLE: *Đổi tên bảng*

Cú pháp:

```
sp_rename 'old_table_name', 'new_table_name';
```

1.2.5 Xóa bảng

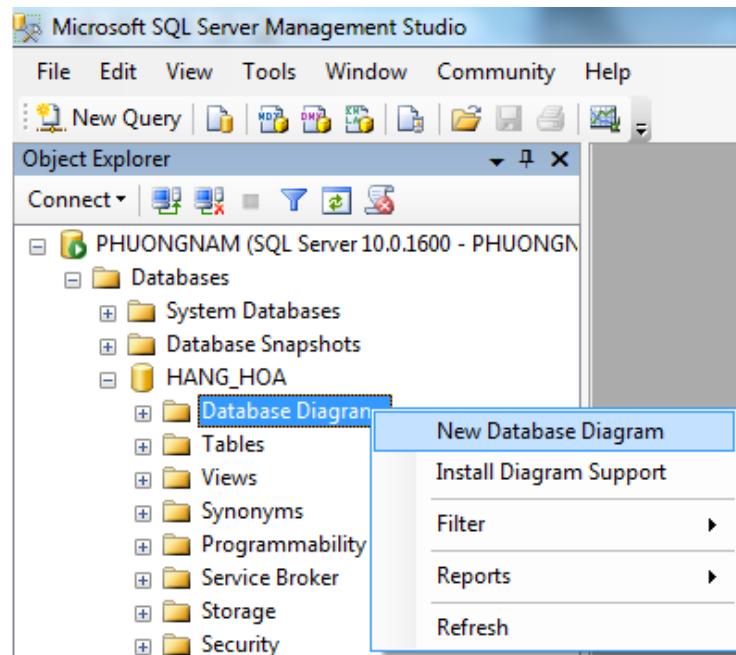
Xóa bảng khỏi cơ sở dữ liệu bằng cú pháp sau

```
DROP TABLE table_name;
```

1.2.6 Tạo lưu đồ (diagram)

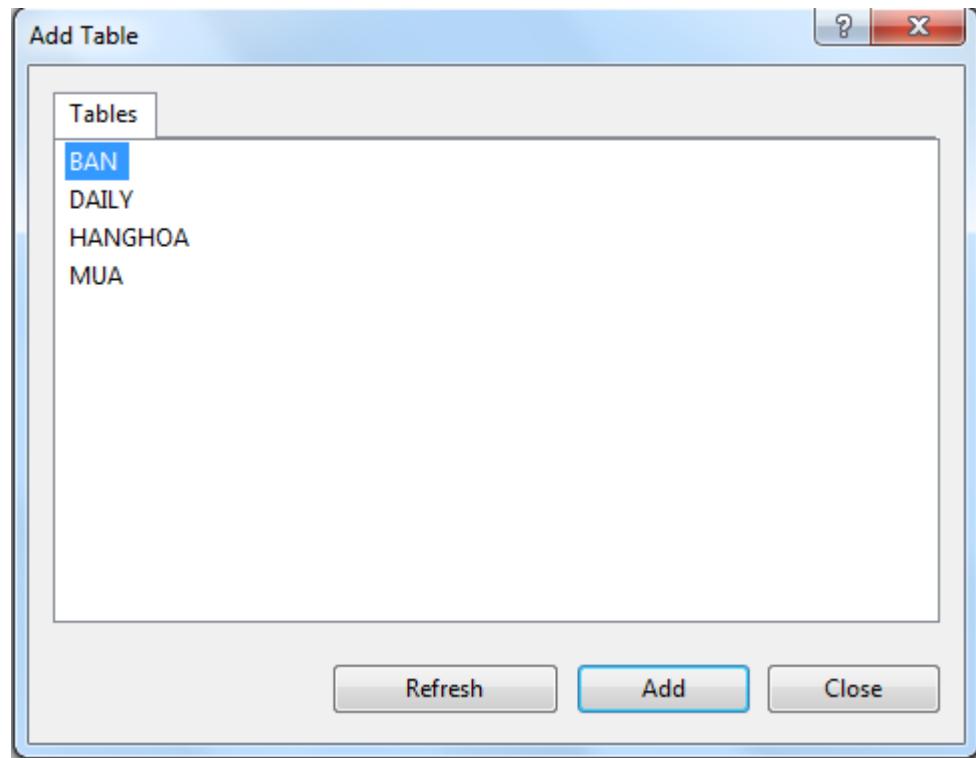
Diagrams là một cửa sổ hiển thị mối quan hệ giữa các bảng của một cơ sở dữ liệu. Tạo diagram ta thực hiện như sau:

Chọn tên database cần tạo → Right Click → Database Diagrams → New Database Diagram

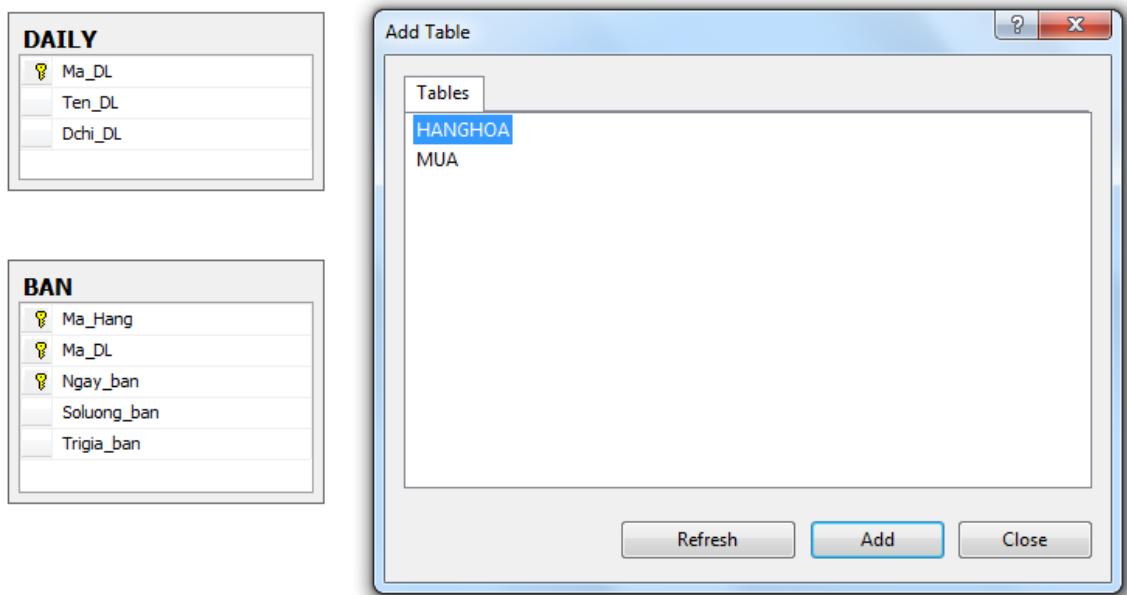


Hình 1.16: Tạo Diagram

Sau khi chọn New Database Diagram sẽ xuất hiện hộp thoại như *Hình 1.17* để thêm các bảng (nhấn nút Add), sau khi thêm xong chọn nút Close.

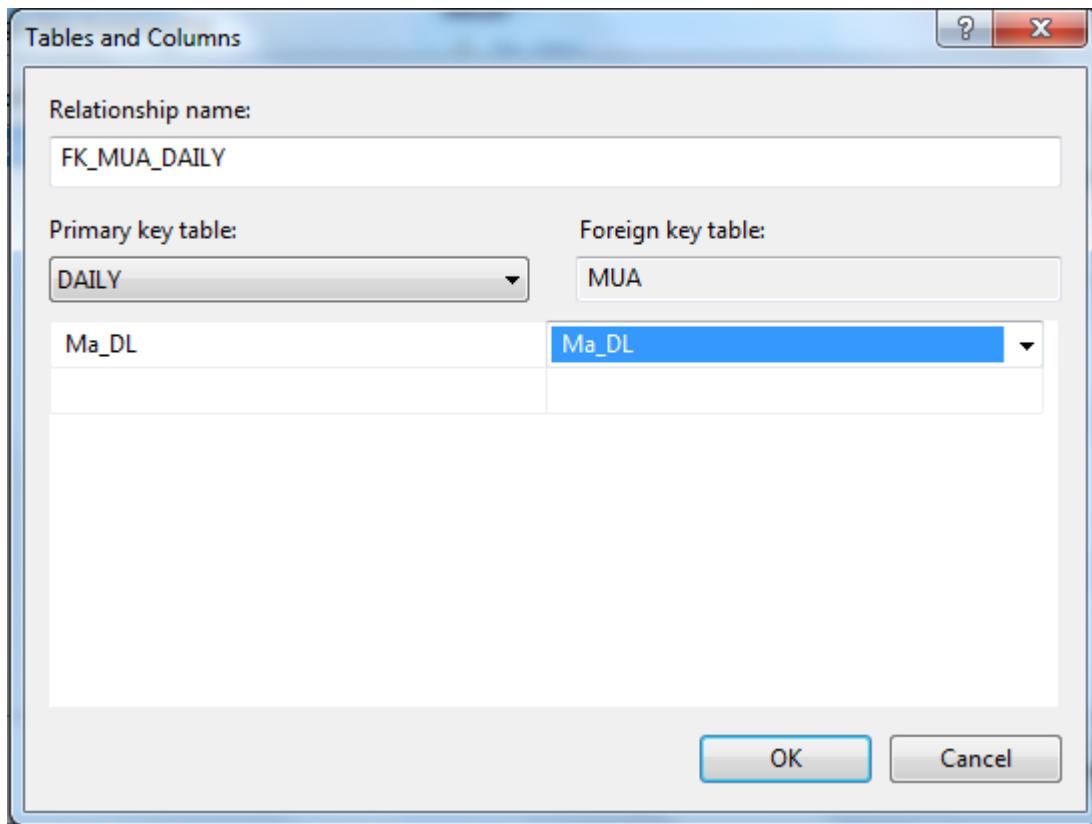


Hình 1.17a: Thêm các bảng vào diagram



Hình 1.17b: Thêm các bảng vào diagram

Sau khi thêm các bảng xong, để thiết lập mối quan hệ giữa các bảng ta chọn cột dữ liệu của cột làm khóa chính trong bảng cha (parent table) và kéo nó đến khóa ngoại trong bảng con (child table), sau đó xuất hiện hộp thoại như *Hình 1.18* để thiết lập mối liên kết giữa hai bảng



Hình 1.18: Thiết lập mối liên kết giữa hai bảng

Tương tự như trên, tiếp tục thiết lập hết mối liên kết giữa tất cả các bảng cho cơ sở dữ liệu, sau đó lưu lại các thiết lập cho diagram vừa tạo. Nếu các thiết lập cho diagram không thực hiện được thì khi lưu hệ thống sẽ báo lỗi.

1.2.7 Thao tác trên dữ liệu

1.2.7.1 Thêm dữ liệu

Cú pháp 1

```
INSERT INTO table
(column1, column2, ... )
VALUES
(expression1, expression2, ... ),
(expression1, expression2, ... ),
...;
```

Cú pháp 2

```
INSERT INTO table  
(column1, column2, ... )  
VALUES  
( DEFAULT | NULL | expression1,  
  DEFAULT | NULL | expression2,  
  ...  
);
```

Cú pháp 3

```
INSERT INTO table  
(column1, column2, ... )  
DEFAULT VALUES;
```

Cú pháp 4

```
INSERT INTO table  
(column1, column2, ... )  
SELECT expression1, expression2, ...  
FROM source_table  
WHERE conditions;
```

1.2.7.2 Xóa dữ liệu

```
DELETE [ TOP (top_value) [ PERCENT ] ]  
FROM table  
WHERE conditions;
```

1.2.7.3 Cập nhật dữ liệu

Cú pháp 1:

```
UPDATE table  
SET column1 = expression1,  
    column2 = expression2,  
    ...  
WHERE conditions;
```

Cú pháp 2:

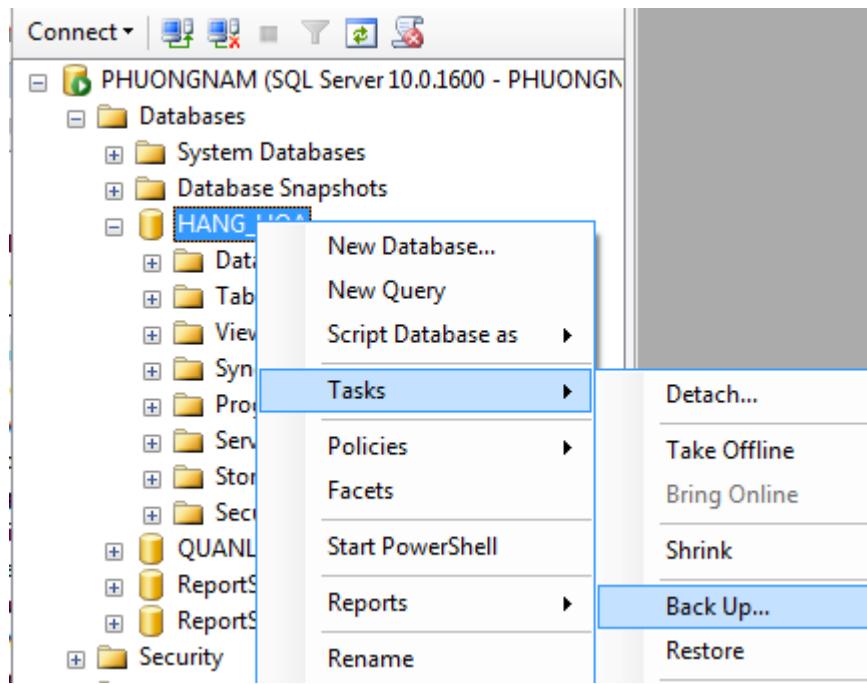
```
UPDATE table1  
SET column1 = (SELECT expression1  
                FROM table2  
                WHERE conditions)  
WHERE conditions;
```

1.2.8 Sao lưu dữ liệu (Back Up)

Thao tác Back Up Database giúp ta lưu lại toàn bộ cấu trúc và dữ liệu hiện có của CSDL. Các bước sao lưu dữ liệu được thực hiện bởi một chuỗi các thao tác như sau:

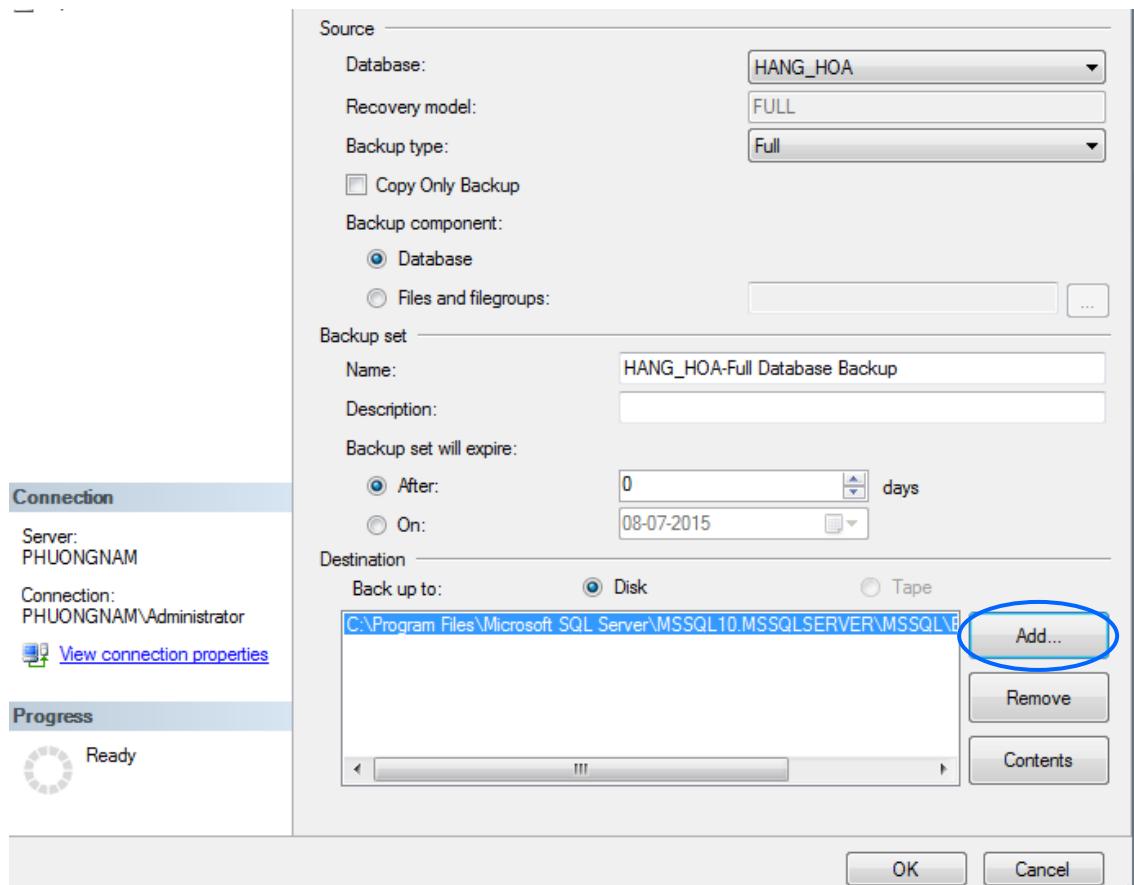
Right Click vào database cần ack up → Tasks → Back up...

như *Hình 1.19* bên dưới (Trang sau)



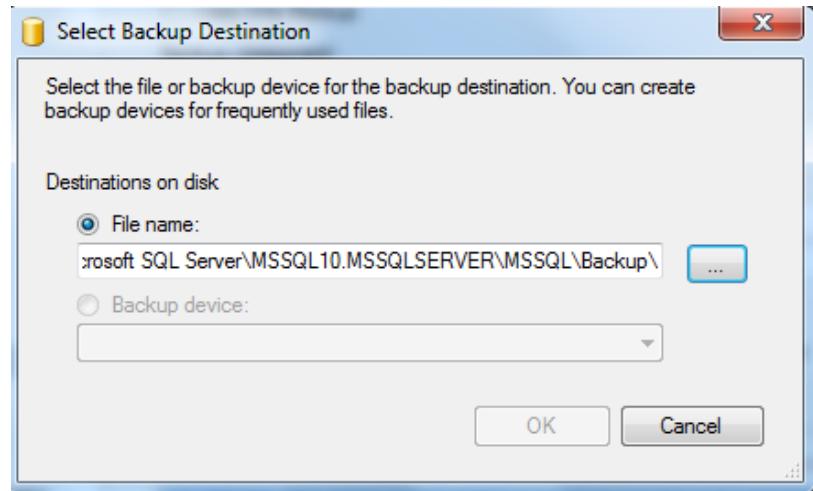
Hình 1.19a: Sao lưu dữ liệu

Sau khi chọn xong thao tác ở *Hình 1.19a* ta được hộp thoại như *Hình 1.19b*



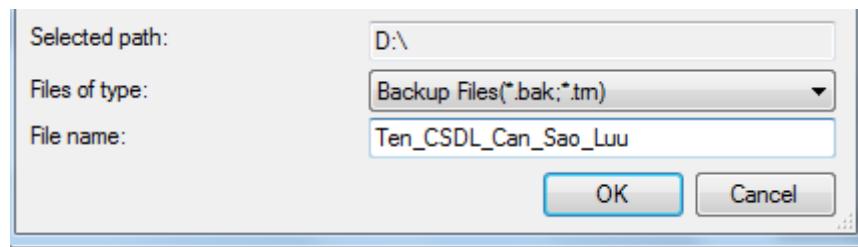
Hình 1.19b: Chọn toàn bộ database

Tại mục Destination nhấp nút Add để thêm đường dẫn đến tập tin sao lưu ta được hộp thoại như *Hình 1.19c* như sau:



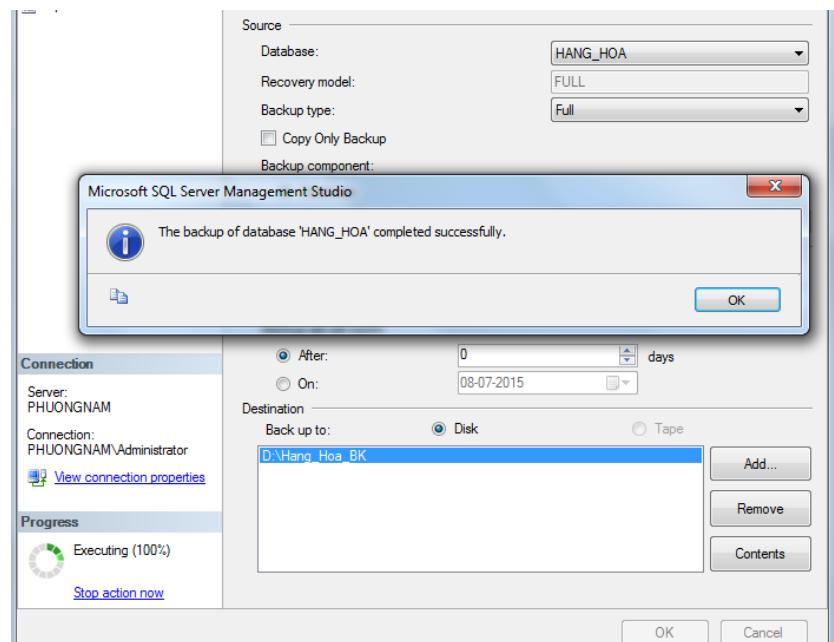
Hình 1.19c: Thêm đường dẫn đến tập tin sao lưu dữ liệu

Sau đó chọn nơi chứa tập tin sao lưu và đặt tên như *Hình 1.19d*



Hình 1.19d: Chọn đường dẫn và đặt tên tập tin cần sao lưu

Cuối cùng chọn Ok để hoàn tất quá trình sao lưu dữ liệu.



Hình 1.19e: Back Up dữ liệu thành công

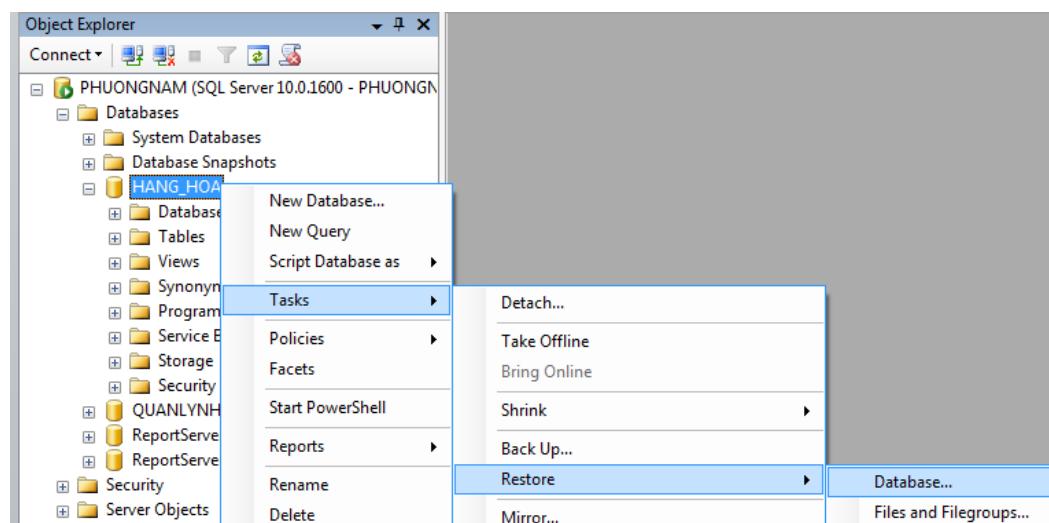
1.2.9 Phục hồi dữ liệu (Restore Database)

Với tập tin sao lưu đã thực hiện ở phần trên, khi ta muốn phục hồi lại toàn bộ cấu trúc và dữ liệu thì sử dụng chức năng Restore Database.

Quá trình phục hồi dữ liệu gồm một chuỗi các thao tác như sau:

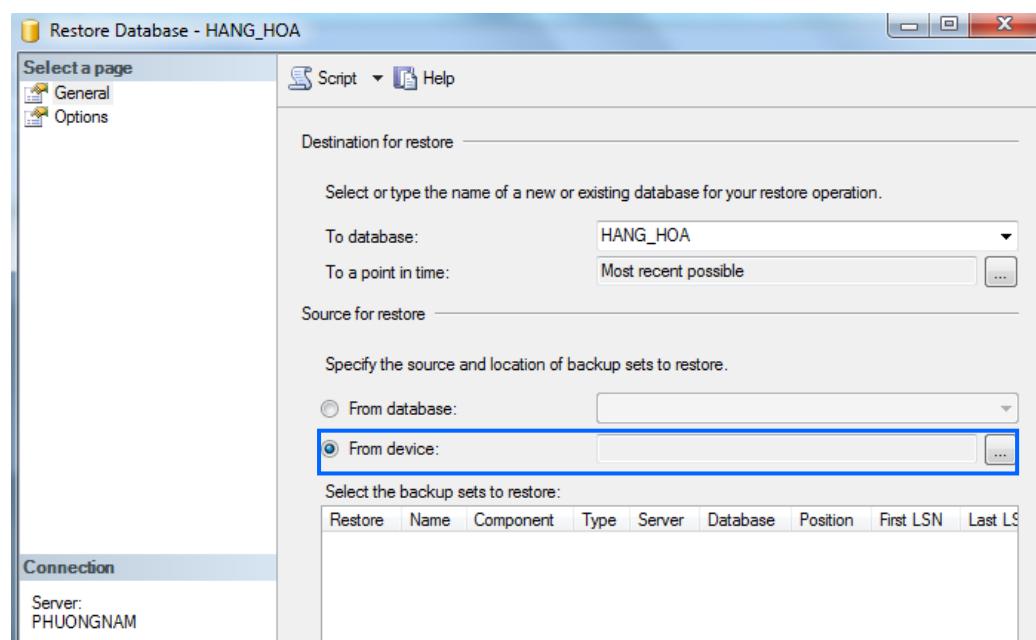
Right Click vào database cần Restore → Tasks → Restore → Database...

Như Hình 1.20a

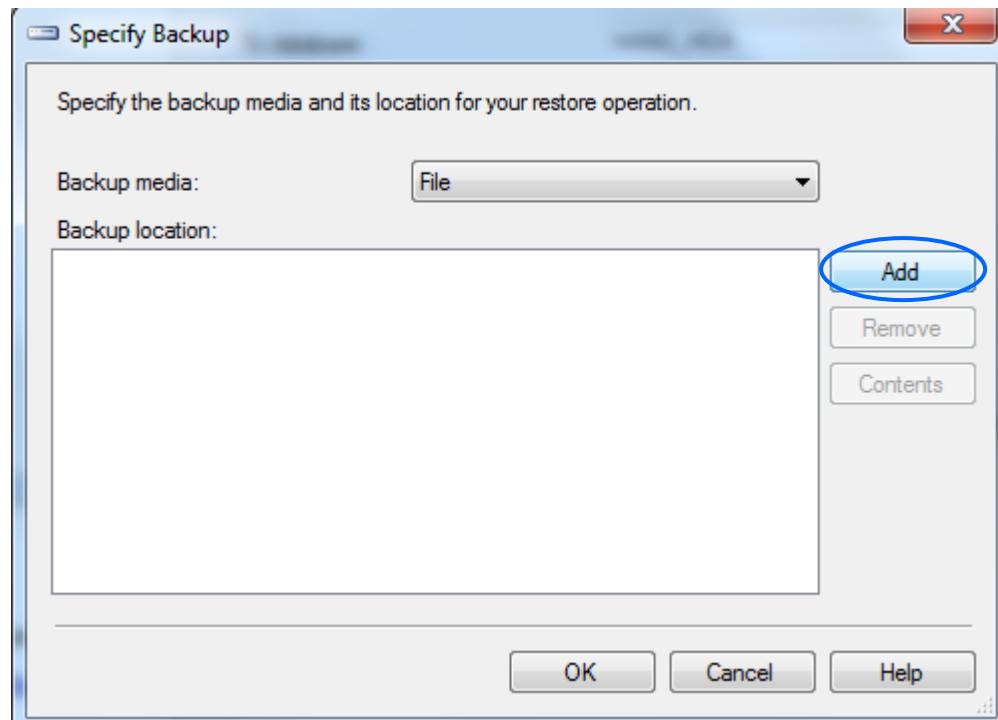


Hình 1.20a: Phục hồi dữ liệu

Chọn mục Database ta được cửa sổ như *Hình 1.20b*. Tại đây ta chọn cơ sở dữ liệu cần phục hồi tại mục **To database** và chọn nơi chứa tập tin sao lưu để phục hồi tại mục **From device** thì được hộp thoại tại *Hình 1.20c*

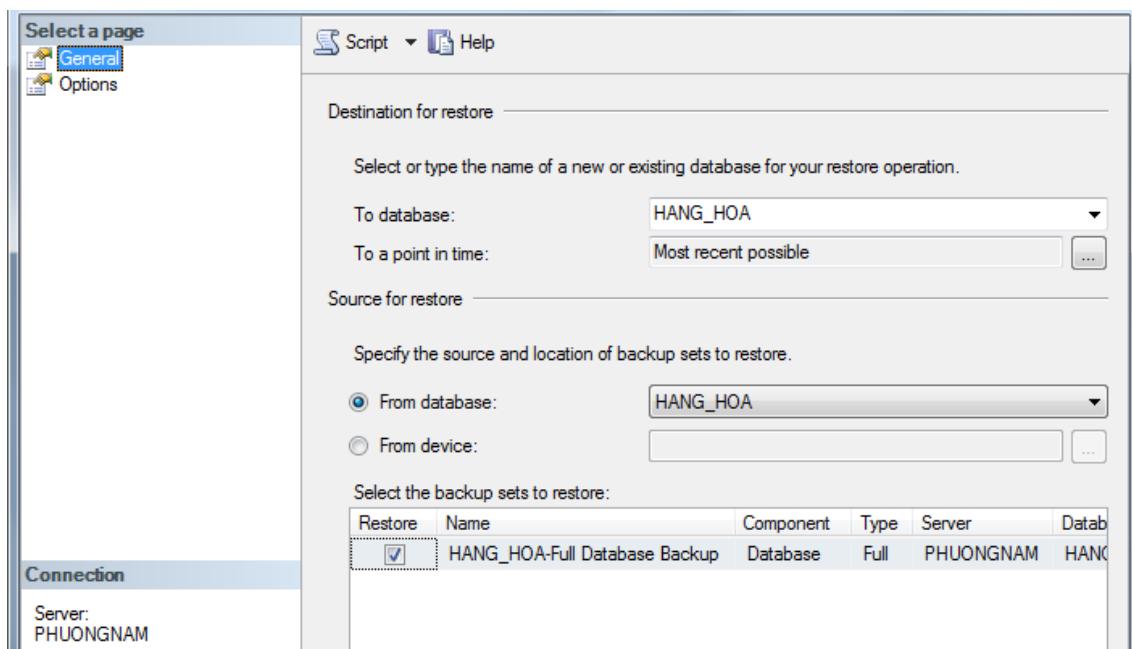


Hình 1.20b: Chọn CSDL cần phục hồi và nơi chứa dữ liệu nguồn để phục hồi

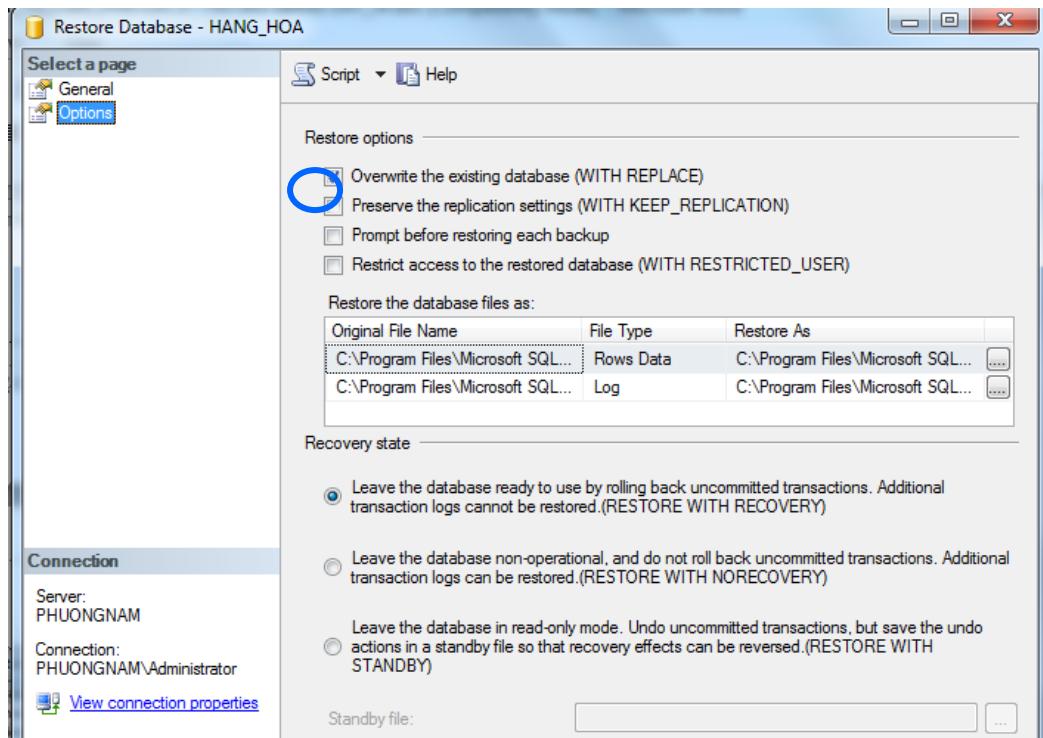


Hình 1.20c: Chọn nơi chứa tập tin nguồn để phục hồi

Tại đây ta click nút Add để chỉ ra đường dẫn nơi chứa dữ liệu nguồn để phục hồi.

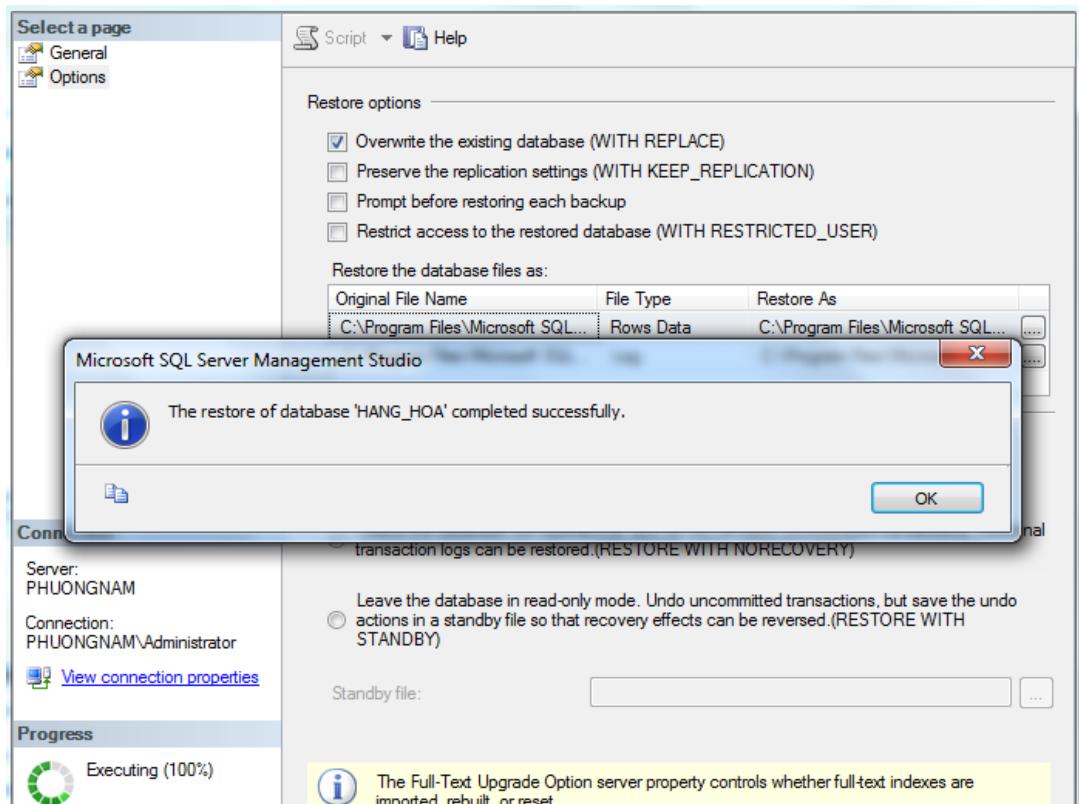


Hình 1.20d: Chọn phục hồi dữ liệu tại mục Restore



Hình 1.20e: Tùy chọn cho phục hồi dữ liệu

Cuối cùng chọn **Ok** để hoàn tất quá trình phục hồi dữ liệu như *Hình 1.20f*



Hình 1.20f: Khôi phục thành công

PHỤ LỤC 2: VIEW

2.1 Giới thiệu về View

View là một bảng ảo, nội dung được định nghĩa bởi một truy vấn (câu Select). Giống như một bảng thực, một View bao gồm một tập các cột và các dòng dữ liệu. Tuy nhiên, một View không là nơi lưu trữ dữ liệu. Các dòng và cột của dữ liệu được tham chiếu từ các bảng trong một truy vấn mà định nghĩa View và là kết quả động khi View được tham chiếu.

Dùng View để:

- Chỉ cho người dùng xem những gì cần cho xem.
- Đơn giản hóa việc truy cập dữ liệu.
- Dùng để lựa chọn những dữ liệu cần thiết ứng với mỗi user.
- Dùng View để nhập (Import), xuất (Export) dữ liệu.
- Kết hợp các dữ liệu khác nhau.

Hạn chế khi định nghĩa View:

- Không bao gồm các mệnh đề COMPUTER hoặc COMPUTER BY.
- Không bao gồm từ khóa INTO.
- Chỉ được dùng ORDER BY chỉ khi từ khóa TOP được dùng.
- Không thể tham chiếu quá 1024 cột.
- Không thể kết hợp với câu lệnh T-SQL khác trong một cùng một bó lệnh.
- Không thể định nghĩa chỉ mục full text trên View.

Partitioned Views: Một partition View kết nối theo chiều dọc các dữ liệu phân tán từ một tập các bảng ở một hay nhiều server, các dữ liệu sẽ hiện lên như thể là chúng được lấy từ một bảng. Có hai loại: Local partition view là view có tham chiếu các table và các view khác nằm trong cùng một sever. Distributed partition view có ít nhất một bảng nằm ở sever khác.

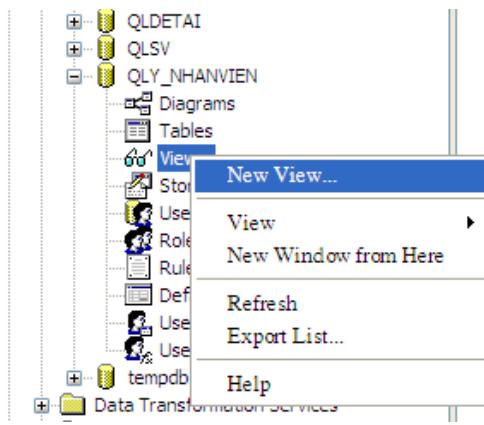
2.2 Tạo, hiệu chỉnh, xóa View bằng công cụ

2.2.1 Tạo bảng View bằng Enterprise Manager:

Các bước tạo View như sau:

Chọn chức năng Views của cơ sở dữ liệu. Right click trên đối tượng **Views** → **New View ...** như *Hình 2.1*

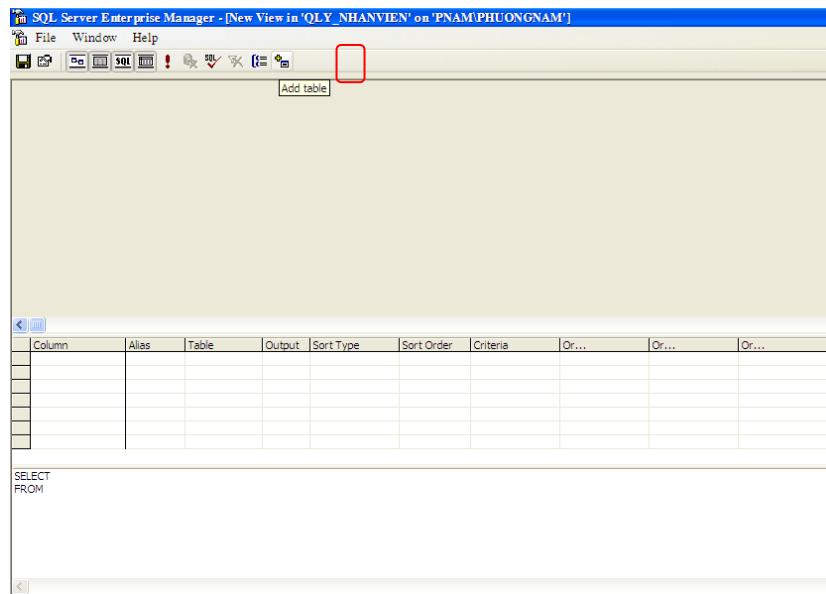
Bước 1:



Hình 2.1: Tạo View trong Enterprise Manager

Bước 2: Thêm các bảng tham gia câu lệnh truy vấn dữ liệu cho View

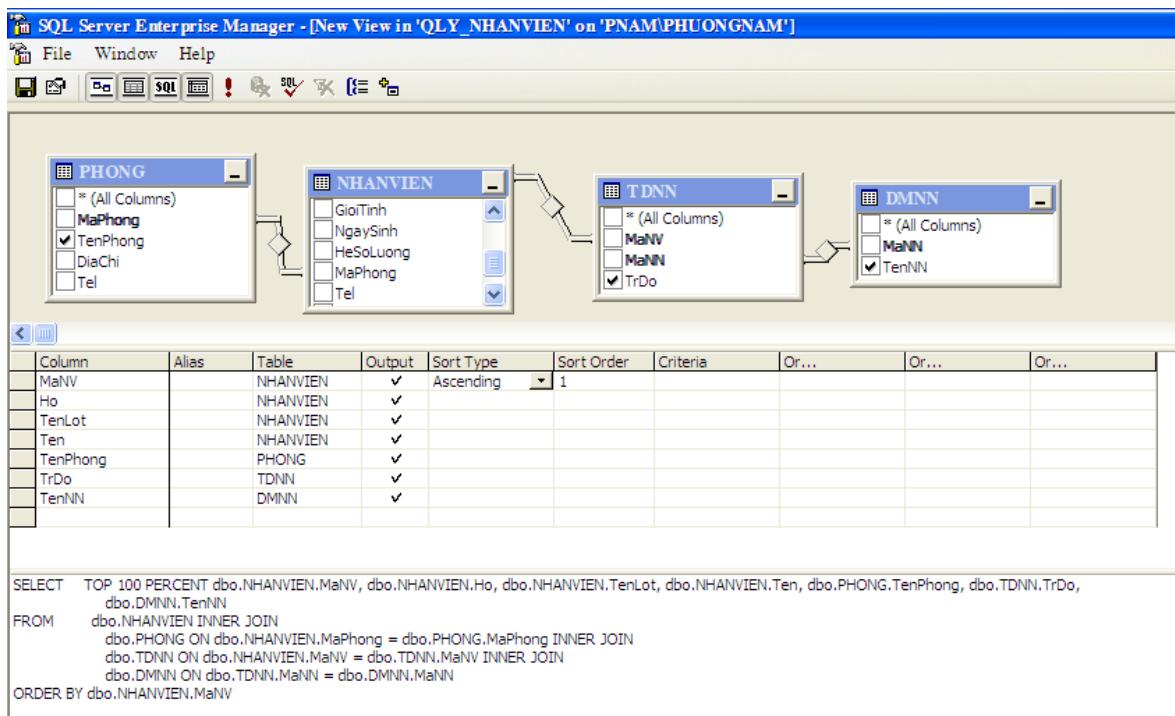
Trong màn hình thiết kế dữ liệu View như *Hình 2.2*, nhấp vào biểu tượng **Add Table** trên thanh công cụ để đưa các bảng dữ liệu làm dữ liệu nguồn cho View.



Hình 2.2: Thêm dữ liệu cho View

Bước 3:

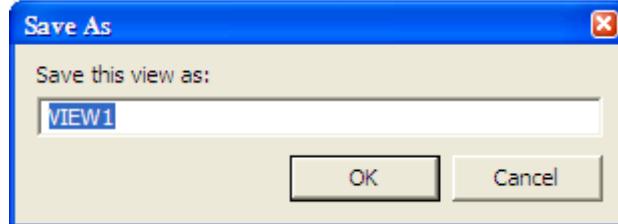
Sau khi chọn các bảng dữ liệu cần thiết, kéo thả cột dữ liệu hoặc click chuột vào cột dữ liệu cần kết xuất, câu truy vấn sẽ tự động được phát sinh như *Hình 2.3*.



Hình 2.3: Thiết kế dữ liệu View

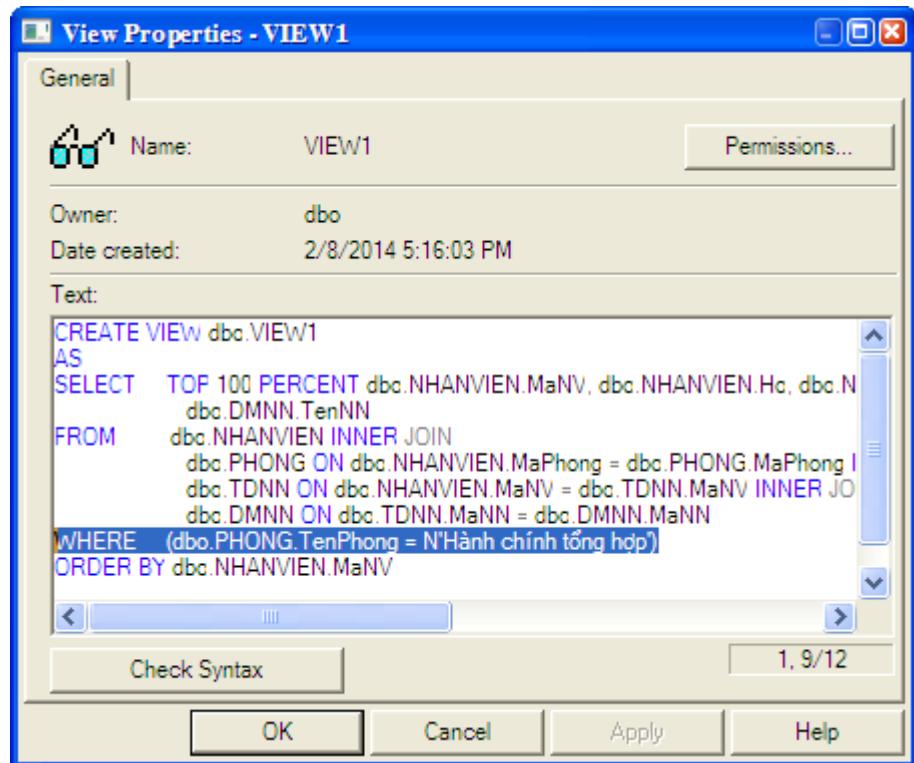
Bước 4:

Nhấn vào biểu tượng **Save** trên thanh công cụ và gõ vào tên của View, tên mặc nhiên là View1,2,3..., nhấn **OK** để kết thúc quá trình tạo View bằng công cụ.



Hình 2.4: Lưu lại View

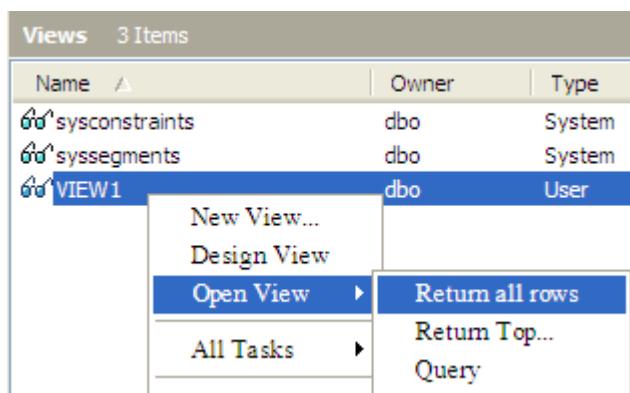
Sau khi tạo xong View, bạn cũng có thể quay lại để sửa đổi nội dung câu lệnh **SELECT** trong View bằng cách chọn chức năng **Design View** để quay lại màn hình thiết kế dữ liệu View trước đó hoặc chọn chức năng **Properties** để có thể sửa trực tiếp câu lệnh **SELECT** bên trong View. Các chức năng này hiển thị trong thực đơn tắt sau khi nhấn chuột phải trên tên của View cần sửa đổi như *Hình 2.5*.



Hình 2.5: Hiển thị câu lệnh SELECT trong View

2.2.2 Xem và cập nhật dữ liệu View:

Sau khi tạo xong View, bạn có thể xem dữ liệu mà View chưa đựng có đúng theo mong muốn hay không bằng cách right click trên tên của View cần xem dữ liệu, chọn **Open View → Return all rows** như *Hình 2.6*.



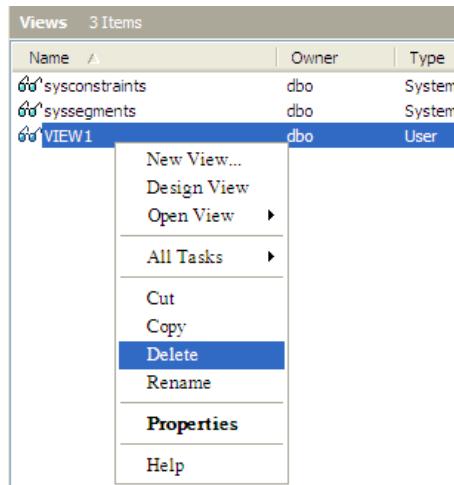
Hình 2.6: Xem dữ liệu View

Để xem nội dung dữ liệu của **VIEW1** vừa tạo ở trên bằng lệnh sau:

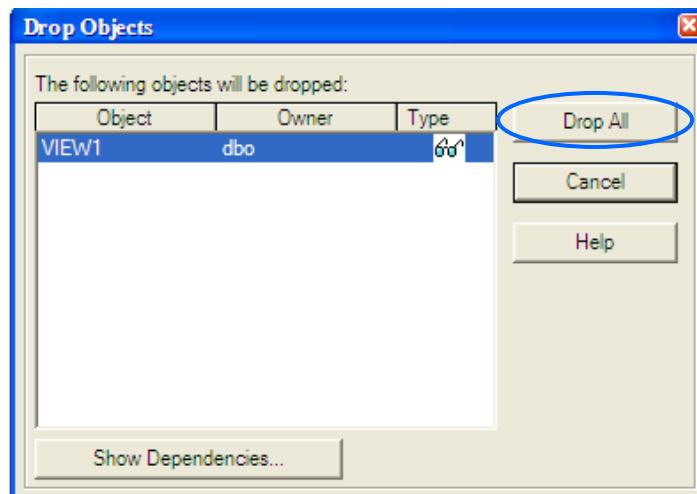
```
SELECT * FROM view1
```

2.2.3 Hủy bỏ View

Giống như các đối tượng khác, bạn có cũng được phép hủy bỏ các View sau khi tạo ra chúng nếu không còn tiếp tục sử dụng nữa. Các bảng được tham chiếu trong câu lệnh **SELECT** của View sẽ không bị hủy bỏ. Để hủy bỏ View, thao tác như *Hình 2.7*, sau đó chọn nút **Drop All** (*Hình 2.8*) để đồng ý hủy bỏ .



Hình 2.7: Hủy bỏ View



Hình 2.8: Hộp thoại xác nhận đồng ý hủy bỏ View

2.3 Tạo, hiệu chỉnh, xóa View bằng lệnh

Cú pháp

```
CREATE VIEW [ <database_name> . ] [ <owner> . ] view_name [  
    (column [,...n]) ]  
    [ WITH <view_attribute> [,...n] ]  
    AS  
        select_statement  
    [ WITH CHECK OPTION ]  
    <view_attribute> ::=  
        { ENCRYPTION | SCHEMABINDING }
```

Giải thích

+ **view_name:** là tên của View. Tên View phải tuân thủ các qui tắc định danh.

+ **Column:** Tên được dùng cho cột trong view. Tên cột chỉ dùng trong trường hợp cột được phát sinh từ một biểu thức, hàm, một bảng, các cột trong các table trùng tên. Tuy nhiên tên cột cũng có thể được áp định trong câu lệnh Select. Không chỉ định tên cột chính là tên các cột trong câu lệnh select

+ **select_statement:** Là câu lệnh select để định nghĩa View. Nó có thể tham chiếu một hoặc nhiều bảng hoặc các View khác với một câu Select phức tạp.

Lưu ý: Để tạo một view, bạn phải có quyền dành riêng trên các bảng hoặc view tham chiếu trong định nghĩa view.

WITH CHECK OPTION: Bắt buộc tất cả các câu lệnh hiệu chỉnh dữ liệu thực thi dựa vào View phải tuyệt đối tôn trọng triết lý đến tập tiêu chuẩn trong câu lệnh Select. Nếu bạn dùng từ khóa này, các dòng không thể được hiệu chỉnh trong cách mà tại sao chúng hiện trong view. Bất kỳ hiệu chỉnh nào mà sẽ gây ra tình trạng thay đổi đều bị hủy bỏ, và một lỗi được hiện ra.

WITH ENCRYPTION: Mã hóa câu lệnh Select tạo ra view

SCHEMABINDING: Kết View với một giản đồ. Khi SCHEMABINDING được chỉ định, câu lệnh Select phải chỉ rõ chủ quyền của các bảng, các view. Các hàm được tham chiếu View hay bảng tham gia trong view được tạo với schema không thể xóa trừ phi view đó bị xóa hoặc thay đổi cơ chế này. Câu lệnh Alter Table trên bảng tham gia trong view cũng bị lỗi.

Ví dụ 1: Xem danh sách nhân viên

```
CREATE VIEW V_DanhsachNV  
AS  
SELECT MaNV, Ho, Tenlot, Ten  
From NhanVien  
Where MaNV Like 'HC%'
```

Xem nội dung View

```
Select *  
From V_DanhsachNV
```

Ví dụ 2:

```
CREATE VIEW V_DanhsachNV_phongban  
AS  
SELECT MaNV, Ho, Tenlot, Ten,Gioitinh, Tenphong  
From NhanVien, phong  
Where NhanVien.Maphong=Phong.Maphong  
and Phong.Tenphong=N'Kỹ Thuật'
```

Xem nội dung View

```
Select * From V_DanhsachNV_phongban
```

2.4 Tạo Partition view

- Các bảng tham gia trong Partition view phải có cấu trúc giống nhau.
- Có một cột có check constraint, với phạm vi của Check constraint ở mỗi bảng là khác nhau.
- Tạo view bằng cách kết các dữ liệu bằng từ khóa UNION ALL

Ví dụ: Ta có 3 table tương ứng dùng để lưu trữ các khách hàng ở 3 miền Bắc, Trung, Nam có cấu trúc tạo như sau:

```
Create Table KH_BAC  
(Makh int primary key,  
TenKh Nchar(30),  
Khuvuc Nvarchar(30) CHECK (Khuvuc='Bac bo')  
)
```

```

Create Table KH_TRUNG
    (Makh int primary key,
     TenKh Nchar(30),
     Khuvuc Nvarchar(30) CHECK (Khuvuc='Trung bo')
    )

Create Table KH_NAM
    (Makh int primary key,
     TenKh Nchar(30),
     Khuvuc Nvarchar(30) CHECK (Khuvuc='Nam bo')
    )

```

Tạo một partition View gộp 3 bảng trên lại với nhau:

```

Create View Khachhang
AS
    Select * From KH_BAC
    UNION ALL
    Select * From KH_TRUNG
    UNION ALL
    Select * From KH_NAM

```

2.5 Truy xuất dữ liệu thông qua View

Tương tự như truy xuất dữ liệu trên bảng

2.5.1 Xem dữ liệu thông qua view

Dùng câu lệnh Select (Ví dụ ở trên)

2.5.2 Hiệu chỉnh dữ liệu thông qua View

Thao tác hiệu chỉnh dữ liệu giống thao tác hiệu chỉnh dữ liệu trên một bảng. Tuy nhiên, View phải thỏa mãn điều kiện sau:

- View chỉ tham chiếu duy nhất một bảng.
- Thao tác Delete không bao giờ được phép thực hiện trên nhiều bảng trong View.
- Các hàm kết hợp Group By, Union, Distinct, Top không dùng trong danh sách chọn trong câu Select của View.
- Không có các cột tính toán.

Hiệu chỉnh dữ liệu thông qua partitioned View

Khi dùng lệnh Insert và update phải tôn trọng các qui tắc sau:

- Tất cả các cột phải có giá trị ngay cả cột chấp nhận **NULL** và cột có giá trị default.
- Từ khóa Default không được sử dụng trong câu Insert, update.
- Phải có giá trị đúng của cột có check constraint.
- Câu lệnh insert không cho phép nếu bảng thành viên có cột có thuộc tính identity, cột timestamp.
- Không insert hoặc Update nếu có một kết self-join trong cùng view hay bảng thành viên.

Khi dùng lệnh delete, ta có thể xóa các mẫu tin trong bảng thành viên thông qua view. Lệnh Delete không thực thi nếu có liên kết Self-join

PHỤ LỤC 3:

THỦ TỤC LUU TRỮ (STORED PROCEDURE)

3.1 Giới thiệu Stored Procedure

Khi chúng ta tạo một ứng dụng với MS SQL Server, ngôn ngữ lập trình T-SQL (Transact-SQL) là ngôn ngữ chính giao tiếp giữa ứng dụng và Database của SQL Server. Khi chúng ta tạo các chương trình bằng T-SQL, hai phương pháp chính có thể dùng để lưu trữ và thực thi cho các chương trình là:

- Lưu trữ các chương trình cục bộ và tạo các ứng dụng để gọi các lệnh đến SQL Server và xử lý các kết quả.
- Lưu trữ những chương trình như các Stored Procedure trong SQL Server và tạo ứng dụng để gọi thực thi các Stored Procedure và xử lý các kết quả.

Đặc tính của Stored Procedure trong SQL Server:

- Stored Procedure là hàm cho phép truyền tham số vào và trả về giá trị.
- Bao gồm 1 tập các lệnh T-SQL để xử lý một chức năng nào đó trong cơ sở dữ liệu.

Ta có thể dùng lệnh **EXECUTE** để thực thi các Stored Procedure. Stored Procedure khác với các hàm xử lý (User-defined Function) là giá trị trả về của chúng không chứa trong tên và chúng không được sử dụng trực tiếp trong biểu thức.

Stored Procedure có những thuận lợi so với các chương trình T-SQL lưu trữ cục bộ là:

- Stored Procedure cho phép điều chỉnh chương trình cho phù hợp:

Chúng ta có thể chỉ tạo Stored Procedure một lần và lưu trữ trong Database một lần, trong chương trình chúng ta có thể gọi nó với số lần bất kỳ. Stored Procedure có thể được chỉ rõ do một người nào đó tạo ra và sự thay đổi của chúng hoàn toàn độc lập với mã nguồn (source code) của chương trình.

- Stored Procedure cho phép thực thi nhanh hơn: nếu sự xử lý yêu cầu một đoạn mã nguồn T-SQL khá lớn hoặc việc thực thi mang tính lặp đi lặp lại thì Stored Procedure thực hiện nhanh hơn việc thực hiện hàng loạt các lệnh T-SQL. Chúng được phân tích cú pháp và tối ưu hóa trong lần thực thi đầu tiên và một phiên bản dịch của chúng trong đó sẽ được lưu trong bộ nhớ để sử dụng cho lần sau, nghĩa là trong những lần thực hiện sau chúng không cần phải phân tích cú pháp và tối ưu lại, mà chúng sẽ sử dụng kết quả đã được biên dịch trong lần đầu tiên.

- Stored Procedure có thể làm giảm bớt ván đề nghẽn đường truyền mạng: giả sử một xử lý mà có sử dụng hàng trăm lệnh của T-SQL và việc thực hiện thông qua từng dòng lệnh đơn, như vậy việc thực thông qua Stored Procedure sẽ tốt hơn, vì nếu không khi thực hiện chúng ta phải gửi hàng trăm lệnh đó lên mạng và điều này sẽ dẫn đến tình trạng nghẽn mạng.
- Stored Procedure có thể sử dụng trong vấn đề bảo mật của máy: vì người sử dụng có thể được phân cấp những quyền để sử dụng các Stored Procedure này, thậm chí họ không được phép thực thi trực tiếp những Stored Procedure này.

3.2 Các Loại Stored Procedure

Stored Procedure có thể được chia thành 5 nhóm như sau:

1. **System Stored Procedure:** Là những Stored Procedure chứa trong Master database và thường bắt đầu bằng tiếp đầu ngữ `sp_`. Các Stored Procedure này thuộc loại built-in và chủ yếu dùng trong việc quản lý database (administration) và security. Ví dụ bạn có thể kiểm tra tất cả các processes đang được sử dụng bởi user `DomainName\Administrators` bạn có thể dùng `sp_who @loginame='DomainName\Administrators'`. Có hàng trăm System Stored Procedure trong SQL Server.
2. **Local Stored Procedure:** Đây là loại thường dùng nhất. Chúng được chứa trong user database và thường được viết để thực hiện một công việc nào đó. Thông thường người ta nói đến Stored Procedure là nói đến loại này. Local Stored Procedure thường được viết bởi người quản trị cơ sở dữ liệu (DBA: Database Administrator) hoặc người lập trình (programmer). Cách tạo Stored Procedure loại này sẽ được trình bày trong phần kế tiếp.
3. **Temporary Stored Procedure:** Là những Stored Procedure tương tự như Local Stored Procedure nhưng chỉ tồn tại cho đến khi kết nối đã tạo ra chúng bị đóng lại hoặc SQL Server shutdown. Các Stored Procedure này được tạo ra trên TempDB của SQL Server nên chúng sẽ bị xóa khi kết nối tạo ra chúng bị cắt đứt hay khi tắt SQL Server. **Temporary Stored Procedure được chia làm 3 loại:** **local** (bắt đầu bằng #), **global** (bắt đầu bằng ##) và Stored Procedure được **tạo ra trực tiếp trên TempDB**. Loại local chỉ được sử dụng bởi kết nối đã tạo ra chúng và bị xóa khi ngắt kết nối, còn loại global có thể được sử dụng bởi bất kỳ kết nối nào. Quyền (Permission) cho loại global là dành cho mọi người (public)

và không thể thay đổi. Loại Stored Procedure được tạo trực tiếp trên TempDB khác với 2 loại trên ở chỗ ta **có thể set permission**, chúng tồn tại kể cả sau khi **kết nối tạo ra chúng bị cắt đứt và chỉ biến mất khi SQL Server shut down**.

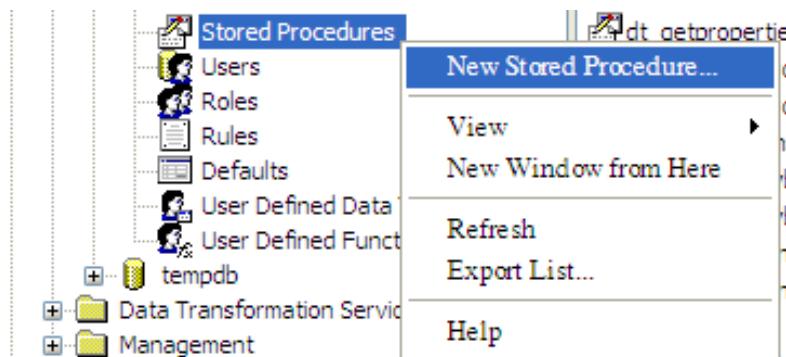
4. **Extended Stored Procedure:** Đây là một loại Stored Procedure sử dụng một chương trình ngoại vi (external program) vốn được biên dịch (compiled) thành một DLL để mở rộng chức năng hoạt động của SQL Server. Loại này thường bắt đầu bằng tiếp đầu ngữ `xp_`. Ví dụ, `xp_sendmail` dùng để gửi mail cho một người nào đó hay `xp_cmdshell` dùng để chạy một DOS command... Ví dụ `xp_cmdshell 'dir c:'`. Nhiều loại Extend Stored Procedure được xem như System Stored Procedure và ngược lại.
5. **Remote Stored Procedure:** Những Stored Procedure gọi Stored Procedure ở server khác.

3.3 Tạo Store Procedure

3.3.1 Tạo Store Procedure bằng Enterprise Manager

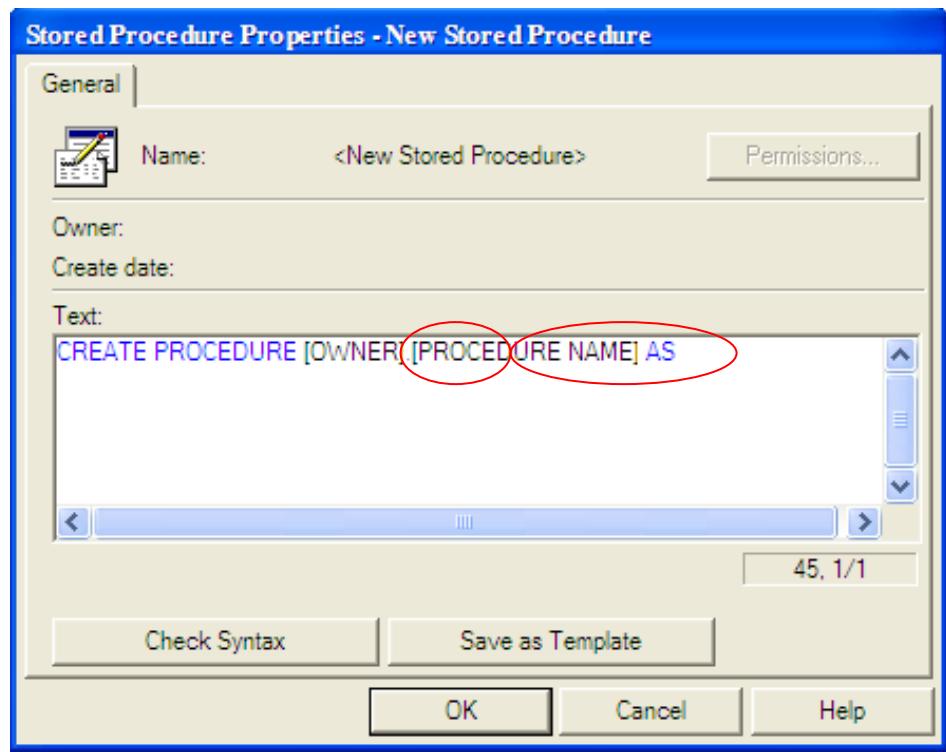
Chọn cơ sở dữ liệu cần tạo Stored Procedure trong Enterprise Manager

Right click trên **Stored Procedures** → **New Stored Procedure...** như *Hình 3.1*



Hình 3.1: Tạo Stored Procedure trong Enterprise Manager

Tạo Stored Procedure bằng Enterprise Manager sẽ phát sinh script tạo Stored Procedure sẵn, ta chỉ cần xác định vai trò (role) người khai thác tại mục **OWNER**, đặt tên thủ tục tại mục **PROCEDURE NAME**, thêm nội dung vào trong phần thân sau từ khóa **AS** như *Hình 3.2*.



Hình 3.2: Tạo Stored Procedure trong Enterprise Manager

3.3.2 Tạo Stored Procedure bằng lệnh

Cú pháp đơn giản:

```
CREATE PROCEDURE procedure_name
    @parameter1 data_type [output] /*các tham số*/,
    @parameter2 data_type [output]

AS

BEGIN
    [khai báo các biến cho xử lý]
    {Các câu lệnh transact-sql}

END

GO
```

Phần **[output]** là phần có thể có hoặc không để xác định loại tham số.

Ví dụ:

```
--Tạo Procedure 1

CREATE PROCEDURE XinChao

    @hoTen nvarchar(50)

AS
```

```

BEGIN
    print N'Xin chào ' + @hoTen
END
GO

```

```

-- Tạo Procedure 2

CREATE PROC Hello

AS

BEGIN
    print N'Hello ' + N'Các bạn'
END
GO

```

3.4 Thực thi Stored Procedure

Sử dụng lệnh **EXECUTE** (có thể viết tắt là **EXEC**) để thực thi một Stored Procedure.

```

EXECUTE procedure_name parameter_value1, parameter_value2,..
EXEC procedure_name parameter_value1, parameter_value2, ...

```

Ví dụ:

EXEC XinChao N'Nam'

Sẽ được kết quả như *hình 2.3*

Hình 3.3: Kết quả thực thi thủ tục

Xin chào

```

-- Tạo Procedure 1
CREATE PROCEDURE XinChao
    @hoTen nvarchar(50)
AS
BEGIN
    print N'Xin chào ' + @hoTen
END
GO

Exec Xinchao N'Nam'

```

EXECUTE Hello

sẽ được kết quả là như *Hình 3.4*

```

CREATE PROC Hello
AS
BEGIN
    print N'Hello ' + N'Các bạn'
END
GO

EXEC Hello

```

Hello Các bạn

Hình 3.4: Kết quả thực thi thủ tục Hello

3.5 Thay đổi nội dung Stored Procedure

Thực thi bằng lệnh

```

ALTER PROCEDURE procedure_name
    @parameter1 data_type [output] /*các tham số*/,
    @parameter2 data_type [output]
AS
BEGIN
    [khai báo các biến cho xử lý]
    {Các câu lệnh transact-sql}
END
GO

```

Lúc này, SQL Server sẽ thay thế Stored Procedure có tên “procedure_name” bằng một Stored Procedure mới có cùng tên.

Ví dụ

```

ALTER PROCEDURE Hello
    @hoten nvarchar(50),
    @lop nvarchar(20)
AS
BEGIN
    print N'Hello ' + @hoten + ' ' + @lop
END
GO

```

Thực thi lại thủ tục trên ta được kết quả sau

```

ALTER PROCEDURE Hello
@hoten nvarchar(50),
@lop nvarchar(20)
AS
BEGIN
    print N'Hello ' + @hoten + ' ' + @lop

END
GO

EXEC Hello N'Phương Nam',N'Công Nghệ TT'

```

Hello Phương Nam Công Nghệ TT

Hình 3.5: Kết quả thực thi lại thủ tục Hello sau khi chỉnh sửa

3.6 Xóa Stored Procedure

Thực thi bằng lệnh

```

DROP PROCEDURE procedure_name
DROP PROC procedure_name

```

3.7 Tham số trong Stored Procedure

Stored Procedure có thể có hai loại tham số chính: tham số **đầu vào** và tham số **đầu ra**.

3.7.1 Tham số đầu vào

Đây là loại tham số mặc định, cho phép truyền các giá trị vào trong Stored Procedure để hỗ trợ xử lý.

Ví dụ:

```

CREATE PROC Cong
@So1 int,
@So2 int
AS
BEGIN
    declare @Kq int
    set @Kq = @So1 + @So2
    print @Kq
END
GO

```

Thực thi bằng lệnh: **EXEC Cong 1, 2**

ta được kết quả như sau:

```

CREATE PROC Cong
    @So1 int,
    @So2 int
AS
BEGIN
    declare @Kq int
    set @Kq = @So1 + @So2
    print @Kq
END
GO

EXEC Cong 1, 2

```

The screenshot shows a SQL query window in SSMS. The code creates a stored procedure named 'Cong' that adds two parameters (@So1 and @So2) and prints the result. It then executes the procedure with inputs 1 and 2. The output pane below the code shows the result: 3.

Hình 3..6: Kết quả thực thi thủ tục có tham số số đầu vào

3.7.2 Tham số đầu ra

Tham số dùng để nhận kết quả trả về từ Stored Procedure. Sử dụng từ khóa **OUTPUT** (hoặc viết tắt là OUT) để xác định tham số.

Ví dụ:

```

ALTER PROC Tru
    @So1 int,
    @So2 int,
    @Kq int output
AS
BEGIN
    set @Kq = @So1 - @So2
END
GO
DECLARE @test int
EXEC Tru 5, 2, @test output
PRINT @test

```

Thực thi được kết quả như sau:

```

DECLARE @test int
EXEC Tru 5, 2, @test output
PRINT @test

```

The screenshot shows a SQL query window in SSMS. The code declares a variable @test, executes the 'Tru' stored procedure with inputs 5 and 2, and outputs the result to @test. Finally, it prints the value of @test. The output pane shows the result: 3.

Hình 3.7: Kết quả thực thi thủ tục có tham số số đầu vào ra

3.8 Trả về giá trị trong Stored Procedure

Ngoài cách sử dụng tham số đầu ra để trả về giá trị. Có thể sử dụng **RETURN** để trả về giá trị từ stored procedure hoặc các câu lệnh **SELECT** khi truy vấn dữ liệu.

3.8.1 Trả về giá trị từ lệnh RETURN

Lệnh RETURN được sử dụng để trả về giá trị từ stored procedure mà không cần sử dụng tham số đầu ra. Giá trị trả về này có một số đặc điểm:

- Giá trị trả về chỉ có thể là số nguyên. Nếu trả về các loại giá trị khác thì lúc thực thi Stored Procedure sẽ báo lỗi (ngoại trừ 1 số kiểu dữ liệu được tự động chuyển đổi sang kiểu số nguyên như:float, double,...).
- Giá trị trả về mặc định là 0.
- Có thể nhận giá trị trả về bằng 1 biến.
- Sau khi gọi RETURN, stored procedure sẽ trả về giá trị và kết thúc xử lý.

Ví dụ:

```
CREATE PROC Test
    @Lenh int
AS
BEGIN
    if (@Lenh = 1)
        return 1
    if (@Lenh = 2) begin
        declare @float float
        set @float = 2.6
        return @float
    end
    if (@Lenh = 3) begin
        declare @char varchar(50)
        set @char = 'hello'
        return @char
    end
END
GO
```

```

declare @test float
EXEC @test = Test 3
print @test

```

Nếu giá trị truyền vào là 1: Stored Procedure trả về giá trị “1”.

Nếu giá trị truyền vào là 2: Stored Procedure trả về giá trị “2”.

Nếu giá trị truyền vào là 3: Stored Procedure báo lỗi không thể chuyển chuỗi “hello” thành số nguyên.

Nếu truyền các giá trị khác: Stored Procedure trả về giá trị “0”.

3.8.2 Trả về dữ liệu từ lệnh SELECT

Mỗi lệnh SELECT đặt trong Stored Procedure sẽ trả về 1 bảng.

```

CREATE PROC TestSelect

```

```

AS

```

```

BEGIN

```

```

    SELECT * FROM NHANVIEN

```

```

    SELECT * FROM PHONG

```

```

END

```

```

GO

```

Thực thi thủ tục trên bằng lệnh :

```

EXEC TestSelect

```

Sẽ được kết quả như sau:

	MaNV	Ho	TenLot	Ten	GoiTinh	NgaySinh	HeSoLuong	MaPhong	Tel	NgayBC
1	HC001	Nguyễn	Thị	Hà	N?	27/02/1950	2.50	HCA	0123456789	02/08/1975
2	HC002	Trần	Văn	Nam	Nam	06/12/1975	3.00	HCA	0123456788	06/08/1997
3	HC03	Nguyễn	Thanh	Huyền	N?	07/03/1978	1.50	HCA	0123456789	09/02/1999
4	KD001	Lê	Tuyết	Anh	N?	03/02/1977	3.00	KDA	0123456787	10/02/2001
5	KD002	Nguyễn	Anh	Tú	Nam	07/04/1942	1.50	KDA	0123456789	49/04/1999
6	KD003	Phạm	An	Thái	Nam	05/09/1977	1.60	KDA	0123456666	09/04/1999
7	KD004	Lê	Văn	Hải	Nam	01/04/1976	2.70	KDA	0123456766	06/08/1997
8	KD005	Phạm		Minh	Nam	01/02/1980	2.00	KDA	0123456676	10/02/2001
	MaPhong	TenPhong	DiaChi						Tel	
1	HCA	Hành chính tổng hợp	123, Phường 5, TP Trà Vinh,...						0743585793	
2	KDA	Kinh Doanh	123, Phường 5, TP Trà Vinh,...						0743585794	
3	KTA	Kỹ thuật	123, Phường 5, TP Trà Vinh,...						0743585795	
4	QTA	Quản trị	123, Phường 5, TP Trà Vinh,...						0743585796	

Hình 3.8: Kết quả thực thi thủ tục với kết quả trả về từ câu lệnh SELECT

3.9 Kết hợp Stored Procedure với các lệnh T-SQL

Các Stored Procedure thông thường được tạo ra nhằm giúp thực hiện một số chức năng cần thao tác trong cơ sở dữ liệu. Khi đó, ta cần phải kết hợp nhiều lệnh T-SQL thao tác với dữ liệu như (SELECT, INSERT, UPDATE, DELETE) và các cấu trúc điều khiển (IF, WHILE, CASE,...).

3.9.1 *Ứng dụng thêm PHÒNG vào cơ sở dữ liệu*

Create PROC ThemPhong

```
@maphong char(3),  
@tenphong nvarchar(40),  
@diachi nvarchar(50),  
@tel char (10)  
AS  
BEGIN  
IF(EXISTS(SELECT * FROM Phong p WHERE p.maphong = @maphong))  
BEGIN  
    PRINT N'Mã phòng ' + @maphong + N' đã tồn tại'  
    RETURN -1  
END  
IF(NOT EXISTS(SELECT * FROM Phong p WHERE p.maphong =  
@maphong))  
BEGIN  
    PRINT N'Mã phòng' + @maphong + N' chưa tồn tại'  
    INSERT INTO phong(maphong,tenphong,diachi,tel)  
    VALUES(@maphong,@tenphong,@diachi,@tel)  
--RETURN 0 /* procedure tự trả về 0 nếu không RETURN */  
END  
END  
GO
```

```
DECLARE @kq INT  
EXEC @kq = ThemPhong 'NSA', N'Nhân sự', N'123, Phường 5, TP Trà  
Vinh','0743585798'  
PRINT @kq
```

3.9.2 *Ứng dụng trả về danh sách các PHÒNG*

```
CREATE PROC InPhong
```

```
    Alter Proc InPhong
```

```
        @maphong varchar(10)
```

```
    AS
```

```
    BEGIN
```

```
        IF(NOT EXISTS(SELECT * FROM Phong p WHERE p.maphong =  
        @maphong))
```

```
            BEGIN
```

```
                PRINT N'Mã Phòng ' + @maphong + N' chưa tồn tại'
```

```
                RETURN -1
```

```
            END
```

```
            SELECT * FROM Phong p where p.maphong = @maphong
```

```
        /*procedure luôn trả về 0 nếu không RETURN*/
```

```
    END
```

```
    GO
```

```
DECLARE @KQ INT
```

```
EXEC @KQ=InPhong 'HCA'
```

```
PRINT @KQ
```

PHỤ LỤC 4: TRIGGER

4.1 Giới thiệu về Trigger

Trigger là một Stored Procedure đặc biệt mà việc thực thi của nó tự động khi có sự kiện xảy ra, các sự kiện gọi Stored Procedure đặc biệt này được định nghĩa trong câu lệnh, thông thường được thực hiện với các sự kiện liên quan đến Insert, Update, Delete dữ liệu.

Trigger được sử dụng trong việc bảo đảm toàn vẹn dữ liệu theo quy tắc xác định, được quản lý theo bảng dữ liệu hoặc khung nhìn.

4.2 Những trường hợp sử dụng trigger

- Sử dụng Trigger khi các biện pháp toàn vẹn dữ liệu như Constraint, rule,... không bảo đảm. Khác với các công cụ bảo đảm toàn vẹn dữ liệu, các công cụ này sẽ thực hiện kiểm tra tính toàn vẹn trước khi đưa dữ liệu vào CSDL (còn gọi là Declarative Data Integrity), còn Trigger thực hiện kiểm tra tính toàn vẹn khi công việc đã thực hiện rồi (còn gọi là Procedural Data Integrity).

- Khi CSDL chưa được chuẩn hóa (Normalization) thì có thể xảy ra dữ liệu thừa, chứa ở nhiều vị trí trong CSDL thì yêu cầu đặt ra là dữ liệu cần cập nhật thống nhất trong mọi nơi. Trong trường hợp này ta phải sử dụng Trigger.

- Khi thay đổi dây chuyền dữ liệu giữa các bảng với nhau (khi dữ liệu bảng này thay đổi thì dữ liệu trong bảng khác cũng được thay đổi theo).

4.3 Đặc điểm của trigger

- Một trigger có thể thực hiện nhiều công việc (theo kịch bản), có thể nhiều sự kiện kích hoạt thực thi trigger, có thể tách rời các sự kiện trong một trigger.

- Trigger không được tạo trên bảng temprate hay system.

- Trigger chỉ thực thi tự động thông qua các sự kiện mà không thực hiện bằng tay.

- Trigger sử dụng được với View.

- Khi trigger thực thi theo các sự kiện Insert hoặc Delete thì dữ liệu khi thay đổi sẽ được chuyển sang các bảng Inserted Table, Deleted Table, là 2 bảng tạm thời chỉ chứa trong bộ nhớ, các bảng này chỉ được sử dụng với các lệnh trong trigger. Các bảng này thường được sử dụng để khôi phục lại phần dữ liệu đã thay đổi (roll back).

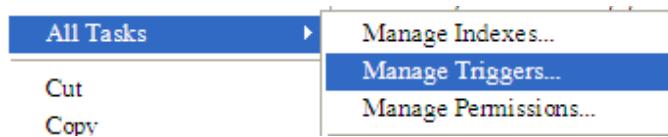
- Trigger chia thành 2 loại **Instead of** và **After**: Instead of là loại trigger mà hoạt động của sự kiện gọi nó sẽ bỏ qua và thay vào nó là các lệnh thực hiện trong trigger. After (tương đương với từ khóa For) đây là loại ngầm định, khác với loại Instead of thì loại trigger này sẽ thực hiện các lệnh trong nó sau khi đã thực hiện xong sự kiện gọi nó.

4.4 Tạo trigger

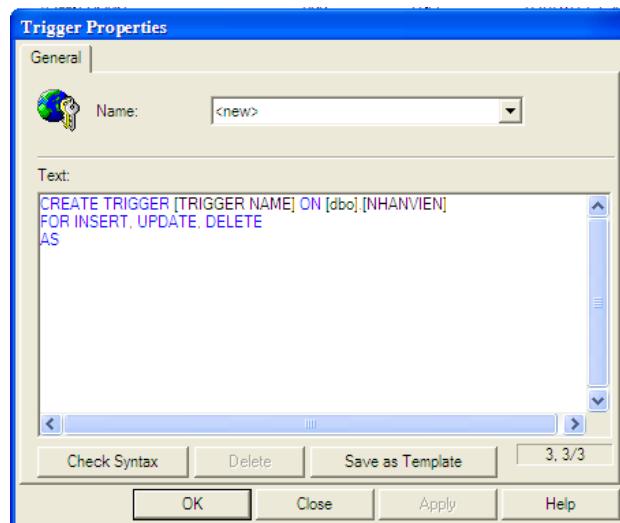
Tạo trigger được thực hiện thông công cụ và câu lệnh:

4.4.1 Tạo trigger bằng công cụ

- Chọn bảng dữ liệu hoặc View.
- Nhấn nút phải chuột.
- All tasks -> Manage Triggers... như hình 4.1



Hình 4.1: Tạo Trigger bằng công cụ



Hình 4.2: Tạo Trigger bằng công cụ

4.4.2 Tạo trigger bằng câu lệnh

Sử dụng lệnh **Create Trigger**, cú pháp chung như sau:

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
    { { FOR | AFTER | INSTEAD OF } { [ INSERT ][,][ UPDATE ] }
        [ WITH APPEND ]
        [ NOT FOR REPLICATION ]
        AS
        [ { IF UPDATE ( column )
            [ { AND | OR } UPDATE ( column ) ]
            [ ...n ]
        | IF ( COLUMNS_UPDATED () { bitwise_operator } updated_bitmask )
            { comparison_operator } column_bitmask [ ...n ]
        }
        sql_statement [ ...n ]
    }
}
```

Các tham số cơ bản:

- + **trigger_name**: tham số chỉ định tên của trigger sẽ tạo.
- + **ON**: Chỉ định lấy tên của bảng hoặc View mà ta sẽ xây dựng trigger trên đó.
- + **table | view**: Tên của bảng hoặc của View.
- + **WITH ENCRYPTION**: Dùng để chỉ định code của trigger mã hóa lưu trữ trên bảng syscomments mà người khác không thể xem code của trigger đó.
- + **FOR**: Định nghĩa hành động mà trigger được kích nổ và kiểu của trigger ta đang tạo. AFTER trigger sẽ là trigger mặc định nếu chỉ có chỉ định FOR.
- + **AFTER**: Chỉ định AFTER trigger. Tùy chọn này chỉ định riêng không lẫn với từ khóa INSTEAD OF. AFTER triggers không được định nghĩa trên view.
- + **INSTEAD OF**: Từ khóa chỉ định đây là INSTEAD OF trigger. Tùy chọn này chỉ định để không lẫn với từ khóa AFTER.
- + **DELETE**: Chỉ định rằng trigger ta đang tạo sẽ được kích hoạt đáp ứng với hành động DELETE của bảng hoặc View.
- + **INSERT**: Chỉ định rằng trigger ta đang tạo sẽ được kích hoạt đáp ứng với hành động INSERT của bảng hoặc View.
- + **UPDATE**: Chỉ định rằng trigger ta đang tạo sẽ được kích hoạt đáp ứng với hành động UPDATE của bảng hoặc View.

- + **NOT FOR REPLICATION**: Chỉ định rằng bất cứ bản sao nào của các hành động chạy ngầm dưới bảng này đều không được kích hoạt trigger này.
- + **AS**: Chỉ định phần code của trigger bắt đầu từ đây.
- + **IF UPDATE (column)**: Được dùng trong các trigger INSERT, UPDATE. Cấu trúc này được dùng để kiểm tra các sửa đổi trên cột chỉ định và sau đó là các hành động trên nó.
- + **{AND | OR} UPDATE (column)**: Chỉ định rằng ta có thể sử dụng chuỗi các cấu trúc UPDATE với nhau để kiểm tra một vài cột tại cùng một thời điểm.
- + **...n**: Chỉ định ta có thể lặp lại các cấu trúc trên nếu cần thiết.
- + **IF(COLUMNS_UPDATED())**: Chỉ dùng trong các trigger INSERT, UPDATE. Hàm trả về một bit chỉ định cột bị chỉnh sửa trong quá trình INSERT, UPDATE trên bảng cơ sở.
- + **bitwise_operator**: Được dùng để so sánh với bit trả về của hàm COLUMNS_UPDATED()
- + **updated_bitmask**: Được sử dụng để kiểm cột nào thực sự được Update trong câu lệnh Insert hoặc Update.
- + **sql_statement**: Các câu lệnh T-SQL
- + **... n**: Chỉ định lặp lại các câu lệnh T-SQL.

Ví dụ:

```

CREATE TRIGGER INSERT_NHANVIEN
ON nhanvien
FOR INSERT
AS
IF (Select Min(HeSoLuong) from nhanvien)<=0
Begin
    Print N'Hệ số lương phải lớn hơn 0'
    RollBack Transaction
    Print N'Không thêm được'
End

```

Kết quả

```

INSERT INTO NHANVIEN VALUES
('KT012', N'Nguyễn', N'Mới', N'Thêm 2', N'Nam', '08/08/1978', 0.00, 'KTA', '012345676', '09/02/1999')

Hệ số lương phải lớn hơn 0
Không thêm được

```

Hình 4.3: Kết quả lệnh Insert khi tạo Trigger

4.5 Sửa, xóa trigger

4.5.1 Sửa, xóa trigger bằng cách sử dụng công cụ

- Chọn trigger trong mục Manager Triggers...
- Thực hiện sửa nội dung hoặc xóa.

4.5.2 Sửa, xóa trigger bằng câu lệnh

- Dùng lệnh **Alter trigger** để sửa.

Cú pháp:

```
ALTER TRIGGER trigger_name
ON ( table | view )
[ WITH ENCRYPTION ]
{
    { ( FOR | AFTER | INSTEAD OF ) { [ DELETE ] [,][ INSERT ][,][ UPDATE ] }
        [ NOT FOR REPLICATION ]
        AS
        sql_statement [ ...n ]
    }
    |
    { ( FOR | AFTER | INSTEAD OF ) { [ INSERT ][,][ UPDATE ] }
        [ NOT FOR REPLICATION ]
        AS
        { IF UPDATE ( column )
            [ { AND | OR } UPDATE ( column ) ]
            [ ...n ]
            | IF ( COLUMNS_UPDATED () { bitwise_operator } updated_bitmask )
                { comparison_operator } column_bitmask [ ...n ]
            }
            sql_statement [ ...n ]
        }
    }
}
```

Ví dụ:

```
ALTER TRIGGER INSERT_NHANVIEN
ON nhanvien
FOR INSERT
AS
IF (Select Max(HeSoLuong) from nhanvien)>10.0
    Begin
        Print N'Hệ số lương không được lớn hơn 10'
        RollBack Transaction
        Print N'Không thêm được'
    End
```

Kết quả:

The screenshot shows a SQL query window with the following content:

```
INSERT INTO NHANVIEN VALUES
('KT012', N'Nguyễn', N'Mới', N'Thêm 2', N'Nam', '08/08/1978', 10.5, 'KTA', '012345678', '09/02/1999')
```

Below the query, an error message is displayed:

Hệ số lương không được lớn hơn 10
Không thêm được

Hình 4.4: Kết quả sau Insert khi sử dụng Trigger

- Sử dụng lệnh **Drop Trigger** để xóa.

Cú pháp:

DROP TRIGGER { trigger } [,...n]

Ví dụ:

DROP Trigger INSERT_NHANVIEN

PHỤ LỤC 5: GIAO TÁC

5.1 Làm việc với **BEGIN TRAN (BEGIN TRANSACTION)** và **COMMIT TRAN**

BEGIN TRAN // Bắt đầu transaction

-- Lệnh SQL 1

-- Lệnh SQL 2

....

COMMIT TRAN // Báo kết thúc transaction

Ví dụ:

BEGIN TRAN

INSERT INTO phong

VALUES ('TTA',N'Truyền thông', N'123, Phường 5, TP Trà
Vinh', '0743585798')

COMMIT TRAN

Tuy nhiên, ta cũng có thể bắt đầu và kết thúc Transaction với tên của
Transaction

BEGIN TRAN tran_name

INSERT INTO phong

VALUES ('KTB',N'Kế toán', N'123, Phường 5, TP Trà
Vinh', '0743585799')

COMMIT TRAN tran_name

5.2 Làm việc với **ROLLBACK TRAN**

Trong trường hợp ta huỷ bỏ những phần đã xử lý thành công, ta có thể sử dụng
phát biểu **ROLLBACK TRAN**

Nếu trong Transaction đơn, khi gặp phải phát biểu **ROLLBACK TRAN** thì
SQL sẽ huỷ tất cả những phần đã xử lý sau phát biểu **BEGIN TRAN**. Tuy nhiên, trong
trường hợp có chỉ định tên của Transaction thì sau phát biểu **BEGIN TRAN** phải chỉ
định tên Transaction

Ví dụ:

Nếu viết lệnh như sau:

DELETE FROM phong

WHERE tenphong=N'Truyền thông'

```
INSERT INTO Phong
```

```
VALUES ('KTAC',N'Kinh tế', N'123, Phường 5, TP Trà Vinh','0743585691')
```

Khi thực thi đoạn lệnh trên thì một record bị xoá, nhưng không thêm được record mới.

Nhưng nếu viết đoạn lệnh trên được cài đặt trong phát biểu **BEGIN TRAN**, **ROLLBACK TRAN** và **COMMIT TRAN** như ví dụ sau:

Ví dụ:

```
BEGIN TRAN capnhatpb
```

```
DELETE FROM phong
```

```
WHERE tenphong=N'Truyền thông'
```

```
INSERT INTO Phong
```

```
VALUES ('KTAC',N'Kinh tế', N'123, Phường 5, TP Trà Vinh','0743585691')
```

```
If @@ERROR != 0 -- neu co loi xay ra, rollback
```

```
Begin
```

```
Print 'rollback'
```

```
ROLLBACK TRAN capnhatpb
```

```
End
```

```
Else
```

```
COMMIT TRANSACTION
```

Khi thực thi đoạn lệnh trên thì mẫu tin bị xoá sẽ được phục hồi.

5.3 Sử dụng Stored Procedure

Nếu khai báo **DELETE** và **INSERT** trong thủ tục nội tại thì hai phát biểu này được xem như một TRANSACTION, có nghĩa là nếu **INSERT** bị lỗi thì tự thân phát biểu **DELETE** sẽ không thực thi.

Ví dụ:

```
CREATE PROC NOTRAN
```

```
AS
```

```
DELETE FROM phong
```

```
WHERE tenphong=N'Truyền thông'
```

```
INSERT INTO Phong
```

```
VALUES ('KTAC',N'Kinh tế', N'123, Phường 5, TP Trà
```

```
Vinh','0743585691')
```

```
GO
```

5.4 Làm việc với SAVE TRAN

Ta có thể chỉ định vị trí mà hệ thống gấp phát biểu **ROLLBACK TRAN** trở về để phục hồi thay vì vị trí ngay sau phát biểu **BEGIN TRAN** bằng cách sử dụng phát biểu **SAVE TRAN**

Ví dụ:

Giả sử ta thêm mới 3 mẫu tin vào bảng **phongban** bằng các lệnh như sau:

```
INSERT INTO PHONG
```

```
VALUES ('MT1,N'moi them 1', N'123, Phường 5, TP Trà Vinh',  
'0743585692')
```

```
INSERT INTO PHONG
```

```
VALUES ('MT2,N'moi them 2', N'123, Phường 5, TP Trà Vinh',  
'0743585693')
```

```
INSERT INTO PHONG
```

```
VALUES ('MT3,N'moi them 3', N'123, Phường 5, TP Trà Vinh',  
'0743585693')
```

```
/* xem kết quả mới thêm*/
```

```
Select *
```

```
From phong
```

Chẳng hạn ta muốn thay đổi tên phòng mới thêm từ :

“moi them 1” thành **“moi cap nhat 1”**

“moi them 2” thành **“moi cap nhat 2”**

“moi them 3” thành **“moi cap nhat 3”**

Và chèn thêm mẫu tin thứ 4, tuy nhiên khi thêm mẫu tin thứ 4 thì việc thêm này vi lỗi, không thể chèn thêm được. Thay vì huỷ bỏ sự thay đổi của 3 mẫu tin trên ta chỉ cần huỷ bỏ mẫu tin thứ 2 và thứ 3.

Ví dụ:

```
Select * From phong --xem lại danh sách các phòng ban
```

```
BEGIN TRAN
```

```
--cập nhật mẫu tin thứ 1
```

```

UPDATE phong SET tenphong = 'moi cap nhat 1'
Where tenphong='moi them 1'
--khai báo điểm trả về
SAVE TRAN del
--cập nhật mẫu tin thứ 2 và thứ 3
UPDATE phong SET tenphong = 'moi cap nhat 2'
Where tenphong='moi them 2'
UPDATE phong SET tenphong = 'moi cap nhat 3'
Where tenphong='moi them 3'
--thêm mẫu tin thứ 4
INSERT INTO PHONG
VALUES ('MT33','day la mau tin moi them de kiem tra loi phat sinh',
N'123, Phường 5, TP Trà Vinh', '0743585693')
IF @@ERROR != 0 -- neu co loi xay ra, rollback
Begin
    PRINT 'rollback'
    ROLLBACK TRAN del
End
ELSE
COMMIT TRANSACTION

```

--liệt kê danh sách các phòng ban để kiểm tra

```

Select *
From phong

```

Nếu không khai báo **SAVE TRAN** thì quá trình phục hồi mẫu tin đã được cập nhật được thực hiện từ đầu.

Ví dụ:

--xem lại danh sách các phòng ban

```

Select *
From phongb

```

BEGIN TRAN

UPDATE Phong SET tenphong = 'moi cap nhat 1'

Where tenphong='moi them 1'

UPDATE Phong SET tenphong = 'moi cap nhat 2'

Where tenphong='moi them 2'

```
UPDATE Phong SET tenphong = 'moi cap nhat 3'
```

```
Where tenphong='moi them 3'
```

```
INSERT INTO PHONG
```

```
VALUES ('MT33','day la mau tin moi them de kiem tra loi phat sinh', N'123,  
Phường 5, TP Trà Vinh', '0743585693')
```

```
IF @@ERROR != 0 -- neu co loi xay ra, rollback
```

```
Begin
```

```
    Print 'rollback'
```

```
    ROLLBACK TRAN
```

```
    End
```

```
    Else
```

```
COMMIT TRANSACTION
```

```
--liệt kê danh sách các phòng ban để kiểm tra
```

```
SELECT *
```

```
FROM phong
```

PHỤ LỤC 6: BÀI TẬP THỰC HÀNH SỐ 1

I. Tạo người dùng trên SQL Server (*Lý thuyết Chương 4*)

II. Viết lệnh tạo CSDL với tên QUANLY_NHANVIEN có lược đồ sau:

(Các lệnh tạo CSDL và tạo bảng tham khảo PHỤ LỤC 1)

1). Viết lệnh tạo các bảng sau

1. **PHONG** (MaPhong (char(3)), TenPhong (Nvarchar(40)),

 DiaChi(Nvarchar(50)), Tel (char(10))

2. **DANHMUC_NN** (MaNN(char(2)), TenNN(Nvarchar(20)))

3. **NHANVIEN** (MaNV(Char(5)), Ho(Nvarchar(20)),

 TenLot(Nvarchar(20)), Ten (Nvarchar(20)),

 GioiTinh(Nvarchar(3)), NgaySinh (Char(10)),

 HeSoLuong (Dec (4,2)), MaPhong(Char(3)),

 Tel(Char(10)), NgayBC(Date/Time))

4. **TRINHDO_NN** (MaNV(char(5)), MaNN(char(2)), TrDo(char(2)))

2). Tạo diagram cho CSDL trên

III. Viết Trigger để kiểm tra các ràng buộc toàn vẹn cho các trường hợp thao tác

trên dữ liệu gồm: (*Cách tạo Trigger tham khảo PHỤ LỤC 4*)

1) Thêm mới,

2) Cập nhật,

3) Xóa dữ liệu

IV. Viết lệnh để nhập dữ liệu cho các bảng trong cơ sở dữ liệu

QUANLY_NHANVIEN theo số liệu sau:

(Lệnh nhập dữ liệu tham khảo PHỤ LỤC 1)

1. **PHONG**

Ma Phong	TenPhong	Diachi	Tel
HCA	HC Tổng hợp	123, Phường 5, TP Trà Vinh, Trà Vinh	0743 585793
KDA	Kinh Doanh	123, Phường 5, TP Trà Vinh, Trà Vinh	0743 585794
KTA	Kỹ thuật	123, Phường 5, TP Trà Vinh, Trà Vinh	0743 585795
QTA	Quản trị	123, Phường 5, TP Trà Vinh, Trà Vinh	0743 585796

2. DANHMUC_NN (Danh mục ngoại ngữ)

MaNN	TenNN
01	Anh
02	Nga
03	Pháp
04	Nhật
05	Trung Quốc
06	Hàn Quốc

3. NHANVIEN

STT	MaNV	Ho	Tenlot	Ten	Gioi Tinh	Ngay Sinh	HeSo Luong	Ma Phong	Tel	NgayBC
1.	HC001	Nguyễn	Thị	Hà	Nữ	27/02/1950	4.50	HCA		02/08/1975
2.	HC002	Trần	Văn	Nam	Nam	06/12/1975	3.00	HCA		06/08/1997
3.	HC003	Nguyễn	Thanh	Huyền	Nữ	07/03/1978	3.50	HCA		09/02/1999
4.	KD001	Lê	Tuyết	Anh	Nữ	02/03/1977	2.50	KDA		10/02/2001
5.	KD002	Nguyễn	Anh	Tú	Nam	07/04/1942	2.60	KDA		29/04/1999
6.	KD003	Phạm	An	Thái	Nam	05/09/1977	3.60	KDA		09/04/1999
7.	KD004	Lê	Văn	Hải	Nam	01/02/1976	2.70	KDA		06/08/1997
8.	KD005	Nguyễn	Phương	Minh	Nam	01/02/1980	3.00	KDA		10/02/2001
9.	KT001	Trần	Đình	Khâm	Nam	12/02/1981	2.70	KTA		01/10/2005
10.	KT002	Nguyễn	Mạnh	Hùng	Nam	08/06/1980	2.30	KTA		10/12/2005
11.	KT003	Phạm	Thanh	Sơn	Nam	08/09/1984	2.00	KTA		17/11/2005
12.	KT004	Vũ	Thị	Hoài	Nữ	12/05/1980	2.50	KTA		10/02/2001
13.	KT005	Nguyễn	Thu	Lan	Nữ	10/05/1977	3.00	KTA		10/12/2001
14.	KT006	Trần	Hoài	Nam	Nam	17/02/1978	2.80	KTA		16/08/1997
15.	KT007	Hoàng	Nam	Sơn	Nam	12/03/1940	4.50	KTA		07/02/1965
16.	KT008	Lê	Thu	Trang	Nữ	07/06/1950	4.50	KTA		08/02/1968
17.	KT009	Khúc	Nam	Hải	Nam	07/02/1980	2.00	KTA		01/01/2005
18.	KT010	Phùng	Trung	Dũng	Nam	08/08/1978	2.20	KTA		09/02/1999

4. Nhập dữ liệu cho bảng **TRINHDO_DNN** gồm các bản ghi:

MaNV	MaNN	TrDo	MaNV	MaNN	TrDo	MaNV	MaNN	TrDo
HC001	01	A	KD003	01	B	KT001	01	B1
HC001	02	B	KD003	02	C	KT001	04	E
HC002	01	C	KD004	01	C	KT002	01	C
HC002	03	C	KD004	04	A	KT002	02	B
HC003	01	B2	KD004	05	A	KT003	01	B1
KD001	01	C	KD005	01	B	KT003	03	C
KD001	02	B	KD005	02	D	KT004	01	D
KD002	01	D	KD005	03	B	KT005	01	C
KD002	02	A	KD005	04	B	KT010	03	A

V. Thực hiện cấp phát quyền cho người dùng truy cập vào cơ sở dữ liệu trên (Lý thuyết Chương 4)

VI. Tạo vai trò của người dùng trên SQL Server và cơ sở dữ liệu trên

VII. Viết View thực hiện các câu yêu cầu sau:

(Cách tạo View tham khảo PHỤ LỤC 2)

- 1) Đưa ra thông tin của nhân viên có mã số KT001?
- 2) Đưa ra danh sách các nhân viên nữ?
- 3) Tìm những nhân viên có họ ‘Nguyễn’?
- 4) Đưa ra danh sách các nhân viên có tên chứa từ ‘Văn’
- 5) Đưa ra những nhân viên có tuổi dưới 30? (Đưa ra cả thông tin tuổi trong kết quả)
- 6) Đưa ra danh sách các nhân viên có tuổi nằm trong khoảng 25 đến 30 tuổi? (Đưa ra cả thông tin tuổi trong kết quả)
- 7) Đưa ra các mã nhân viên đã học các ngoại ngữ 01 ở trình độ C trở lên?
- 8) Đưa ra danh sách các nhân viên vào biên chế trước năm 2000?
- 9) Đưa ra danh sách các nhân viên đã vào biên chế hơn 10 năm?

VIII. Thực hiện cấp phát quyền cho người dùng trên các View trên

IX. Sử dụng ít nhất hai quyền khác nhau vừa cấp phát ở mục VIII để truy cập vào các View trên và so sánh quyền truy cập giữa các người dùng

X. Viết các thủ tục lưu trữ thực hiện các câu yêu cầu sau:

(Cách tạo thủ tục tham khảo PHỤ LỤC 3)

- 1) Đưa ra danh sách các nhân viên nam nay đủ tuổi nghỉ hưu
(Nam ≥ 60 tuổi, Nữ ≥ 55 tuổi)?

- 2) Cho biết số người đến tuổi nghỉ hưu (số Nam riêng, số Nữ riêng). *Các giá trị này được truyền cho tham số đầu ra của thủ tục*
- 3) Cho biết thông tin (Mã phòng, tên phòng, điện thoại liên hệ) về các phòng ban?
- 4) Đưa ra thông tin (họ tên, ngày sinh, ngày vào biên chế) *về 2 nhân viên đầu tiên* trong bảng nhân viên?
- 5) Cho biết mã nhân viên, họ tên, ngày sinh, lương của các nhân viên có hệ số lương nằm trong **khoảng từ 2.00 đến 3.00**?
- 6) Cho biết số người có hệ số lương nằm trong khoảng từ **3.50 đến 4.50** (*tham số này được truyền cho tham số đầu ra của thủ tục*)
- 7) Đưa ra danh sách các nhân viên chưa có số điện thoại?
- 8) Đưa ra danh sách các nhân viên sinh nhật hàng tháng (*số tháng sinh nhật muốn tìm được truyền từ tham số đầu vào của thủ tục*)
- 9) Đếm số nhân viên sinh nhật hàng tháng (*Các giá trị này cũng được truyền cho tham số đầu ra của thủ tục*)
- 10) Hãy đưa ra danh sách nhân viên theo chiều tăng dần của hệ số lương?
- 11) Cho biết tổng số nhân viên và trung bình lương phòng Kinh doanh? (*Kết quả truyền qua tham số của thủ tục*)
- 12) Cho biết tổng lương của mỗi phòng?
- 13) Cho biết các phòng có tổng lương lớn hơn nhất?
- 14) Cho biết danh sách mã nhân viên, họ tên, mã phòng và tên phòng họ làm việc?

XI. Thực hiện cấp phát quyền cho người dùng trên các thủ tục lưu trữ trên

XII. Thực hiện loại bỏ quyền cho người dùng truy cập vào cơ sở dữ liệu và View trên

XIII. Thực hiện việc sao lưu và khôi phục dữ liệu

PHỤ LỤC 7: BÀI TẬP THỰC HÀNH SỐ 2

I. Viết lệnh tạo cơ sở dữ liệu có tên là QUANLY_SINHVIEN gồm các bảng sau:

1). Viết lệnh tạo các bảng sau

1. KHOA (**MaKhoa (char(4))**, TenKhoa(nvarchar(30)), DiaChi(nvarchar(50)),
DienThoai(varchar(10)))
2. LOP (**MaLop (char(10))**, TenLop(nvarchar(30)), MaKhoa(char(4)))
3. SINHVIEN (**MaSV(char(8))**, HoTenSV(nvarchar(30)), NgaySinh(Datetime),
MaLop(char(10)))
4. MONHOC (**MaMon(varchar(5))**, TenMon(nvarchar(30)), SoTinChi(SmallInt))
5. GIANGVIEN (**MaGV(char(9))**, HoTenGV(nvarchar(40)), MaKhoa(char(4)))
6. GIANGDAY (**MaMon(varchar(5))**, **MaGV(char(9))**, **NamHoc(char(9))**,
HocKy(SmallInt), **MaLop (char(10))**)
7. DANGKYHOC (**MaSV(char(8))**, **MaMon(varchar(5))**, **NamHoc(char(9))**,
HocKy(SmallInt), **MaGV(char(9))**, Diem (SmallInt))

2). Tạo diagram cho CSDL trên

II. Viết Trigger để kiểm tra các ràng buộc toàn vẹn cho các trường hợp thao tác trên dữ liệu gồm: (Cách tạo Trigger tham khảo PHỤ LỤC 4)

- 1) Thêm mới,
- 2) Cập nhật,
- 3) Xóa dữ liệu

III. Viết lệnh để nhập dữ liệu cho các bảng trong cơ sở dữ liệu QUANLY_SINHVIEN theo số liệu sau:

1.Khoa

MaKhoa	TenKhoa	DiaChi	DienThoai
TOAN	Toán – Tin	Nhà C1	0743 447325
CNTT	Công nghệ Thông tin	Nhà C2	0743 447326
D_LY	Địa lý	Nhà A1	0743 447327
HHOC	Hóa học	Nhà A2	743 328

2. Lop

MaLop	TenLop	MaKhoa
DA11TO12A1	ĐH Toán A1 2012	TOAN
DA12TT12A1	ĐH CNTT A1 2012	CNTT
DA12DL12A1	ĐH Địa lý A1 2012	D_LY
DA12TT12A2	ĐH CNTT A2 2012	CNTT
DA12DL12A2	ĐH Địa lý A2 2012	D_LY
DA12HH12A1	ĐH Hóa học A1 2012	HHOC

3.SinhVien

MaSV	HoTenSV	NgaySinh	MaLop
K6100001	Phạm Văn Bình	24-02-1990	DA11TO12A1
K6100002	Nguyễn Thị Hoài Thu	12-04-1991	DA12TT12A1
K6100003	Trần Ngọc Thanh	15-04-1990	DA12DL12A1
K6100004	Nguyễn Tân Hùng	03-02-1992	DA12TT12A2
K6100005	Trương Thành Sang	04-12-1990	DA12DL12A2
K6100006	Nguyễn Anh Dũng	03-03-1982	DA12HH12A1
K6100007	Phạm Hồng Ánh	24-02-1990	DA11TO12A1
K6100008	Lê Thị Liễu	12-04-1991	DA12TT12A1
K6100009	Phạm Ngọc Sương	17-04-1990	DA12DL12A1
K6100010	Nguyễn Tân	03-12-1992	DA12TT12A2
K6100011	Thạch Thanh Sang	04-02-1990	DA12DL12A2
K6100012	Nguyễn Hồng Anh Ngọc	03-12-1992	DA12HH12A1
K6100013	Nguyễn Hồng Ngọc	13-12-1992	DA12TT12A2
K6100014	KimThanh Sang	04-02-1990	DA12DL12A2
K6100015	Hồng Anh Ngọc	13-12-1992	DA12HH12A1

4.MonHoc

MaMon	TenMon	SoTinChi
GTA1	Giải tích 1	2
DSTT	Đại số tuyến tính	3
HHAf	Hình học Afin	2
XSTK	Xác suất thống kê	2

MaMon	TenMon	SoTinChi
THDC	Tin học đại cương	3
KTLT	Kỹ thuật lập trình	2

3. GiangVien

MaGV	HoTenGV	MaKhoa
15.111.01	Phạm Bình Minh	TOAN
15.111.02	Nguyễn Hoài Thu	CNTT
15.111.03	Trần Ngọc Ân	D_LY
15.111.04	Nguyễn Hùng Anh	HHOC
15.111.05	Phạm Ngọc Diệp	TOAN
15.111.06	Lê Anh Dũng	CNTT
15.111.07	Phạm Hồng Ánh	TOAN
15.111.08	Lê Thị Liễu Phượng	HHOC
15.111.09	Phạm Ngọc Thu Sương	CNTT
15.111.10	Võ Anh Tài	CNTT

4. GiangDay

MaMon	MaGV	NamHoc	HocKy	Lop
GTA1	15.111.01	2012-2013	1	DA11TO12A1
DSTT	15.111.05	2012-2013	2	DA11TO12A1
HHAFF	15.111.07	2012-2013	2	DA11TO12A1
XSTK	15.111.01	2011-2012	2	DA12HH12A1
THDC	15.111.10	2012-2013	1	DA12DL12A1
THDC	15.111.09	2011-2012	1	DA12HH12A1
THDC	15.111.02	2012-2013	1	DA12DL12A2
KTLT	15.111.02	2012-2013	2	DA12TT12A1

5. DangKyHoc

Stt	MaSV	MaMon	NamHoc	HocKy	MaGV	Diem
1.	K6100001	GTA1	2012-2013	1	15.111.01	6
2.	K6100001	DSTT	2012-2013	2	15.111.05	7
3.	K6100001	HHAFF	2012-2013	2	15.111.07	5
4.	K6100002	DSTT	2012-2013	2	15.111.05	4
5.	K6100002	XSTK	2011-2012	2	15.111.01	3

Stt	MaSV	MaMon	NamHoc	HocKy	MaGV	Diem
6.	K6100002	GTA1	2012-2013	1	15.111.01	2
7.	K6100003	HHAF	2012-2013	2	15.111.07	5
8.	K6100003	GTA1	2012-2013	1	15.111.01	9
9.	K6100003	XSTK	2011-2012	2	15.111.01	10
10.	K6100004	XSTK	2011-2012	2	15.111.01	4
11.	K6100004	DSTT	2012-2013	2	15.111.05	7
12.	K6100004	KT LT	2012-2013	2	15.111.02	5
13.	K6100012	THDC	2012-2013	1	15.111.10	6
14.	K6100013	THDC	2011-2012	1	15.111.09	9
15.	K6100013	KT LT	2012-2013	2	15.111.02	8
16.	K6100014	THDC	2012-2013	1	15.111.02	10
17.	K6100015	THDC	2012-2013	1	15.111.02	3
18.	K6100015	GTA1	2012-2013	1	15.111.01	6
19.	K6100015	DSTT	2011-2012	1	15.111.05	7
20.	K6100015	XSTK	2012-2013	1	15.111.01	8

IV.Thực hiện cấp phát quyền cho người dùng truy cập vào cơ sở dữ liệu trên

V.Tạo vai trò của người dùng trên SQL Server và cơ sở dữ liệu trên

VI.Viết View thực hiện các câu yêu cầu sau:

- 1) Hiển thị danh sách tất cả các sinh viên
- 2) Hiển thị danh sách tất cả các sinh viên (gồm: họ tên sinh viên, tên lớp, tên khoa)
- 3) Hiển thị danh sách các giảng viên (gồm họ tên giảng viên, tên khoa)
- 4) Hiển thị danh sách các giảng viên giảng dạy các môn học (Tên giảng viên, tên khoa, tên môn học)
- 5) Hiển thị danh sách các giảng viên không dạy môn học nào trong năm học 2011-2012

VII.Thực hiện cấp phát quyền cho người dùng trên các View trên.

VIII. Viết các thủ tục lưu trữ thực hiện các yêu cầu sau:

- 1) Hiển thị mã, tên các môn học được sinh viên **Nguyễn Thị Hoài Thu** đăng ký học
- 2) Đếm số môn học mà sinh viên **Trần Ngọc Thành** đã đăng ký học (*Kết quả truyền cho tham số đầu ra của thủ tục*)
- 3) Hiển thị danh sách các sinh viên đăng ký học môn Giải tích 1 trong học kỳ 1 (trong danh sách cần chứa các thông tin: Mã sinh viên, Họ tên, Tên Khoa).
- 4) Hiển thị danh sách sinh viên sắp xếp theo tên khoa, mỗi khoa lại sắp xếp theo họ tên. Danh sách chứa các thông tin sau: mã sinh viên, họ tên sinh viên, ngày sinh, tên khoa.

- 5) Hiển thị bảng thông kê sinh viên theo khoa. Bảng chứa thông tin Mã Khoa, Tên khoa, Số lượng sinh viên.
- 6) Hiển thị danh sách các thầy dạy trong từng kỳ. Trong danh sách cần có các thông tin: Họ tên thầy, Tên môn, Học kỳ, Số lượng sinh viên.
- 7) Hiển thị danh sách sinh viên đăng ký nhiều môn học nhất
- 8) Đếm số sinh viên đăng ký nhiều môn học nhất/học kỳ (truyền kết quả qua tham số của thủ tục)
- 9) Đếm số giảng viên giảng dạy nhiều môn nhất/ ít môn nhất (truyền kết quả qua tham số của thủ tục)
- 10) Hiển thị danh sách sinh viên đăng ký ít môn học nhất
- 11) Các môn học mà có số lượng sinh viên đăng ký trong một kỳ nhỏ hơn 2 sẽ không tổ chức lớp học. Hãy hiển thị danh sách các môn học bị hủy (trong danh sách có Tên môn, Học kỳ, Số lượng SV đăng ký).
- 12) Đưa ra họ tên, tên lớp và số điểm trung bình của sinh viên có điểm trung bình lớn nhất trong học từng học kỳ/năm học (*các tham số đầu vào, đầu ra được truyền qua tham số của thủ tục*)

IX. Thực hiện cấp phát quyền cho người dùng trên các Thủ tục lưu trữ trên.

X. Sử dụng ít nhất hai quyền khác nhau vừa cấp phát ở mục VI để truy cập vào các Thủ tục lưu trữ trên và so sánh quyền truy cập giữa các người dùng

XI. Thực hiện loại bỏ quyền cho người dùng truy cập vào cơ sở dữ liệu và Thủ tục lưu trữ trên.

XII. Thực hiện chế độ sao lưu và khôi phục dữ liệu