

MỤC LỤC

TỔNG QUAN MÔN HỌC	6
-------------------------	---

Phần lý thuyết

Chương 1 – Các biện pháp bảo vệ CSDL	8
1 Các biện pháp bảo vệ bằng máy tính.....	8
2 Cấp quyền.....	9
2.1 Lệnh SQL để cấp quyền:	9
2.2 Lệnh SQL để thu hồi quyền:.....	10
3 Khung nhìn.....	10
3.1 Lệnh dùng để tạo view:.....	10
3.2 Lệnh dùng để xóa view	11
3.3 Thuận lợi và bất lợi của việc sử dụng view	11
4 Sao lưu và phục hồi	12
5 Toàn vẹn dữ liệu.....	12
6 Mật hoá dữ liệu	13
7 RAID (Redundancy Array of Independent Disks).....	14
8 Các khái niệm và cấu trúc lưu trữ CSDL Oracle	16
8.1 Giới thiệu	16
8.2 Database và Instance.....	16
8.3 Cấu trúc lưu trữ của CSDL Oracle.....	17
8.4 Schema và schema objects.....	23
8.5 Data dictionary.....	23
9 Mở/tắt CSDL và thể hiện (startup/shutdown)	23
9.1 Giới thiệu	23
9.2 Mở CSDL.....	24
9.3 Tắt CSDL.....	25
10 Oracle Net	25
11 Các biện pháp bảo vệ CSDL Oracle	26
11.1 Quản lý người dùng	26
11.2 Quản lý quyền.....	29
11.3 Quản lý Role (vai trò)	33
11.4 Import và export.....	37
11.5 Oracle Data Pump	39
Chương 2 Ngôn ngữ SQL & PL/SQL trong Oracle.....	41
1 Bảng dữ liệu và các đối tượng liên quan.....	41
1.1 Quản lý bảng.....	41
1.2 Sequence	43
1.3 Index	44
1.4 Biểu thức chính quy (regular expression).....	45

2	Các kiểu dữ liệu trong Oracle	46
3	Cấu trúc chương trình PL/SQL	47
3.1	Cấu trúc lập trình trong Oracle	47
3.2	Các dạng chương trình PL/SQL.....	47
3.3	Cấu trúc khối PL/SQL vô danh.....	48
4	Các kiểu dữ liệu cơ bản của PL/SQL	49
5	Thuộc tính	50
6	Kiểu dữ liệu phức.....	50
6.1	Kiểu dữ liệu con do người dùng định nghĩa	50
6.2	Kiểu TABLE.....	51
7	Các loại mệnh đề.....	51
7.1	Mệnh đề gán.....	51
7.2	Mệnh đề lệnh (SQL command)	52
8	Các cấu trúc điều khiển	52
8.1	Cấu trúc rẽ nhánh.....	52
8.2	Cấu trúc lặp	54
8.3	Cấu trúc Ngoại lệ (Exception)	55
9	Kiểu con trỏ (Cursor) 57	57
9.1	Cú pháp khai báo con trỏ:	57
9.2	Cú pháp mở con trỏ:.....	58
9.3	Cú pháp lấy dữ liệu	58
9.4	Cú pháp đóng con trỏ	58
9.5	Một số thuộc tính của con trỏ:	59
9.6	Mệnh đề SELECT FOR UPDATE trong Cursor.....	59
9.7	Mệnh đề WHERE CURRENT OF trong Cursor	59
10	Thủ tục, hàm và trigger	60
10.1	Thủ tục	61
10.2	Hàm.....	62
10.3	Trigger	63
	Chương 3. Quản lý giao dịch & phục hồi	67
1	Giao dịch (transaction).....	67
1.1	Định nghĩa Giao dịch:.....	67
1.2	Trạng thái của Giao dịch:.....	68
1.3	Các thuộc tính của một GD.....	68
2	Điều khiển cạnh tranh (Concurrency control).....	70
2.1	Sự cần thiết phải có quản lý cạnh tranh	70
2.2	Lịch trình (schedule):.....	72
2.3	Tính khả tuần tự của một lịch trình.....	75
2.4	Tính khả phục hồi của lịch trình	75
2.5	Các kỹ thuật quản lý cạnh tranh.....	76
2.6	Các kỹ thuật lạc quan (optimistic techniques)	84
2.7	Độ mịn của mục dữ liệu (data granularity).....	85
3	Phục hồi CSDL	88

3.1	Sự cần thiết phải phục hồi dữ liệu	88
3.2	Các GD và sự phục hồi	88
3.3	Các tiện ích để phục hồi	89
4	Quản lý giao dịch trong Oracle:	93
4.1	Tổng quan	93
5	Sao lưu, phục hồi trong Oracle:	97
5.1	Sao lưu	98
5.2	Phục hồi (recovery).....	102

Phân thực hành

BÀI 1. LÀM QUEN VỚI ORACLE.....	105
1 Nội dung.....	105
2 Bài tập có hướng dẫn.....	105
3 Bài tập tự làm	115
BÀI 2. QUẢN LÝ BẢNG DỮ LIỆU.....	116
1 Nội dung.....	116
2 Bài tập có hướng dẫn.....	116
3 Bài tập tự làm	118
BÀI 3. LẬP TRÌNH PL/SQL.....	120
1 Nội dung.....	120
2 Bài tập có hướng dẫn.....	120
3 Bài tập tự làm	125
BÀI 4. HÀM – THỦ TỤC	127
1 Nội dung.....	127
2 Bài tập có hướng dẫn.....	127
3 Bài tập tự làm	128
BÀI 5. TRIGGER	130
1 Nội dung.....	130
2 Bài tập có hướng dẫn.....	130
3 Bài tập tự làm	132
BÀI 6. QUẢN LÝ GIAO DỊCH, SAO LƯU VÀ PHỤC HỒI	133
1 Nội dung.....	133
2 Bài tập có hướng dẫn.....	133
3 Bài tập tự làm	135
ÔN TẬP	136
1 Nội dung.....	136
2 Bài tập	136
Phụ lục.....	139
1 Các lỗi thường gặp	139
2 Các hàm xử lý dữ liệu	140

DANH MỤC HÌNH

Hình 1. Các mức RAID	15
Hình 2. Database Instance	17
Hình 3. Môi quan hệ giữa Data block, extend và Segment	20
Hình 4. Môi tương quan giữa Tablespace và Datafiles	21
Hình 5. Quan hệ giữa cấu trúc luận lý và vật lý trong CSDL	21
Hình 6. Cấp phát thêm vùng lưu trữ cho CSDL	22
Hình 7. Quy trình mở/tắt CSDL Oracle.....	24
Hình 8. Role trong database.....	34
Hình 9. Cấu trúc một khối PL/SQL	47
Hình 10. Hai dạng khối chương trình PL/SQL.....	48
Hình 11. Vị trí lưu trữ các chương trình trong schema	61
Hình 12. Các sự kiện của Trigger	64
Hình 13. Tổ chức lưu trữ trong quá trình thực hiện giao dịch cập nhật CSDL	67
Hình 14. Biểu đồ trạng thái tương ứng với một GD.....	68
Hình 15. Các thành phần liên quan việc quản lý GD trong HQTCSDL	70
Hình 16. Các lịch trình tuần tự	72
Hình 17. Đồ thị chờ có chu trình	82
Hình 18. Sự phân cấp của độ mịn.....	87
Hình 19. Sự cố xảy ra khi các GD đang thực hiện cạnh tranh	89
Hình 20. Các lựa chọn sao lưu toàn CSDL.....	99
Hình 21. Một chiến lược sao lưu tăng dần chênh lệch	101
Hình 22. Một chiến lược sao lưu tăng dần tích lũy	101
Hình 23. Nguyên lý sao lưu, hoàn lại và phục hồi CSDL	103
Hình 24. Oracle Net Manager.....	105
Hình 25. Đặt Net Service Name	106
Hình 26. Chọn giao thức mạng được sử dụng	106
Hình 27. Các thiết lập của giao thức.....	106
Hình 28. Thiết lập tên service.....	107
Hình 29. Kiểm tra kết nối	107
Hình 30. Thông báo kết nối thành công	107
Hình 31. Giao diện SQL Developer	108
Hình 32. Kết nối trực tiếp	108
Hình 33. Kết nối thông qua Oracle Net	109
Hình 34. Cửa sổ chính của Oracle SQL Developer.....	109
Hình 35. Lỗi đăng nhập do chưa có quyền kết nối đến CSDL.....	110
Hình 36. Dùng các biến	120
Hình 37. Cửa sổ nhập dữ liệu	120

DANH MỤC BẢNG

Bảng 1. Một số quyền hệ thống thông dụng.....	30
Bảng 2. Một số quyền trên đối tượng thông dụng.....	32
Bảng 3. Các roles được định nghĩa sẵn	36
Bảng 4. Thông tin về các roles	37
Bảng 5. Các tham số trong lệnh imp và exp.....	38
Bảng 6. Các tham số tương đương giữa imp/exp và Oracle Data Pump.....	40
Bảng 7. Các mẫu biểu thức chính quy	45
Bảng 8. Các kiểu dữ liệu trong Oracle	47
Bảng 9. Các ngoại lệ thường xảy ra.....	57
Bảng 10. Diễn giải các thuộc tính của con trỏ.....	59
Bảng 11. Lịch trình cạnh tranh	71
Bảng 12. Lịch trình gây mất dữ liệu cập nhật.....	73
Bảng 13. Lịch trình phụ thuộc vào GD không hoàn tất.....	74
Bảng 14. Lịch trình phân tích không nhất quán.	74
Bảng 15. Lịch trình không khả phục hồi	75
Bảng 16. Giao thức 2PL giải quyết vấn đề mất dữ liệu đã cập nhật.	79
Bảng 17. Giao thức 2PL giải quyết vấn đề phụ thuộc vào GD không hoàn tất....	79
Bảng 18. Giao thức 2PL giải quyết vấn đề phân tích không nhất quán.	80
Bảng 19. Lịch trình cuộn nhiều tầng (cascade rollback).	80
Bảng 20. Tình trạng khóa chết.....	81
Bảng 21. Sự tương thích của các khóa trong cơ chế khóa đa hạt.....	87
Bảng 22. Một đoạn của tập tin nhật ký.....	90
Bảng 23. Các mức cô lập và các hiện tượng đọc.....	94
Bảng 24. Bảng tương thích của các khoá	96

TỔNG QUAN MÔN HỌC

Mục đích yêu cầu

Môn Hệ quản trị CSDL là một môn học giúp sinh viên nắm được các kiến thức và các kỹ năng cơ bản để quản trị một CSDL quan hệ có phân quyền và bảo mật. Đồng thời, sinh viên có khả năng sử dụng ngôn ngữ lập trình để tương tác mạnh mẽ với CSDL thông qua việc thiết kế và sử dụng thủ tục, hàm và trigger. Đa số các hệ quản trị sử dụng phổ biến trên thị trường hiện nay đều hỗ trợ các tính năng trình bày trong giáo trình này. Để minh họa, phần mềm được chọn để thực hành là hệ quản trị CSDL Oracle 10g R2. Sinh viên sẽ thực hành trong 6 buổi (30 tiết) ứng với 6 bài thực hành với các chủ đề khác nhau.

Sau khi hoàn tất học phần này, sinh viên có thể:

- Kết nối, quản lý CSDL (tạo/xóa/khởi động/tắt CSDL) .
- Quản lý và cấp quyền cho người dùng và sao chép CSDL.
- Thực hiện các thao tác cơ bản trên CSDL (tạo bảng, cài đặt ràng buộc trên bảng; thêm, sửa, xóa dữ liệu).
- Lập trình tạo hàm, thủ tục, trigger với ngôn ngữ PL/SQL.
- Hiểu được sự điều khiển cạnh tranh giữa các giao dịch.
- Quản lý sao lưu, phục hồi.
- Hiểu và cài đặt các biện pháp bảo vệ hệ thống CSDL.

Nội dung cốt lõi

Tài liệu gồm 3 chương lý thuyết và 6 bài thực hành. Mỗi bài thực hành sẽ tập trung vào một số nội dung với các bài tập có hướng dẫn và bài tập tự làm.

Kiến thức tiên quyết

Như một môn bắt buộc, môn học này được đưa vào giảng dạy cho sinh viên chuyên ngành Công Nghệ Thông Tin với yêu cầu sinh viên đã hoàn thành học phần Hệ Cơ Sở Dữ Liệu (CT106).

Phương pháp học tập

Với mục tiêu nâng cao khả năng tự học tập và tự nghiên cứu của sinh viên, người soạn đã cố gắng lồng ghép vào nội dung các ví dụ minh họa đơn giản, cụ thể; đồng thời bố trí bố cục với mong muốn tạo sự dễ hiểu cho sinh viên và người đọc.

Để học tốt môn học này, trước hết sinh viên cần phải:

- Tham gia đầy đủ các buổi học lý thuyết trên lớp và các buổi thực hành trên phòng máy.
- Trước mỗi buổi thực hành, SV tự nghiên cứu và làm phần bài tập có hướng dẫn ở nhà.
- Trong mỗi buổi thực hành, sinh viên chỉ thực hiện phần bài tập không có hướng dẫn.
- Sinh viên cần lưu lại tất cả các câu lệnh đã làm ở mỗi bài để sử dụng cho các bài sau.

Thời lượng bố trí cho từng chương, từng bài

Chương I: Các biện pháp bảo vệ CSDL	6t – 2 buổi LT
Chương II: Ngôn ngữ SQL & PL/SQL	3t – 1 buổi LT
Chương III: Quản lý giao dịch, sao lưu và phục hồi	6t – 2 buổi LT
Phân thực hành:	
Bài 1: Làm quen với Oracle, quản lý người dùng, import/export	5t – 1 buổi TH
Bài 2: DDL & DML	5t – 1 buổi TH
Bài 3: Ngôn ngữ PL/SQL	5t – 1 buổi TH
Bài 4: Hàm, thủ tục	5t – 1 buổi TH
Bài 5: Trigger	5t – 1 buổi TH
Bài 6: Quản lý giao dịch & sao lưu, phục hồi CSDL	5t – 1 buổi TH

Một số quy ước

Để tạo sự dễ dàng trong việc theo dõi giáo trình, các phần quan trọng được viết theo các định dạng khác nhau và quy ước như sau:

- *Phần quan trọng*
- **CODE**

Các ký hiệu trình bày trong cú pháp của các câu lệnh có ý nghĩa như sau:

CHỮ HOA	từ khóa
chữ thường	từ do người dùng định nghĩa, cần thay thế khi viết lệnh cụ thể.
{A B}	mục này bắt buộc phải có, nhưng có thể chọn A hoặc B.
[A]	mục A này không bắt buộc.
A ...	mục A có thể xuất hiện nhiều lần.
::=	được thay bằng

Chương 1 – Các biện pháp bảo vệ CSDL

Trong chương này từ mục 1 đến mục 7 là lý thuyết tổng quát về các biện pháp bảo vệ CSDL bằng máy tính. Tùy theo từng HQTCSDDL mà mức độ và cách thức cài đặt các biện pháp này khác nhau. Từ mục 8 của chương trở về sau sử dụng HQTCSDDL Oracle để minh họa các biện pháp cụ thể này.

1 Các biện pháp bảo vệ bằng máy tính

An toàn trong CSDL (DB security) là sự bảo vệ CSDL khỏi những đe dọa có chủ ý hay vô tình thông qua các biện pháp có sử dụng máy tính hoặc không có 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.

Chúng ta xem xét an toàn CSDL trong các tình huống sau:

- CSDL bị đánh cắp hay gian lận
- CSDL mất đi tính bảo mật
- CSDL mất tính riêng tư
- CSDL mất tính toàn vẹn
- CSDL mất tính sẵn sàng

Nguy cơ (threat) là những tình huống hay sự kiện, có thể là cố ý hay vô tình, sẽ ảnh hưởng bất lợi đến một hệ thống và vì vậy ảnh hưởng đến cả tổ chức.

Nguy cơ có thể là hữu hình như mất mát về phần cứng, phần mềm, dữ liệu, hay vô hình như sự tin nhiệm của khách hàng.

Mỗi nguy cơ phải được xem như một sự vi phạm an ninh có thể xảy ra. Tổ chức phải nhận định được tất cả các nguy cơ tiềm ẩn, hay ít ra là các nguy cơ quan trọng, để từ đó đưa ra các kế hoạch phòng tránh cũng như biện pháp đối phó phù hợp. Khi xây dựng các biện pháp này cần lưu ý chi phí thực hiện và mức độ cản trở đối với người dùng.

Có rất nhiều biện pháp bảo vệ hệ thống khác nhau có thể chia thành 2 loại là các biện pháp không sử dụng máy tính và những biện pháp có sử dụng máy tính. Các biện pháp không sử dụng máy tính bao gồm các bố trí thiết bị một cách an toàn, quản trị nhân sự, chính sách an ninh cũng như kế hoạch đối phó với những bất ngờ,... Trong khuôn khổ của giáo trình này, ta chỉ quan tâm những biện pháp bảo vệ có sử dụng máy tính.

Các biện pháp bảo vệ bằng máy tính đề cập trong giáo trình này bao gồm:

- Cấp quyền (authorization)
- Khung nhìn (Views)
- Sao lưu và phục hồi (Backup and restore)
- Toàn vẹn dữ liệu (Integrity)
- Mật hóa (Encryption)
- Công nghệ RAID

2 Cấp quyền

Cấp quyền là sự gán quyền cho một người dùng hay chương trình để có thể truy cập vào một hệ thống hay một đối tượng của hệ thống.

Chứng thực người dùng (authentication) là cơ chế để xác định một người dùng là ai.

Cơ chế chứng thực người dùng phổ biến nhất là sử dụng định danh người dùng cùng với mật khẩu, mặc dù cách này không thể đảm bảo một cách hoàn toàn. Vì vậy, cần phải có những thủ tục đi kèm để nâng cao tính an toàn cho hệ thống như:

Mật khẩu phải được giữ kín và thay đổi định kỳ.

Định danh và mật khẩu lưu trong hệ thống phải ở dạng mã hóa.

Đưa ra các quy định về thành phần của mật khẩu.

Đề ra các thủ tục cần thiết và theo dõi sát việc cấp quyền cho người dùng.

Một số HQTCSDL sử dụng các định danh người dùng riêng, một số sử dụng các định danh người dùng của hệ điều hành bên dưới hoặc hỗ trợ cả hai.

Chủ sở hữu (ownership) của một đối tượng trong CSDL là người đã tạo ra đối tượng đó. Các đối tượng có sẵn trong HQTCSDL sẽ thuộc về bản thân HQTCSDL mà đại diện là nhà quản trị CSDL (DBA). Chủ sở hữu có mọi *quyền* (privileges) trên đối tượng của mình và có thể gán quyền trên đối tượng đó cho người khác.

HQTCSDL ghi nhận tất cả các quyền đã gán cho người dùng nào và gán bởi ai; để có thể duy trì đúng đắn tập quyền trên từng người dùng khi quyền được gỡ bỏ.

Các HQTCSDL có thể hỗ trợ nhiều mức cấp quyền khác nhau (mức người dùng - *user* hoặc nhóm người dùng - *role*). Định danh người dùng thường có độ ưu tiên cao hơn định danh nhóm.

Mỗi người dùng được tạo ra có thể được gán 2 loại quyền:

- Quyền đối tượng (Object Privileges). Các quyền này cho phép người dùng thực hiện các thao tác nào đó trên một đối tượng của CSDL; chẳng hạn quyền `SELECT`, `INSERT`, `UPDATE`,... dữ liệu trong một bảng nào đó.
- Quyền hệ thống (System Privileges): Các quyền này cho phép người dùng thực hiện thao tác nào đó trong CSDL nhưng không gắn với một đối tượng đã tồn tại nào; chẳng hạn quyền tạo bảng, tạo view,...

2.1 Lệnh SQL để cấp quyền:

Các quyền định nghĩa bởi chuẩn ISO: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `REFERENCES`, `USAGE`

Gán quyền trên đối tượng cho người dùng bằng lệnh:

```
GRANT {danh_sách_quyền | ALL PRIVILEGES}
ON    tên_đối_tượng
TO    {danh_sách_định_danh_được_cấp | PUBLIC}
[WITH GRANT OPTION]
```

Tùy chọn `WITH GRANT OPTION` cho phép người dùng được cấp tiếp tục cấp những quyền này cho người khác.

Lệnh gán quyền hệ thống thông thường cũng được các Hệ QTCSDL cài đặt tương tự nhưng bỏ đi mệnh đề `ON`.

2.2 Lệnh SQL để thu hồi quyền:

Lệnh REVOKE được dùng để gỡ bỏ quyền trên đối tượng khỏi người dùng đã được cấp:

```
REVOKE [GRANT OPTION FOR] {ds_quyền| ALL PRIVILEGES}
ON      đối_tượng
FROM    {danh_sách_định_danh_được_cấp | PUBLIC} [RESTRICT |
CASCADE]
```

RESTRICT: mặc định, sẽ ngưng thực thi lệnh Revoke nếu kết quả câu lệnh làm ảnh hưởng đến các đối tượng phụ thuộc.

CASCADE: Các đối tượng phụ thuộc sẽ bị xóa hoặc gỡ ra.

Lệnh thu hồi quyền hệ thống thông thường cũng được các Hệ QTCSDL cài đặt tương tự nhưng bỏ đi mệnh đề ON.

3 Khung nhìn

Khung nhìn là kết quả động của một hoặc nhiều thao tác quan hệ trên các quan hệ cơ sở (base table) để sinh ra một quan hệ khác. Một khung nhìn là một bảng ảo (virtual table), nó không thực sự tồn tại trong CSDL. Một người dùng có thể được gán các quyền thích hợp trên một khung nhìn định nghĩa trên nhiều bảng. Bằng cách này ta sẽ đảm bảo tính an toàn cho dữ liệu tốt hơn là gán quyền trên nhiều bảng cơ sở.

3.1 Lệnh dùng để tạo view:

```
CREATE VIEW  tên_view [(tên_cột [, ...])]
AS  câu_truy_vấn
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

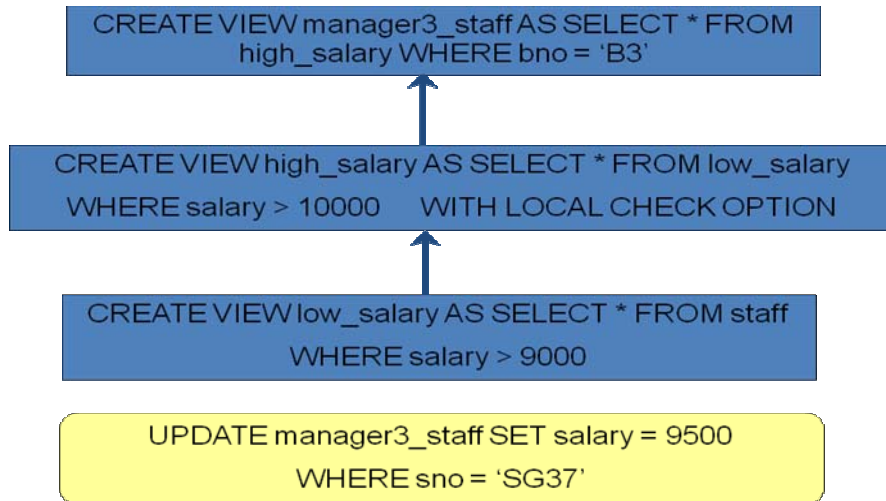
Các dòng đi vào và ra khỏi view được gọi là *các dòng di cư* (migrating rows).

WITH CHECK OPTION là tùy chọn để đảm bảo rằng các dòng không di cư ra khỏi view qua các thao tác insert và update trên view (nghĩa là, nếu một dòng không thỏa điều kiện WHERE trong câu truy vấn của view thì dòng đó sẽ không được cập nhật vào bảng cơ sở).

Nếu chọn CASCADE thì việc đảm bảo này cũng áp dụng trên các view định nghĩa trên view này (đây là chế độ mặc định).

Nếu chọn LOCAL thì việc đảm bảo này cũng áp dụng trên các view định nghĩa trên view này trừ khi dòng đó cũng di cư khỏi view hay table nền của view này.

Ví dụ minh họa sau đây cho thấy lệnh UPDATE sẽ không được thực thi vì nó làm cho dòng muốn cập nhật có sno = SG37 di cư ra khỏi view high_salary.



3.2 Lệnh dùng để xóa view

```
DROP VIEW tên_view [RESTRICT | CASCADE]
```

Nếu có các đối tượng khác tham chiếu view này thì trường hợp có:

- RESTRICT: lệnh này sẽ không được thực hiện
- CASCADE: lệnh này sẽ xóa luôn cả các đối tượng kia.

3.3 Thuận lợi và bất lợi của việc sử dụng view

Các thuận lợi khi sử dụng view là:

- *Độc lập dữ liệu*: Một view giúp thể hiện một bức tranh nhất quán, không đổi về cấu trúc của CSDL, thậm chí khi các bảng nguồn bên dưới có thay đổi (như thêm, bớt cột; mối quan hệ thay đổi; các bảng được tách ra, cấu trúc lại, đổi tên).
- *Tính mới nhất*: mọi sự thay đổi dữ liệu trong các bảng nền ngay lập tức được phản ánh trong view.
- *Nâng cao tính an toàn*: Nếu mỗi người dùng được cấp quyền truy cập vào CSDL chỉ thông qua một tập nhỏ các views chứa dữ liệu thích hợp thì sẽ giới hạn và quản lý sự truy cập của người dùng vào CSDL tốt hơn.
- *Giảm sự phức tạp*: Một view có thể đơn giản hoá câu truy vấn, thay vì phải truy vấn trên nhiều bảng thì trở thành truy vấn trên một bảng.
- *Thuận tiện*: Tạo sự đơn giản cho người dùng khi nhìn vào CSDL chỉ thấy những gì họ cần.
- *Khả năng tùy biến*: view cung cấp cơ chế để tùy chỉnh diện mạo của CSDL, vì vậy có thể tạo nhiều cách nhìn khác nhau vào cùng một CSDL.
- *Toàn vẹn dữ liệu*: Nếu mệnh đề WITH CHECK OPTION được dùng khi tạo view, thì SQL đảm bảo không có dòng nào không thoả mệnh đề WHERE trong định nghĩa view mà được cập nhật thông qua view đó.

Bất lợi:

- *Hạn chế cập nhật*: mọi cập nhật trong một bảng nền phải được phản ánh ngay tức thì trong tất cả các view có tham chiếu đến bảng này. Tương tự như vậy, nếu ta cập nhật dữ liệu thông qua view thì các bảng nền cũng thay đổi. Tuy nhiên, có một số giới hạn như sau:

Chỉ cho phép cập nhật trên các view định nghĩa trên 1 bảng và có chứa các thuộc tính khoá chính hoặc khoá ứng viên (khóa duy nhất).

Không cho phép cập nhật trên view liên quan đến nhiều bảng. Tuy nhiên, điều này có thể khắc phục bằng cách sử dụng instead of trigger.

Không cho phép cập nhật trên view có kết tập hay nhóm dữ liệu.

- *Hạn chế cấu trúc*: cấu trúc của view được xác định ở thời điểm tạo ra nó. Nếu trong định nghĩa, ta dùng SELECT * FROM... thì view sẽ lấy tất cả các cột. Nhưng nếu sau đó bảng nền có thêm cột mới thì cột mới này sẽ không có mặt trong view trừ khi ta xoá view và tạo lại nó.
- *Hiệu quả hoạt động*: Khi sử dụng view ta phải mất một ít thời gian để thực thi câu lệnh select trong view. Trong một số trường hợp, thời gian này không đáng kể; trong những trường hợp khác nó có thể là một vấn đề.

4 Sao lưu và phục hồi

Sao lưu là quá trình sao chép CSDL và các tập tin nhật ký (có thể kể cả chương trình) vào các thiết bị lưu trữ dự phòng một cách định kỳ. Đây là công việc của các nhà quản trị CSDL và cần phải thực hiện một cách có tổ chức, các qui định, bước thực hiện sao lưu phải được mô tả rõ ràng. Sao lưu là một cách bảo vệ dữ liệu chống lại các lỗi ứng dụng và sự mất mát dữ liệu không mong đợi. Nếu dữ liệu gốc bị mất, chúng có thể được xây dựng lại từ bản sao lưu. Thông tin về phiên bản sao lưu phải chính xác. Bản sao lưu phải được lưu trữ nhiều nơi. Nơi lưu trữ các bản sao lưu phải được đảm bảo an toàn.

- Các HQTCSDL ngày nay thường hỗ trợ chức năng tự động thực hiện sao lưu theo lịch trình định trước của nhà quản trị. Ta có thể lựa chọn thực hiện sao lưu toàn bộ CSDL (complete backup) hoặc chỉ sao lưu các thay đổi mới diễn ra (incremental backup) kể từ lần sao lưu trước. Thông thường, các bản sao lưu này được lưu trong bộ lưu trữ offline như băng từ.

Nếu có sự cố làm cho CSDL bị hỏng, các bản sao chép dự phòng và các chi tiết lưu trong các tập tin nhật ký sẽ được sử dụng để phục hồi lại hệ thống về tình trạng ổn định gần nhất có thể. Quá trình phục hồi này cũng cần phải được mô tả rõ ràng và phải tổ chức kiểm thử thường xuyên để đảm bảo hệ thống có thể phục hồi nếu có sự cố.

Điểm kiểm tra (checkpoint): Là điểm diễn ra sự đồng bộ giữa CSDL và tập tin nhật ký giao dịch. Tại thời điểm này, tất cả các vùng đệm sẽ được ghi ra bộ lưu trữ thứ cấp và một mẫu tin kiểm tra sẽ được ghi vào trong nhật ký.

Cơ chế này giúp cho các cập nhật đang diễn ra trên CSDL được ghi ra ngoài đĩa.

Các kỹ thuật này sẽ được thảo luận chi tiết hơn trong chương quản lý giao dịch.

5 Toàn vẹn dữ liệu

Việc quản lý toàn vẹn cũng góp phần duy trì một hệ thống CSDL an toàn, bảo vệ dữ liệu luôn luôn hợp lệ, không bị sai.

RBTV thường được chia làm 4 loại và các HQTCSL hiện nay hỗ trợ nhiều công cụ cho phép cài đặt hầu hết các ràng buộc này trong CSDL thay vì phải lập trình trong ứng dụng như trước kia:

- Ràng buộc về khóa hay ràng buộc thực thể: PRIMARY KEY, UNIQUE KEY
- Ràng buộc về miền trị: CHECK CONSTRAINT, DEFAULT VALUE
- Ràng buộc về tham chiếu: FOREIGN KEY
- Ràng buộc khác: PROCEDURE, FUNCTION, TRIGGER

Các công cụ để cài đặt ba loại ràng buộc đầu đã được học trong học phần Hệ CSDL. Trong học phần này sẽ tiếp tục học các công cụ để cài đặt loại ràng buộc còn lại.

6 Mật hoá dữ liệu

Nếu hệ CSDL lưu giữ các dữ liệu quan trọng thì cần lưu ý mã hoá dữ liệu để tránh các nguy cơ từ bên ngoài hay truy cập trái phép có thể xảy ra. Một số hệ QTCSDL cung cấp tiện ích mã hóa nhằm mục đích này.

Mật hóa là sự mã hóa dữ liệu bằng một giải thuật đặc biệt làm cho dữ liệu không thể đọc được nếu không có khóa giải mã (decryption key).

Có nhiều kỹ thuật mã hóa dữ liệu, có thể chia thành 2 loại:

- Khả đảo (reversible): Có thể giải mã dữ liệu về như ban đầu.
- Bất khả đảo (irreversible): không cho phép giải mã dữ liệu về như ban đầu.

Một hệ thống mật hóa khả đảo bao gồm: khóa để mã, giải thuật mã hóa, khóa để giải mã và giải thuật giải mã.

Có hai loại hệ thống mật hóa:

- *Đối xứng (symmetric)*: Hệ thống đối xứng sử dụng cùng 1 khóa để mã và giải mã chẳng hạn như DES (Data Encryption Standard) do IBM phát triển. Giải thuật chuyển mỗi khối văn bản 64-bit sử dụng khoá 56-bit. DES được cho là chưa an toàn lắm, cần phải có khoá dài hơn nữa. Hệ mã hoá PGP (Pretty Good Privacy) vì vậy sử dụng khoá 128-bit.
- *Bất đối xứng (asymmetric)*: sử dụng các khóa khác nhau để mã và giải mã. Điển hình là hệ thống mật hóa khóa công cộng – public key cryptosystems như RSA, mỗi đối tượng sẽ có 2 khoá: khoá riêng (private key) và khoá công cộng (public key). Người A muốn gửi thông điệp đến người B sẽ dùng khoá công cộng của B để mã hoá, khi đó chỉ có B mới có thể đọc được thông điệp vì B mới có khoá riêng tương ứng để giải mã. Hệ mã hoá này cũng được dùng để gửi ‘chữ ký điện tử’ kèm theo thông điệp để chứng tỏ thông điệp đến từ người đã ‘ký’ lên thông điệp.

Thông thường, giải thuật đối xứng chạy nhanh hơn. Nhưng trong thực tế, ta thường dùng phối hợp cả hai loại. Hệ thống dùng khoá công cộng để mã hoá một khoá sinh ra ngẫu nhiên, và khoá ngẫu nhiên này được giải thuật đối xứng dùng để mã hoá thông điệp.

7 RAID (Redundancy Array of Independent Disks)

Một trong các phần cứng có nguy cơ bị hư và ảnh hưởng nghiêm trọng đến hệ CSDL đó là các ổ đĩa cứng. Để đảm bảo cho hệ thống vẫn có thể hoạt động cho dù sự cố có xảy ra, một giải pháp đã ra đời, đó là công nghệ RAID. RAID hoạt động bằng cách sắp xếp một dãy các đĩa độc lập để cải tiến độ tin cậy và tăng hiệu suất hoạt động cho hệ thống.

Hiệu suất được tăng lên nhờ kỹ thuật ‘tháo rời’ dữ liệu (data stripping): dữ liệu được chia thành các phần bằng nhau và được phân phối cho nhiều đĩa một cách trong suốt. Điều này tạo cho ta có vẻ như có một đĩa lớn, với tốc độ nhanh từ các đĩa nhỏ hơn. Kỹ thuật này cải tiến hiệu suất vào/ra đĩa bằng cách cho phép nhiều dịch vụ vào/ra thực hiện song song. Đồng thời cũng cân bằng tải giữa các đĩa.

Độ tin cậy được cải tiến nhờ việc lưu trữ thông tin lặp lại trên nhiều đĩa sử dụng cơ chế chẵn lẻ (parity) hay cơ chế sửa lỗi (error-correcting). Mỗi byte được lưu kèm thêm 1 bit cho biết số bit 1 trong byte là chẵn hay lẻ. Nếu có bit nào bị lỗi thì giá trị của bit chẵn lẻ sẽ không còn trùng khớp với số bit 1 trong byte. Cơ chế sửa lỗi thì cần thêm vài bit nữa cho mỗi byte để có thể phục hồi lại bản gốc khi có 1 bit bị hư.

Có nhiều mức RAID ứng với nhiều mức cấu hình đĩa khác nhau:

RAID 0 – Không dư thừa (Nonredundant): không lưu dữ liệu lặp lại do đó tạo hiệu suất ghi tốt nhất vì không cần phải lặp lại việc cập nhật. Việc chia dữ liệu được thực hiện ở mức khối.

RAID 1 – phản chiếu (mirror): Mức này duy trì hai phiên bản dữ liệu giống hệt nhau trên các đĩa khác nhau. Để duy trì tính nhất quán khi có lỗi xảy ra, việc ghi có thể không được thực hiện đồng thời. Đây là giải pháp lưu trữ đắt tiền nhất.

RAID 0 + 1 – Giải pháp này kết hợp cả phân chia và phản chiếu.

RAID 2 – Memory-style error-correcting codes (MSECC): Với mức này, đơn vị phân chia là bit và mã Hamming được sử dụng như là cơ chế dư thừa để có thể sửa lỗi.

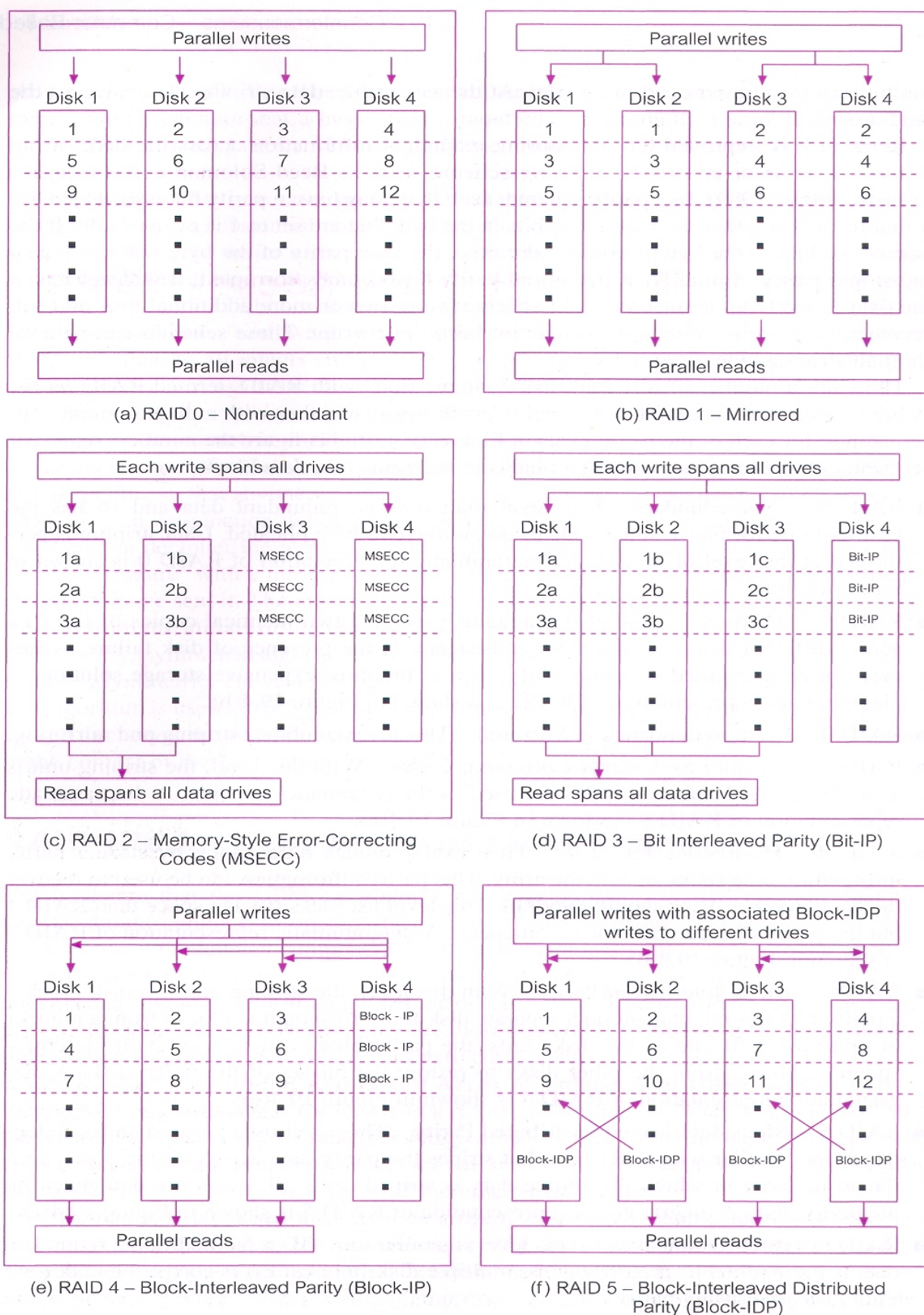
RAID 3 – Bit-interleaved Parity: Mức này cung cấp dư thừa bằng cách lưu trữ thông tin chẵn lẻ trên một đĩa trong dãy đĩa. Thông tin này có thể được sử dụng để phục hồi dữ liệu trên các đĩa kia nếu chúng bị hư. Mức này sử dụng ít không gian lưu trữ hơn RAID 1 nhưng đĩa chẵn lẻ có thể trở thành một nút ‘cổ chai’.

RAID 4 – Block-interleaved Parity: Với mức này, đơn vị chia dữ liệu là 1 khối đĩa – Khối chẵn lẻ được duy trì trên một đĩa riêng biệt cho các khối tương ứng từ một số đĩa khác. Nếu một trong các đĩa bị lỗi, khối chẵn lẻ được dùng kết hợp với các khối tương ứng từ các đĩa khác để phục hồi lại các khối ở đĩa hư.

RAID 5 – Block-interleaved Distributed Parity: Mức này sử dụng dữ liệu chẵn lẻ tương tự như mức 3 nhưng chia dữ liệu chẵn lẻ được chia ra lưu trên nhiều đĩa, tương tự như cách chia dữ liệu nguồn. Điều này tránh tình trạng nút cổ chai ở đĩa chẵn lẻ như mức 3.

RAID 6 – P+Q Redundancy: Mức này tương tự như mức 5 nhưng có thêm dữ liệu dư thừa để bảo vệ hệ thống khi có nhiều lỗi xuất hiện trên đĩa. Khi đó mã sửa lỗi sẽ được dùng thay cho mã chẵn lẻ.

Việc chọn lựa mức RAID nào để cài đặt cho hệ thống phụ thuộc vào nhiều yếu tố như bản chất dữ liệu, chi phí, mức độ quan trọng của dữ liệu, v.v....



Hình 1. Các mức RAID. Các con số biểu thị cho các khối dữ liệu tuần tự và các ký tự chỉ các phân đoạn của một khối dữ liệu.

8 Các khái niệm và cấu trúc lưu trữ CSDL Oracle

8.1 Giới thiệu

Oracle Database là phần mềm HQTCSDDL được cung cấp bởi công ty Oracle. Nhìn chung, các HQTCSDDL đều có khả năng quản lý một cách tin cậy số lượng lớn dữ liệu trong môi trường đa người dùng, sao cho nhiều người dùng có thể truy cập đồng thời cùng một dữ liệu mà vẫn giữ được tính nhất quán của dữ liệu đó. HQTCSDDL cũng cần được bảo vệ chống lại các xâm nhập trái phép và có khả năng phục hồi dữ liệu khi có lỗi xảy ra.

Bản phát hành gần đây nhất của Oracle Database là 11g, bản này gồm các phiên bản sau:

- Standard Edition One (SE1) cung cấp các tính năng dễ sử dụng, mạnh mẽ cho các nhóm làm việc, các đơn vị phòng ban và các ứng dụng web.
- Standard Edition (SE) được thiết kế cho các máy chủ đơn hoặc phân tán.
- Enterprise Edition (EE) cung cấp quản lý dữ liệu một cách hiệu quả, tin cậy và bảo mật cho các ứng dụng xử lý giao dịch trực tuyến (OLTP), các ứng dụng quản lý nội dung và web 2.0.

Cả 3 phiên của Oracle Database 11g đều được xây dựng dựa trên kiến trúc chung và hoàn toàn tương thích với nhau. Các phiên bản này đều có khả năng triển khai trên các hệ điều hành khác nhau (Windows, Linux, Unix) và kèm theo rất nhiều công cụ hỗ trợ phát triển ứng dụng cũng như các thư viện lập trình.

Chữ “g” trong Oracle Database 11g là viết tắt cụm từ **Grid Computing** (tính toán lưới – GC). Đây là kiến trúc mới của ngành công nghệ thông tin trong thiết kế và cài đặt các hệ thống thông tin. Với GC, nhóm các phần cứng độc lập cùng với các thành phần phần mềm được kết nối nhau và được phân phối theo yêu cầu thay đổi của tải công việc (workloads). So với các kiến trúc trước đây, GC cung cấp dịch vụ có chất lượng cao hơn và chi phí thấp hơn.

- Dịch vụ có chất lượng cao vì GC không có sự cố do một thành phần đơn lẻ gây ra và có một kiến trúc bảo mật mạnh mẽ.
- Chi phí thấp do GC gia tăng tối đa khả năng sử dụng các tài nguyên hệ thống. Các phần cứng hoặc thành phần phần mềm “rảnh rỗi” không còn tận hiến cho một tác vụ chuyên biệt mà sẽ được cấp phát cho các tác vụ khác có nhu cầu sử dụng.

Vì học phần này được xem như là phần nhập môn, cung cấp cho sinh viên các kiến thức cơ bản về Hệ quản trị CSDL nói chung và Hệ quản trị Oracle nói riêng, chúng tôi thiết kế các bài thực hành trên phiên bản Oracle 10g R2 với đầy đủ các tính năng cơ bản và đòi hỏi cấu hình máy thực hành nhẹ nhàng hơn so với các phiên bản Oracle 11g.

8.2 Database và Instance

Các HQTCSDDL đều dùng cả bộ nhớ máy tính và các thiết bị lưu trữ như ổ cứng để hoạt động. Các ổ cứng cung cấp khả năng lưu trữ lâu dài và một không gian rộng lớn đủ chứa hàng triệu mẫu tin có thể lên đến hàng gigabyte. Tuy nhiên, truy cập dữ liệu từ ổ cứng chậm hơn nhiều so với truy cập từ bộ nhớ. Vì thế các hệ CSDL đều sử dụng bộ nhớ vào việc nạp trước dữ liệu nhằm tăng tốc độ truy vấn.

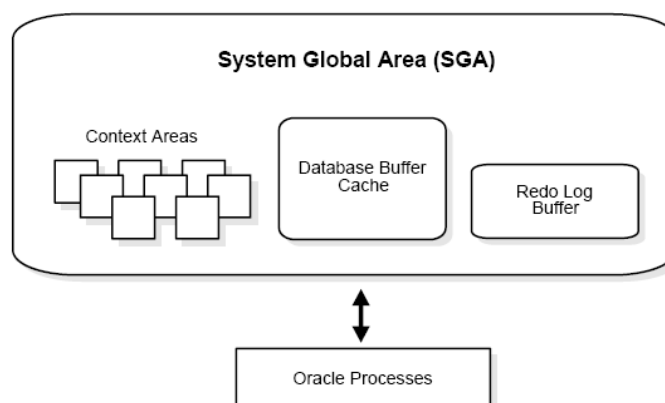
Trong Oracle, một CSDL (**database**) là một tập hợp các tập tin hệ thống lưu trữ dữ liệu do người dùng hoặc chương trình đưa vào và thông tin về cấu trúc của CSDL (**metadata**). Để có thể truy vấn và cập nhật CSDL, Oracle phải khởi động một số tiến trình nền và cấp phát một vài vùng nhớ sử dụng trong suốt quá trình thao tác trên CSDL.

Khi một CSDL được khởi động (start), một SGA (**System Global Area**) được cấp phát và các tiến trình nền (Oracle background processes) được khởi động. Sự kết hợp giữa SGA và các tiến trình nền được gọi là *thể hiện CSDL* (Database Instance hoặc Oracle Instance).

SGA là một vùng nhớ chia sẻ dùng để lưu trữ dữ liệu và các thông tin điều khiển của một thể hiện (instance). Khi kết nối đến server, người dùng được chia sẻ các dữ liệu có trong SGA. Vùng nhớ này sẽ được giải phóng khi thể hiện được tắt (shutdown) và mỗi thể hiện có một SGA riêng biệt.

Một tiến trình là một cơ chế hoạt động trong hệ điều hành, nó thực hiện một dãy các bước liên tiếp nhau và có một vùng nhớ riêng. Các tiến trình nền trong Oracle được chia thành 2 dạng: tiến trình người dùng (user process) và tiến trình máy chủ (server process). Tiến trình người dùng được tạo ra và duy trì nhằm thực thi các mã lệnh của các trình ứng dụng (như Oracle Forms, ...) hay các công cụ Oracle (như Oracle Enterprise Manager, Database Configuration Assistance, ...). Tiến trình máy chủ có nhiệm vụ điều khiển các yêu cầu từ các tiến trình người dùng có kết nối. Khi tiến trình máy chủ khởi động, Oracle cũng cấp phát một vùng đệm bộ nhớ PGA (Program Global Area) để chứa đựng dữ liệu và thông tin điều khiển cho mỗi tiến trình này.

Nhìn chung, mỗi thể hiện có một tập hợp các tiến trình duy trì và thúc ép mối quan hệ giữa cấu trúc vật lý của CSDL và cấu trúc bộ nhớ. Số lượng các tiến trình phụ thuộc vào cấu hình của mỗi thể hiện CSDL. Trong một server, nhiều CSDL có thể tồn tại song song. Vì vậy, để không bị lẫn lộn giữa các CSDL khác nhau, mỗi thể hiện CSDL được nhận dạng bằng một SID riêng biệt (System Identifier). Một CSDL có thể được mở (open hay mount) bởi nhiều hơn một thể hiện, nhưng một thể hiện chỉ có thể mở nhiều nhất một CSDL mà thôi.



Hình 2. Database Instance

8.3 Cấu trúc lưu trữ của CSDL Oracle

Trong hệ quản trị Oracle, một CSDL được cấu thành bởi 2 cấu trúc: *luận lý* và *vật lý*. Cấu trúc luận lý mô tả các vùng nhớ dùng để lưu trữ các đối tượng như các

bảng, các hàm... Ngược lại, cấu trúc vật lý được xác định bởi các tập tin hệ thống hình thành nên CSDL.

8.3.1 Cấu trúc vật lý (physical database structure)

Cấu trúc vật lý liên quan đến các tập tin được lưu trữ trên thiết bị phần cứng, chúng liên quan đến vấn đề lưu trữ dữ liệu, các tham số, nhật ký, lưu vết, sao lưu... để đảm bảo hệ thống hoạt động ổn định.

8.3.1.1 Datafiles

Oracle lưu trữ dữ liệu trong các datafiles (có phần mở rộng *.dbf) tương ứng với tablespace. Bản chất của một CSDL là một tập hợp các datafiles được lưu trữ trên các thiết bị khác nhau như đĩa từ, đĩa quang học... Datafile có các tính chất sau:

- Mỗi datafile chỉ liên kết với một CSDL.
- Datafile có một số đặc tính để có thể mở rộng tự động khi độ lớn của CSDL vượt quá giới hạn.
- Một hoặc nhiều datafiles tạo thành một đơn vị lưu trữ luận lý được gọi là tablespace.

Dữ liệu trong datafile được đọc trong lúc thực thi của CSDL và được lưu trữ trong bộ nhớ đệm của phần mềm Oracle nhằm tăng tốc độ truy cập. Dữ liệu được thêm mới hoặc sửa đổi không nhất thiết được ghi ngay lập tức lên datafile. Để giảm số lượng truy cập thiết bị lưu trữ và tăng hiệu quả thực thi, các dữ liệu này được lưu tạm trong bộ nhớ và được ghi đồng loạt lên các datafile vào một thời điểm định sẵn. Nhiệm vụ này được đảm trách bởi tiến trình nền *Database Background Writer* (DBWn).

8.3.1.2 Control Files

Các Control Files có phần mở rộng *.ctl. Mỗi thể hiện CSDL có ít nhất 1 tập tin dạng này. Nó rất cần thiết cho hoạt động bình thường của CSDL. Các tập tin này được cập nhật thường xuyên trong quá trình hoạt động của CSDL, vì vậy nó luôn được đặt vào chế độ ghi khi CSDL được mở. Vì lý do nào đó các tập tin không truy cập được, CSDL sẽ không hoạt động đúng đắn. Các Control Files ghi nhận các thông tin của CSDL như tên CSDL, các thông tin về tablespace và các redo-log file... Control File còn được dùng trong thao tác phục hồi dữ liệu.

8.3.1.3 Redo Log Files

Các tập tin Redo Log có phần mở rộng *.rdo hoặc *.log. Mỗi thể hiện CSDL duy trì một tập hợp các tập tin dạng này để ghi lại toàn bộ các giao dịch (transaction) và các thay đổi trên datafiles. Khi CSDL bị lỗi, các tập tin này được dùng để phục hồi lại dữ liệu đến thời điểm mẫu tin redo cuối cùng được ghi.

Mỗi CSDL duy trì ít nhất 2 nhóm tập tin Redo Log và mỗi tập tin chỉ thuộc vào một nhóm. Tại mỗi thời điểm, chỉ một nhóm tập tin ở trạng thái “hiện thời” để ghi nhận các mẫu tin redo. Khi dung lượng vượt quá giới hạn, nhóm này sẽ chuyển trạng thái “hiện thời” sang nhóm khác.

8.3.1.4 Archived Redo Log Files

Vì Oracle sử dụng xoay vòng các nhóm tập tin redo log, nên có khả năng một nhóm tập tin đã có chứa các mẫu tin redo bị ghi đè và như vậy thông tin redo bị thất

lạc. Vì vậy, Oracle phải đảm bảo các tập tin này không bị ghi đè nếu chưa được lưu ra tập tin Archived Redo Log.

Khi một thể hiện CSDL được thực thi dưới chế độ ARCHIVELOG, tiến trình ARCH của Oracle sẽ dùng các tập tin dạng này lưu lại các mẫu tin redo trên nhóm tập tin redo log.

8.3.1.5 Parameter Files

Tập tin này lưu trữ các thông số cấu hình cho thể hiện và CSDL như: độ lớn của datablock, đường dẫn đến nơi lưu trữ datafiles, độ lớn của vùng nhớ SGA, số sessions, số processes... Ngoài ra, CSDL Oracle còn có Server Parameter File (SPFILE) lưu trữ các thông số khởi động của CSDL, nhà quản trị có thể thay đổi các thông số trực tiếp trên tập tin này theo nhu cầu hệ thống.

8.3.1.6 Alert và Trace Log Files

Mỗi tiến trình nền và máy chủ đều có thể lưu vết hoạt động trên một Trace Log File tương ứng. Mỗi khi có một lỗi nội tại trong tiến trình, thông tin về lỗi đó được ghi ra tập tin. Phần lớn thông tin ghi ra Trace Log File dành cho nhà quản trị CSDL, các thông tin khác phục vụ cho dịch vụ hỗ trợ của Oracle (Oracle Support Service). Các thông tin này còn được dùng để điều chỉnh các ứng dụng và thể hiện CSDL.

8.3.1.7 Backup Files

Đây là các tập tin có định dạng đặc biệt được định nghĩa bởi Oracle. Các tập tin này phục vụ cho quá trình khôi phục CSDL mỗi khi có sự cố về thiết bị lưu trữ hay lỗi do người dùng gây ra.

8.3.2 Cấu trúc luận lý (logical database structure)

Cấu trúc luận lý bao gồm data block, extend, segment và tablespace; mô tả các vùng nhớ lưu trữ dữ liệu và cho phép Oracle điều khiển việc sử dụng đĩa cứng đến mức nhỏ nhất.

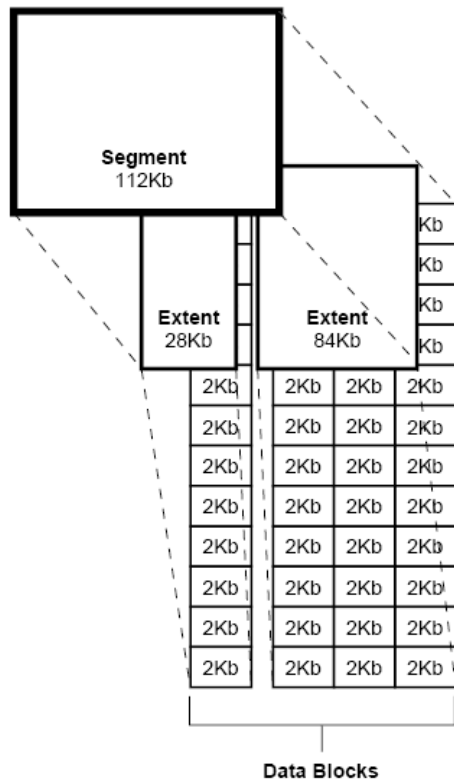
8.3.2.1 Data blocks

Đơn vị lưu trữ nhỏ nhất được dùng trong CSDL, được xác định bởi một số bytes nhất định trong vùng nhớ vật lý của CSDL. Độ lớn của data block được xác định khi CSDL được tạo ra và được định nghĩa bởi tham số khởi tạo DB_BLOCK_SIZE.

8.3.2.2 Extends

Đơn vị lưu trữ luận lý nhỏ nhất được cấp phát cho một đối tượng CSDL, bao gồm một dãy liên tục các khối dữ liệu (data blocks) được cấp phát để lưu trữ một kiểu thông tin nhất định.

8.3.2.3 Segments



Hình 3. Mối quan hệ giữa Data block, extent và Segment

Tập hợp các extent được cấp phát để lưu trữ các cấu trúc dữ liệu nhất định và các extent này được lưu trữ trong cùng một tablespace. Ví dụ, dữ liệu của một bảng được lưu trong một table segment, các chỉ mục được lưu trong các index segment... Khi các extent cấp phát cho segment đầy dữ liệu, Oracle sẽ cấp phát thêm extent cho segment đó. Vì các extent được cấp phát khi cần thiết nên các extent trong cùng một segment có thể không liên tục trong bộ nhớ. Một số loại segments được định nghĩa trong Oracle:

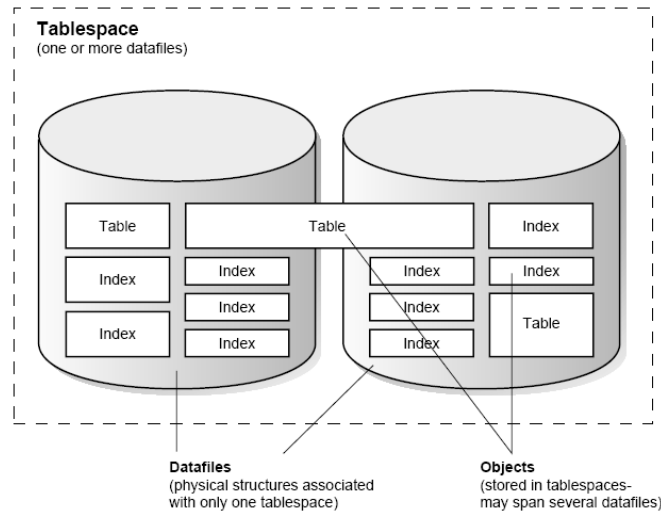
- Data segment: dữ liệu của các bảng được lưu trong các extends của data segment.
- Index segment: lưu trữ các thông tin về chỉ mục.
- Temporary segment: được tự động tạo ra bởi Oracle mỗi khi câu lệnh SQL cần một

vùng lưu trữ dữ liệu tạm thời để hoàn tất thực thi. Khi câu lệnh SQL kết thúc hoạt động, vùng nhớ này được giải phóng và trả lại hệ thống cho các yêu cầu khác.

- Rollback segment: không chứa các đối tượng CSDL, mà chứa “hình ảnh trước” (before image) của dữ liệu được thay đổi trong khi một giao tác chưa hoàn thành. Các thay đổi trên dữ liệu có thể được cuộn lại bằng cách sử dụng segment này.

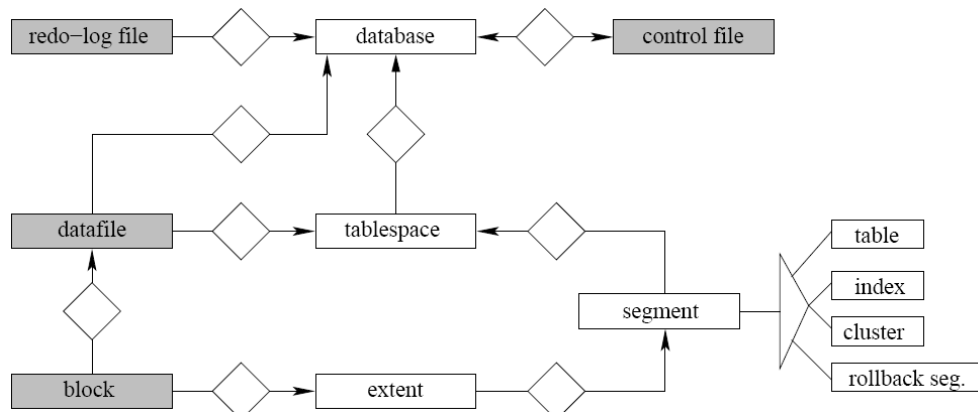
8.3.2.4 Tablespace

Một CSDL được chia thành các phân vùng luận lý được gọi là tablespaces. Tất cả các đối tượng của CSDL đều được lưu trữ trong các tablespace. Về mặt vật lý, mỗi tablespace bao gồm một hoặc nhiều datafiles. Từ phiên bản 10g trở đi, một CSDL Oracle có ít nhất một tablespace mang tên SYSTEM chứa từ điển dữ liệu (data dictionary) và một mang tên SYSAUX nhằm chia tải với SYSTEM. Các tablespace khác được tạo ra nhằm phục vụ cho các ứng dụng và các tác vụ khác nhau.



Hình 4. Mối tương quan giữa Tablespace và Datafiles

Các tablespace có thể đặt ở chế độ smallfile hoặc bigfile khi được tạo ra. Smallfile là chế độ truyền thống, tablespace có thể đến 1022 datafiles. Bigfile là chế độ có từ phiên bản 10g, tablespace chỉ chứa 1 datafile nhưng có độ lớn đến 2^{32} data blocks, tùy theo độ lớn của data block, bigfile tablespace có độ lớn đến 8 exabytes (1 exabyte = 10^{18} bytes). Chế độ này cho phép Oracle sử dụng khả năng của hệ thống 64-bit để tạo và quản lý các tập tin có dung lượng cực lớn.



Hình 5. Quan hệ giữa cấu trúc luận lý và vật lý trong CSDL

Tablespace có thể đặt ở chế độ online (truy cập được) hoặc offline (không truy cập được). Tablespace thường được đặt ở chế độ online để người dùng có thể truy cập dữ liệu. Tuy nhiên, đôi khi nhà quản trị cần đặt một tablespace ở chế độ offline nhằm mục đích bảo trì, người dùng chỉ không truy cập được tablespace này nhưng vẫn truy cập được các tablespaces khác trong CSDL.

Oracle định nghĩa 3 loại tablespaces khác nhau:

- Tablespaces cố định (permanent tablespace) chứa các đối tượng CSDL tồn tại lâu dài (persistent schema objects), các đối tượng này được lưu trong datafile.
- Undo tablespace là một kiểu không gian dữ liệu cố định được dùng để quản lý việc hoàn tác dữ liệu (undo data) nếu CSDL được đặt dưới chế độ quản lý hoàn tác tự động (automatic undo management). Oracle khuyến cáo sử dụng undo tablespace thay vì sử dụng rollback segment cho việc hoàn tác dữ liệu.
- Tablespaces tạm thời (temporary tablespace) chứa các đối tượng CSDL trong một phiên làm việc. Tablespace dạng này được dùng để quản lý vùng nhớ cho các thao tác sắp xếp (sort operations) trên CSDL. Ví dụ, nếu ta kết nối hai

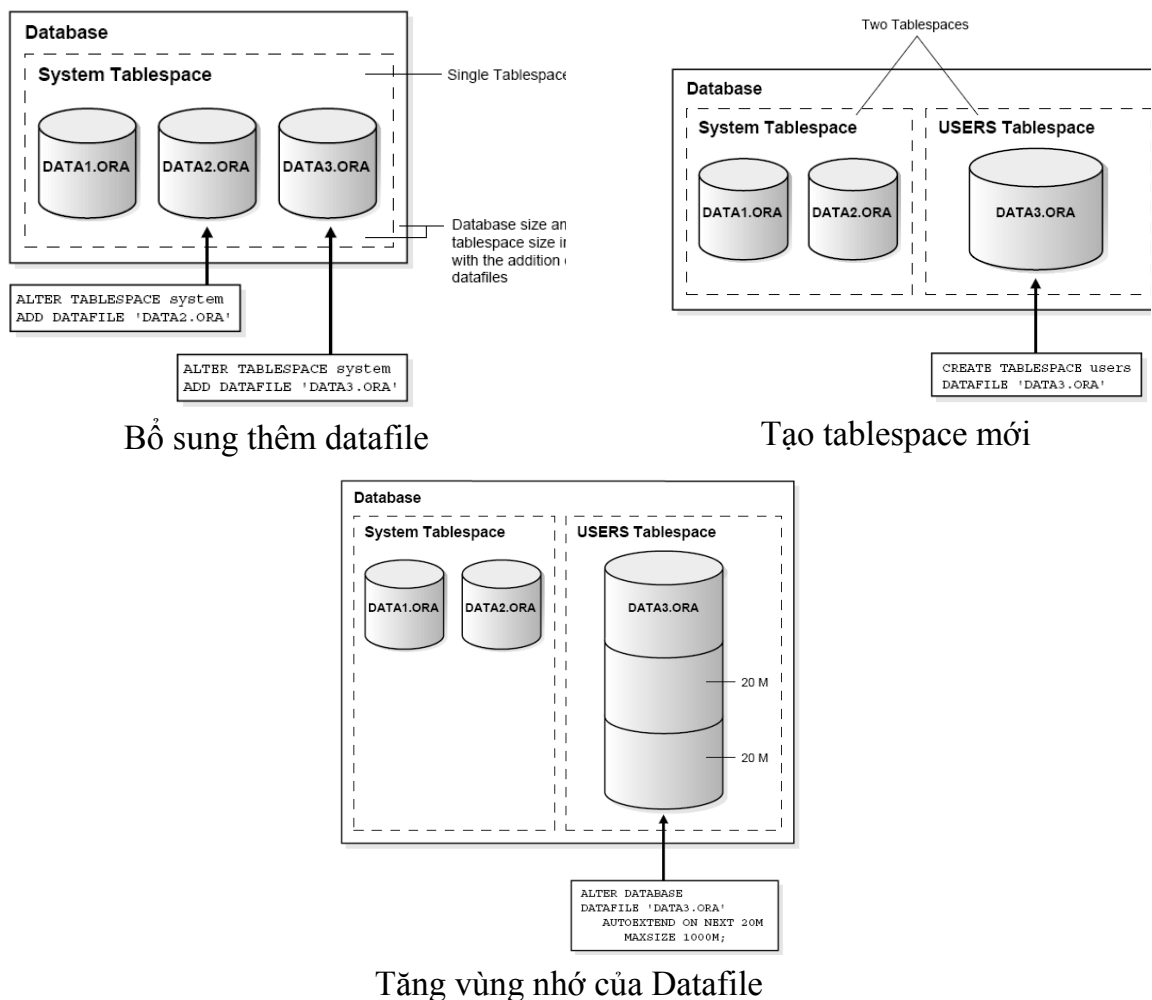
bảng dữ liệu rất lớn, Oracle không thể thực hiện thao tác sắp xếp trong bộ nhớ, một vùng nhớ sẽ được cấp phát trong temporary tablespace để thực hiện thao tác này. Các lệnh SQL cần cung cấp vùng nhớ thực hiện thao tác sắp xếp gồm: CREATE INDEX, ANALYZE, Select DISTINCT, ORDER BY, GROUP BY, UNION, INTERSECT, MINUS,...

Tablespace lưu trữ nhiều loại **đối tượng giản đồ** (schema object) như bảng, chỉ mục, hàm, thủ tục, trigger, ... Bảng dữ liệu chỉ là một loại đối tượng giản đồ, được định danh duy nhất bởi tên và bao gồm nhiều hàng lưu trữ dữ liệu, mỗi hàng được gọi là 1 tuple hay 1 record. Một bảng dữ liệu có thể có nhiều cột. Mỗi cột được định nghĩa bởi 1 tên và 1 kiểu dữ liệu, mô tả thuộc tính của 1 bộ (tuple). Một bảng dữ liệu có thể có tối đa 254 cột có kiểu dữ liệu giống nhau hoặc khác nhau.

8.3.3 Cấp phát thêm vùng lưu trữ cho CSDL

Ta có 3 cách để mở rộng vùng lưu trữ cho CSDL :

- Bổ sung thêm datafile cho tablespace.
- Tạo một tablespace mới.
- Tăng vùng nhớ của datafile.



Hình 6. Cấp phát thêm vùng lưu trữ cho CSDL

8.4 Schema và schema objects

Schema là một tập hợp các đối tượng CSDL (database object hoặc schema object) thuộc về một người dùng. Mỗi người dùng trong 1 CSDL sở hữu duy nhất một schema có tên trùng với tên người dùng, ngược lại mỗi schema chỉ tương ứng với một người dùng mà thôi. Schema Object là cấu trúc luận lý liên quan trực tiếp đến dữ liệu của CSDL, bao gồm các cấu trúc như bảng (table), khung nhìn (view), các thủ tục trữ sẵn (stored procedure)... Các schema objects thường dùng:

- Bảng dữ liệu: đơn vị lưu trữ dữ liệu cơ bản nhất trong CSDL
- Chỉ mục: cấu trúc tùy chọn kết hợp với bảng dữ liệu, được sử dụng nhằm tăng hiệu quả kết xuất dữ liệu.
- Khung nhìn: cách biểu diễn dữ liệu từ một hoặc nhiều bảng hoặc các khung nhìn khác. Thực chất, khung nhìn không chứa đựng dữ liệu mà được xem như một câu truy vấn trữ sẵn (stored query). Dữ liệu trong khung nhìn có thể được cập nhật, xóa, thêm mới như bảng dữ liệu với một số hạn chế. Mọi thay đổi trên khung nhìn ảnh hưởng trực tiếp đến các bảng dữ liệu hoặc khung nhìn khác có liên quan.
- Synonym: là bí danh cho một schema object bất kỳ hoặc một synonym khác. Thông tin về synonym được lưu trữ trong từ điển dữ liệu.

8.5 Data dictionary

Từ điển dữ liệu là một tập hợp các bảng (table) và khung nhìn (view) giống như các CSDL khác, là nơi lưu trữ các thông tin về cấu trúc vật lý và luận lý của CSDL. Các thông tin này bao gồm:

- Thông tin người dùng (quyền, vai trò...)
- Ràng buộc toàn vẹn của các bảng dữ liệu.
- Tên và kiểu dữ liệu của các cột trong bảng dữ liệu.
- Thông tin về vùng nhớ được cấp phát và sử dụng của các schema object.
- ...

Thể hiện CSDL luôn truy cập đến từ điển dữ liệu để phân tích cú pháp câu lệnh SQL. SYS là chủ nhân và là người dùng duy nhất có toàn quyền thao tác trên các bảng và khung nhìn của từ điển dữ liệu.

Để tham khảo các bảng và khung nhìn của từ điển dữ liệu, sử dụng câu lệnh:

```
SELECT * FROM DICT[IONARY]
```

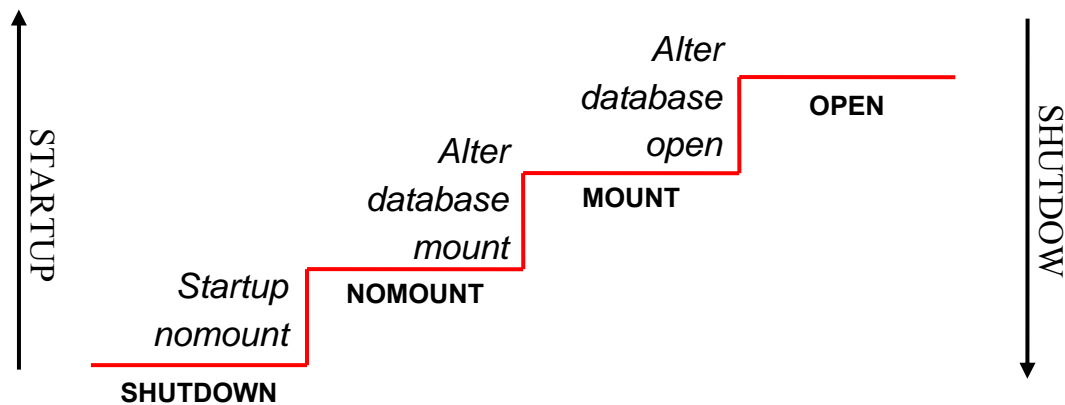
9 Mở/tắt CSDL và thể hiện (startup/shutdown)

9.1 Giới thiệu

Mỗi CSDL Oracle ở trạng thái thực thi đều kết hợp với một thể hiện CSDL (database instance). Khi CSDL được khởi động trên một máy chủ, Oracle cấp phát vùng nhớ SGA và khởi động các tiến trình nền của phần mềm. Sự kết hợp giữa SGA và các tiến trình được gọi là thể hiện CSDL.

Sau khi khởi động thể hiện, Oracle kết hợp thể hiện đó với CSDL tương ứng, lúc này ta nói CSDL đã được liên kết (**mounted database**). Khi đó, CSDL sẵn sàng được mở (**open**) để người dùng truy cập.

Chỉ duy nhất nhà quản trị CSDL có quyền khởi động thể hiện và mở CSDL, cũng như tắt CSDL và giải phóng thể hiện.



Hình 7. Quy trình mở/tắt CSDL Oracle

Chú ý: việc mở/tắt cơ sở dữ liệu bằng lệnh được thực hiện trong chương trình SQL*Plus.

9.2 Mở CSDL

Cách đơn giản nhất để mở CSDL là sử dụng dòng lệnh `startup`, dòng lệnh này sẽ đưa CSDL từ trạng thái shutdown lên open.

▪ Trạng thái nomount

Khi thực hiện lệnh `startup`, đầu tiên CSDL đi vào trạng thái nomount. Trong suốt trạng thái này, Oracle đọc tập tin chứa các thông số khởi động (`init.ora`) để lấy thông tin về cấu hình CSDL như độ lớn vùng nhớ SGA,... Tiếp theo, Oracle khởi động thể hiện CSDL để vào cuối trạng thái nomount.

Đối với một số thao tác khôi phục (recovery) yêu cầu CSDL phải ở trạng thái nomount, chúng ta dùng lệnh `startup nomount`.

Nếu sử dụng lệnh `startup`, CSDL sẽ tiếp tục tự động chuyển sang trạng thái mount.

▪ Trạng thái mount

Khi vào trạng thái này, Oracle mở tập tin điều khiển (control file) để đọc thông tin về vị trí của các data files và các thông tin về cấu trúc vật lý khác. Khi đã định dạng vị trí của các data files, CSDL sẵn sàng được mở.

Đối với một số thao tác khôi phục (**recovery**) yêu cầu CSDL phải ở trạng thái mount, chúng ta dùng lệnh `startup mount`.

Nếu thể hiện đã được khởi động bằng lệnh `startup nomount`, để vào trạng thái mount, chúng ta dùng lệnh `alter database mount`.

Nếu sử dụng lệnh `startup`, CSDL sẽ tiếp tục tự động chuyển sang trạng thái open.

▪ Trạng thái open

Nếu CSDL đang ở trạng thái mount, chúng ta dùng lệnh `alter database open` để vào trạng thái open.

Khi đưa CSDL vào trạng thái open, Oracle truy cập đến tất cả các data file liên quan đến CSDL. Khi đó, Oracle đảm bảo tính nhất quán của tất cả data files. Vào lúc này, CSDL đã sẵn sàng cho người dùng truy cập dữ liệu.

Ngoài cách khởi động thông thường, chúng ta có thể khởi động CSDL ở chế độ hạn chế (restrict mode) bằng câu lệnh `startup restrict`. Ở chế độ này, chỉ những người dùng có quyền đặc biệt (SYS, SYSTEM...) mới có thể truy cập CSDL.

Nếu CSDL đã được mở, chúng ta dùng lệnh `alter system enable restrict session` để đưa CSDL vào trạng thái này. Ngược lại, ta dùng lệnh `alter system disable restrict session`.

9.3 Tắt CSDL

Một số thao tác yêu cầu nhà quản trị phải đóng CSDL như thay đổi cấu hình, sao lưu dữ liệu hay nâng cấp phần mềm Oracle. Khi đó, ta sử dụng lệnh `shutdown`. Đóng CSDL cũng trải qua một số trạng thái theo quy trình ngược lại so với mở CSDL. Tuy nhiên, ta chỉ có thể đưa CSDL về trạng thái shutdown mà không dừng lại ở các trạng thái trung gian được.

- Đóng CSDL: Oracle ghi tất cả thông tin dữ liệu từ SGA ra data files và redo log files. Tiếp theo, tất cả data files và redo log files được đóng lại. Vì thế, CSDL không còn truy cập được. Tuy nhiên, control files vẫn còn được mở vì CSDL vẫn được liên kết.
- Ngắt liên kết (unmount): Oracle ngắt liên kết giữa thể hiện và CSDL. Lúc này, control files được đóng lại và thể hiện vẫn còn trong bộ nhớ.
- Tắt thể hiện: thể hiện được giải phóng khỏi vùng nhớ và các tiến trình nền kết thúc thực thi.

CSDL Oracle có thể được đóng ở 3 chế độ khác nhau:

- Chế độ bình thường (normal): dùng lệnh `shutdown`.

Đây là chế độ mặc định, CSDL chỉ tắt khi tất cả các thao tác trên CSDL kết thúc. Tất cả thay đổi trên dữ liệu đều được ghi lên data files. Tuy nhiên, chế độ này ít dùng vì đòi hỏi Oracle phải chờ đợi tất cả người dùng “log off” khỏi CSDL. Nếu một người dùng nào đó đăng nhập và quên tắt phiên làm việc, CSDL có khả năng không bao giờ tắt được.

- Chế độ tức khắc (immediate): dùng lệnh `shutdown immediate`.

Chế độ này ngăn chặn các truy cập mới, cuộn lại tất cả các giao dịch chưa hoàn tất và tắt CSDL. Tất cả thay đổi trên dữ liệu đều được ghi lên data files

- Chế độ dờ bỏ (abort): dùng lệnh `shutdown abort`.

Đây là cách tắt CSDL ép buộc, cách này ít được sử dụng vì ảnh hưởng đến dữ liệu vì mọi thay đổi sẽ không được ghi ra data files và các giao dịch không được cuộn lại. Đây là lựa chọn cuối cùng nếu CSDL không tắt được bằng các chế độ khác.

10 Oracle Net

Oracle Net là một **middleware** được cài đặt trên **Oracle Client và Oracle Database Server**. Oracle Net cung cấp các giải pháp kết nối trong môi trường tin học không đồng nhất và phân tán, giúp cho việc cấu hình và quản lý mạng dễ dàng hơn. Oracle Net **cho phép thiết lập một phiên kết nối mạng (network session) từ một ứng dụng trình khách (Client Application) đến máy chủ CSDL (Database Server)**. Một khi phiên kết nối mạng được thiết lập, **Oracle Net đóng vai trò như một kênh trao đổi dữ liệu**

giữa ứng dụng khách và máy chủ. Nó có nhiệm vụ thiết lập, quản lý kết nối mạng và trao đổi các thông điệp (messages) giữa trình khách và máy chủ.

Phiên kết nối mạng được thiết lập thông qua listener module, một tiến trình độc lập của máy chủ CSDL. Listener nhận các yêu cầu kết nối của trình khách và quản lý việc dẫn đường đến máy chủ. Mỗi khi trình khách yêu cầu kết nối, listener nhận yêu cầu này. Nếu thông tin cung cấp bởi trình khách tương ứng với các thông tin của listener, trình khách được phép truy cập đến máy chủ.

Đối với client, database là một dịch vụ (service) thực thi công việc thay cho client. Ở đây dịch vụ mà database cung cấp cho client chính là lưu trữ dữ liệu và gọi lại dữ liệu khi cần. Khi kết nối, client nhận biết database qua tên dịch vụ (service name) mặc định trùng với global database name (kết hợp của tên CSDL và tên miền).

Để yêu cầu thiết lập kết nối, người dùng phải gửi đi tên tài khoản, mật khẩu cùng định danh (identifier) của dịch vụ cần kết nối. Định danh này được gọi là định danh kết nối cho phép xác định:

- Dịch vụ cần kết nối.
- Đường dẫn mạng đến dịch vụ này.

Định danh kết nối có thể được biểu thị bằng nhiều cách. Cách thức được áp dụng rộng rãi nhất là tên dịch vụ mạng (***net service name***) tương ứng với một *mô tả kết nối (connect descriptor)*. Mô tả này chứa đựng các thông tin về dịch vụ cần kết nối và đường dẫn mạng. Dịch vụ được xác định bởi tên của dịch vụ (tên của CSDL). Đường dẫn mạng cung cấp các thông tin về địa chỉ mạng, cổng hoạt động và giao thức mạng sử dụng của Listener.

Khi yêu cầu kết nối, client gọi phương thức Service Naming để thiết lập liên lạc với Listener được chỉ định trong *mô tả kết nối* hay *net service name*. Listener chấp nhận kết nối với client thông qua một giao thức mạng. Nó so sánh các thông tin do client cung cấp với các thông tin tương ứng được cung cấp bởi dịch vụ CSDL, nếu các thông tin này khớp nhau, kết nối sẽ được phép thiết lập giữa client và database server.

Phương thức giao tiếp mạng (network protocol) thường được sử dụng trong mạng Oracle là TCP/IP, có cổng mặc định 1521. Ngoài ra, đối với lập trình viên và quản trị viên, ứng dụng phía trình khách (application client) là ứng dụng của nhà phát triển thứ ba (third party), cung cấp một giao diện tương tác thân thiện với hệ quản trị Oracle. Hiện nay, các ứng dụng này được phát triển rất nhiều trên thế giới như Maestro, SQL Navigator, Oracle SQL Developer...

11 Các biện pháp bảo vệ CSDL Oracle

11.1 Quản lý người dùng

Việc phân quyền sử dụng là cần thiết trong công việc quản trị. Có 2 user accounts được tự động tạo ra ngay từ khi tạo database và được gán quyền DBA (**DataBase Administration**) là: SYS và SYSTEM.

- **SYS**: được tạo tự động và gán quyền DBA. Mật khẩu mặc định là change_on_install. Có quyền sở hữu các bảng và các khung nhìn của từ điển dữ liệu trong CSDL.

- **SYSTEM:** được tự động tạo ra với mật khẩu ban đầu là manager và cũng được gán quyền DBA. Tuy nhiên, SYSTEM còn được sở hữu cả một số table, view mở rộng chứa các thông tin **sử dụng cho các tools của Oracle**.

Lưu ý: Ngay khi tạo CSDL, Oracle đã tạo sẵn một quyền gọi là "DBA". Quyền này cho phép thực hiện các thao tác quản trị đối với CSDL. Người dùng có quyền DBA có thể can thiệp được tới các quyền của các user khác sử dụng trong hệ thống. Vì thế, những quản trị viên database cần thay đổi mật khẩu của mình tránh sử dụng mật khẩu mặc định do Oracle cung cấp vì user khác có thể biết và sử dụng để truy nhập vào hệ thống một cách trái phép, gây xáo trộn hệ thống.

Khi tạo một tài khoản mới, ta cần xác định các thông số sau cho tài khoản đó :

- Default Tablespace (tablespace mặc định)

Default Tablespace là tablespace mặc định chứa các segments được tiến trình của người dùng sử dụng để lưu trữ dữ liệu trong trường hợp người dùng không chỉ rõ tên tablespace ngay khi tạo segment.

- Tablespace Quotas (hạn mức tablespace)

Tablespace quotas là dung lượng lưu trữ tối đa ứng với khả năng lưu trữ vật lý được phép của người dùng này trong CSDL.

- Temporary Tablespace (tablespace trung gian)

Temporary tablespace là nơi Oracle server cấp phát các extents phục vụ cho công việc sắp xếp (sort) dữ liệu mỗi khi người dùng thực hiện lệnh truy vấn có sắp xếp.

- Account Locking (khóa account)

Các Accounts có thể bị khóa (locked) để ngăn cản việc người dùng thâm nhập vào CSDL. Việc này có thể được thực hiện một cách tự động hoặc do điều khiển của nhà quản trị CSDL.

- Resource Limits (hạn chế tài nguyên)

Là những giới hạn được đưa ra cho người dùng này về các tài nguyên của hệ thống như: thời gian sử dụng CPU, truy xuất vào ra, số lượng các sessions được mở tối đa,...

11.1.1 Các bước thực hiện khi tạo mới người dùng

- Lựa chọn username (tên user dùng để truy cập CSDL) và cơ chế xác nhận đối với user này.
- Chỉ ra các tablespaces cho user dùng để lưu trữ dữ liệu.
- Gán các default tablespace và temporary tablespace.
- Phân bổ hạn mức sử dụng trên từng tablespace.
- Phân quyền truy nhập (privileges - quyền hoặc roles – vai trò) cho user vừa tạo lập.

11.1.2 Tạo mới người dùng

Cú pháp

```
CREATE USER tên_user  
IDENTIFIED {BY mật_khẩu | EXTERNALLY}  
[ DEFAULT TABLESPACE tên_tablespace ]
```

```
[ TEMPORARY TABLESPACE tên_ tblsp]
[ QUOTA {số_nguyên [K | M ] | UNLIMITED } ON tablespace
[ QUOTA { số_nguyên [K | M ] | UNLIMITED } ON
tên_tablespace ] ...]
```

Với:

tên_user Tên truy nhập của người dùng.

BY mật_khẩu Xác định cơ chế xác nhận bởi CSDL với mật khẩu truy nhập là mật_khẩu.

EXTERNALLY Xác định cơ chế xác nhận user bởi hệ điều hành.

DEFAULT/TEMPORARY tên_tblsp Xác định tablespace mặc định/tạm thời cho người dùng.

QUOTA Xác định lượng không gian tối đa cấp phát cho người dùng để lưu trữ các đối tượng trong từng tablespace. Từ khoá UNLIMITED cho biết không giới hạn không gian cấp phát.

Ví dụ : Tạo một người dùng có tên và password là userTest, cấp Quota 1M trên tablespace USERS

```
CREATE USER userTest IDENTIFIED BY usertest
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP
QUOTA 1M ON USERS ;
```

11.1.3 Thay đổi mật khẩu của người dùng

Cú pháp:

```
ALTER USER tên_user [ IDENTIFIED {BY mật_khẩu | EXTERNALLY } ]
```

Ví dụ: Đổi mật khẩu của người dùng userTest thành ptest

```
ALTER USER userTest IDENTIFIED BY ptest;
```

11.1.4 Thay đổi hạn mức (quota) sử dụng tablespace

Trong một số trường hợp, ta có thể thay đổi hạn mức sử dụng tablespace khi:

- Các bảng của người dùng đó không thể mở rộng để lưu trữ thêm được nữa.
- Các ứng dụng được cải tiến đòi hỏi bổ sung thêm các tables hay indexes.
- Các đối tượng được tổ chức lại và được đặt trên nhiều tablespaces khác nhau.

Cú pháp:

```
ALTER USER tên_user
[ DEFAULT TABLESPACE tên_tablespace]
[ TEMPORARY TABLESPACE tên_tablespace]
[ QUOTA {số_nguyên [K | M] | UNLIMITED } ON tên_tablespace
[ QUOTA {số_nguyên [K | M] | UNLIMITED } ON
tên_tablespace ] ... ]
```

Ví dụ: Tăng hạn mức cho người dùng userTest thành 2M

```
ALTER USER userTest QUOTA 2M ON USERS ;
```

11.1.5 Xóa người dùng

Cú pháp:

```
DROP USER tên_user [CASCADE]
```

Lưu ý:

- CASCADE sẽ huỷ tất cả các đối tượng trong schema trước khi xoá người dùng. Nó cần được chỉ rõ khi schema có chứa đối tượng.
- Ta không thể huỷ được các người dùng hiện đang kết nối tới Oracle server.

11.1.6 Xem thông tin về người dùng

Ta có thể lấy các thông tin liên quan tới user trong data dictionary DBA_USERS và DBA_TS_QUOTAS.

- Với mỗi người dùng, ta có thể xác định được các thông tin về hạn mức.

Ví dụ: xem các thông tin hạn mức của người dùng userTest.

```
SELECT tablespace_name, blocks, max_blocks, bytes, max_bytes  
FROM dba_ts_quotas  
WHERE username = 'userTest';
```

Nếu kết quả trả về có giá trị -1 trong cột MAX_BLOCKS và MAX_BYTES nghĩa là quota không giới hạn (UNLIMITED).

- Hoặc ta cũng có thể lấy các thông tin về tài khoản của người dùng

Ví dụ:

```
SELECT username, account_status, temporary_tablespace  
FROM dba_users;
```

11.2 Quản lý quyền

Mỗi người dùng được tạo ra có thể được gán 2 loại quyền:

- Quyền hệ thống (System Privileges): Các quyền này cho phép người dùng thực hiện thao tác nào đó trong CSDL chẳng hạn tạo bảng, tạo view,..
- Quyền đối tượng (Object Privileges). Các quyền này cho phép người dùng thực hiện thao tác nào đó trong một đối tượng của CSDL chẳng hạn quyền SELECT, INSERT, UPDATE,.. trong một bảng nào đó.

11.2.1 Quyền hệ thống

11.2.1.1 Các loại quyền hệ thống

Oracle database có **khoảng 140 quyền hệ thống** và con số này đang tiếp tục tăng lên. Các quyền hệ thống có thể chia ra như sau:

- Các quyền cho phép thực hiện các thao tác truy cập, tạo dung lượng lưu trữ trên hệ thống ví dụ như: CREATE SESSION, CREATE TABLESPACE.
- Các quyền cho phép quản lý các đối tượng thuộc về một người dùng ví dụ như: CREATE TABLE.
- Các quyền cho phép quản lý các đối tượng trong bất cứ một schema nào ví dụ như câu lệnh: CREATE ANY TABLE.

Có thể điều khiển các quyền bằng cách câu lệnh GRANT hay REVOKE.

Phân loại	Các quyền thông dụng
	CREATE TABLE

TABLE	CREATE ANY TABLE ALTER ANY TABLE DROP ANY TABLE SELECT ANY TABLE UPDATE ANY TABLE DELETE ANY TABLE
SESSION	CREATE SESSION ALTER SESSION RESTRICTED SESSION
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE UNLIMITED TABLESPACE

Bảng 1. Một số quyền hệ thống thông dụng

Chú ý:

- CREATE SESSION là quyền tối thiểu nhất của một user để có thể kết nối vào CSDL.
- Các quyền như CREATE TABLE, CREATE PROCEDURE, CREATE TRIGGER bao gồm cả các quyền xoá các đối tượng đó.
- CREATE TABLE bao gồm các quyền CREATE INDEX và ANALYZE. Với quyền này, người dùng cần có đủ quota trên tablespace hay phải được gán UNLIMITED TABLESPACE.
- Để có thể xoá hết dữ liệu (truncate) của các bảng thì quyền DROP ANY TABLE phải được sử dụng.

11.2.1.2 Gán các quyền hệ thống

Sử dụng cú pháp sau đây để gán quyền hệ thống cho người dùng

```
GRANT {quyền_hệ_thống|tên_role}[, {quyền_hệ_thống|tên_role} ]...  
TO {tên_user|tên_role|PUBLIC}[, {tên_user|tên_role|PUBLIC} ]...  
[WITH ADMIN OPTION]
```

Với:

quyền_hệ_thống	chỉ định quyền hệ thống sử dụng.
tên_role	chỉ định tên Role được gán.
PUBLIC	gán quyền hệ thống cho tất cả người dùng.
WITH ADMIN OPTION	cho phép người dùng được gán quyền có thể gán quyền hay Role đó cho người dùng khác.

Ví dụ:

```
GRANT CREATE SESSION,CREATE TABLE TO userTest;
```

Hoặc nếu muốn cho userTest này có thể cấp lại quyền của mình cho user khác, ta sử dụng thêm mệnh đề WITH ADMIN OPTION

```
GRANT CREATE SESSION,CREATE TABLE TO userTest  
WITH ADMIN OPTION;
```

Một số hướng dẫn

- Người dùng được gán quyền a với tùy chọn WITH ADMIN OPTION thì có thể tiếp tục gán quyền a đó cho một người dùng khác, thậm chí với tùy chọn WITH ADMIN OPTION.
- Bất cứ một người dùng nào có quyền GRANT ANY ROLE có thể gán bất kì quyền nào trong CSDL cho người dùng khác.
- Một người dùng nếu được gán quyền a với tùy chọn WITH ADMIN OPTION thì có thể gán quyền này hay lấy lại các quyền này từ bất cứ người dùng hay role nào trong database.

11.2.1.3 Thu hồi các quyền hệ thống

Sử dụng cú pháp sau đây để lấy lại các quyền hệ thống:

```
REVOKE {quyền_hệ_thống|tên_role}
[, {quyền_hệ_thống|tên_role} ]...
FROM {tên_user|tên_role|PUBLIC}
[, {tên_user|tên_role|PUBLIC} ]...
```

Ví dụ:

```
REVOKE CREATE TABLE FROM userTest;
```

Chú ý:

- Lệnh REVOKE chỉ có thể lấy lại quyền của người dùng đã được gán trực tiếp bằng lệnh GRANT.
- Thu hồi các quyền hệ thống có thể ảnh hưởng đến một số các đối tượng phụ thuộc. Ví dụ: nếu quyền SELECT ANY TABLE được gán cho một người dùng và người dùng đó được gán các thủ tục hay view mà sử dụng các bảng thuộc về các người dùng khác thì việc lấy lại các quyền sẽ làm cho các thủ tục hay view đó trở nên không hợp lệ.

11.2.1.4 Xem thông tin về các quyền hệ thống

Thông tin về các quyền được lấy từ các view của data dictionary: DBA_SYS_PRIVS và SESSION_PRIVS. Các thông tin bao gồm:

- DBA_SYS_PRIVS: GRANTEE, PRIVILEGE, ADMIN OPTION
- SESSION_PRIVS: PRIVILEGE

Ví dụ 1: Liệt kê các quyền hệ thống được gán cho user và role:

```
SELECT * FROM DBA_SYS_PRIVS;
```

Ví dụ 2: Muốn biết user hiện hành có quyền gì.

```
SELECT * FROM SESSION_PRIVS;
```

11.2.2 Quyền đối tượng

11.2.2.1 Các quyền trên đối tượng

Quyền trên đối tượng được gán cho người dùng là thao tác mà người dùng có thể thực hiện trên đối tượng đó. Bảng dưới đây liệt kê các quyền thông dụng có thể được gán trên một đối tượng:

Quyền	Table	View	Procedure
ALTER	X		
DELETE	X	X	
EXECUTE			X
INSERT	X	X	
SELECT	X	X	
UPDATE	X	X	

Bảng 2. Một số quyền trên đối tượng thông dụng

11.2.2.2 Gán các quyền trên đối tượng

Sử dụng cú pháp sau đây để gán một quyền trên đối tượng:

```
GRANT { quyền_đối_tg [(ds_cột)] [, quyền_đối_tg [(ds_cột)] ]...
      | ALL [PRIVILEGES]}
ON [tên_schema.]tên_đối_tượng
TO {tên_user|tên_role|PUBLIC} [, {tên_user|tên_role|PUBLIC} ]...
[WITH GRANT OPTION]
```

Với:

quyền_đối_tg	Chỉ định quyền đối tượng được gán
ds_cột	Chỉ định các cột của một bảng hay view (tuỳ chọn này chỉ sử dụng khi gán các quyền INSERT hay UPDATE.
ALL	Gán tất cả các quyền cho đối tượng mà đã được gán với WITH GRANT OPTION.
tên_đối_tượng	chỉ định đối tượng trên đó các quyền được gán.
WITH GRANT OPTION	cho phép người được gán quyền có thể gán các quyền đó cho một người dùng khác.

Lưu ý:

- Để gán các quyền trên đối tượng, đối tượng đó phải thuộc về schema của người dùng thực hiện gán hoặc người dùng đó có quyền WITH GRANT OPTION.
- Mặc định nếu một đối tượng thuộc về một người dùng nào đó thì người dùng đó có đầy đủ các quyền trên đối tượng đó.
- Tuỳ chọn WITH GRANT OPTION không dùng cho việc gán các quyền đối tượng cho các Role.

Ví dụ: Đăng nhập với tài khoản của người dùng scott và password là tiger, sau đó thực hiện cấp quyền xem và cập nhật dữ liệu trên bảng Emp cho userTest

```
GRANT select,update ON Emp to userTest;
```

11.2.2.3 Thu hồi các quyền trên đối tượng

Sử dụng cú pháp sau đây để lấy lại quyền đã cấp:

```
REVOKE { quyền_đối_tg [, quyền_đối_tg]... | ALL [PRIVILEGES] }
ON [tên_schema.]tên_đối_tượng
FROM {tên_user|tên_role|PUBLIC}
[, {tên_user|tên_role|PUBLIC}...]
```


Với:

`quyền_đối_tg` Chỉ định quyền trên đối tượng đã được gán .

`ALL` Thu hồi toàn bộ các quyền trên đối tượng đã được gán cho người dùng.

`ON` Chỉ định đối tượng trên đó các quyền trên đối tượng được thu hồi.

`FROM` Chỉ định người dùng hay role bị thu hồi quyền.

`CASCADE CONSTRAINTS` Xóa tất cả các tham chiếu mà việc thu hồi đã được định nghĩa do sử dụng quyền `REFERENCES` hay `ALL`.

Ví dụ: Đăng nhập với tài khoản của người dùng scott và password là tiger, sau đó thực hiện thu hồi quyền cập nhật dữ liệu trên bảng Emp từ userTest

```
REVOKE update ON Emp FROM userTest;
```

11.2.2.4 Thông tin về các quyền trên đối tượng

Thông tin về các quyền được lưu trữ trong các data dictionary. Một số thông tin ta cần quan tâm:

- `DBA_TAB_PRIVS`: GRANTEE, OWNER, TABLE_NAME, GRANTOR, PRIVILEGE, GRANTABLE
- `DBA_COL_PRIVS`: GRANTEE, OWNER, TABLE_NAME, COLUMN_NAME, GRANTOR, PRIVILEGE, GRANTABLE

Ví dụ: DBA có thể truy vấn bảng `DBA_TAB_PRIVS` để lấy thông tin về các quyền trên đối tượng được gán cho người dùng userTest.

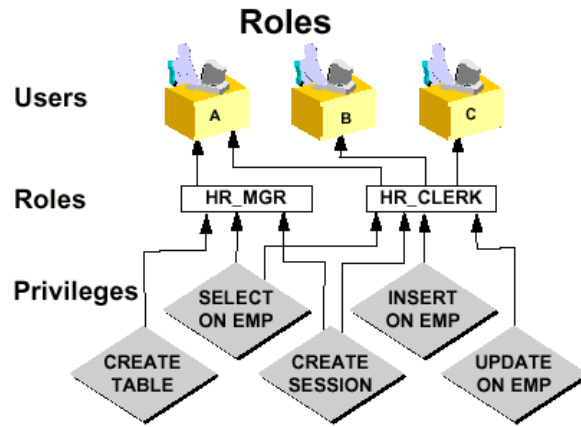
```
SELECT * FROM dba_tab_privs  
WHERE GRANTEE='userTest'
```

11.3 Quản lý Role (vai trò)

11.3.1 Khái niệm Role

Giả sử một CSDL được cấp M quyền như nhau cho N users có cùng chức năng trong hệ thống, như vậy hệ thống cần thực hiện $N \times M$ thao tác gán quyền. Hơn nữa, sau một thời gian, giả sử hệ thống cần thu hồi một quyền nào đó trên N users này, hệ thống phải thực hiện N thao tác thu quyền. Mặt khác, các thao tác gán quyền và thu quyền này là như nhau cho mỗi user, việc này có thể dẫn đến sự nhầm lẫn và mất thời gian trong công việc. Ý tưởng ở đây là sử dụng một nhóm các quyền, nhóm này sẽ được gán cho các users có cùng chức năng trong hệ thống, và việc gán hay thu hồi quyền trên một nhóm sẽ ảnh hưởng trực tiếp đến các user thuộc vào nhóm đó. Vì thế, công việc gán quyền sẽ trở nên nhẹ nhàng và linh động hơn.

Theo ý tưởng đó, Oracle cung cấp một công cụ cho phép quản lý một cách dễ dàng các quyền thông qua việc sử dụng vai trò (Role). Role là một tập hợp các quyền được chỉ định bằng một tên riêng và có thể được gán cho các user hay Role khác.



Hình 8. Role trong database

11.3.2 Các tính chất của Role

- Được gán và lấy lại từ người dùng.
- Có thể gán role cho bất cứ người dùng nào ngoại trừ cho chính nó.
- Có thể bao gồm cả quyền hệ thống (system privileges) và quyền đối tượng (object privileges).
- Có thể enable và disable các Role được gán cho các người dùng.
- Có thể yêu cầu password khi cần enable các Role.
- Tên các Role không trùng với tên người dùng và tên các Role đang tồn tại.
- Không thuộc về bất cứ người dùng nào và không thuộc về bất cứ schema nào.

11.3.3 Lợi ích của việc sử dụng Role

Giảm công việc gán các quyền: sử dụng các Role đơn giản hoá việc quản lý các quyền, bằng cách gán một tập các quyền cho người dùng. Có thể gán các quyền cho một Role và sau đó gán Role đó cho các người dùng.

Quản lý các quyền một cách linh động: khi thay đổi các quyền có trong một Role thì quyền của tất cả các người dùng đã được gán các Role đó sẽ bị thay đổi theo.

11.3.4 Tạo role

Role được tạo ra phải đảm bảo không trùng tên với các người dùng hoặc role khác.

Cú pháp:

```
CREATE ROLE tên_role [NOT IDENTIFIED | IDENTIFIED
    {BY mật_khẩu | EXTERNALLY }]
```

Với:

tên_role	tên của Role
NOT IDENTIFIED	chỉ định không cần kiểm tra Role khi enable Role
BY mật_khẩu	mật khẩu người dùng cần cung cấp khi enable Role
EXTERNALLY	chỉ định người dùng phải được xác lập bởi dịch vụ bên ngoài (như hệ điều hành hay dịch vụ bên thứ ba) trước khi kích hoạt Role.

Ví dụ: Tạo role có tên StudentsGroup

```
CREATE ROLE StudentsGroup;
```

11.3.5 Sửa chữa các Role

Cú pháp:

```
ALTER ROLE tên_role {NOT IDENTIFIED | IDENTIFIED  
    {BY mật_khẩu | EXTERNALLY }};
```

Với:

tên_role	tên của Role cần thay đổi.
NOT IDENTIFIED	chỉ định không cần xác nhận khi enable Role
IDENTIFIED	chỉ định cần xác nhận khi enable các Role
BY mật_khẩu	cung cấp mật khẩu xác nhận khi enable Role
EXTERNALLY	chỉ định user cần được xác nhận bởi dịch vụ bên ngoài (có chế xác nhận bởi hệ điều hành)

11.3.6 Cấp quyền cho Role

Cú pháp giống như cấp quyền cho người dùng.

11.3.7 Gán các Role cho người dùng

Cú pháp :

```
GRANT tên_role [, tên_role ]...  
    TO {tên_user|tên_role|PUBLIC} [,  
{tên_user|tên_role|PUBLIC} ]...  
    [WITH ADMIN OPTION]
```

Với :

tên_role	tên của Role
tên_user	tên của user được gán vào Role_name
tên_role	tên của Role được gán vào Role_name
PUBLIC	Gán cho tất cả các người dùng
WITH ADMIN OPTION	cho phép người dùng được gán Role có thể gán Role tương ứng cho người dùng khác.

Ví dụ:

```
GRANT StudentsGroup TO userTest;
```

11.3.8 Thu hồi các Role từ các user

Sử dụng cú pháp sau đây để thu hồi các Role từ các người dùng:

```
REVOKE tên_role [, tên_role ]...  
    FROM {tên_user|tên_role|PUBLIC} [,  
{tên_user|tên_role|PUBLIC} ]...
```

Với:

tên_role	tên của các Role cần thu hồi.
tên_user	tên người dùng bị thu hồi Role.
tên_role	tên của các Role bị thu hồi Role.

PUBLIC

thu hồi các quyền hay Role từ tất cả các người dùng.

Ví dụ: Chuyển userTest ra khỏi role StudentsGroup

REVOKE StudentsGroup FROM userTest;

11.3.9 Xoá các Role

Để xoá các Role từ database sử dụng câu lệnh sau:

DROP ROLE role_name;

11.3.10 Các Role được định nghĩa sẵn

Tên Role	Diễn giải
CONNECT	Role cung cấp sẵn để tương thích với các phiên bản trước đó
RESOURCE	
DBA	Tất cả các quyền hệ thống, có tùy chọn: WITH ADMIN OPTION
EXP_FULL_DATABASE	Quyền export dữ liệu của database
IMP_FULL_DATABASE	Quyền import dữ liệu vào database
DELETE_CATALOG_ROLE	Quyền xoá dữ liệu
EXECUTE_CATALOG_ROLE	Quyền thực hiện một thủ tục
SELECT_CATALOG_ROLE	Quyền lấy dữ liệu

Bảng 3. Các roles được định nghĩa sẵn

Các Role có tên DELETE_CATALOG_ROLE, EXECUTE_CATALOG_ROLE và SELECT_CATALOG_ROLE cho phép thực hiện truy xuất tới các views và các packages trong data dictionary. Các Role này có thể gán cho user không có quyền DBA nhưng muốn xem thông tin trong các bảng và view thuộc data dictionary.

11.3.11 Thông tin về các role

Thông tin về các Role được lấy trong data dictionary. Có rất nhiều tables và views chứa thông tin về các quyền được gán cho user.

Tên view	Diễn giải
DBA_ROLES	Tất cả các Role trong CSDL
DBA_ROLE_PRIVS	Các Role đã được gán quyền cho user hay Role khác
ROLE_PRIVS	Các Role đã được gán quyền cho Role khác
DBA_SYS_PRIVS	Quyền hệ thống gán cho user hay Role
ROLE_SYS_PRIVS	Quyền hệ thống gán cho Role
ROLE_TAB_PRIVS	Quyền trên table được gán cho Role
SESSION_ROLES	Các Role được phép của user hiện thời

Bảng 4. Thông tin về các roles

Ví dụ: Xem thông tin về các quyền cấp phát cho người dùng

```
SELECT Role, password_required FROM dba_Roles;
```

11.4 Import và export

11.4.1 Giới thiệu

Chức năng export cho phép xuất nội dung luận lý của một CSDL vào một tập tin nhị phân định nghĩa bởi Oracle được gọi là tập tin dump. Chức năng import sẽ dùng tập tin này để tạo lại các đối tượng CSDL trên một lược đồ được chỉ định hoặc trên toàn bộ CSDL. Tập tin dump xuất ra từ một CSDL có thể được dùng để tạo lại dữ liệu trên cùng CSDL đó hoặc trên một CSDL khác, ngay cả khi các CSDL này được cài đặt dưới những cấu hình phần cứng và phần mềm khác nhau. Ví dụ, tập tin dump của một CSDL trên hệ điều hành Windows có thể dùng để tạo lại các đối tượng CSDL trên hệ điều hành Linux. Để thực hiện 2 chức năng này ta dùng 2 lệnh hệ thống exp và imp (thực thi bằng công cụ Command Console của Windows hoặc công cụ Terminal của Linux).

Từ phiên bản 10g, Oracle cung cấp thêm một tính năng mới được gọi là Oracle Data Pump, có hiệu quả thực thi vượt trội so với import và export truyền thống.

11.4.2 Các phương thức của chức năng import và export

11.4.2.1 Cấp độ CSDL

Đây là phương thức phức tạp nhất. Với chức năng export, tất cả các đối tượng của CSDL được xuất ra tập tin dump trừ các đối tượng của một số người dùng như: SYS, ORDSYS, CTXSYS, MDSYS, và ORDPLUGINS. Đồng thời, tập tin dump bao gồm các thông tin liên quan đến cấu trúc của CSDL như định nghĩa các tablespace và các segments rollback... Với chức năng import, tất cả các đối tượng sẽ được tạo lại trong CSDL đích. Tham số FULL cho phép xác định phương thức này trong các chức năng import và export.

Chú ý: trong trường hợp import cả CSDL, cần phải tạo lại tất cả người dùng và các quyền tương ứng trong CSDL nguồn.

11.4.2.2 Cấp độ người dùng

Tất cả các đối tượng của người dùng bao gồm các bảng dữ liệu, thủ tục, trigger... đều được xuất ra tập tin dump. Trong chức năng export, tham số OWNER cho phép chỉ định các đối tượng của người dùng cần xuất. Với chức năng import, tham số FROMUSER chỉ định tạo lại các đối tượng của người dùng (hoặc lược đồ) nào được xuất ra và lưu trữ trong tập tin dump; mặt khác, tham số TOUSER chỉ định người dùng (hoặc lược đồ) đích mà tại đó các đối tượng CSDL được tạo lại.

11.4.2.3 Cấp độ bảng dữ liệu

Tất cả các đối tượng liên quan đến bảng dữ liệu (index, ràng buộc, các quyền...) sẽ được xuất ra tập tin dump.

11.4.2.4 Cấp độ tablespace

Các metadata của tablespace và các đối tượng CSDL được xuất ra tập tin dump.

11.4.3 Yêu cầu về quyền

Hành động	Quyền hoặc vai trò cần thiết
Export schema	CREATE SESSION
Export schema của người dùng khác	SYSDBA, EXP_FULL_DATABASE và DBA
Export toàn bộ CSDL hoặc tablespace	EXP_FULL_DATABASE
Import các đối tượng	IMP_FULL_DATABASE

11.4.4 Các tham số

Tham số	Ý nghĩa	Giá trị mặc định	imp	exp
Userid	Chuỗi kết nối CSDL dưới dạng <tên người dùng>/<mật khẩu>[@<tên dịch vụ mạng>]		x	x
Constraints	Imp/Exp các ràng buộc	Y		x
File	Tên tập tin DUMP	expdat.dmp	x	x
Log	Tên tập tin nhật ký		x	x
Full	Imp/Exp toàn bộ CSDL	N	x	x
Rows	Exp dữ liệu ra file dump	Y		x
Grants	Imp/Exp các quyền trên đối tượng	Y	x	x
Indexes	Imp/Exp các chỉ mục	Y	x	x
Owner	Tên người dùng cần thực hiện Exp	userid		x
Query	Định nghĩa điều kiện lọc các dữ liệu cần Export			x
Tables	Các bảng cần Imp/Exp		x	x
Fromuser	Tên người dùng được Export		x	
Touser	Tên người dùng đích		x	
Tablespace	Imp/Exp các đối tượng được lưu trữ trong tablespace		x	x

Bảng 5. Các tham số trong lệnh imp và exp

Ví dụ

Giả sử ta đã tạo *Net Service Name* **xe** và các người dùng SYS, SCOTT và TEST đã có sẵn trong hệ thống.

- Export toàn bộ CSDL (full=y) không có dữ liệu (rows=n):
`exp
userid=sys/hqtcsql@xe file=D:\export_full.dmp
log=D:\export_full.log full=y rows=n`

- Export toàn bộ schema của người dùng SCOTT: `exp
userid=sys/hqtcsdl@xe file=D:\scott.dmp log=D:\scott.log
owner=scott`
- Export bảng DEPT trong schema SCOTT: `exp userid=scott/tiger@xe
file=D:\scott_dept.dump log=scott_dept.log tables=dept`
- Import tất cả schema: `imp userid=system/hqtcsdl@xe
file=D:\export_full.dmp log=D:\imp_full.log`
- Import các đối tượng của SCOTT vào schema TEST: `imp
userid=system/hqtcsdl@xe file=D:\scott.dmp
log=D:\imp_scott_test.log fromuser=scott touser=test`
- Import bảng DEPT của SCOTT vào schema TEST: `imp
userid=system/hqtcsdl@xe file=D:\scott_dept.dmp fromuser=scott
touser=test tables=scott.dept`

11.5 Oracle Data Pump

Được tích hợp từ phiên bản 10g, Oracle Data Pump hỗ trợ di chuyển nhanh chóng dữ liệu và metadata giữa các CSDL với nhau. Data Pump thừa hưởng tất cả tính năng của phương pháp import/export truyền thống; nhưng có tốc độ thực hiện nhanh hơn imp/exp ít nhất 2 lần. Khác với lệnh imp, tập tin xuất ra từ lệnh expdp sẽ lưu trên server, tập tin dữ liệu mà impdp thao tác cũng phải lưu trên server; vì vậy, ta cần tạo một thư mục chứa các tập tin dump này và quản lý thông qua thư mục đối tượng. Người dùng nào được gán quyền đọc ghi trên thư mục này thì mới có thể sử dụng nó.

11.5.1 Thư mục đối tượng (Directory Objects)

Oracle Data Pump thao tác với các tập tin trên server tại các thư mục được định nghĩa và cấu hình sẵn bởi nhà quản trị.

11.5.2 Qui trình thực hiện

Bước 1: Nhà quản trị định nghĩa các thư mục đối tượng và gán các quyền READ, WRITE cho những người dùng liên quan. Ví dụ, tạo một thư mục để người dùng HR và SCOTT thực hiện import và export. Mở cửa sổ Command, gõ lần lượt các lệnh sau:

- Tạo thư mục mới trong hệ điều hành, ví dụ D:\backup
- `sqlplus sys/hqtcsdl as sysdba (đăng nhập SQLPLUS)`
- `CREATE DIRECTORY dpump_dir AS 'D:\backup';` (chú ý: thư mục này phải được tạo trước bằng các công cụ của hệ điều hành).
- `GRANT READ,WRITE ON DIRECTORY dpump_dir TO hr, scott;`

Bước 2: Người dùng sử dụng các lệnh **expdp** và **impdp** để thực hiện pump dữ liệu trong cửa sổ Command như các ví dụ sau:

- HR thực hiện export toàn schema của mình:
`expdp hr/hr DIRECTORY=dpump_dir dumpfile=hr.dmp`
- SCOTT thực hiện export toàn schema của mình:
`expdp scott/tiger DIRECTORY=dpump_dir dumpfile=scott.dmp`
- HR thực hiện import bảng *emp* và *dept* của SCOTT vào schema của mình :
`Impdp hr/hr@hqtcsdl DIRECTORY=dpump_dir
DUMPFILE=scott.dmp TABLES={scott.emp, scott.dept}
REMAP_SCHEMA=scott:h`

11.5.3 Các tham số tương đương giữa imp/exp truyền thống và Data Pump

Tham số truyền thống	Tham số Data Pump
Userid	Dạng username/password
Constraints	Exclude=Constraint
File	Dumpfile
Log	Logfile
Full	Full
Rows (Y, N)	Contents (All, Metadata_Only)
Grants	Exclude= Grant
Indexes	Exclude=Index
Owner	Schemas
Query	Query
Tables	Tables
Fromuser	REMAP_SCHEMA=<nguồn>:<đích>
Touser	
Tablespace	Tablespace

Bảng 6. Các tham số tương đương giữa imp/exp và Oracle Data Pump

Chương 2 Ngôn ngữ SQL & PL/SQL trong Oracle

Trong học phần Hệ CSDL, chúng ta đã được giới thiệu về ngôn ngữ SQL một cách đầy đủ, chương này chỉ tập trung minh họa một phần của SQL đó là các lệnh liên quan đến việc quản lý cũng như các đối tượng liên quan đến bảng dữ liệu trong Oracle. Ngoài ra, để tăng thêm sức mạnh của mình, các HQTCSDDL thường trang bị thêm ngôn ngữ cho phép người dùng lập trình cấu trúc như MS SQL Server có thêm ngôn ngữ T-SQL, Oracle có PL/SQL. Phần còn lại của chương này sẽ tập trung vào ngôn ngữ PL/SQL.

1 Bảng dữ liệu và các đối tượng liên quan

1.1 Quản lý bảng

1.1.1 Lệnh tạo bảng dữ liệu

```
CREATE [GLOBAL TEMPORARY] TABLE [tên_schema.]tên_table  
  ( tên_cột kiểu_cột  
    [DEFAULT biểu_thức_mặc_định] [Các_ràng_buộc_trên_cột]  
    [,tên_cột kiểu_cột [,...]]  
    [, Các_ràng_buộc_trên_bảng] ) ;
```

Các ràng buộc (RB) trên cột có thể bao gồm một hoặc nhiều loại sau:

```
[CONSTRAINT tên_ràng_buộc] {UNIQUE|PRIMARY KEY}  
[CONSTRAINT tên_ràng_buộc] CHECK(điều_kiện_kiểm_tra)  
[CONSTRAINT tên_ràng_buộc] [NOT] NULL  
[CONSTRAINT      tên_ràng_buộc] REFERENCES      tên_bảng  
[(các_cột_khóa_tham_chiếu)] [ON DELETE {CASCADE|SET NULL}]
```

Các ràng buộc trên bảng có thể bao gồm một hoặc nhiều loại sau:

```
CONSTRAINT      tên_ràng_buộc      {UNIQUE|PRIMARY      KEY} (tên_cột  
[,tên_cột...])  
CONSTRAINT tên_ràng_buộc CHECK(điều_kiện_kiểm_tra)  
CONSTRAINT tên_ràng_buộc FOREIGN KEY (danh_sách_cột)  
REFERENCES tên_table [(các_cột_khóa_tham_chiếu)]  
[ON DELETE {CASCADE|SET NULL}]
```

Ví dụ: Tạo 2 bảng dữ liệu sau:

CUAHANG(MACH,TENCH,DCHI,DTHOAI)

NHANVIEN(MANV,TENNV,NGAYSINH,MACH)

```
CREATE TABLE CUAHANG (  
  MACH INTEGER PRIMARY KEY,  
  TENCH NVARCHAR2(20),  
  DCHI NVARCHAR2(30),  
  DTHOAI VARCHAR(10) ) ;
```

```
CREATE TABLE NHANVIEN (  
    MANV VARCHAR(10) PRIMARY KEY,  
    TENNV NVARCHAR2(30),  
    NGAYSINH DATE,  
    DCHI NVARCHAR2(30),  
    MACH INTEGER CONSTRAINT NV_FK REFERENCES CUAHANG(MACH) ) ;
```

Ngoài ra, ta có thể tạo ra bảng có cấu trúc giống như cấu trúc của kết quả trả về từ câu truy vấn SQL và dữ liệu là các bộ trả về từ câu truy vấn đó, với cú pháp:

```
CREATE TABLE tên_bảng AS câu_truy_vấn_SQL
```

Ví dụ: tạo bảng có cấu trúc và chứa dữ liệu của bảng NHANVIEN

```
CREATE TABLE NV_BACKUP AS SELECT * FROM NHANVIEN;
```

1.1.2 Lệnh sửa đổi bảng dữ liệu

```
ALTER TABLE tên_bảng  
    định_nghĩa_thay_đổi [,định_nghĩa_thay_đổi] ... ;
```

Định nghĩa thay đổi:

```
ADD (tên_cột kiểu_cột[,...])  
| MODIFY (tên_cột [kiểu_dữ_liệu][DEFAULT biểu_thức][,...])  
| DROP COLUMN (tên_cột[,...])  
| RENAME COLUMN tên_cũ TO tên_mới  
| DROP PRIMARY KEY [CASCADE]  
| ADD [CONSTRAINT [tên_ràng_buộc]] PRIMARY KEY (tên_cột[,...])  
| ADD [CONSTRAINT tên_ràng_buộc]] FOREIGN KEY (tên_cột[,...])  
    REFERENCES tên_table [(DS_cột_khóa_tham_chiếu)]  
| DROP CONSTRAINT tên_ràng_buộc [CASCADE]  
| RENAME TO tên_bảng
```

Ví dụ: Xóa ràng buộc khóa ngoại của bảng NHANVIEN

```
ALTER TABLE NHANVIEN  
DROP CONSTRAINT NV_FK ;
```

Tạo lại ràng buộc khóa ngoại của bảng NHANVIEN

```
ALTER TABLE NHANVIEN  
ADD CONSTRAINT NV_FK FOREIGN KEY (MACH)  
REFERENCES CUAHANG(MACH) ;
```

1.1.3 Lệnh xóa bảng dữ liệu

```
DROP TABLE tên_bảng [,tên_bảng] ... [CASCADE CONSTRAINTS]
```

- CASCADE CONSTRAINTS: xóa các ràng buộc toàn vẹn liên quan đến các khóa chính trong bảng. Nếu bỏ qua từ khóa này và trong bảng có tồn tại các ràng buộc toàn vẹn liên quan đến các bảng khác, Oracle báo lỗi và không thực thi câu lệnh này.

Ví dụ:

```
DROP TABLE NHANVIEN CASCADE CONSTRAINTS ;
```

1.1.4 Lệnh chèn record vào bảng

```
INSERT INTO tên_bảng [(cột_i, . . . , cột_j>)]  
VALUES (giá_trị_i, . . . , giá_trị_j>) ;
```

Ví dụ:

```
INSERT INTO CUAHANG  
VALUES ( 1, 'CUA HANG 1', '123 LY TU TRONG', '071088822') ;
```

1.1.5 Lệnh cập nhật record trong bảng

```
UPDATE tên_bảng SET  
cột_i = biểu_thức_i, . . . , cột_j = biểu_thức_j>  
[WHERE điều_kiện>];
```

Ví dụ:

```
UPDATE CUAHANG  
SET TENCH='CUA HANG CUA TOI '  
WHERE MACH=1;
```

1.1.6 Lệnh xóa record trong bảng

```
DELETE FROM tên_bảng [WHERE điều_kiện];
```

Ví dụ:

```
DELETE FROM CUAHANG WHERE MACH=1;
```

1.1.7 Lệnh xóa tất cả records trong bảng

Đôi khi, chúng ta cần xóa tất cả các records trong bảng dữ liệu nhưng vẫn muốn giữ lại cấu trúc của nó. Cách đơn giản nhất là thực hiện lệnh `DROP` để xóa bảng và tạo lại bảng đó với lệnh `CREATE TABLE` cùng với các lệnh khác để tạo các khóa, index và trigger... các thao tác này có thể làm mất nhiều thời gian. Vì vậy, các Hệ quản trị đều cài đặt một lệnh DDL có tên `TRUNCATE` để xóa tất cả các dòng trong bảng dữ liệu nhưng vẫn giữ lại cấu trúc và các ràng buộc toàn vẹn của bảng đó. Lệnh `TRUNCATE` có thể thực hiện nhanh hơn lệnh `DELETE` trong trường hợp này. Tuy nhiên, chúng ta không thể `ROLLBACK` hay sử dụng `FLASHBACK TABLE` để lấy lại dữ liệu bị xóa bởi lệnh này.

Cú pháp:

```
TRUNCATE TABLE tên_bảng ;
```

Để người dùng thực hiện được lệnh này, bảng đó phải thuộc lược đồ (schema) của người dùng đó hoặc người dùng đó được gán quyền `DROP ANY TABLE`.

Ví dụ:

```
TRUNCATE TABLE CUAHANG ;
```

1.2 Sequence

Sequence là danh sách tuần tự các số nguyên, được quản lý tự động bởi Oracle sever. Sequence dùng để tạo giá trị một cách tự động cho khóa chính. Có thể dùng chung một Sequence cho nhiều đối tượng. Con số sequence này có chiều dài tối đa là 38 số. Sequence tương đương với autonumber trong một số hệ quản trị khác.

1.2.1 Tạo Sequence

```
CREATE SEQUENCE tên_sequence  
INCREMENT BY số_nguyên START WITH số_nguyên  
[MAXVALUE số_nguyên] [MINVALUE số_nguyên]  
[CYCLE/NO CYCLE] [CACHE số_nguyên/NOCACHE];
```

Trong đó:

INCREMENT BY : Chỉ định khoảng cách của dãy số tuần tự
START WITH : Chỉ định số đầu tiên của dãy số tuần tự
MAXVALUE : Giá trị lớn nhất của dãy tuần tự
MINVALUE : Giá trị nhỏ nhất của dãy tuần tự
CYCLE/NO CYCLE : Dãy tuần tự có quay vòng khi đến điểm cuối. Mặc định là NO CYCLE
CACHE : Chỉ định bao nhiêu số sequence được lưu trong bộ nhớ.

Ví dụ:

```
CREATE SEQUENCE supplier_seq MINVALUE 1  
START WITH 1 INCREMENT BY 1 CACHE 20;
```

1.2.2 Sử dụng Sequence

Để làm việc với các sequence, dùng lệnh SQL với các cột giả (pseudocolumn) sau:

CURRVAL Cho giá trị hiện thời của sequence.

NEXTVAL Tăng giá trị hiện thời của sequence và cho giá trị sau khi tăng.

Ví dụ:

```
INSERT INTO suppliers (supplier_id, supplier_name)  
VALUES (supplier_seq.NEXTVAL, 'Kraft Foods');
```

1.2.3 Sửa Sequence

```
ALTER SEQUENCE tên_sequence  
INCREMENT BY số_nguyên START WITH số_nguyên  
[MAXVALUE số_nguyên] [MINVALUE số_nguyên]  
[CYCLE/NO CYCLE];
```

1.2.4 Xóa Sequence

```
DROP SEQUENCE tên_sequence;
```

1.3 Index

Index là một cấu trúc tổ chức lưu trữ dữ liệu, được server sử dụng để tìm một dòng trong bảng một cách nhanh chóng.

Cú pháp:

```
CREATE INDEX tên_index  
ON TABLE ( cột [,cột...]);
```

Một số nhận xét về Index:

- Dùng index để tăng tốc câu lệnh truy vấn.
- Index những cột nào dùng để nối giữa các bảng lẫn nhau.

- Không nên dùng Index cho các bảng nào chỉ có vài dòng dữ liệu.
- Primary và unique key tự động có index, nhưng nên có index cho foreign key.

1.4 Biểu thức chính quy (regular expression)

Biểu thức chính quy là chuỗi ký tự đặc biệt mô tả một mẫu tìm kiếm kết quả. Biểu thức chính quy được hỗ trợ trong hầu hết các ngôn ngữ lập trình cấp cao như Perl, PHP, Java, .NET... Trong chương trình học này, biểu thức chính quy được áp dụng để cài đặt ràng buộc miền trị cho các bảng.

Mẫu biểu thức chính quy

Mẫu	Ý nghĩa
*	Xuất hiện một hoặc nhiều lần
.	Bất cứ ký tự gì ngoài NOT NULL
?	Xuất hiện 0 hoặc 1 lần
+	Xuất hiện 1 hoặc nhiều lần
	OR
^	Bắt đầu dòng
\$	Kết thúc dòng
[...]	So khớp với danh sách các ký tự
{ i }, { i, }, { i, j }	So khớp i lần, ít nhất i lần, ít nhất i lần nhưng <= j lần
\d, \D	So khớp với ký tự là số, không là số
\s, \S	So khớp với các ký tự khoảng trắng (space, tab, ...)

Bảng 7. Các mẫu biểu thức chính quy

Các hàm áp dụng biểu thức chính quy

- `REGEXP_LIKE(source, pattern[, options])`: trả về *true* nếu *source* so khớp với *pattern*. *source* là chuỗi ký tự, tên biến hoặc tên cột của bảng; *pattern* là biểu thức chính quy; *options* có thể mang giá trị *i* (không phân biệt hoa thường), *c* (phân biệt hoa thường).
- `REGEXP_REPLACE(source, pattern, replace[, position, occurrence, options])`: trả về chuỗi *source* với các phần so khớp được thay thế bởi chuỗi *replace*. *position* xác định vị trí bắt đầu tìm, mặc định là 1. *occurrence* xác định chuỗi so khớp được thay thế tại thứ tự xuất hiện nào, mặc định là 0 (thay thế tất cả).
- `REGEXP_INSTR(source, pattern, position, occurrence, begin_end, options)`: trả về vị trí bắt đầu hoặc kết thúc của chuỗi so khớp trong *source*, trả về 0 nếu không có phần nào so khớp. *begin_end* xác định giá trị trả về là vị trí bắt đầu hay kết thúc của phần so khớp, mặc định là 0 (vị trí bắt đầu).
- `REGEXP_SUBSTR(source, pattern, position, occurrence, options)`: trả về chuỗi so khớp trong *source*, trả về NULL nếu không tìm thấy.

Các hàm trên (ngoại trừ `regexp_like`) đều có thể sử dụng trong mệnh đề `SELECT`, `regexp_like` có thể được đặt trong mệnh đề `WHERE` hoặc trong lệnh cài đặt ràng buộc toàn vẹn miền trị.

Ví dụ

- Thay thế mỗi ký tự số trong chuỗi bằng một ký tự *

```
SELECT REGEXP_REPLACE('So DT 1: 0909123456, So DT 2: 01234567890', '\d', '*') FROM dual;
```

- Thay thế mỗi số điện thoại di động trong chuỗi bằng một ký tự *

```
SELECT REGEXP_REPLACE('So DT 1: 0909123456, So DT 2: 01234567890', '0\d{9,10}', '*') FROM dual;
```

- Trích ra số điện thoại di động đầu tiên tìm thấy trong chuỗi

```
SELECT REGEXP_SUBSTR('So DT 1: 0909123456, So DT 2: 01234567890', '0\d{9,10}') FROM dual;
```

- Tìm vị trí bắt đầu của số điện thoại thứ 2 tìm thấy trong chuỗi

```
SELECT REGEXP_INSTR('So DT 1: 0909123456, So DT 2: 01234567890', '0\d{9,10}', 1, 2) FROM dual;
```

- Định nghĩa trường Số di động của bảng Test phải tuân theo định dạng “bắt đầu bằng số 0, tiếp theo là 9 hoặc 10 chữ số”

```
CREATE TABLE Test (id INTEGER PRIMARY KEY, name VARCHAR2(50)
                    sodtdd VARCHAR2(11));

ALTER TABLE Test ADD CONSTRAINT check_sodtdd CHECK
(REGEXP_LIKE (Sodtdd, '0\d{9,10}'));
```

2 Các kiểu dữ liệu trong Oracle

Tên kiểu	Giải thích	Biểu diễn hằng
number(p,s)	Kiểu số thập phân ($10^{-84} \rightarrow 10^{127}$)	1894.1204
Integer	Kiểu số nguyên, tương đương với Number(38)	1, 20
Float	Kiểu số thực, tương đương với Number	101.5E5, 0.5E-2
Date	Ngày tháng (1/1/-4712 \rightarrow 31/12/9999)	'10-FEB-04', '10/02/04' (tùy vào định dạng thể hiện ngày của hệ thống)
Char	Ký tự có độ dài ô nhớ cố định và tối đa là 255 ký tự (không hỗ trợ Unicode)	'50% complete.'
Nchar	Ký tự có độ dài ô nhớ cố định và tối đa là 255 ký tự (hỗ trợ Unicode)	‘Nguyễn Văn Minh’ ‘Lương Tâm’
Varchar2	Ký tự có độ dài ô nhớ không cố định và tối đa là 2000 ký tự (không hỗ trợ Unicode)	'50% complete.'

Nvarchar2	Ký tự có độ dài ô nhớ không cố định và tối đa là 2000 ký tự (hỗ trợ Unicode)	‘Nguyễn Văn Minh’ ‘Lương Tâm’
-----------	--	----------------------------------

Bảng 8. Các kiểu dữ liệu trong Oracle

3 Cấu trúc chương trình PL/SQL

3.1 Cấu trúc lập trình trong Oracle

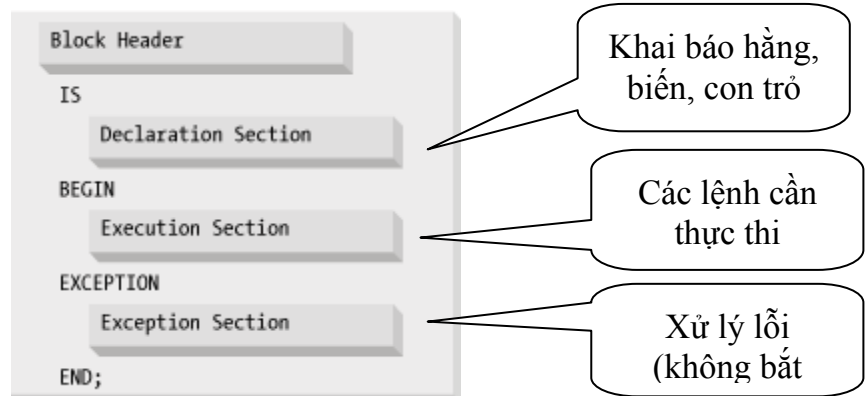
Oracle cho phép thực hiện các đoạn chương trình bên trong nó. Ngôn ngữ được dùng để lập trình cho Oracle được gọi là PL/SQL (Procedural Language/SQL). Ngôn ngữ này vừa có thể xử lý dữ liệu bằng các cú pháp lập trình thông thường, vừa có thể thực hiện các câu lệnh SQL (DML). Phần sau ta sẽ giới thiệu các cấu trúc lập trình của Oracle sử dụng PL/SQL.

Mục tiêu chính của PL/SQL là để:

- Tăng thêm sức mạnh của ngôn ngữ SQL,
- Xử lý kết quả của câu lệnh truy vấn trên từng dòng (dùng cursor),
- Phát triển các chương trình ứng dụng trên CSDL dạng module,
- Tái sử dụng những đoạn code (dùng procedure),
- Giám chi phí trong việc bảo trì và thay đổi ứng dụng.

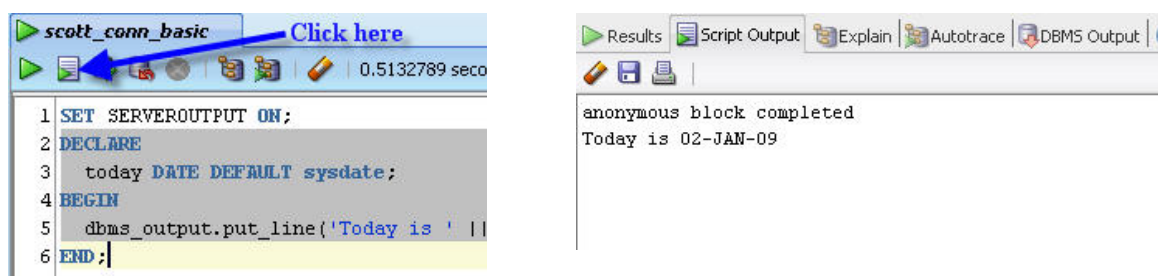
3.2 Các dạng chương trình PL/SQL

Cấu trúc cơ bản của một chương trình PL/SQL là một khối (block). Có thể là khối vô danh (anonymous blocks) hoặc khối được đặt tên (function, procedure, package).

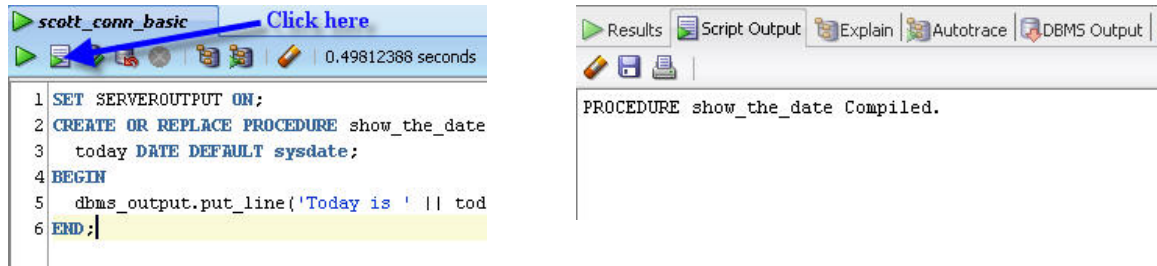


Hình 9. Cấu trúc một khối PL/SQL

Các khối được đặt tên sẽ được lưu trữ ngay trong CSDL Oracle và sẽ được gọi theo phương pháp thủ tục lưu trữ sẵn trong các ngôn ngữ lập trình khác như các ngôn ngữ .NET, Java, PHP,... Các khối này sẽ được giới thiệu chi tiết trong các bài sau.



Khối vô danh



Khối được đặt tên

Hình 10. Hai dạng khối chương trình PL/SQL

3.3 Cấu trúc khối PL/SQL vô danh

Khối PL/SQL vô danh là khối có thể thực hiện lệnh trực tiếp không thông qua tên chương trình con. Một khối PL/SQL vô danh bao gồm 3 phần : khai báo biến, thân khối, và xử lý ngoại lệ.

Cấu trúc cơ bản của khối như sau :

```
DECLARE
    [tên_biến kiểu_biến> [NULL | NOT NULL]
        [DEFAULT trị_mặc_định ]; ]
BEGIN
    câu_lệnh | khối_lệnh;
    [EXCEPTION
        WHEN tên_ngoại_lệ THEN
            câu_lệnh; ]
END;
```

Do các khối có thể lồng nhau nên ta có một số lưu ý về tầm của biến trong khối:

- Khối con sẽ truy xuất được biến khai báo trong khối cha.
- Khối cha không truy xuất được biến được khai báo bên trong khối con.

Ví dụ:

```
SET SERVEROUTPUT ON;
DECLARE
v_deptno    NUMBER(2);
v_loc    VARCHAR2(15);
BEGIN
    DECLARE
        v_dname VARCHAR(10);
    BEGIN
        v_dname := 'SALES';
        DBMS_OUTPUT.put_line(v_dname);

        SELECT deptno, loc INTO v_deptno, v_loc
        FROM dept
        WHERE dname = v_dname; -- Câu SQL
        DBMS_OUTPUT.put_line('v_deptno: ' ||
            v_deptno || ' and v_loc:' || v_loc);
    END;
    -- Câu lệnh sau đây sẽ báo lỗi:
    DBMS_OUTPUT.put_line(v_dname);
END;
```

} Khối con
 } Khối cha

Chú ý: cú pháp “**SET SERVEROUTPUT ON**” được khai báo ở đầu khối PL/SQL nhằm xuất ra màn hình các kết quả trả về từ server Oracle.

4 Các kiểu dữ liệu cơ bản của PL/SQL

PL/SQL hỗ trợ nhiều kiểu dữ liệu cơ bản như sau:

- **BINARY_INTEGER**: kiểu dữ liệu số nguyên được lưu dưới dạng nhị phân. Kiểu này có miền xác định từ -2^{31} đến 2^{31-1} . Ngoài ra, Oracle định nghĩa thêm các kiểu dữ liệu con (subtypes) thừa kế từ kiểu **BINARY_INTEGER**:
 - + **NATURAL**: kiểu số nguyên với miền xác định $[0 .. 2^{31-1}]$.
 - + **POSITIVE**: kiểu số nguyên với miền xác định $(0 .. 2^{31-1}]$.
 - + **NATURALN**, **POSITIVEN**: tương đương các kiểu **NATURAL** và **POSITIVE** nhưng các biến kiểu này không nhận giá trị NULL.
 - + **SIGNTYPE**: kiểu số nguyên giới hạn trong 3 giá trị -1,0,1.
- **NUMBER**[(precision, scale)]: kiểu số thực có miền xác định từ 10^{-130} đến 10^{126} . Ta có thể quy định miền giá trị của biến kiểu này thông qua thông số precision (số chữ số nguyên) và scale (số chữ số thập phân).
 - + Precision có giá trị lớn nhất là 38. Mặc định, một biến kiểu này có phần precision là 38 nếu không được chỉ ra.
 - + Scale có miền xác định từ -84 đến 127. Với scale âm, số được làm tròn về bên trái dấu chấm thập phân. Với scale dương, số được làm tròn về bên phải dấu chấm thập phân.Ví dụ: số thập phân 12345.678 có giá trị là 12300 nếu scale = -2, có giá trị là 12345.68 nếu scale = 2 và có giá trị là 12345 nếu scale = 0.

Các kiểu dữ liệu con của **NUMBER** là:

 - + **DEC**, **DECIMAL**, **NUMERIC**, **DOUBLE PRECISION**, **FLOAT**: kiểu số thực có phần precision đến 38 chữ số thập phân.
 - + **INTEGER**, **INT**, **SMALLINT**: kiểu số nguyên đến 38 chữ số thập phân.
- **BOOLEAN**: kiểu dữ liệu luận lý. Biến kiểu **BOOLEAN** có thể gán 3 giá trị là **TRUE**, **FALSE** và **NULL**.
- **CHAR**(max_length): là kiểu dữ liệu ký tự, có độ dài cho trước xác định qua thông số max_length.
- **DATE**: kiểu dữ liệu ngày tháng. Đây là một kiểu dữ liệu có thuộc tính đặc biệt. Nó có thể chấp nhận cả số lẫn chữ mô tả ngày tháng. Ngoài ra, nó còn chứa thông tin về thế kỷ, năm, tháng, ngày, giờ, phút và giây.
- **VARCHAR2** (max_length): kiểu dữ liệu chuỗi ký tự có chiều dài thay đổi, chiều dài tối đa được xác định bởi max_length (tối đa 32767). Tuy nhiên, độ rộng tối đa của cột dữ liệu (database column) kiểu **VARCHAR2** chỉ giới hạn trong 4000 bytes, nên ta không thể chèn giá trị kiểu **VARCHAR2** có độ dài hơn 4000 bytes vào cột dữ liệu này.
- **LONG**: kiểu dữ liệu chuỗi ký tự có chiều dài thay đổi giống kiểu **VARCHAR2** với chiều dài tối đa là 32760 bytes. Ta có thể chèn giá trị kiểu **VARCHAR2** có độ dài bất kỳ vào cột dữ liệu kiểu **LONG**.

5 Thuộc tính

Các biến trong PL/SQL cũng như các cột và bảng dữ liệu đều có các thuộc tính cho phép ta tham chiếu đến kiểu dữ liệu và cấu trúc của một phần tử mà không cần lặp lại định nghĩa của nó. Dấu % được dùng như một chỉ thị lấy thuộc tính. Các thuộc tính thường dùng:

- **%ROWTYPE**: thuộc tính dùng để lấy kiểu của kiểu của mẫu tin trong một bảng dữ liệu nào đó. Cú pháp:

tên_biến	tên_bảng%ROWTYPE
----------	------------------

Ví dụ: EMP%ROWTYPE tham chiếu đến kiểu của mẫu tin trong bảng EMP.

- **%TYPE**: thuộc tính dùng để lấy kiểu của một trường hoặc một biến nào đó. Việc sử dụng thuộc tính này sẽ mang lại nhiều thuận lợi nếu kiểu của biến thường xuyên thay đổi. Cú pháp:

tên_biến	{tên_cột tên_biến_khác}%TYPE
----------	--------------------------------

Ví dụ: EMP.NAME%TYPE tham chiếu đến kiểu dữ liệu của trường NAME trong bảng EMP. Khi trường kiểu dữ liệu NAME của bảng EMP bị thay đổi, EMP.NAME%TYPE cũng thay đổi theo.

Ví dụ:

```
DECLARE
    total_sales NUMBER(20,2);
    monthlySales total_sales%TYPE; /* biến monthlySales sẽ lấy
                                    kiểu từ biến total_sales
                                    */
    vEname employees.last_name%TYPE; /* biến vEname sẽ lấy kiểu
                                       từ kiểu dữ liệu của cột last_name trong
                                       bảng employee */
BEGIN
    NULL; -- không lệnh nào cả nên đặt NULL ở đây
END;
```

6 Kiểu dữ liệu phức

Oracle hỗ trợ nhiều kiểu dữ liệu phức. tuy nhiên do thời gian hạn hẹp, giáo trình chỉ đề cập đến một số kiểu dữ liệu có sử dụng trong giáo trình.

6.1 Kiểu dữ liệu con do người dùng định nghĩa

Mỗi kiểu dữ liệu cơ bản định nghĩa một tập hợp các giá trị và các phép toán áp dụng trên các biểu thức kiểu này. Kiểu dữ liệu con (subtype) kế thừa tập hợp các phép toán của kiểu dữ liệu cha (base type) nhưng nó chỉ xác định trong một khoảng giá trị của kiểu cha. Tuy nhiên, kiểu dữ liệu con khi được định nghĩa không tạo ra một kiểu dữ liệu mới, thực chất nó chỉ thay thế một ràng buộc trên kiểu dữ liệu cha.

Kiểu dữ liệu con được khai báo trong phần định nghĩa (declare) của khối PL/SQL, chương trình con và package theo cú pháp:

```
SUBTYPE tên_kiểu_con IS tên_kiểu_cha [(rang_buộc)] [NOT NULL]
```

Ví dụ:

```
SET SERVEROUTPUT ON;
```

```
DECLARE
    -- Định nghĩa kiểu con BirthDate
    SUBTYPE BirthDate IS DATE NOT NULL;
    -- Khai báo biến kiểu BirthDate
    myBirthDate BirthDate := '14-mar-2008';

BEGIN
    DBMS_OUTPUT.put_line('My birthday is: ' || myBirthdate);
END;
```

6.2 Kiểu TABLE

Tương tự kiểu mảng của ngôn ngữ lập trình có cấu trúc. Cú pháp khai báo:

```
TYPE tên_kiểu_bảng IS
    TABLE OF kiểu [NOT NULL] INDEX BY BINARY_INTEGER;
tên_biến tên_kiểu_bảng;
```

Ví dụ: tìm tên những nhân viên có lương lớn hơn 10000.

```
SET SERVEROUTPUT ON;
DECLARE
    TYPE Hr_Emp IS TABLE OF hr.employees%ROWTYPE
        INDEX BY BINARY_INTEGER;
    emp_tab Hr_Emp;
    x INTEGER;
BEGIN
    SELECT * BULK COLLECT INTO emp_tab
    FROM hr.employees
    WHERE salary > 10000;
    FOR x IN 1..emp_tab.COUNT LOOP
        dbms_output.put_line(emp_tab(x).FIRST_NAME);
    END LOOP;
END;
```

7 Các loại mệnh đề

7.1 Mệnh đề gán

Cú pháp:

tên_biến := biểu_thức;

Ví dụ:

```
DECLARE
    total_sales    NUMBER(15,2);
    emp_id         VARCHAR2(9);
    company_number NUMBER DEFAULT 10; -- Gán giá trị mặc định
BEGIN
    total_sales    := 350000;
    emp_id         := 3;
    dbms_output.put_line('Employee ' || emp_id ||
```

```
' , sold total product value: ' || total_sales);  
END;
```

7.2 Mệnh đề lệnh (SQL command)

- Mệnh đề SELECT...INTO truy vấn giá trị được chứa trong bảng và lưu vào biến. Cú pháp:

```
SELECT    tên_cột_1,    tên_cột_2...    INTO    biến1,    biến2...  
[biến_con_trò]  
FROM tên_bảng_1, tên_bảng_2...  
[WHERE điều_kiện_1, điều_kiện_2... ]
```

Với:

INTO biến1, biến2... [biến_con_trò]: đưa kết quả trả về từ select vào các biến.

Ví dụ:

```
SET SERVEROUTPUT ON;  
DECLARE  
    v_deptno dept.deptno%TYPE;  
    v_loc dept.loc%TYPE;  
BEGIN  
    SELECT deptno, loc INTO v_deptno, v_loc  
    FROM dept  
    WHERE dname = 'SALES';  
    DBMS_OUTPUT.put_line(v_deptno || ' : ' || v_loc );  
END ;
```

- Mệnh đề INSERT, DELETE và UPDATE: tương tự mệnh đề trong ngôn ngữ SQL. Tuy nhiên, các giá trị tham gia vào mệnh đề có thể được chỉ định bởi các biểu thức. Cú pháp:

Ví dụ:

```
INSERT INTO counts (sales_set, non_sales_set)  
VALUES (v_sales_count, v_non_sales);
```

Trong đó:

v_sales_count, v_non_sales: các biến (biểu thức).

8 Các cấu trúc điều khiển

8.1 Cấu trúc rẽ nhánh

8.1.1 Mệnh đề IF:

Mệnh đề IF có 2 cú pháp. Cú pháp dành cho IF ... ELSE... END IF và cú pháp IF...ELSIF...ELSIF... ELSE... END IF. Cú pháp đầu tiên chỉ cho một điều kiện đúng hoặc sai. Cú pháp thứ hai có thể kiểm tra nhiều điều kiện lần lượt cho đến hết.

<pre>IF điều_kiện_1 THEN lệnh_1; ELSIF điều_kiện_2 THEN</pre>

```
        lệnh_2;  
ELSE  
        lệnh_3  
END IF;
```

Ví dụ:

```
DECLARE  
    grade CHAR(1);  
BEGIN  
    grade := 'B';  
    IF grade = 'A' THEN  
        dbms_output.put_line('Excellent');  
    ELSIF grade = 'B' THEN  
        dbms_output.put_line('Very Good');  
    ELSIF grade = 'C' THEN  
        dbms_output.put_line('Good');  
    ELSIF grade = 'D' THEN  
        dbms_output.put_line('Fair');  
    ELSIF grade = 'F' THEN  
        dbms_output.put_line('Poor');  
    ELSE  
        dbms_output.put_line('No such grade');  
    END IF;  
END;
```

8.1.2 Mệnh đề CASE:

Tương tự như mệnh đề IF, mệnh đề CASE cũng chỉ thực thi một trong những chuỗi lệnh cho trước tùy theo giá trị của biến lựa chọn (selector). Cú pháp:

```
[<<tên_nhân>>]  
CASE biến  
    WHEN biểu_thức_1 THEN chuỗi_lệnh_1;  
    WHEN biểu_thức_2 THEN chuỗi_lệnh_2;  
    ...  
    WHEN biểu_thức_N THEN chuỗi_lệnh_N;  
    [ELSE chuỗi_lệnh_N+1;]  
END CASE [tên_nhân];
```

Ví dụ:

```
DECLARE  
    grade CHAR(1) := 'B';  
BEGIN  
    CASE grade  
        WHEN 'A' THEN dbms_output.put_line('Excellent');  
        WHEN 'B' THEN dbms_output.put_line('Very Good');  
        WHEN 'C' THEN dbms_output.put_line('Good');  
        WHEN 'D' THEN dbms_output.put_line('Fair');  
        WHEN 'F' THEN dbms_output.put_line('Poor');
```

```
ELSE dbms_output.put_line('No such grade');
END CASE;
END;
```

8.1.3 Mệnh đề Search CASE:

Tương tự như mệnh đề CASE nhưng ở đây không có biến lựa chọn (selector). Cú pháp:

```
[<<tên_nhãn>>]
CASE
    WHEN đk_tìm_kiểm_1 THEN chuỗi_lệnh_1;
    WHEN đk_tìm_kiểm_2 THEN chuỗi_lệnh_2;
    ...
    WHEN đk_tìm_kiểm_N THEN chuỗi_lệnh_N;
    [ELSE chuỗi_lệnh_N+1;]
END CASE [tên_nhãn];
```

Ví dụ:

```
DECLARE
    quantity NUMBER;
    projected NUMBER;
    needed NUMBER;
BEGIN
    quantity := -10;
    <<here>>
    CASE
        WHEN quantity is null THEN
            dbms_output.put_line('So luong rong');
        WHEN quantity + projected >= needed THEN
            dbms_output.put_line('So luong ' || quantity || ' dap
ung.');
```

du.');

```
        WHEN quantity >= 0 THEN
            dbms_output.put_line('So luong ' || quantity || 'khong
du.');
```

```
    END CASE here;
    EXCEPTION
        WHEN CASE_NOT_FOUND THEN -- Case không tìm thấy
            dbms_output.put_line('So luong da nho hon 0.');
```

```
END;
```

8.2 Cấu trúc lặp

8.2.1 Mệnh đề LOOP:

Mệnh đề LOOP gồm có vòng lặp LOOP không xác định cận và vòng lặp LOOP xác định cận.

```
LOOP
    Lệnh;
    EXIT WHEN điều_kiện;
```

```
END LOOP;
```

8.2.2 Mệnh đề WHILE:

Gần giống như mệnh đề LOOP, nhưng khác biệt ở chỗ điều kiện ngừng vòng lặp sẽ được kiểm tra trước mỗi lần lặp.

```
WHILE điều_kiện  
LOOP  
    Lệnh;  
END LOOP;
```

8.2.3 Mệnh đề FOR:

Xác định trước cận dưới và cận trên của vòng lặp. Cú pháp:

```
FOR biến_chạy IN [REVERSE] cận_dưới..cận_trên  
LOOP  
    lệnh | chuỗi_lệnh;  
END LOOP;
```

Từ khóa REVERSE xác định biến chạy của vòng lặp đi từ cận trên đến cận dưới

8.2.4 Mệnh đề GOTO:

Lệnh lặp vô điều kiện. Khi chương trình PL/SQL thực hiện đến lệnh này, con trỏ chương trình sẽ chuyển đến vị trí nhãn và thực hiện các lệnh tiếp theo sau nhãn. Cú pháp:

```
<<tên_nhãn>>  
...  
    lệnh;  
GOTO tên_nhãn;
```

8.3 Cấu trúc Ngoại lệ (Exception)

Ngoại lệ là một trạng thái lỗi được kích hoạt khi xảy ra một vấn đề nào đó trong chương trình. Khi ngoại lệ được kích hoạt, việc thực thi mã lệnh được dừng lại tại dòng lệnh xảy ra ngoại lệ và điều khiển được chuyển cho phần xử lý ngoại lệ của khối lệnh tương ứng. Nếu khối lệnh này không có phần xử lý ngoại lệ thì điều khiển được chuyển cho phần xử lý ngoại lệ của khối cha. Như vậy, điều khiển được chuyển cho phần xử lý ngoại lệ của khối cha nếu khối con không có phần xử lý ngoại lệ. Nếu việc chuyển điều khiển được tiếp diễn và khối cha cao nhất không có phần xử lý ngoại lệ thì việc thực thi kết thúc (halted) với lỗi không xử lý ngoại lệ.

Phần xử lý ngoại lệ sẽ trả lại điều khiển sau khi kết thúc thực thi, nhưng điều khiển không được trả lại ngay dòng lệnh kích hoạt ngoại lệ mà được trả lại cho dòng lệnh ngay sau khối lệnh chứa dòng lệnh gây lỗi.

Phần xử lý ngoại lệ của khối là nơi để khai báo các dòng lệnh thực hiện:

- Đưa ra các thông báo lỗi có ý nghĩa hơn so với mã lỗi trả về của ngoại lệ.
- Thực hiện các thao tác xóa vết các dữ liệu đã được thay đổi trước khi lỗi xảy ra để tránh vấn đề dữ liệu không nhất quán về sau, như thực hiện lệnh ROLLBACK.

Cú pháp phần xử lý ngoại lệ trong khối lệnh:

```
EXCEPTION
```

```
WHEN {tên_ngoại_lệ [OR tên_ngoại_lệ] | OTHERS} THEN  
lệnh | khối_lệnh;
```

Để lấy thông tin của một ngoại lệ, ta có thể sử dụng 2 hàm:

- **SQLCODE**: trả về mã lỗi của ngoại lệ.
- **SQLERRM**: trả về thông báo lỗi của ngoại lệ.

Ví dụ:

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    a NUMBER := 6;
```

```
    b NUMBER;
```

```
BEGIN
```

```
    BEGIN
```

```
        b := 0;
```

```
        a := a / b;
```

```
    EXCEPTION      -- bắt đầu phần xử lý ngoại lệ
```

```
        WHEN ZERO_DIVIDE THEN -- bắt ngoại lệ chia cho 0
```

```
            DECLARE
```

```
                ma_loi NUMBER := SQLCODE;  -- lấy mã của ngoại lệ
```

```
                thông_bao VARCHAR2(512) := SQLERRM;  -- lấy thông báo
```

```
            BEGIN
```

```
                dbms_output.put_line('Ma loi: ' || ma_loi);
```

```
                dbms_output.put_line('Thong bao loi: ' || thông_bao);
```

```
                dbms_output.put_line('Truoc loi. Gia tri cua a = ' ||
```

```
a);
```

```
                dbms_output.put_line('Truoc loi. Gia tri cua b = ' ||
```

```
b);
```

```
            END;
```

```
        WHEN OTHERS THEN
```

```
            dbms_output.put_line('Khong biet loi gi.');
```

```
    END;
```

```
    b := 7;
```

```
    dbms_output.put_line('Sau loi. Gia tri cua b = ' || b);
```

```
END;
```

8.3.1.1.1 Các ngoại lệ hệ thống (system-defined exceptions) thường xảy ra

Tên ngoại lệ	Diễn giải
CURSOR_ALREADY_OPEN	mở một con trỏ đang mở
INVALID_CURSOR	con trỏ sai hoặc chưa khởi tạo
NO_DATA_FOUND	Khi lệnh SELECT ... INTO không có dữ liệu trả về
TOO_MANY_ROWS	có nhiều hơn một mẫu tin được select hay fetch vào một record, hay một biến
VALUE_ERROR	giá trị của biến bị gán sai kiểu

ZERO_DIVIDE

thực hiện phép chia cho 0

Bảng 9. Các ngoại lệ thường xảy ra**8.3.1.1.2 Ngoại lệ do lập trình viên định nghĩa (programmer-defined exceptions)**

Để kích hoạt hoặc xử lý một ngoại lệ tự định nghĩa, ta phải khai báo và gán tên cho ngoại lệ đó.

Ví dụ:

```
SET SERVEROUTPUT ON
DECLARE
    sl1 NUMBER := -2;
    sl2 NUMBER := 3;
    tong NUMBER := 0;
    sl_phai_duong EXCEPTION;
    FUNCTION myFunc(sl NUMBER) RETURN NUMBER IS
    BEGIN
        IF(sl > 0) THEN
            RETURN sl * 20;
        ELSE RAISE sl_phai_duong; -- Lệnh RAISE kích hoạt ngoại lệ
        END IF;
    END myFunc;
BEGIN
    tong := myFunc(sl2);
    tong := tong + myFunc(sl1);
    dbms_output.put_line('Tong = ' || tong);
    EXCEPTION
        WHEN sl_phai_duong THEN
            dbms_output.put_line('(Exception) Tong = ' || tong);
            dbms_output.put_line('Loi xay ra, so luong phai > 0');
END;
```

9 Kiểu con trỏ (Cursor)

Là kiểu dữ liệu có cấu trúc cho phép xử lý dữ liệu gồm nhiều dòng. Số dòng được lấy ra xử lý phụ thuộc vào lệnh xử lý sau đó. Trong quá trình xử lý, thao tác sẽ tác động lên từng mẫu tin của dữ liệu đã được nạp vào con trỏ. Con trỏ có thể được dịch chuyển từ mẫu tin này đến mẫu tin khác, nhờ đó mà sử dụng con trỏ có thể xử lý được hết dữ liệu của một bảng dữ liệu.

Các bước sử dụng biến cursor: Khai báo → mở cursor → lấy dữ liệu để xử lý → đóng cursor

9.1 Cú pháp khai báo con trỏ:

<code>CURSOR tên_con_trỏ [(ds_tham_số>)] IS lệnh_select;</code>
--

Ví dụ 1: Khai báo cursor không có tham số

```
DECLARE
    CURSOR c_Dept1 IS
        SELECT deptno, dname
```

```
FROM dept
WHERE deptno>10;

BEGIN
    NULL;
END;
```

Ví dụ 2: Khai báo cursor có tham số

```
CURSOR c_Dept2(p_Deptno NUMBER) IS
    SELECT deptno, dname
    FROM dept
    WHERE deptno>p_Deptno;

BEGIN
    NULL;
END;
```

9.2 Cú pháp mở con trỏ:

```
OPEN [tên_con_trỏ [(ds_giá_trị_truyền_vào_tham_số)] ];
```

Ví dụ:

```
OPEN c_Dept1;      -- trường hợp CURSOR được khai báo không có tham số
```

Hoặc:

```
OPEN c_Dept2(10); -- trường hợp CURSOR được khai báo có tham số
```

9.3 Cú pháp lấy dữ liệu

```
FETCH Tên_tên_con_trỏ INTO tên_biến_con_trỏ;
```

9.4 Cú pháp đóng con trỏ

```
CLOSE tên_con_trỏ;
```

Ví dụ:

```
DECLARE
-- Khai báo cursor để truy vấn dữ liệu
CURSOR c_Emp IS
    SELECT *
    FROM EMP
    WHERE deptno = 10;
-- Khai báo biến cursor tương ứng để chứa dòng dữ liệu
v_Emp c_EMP%ROWTYPE;
BEGIN
    -- Mở cursor
    OPEN c_Emp;
    dbms_output.put_line('EMPNO | ENAME | JOB');
    LOOP
        -- Lấy dòng dữ liệu từ cursor
        FETCH c_Emp INTO v_Emp;
        -- Thoát khỏi vòng lặp khi biến v_Emp rỗng
        EXIT WHEN c_emp%NOTFOUND;
        -- Các câu lệnh xử lý v_Emp
```

```

dbms_output.put_line(v_Emp.empno || ' | ' || v_Emp.ename ||
                    ' | ' || v_emp.job);

END LOOP;
CLOSE c_Emp;
END;
```

9.5 Một số thuộc tính của con trỏ:

Thuộc tính	Diễn giải
%notfound	Trả về giá trị True nếu lệnh fetch vừa thực hiện trả về rỗng
<u>%found</u>	Trả về giá trị True nếu lệnh fetch vừa thực hiện trả về khác rỗng <i>'+tìm về'</i>
%rowcount	Trả về số mẫu tin đã được lệnh fetch truy xuất
%isopen	Trả về giá trị True nếu CURSOR được mở, ngược lại trả về False.

Bảng 10. Diễn giải các thuộc tính của con trỏ

9.6 Mệnh đề SELECT FOR UPDATE trong Cursor

Bạn thử nghĩ điều gì sẽ xảy ra khi dữ liệu mà ta đang thao tác trên cursor bị cập nhật bởi một người dùng khác? Để tránh tình trạng này, ta sẽ khóa các mẫu tin được truy xuất bằng cách dùng mệnh đề FOR UPDATE trong câu lệnh SELECT khi khai báo cursor. Các mẫu tin sẽ bị khóa khi ta mở cursor và sẽ tự động mở khóa (unlock) khi gặp thao tác COMMIT hoặc ROLLBACK kế tiếp. Trong khi các mẫu tin bị khóa, không ai khác ngoài người dùng đang giữ quyền khóa các mẫu tin được cập nhật các mẫu tin này.

```

SELECT ...
FROM ...
FOR UPDATE [OF column_reference];
```

Trong đó, [OF column_reference] chỉ định các cột cụ thể sẽ bị khóa, dĩ nhiên các cột đó phải có mặt trong mệnh đề SELECT.

Ví dụ: Tạo Cursor truy xuất dữ liệu từ bảng emp và sẽ khóa 2 cột lương (sal), thưởng (comm) khi cursor này được mở.

```

DECLARE
    CURSOR c_Emp2 IS
        SELECT ename, job, sal, comm
        FROM emp
        WHERE deptno=10
        FOR UPDATE OF sal, comm;
BEGIN
    --cac lenh xu ly
END;
```

9.7 Mệnh đề WHERE CURRENT OF trong Cursor

Ở phần trước ta đã biết mệnh đề FOR UPDATE – các mẫu tin của table có liên quan trong câu SELECT sẽ bị khóa khi cursor được mở. Vì vậy một vấn đề phát sinh

là nếu trong cursor ta đang thực hiện (có sử dụng FOR UPDATE) lại cần cập nhật dữ liệu trên chính bảng đó!

Để giải quyết vấn đề này, Oracle cho phép ta sử dụng mệnh đề **WHERE CURRENT OF**. Cú pháp này chỉ định rằng câu lệnh UPDATE hoặc DELETE sẽ được sử dụng để sửa đổi dữ liệu trên những dòng của cursor hiện hành có định nghĩa FOR UPDATE. Cú pháp khai báo:

```
[lệnh_update | lệnh_delete] WHERE CURRENT OF tên_con_trỏ;
```

Khi sử dụng WHERE CURRENT OF, ta không cần phải lặp lại mệnh đề WHERE đã có trong câu lệnh SELECT.

Ví dụ :

```
DECLARE
    CURSOR c_1 IS SELECT empno, ename, salary
                   FROM emp WHERE comm IS NULL
                   FOR UPDATE OF comm;
    var_comm NUMBER(10,2);
BEGIN
    FOR r_1 IN c_1 LOOP
        IF r_1.salary < 5000 THEN
            var_comm := r_1.salary * 0.25;
        ELSIF r_1.salary < 10000 THEN
            var_comm := r_1.salary * 0.20;
        ELSIF r_1.salary < 30000 THEN
            var_comm := r_1.salary * 0.15;
        ELSE
            var_comm := r_1.salary * 0.12;
        END IF;
        UPDATE emp SET comm = var_comm
        WHERE CURRENT OF c_1;
    END LOOP;
END;
```

10 Thủ tục, hàm và trigger

Chương trình được viết bằng ngôn ngữ PL/SQL ngoài dạng khối vô danh như trên, còn có các dạng sau:

- Function: chương trình con là hàm hoàn chỉnh.
- Procedure: chương trình con là thủ tục hoàn chỉnh.
- Trigger : Bẫy được gắn kết vào một bảng nào đó nhằm kiểm tra tính toàn vẹn của dữ liệu.
- Package: gói khai báo chứa các hàm hoặc thủ tục bên trong.
- Package Body: chứa các cài đặt các hàm hoặc thủ tục của gói đã được khai báo trong phần package.

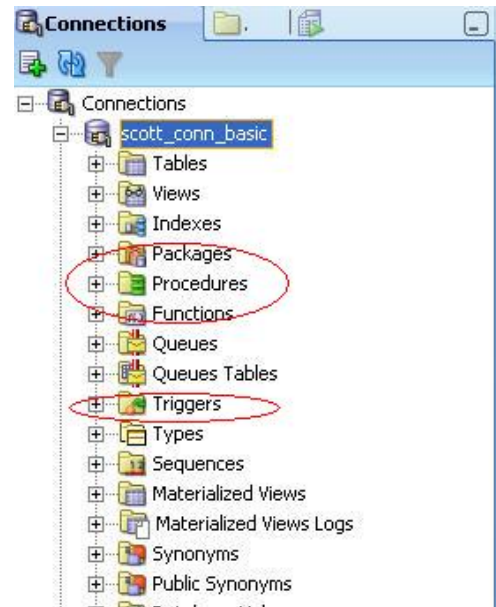
Trong các dạng trên, hàm được lưu trữ trong các schema qua node function, thủ tục được lưu trong node procedure.

Như các ngôn ngữ lập trình khác, PL/SQL cũng bao gồm hai loại chương trình con: hàm và thủ tục. **Hàm là chương trình con có kết quả trả về, thủ tục không có kết quả trả về.**

10.1 Thủ tục

Thủ tục là một nhóm các lệnh thực hiện các chức năng nào đó được gom lại trong một khối nhằm làm tăng khả năng xử lý, khả năng sử dụng chung, tăng tính bảo mật và an toàn dữ liệu và tăng tiện ích trong phát triển.

Thủ tục có thể được lưu giữ ngay trong CSDL như một đối tượng của CSDL, sẵn sàng cho việc tái sử dụng. Thủ tục còn được gọi là **Stored Procedure**. Với các **Stored Procedure**, ngay khi tạo chúng đã được biên dịch thành dạng p-code vì thế có thể nâng cao khả năng thực hiện.



Hình 11. Vị trí lưu trữ các chương trình trong schema

10.1.1 Tạo một thủ tục

```
CREATE [OR REPLACE] PROCEDURE
tên_thủ_tục
[ (tham_số_1 [IN | OUT | IN OUT] kiểu_dl
  [, tham_số_2> [IN | OUT | IN OUT] kiểu_dl [DEFAULT
    trị_mặc_định]])]
IS
  [tên_biến kiểu_dl> [NULL | NOT NULL] [DEFAULT trị_mặc_định];]
BEGIN
  [chuỗi_lệnh] -- Đây là khối PL/SQL trong chương trình
  [ EXCEPTION -- Phần xử lý ngoại lệ
    WHEN tên_ngoại_lệ THEN
      khối_lệnh ]
END;
```

Các tùy chọn IN, OUT, và IN OUT chỉ định cách mà tham số sẽ được sử dụng :

- Chế độ mặc định là IN. Có nghĩa là tham số có thể tham chiếu trong thân thủ tục nhưng không được phép thay đổi.
- OUT : Có thể gán giá trị cho tham số này trong thân thủ tục nhưng giá trị của nó không được phép tham chiếu đến (thường dùng cho kết quả trả về).
- IN OUT : Cho phép cả 2 chế độ trên.

Ví dụ: Viết một thủ tục để thực hiện việc tăng lương cho các nhân viên với các tham số đầu vào là mã phòng ban và phần trăm lương tăng thêm.

```
CREATE PROCEDURE raise_salary(dno NUMBER, percent NUMBER DEFAULT
0.5)
IS
BEGIN
  UPDATE scott.emp SET sal = sal * ((100 + percent)/100)
  WHERE deptno = dno;
```

```
COMMIT;  
END raise_salary;
```

10.1.2 Gọi thủ tục

Cú pháp gọi chương trình con trong PL/SQL. Nếu chương trình con bị gọi cùng schema với chương trình gọi thì có thể lược bỏ tên_schema . Trong trường hợp gọi chương trình con toàn cục thì cũng không cần tên_schema.

Nếu gọi thủ tục trong khối lệnh PL/SQL thì cú pháp là:

```
Tên_schema.tên_thủ_tục[(tham_số_1, tham_số_2,...)]
```

Nếu gọi thủ tục ngoài khối lệnh PL/SQL thì cú pháp là:

```
EXECUTE Tên_schema.tên_thủ_tục[(tham_số_1, tham_số_2,...)]
```

Ví dụ: Gọi thủ tục để tăng lương 10% cho các nhân viên làm việc ở phòng số 20.

```
EXECUTE raise_salary(20,10);
```

10.1.3 Xóa thủ tục

Cú pháp xóa thủ tục:

```
DROP PROCEDURE tên_thủ_tục
```

Ví dụ: DROP PROCEDURE raise_salary;

10.2 Hàm

Tương tự như thủ tục, hàm (function) cũng là nhóm các lệnh PL/SQL thực hiện chức năng nào đó. Khác với thủ tục, các hàm sẽ trả về một giá trị ngay tại lời gọi của nó.

10.2.1 Tạo một hàm

```
CREATE [OR REPLACE] FUNCTION tên_hàm  
[(tham_số_1> [IN | OUT | IN OUT] kiểu_dl  
[,tham_số_2 [IN | OUT | IN OUT] kiểu_dl [DEFAULT trị_mặc_định]])]  
RETURN kiểu_trả_về IS  
    [tên_biến kiểu_dl [NULL | NOT NULL] [DEFAULT trị_mặc_định];]  
BEGIN  
    [chuỗi_lệnh]  
    RETURN giá_trị_trả_về ;  
[EXCEPTION  
    WHEN tên_ngoại_lệ THEN  
        khối_lệnh ]  
END;
```

Ví dụ: Viết một hàm để trả về kết quả là năm của một ngày được truyền vào từ tham số.

```
CREATE OR REPLACE FUNCTION GETYEAR(pDate IN Date)  
    RETURN NUMBER IS  
BEGIN  
    RETURN EXTRACT(year from pDate);  
EXCEPTION  
    WHEN OTHERS THEN RETURN 0;  
END;
```

10.2.2 Gọi hàm

Do hàm có kết quả trả về nên việc gọi hàm phải thông qua một biến để lưu kết quả của hàm hoặc phải được đặt vào câu lệnh select giống như cách sử dụng các hàm của hệ thống cung cấp sẵn.

Ví dụ: Gọi thử hàm GETYEAR vừa tạo ở trên

```
SELECT hiredate, GETYEAR (hiredate) as HireYear  
FROM scott.emp;
```

10.2.3 Xóa hàm

```
DROP FUNCTION tên_hàm
```

10.2.4 Một số hạn chế khi sử dụng hàm và thủ tục

- Không cho phép hàm trả về kiểu dữ liệu như RECORD, TABLE.
- Các tham số của hàm (và cả thủ tục) chấp nhận mọi kiểu dữ liệu có trong Oracle. Tuy nhiên với kiểu char, varchar2, và kiểu number thì không chỉ định độ rộng. Ví dụ, *tham số có kiểu number(6) sẽ gây ra lỗi khi biên dịch và phải được sửa lại là number.*

10.2.5 Lợi ích của việc sử dụng hàm, thủ tục

- Nâng cao hiệu suất: Tránh việc tái sử dụng các câu lệnh nhiều lần bởi nhiều User khác nhau. Giảm thiểu thời gian biên dịch câu lệnh PL/SQL trong pha phân tích câu lệnh. Giảm thiểu số lần gọi lệnh thực hiện trên CSDL, từ đó, làm giảm lưu lượng truyền thông trên mạng.
- Nâng cao khả năng bảo trì: Ta có thể dễ dàng sửa nội dung bên trong các hàm, thủ tục mà không ảnh hưởng đến việc giao tiếp của chúng (các tham số và lời gọi vẫn y nguyên). Thay đổi nội dung của một hàm, hay thủ tục có thể ứng dụng được ngay cho nhiều user khác nhau.
- Tăng tính bảo mật và toàn vẹn của dữ liệu: Với việc điều khiển truy nhập dữ liệu gián tiếp đối với các đối tượng trong CSDL sẽ làm nâng cao tính bảo mật của dữ liệu.

10.3 Trigger

Database trigger là những thủ tục được thực hiện ngầm định ngay khi thực hiện các lệnh DML như INSERT, DELETE, UPDATE nhằm đảm bảo tính an toàn của dữ liệu. Thiết kế các database trigger nên chú ý các vấn đề sau:

- Chỉ sử dụng database trigger đối với các thao tác trọng tâm. Không sử dụng database trigger để thực hiện các ràng buộc sẵn có trong CSDL Oracle. Ví dụ: dùng database trigger để thay thế cho các constraint.
- Sử dụng database trigger có thể gây rối, khó khăn cho việc bảo trì và phát triển hệ thống lớn. Vì thế, ta chỉ sử dụng database trigger khi thật cần thiết.

Khi tạo database trigger, ta cần xác định một số vấn đề như:

- Thời điểm thực hiện: BEFORE, AFTER.
- Hành động thực hiện: INSERT, UPDATE, DELETE.
- Đối tượng tác động: bảng dữ liệu nào.

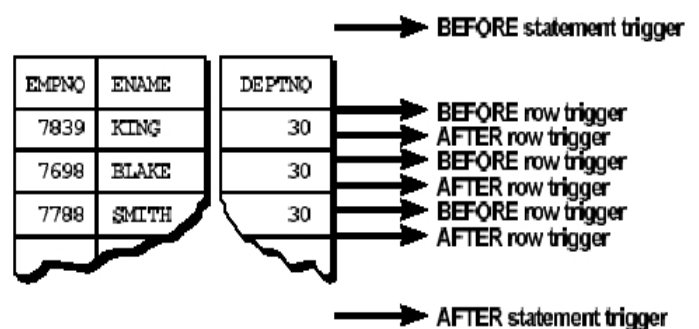
- Mức kích hoạt: kích hoạt ở mức dòng lệnh hay mức dòng dữ liệu.
- Mệnh đề điều kiện thực hiện.
- Nội dung của trigger.

10.3.1 Phân loại TRIGGER

- Ta có thể phân loại trigger theo thời gian thực hiện như: **BEFORE** và **AFTER**.
 - **BEFORE trigger**: được kích hoạt trước khi thực hiện câu lệnh. Việc này có thể cho phép ta loại bớt các phép xử lý không cần thiết, thậm chí có thể rollback dữ liệu trong trường hợp có thể gây ra các ngoại lệ (exception). Trigger thuộc loại này thường được sử dụng đối với các thao tác INSERT hoặc UPDATE.
 - **AFTER trigger**: câu lệnh được thực hiện xong thì trigger mới được kích hoạt. Thực hiện các công việc thường phải làm sau khi đã thực hiện câu lệnh.
 - **INSTEAD OF trigger**: Loại trigger này cho phép người sử dụng có thể thay đổi một cách trong suốt dữ liệu của một số view mà không thể thực hiện thay đổi trực tiếp được. Với INSTEAD OF trigger, ta có thể thực hiện với cả ba thao tác: insert, update, delete trên view
- Ta cũng có thể phân loại trigger theo loại câu lệnh kích hoạt như: **INSERT**, **UPDATE**, **DELETE**.

Trong các trigger thuộc một trong ba loại lệnh: INSERT, UPDATE, DELETE, thì trigger UPDATE cần phải chỉ rõ thêm tên cột dữ liệu kích hoạt trigger mỗi khi giá trị dữ liệu trong đó bị thay đổi. Bên trong Trigger có thể có chứa các lệnh thao tác dữ liệu. Do đó, cần phải tránh trường hợp lặp lại theo kiểu đệ quy.

- Một cách khác ta cũng có thể phân loại trigger theo mức kích hoạt.
 - **Trigger mức câu lệnh**: Trigger được kích hoạt mỗi khi thực hiện câu lệnh.
 - **Trigger mức dòng dữ liệu**: Trigger được kích hoạt nhiều lần ứng với mỗi dòng dữ liệu chịu ảnh hưởng bởi thao tác thực hiện lệnh



Hình 12. Các sự kiện của Trigger

10.3.2 Tạo Trigger

- Cú pháp lệnh tạo trigger mức câu lệnh

```
CREATE [OR REPLACE] TRIGGER tên_trigger
    thời_điểm sự_kiện_1 [OR sự_kiện_2 OR sự_kiện_3] ON
tên_bảng
BEGIN
    Khối_lệnh;
END;
```

- Cú pháp lệnh tạo trigger mức dòng dữ liệu

```
CREATE [OR REPLACE] TRIGGER tên_trigger
    thời_điểm sự_kiện_1 [OR sự_kiện_2 OR sự_kiện_3] ON
tên_bảng
    [REFERENCING OLD AS dòng_cũ | NEW AS dòng_mới] FOR EACH
ROW
    [WHEN điều_kiện]
BEGIN
    khối_lệnh;
END;
```

Trong đó:

trigger_name	Tên trigger
thời_điểm AFTER (sau).	Thời điểm kích hoạt trigger: BEFORE (trước),
sự_kiện UPDATE.	Loại câu lệnh kích hoạt trigger: INSERT, DELETE,
dòng_cũ, dòng_mới	Tên biến thay thế cho dòng dữ liệu trước và sau khi
FOR EACH ROW	thay đổi của dòng dữ liệu đang xử lý.
dữ liệu.	Trigger thuộc loại được kích hoạt ứng với mỗi dòng
WHEN	Chỉ ra một số điều kiện ràng buộc thực hiện trigger
tên_bảng	(chỉ sử dụng kết hợp với FOR EACH ROW).
khối_lệnh	Tên bảng dữ liệu sẽ gắn trigger trên đó.
	Nội dung khối lệnh SQL và PL/SQL trong trigger.

Chú ý :

- Chỉ có trigger mức dòng dữ liệu mới có thể truy cập giá trị của các thuộc tính trong mỗi dòng.
 - Với update trigger, ta có thể lấy giá trị cũ của thuộc tính (giá trị trước khi cập nhật) bằng cú pháp **:old.tên_cột** và giá trị mới của thuộc tính (giá trị sau khi cập nhật) bằng cú pháp **:new.tên_cột**
 - Với insert trigger, chỉ có **:new.tên_cột** được sử dụng.
 - Với delete trigger chỉ có **:old.tên_cột** được sử dụng. Nó lưu giữ giá trị của thuộc tính trước khi bị xóa.

- Tất cả câu lệnh SQL và PL/SQL có thể sử dụng trong trigger ngoại trừ commit và rollback
- Raise_application_error là hàm cho phép trigger kết thúc thực thi và trả về thông báo lỗi được cấp trong tham số gồm mã lỗi (là một số từ -20000 đến -20999, các số khác Oracle dành riêng để sử dụng), và một chuỗi thông báo.
- Mệnh đề IF được dùng để kiểm tra thao tác nào đã xảy ra như dưới đây:

```
CREATE OR REPLACE TRIGGER tên_trigger
  AFTER INSERT OR DELETE OR UPDATE ON tên_bảng
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    khối_lệnh
  END IF ;
  IF UPDATING THEN
    khối_lệnh
  END IF ;
  IF DELETING THEN
    khối_lệnh
  END IF ;
END;
```

Ví dụ : Viết một trigger để kiểm tra rằng khi người dùng nhập mới hoặc cập nhật trên bảng Emp thì phải thỏa điều kiện :

- Lương (Sal) phải nhiều hơn thưởng (Comm)
- Lương mới phải cao hơn lương cũ
- Không được tăng lương quá 10%

```
CREATE OR REPLACE TRIGGER check_sal_comm_EMP
  AFTER INSERT OR UPDATE OF SAL, Comm on EMP
  FOR EACH ROW
BEGIN
  -- Lương (Sal) phải nhiều hơn thưởng (Comm)
  IF (:new.SAL < :new.Comm) THEN
    raise_application_error(-20225, 'Lương phải lon hơn
    thưởng');
  --Lương mới phải cao hơn lương cũ
  ELSIF (:new.SAL < :old.SAL) THEN
    raise_application_error(-20230, 'Lương moi phai cao hơn lương
    cu');
  --Không được tăng lương quá 10%
  ELSIF (:new.SAL > 1.1 * :old.SAL) THEN
    raise_application_error(-20235, 'Khong duoc tang qua
    10%');
  END IF;
END;
```

10.3.3 Xóa Trigger

```
DROP TRIGGER tên_trigger;
```

Chương 3. Quản lý giao dịch & phục hồi

Trong chương này, các mục 1, 2 và 3 sẽ trình bày lý thuyết chung về quản lý giao dịch, cạnh tranh và phục hồi trong các HQTCSDL. Từ mục 4 trở đi sẽ minh họa các lý thuyết này bằng công cụ của Hệ QTCSDL Oracle.

1 Giao dịch (transaction)

1.1 Định nghĩa Giao dịch:

Giao dịch (GD) là một hành động hay một chuỗi các hành động được thực hiện bởi 1 người dùng hoặc 1 chương trình ứng dụng, trong đó có truy cập hoặc thay đổi nội dung của một CSDL.

Một GD 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.

Ta xét một ví dụ: Một hệ thống ngân hàng gồm một số tài khoản và một tập các giao dịch 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 hạng mục dữ liệu X từ CSDL đến buffer của giao dịch thực hiện hoạt động READ này.

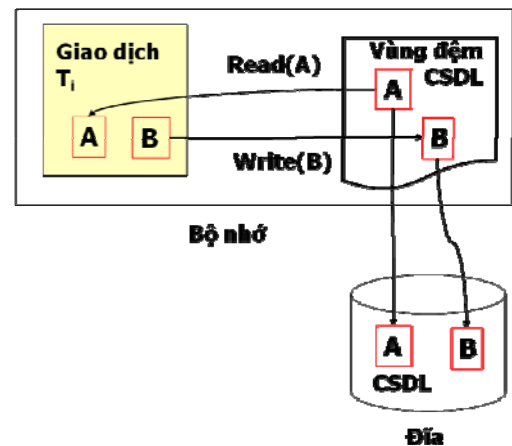
WRITE(X). chuyển hạng mục dữ liệu X từ buffer của giao dịch thực hiện WRITE đến CSDL.

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 : **READ(A);** T_i là một giao
 A:=A - 50; dịch chuyển 50 từ tài
 WRITE(A) khoản A sang tài khoản
 READ(B); B. Giao dịch này có thể
 B:=B + 50; được xác định như sau:
 WRITE(B);

Nếu tất cả các cập nhật này không được thực hiện, CSDL sẽ ở trong trạng thái không nhất quán: nghĩa là tổng số tiền A + B có thể bị thay đổi.

Một GD sẽ luôn luôn chuyển CSDL từ một trạng thái nhất quán này sang một trạng thái nhất quán khác, mặc dù chúng ta chấp nhận sự nhất quán có thể bị phá vỡ trong khi GD đang được thực hiện. Chẳng hạn như khi GD đang thực hiện, thì có những thời điểm mà tổng A + B thay đổi. Tuy nhiên, khi GD kết thúc thì tổng A + B phải như cũ.



Hình 13. Tổ chức lưu trữ trong quá trình thực hiện giao dịch cập nhật CSDL

1.2 Trạng thái của Giao dịch:

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

- **Hoạt động (Active)** Trạng thái khởi đầu; giao dịch giữ trong trạng thái này trong khi nó đang thực hiện.
- **Hoàn tất một phần (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.

Và kết thúc với một trong hai trạng thái:

- **Hủy bỏ (Aborted)** Sau khi giao dịch đã bị cuộn lại (rolled back) và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao dịch.

Một GD bị hủy có thể được khởi động lại sau đó, tùy thuộc vào nguyên nhân gây lỗi, và có thể nó sẽ thực thi thành công và hoàn tất.

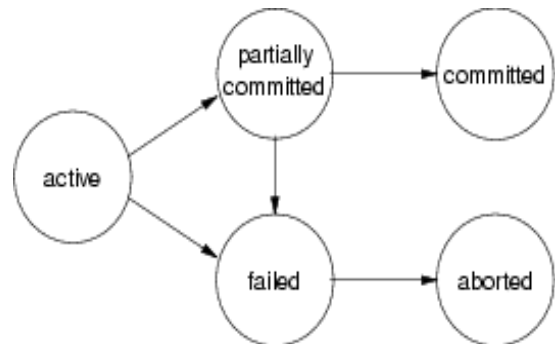
- **Hoàn tất (Committed)** Sau khi hoàn thành thành công giao dịch CSDL đạt tới trạng thái nhất quán mới.

Một GD hoàn tất thì không thể bị hủy bỏ. Nếu ta thấy việc thực hiện GD đã hoàn tất là một sai lầm, thì ta phải thực hiện một GD khác có hiệu ứng ngược lại, gọi là *GD bù* (compensating transaction).

Một HQTCSDL không có một cách cố hữu nào để biết được những cập nhật nào phải được nhóm lại thành một GD. Vì vậy, nó phải cung cấp một phương pháp để giúp dùng chỉ ra ranh giới của một GD. Các từ khóa BEGIN TRANSACTION, COMMIT, ROLLBACK được hầu hết các HQTCSDL đưa vào ngôn ngữ thao tác dữ liệu để phân định các GD. Nếu không có các từ này, thì toàn bộ chương trình sẽ được xem như một GD, khi đó HQTCSDL sẽ tự động thực hiện hành động COMMIT khi chương trình kết thúc đúng đắn và thực hiện ROLLBACK trong trường hợp ngược lại.

Một GD ở trạng thái hoàn tất một phần đã hoàn thành sự thực hiện của nó, nhưng nó vẫn có thể bị hủy bỏ, vì kết quả hiện tại vẫn có thể còn lưu trữ tạm thời trong bộ nhớ chính và như thế một sự cố phần cứng vẫn có thể ngăn cản sự hoàn tất của giao dịch. 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 dịch, khi hệ thống tái khởi động sau sự cố. Sau khi các thông tin sau cùng này được viết lên đĩa, giao dịch chuyển sang trạng thái committed.

Với giả thiết sự cố hệ thống không gây ra sự mất dữ liệu trên đĩa, một giao dịch đi vào trạng thái Failed nếu hệ thống xác định rằng giao dịch không thể tiến triển bình thường được nữa (do lỗi phần cứng hoặc phần mềm). Như vậy, giao dịch phải được cuộn lại rồi chuyển sang trạng thái hủy bỏ.



Hình 14. Biểu đồ trạng thái tương ứng với một

1.3 Các thuộc tính của một GD

Tất cả các GD đều phải có 4 thuộc tính sau, viết tắt là ACID:

a. Tính nguyên tử (Atomicity) hay tính 'Tất cả hoặc không có gì'

Một GD là một đơn vị không phân chia được nghĩa là hoặc nó được thực hiện toàn bộ hoặc không thực hiện tí gì.

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 mà giao dịch đang thực hiện hành động ghi, nếu giao dịch không hoàn tất, giá trị cũ sẽ được khôi phục để đặt trạng thái của hệ thống trở lại trạng thái trước khi giao dịch diễn ra.

Trách nhiệm: Đảm bảo tính nguyên tử là trách nhiệm của hệ quản trị CSDL, và được quản lý bởi một thành phần được gọi là thành phần quản lý giao dịch (transaction-management component).

b. Tính nhất quán (consistency)

Một GD phải chuyển CSDL từ một trạng thái nhất quán này sang một trạng thái nhất quán khác. Tính nhất quán của CSDL yêu cầu chỉ có những dữ liệu hợp lệ, chính xác (nghĩa là không vi phạm các ràng buộc dữ liệu) mới được ghi vào CSDL.

Chẳng hạn như trong ví dụ trên thì tổng của A và B phải không thay đổi bởi sự thực hiện giao dịch. Nếu không có yêu cầu nhất quán, tiền có thể được tạo ra hay bị phá hủy bởi giao dịch. Dễ dàng kiểm nghiệm rằng nếu CSDL nhất quán trước một thực hiện giao dịch, nó vẫn nhất quán sau khi thực hiện giao dịch.

Trách nhiệm: đảm bảo tính nhất quán cho một giao dịch là trách nhiệm của người lập trình ứng dụng hay người đã viết ra giao dịch. Nhiệm vụ này có thể thực hiện nhờ các chức năng kiểm tra tự động các ràng buộc toàn vẹn của hệ QTCSDDL.

c. Tính cô lập (Isolation)

Các GD phải thực hiện độc lập nhau. Hay nói cách khác, các kết quả giữa chừng của một GD chưa hoàn tất thì không thể để các GD khác nhìn thấy.

Tính cô lập của một giao dịch đảm bảo rằng cho dù nhiều giao dịch được thực hiện cùng lúc cũng sẽ dẫn hệ thống đến một trạng thái tương đương với trạng thái có được bằng cách thực hiện các giao dịch này một cách tuần tự.

Trách nhiệm: Đảm bảo tính cô lập là trách nhiệm của một thành phần của hệ QTCSDDL được gọi là thành phần quản lý cạnh tranh (concurrency-control component).

d. Tính bền vững (Duration)

Tính chất bền vững đảm bảo rằng mỗi khi một GD hoàn tất, tất cả các cập nhật đã thực hiện trên CSDL vẫn còn đó, ngay cả khi xảy ra sự cố hệ thống sau khi giao dịch 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 dịch đã được viết lên đĩa trước khi giao dịch kết thúc,

Hoặc thông tin về sự cập nhật được thực hiện bởi giao dịch được viết lên đĩa đủ cho phép CSDL xây dựng lại các cập nhật khi hệ CSDL được khởi động lại sau sự cố.

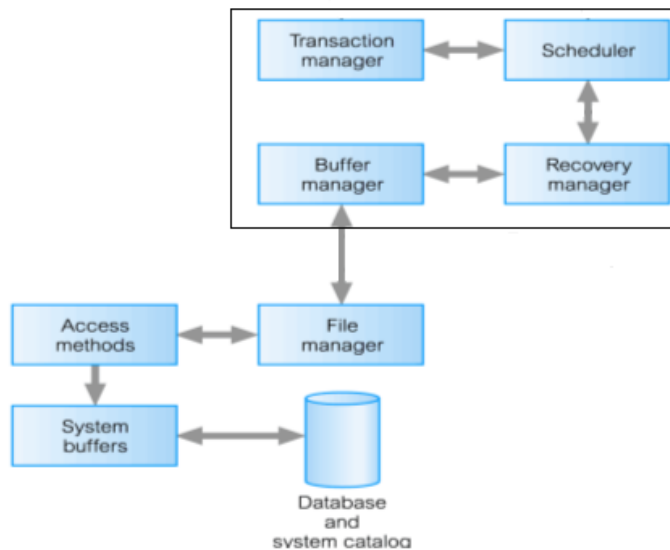
Trách nhiệm: Đảm bảo tính bền vững là trách nhiệm của một thành phần của hệ QTCSDL gọi là *thành phần quản lý phục hồi* (recovery-management component). Hai thành phần quản lý giao dịch và quản lý phục hồi quan hệ mật thiết với nhau.

Hình 15 được trích ra từ kiến trúc tổng quát của hệ QTCSDL gồm bốn thành phần vừa đề cập: quản lý GD, điều khiển cạnh tranh, quản lý phục hồi và quản lý vùng đệm.

Quản lý GD (transaction manager) sẽ phối hợp các GD thay cho các chương trình ứng dụng. Nó giao tiếp với *bộ lập lịch trình* (scheduler), là một module chịu trách nhiệm về việc thực hiện một chiến lược cụ thể để điều khiển sự thực hiện cạnh tranh

của các GD. Bộ lập lịch trình đôi khi còn được gọi là *bộ quản lý khóa chốt* (lock manager) nếu giao thức điều khiển cạnh tranh là dựa trên khóa chốt. Mục tiêu của bộ lập lịch trình là tối đa hóa khả năng cạnh tranh nhưng không cho phép các GD đang thực hiện đồng thời can thiệp lẫn nhau, và nhờ đó đảm bảo tính toàn vẹn và nhất quán cho CSDL.

Nếu có lỗi xảy ra trong khi một GD đang thực hiện, thì CSDL có thể ở trạng thái không nhất quán. Nhiệm vụ của *bộ quản lý phục hồi* (recovery manager) là đảm



Hình 15. Các thành phần liên quan việc quản lý GD trong HQTCSDL

bảo cho CSDL được phục hồi về trạng thái nhất quán trước khi bắt đầu GD. Cuối cùng, bộ quản lý vùng đệm (buffer manager) chịu trách nhiệm chuyển dữ liệu qua lại giữa bộ lưu trữ ngoài và bộ nhớ trong.

2 Điều khiển cạnh tranh (Concurrency control)

Quản lý cạnh tranh là quá trình quản lý các thao tác đang diễn ra đồng thời trên CSDL mà không cho phép chúng can thiệp lẫn nhau.

2.1 Sự cần thiết phải có quản lý cạnh tranh

Một mục tiêu chính của việc phát triển CSDL là cho phép người dùng truy cập vào các dữ liệu chia sẻ một cách đồng thời hay cạnh tranh.

Các truy cập cạnh tranh thì khá dễ dàng nếu tất cả các người dùng chỉ đọc dữ liệu, vì không có cách nào họ có thể can thiệp lẫn nhau; hoặc hai hay nhiều người dùng đang cập nhật các dữ liệu khác nhau.

Nhưng khi hai hay nhiều người dùng đang truy cập một cách đồng thời vào một dữ liệu và có ít nhất một người cập nhật dữ liệu, thì sẽ có khả năng xảy ra sự can thiệp

dẫn đến sự không nhất quán. Tuy nhiên, không phải vì thế mà ta buộc các giao dịch này không được thực hiện song song, hay phải xếp hàng chờ vì nhiều lợi ích:

Trước hết, một giao dịch gồm nhiều bước. Một vài bước liên quan tới hoạt động I/O; các bước khác liên quan đến hoạt động CPU. Ta có thể tận dụng sự hoạt động song song của hệ thống CPU và I/O để chạy nhiều giao dịch song song nhờ *tính đa chương* (multiprogramming). Chẳng hạn hai GD có thể thực hiện đồng thời. Hệ thống bắt đầu với GD thứ nhất cho đến khi nó đạt đến một thao tác nhập/xuất (I/O). Trong khi thao tác I/O đang được thực hiện, CPU tạm hoãn GD thứ nhất và thực thi lệnh trong GD thứ hai. Khi GD thứ hai đạt đến một thao tác I/O thì điều khiển sẽ quay trở lại với GD thứ nhất và các thao tác của GD này sẽ được tiếp tục lại từ điểm mà nó đã tạm hoãn. GD thứ nhất sẽ tiếp tục cho đến khi nó đạt đến một thao tác I/O khác. Cứ như thế, các thao tác trong hai GD xen kẽ với nhau để đạt được sự thực hiện cạnh tranh. Như vậy sẽ tăng *thông lượng* (throughput) của hệ thống có nghĩa là tăng số lượng giao dịch có thể được thực hiện trong một khoảng thời gian đã cho, cũng có nghĩa là hiệu suất sử dụng bộ xử lý và đĩa tăng lên.

Ngoài ra, có thể có sự trộn lẫn các giao dịch đang chạy trong hệ thống, cái thì dài cái thì ngắn. Nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến khi hoàn tất, điều đó dẫn đến một sự trì hoãn không lường trước được trong việc chạy một giao dịch. Nếu các giao dịch đang hoạt động trên các phần khác nhau của CSDL, sẽ tốt hơn nếu ta cho chúng chạy 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 dịch, đồng thời làm giảm *thời gian đáp ứng trung bình* là thời gian để một giao dịch được hoàn tất sau khi đã được đệ trình.

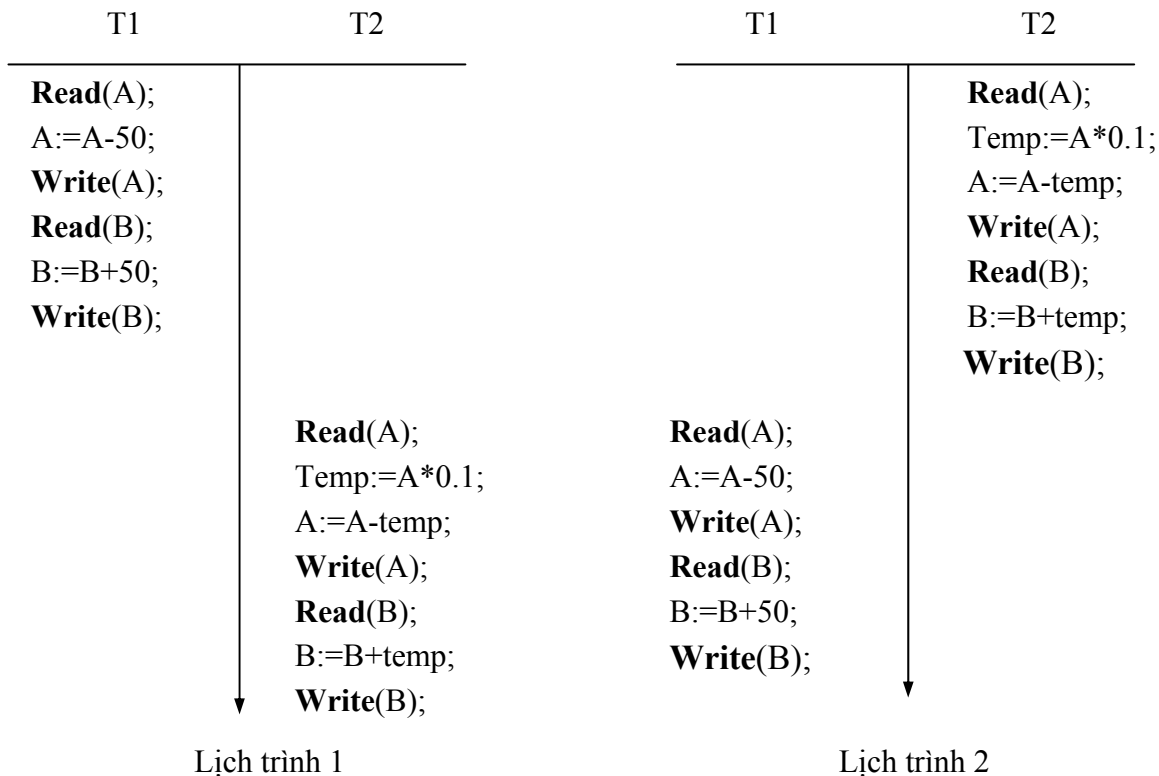
Tuy nhiên, khi thực hiện song song, mặc dù bản thân hai GD có thể hoàn toàn đúng, sự đan xen các thao tác như mô tả có thể sinh ra các kết quả không đúng, vì vậy làm ảnh hưởng đến tính toàn vẹn và tính nhất quán của CSDL. Chúng ta sẽ xem xét ba ví dụ về các vấn đề tiềm ẩn có thể gây ra bởi sự cạnh tranh:

- Vấn đề về mất dữ liệu đã cập nhật (lost update),
- Vấn đề về sự phụ thuộc vào các GD không hoàn tất (uncommitted dependency),
- Vấn đề phân tích không nhất quán (inconsistent analysis).

T_1 :	Read(A); A:=A-50; Write(A); Read(B); B:=B+50; Write(B);	T_2 :	Read(A); Temp:=A*0.1; A:=A-temp; Write(A); Read(B); B:=B+temp; Write(B);
---------	--	---------	--

Bảng 11. Lịch trình cạnh tranh

Để minh họa các vấn đề này, ta xét hệ thống ngân hàng đơn giản với một số tài khoản và một tập hợp các giao dịch truy xuất, cập nhật các tài khoản này. Giả sử T_1 và T_2 là hai giao dịch chuyển khoản từ một tài khoản sang một tài khoản khác: T_1 chuyển 50\$ từ tài khoản A sang tài khoản B. T_2 chuyển 10% số dư từ tài khoản A sang tài khoản B.



Hình 16. Các lịch trình tuần tự

Giả sử giá trị hiện tại của A và B tương ứng là 1000\$ và 2000\$. Giả sử rằng hai giao dịch này được thực hiện tuần tự theo thứ tự T1 rồi tới T2 như trong lịch trình 1. Như vậy, dãy thực hiện này là như hình 16, trong đó dãy các bước chỉ thị theo thứ tự từ trên xuống: Giá trị sau cùng của các tài khoản A và B, sau khi thực hiện dãy các chỉ thị theo trình tự này tương ứng là 855\$ và 2145\$. Như vậy, tổng giá trị của hai tài khoản này (A + B) được bảo tồn sau khi thực hiện cả hai giao dịch. Tương tự, nếu hai giao dịch được thực hiện tuần tự theo trình tự T2 rồi đến T1 như trong lịch trình 2. Khi đó các giá trị cuối cùng của tài khoản A và B tương ứng sẽ là 850\$ và 2150\$.

2.2 Lịch trình (schedule):

Lịch trình là một chuỗi các thao tác thực hiện bởi một tập hợp các GD cạnh tranh mà vẫn đảm bảo thứ tự của các thao tác trong từng GD đơn lẻ.

Ví dụ, đối với giao dịch T1, chỉ thị Write(A) phải xuất hiện trước chỉ thị Read(B), trong bất kỳ lịch trình hợp lệ nào.

Lịch trình tuần tự (serial schedule) là một lịch trình trong đó các thao tác của một GD được thực hiện liên tiếp nhau, không có bất kỳ thao tác nào của các GD khác xen vào giữa. Các lịch trình trong hình 16 là tuần tự.

Trong lịch trình tuần tự, ta sẽ không có các giao dịch đan xen lẫn nhau, vì chỉ có một giao dịch thực hiện tại một lúc. Tuy nhiên, không có bất kỳ sự đảm bảo nào về kết quả của tất cả các lịch trình tuần tự đều sẽ giống nhau cho một tập hợp các giao dịch. Chẳng hạn như hai lịch trình 1 và 2 trong hình 16 cho kết quả hoàn toàn khác nhau.

Lịch trình không tuần tự (nonserial schedule) là một lịch trình trong đó các thao tác từ một tập hợp các giao dịch cạnh tranh đan xen lẫn nhau. Khi đó, các lịch trình

này có thể đưa CSDL về tình trạng không nhất quán hay các kết quả không chính xác như phân tích trong các phần kế tiếp.

2.2.1 Vấn đề 1: Mất dữ liệu đã cập nhật

Tình huống: Kết quả của một thao tác cập nhật hoàn tất một cách thành công bởi một người dùng có thể bị ghi chồng lên bởi một người dùng khác.

Ví dụ: Trong bảng 12, GD T1 rút khỏi TK A 10\$ đang thực hiện cạnh tranh với GD T2 nạp vào TK A 100\$. Giả sử ban đầu A có 100 đồng.

Nếu các GD này được thực hiện một cách tuần tự (serrially), cái này xong rồi đến cái kia, không có các thao tác xen kẽ, thì kết quả cuối cùng tài khoản này sẽ còn 190 đồng, bất kể là GD nào được thực hiện trước. Nhưng, nếu T1 và T2 thực hiện cạnh tranh theo lịch trình sau:

T₁	T₂	TK A
	Begin Transaction	100
Begin Transaction	x:=Read(A);	100
x := Read(A);		100
	x = x + 100 ;	100
x:=x-10;		100
	Write(A);	200
Write(A);		90
	Commit	90
Commit		90

Bảng 12. Lịch trình gây mất dữ liệu cập nhật

Sau khi thực hiện lịch trình 3, ta đạt tới trạng thái trong đó giá trị cuối của A 90\$ (lẽ ra phải là 190). Ở đây, ta thấy giá trị mà T2 đã cập nhật trên A đã bị T1 ghi chồng lên.

Giải pháp: Việc đánh mất dữ liệu đã cập nhật của một GD có thể tránh được bằng cách ngăn không cho một GD khác đọc giá trị của mục dữ liệu đang được cập nhật cho đến khi việc cập nhật hoàn tất.

2.2.2 Vấn đề 2: phụ thuộc vào các GD không hoàn tất

Tình huống: Vấn đề về sự phụ thuộc vào các GD không hoàn tất xảy ra khi một GD cho phép các GD khác nhìn thấy các kết quả tạm thời trước khi nó hoàn tất.

Ví dụ: GD T4 cập nhật TK A thành 200 đồng, nhưng sau đó nó bị hủy bỏ GD, và vì vậy A sẽ được phục hồi về giá trị ban đầu của nó là 100 đồng. Tuy nhiên, trong lúc này, GD T3 đã đọc giá trị mới của T4 (200 đồng), và sử dụng giá trị này làm cơ sở cho việc trừ 10 đồng khỏi tài khoản, làm cho giá trị mới không đúng là 190 đồng, thay vì chỉ là 90 đồng.

T ₃	T ₄	TK A
	Begin	100
	Read(A);	100
	A = A +	100
Begin	Write(A);	200
Read(A);	:	200
A:=A-10;	Rollback	100
Write(A);		190
Commit		190

Bảng 13. Lịch trình phụ thuộc vào GD không hoàn tất

Giải pháp: Vấn đề này có thể tránh được bằng cách ngăn không cho một GD đọc giá trị đang được cập nhật bởi một giao dịch chưa kết thúc khác (nghĩa là chưa hoàn tất hoặc chưa hủy bỏ).

Hai vấn đề trên tập trung vào các GD cập nhật CSDL và sự can thiệp lẫn nhau của chúng có thể làm cho CSDL bị sai. Tuy nhiên, các GD chỉ đọc CSDL cũng có thể sai.

2.2.3 Vấn đề 3: Phân tích không nhất quán:

Tình huống: GD chỉ đọc được phép đọc một phần kết quả của các GD chưa hoàn tất mà các GD này cùng lúc cũng đang cập nhật CSDL. Điều này gọi là dirty read hay unrepeatable read.

Ví dụ: GD tính tổng T6 đang thực hiện cạnh tranh với GD T5: GD T6 cộng giá trị của các tài khoản X (100 đồng), tài khoản Y (50 đồng) và tài khoản Z (25 đồng). Kết quả đúng sẽ là 175 đồng. GD T5 chuyển 10 đồng từ X sang Z

Với lịch trình như bảng 14, ta thấy T6 bây giờ có kết quả sai (nhiều hơn 10 đồng).

T ₅	T ₆	X	Y	Z	Sum
	Begin Transaction	100	50	25	
Begin Transaction	Sum = 0	100	50	25	0
Read(X);		100	50	25	0
X = X -	Read(X);	100	50	25	100
Write (X);	Sum = Sum +	90	50	25	100
	Read(Y);	90	50	25	150
Read (Z)	Sum = Sum +	90	50	25	150
Z = Z + 10		90	50	25	150
Write(Z);		90	50	35	150
Commit	Read(Z)	90	50	35	185
	Sum = Sum + Z	90	50	35	185
	Commit	90	50	35	185

Bảng 14. Lịch trình phân tích không nhất quán.

Giải pháp: Vấn đề này có thể tránh được bằng cách ngăn không cho một GD đọc giá trị đang được cập nhật bởi một GD chưa hoàn tất khác.

2.3 Tính khả tuần tự của một lịch trình

Các vấn đề chủ yếu nảy sinh khi cho phép các GD thực hiện cạnh tranh đã được minh họa như trên. Mục tiêu chính của giao thức điều khiển cạnh tranh là lập lịch trình cho các GD sao cho có thể thực hiện cạnh tranh nhưng vẫn tránh được các sự can thiệp lẫn nhau có thể dẫn đến kết quả sai.

Ba vấn đề vừa mô tả trên là kết quả của sự quản lý cạnh tranh sai, dẫn CSDL đến tình trạng không nhất quán như trong tình huống đầu, và trình bày cho người dùng một kết quả sai như trong tình huống thứ 3. Sự thực hiện tuần tự sẽ ngăn không cho xuất hiện các vấn đề này. Dù cho lịch trình tuần tự nào được chọn, thì sự thực hiện tuần tự cũng không bao giờ làm cho CSDL ở tình trạng không nhất quán.

Lịch trình khả tuần tự (serializable schedule): là các lịch trình không tuần tự cho phép các GD thực hiện cạnh tranh nhưng vẫn cho kết quả giống như lịch trình tuần tự.

Trong thực tế, các HQTCSDL không kiểm tra tính khả tuần tự của một lịch trình. Vì việc kiểm tra này có thể là không thực tế, do sự xen kẽ của các thao tác trong các giao dịch được xác định bởi hệ điều hành chứ không phải là HQTCSDL. Thay vào đó, tiếp cận thường dùng là sử dụng các giao thức (protocols) đã biết để tạo ra các lịch trình khả tuần tự. Hai trong số các giao thức này sẽ được trình bày trong phần 2.5

2.4 Tính khả phục hồi của lịch trình

Tính khả tuần tự nhận định các lịch trình duy trì tính nhất quán của CSDL, với giả thiết là không có GD nào trong lịch trình bị thất bại. Một khía cạnh khác cần kiểm tra là tính khả phục hồi của các giao dịch trong lịch trình.

Nếu một GD thất bại, tính nguyên tử của GD yêu cầu chúng ta phải xóa bỏ các tác dụng của giao dịch đó. Ngoài ra, tính bền vững phát biểu rằng, một khi GD hoàn tất, các thay đổi của nó không thể được hủy bỏ.

T_9	T_{10}
Begin Transaction	Begin Transaction
Read(X)	Read(X)
$X = X + 100$	$X = X * 1.1$
Write (X)	Write (X)
	Read(Y)
	$Y = Y * 1.1$
	Write (Y)
Read(Y);	Commit
$Y = Y - 100$	
Write (Y);	
Rollback	

Bảng 15. Lịch trình không khả phục hồi

Hãy xem xét hai GD trong bảng 15, ở cuối GD T_9 , T_9 quyết định cuộn ngược tác dụng của nó. T_{10} đã đọc giá trị cập nhật trên X bởi T_9 , sau đó cập nhật X và commit các thay đổi. Nói đúng ra, khi hủy bỏ T_9 , ta cần phải hủy bỏ T_{10} vì nó đã sử dụng giá trị X đã bị hủy. Tuy nhiên, tính bền vững không cho phép điều này. Nói cách

khác, lịch trình này là không thể phục hồi (unrecoverable schedule), các lịch trình như vậy nên cấm thực hiện. Điều này dẫn đến định nghĩa về lịch trình khả phục hồi:

Lịch trình khả phục hồi (recoverable schedule) là một lịch trình trong đó với mỗi cặp GD T_i và T_j , nếu T_j đọc một mục dữ liệu mà trước đó đã được ghi bởi T_i , thì thao tác commit của T_i phải thực hiện trước thao tác commit của T_j .

2.5 Các kỹ thuật quản lý cạnh tranh

Tính khả tuần tự có thể đạt được bằng một số cách. Có hai kỹ thuật điều khiển cạnh tranh cơ bản cho phép các GD thực hiện song song một cách an toàn dựa trên một số ràng buộc nào đó. Hai phương pháp đó là: khóa chốt và nhãn thời gian.

Một cách cơ bản, khóa chốt và nhãn thời gian là các *tiếp cận thận trọng* (hay còn gọi là bi quan) vì các phương pháp này làm cho các GD bị ngưng trệ khi chúng xung đột với các giao dịch khác tại một thời điểm nào đó trong tương lai. Các *phương pháp lạc quan*, sẽ trình bày trong mục 2.6, thì dựa trên lập luận rằng các xung đột thì hiếm khi nên chúng cho phép các GD được tiến hành không đồng bộ và chỉ kiểm tra xung đột vào lúc cuối, khi giao dịch hoàn tất.

2.5.1 Khóa chốt (locking)

Khóa chốt là một thủ tục được sử dụng để điều khiển truy cập cạnh tranh đến dữ liệu. Khi một GD đang truy cập vào CSDL, một khóa chốt có thể từ chối truy cập đến từ các GD khác để nhằm tránh các kết quả không đúng.

Các phương pháp khóa chốt được sử dụng phổ biến nhất để bảo đảm tính khả tuần tự của các GD cạnh tranh. Có một vài sự khác biệt, nhưng tất cả các phương pháp này đều có chung một đặc điểm cơ bản, đó là một GD phải yêu cầu một khóa đọc (còn gọi là khóa chia sẻ - shared lock) hay khóa ghi (còn gọi là khóa độc quyền – exclusive lock) trên một mục dữ liệu trước khi thực hiện một thao tác đọc hoặc ghi trên CSDL tương ứng.

Khóa chốt ngăn không cho các giao dịch khác thay đổi mục dữ liệu hoặc thậm chí là đọc nó, trong trường hợp là khóa ghi.

Các mục dữ liệu với nhiều kích cỡ khác nhau, lớn như toàn bộ CSDL đến nhỏ như một trường, đều có thể bị khóa. Kích cỡ của các mục dữ liệu sẽ xác định tính tinh vi hay độ mịn (granularity) của khóa. Khóa thật sự có thể được cài đặt bằng một bit trên mục dữ liệu chỉ định rằng một phần của CSDL đã bị khóa, hay bằng cách giữ một danh sách các phần bị khóa của CSDL, hay bằng các phương tiện khác. Chúng ta sẽ xem xét độ mịn của khóa trong phần 2.7.

Để hiểu các quy luật cơ bản của khóa chốt, ta cần hiểu các khái niệm sau:

Khóa đọc (read lock): Nếu một GD có một khóa đọc trên một mục dữ liệu, nó có thể đọc nhưng không thể cập nhật mục dữ liệu đó.

Khóa ghi (write lock) Nếu một GD có một khóa ghi trên một mục dữ liệu, nó có thể đọc và cập nhật mục dữ liệu đó.

Bởi vì các thao tác đọc không thể xung đột, nên có thể có nhiều GD giữ khóa đọc một cách đồng thời trên cùng một mục dữ liệu. Ngược lại, một khóa ghi cho phép một GD độc quyền truy cập trên mục dữ liệu đó. Vì vậy, ngay khi một GD giữ một khóa ghi trên mục dữ liệu, thì không một GD nào khác có thể đọc hoặc cập nhật mục dữ liệu đó. Các khóa được sử dụng theo các cách sau đây:

- Bất kỳ một GD nào cần truy cập một mục dữ liệu, trước hết nó phải khóa mục đó lại, bằng cách yêu cầu một khóa đọc cho truy cập chỉ đọc hoặc khóa ghi cho truy cập có cả đọc và ghi.
- Nếu mục dữ liệu này chưa bị khóa bởi một GD nào khác, thì khóa sẽ được cấp.
- Nếu mục dữ liệu này đang bị khóa, HQTCSĐL sẽ xác định khóa đang yêu cầu này có tương thích với khóa đã có hay không. Nếu một khóa đọc đang được yêu cầu trên một mục mà đang bị khóa bằng một khóa đọc, thì yêu cầu này sẽ được đáp ứng; bằng không thì GD buộc phải đợi cho đến khi khóa đã có được giải phóng.
- Một GD tiếp tục giữ một khóa cho đến khi nó giải phóng khóa một cách tường minh lúc đang thực thi hoặc lúc nó kết thúc (hủy bỏ hoặc hoàn tất). Chỉ khi nào khóa ghi được giải phóng thì hiệu ứng của thao tác ghi mới được nhìn thấy bởi các giao dịch khác.

Ngoài các luật này ra, một số hệ thống cho phép một GD có một khóa đọc trên một mục và sau đó *nâng cấp* (upgrade) khóa lên thành khóa ghi. Điều này cho phép một GD xem xét một dữ liệu trước rồi sau đó mới quyết định liệu nó có muốn cập nhật dữ liệu đó hay không.

Nếu khả năng nâng cấp không được hỗ trợ, một GD phải giữ các khóa ghi trên tất cả các mục dữ liệu mà nó có thể phải cập nhật trong suốt quá trình thực hiện GD, và vì vậy có thể làm giảm mức độ cạnh tranh trong hệ thống. Cũng vì lý do này, một số hệ thống cho phép một GD giữ một khóa ghi và sau đó *giáng cấp* (downgrade) khóa xuống thành khóa đọc.

Tuy nhiên, nếu chỉ sử dụng khóa trong quản lý các GD cạnh tranh như mô tả bên trên, thì không đủ để đảm bảo tính khả tuần tự của các lịch trình như trong ví dụ sau đây.

Ví dụ: Lịch trình khóa chốt không đúng:

Xét lại hai GD trong bảng 15, giả sử T9 commit ở cuối GD của nó. Một lịch trình hợp lệ có thể được thực hiện bằng cách sử dụng các luật khóa chốt như sau:

$S = \{ \text{write_lock}(T9, X), \text{read}(T9, X), \text{write}(T9, X), \text{unlock}(T9, X),$
 $\text{write_lock}(T10, X), \text{read}(T10, X), \text{write}(T10, X), \text{unlock}(T10, X),$
 $\text{write_lock}(T10, Y), \text{read}(T10, Y), \text{write}(T10, Y), \text{unlock}(T10, Y),$
 $\text{commit}(T10),$
 $\text{write_lock}(T9, Y), \text{read}(T9, Y), \text{write}(T9, Y), \text{unlock}(T9, Y), \text{commit}(T9) \}$

Nếu ban đầu, $X = 100$, $Y = 400$, thì kết quả sẽ là:

$X = 220$, $Y = 330$, nếu T9 thực hiện trước T10;

hoặc $X = 210$, $Y = 340$, nếu T10 thực hiện trước T9.

Tuy nhiên, kết quả của lịch trình S hiện tại phải là $X = 220$ và $Y = 340$. (chứng tỏ S không phải là một lịch trình khả tuần tự.)

Nguyên nhân dẫn đến tình trạng sai này là lịch trình giải phóng các khóa mà nó đang giữ ngay khi thao tác đọc/ghi tương ứng thực hiện xong và mục dữ liệu đó (chẳng hạn như X) không cần truy cập nữa. Tuy nhiên, bản thân GD đang khóa các mục khác (Y), sau khi nó giải phóng khóa trên X. Mặc dù điều này có vẻ cho phép mức độ cạnh tranh nhiều hơn, nhưng nó cho phép các GD xen kẽ với nhau, làm mất đi tính độc lập và tính nguyên tử của toàn bộ GD.

Vì vậy, để đảm bảo tính khả tuần tự, chúng ta phải tuân theo một giao thức khác liên quan đến việc đặt các thao tác khóa và mở khóa trong mọi GD. Giao thức nổi tiếng nhất là khóa chốt hai kỳ (two-phase locking – 2PL).

2.5.2 Giao thức khóa hai kỳ (2PL)

Định nghĩa: Một GD tuân theo giao thức khóa hai kỳ nếu như tất cả các thao tác khóa đều tiến hành trước thao tác mở khóa trong GD đó.

Theo luật của giao thức này, mỗi GD có thể chia ra thành 2 kỳ:

Kỳ phát triển (growing phase) là kỳ đầu, ở đó nó yêu cầu tất cả các khóa cần thiết nhưng không giải phóng khóa nào,

Kỳ co lại (shrinking phase) là kỳ sau, ở đó nó giải phóng các khóa của nó và không thể yêu cầu thêm bất kỳ khóa mới nào.

Không bắt buộc tất cả các khóa phải có được một cách đồng thời. Thông thường, GD yêu cầu một số khóa, thực hiện một số xử lý và tiếp tục yêu cầu thêm các khóa cần thiết khác. Tuy nhiên, nó sẽ không bao giờ giải phóng khóa nào cho đến khi nó đạt đến giai đoạn không cần thêm khóa nào mới nữa. Các quy luật được phát biểu như sau:

Một GD phải đạt được khóa trên một mục trước khi nó xử lý mục đó. Khóa này có thể là đọc hoặc ghi, tùy thuộc vào nhu cầu truy cập.

Một khi GD giải phóng một khóa, nó không thể yêu cầu thêm khóa mới nào.

Nếu việc nâng cấp khóa được cho phép, thì việc nâng cấp này chỉ có thể diễn ra trong kỳ phát triển và GD có thể phải đợi cho đến khi GD khác giải phóng khóa đọc trên mục đó. Việc giáng cấp chỉ có thể thực hiện trong kỳ co lại. Chúng ta sẽ xem xét cách thức khóa hai kỳ giải quyết ba vấn đề đã đề cập trong mục 2.1

2.5.2.1 Tránh vấn đề mất dữ liệu đã cập nhật bằng 2PL

Một giải pháp cho vấn đề mất dữ liệu đã cập nhật đã đề cập trong mục 2.2.1 được mô tả trong bảng 16 như sau:

Đầu tiên T2 yêu cầu khóa ghi trên A. Sau đó nó có thể đọc giá trị của A trong CSDL, tăng lên 100 đồng, và ghi giá trị mới vào CSDL.

Khi T1 bắt đầu, nó cũng yêu cầu khóa ghi trên A. Tuy nhiên, vì A đang được khóa bởi T2, yêu cầu này của T1 không được đáp ứng ngay lập tức và T1 phải đợi cho đến khi T2 giải phóng khóa. Và điều này chỉ xảy ra khi T2 hoàn tất.

Time	T ₁	T ₂	TK A
t ₁		Begin	100
t ₂	Begin Transaction	Write lock(A)	100
t ₃	Write lock(A)	Read(A);	100
t ₄	WAIT	A = A + 100 ;	100
t ₅	WAIT	Write(A);	200
t ₆	WAIT	Commit/	200
t ₇	Read(A);		200
t ₈	A:=A-10;		190
t ₉	Write(A);		190

t_{10}	Commit/		190
----------	---------	--	-----

Bảng 16. Giao thức 2PL giải quyết vấn đề mất dữ liệu đã cập nhật.

2.5.2.2 Tránh vấn đề phụ thuộc vào GD không hoàn tất bằng 2PL

Một giải pháp của vấn đề đã đề cập trong mục 2.2.2 được mô tả trong bảng 17 như sau:

Đầu tiên T4 yêu cầu khóa ghi trên X. Sau đó nó có thể đọc giá trị trên X từ CSDL, tăng nó lên 100 đồng, và ghi giá trị mới vào CSDL.

Khi hành động cuộn lại được thực hiện, các cập nhật của T4 sẽ được gỡ bỏ và giá trị của X trong CSDL được trả về như ban đầu tức là 100.

Khi T3 bắt đầu, nó cũng yêu cầu một khóa ghi trên X. Tuy nhiên vì X đang bị khóa bởi T4, yêu cầu này không được đáp ứng ngay lập tức và T3 phải đợi cho đến khóa T4 giải phóng khóa. Và điều này chỉ diễn ra khi việc cuộn lại của T4 diễn ra xong.

Time	T ₃	T ₄	TK A
t_1		Begin	100
t_2		Write lock(A)	100
t_3	Begin Transaction	Read(A);	100
t_4	Write lock(A)	$A = A + 100$;	100
t_5	WAIT	Write(A);	200
t_6	WAIT	Rollback/	100
t_7	Read(A);		100
t_8	$A := A - 10$;		90
t_9	Write(A);		90
t_{10}	Commit/ Unlock(A)		90

Bảng 17. Giao thức 2PL giải quyết vấn đề phụ thuộc vào GD không hoàn tất.

2.5.2.3 Tránh vấn đề phân tích không nhất quán bằng 2PL

Một giải pháp cho vấn đề đã đề cập trong mục 2.2.3 được mô tả trong bảng 18 như sau:

Đầu tiên T5 yêu cầu và có được khóa ghi trên X. Sau đó, khi T6 muốn có khóa đọc trên X, yêu cầu này sẽ không được đáp ứng tức thì và T6 phải đợi cho đến khi T5 giải phóng khóa, chính là khi T5 hoàn tất.

Time	T ₅	T ₆	X	Y	Z	Sum
t_1		Begin Transaction	100	50	25	
t_2	Begin Transaction	Sum = 0	100	50	25	0
t_3	Write lock (X)		100	50	25	0
t_4	Read(X);	Read lock (X)	100	50	25	0
t_5	$X = X - 10$;	WAIT	100	50	25	0
t_6	Write (X);	WAIT	90	50	25	0
t_7	Write lock (Z)	WAIT	90	50	25	0
t_8	Read (Z)	WAIT	90	50	25	0
t_9	$Z = Z + 10$	WAIT	90	50	25	0

t ₁₀	Write(Z):	WAIT	90	50	35	0
t ₁₁	Commit/ Unlock(X,Z)	WAIT	90	50	35	0
t ₁₂		WAIT	90	50	35	0
t ₁₃		Read(X):	90	50	35	90
t ₁₄		Sum = Sum + X	90	50	35	90
t ₁₅		Read lock(Y)	90	50	35	90
t ₁₆		Read(Y):	90	50	35	140
t ₁₇		Sum = Sum + Y	90	50	35	140
t ₁₈		Read lock(Z)	90	50	35	140
t ₁₉		Read(Z)	90	50	35	175
t ₂₀		Sum = Sum + Z	90	50	35	175
		Commit/Unlock(X,Y,Z)				

Bảng 18. Giao thức 2PL giải quyết vấn đề phân tích không nhất quán.

Ta có thể chứng minh nếu mọi GD trong một lịch trình đều tuân theo giao thức khóa hai kỳ, thì lịch trình này được đảm bảo là khả tuần tự xung đột.

2.5.2.4 Các vấn đề nảy sinh khi sử dụng giao thức khóa 2 kỳ:

Tuy nhiên thời điểm giải phóng khóa cũng có thể phát sinh vấn đề như ví dụ sau đây.

Tim	T₁₄	T₁₅	T₁₆
t ₁	Begin Transaction		
t ₂	Write lock(X)		
t ₃	Read(X)		
t ₄	Read lock(Y)		
t ₅	Read(Y)		
t ₆	X = X + Y		
t ₇	Write(X);		
t ₈	Unlock(X);	Begin Transaction	
t ₉	:	Write lock(X)	
t ₁₀	:	Read(X);	
t ₁₁	:	X = X + 100 ;	
t ₁₂	:	Write(X);	
t ₁₃	:	Unlock(X)	
t ₁₄	:	:	
t ₁₅	roolback	:	Begin Transaction
t ₁₆		Roolback	Read lock(X)
t ₁₇			:
t ₁₈			Roolback

Bảng 19. Lịch trình cuộn nhiều tầng (cascade rollback).

Khi T₁₄ thất bại và phải cuộn lại, thì T₁₅ cũng phải cuộn lại theo vì T₁₅ đã đọc giá trị ghi bởi T₁₄. Tương tự, T₁₆ cũng phụ thuộc T₁₅, nên nó cũng phải bị cuộn lại. Tình huống mà ở đó một GD dẫn đến một chuỗi cuộn lại, được gọi là cuộn nhiều tầng.

Ta không muốn xảy ra tình huống này, vì nó có thể dẫn đến việc phải hủy đi một lượng công việc khá lớn. Một cách để tránh tình huống này là chỉ giải phóng tất cả khóa ở cuối GD. Khóa hai kỳ biến đổi này gọi là khóa **hai kỳ nghiêm ngặt** (rigorous 2PL). Một biến đổi của khóa hai kỳ khác nữa gọi là khóa **hai kỳ chặt chẽ** (strict 2PL)

ở đó chỉ giữ các khóa ghi đến cuối GD. Hầu hết các HQTCS DL đều cài đặt một trong hai biến đổi này.

Một vấn đề khác nảy sinh của khóa hai kỳ nữa đó là nó có thể dẫn đến khóa chết (deadlock), vì các GD có thể đợi nhận khóa trên các mục dữ liệu. Nếu hai GD đợi nhận các khóa trên các mục mà đã được giữ bởi đối phương, thì khóa chết sẽ diễn ra và vì vậy ta cần có một cơ chế phát hiện khóa chết và phục hồi nó. Ta sẽ thảo luận khóa chết trong phần kế.

Các GD cũng có thể bị dẫn đến tình trạng khóa sống (livelock), nghĩa là, bị bỏ ở tình trạng chờ mãi mãi, không thể đạt được thêm khóa mới nào, mặc dù HQTCS DL không bị khóa chết. Điều này có thể xảy ra khi giải thuật chờ không công bằng và không xem xét đến thời gian mà GD đã đợi. Để tránh khóa sống, một hệ thống ưu tiên được sử dụng, nơi mà một GD càng chờ lâu thì độ ưu tiên của nó càng cao. Hoặc sử dụng một hàng đợi theo kiểu *đến trước được phục vụ trước*.

2.5.2.4.1 Khóa chết (dead lock)

Định nghĩa:

Khóa chết là một tình huống bế tắc khi hai hay nhiều GD đang chờ lẫn nhau để có được các khóa đang giữ bởi đối phương.

Time	T ₁₇	T ₁₈
t ₁	Begin Transaction	
t ₂	Write lock(X)	Begin Transaction
t ₃	Read(X)	Write lock(Y)
t ₄	X = X – 10	Read(Y)
t ₅	Write(X);	Y = Y + 100
t ₆	Write lock(Y)	Write(Y);
t ₇	WAIT	Write lock(X)
t ₈	WAIT	WAIT
t ₉	WAIT	WAIT
t ₁₀	...	WAIT
t ₁₁

Bảng 20. Tình trạng khóa chết

Bảng 20 minh họa hai GD T₁₇ và T₁₈ đang ở tình trạng khóa chết:

Ở thời điểm t₂, GD T₁₇ yêu cầu và đạt được khóa ghi trên X. Ở thời điểm t₃, GD T₁₈ đạt được khóa ghi trên Y. Sau đó, tại thời điểm t₆, T₁₇ yêu cầu khóa ghi trên Y. Vì T₁₈ đang giữ khóa trên Y, T₁₇ phải đợi. Trong khi đó, tại thời điểm t₇, T₁₈ yêu cầu khóa trên X, mà khóa này đang bị T₁₇ giữ.

Cả hai GD không thể tiếp tục bởi vì mỗi GD đang chờ nhận khóa, và nó không thể nhận được cho đến khi GD kia hoàn thành. Một khi xảy ra khóa chết, các ứng dụng liên quan không thể giải quyết vấn đề. Thay vào đó, HQTCS DL phải phát hiện ra sự tồn tại khóa chết và phá đi tình trạng này bằng một cách nào đó.

Thật không may là chỉ có một cách để phá bỏ tình trạng khóa chết là hủy bỏ một hoặc nhiều GD. Điều này liên quan đến việc hủy bỏ tất cả các thay đổi đã được thực hiện bởi các GD này. Việc giải quyết tình trạng khóa chết phải được trong suốt đối với người dùng, và vì vậy HQTCS DL phải tự động bắt đầu lại các GD đã bị hủy.

Các kỹ thuật xử lý khóa chết

Có hai kỹ thuật thông thường được dùng để xử lý khóa chết là: ngăn chặn khóa chết và phát hiện – phục hồi khóa chết.

Sử dụng kỹ thuật ngăn chặn khóa chết, HQTCSDL phải nhìn trước để xác định liệu một GD có khả năng gây ra khóa chết hay không, và không bao giờ để nó xảy ra.

Sử dụng kỹ thuật phát hiện và phục hồi khóa chết, HQTCSDL cho phép khóa chết xảy ra nhưng phát hiện sự xuất hiện của nó và phá nó đi. Vì việc phát hiện và phá khóa chết dễ hơn là ngăn chặn nó, có nhiều hệ thống sử dụng kỹ thuật này hơn.

➤ **Ngăn chặn khóa chết**

Một cách có thể ngăn chặn khóa chết là ra lệnh cho các GD sử dụng các nhãn thời gian, mà chúng ta sẽ thảo luận trong phần kế tiếp. Có hai giải thuật được đề nghị bởi Rosenkrantz et al. (1978):

Một giải thuật là Wait-Die: chỉ cho phép một GD cũ hơn đợi một GD mới hơn, nếu không GD đó sẽ bị hủy (die – chết) và khởi động lại với cùng nhãn thời gian, để cuối cùng nó sẽ trở thành GD đang hoạt động cũ nhất và sẽ không chết nữa.

Giải thuật thứ hai, Wound-Wait, sử dụng một tiếp cận ngược lại: chỉ có một GD mới có thể đợi một giao dịch cũ hơn. Nếu một GD cũ hơn yêu cầu một khóa đang được giữ bởi một GD mới hơn, thì GD mới hơn sẽ bị hủy (wounded – bị tổn thương).

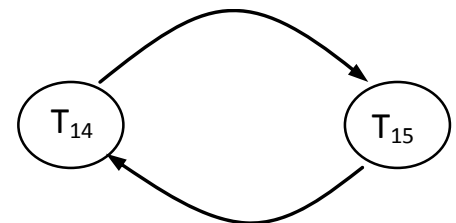
➤ **Phát hiện khóa chết**

Phát hiện khóa chết thường được quản lý bởi việc xây dựng một đồ thị chờ (wait-for graph – WFG), cho thấy sự phụ thuộc của các GD. Một GD T_i được gọi là phụ thuộc vào T_j , nếu GD T_j đang giữ các khóa trên một mục dữ liệu mà T_i đang chờ.

Đồ thị chờ được xây dựng như sau:

Tạo một nút cho mỗi GD

Tạo một cạnh trực tiếp từ T_i đến T_j , nếu GD T_i đang chờ để khóa một mục mà đang bị khóa bởi T_j .



Hình 17. Đồ thị chờ có chu trình

Khóa chết tồn tại khi và chỉ khi đồ thị chờ chứa một chu trình (Holt, 1972). Hình 17 minh họa đồ thị chờ của các giao dịch trong bảng 20. Rõ ràng tồn tại một chu trình trong đồ thị, vì vậy chúng ta có thể kết luận rằng hệ thống bị khóa chết.

Vì đồ thị chờ có chu trình là điều kiện cần và đủ để tồn tại khóa chết, nên giải thuật phát hiện khóa chết sẽ sinh ra đồ thị vào những khoảng thời gian định kỳ và kiểm tra xem có chu trình không.

Việc chọn lựa khoảng thời gian giữa hai lần thực hiện của giải thuật là quan trọng. Nếu khoảng thời gian này là nhỏ quá, thì việc phát hiện khóa chết sẽ mất một khoảng thời gian khá lớn; nếu khoảng thời gian này là quá lớn, thì khóa chết có thể sẽ không bị phát hiện trong một khoảng thời gian dài.

Một giải thuật phát hiện khóa chết động có thể bắt đầu với một khoảng thời gian nào đó. Nếu không phát hiện được khóa chết nào, thì khoảng thời gian sẽ tăng lên, chẳng hạn nhân lên gấp đôi, và mỗi lần phát hiện được khóa chết, thì khoảng thời gian sẽ được giảm xuống, chẳng hạn, bằng nửa khoảng thời gian trước đó, với giả thuyết là không vượt quá một giới hạn trên và dưới cho trước.

Sử dụng khóa kết hợp với giao thức khóa hai kỳ sẽ đảm bảo tính khả tuần tự của các lịch trình. Thứ tự của các GD trong một lịch trình tuần tự tương đương dựa trên thứ tự mà trong đó các GD khóa các mục dữ liệu chúng cần. Nếu một GD cần một mục dữ liệu đã bị khóa, nó có thể bị buộc phải đợi cho đến khi mục đó được giải phóng. Một tiếp cận khác cũng đảm bảo tính khả tuần tự là sử dụng các nhãn thời gian trên giao dịch để sắp xếp sự thực hiện của GD để tạo thành một lịch trình tuần tự tương đương.

2.5.3 Nhãn thời gian (timestamp)

Các phương pháp dùng nhãn thời gian để điều khiển cạnh tranh thì hoàn toàn khác với phương pháp khóa chốt. Vì không hề có khóa, nên cũng không có khóa chốt. Các phương pháp khóa chốt nói chung ngăn chặn xung đột bằng cách buộc các GD phải đợi. Với các phương pháp nhãn thời gian, hoàn toàn không có chờ đợi; các GD có xung đột chỉ đơn giản là bị cuộn lại và được khởi động lại.

Nhãn thời gian của một GD là một định danh duy nhất được tạo ra bởi HQTCSDL cho thấy thời điểm bắt đầu tương đối của GD đó.

Các nhãn thời gian (TG) có thể được tạo ra một cách đơn giản bằng cách sử dụng đồng hồ của hệ thống ở thời điểm GD bắt đầu, hoặc bằng cách tăng một con số đếm luận lý mỗi khi một GD mới bắt đầu.

Định nhãn thời gian (timestamping) là một giao thức điều khiển cạnh tranh mà trong đó mục tiêu cơ bản là sắp xếp các GD một cách toàn cục theo một cách mà các GD cũ hơn, nghĩa là GD với nhãn TG nhỏ hơn, sẽ được ưu tiên hơn khi có xung đột.

Với kỹ thuật định nhãn TG, nếu một GD muốn đọc hoặc ghi một mục dữ liệu, thì:

Các thao tác này chỉ được phép thực hiện nếu cập nhật cuối cùng trên mục dữ liệu đó được thực hiện bởi một GD cũ hơn.

Nếu ngược lại, GD đang muốn đọc/ghi sẽ phải khởi động lại và được cấp cho một nhãn TG mới. Các nhãn TG mới phải được cấp cho các GD khởi động lại để tránh cho chúng không tiếp tục bị hủy và khởi động lại nữa. Nếu không có các nhãn TG mới, một GD với một nhãn TG cũ sẽ có thể sẽ không có khả năng hoàn tất vì các GD trẻ hơn khác đã được hoàn tất.

Ngoài các nhãn TG cho các GD, còn có nhãn TG cho các mục dữ liệu. Mỗi mục dữ liệu chứa một **nhãn đọc** (read-timestamp), ghi nhận nhãn TG của GD cuối cùng đã đọc nó và một **nhãn ghi** (write-timestamp), ghi nhận nhãn TG của GD cuối cùng đã ghi vào mục đó. Với mỗi GD T với nhãn ts(T), giao thức sắp xếp nhãn TG sẽ hoạt động như sau:

Trường hợp giao dịch T phát ra một lệnh read(x):

Nếu GD T yêu cầu đọc 1 mục dữ liệu (x) đã được cập nhật bởi một GD mới hơn; nghĩa là $ts(T) < write_timestamp(x)$.

Điều này nghĩa là một GD trước đó đang cố gắng đọc giá trị của một mục dữ liệu đã được cập nhật bởi một GD sau. Giao dịch trước đó đã quá trễ để đọc một giá trị cũ trước đó, và bất kỳ các giá trị nào khác nó đang có dường như không nhất quán với giá trị đã cập nhật trên mục đó.

Trong trường hợp này, GD T phải bị hủy và khởi động lại với một nhãn TG mới.

Ngược lại, nếu $ts(T) \geq write_timestamp(x)$, thì thao tác đọc có thể tiến hành. Khi đó, chúng ta đặt lại $read_timestamp(x) = \max(ts(T), read_timestamp(x))$.

Trường hợp giao dịch T phát ra lệnh write(x):

Nếu GD T yêu cầu ghi trên một mục dữ liệu (x) mà giá trị của nó đã được đọc bởi một GD mới hơn; nghĩa là, $ts(T) < read_timestamp(x)$.

Điều này có nghĩa là GD sau đã sử dụng giá trị hiện tại của mục dữ liệu và nếu bây giờ cập nhật nó sẽ dẫn đến lỗi. Điều này xảy ra khi một GD bị trễ trong việc ghi và một GD mới hơn đã đọc giá trị cũ hoặc đã ghi vào đó một giá trị mới hơn.

Trong trường hợp này, giải pháp duy nhất là cuộn GD T và khởi động lại nó sử dụng một nhãn TG mới hơn.

Hoặc nếu GD T yêu cầu ghi trên một mục dữ liệu (x) mà giá trị của nó đã được ghi bởi một GD mới hơn; nghĩa là, $ts(T) < write_timestamp(x)$.

Điều này nghĩa là GD T đang muốn ghi một giá trị lỗi thời của x. GD T phải được cuộn lại và khởi động lại với một nhãn TG mới hơn.

Ngược lại, thao tác ghi có thể được tiến hành. Khi đó, chúng ta sẽ đặt lại $write_timestamp(x) = ts(T)$.

Sự sắp xếp này, được gọi là kỹ thuật sắp xếp nhãn TG cơ bản (basic timestamp ordering), đảm bảo rằng các GD là khả tuần tự xung đột, và các kết quả là tương đương với một lịch trình tuần tự trong đó các GD được thực hiện theo thứ tự thời gian ghi trên các nhãn TG. Hay nói khác hơn, các kết quả sẽ giống như toàn bộ GD 1 được thực hiện xong, sau đó toàn bộ GD 2 được thực hiện và cứ như thế, không hề xen kẽ. Tuy nhiên kỹ thuật sắp xếp nhãn thời gian cơ bản này không đảm bảo các lịch trình là khả phục hồi.

2.6 Các kỹ thuật lạc quan (optimistic techniques)

Trong một số môi trường, các xung đột giữa các GD hiếm khi xảy ra, và các xử lý thêm vào bởi các giao thức khóa và nhãn thời gian là không cần thiết cho nhiều GD.

Các **kỹ thuật lạc quan** dựa trên giả thiết rằng xung đột thì hiếm, và sẽ hiệu quả hơn nếu cho phép các GD được tiến hành mà không cần bắt chúng phải trì hoãn để đảm bảo tính khả tuần tự (Kung & Robinson, 1981).

Khi một giao dịch muốn hoàn tất, hệ thống mới kiểm tra xem liệu có xung đột đã xảy ra hay không. Nếu có xung đột, thì GD phải bị hủy và khởi động lại. Vì giả thiết là xung đột hiếm xảy ra, nên việc cuộn lại cũng sẽ hiếm.

Các kỹ thuật lạc quan này tạo cơ hội cạnh tranh nhiều hơn các kỹ thuật cũ, vì chúng không đòi hỏi phải khóa dữ liệu.

Giao thức điều khiển cạnh tranh lạc quan: Một GD sẽ trải qua ba kỳ nếu là GD cập nhật, và trải qua 2 kỳ nếu là GD chỉ đọc:

Kỳ đọc: kỳ này kéo dài từ lúc bắt đầu GD cho đến ngay trước hành động commit. GD đọc các giá trị của tất cả các mục dữ liệu nó cần từ CSDL và lưu chúng vào các biến cục bộ. Các cập nhật sẽ được áp trên bản sao chép cục bộ của dữ liệu, không phải trên CSDL.

Kỳ kiểm tra: Kỳ này theo sau kỳ đọc. Các kiểm tra sẽ được thực hiện để đảm bảo tính khả tuần tự không bị vi phạm nếu các cập nhật của GD được đưa vào CSDL.

Đối với các GD chỉ đọc, điều này bao gồm việc kiểm tra xem các giá trị dữ liệu đã đọc vào các biến vẫn còn là các giá trị hiện hành trong các mục dữ liệu tương ứng. Nếu không có sự can thiệp nào đã diễn ra, thì GD được hoàn tất. Nếu không, GD phải bị hủy và khởi động lại.

Đối với một GD có cập nhật, việc kiểm tra bao gồm việc xác định liệu GD đó có đưa CSDL về một tình trạng nhất quán với tính khả tuần tự được duy trì hay không. Nếu không thì GD phải bị hủy.

Kỳ ghi: Kỳ này theo sau kỳ kiểm tra thành công đối với các giao dịch cập nhật. Trong kỳ này, các cập nhật đã thực hiện trên biến cục bộ sẽ được chép vào CSDL.

Kỳ kiểm tra kiểm tra các thao tác đọc và ghi của các GD là các thao tác có thể xen kẽ nhau. Mỗi GD T được gán:

- Một nhãn **Start(T)** ghi nhận thời điểm bắt đầu thực thi GD,
- Một nhãn **Validation(T)** tại thời điểm bắt đầu vào kỳ kiểm tra,
- Và một nhãn **Finish(T)** ghi nhận thời điểm kết thúc (bao gồm cả kỳ ghi nếu có).

Để có thể thành công vượt qua kỳ kiểm tra, một GD T phải thỏa một trong các điều kiện sau:

1. Tất cả các GD S với nhãn TG sớm hơn T phải hoàn thành trước khi GD T bắt đầu: nghĩa là $Finish(S) < Start(T)$
2. Nếu GD T bắt đầu trước khi một GD trước đó S kết thúc, thì:
 - a. Tập hợp các mục dữ liệu được ghi bởi GD trước đó không phải là các mục được đọc bởi giao dịch hiện tại; và,
 - b. GD trước đó phải hoàn tất kỳ ghi của nó trước khi GT hiện tại đi vào kỳ kiểm tra; nghĩa là, $Start(T) < Finish(S) < Validation(T)$

Luật 2a đảm bảo các thao tác ghi của GD trước đó không được đọc bởi GD hiện tại; luật 2b đảm bảo các thao tác ghi được thực hiện một cách tuần tự, không thể gây xung đột.

Mặc dù các kỹ thuật lạc quan là rất hữu hiệu khi có ít xung đột, chúng có thể dẫn đến sự cuộn lại của các GD một cách đơn lẻ. Chú ý rằng hành động cuộn lại chỉ liên quan đến các bản sao chép dữ liệu cục bộ, vì vậy không có tình huống phải cuộn nhiều tầng, vì các thao tác ghi thực sự chưa đụng đến CSDL. Tuy nhiên, nếu GD bị hủy đó là một GD dài, thì ta sẽ mất đi thời gian xử lý quý báu, vì GD phải được khởi động lại.

Nếu trong một môi trường mà việc cuộn lại này xảy ra thường xuyên, thì đây là một dấu hiệu cho thấy không nên lựa chọn phương pháp lạc quan để điều khiển cạnh tranh trong môi trường đó.

2.7 Độ mịn của mục dữ liệu (data granularity)

2.7.1 Định nghĩa:

Độ mịn là kích cỡ của mục dữ liệu được chọn như là một đơn vị bảo vệ bởi giao thức điều khiển cạnh tranh.

Tất cả các giao thức điều khiển cạnh tranh mà chúng ta đã thảo luận đều giả sử CSDL bao gồm một số lượng các 'mục dữ liệu', mà ta chưa định nghĩa khái niệm này.

Một mục dữ liệu điển hình được chọn như là một trong những thứ có kích cỡ từ lớn đến nhỏ sau đây:

- Toàn bộ CSDL
- Một tập tin (file)
- Một trang tin (page), đôi khi còn gọi là một vùng hay một không gian CSDL – một đoạn đĩa vật lý nơi lưu trữ các quan hệ
- Một mẫu tin (record)
- Một giá trị của một trường trong một mẫu tin.

Kích cỡ hay độ mịn của các mục dữ liệu bị khóa trong một thao tác có ảnh hưởng quan trọng đến hiệu quả chung của một giải thuật điều khiển cạnh tranh.

Tuy nhiên, có một vài thỏa hiệp ta cần lưu ý khi chọn lựa kích cỡ của mục dữ liệu. Chúng ta sẽ thảo luận các thỏa hiệp này trong ngữ cảnh của khóa chốt, mặc dù các lý lẽ tương tự có thể thực hiện cho các kỹ thuật điều khiển cạnh tranh khác.

Xét một GD cập nhật một bộ trong một quan hệ. Giải thuật điều khiển cạnh tranh có thể cho phép GD chỉ khóa trên bộ đó thôi, nghĩa là độ mịn của dữ liệu bị khóa là một mẫu tin đơn lẻ. Mặc khác, chúng cũng có thể khóa toàn bộ CSDL, nghĩa là độ mịn là toàn bộ CSDL. Trong trường hợp thứ 2, độ mịn này sẽ ngăn không cho bất cứ GD nào khác thực hiện cho đến khi khóa được mở. Rõ ràng đây là điều ta không muốn. Mặc khác, nếu một GD cập nhật đến 95% số mẫu tin trong một tập tin, thì việc cho phép nó khóa toàn bộ tập tin đó thay vì khóa từng mẫu tin một sẽ hiệu quả hơn nhiều.

Vì vậy, mục dữ liệu càng lớn, thì mức độ cạnh tranh cho phép sẽ càng thấp. Ngược lại, mục dữ liệu càng nhỏ, thì thông tin về khóa chốt cần lưu trữ sẽ càng nhiều. Kích cỡ tốt nhất phải tùy thuộc vào bản chất của các giao dịch: Nếu một giao dịch thường truy cập một số lượng mẫu tin nhỏ, thì độ mịn của mục dữ liệu ở mức mẫu tin sẽ thuận lợi hơn. Ngược lại một giao dịch thường truy nhiều mẫu tin của cùng một tập tin, thì sẽ tốt hơn nếu ta chọn độ mịn của mục dữ liệu là vùng hay tập tin để giao dịch có thể xem xét tất cả các mẫu tin đó như một (hay một vài) mục dữ liệu.

Một số kỹ thuật đã được đưa ra có hỗ trợ kích cỡ mục dữ liệu động (dynamic). Với các kỹ thuật này, tùy thuộc vào loại giao dịch đang thực hiện, mục dữ liệu sẽ có kích cỡ thay đổi sao cho phù hợp nhất.

Một cách lý tưởng, HQTCSDL nên hỗ trợ khóa với độ mịn hỗn hợp gồm các mẫu tin, trang tin và tập tin. Một số hệ thống tự động nâng cấp độ mịn của các khóa từ mẫu tin hoặc trang tin lên tập tin nếu một GD nào đó đang khóa nhiều hơn một tỉ lệ phần trăm các mẫu tin hoặc trang tin nào đó trong tập tin.

I.1.2. Sự phân cấp của độ mịn

Chúng ta sẽ biểu diễn các độ mịn của khóa trên một cấu trúc phân cấp nơi mà mỗi nút sẽ biểu diễn cho các mục dữ liệu với các kích cỡ khác nhau, như trong hình 18.

Mỗi khi một nút bị khóa thì tất cả các nút con cháu của nó cũng sẽ bị khóa. Ví dụ, nếu một GD khóa trang page2, thì tất cả các record và các field của chúng đều sẽ bị khóa. Nếu một GD khác yêu cầu một khóa *không tương thích* (incompatible) trên cùng một nút, thì HQTCSDL biết là không thể cấp khóa đó.

Nếu một GD khác yêu cầu một khóa trên bất kỳ nút con nào của một nút đã bị khóa, HQTCSDL sẽ kiểm tra con đường phân cấp từ nút gốc đến nút được yêu cầu để

xác định liệu bất kỳ nút tổ tiên nào của nó bị khóa hay không trước khi quyết định cấp khóa.

Vì vậy, nếu ta đang yêu cầu một khóa ghi trên record Record1, HQTCSDL sẽ kiểm tra cha của nó (Page2), ông của nó (File2), và bản thân của nút CSDL (Database) để xác định xem có nút nào bị khóa hay không. Khi nó phát hiện Page2 đã bị khóa, thì nó sẽ từ chối yêu cầu.

Hay trong trường hợp một GD yêu cầu khóa trên một nút và một nút con cháu của nút này đang bị khóa.

Ví dụ như yêu cầu khóa trên File2, HQTCSDL sẽ kiểm tra mỗi trang trong file này, mỗi record trong các trang đó, và mỗi field trong mỗi record để xem có nút nào bị khóa hay không.

Để làm giảm công việc tìm kiếm này, HQTCSDL sử dụng một kiểu khóa mới gọi là **khóa đa hạt** (multiple-granularity locking). Khi một nút nào đó bị khóa, thì một khóa intention sẽ được đặt lên tất cả các nút tổ tiên của nút đó. Vì vậy, nếu một nút con cháu của File2 bị khóa và có một yêu cầu khóa trên nút File2 thì sự hiện diện của khóa intention trên File2 sẽ cho biết là có một nút hậu duệ của nó đã bị khóa.

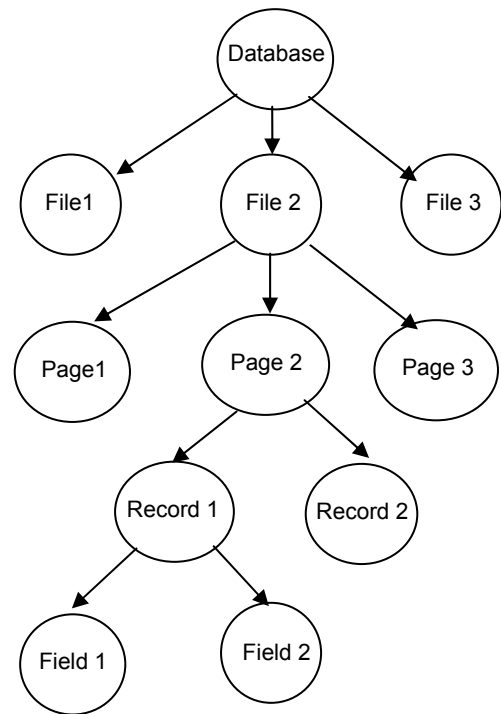
Khóa intention có thể là *Shared* (chia sẻ để đọc) hoặc *exclusive* (độc quyền để ghi). Một khóa *Intention Shared* (IS) chỉ mâu thuẫn với một khóa độc quyền; Một khóa *Intention exclusive* (IX) mâu thuẫn với cả khóa shared và exclusive, v.v... Khóa S và IX là không tương thích; tuy nhiên, nếu 2 khóa này là của cùng 1 giao dịch yêu cầu thì hệ thống sẽ cấp 1 khóa cho nó gọi là khóa SIX

Để đảm bảo tính khả tuần tự với các mức khóa khác nhau, một giao thức khóa hai kỳ được sử dụng như sau:

- Không có khóa nào được cấp một khi có bất kỳ nút nào đó được giải phóng
- Không có nút nào bị khóa cho đến khi các nút cha của nó bị khóa bởi một khóa Intention.
- Không có nút nào được giải phóng cho đến khi tất cả các nút con cháu của nó được giải phóng.

Bằng cách này, các khóa được áp dụng từ trên gốc xuống, sử dụng các khóa intention cho đến giai đoạn nút đạt được khóa đọc hoặc khóa ghi thật sự và các khóa sẽ được giải phóng từ dưới lên.

Tuy nhiên, khóa chết vẫn có thể xảy ra và phải được xử lý như đã thảo luận ở trên.



Hình 18. Sự phân cấp của độ mịn

	IS	IX	S	SIX	X
IS	√	√	√	√	X
IX	√	√	X	X	X
S	√	X	√	X	X
SIX	√	X	X	X	X
X	X	X	X	X	X

Bảng 21. Sự tương thích của các khóa trong cơ chế khóa đa hạt

3 Phục hồi CSDL

Phục hồi CSDL (DB recovery) là quá trình khôi phục CSDL về trạng thái đúng khi có lỗi xảy ra.

3.1 Sự cần thiết phải phục hồi dữ liệu

Việc lưu trữ dữ liệu thông thường bao gồm bốn loại thiết bị với mức độ tin cậy tăng dần là: bộ nhớ trong, đĩa từ, băng từ và đĩa quang.

Bộ nhớ trong là bộ lưu trữ *không ổn định* (volatile), dữ liệu sẽ bị mất khi hệ thống có sự cố, các thiết bị còn lại là các thiết bị *ổn định* (non-volatile).

Hệ thống có *bộ lưu trữ ổn định* (stable storage) là hệ thống lưu trữ thông tin lặp lại trong nhiều thiết bị ổn định (thường là đĩa) với các cơ chế chịu lỗi độc lập nhau. Ví dụ như, ta có thể mô phỏng bộ lưu trữ ổn định bằng công nghệ RAID đã đề cập ở chương 1.

Có rất nhiều loại lỗi có thể ảnh hưởng đến việc xử lý của CSDL, mỗi loại phải được xử lý khác nhau. Một vài lỗi chỉ ảnh hưởng đến bộ nhớ trong, trong khi các loại khác liên quan đến bộ lưu trữ ổn định. Một số nguyên nhân gây lỗi là:

- Hệ thống bị ngưng lại (crash) do lỗi phần cứng hay phần mềm, dẫn đến mất dữ liệu trong bộ nhớ trong.
- Lỗi thiết bị lưu trữ, như đầu đọc bị hư hoặc thiết bị không đọc được, làm mất một phần dữ liệu trong bộ lưu trữ thứ cấp.
- Lỗi phần mềm ứng dụng, như là các lỗi luận lý trong chương trình truy cập CSDL, làm cho một hoặc nhiều GD bị thất bại.
- Thiên tai như lửa, lũ lụt, động đất; hoặc mất điện.
- Sự bất cẩn hay sự phá huỷ dữ liệu hoặc phương tiện không có chủ ý bởi người sử dụng
- Các hành động phá hoại dữ liệu, các thiết bị phần cứng hoặc phần mềm có chủ ý.

Cho dù là nguyên nhân gì, thì có hai ảnh hưởng chủ yếu mà chúng ta cần quan tâm là *mất dữ liệu trong bộ nhớ trong, bao gồm các vùng đệm CSDL; và mất bản sao chép của CSDL trong đĩa*. Trong phần còn lại của chương này chúng ta sẽ thảo luận các khái niệm và kỹ thuật dùng để giảm thiểu các ảnh hưởng này và cho phép phục hồi sau sự cố.

3.2 Các GD và sự phục hồi

Các GD đại diện cho đơn vị phục hồi cơ bản trong một hệ thống CSDL. Khi có lỗi xảy ra, việc đảm bảo hai trong bốn tính chất ACID của giao dịch, tính nguyên tử và tính bền vững, là vai trò của bộ quản lý phục hồi.

Bộ phận này phải đảm bảo rằng khi phục hồi sau lỗi, thì hoặc là tất cả các ảnh hưởng của một GD cho trước là được ghi nhận vĩnh cửu trong CSDL hoặc không có ảnh hưởng nào được ghi. Đây là điều không đơn giản vì các thao tác ghi trong CSDL không phải là một hành động nguyên tử, và vì vậy có thể dẫn đến trường hợp một GD đã hoàn tất, nhưng các ảnh hưởng của nó thì chưa được ghi vĩnh cửu vào CSDL đơn giản bởi vì nó chưa chạm đến CSDL (mà chỉ ghi nhận trong bộ nhớ trong).

Chỉ khi nào các vùng đệm cho CSDL trong bộ nhớ được *đẩy ra* (flush) bộ lưu trữ thứ cấp thì các thao tác cập nhật mới được xem như là vĩnh cửu. Hành động flush này

có thể được bypass bằng các lệnh cụ thể (như commit GD), hoặc xảy ra một cách tự động khi vùng đệm đầy. Hành động ghi vùng đệm một cách tường minh thì được gọi là *ghi-ép-buộc* (force-writing).

Nếu có sự cố xảy ra giữa các lần ghi vùng đệm ra bộ lưu trữ thứ cấp, thì bộ quản lý phục hồi phải xác định tình trạng của GD đã thực hiện hành động ghi tại thời điểm xảy ra lỗi:

Nếu GD đã hoàn tất, tức là đã ra lệnh commit, thì để đảm bảo tính bền vững, bộ quản lý phục hồi phải redo các cập nhật của GD đó lên CSDL (điều này gọi là cuộn tiến rollforward).

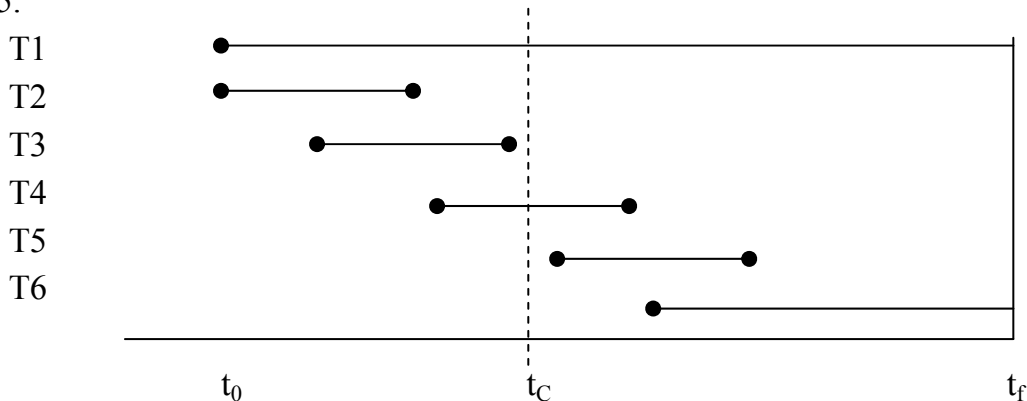
Ngược lại, nếu GD chưa ra lệnh commit tại thời điểm xảy ra lỗi, thì bộ quản lý phục hồi sẽ phải undo (hay cuộn ngược lại rollback) mọi ảnh hưởng của nó trên CSDL để đảm bảo tính nguyên tử của GD.

Hình 19 minh họa một số GD đang thực hiện cạnh tranh T1 ..T6:

HQTCSDL bắt đầu tại thời điểm t_0 , nhưng bị sự cố tại thời điểm t_f . Chúng ta giả thiết rằng dữ liệu cho GD T2 và T3 đã được ghi ra bộ lưu trữ thứ cấp trước khi có lỗi.

Rõ ràng T1 và T6 chưa hoàn tất ở thời điểm xảy ra sự cố; vì vậy, khi khởi động lại, bộ quản lý phục hồi phải undo các GD T1 và T6.

Tuy nhiên, các ảnh hưởng của các GD đã hoàn tất đã được chép vào CSDL trong bộ lưu trữ thứ cấp đến đâu thì không rõ, vì các vùng đệm CSDL trong bộ nhớ có thể đã chép ra đĩa, cũng có thể chưa. Vì vậy, HQTCSDL phải redo tất cả các GD T2, T3, T4 và T5.



Hình 19. Sự cố xảy ra khi các GD đang thực hiện cạnh tranh

3.3 Các tiện ích để phục hồi

Một HQTCSDL có thể cung cấp các tiện ích sau để giúp cho việc phục hồi:

- Một cơ chế sao lưu, để tạo các bản sao chép định kỳ lưu lại của CSDL.
- Các phương tiện ghi nhật ký, để theo dõi tình trạng hiện tại của các GD và các thay đổi của CSDL.
- Một phương tiện kiểm tra, để cho phép các cập nhật đang thực hiện trên CSDL có thể được chép ra vĩnh cửu.
- Một bộ quản lý phục hồi, cho phép hệ thống phục hồi CSDL về tình trạng nhất quán sau sự cố.

3.3.1 Cơ chế sao lưu

HQTCSDL phải cung cấp cơ chế để chép các bản sao của CSDL và tập tin nhật ký tại trong các khoảng thời gian định trước mà không cần phải tạm ngưng hệ thống.

Bản sao lưu của CSDL có thể là toàn bộ CSDL hoặc chỉ sao lưu các thay đổi mới (incremental backup). Bản này chỉ chứa các thay đổi đã thực hiện kể từ lần sao lưu trước. Thông thường, các bản sao lưu này được lưu trong bộ lưu trữ offline như băng từ.

3.3.2 Tập tin nhật ký

Để theo dõi các GD trong CSDL, HQTCSL duy trì một tập tin đặc biệt gọi là log (hay journal) chứa các thông tin về tất cả các cập nhật trên CSDL. Tập tin nhật ký có thể gồm các dữ liệu sau:

- Mẫu tin GD, chứa:
 - Định danh của GD.
 - Kiểu mẫu tin nhật ký (GD bắt đầu, xen, cập nhật, xoá, huỷ, hoàn tất).
 - Định danh của mục dữ liệu bị ảnh hưởng bởi thao tác cập nhật trên CSDL (xen, xoá, sửa).
 - Hình-ảnh-trước (before-image) của mục dữ liệu: đó là giá trị trước khi thay đổi (nếu là xoá hay cập nhật)
 - Hình-ảnh-sau (after-image) của mục dữ liệu: đó là giá trị sau khi thay đổi (nếu là xen hoặc sửa).
 - Các thông tin quản lý nhật ký, như con trỏ đến mẫu tin nhật ký kế trước và kế tiếp cho GD đó.
- Mẫu tin kiểm tra.

Ngoài mục đích phục vụ cho phục hồi CSDL, nhật ký còn được dùng cho việc quản lý hiệu quả và kiểm tra. Vì vậy, nó còn cần phải lưu thêm thông tin về việc đọc CSDL, các đăng nhập, đăng xuất của người dùng, v.v...

Tid	Time	Operation	Object	Before Image	After Image	PPtr	NPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old val.)	(new val.)	1	7
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new val.)	3	5
T2	10:17	DELETE	STAFF SA9	(old val.)		4	9
T3	10:18	START				0	10
T1	10:18	COMMIT				2	0
	10:19	CHECK POINT	T2, T3				
T2	10:19	COMMIT				5	0
T3	10:20	INSERT	PROPERTY PG4		(new val.)	6	11
T3	10:21	COMMIT				10	0

Bảng 22. Một đoạn của tập tin nhật ký, trong đó có 3 GD đang diễn ra song song.

3.3.3 Xác lập thời điểm kiểm tra (checkpointing)

Các thông tin trong tập tin nhật ký được dùng để phục hồi CSDL bị lỗi. Một khó khăn nảy sinh cho cơ chế này là khi có lỗi xảy ra, chúng ta không biết phải lục lại tập

tin nhật ký bao xa để tránh phải redo lại các GD đã được ghi ra CSDL một cách an toàn. Để hạn chế việc tìm kiếm và các xử lý cần thực hiện trên tập tin nhật ký, chúng ta sử dụng một kỹ thuật xác lập thời điểm kiểm tra (checkpointing).

Điểm kiểm tra (checkpoint) là một điểm mà tại đó sự đồng bộ giữa CSDL và tập tin nhật ký GD được ghi nhận. Khi đó, tất cả các vùng đệm phải được ghi-ép-buộc ra bộ lưu trữ thứ cấp.

Các điểm kiểm tra được lập lịch tại các khoảng thời gian định trước và tại điểm kiểm tra, các thao tác sau sẽ được tiến hành:

- Ghi tất cả các mẫu tin nhật ký có trong bộ nhớ trong ra bộ lưu trữ thứ cấp
- Ghi các khối đã được sửa đổi trong vùng đệm CSDL ra bộ lưu trữ thứ cấp.
- Ghi một mẫu tin kiểm tra vào tập tin nhật ký. Mẫu tin này chứa các định danh của các GD đang hoạt động tại thời điểm kiểm tra.

Nếu các GD thực hiện tuần tự:

Khi có lỗi xảy ra chúng ta chỉ cần kiểm tra tập tin nhật ký để tìm ra GD cuối cùng khởi động trước thời điểm checkpoint cuối cùng. Bất cứ GD nào đã hoàn tất trước thời điểm này, đều đã được ghi ra ngoài tại thời điểm kiểm tra. Vì vậy, chúng ta chỉ cần:

Redo GD đang hoạt động tại thời điểm kiểm tra và các GD theo sau mà có cả hai mẫu tin start và commit của chúng xuất hiện trong nhật ký.

Undo lại GD đang hoạt động tại thời điểm xảy ra lỗi, tức là GD chỉ có mẫu tin start.

Nếu các GD thực hiện song song: Thì chúng ta:

Redo tất cả các GD đã hoàn tất kể từ điểm kiểm tra

Undo tất cả các GD đang hoạt động tại thời điểm xảy ra sự cố.

Xét hình 19, nếu ta giả sử điểm kiểm tra xảy ra tại thời điểm tc, thì:

Đối với T2 và T3: chúng ta biết rằng các thay đổi bởi T2 và T3 đã được ghi vào bộ lưu trữ thứ cấp. Vì vậy, bộ quản lý phục hồi không cần redo cho hai GD này.

Đối với T4 và T5: thì ta phải redo, vì hai GD này hoàn tất sau thời điểm kiểm tra.

Đối với T1 và T6: thì ta phải undo, vì hai GD này đang hoạt động tại thời điểm xảy ra sự cố.

Nói chung, checkpointing là một thao tác khá rẻ, và nó thường được thực hiện khoảng 4 lần trong 1 giờ. Bằng cách này, ta sẽ không phải phục hồi nhiều hơn lượng công việc trong khoảng 15-20 phút.

3.3.4 Các kỹ thuật phục hồi

Tùy theo mức độ CSDL bị hư mà thủ tục phục hồi sẽ khác nhau. Ta xét hai tình huống:

Nếu CSDL bị hư, không đọc được nữa, thì ta cần sử dụng bản sao lưu cuối cùng và chạy lại các thao tác của các GD hoàn tất đã ghi nhận trong tập tin nhật ký.

Nếu CSDL không bị hư vật lý nhưng trở nên không nhất quán, chẳng hạn như khi hệ thống bị treo khi đang thực hiện giao dịch, thì ta cần undo các thay đổi gây ra tình trạng không nhất quán. Cũng có thể phải redo một số giao dịch để đảm bảo các cập nhật của nó được lưu ra bộ lưu trữ thứ cấp. Chúng ta sẽ xem xét hai kỹ thuật để phục hồi cho trường hợp này là *cập nhật trì hoãn* (deferred update) và *cập nhật tức thì*

(immediate update). Ngoài ra, còn có một kỹ thuật khác nữa đó là *tạo trang bóng* (shadow paging).

3.3.4.1 Cập nhật trì hoãn

Ý tưởng chủ yếu là trì hoãn các cập nhật thực sự lên CSDL cho đến khi GD kết thúc thành công và đạt đến điểm hoàn tất. Trong khi GD đang thực hiện, các cập nhật chỉ ghi nhận trong nhật ký và trong không gian thực hiện của GD. Sau khi GD đi đến điểm hoàn tất và nhật ký được ghi-ép-buộc ra đĩa, các cập nhật mới được ghi vào CSDL. Nếu 1 GD thất bại trước khi đến lúc hoàn tất, thì không cần phải undo bất cứ thao tác nào, vì GD chưa ảnh hưởng gì đến CSDL.

Các bước liên quan đến giao thức cập nhật trì hoãn như sau:

1. Khi 1 GD T bắt đầu, mẫu tin `start_transaction(T)` được ghi vào nhật ký GD;
2. Khi có một thao tác cập nhật lên mục `x` của CSDL thì mẫu tin `write_item(T, x, old_value, new_value)` được ghi vào nhật ký;
3. Khi một GD sắp sửa hoàn tất, thì ghi vào nhật ký `commit(T)`, ghi tất cả các mẫu tin trong nhật ký ra đĩa;
4. Hoàn tất GD, sử dụng nhật ký để cập nhật CSDL, việc ghi dữ liệu ra đĩa không cần thực hiện ngay.
5. Nếu GD bị hủy, thì xóa đi các mẫu tin nhật ký và đừng ghi các thay đổi ra đĩa.

CSDL không bao giờ được cập nhật cho đến sau khi GD hoàn tất, do đó không bao giờ có thao tác nào cần phải UNDO. Vì vậy, kỹ thuật này còn được gọi là giải thuật NO-UNDO/REDO. Hành động REDO là cần thiết trong trường hợp hệ thống thất bại sau khi GD hoàn tất nhưng trước khi tất cả các thay đổi được ghi vào CSDL. Trong trường hợp này, các thao tác sẽ được thực hiện lại nhờ các mục trong nhật ký.

Khi có sự cố xảy ra, dựa vào nhật ký, hệ thống có thể xác định các GD nào đang diễn ra tại thời điểm bị lỗi. Bắt đầu từ mẫu tin cuối cùng trong tập tin nhật ký để đi ngược về mẫu tin checkpoint gần nhất :

Nếu GD có cả mẫu tin bắt đầu và hoàn tất thì cần phải redo nó. Thủ tục redo sẽ thực hiện tất cả các thao tác cập nhật CSDL sử dụng hình ảnh sau trong mẫu tin nhật ký, theo đúng thứ tự nó đã được ghi vào nhật ký.

Nếu GD có mẫu tin bắt đầu và mẫu tin hủy thì ta không cần làm gì cả vì chúng chưa được ghi vào CSDL.

3.3.4.2 Cập nhật tức thì

Trong kỹ thuật này, CSDL có thể được cập nhật ngay bởi các thao tác của một GD, trước khi GD tiến đến điểm hoàn tất. Tuy nhiên, những thao tác này được ghi nhận trong nhật ký trên đĩa bằng việc ghi-ép-buộc trước khi chúng được áp dụng vào CSDL để có thể phục hồi được.

Các bước liên quan đến giao thức cập nhật tức thì như sau:

1. Khi một GD T bắt đầu, ghi vào nhật ký `start_transaction(T)`;
2. Khi có một thao tác cập nhật mục `x` trong CSDL, ghi vào nhật ký `write_item(T, x, old_value, new_value)`;
3. Ghi nhật ký ra đĩa;
4. Một khi mẫu tin nhật ký đã ghi xong, ghi cập nhật vào vùng đệm CSDL;
5. Khi thuận tiện thì ghi vùng đệm CSDL ra đĩa;

6. Khi một GD sắp sửa hoàn tất, ghi 1 mẫu tin nhật ký commit(T);
7. Ghi nhật ký ra đĩa.

Nếu hệ thống bị lỗi, việc phục hồi liên quan đến việc sử dụng nhật ký để redo và undo các GD:

- Nếu GD có cả mẫu tin bắt đầu và hoàn tất trong nhật ký, thì chúng ta sẽ redo bằng cách sử dụng hình ảnh sau (new_value) trong nhật ký.
- Nếu GD chỉ có mẫu tin bắt đầu nhưng không có mẫu tin hoàn tất, ta cần phải undo GD này bằng cách sử dụng hình ảnh trước trong mẫu tin nhật ký. Thao tác undo sẽ thực hiện theo thứ tự ngược lại với thứ tự các mẫu tin được ghi vào nhật ký.

3.3.4.3 Tạo trang bóng

Một giải pháp khác với cơ chế phục hồi bằng nhật ký như hai kỹ thuật trên là tạo trang bóng. Cơ chế này duy trì các bản hai-trang trong suốt quá trình thực hiện của GD: một bản trang hiện tại và một bản trang bóng. Khi một GD bắt đầu, các bản này giống nhau. Bản trang bóng không bao giờ thay đổi, vì vậy được dùng để phục hồi CSDL khi có sự cố xảy ra. Trong suốt GD, bản trang hiện tại được dùng để ghi nhận mọi thay đổi trên dữ liệu. Khi GD hoàn tất, bản trang hiện tại trở thành bản trang bóng. Kỹ thuật này có ưu điểm là không mất thời gian duy trì nhật ký, và phục hồi nhanh một cách đáng kể vì không cần thực hiện undo hay redo. Tuy nhiên, nó cũng có nhược điểm là phân đoạn dữ liệu và cần phải thu hồi 'rác' một cách định kỳ.

4 Quản lý giao dịch trong Oracle:

4.1 Tổng quan

Trong môi trường CSDL đa người dùng, câu lệnh trong các GD hoạt động song song có khả năng cập nhật trên cùng dữ liệu, điều này có thể dẫn đến dữ liệu không nhất quán. Vì thế, việc điều khiển tính nhất quán và tính cạnh tranh dữ liệu là rất cần thiết trong môi trường đa người dùng.

Các GD hoạt động song song có thể gây ra các hiện tượng đọc sau:

1. Đọc bẩn (dirty read): một GD đọc dữ liệu đang được viết bởi một GD khác chưa hoàn tất.
2. Đọc lặp lại (repeatable read): một GD đọc lại dữ liệu mà nó đã đọc trước đó và nhận ra rằng dữ liệu này đã được thay đổi hoặc xóa bởi một GD khác đã hoàn tất.
3. Đọc ma (phantom read): một GD thực hiện lại một truy vấn trả về một tập hợp các dữ liệu thỏa mãn một điều kiện tìm kiếm và nhận ra sự có mặt một vài dữ liệu mới (thỏa mãn điều kiện tìm kiếm) vừa được thêm vào bởi các GD khác đã hoàn tất.

Chuẩn SQL92 định nghĩa 4 mức cô lập dữ liệu nhằm tránh các hiện tượng đọc này:

Mức cô lập	Đọc bẩn	Đọc lặp lại	Đọc ma
Read Uncommitted	Y	Y	Y

Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N

Bảng 23. Các mức cô lập và các hiện tượng đọc (Y: hiện tượng đọc có thể xảy ra trong mức cô lập, N: hiện tượng đọc không thể xảy ra trong mức cô lập)

Oracle cung cấp 3 mức cô lập: Read Committed, Serializable và Read-only (không thuộc chuẩn SQL92).

1. Read Committed: đây là mức cô lập mặc định cho các GD. Mỗi câu truy vấn trong GD chỉ nhìn thấy các dữ liệu đã được hoàn tất (committed) trước khi câu truy vấn này bắt đầu thực hiện (không phải GD chứa câu truy vấn này). Vì Oracle cho phép các GD khác thay đổi dữ liệu được đọc bởi câu truy vấn, nên dữ liệu có thể bị thay đổi giữa hai lần thực hiện của một câu truy vấn. Vì thế, một GD thực hiện một câu truy vấn nhiều lần có thể gặp các hiện tượng đọc lặp lại và đọc ma.
2. Serializable: GD ở mức cô lập này chỉ nhìn thấy những dữ liệu đã được hoàn tất trước khi GD bắt đầu và những dữ liệu ảnh hưởng bởi GD đó thông qua các câu lệnh `INSERT`, `UPDATE` và `DELETE`. GD ở mức cô lập này không gặp các hiện tượng đọc lặp lại và đọc ma.
3. Read-only: GD ở mức cô lập này chỉ nhìn thấy những dữ liệu đã được hoàn tất trước khi GD bắt đầu và không cho phép thực hiện các câu lệnh `INSERT`, `UPDATE` và `DELETE`.

Mức cô lập Read Committed thích hợp với những môi trường trong đó các GD rất ít xung đột lẫn nhau. Ngược lại, mức cô lập Serializable thích hợp với các môi trường:

- Với CSDL lớn và các GD "ngắn" chỉ cập nhật trên ít dòng dữ liệu.
- Với tỷ lệ hai GD cùng thay đổi trên cùng một dữ liệu là rất thấp.
- Với các GD "dài" chủ yếu chỉ đọc dữ liệu.

Khi GD ở mức cô lập Serializable cố gắng cập nhật hoặc xóa dữ liệu được hoàn tất bởi GD khác sau khi GD này bắt đầu, Oracle sẽ xuất ra lỗi:

ORA-08177: Cannot serialize access for this transaction

Khi đó, trình ứng dụng có thể thực hiện theo một trong các hành động sau:

- Hoàn tất các công việc đã được thực hiện thành công trước đó.
- Thực hiện thêm một số câu lệnh khác để đảm bảo tính toàn vẹn của dữ liệu (ví dụ, thực hiện cuộn lại)
- Thực hiện lại toàn bộ GD.

Ngoài ra, Oracle còn cung cấp cho người dùng cơ chế khoá (locks). Khoá là những cơ chế giúp người dùng giải quyết vấn đề truy xuất đến cùng hạng mục dữ liệu giữa các GD khác nhau như bảng (table) hay dòng dữ liệu (rows). Trong tất cả các trường hợp, Oracle tự động thu tất cả các khoá cần thiết khi thực thi một câu lệnh mà không cần sự tác động của người dùng. Oracle sẽ áp dụng mức thấp nhất có thể áp dụng được để đảm bảo tính toàn vẹn của dữ liệu. Tuy nhiên, Oracle vẫn cho phép người dùng tự thực hiện khoá dữ liệu.

Oracle cung cấp 2 chế độ khoá:

- Khoá loại trừ (exclusive lock): hạng mục dữ liệu được khoá ở mức này sẽ không thể được chia sẻ với các GD khác, mức này phù hợp khi GD muốn thực hiện những thay đổi trên hạng mục dữ liệu.
- Khoá chia sẻ (shared lock): hạng mục dữ liệu được khoá ở mức này có thể được chia sẻ với các GD khác phù hợp, thông thường mức này ngăn chặn những thao tác viết trên hạng mục dữ liệu.

Khoá được giữ trong suốt quá trình GD tồn tại. Oracle sẽ giải phóng khoá khi GD được hoàn tất hoặc được thực hiện lại (undo) toàn bộ. Để nâng cao mức độ cạnh tranh của các GD, Oracle chỉ cung cấp cơ chế chuyển đổi khoá và không cung cấp cơ chế nâng cấp khoá vì nâng cấp khoá rất dễ dẫn đến tình trạng khoá chết (deadlock). Oracle chia khoá thành 3 loại sau:

1. **Khoá dữ liệu (DML locks):** đảm bảo tính toàn vẹn của dữ liệu được truy cập đồng thời bởi các GD khác nhau. Các câu lệnh DML có thể thu cả 2 loại khoá: khoá mức bảng (table-level lock) và khoá mức dòng dữ liệu (row-level lock)
 - Khoá mức dòng dữ liệu (TX): được dùng chủ yếu để tránh việc thay đổi trên cùng dòng dữ liệu của các GD khác nhau. Trước khi một GD thay đổi một dòng dữ liệu, GD đó cần thu khoá dòng dữ liệu. Oracle không giới hạn số lượng khoá dòng dữ liệu được giữ bởi một GD. Khoá dòng dữ liệu là mức hạt mịn nhất trong sơ đồ khoá đa hạt của Oracle.

Một GD chỉ yêu cầu một khoá dòng dữ liệu loại trừ (exclusive row-level lock) khi thực hiện câu lệnh `INSERT`, `UPDATE`, `DELETE` và `SELECT ... FOR UPDATE`. Khi một GD giữ khoá dòng dữ liệu, GD đó cũng giữ khoá mức bảng của bảng dữ liệu tương ứng.

- Khoá mức bảng (TM): thường được dùng để quản lý cạnh tranh với sự có mặt của các câu lệnh DDL, như ngăn chặn việc xoá một bảng trong khi bản đó đang được thao tác bởi các lệnh DML. Một GD yêu cầu một khoá mức bảng khi thực hiện câu lệnh `INSERT`, `UPDATE`, `DELETE` và `SELECT ... FOR UPDATE`. Khoá mức bảng có thể ở nhiều chế độ khác nhau: chia sẻ dòng (Row Share – RS), loại trừ dòng (Row Exclusive – RX), chia sẻ dòng loại trừ (Share Row Exclusive – SRX) và loại trừ (Exclusive – X).

Câu lệnh SQL	Chế độ khoá thu được		Chế độ khoá bảng có thể cấp phát cho các GD khác				
	Dòng	Bảng	RS	RX	S	SRX	X
<code>SELECT ... FROM table</code>		Không	Y	Y	Y	Y	Y
<code>INSERT INTO table ...</code>	X	RX	Y	Y	N	N	N
<code>UPDATE table ...</code>	X	RX	Y*	Y*	N	N	N
<code>DELETE FROM table ...</code>	X	RX	Y*	Y*	N	N	N
<code>SELECT ... FROM table FOR UPDATE OF ...</code>	X	RS	Y*	Y*	Y*	Y*	N

LOCK TABLE <i>table</i> IN ROW SHARE MODE		RS	Y	Y	Y	Y	N
LOCK TABLE <i>table</i> IN ROW EXCLUSIVE MODE		RX	Y	Y	N	N	N
LOCK TABLE <i>table</i> IN SHARE MODE		S	Y	N	Y	N	N
LOCK TABLE <i>table</i> IN SHARE ROW EXCLUSIVE MODE		SRX	Y	N	N	N	N
LOCK TABLE <i>table</i> IN EXCLUSIVE MODE		X	N	N	N	N	N

Bảng 24. Bảng tương thích của các khoá

Bảng trên mô tả các chế độ khoá mức bảng/dòng thu được khi GD thực hiện các lệnh DML và các chế độ khoá cho phép các GD khác yêu cầu khi thực hiện câu lệnh DML thao tác trên cùng bảng dữ liệu (Y*: cho phép nếu không tồn tại xung đột khoá mức độ dòng dữ liệu)

2. **Khoá từ điển (DDL locks):** bảo vệ định nghĩa của các đối tượng lược đồ trong khi đối tượng đó được thao tác hoặc được tham chiếu đến bởi các lệnh DDL sắp thực thi. Các câu lệnh DDL tự hoàn tất GD một cách không tường minh. Một đối tượng giản đồ chỉ được khoá trong khi thực hiện câu lệnh DDL, người dùng không thể yêu cầu khoá một cách tường minh và từ điển dữ liệu không bao giờ được khoá.
3. **Khoá nội tại (Internal locks):** bảo vệ cấu trúc bộ nhớ và cấu trúc bên trong của CSDL, người dùng không thể truy cập đến loại khoá.

4.1.1 Các cú pháp

Oracle không cung cấp câu lệnh tường minh để khởi động một GD, các GD được khởi động một cách không tường minh trong hai trường hợp sau:

- Câu lệnh thực thi đầu tiên của một phiên làm việc mới (user session) tự khởi động một GD.
- Câu lệnh thực thi đầu tiên tiếp theo một GD vừa được hoàn tất (committed) tự khởi động một GD.

Để kết thúc một GD, người dùng có thể thực hiện một trong các cách sau:

- Thực thi lệnh COMMIT để hoàn tất GD.
- Thực thi lệnh ROLLBACK để cuộn lại GD.
- Thực thi một lệnh DDL sẽ kết thúc giao dịch hiện tại một cách ngầm định.
- Kết thúc phiên làm việc của người dùng cũng kết thúc GD một cách ngầm định.

Để thiết lập mức cô lập của một GD, ta có thể thực hiện một trong các lệnh sau:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;
```


Để giảm tải đường truyền và chi phí thực hiện câu lệnh SET TRANSACTION trước mỗi GD, ta có thể thực hiện câu lệnh ALTER SESSION để đặt tất cả các GD sau đó bằng mức cô lập mong muốn.

<pre>ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE; ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;</pre>
--

Oracle luôn tự động thực hiện khoá các hạng mục dữ liệu để đảm bảo tính toàn vẹn của dữ liệu. Tuy nhiên, người dùng có thể tự thay thế chế độ khoá mặc định nhằm thoả mãn các nhu cầu của trình ứng dụng. Việc thay thế này rất hữu ích trong các trường hợp sau:

- Ứng dụng yêu cầu việc đọc bền vững hay đọc lặp lại ở mức giao dịch. Nói cách khác, các câu lệnh truy vấn trong giao dịch phải phản ánh tính bền vững của dữ liệu, không bị ảnh hưởng của những thay đổi trên dữ liệu của các giao dịch khác. Giao dịch có thể đạt đến việc đọc bền vững bằng cách sử dụng khoá loại trừ hay các mức cô lập Read Only và Serializable.
- Ứng dụng có các giao dịch hoạt động trong môi trường cạnh tranh tài nguyên nhưng không muốn một giao dịch phải đợi một giao dịch khác hoàn tất để có thể tiếp tục các câu lệnh bên trong nó.

Người dùng có thể khoá dữ liệu tường minh bằng các câu lệnh sau:

```
SET TRANSACTION ISOLATION LEVEL ...  
LOCK TABLE ...  
SELECT ... FOR UPDATE
```

5 Sao lưu, phục hồi trong Oracle:

Oracle sử dụng một số khái niệm sau trong sao lưu và phục hồi:

- SCN (System Change Number) là một số đếm tuần tự, xác định chính xác một thời điểm trong CSDL. Số này rất hữu ích cho việc tiến hành phục hồi CSDL. Tất cả header của datafiles sẽ có cùng SCN khi CSDL được tắt ở chế độ bình thường (normal).
- Checkpoint: xảy ra khi tiến trình DBWR (database writer) ghi tất cả thay đổi trên bộ nhớ đệm trong SGA và các datafiles. Đồng thời các header của datafiles được cập nhật SCN tại thời điểm checkpoint, ngay cả khi datafile không có thay đổi gì. Checkpoint cũng xảy ra khi các nhóm redo log chuyển trạng thái “hiện thời” cho nhau (redo log switch) hay sau một khoảng thời gian được chỉ định trong tập tin cấu hình hệ thống.
- Image copy: là bản sao của một datafile, một controlfile hay một archived log.
- Backup set: là một cấu trúc luận lý chứa đựng dữ liệu một hoặc nhiều datafiles, archived redo logs hoặc control files.
- Flash Recovery Area: là nơi lưu trữ tất cả các tập tin liên quan đến phục hồi CSDL như archive redo log, control files, tập tin sao lưu tạo ra bởi RMAN và nhật ký flashback.

5.1 Sao lưu

5.1.1 Giới thiệu

Trong các hệ quản trị CSDL, bản sao lưu là bản sao của dữ liệu. Bản sao này chứa các thành phần quan trọng của CSDL như control files và data files. Bản sao lưu là một yếu tố bảo vệ chống lại các lỗi ứng dụng và sự mất mát dữ liệu không mong đợi. Nếu dữ liệu gốc bị mất, chúng có thể được xây dựng lại từ bản sao lưu.

Sao lưu được chia thành sao lưu vật lý (physical backup) và sao lưu luận lý (logical backup).

- Sao lưu vật lý là công việc chủ yếu của chiến lược sao lưu và phục hồi (recover) dữ liệu, bản sao lưu vật lý là bản sao của các tập tin vật lý trong CSDL. Để thực hiện sao lưu vật lý, ta có thể sử dụng công cụ RMAN (Recovery Manager) hoặc công cụ của hệ điều hành (copy tập tin).
- Sao lưu luận lý chứa dữ liệu luận lý như bảng dữ liệu, thủ tục trữ sẵn... được trích ra từ một công cụ hỗ trợ của Oracle và lưu ở dạng tập tin nhị phân. Sao lưu luận lý thường được dùng để hỗ trợ sao lưu vật lý.

Có 2 cách thực hiện sao lưu và phục hồi trong Oracle:

- Sử dụng RMAN: một công cụ hỗ trợ sao lưu (backup), hoàn lại (restore) và phục hồi (recover) các tập tin CSDL.
- Người dùng tự quản lý: sử dụng các lệnh của hệ điều hành để backup và sử dụng SQL*Plus để thực hiện phục hồi CSDL.

5.1.2 Phân loại sao lưu

5.1.2.1 Sao lưu nhất quán (consistent backup)

Sao lưu nhất quán CSDL hay một phần CSDL là sao lưu mà tất cả các data files và controlfiles được kiểm soát thời điểm (checkpointed) với cùng SCN. Cách duy nhất để sao lưu nhất quán toàn CSDL là tắt CSDL ở chế độ normal, immediate hoặc transactional và thực hiện sao lưu khi CSDL đã đóng. Bởi vì, nếu CSDL được tắt ở chế độ khác, ví dụ thể hiện bị lỗi hay CSDL tắt ở chế độ abort, thì các datafiles sẽ không được nhất quán.

Oracle thực hiện nhất quán hóa các datafiles và controlfiles với cùng SCN tại thời điểm kiểm soát (checkpoint) CSDL. Trong quá trình sao lưu nhất quán, chỉ các tablespaces ở chế độ chỉ đọc (read-only) và offline được phép có SCN cũ hơn so với thời điểm kiểm soát. Vì các tablespaces này vẫn nhất quán với các datafiles khác do dữ liệu của chúng không có thay đổi gì mới.

Điểm quan trọng của dạng sao lưu này là chúng ta có thể mở lại CSDL ngay sau khi hoàn lại bản sao nhất quán toàn CSDL mà không cần thực hiện thao tác phục hồi dữ liệu. Nghĩa là không cần thao tác nào hết để các datafiles hoàn lại được đúng. Và tất nhiên, với kiểu sao lưu này, tất cả GD (transactions) kể từ thời điểm sao lưu sẽ bị mất vĩnh viễn.

Sao lưu nhất quán là lựa chọn sao lưu hợp lệ duy nhất của các CSDL hoạt động ở chế độ NOARCHIVELOG, nếu không, các thao tác phục hồi cần thực hiện để đảm bảo tính nhất quán của CSDL. Ở chế độ này, Oracle không lưu lại nhật ký làm lại (redo logs). Sao lưu nhất quán toàn CSDL cũng là một lựa chọn hợp hệ cho CSDL

hoạt động ở chế độ ARCHIVELOG. Khi thực hiện hoàn lại, cùng với các nhật ký được lưu trữ, chúng ta có thể mở CSDL ngay lập tức.

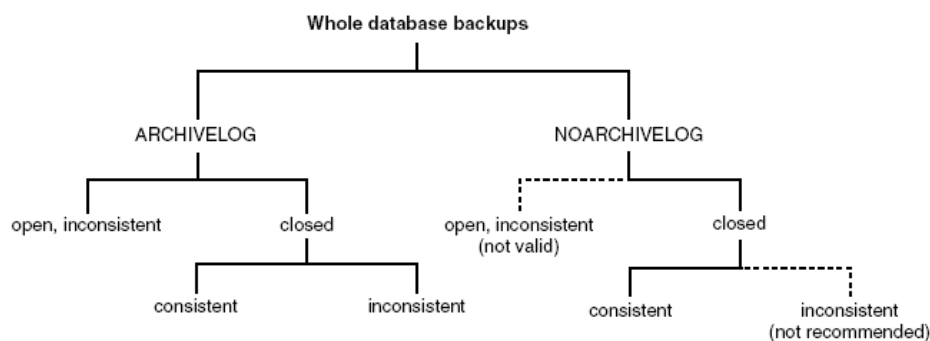
5.1.2.2 Sao lưu không nhất quán (inconsistent backup)

Đây là lựa chọn duy nhất nếu CSDL phải hoạt động 24/24 và 7 ngày trong tuần, với yêu cầu CSDL phải hoạt động ở chế độ ARCHIVELOG. Trong dạng sao lưu này, các tập tin sao lưu được tạo tại các thời điểm khác nhau. Điều này có thể xảy ra, vì các datafiles có thể bị thay đổi trong quá trình sao lưu. Việc phục hồi làm các bản sao không nhất quán thành nhất quán bằng cách đọc các nhật ký lưu trữ (archived redo log) và làm lại (redo log), chọn SCN xa nhất trong tất cả header của datafiles và phục hồi lại dữ liệu của toàn bộ CSDL từ thời điểm tương ứng với SCN đó.

Ở chế độ ARCHIVELOG, CSDL không cần sao lưu toàn bộ CSDL cùng một lúc. Ví dụ, nếu CSDL có 7 tablespaces thì mỗi đêm ta có thể sao lưu controlfiles và 1 tablespace. Như vậy, nếu có sự cố xảy ra, đầu tiên ta cần hoàn lại toàn bộ datafiles và controlfiles gần nhất, sau đó tiến hành phục hồi dữ liệu từ tất cả archived redo log được tạo ra từ thời điểm sao lưu xa nhất.

5.1.2.3 Sao lưu toàn bộ CSDL

Đây là kiểu sao lưu phổ biến nhất. Oracle thực hiện sao lưu tất cả datafiles và controlfile trong CSDL. Kiểu sao lưu này có thể được thực hiện ở chế độ ARCHIVELOG và NOARCHIVELOG. Tuy nhiên, cần phải cân nhắc kiểu sao lưu này để có được dữ liệu nhất quán.



Hình 20. Các lựa chọn sao lưu toàn CSDL

Để sao lưu được nhất quán, cần phải phục hồi dữ liệu từ các tập tin Redo Logs sau khi hoàn lại dữ liệu (restore).

5.1.2.4 Sao lưu từng phần CSDL

Sao lưu datafile

Sao lưu datafile là dạng sao lưu một datafile đơn lẻ, không được sử dụng rộng rãi như sao lưu tablespace. Dạng sao lưu này không hợp lệ với CSDL ở chế độ ARCHIVELOG. Sao lưu datafile chỉ hợp lệ với CSDL ở chế độ NOARCHIVELOG nếu:

- Các datafiles trong tablespace đó đều được sao lưu. Vì tablespace sẽ không thể hoàn lại nếu thiếu 1 datafile.
- Các datafiles phải ở chế độ chỉ đọc hoặc offline.

Sao lưu controlfile

Việc sao lưu controle là rất quan trọng, vì CSDL sẽ không liên kết (mount) hay mở được nếu thiếu tập tin này. Ta có thể thiết đặt RMAN tự động sao lưu controlfile mỗi khi thực hiện việc sao lưu bằng câu lệnh `CONFIGURE CONTROLFILE AUTOBACKUP`. Ngoài ra, ta cũng có thể thực hiện sao lưu thủ công bằng các phương pháp sau:

- Dùng lệnh `RMAN BACKUP CURRENT CONTROLFILE`.
- Dùng lệnh `SQL ALTER DATABASE BACKUP CONTROLFILE`.
- Dùng lệnh `SQL ALTER DATABASE BACKUP CONTROLFILE TRACE` để xuất tập tin script SQL chứa nội dung của controlfile, ta có thể dùng script này để tạo controlfile mới.

Sao lưu archived redo log

Archived redo log là những thông tin thiết yếu cho phục hồi các sao lưu không nhất quán. Cách duy nhất để phục hồi sao lưu không nhất quán mà không sử dụng archived log là sao lưu từng phần (incremental backups) hỗ trợ bởi RMAN. Ta cần phải sao lưu archived redo log thường xuyên bằng các phương pháp sau:

- Dùng lệnh `RMAN BACKUP ARCHIVELOG`.
- Dùng lệnh `RMAN BACKUP ... PLUS ARCHIVELOG`.
- Dùng các công cụ của hệ điều hành.

5.1.2.5 Sao lưu tăng dần (incremental backup)

Sao lưu tăng dần chỉ lưu lại những data blocks có thay đổi so với một lần sao lưu trước đó. Ta có thể thực hiện sao lưu tăng dần trên toàn bộ CSDL, trên từng tablespace hay datafile. Các nguyên nhân chủ yếu khi thực hiện sao lưu tăng dần:

- Giảm thời gian cần thiết khi thực hiện sao lưu hằng ngày.
- Giảm lưu lượng băng thông khi thực hiện sao lưu qua mạng.
- Giảm độ lớn các bản sao lưu cho CSDL ở chế độ `NOARCHIVELOG`.

Giải thuật sao lưu tăng dần

Mỗi data block trong các datafile đều chứa một SCN của lần thay đổi gần nhất của nó. Trong sao lưu tăng dần, SCN của mỗi data block được so sánh với SCN của lần checkpoint thực hiện sao lưu. Nếu SCN của data block nào lớn hơn hoặc bằng thì nó được đưa vào bảng sao lưu.

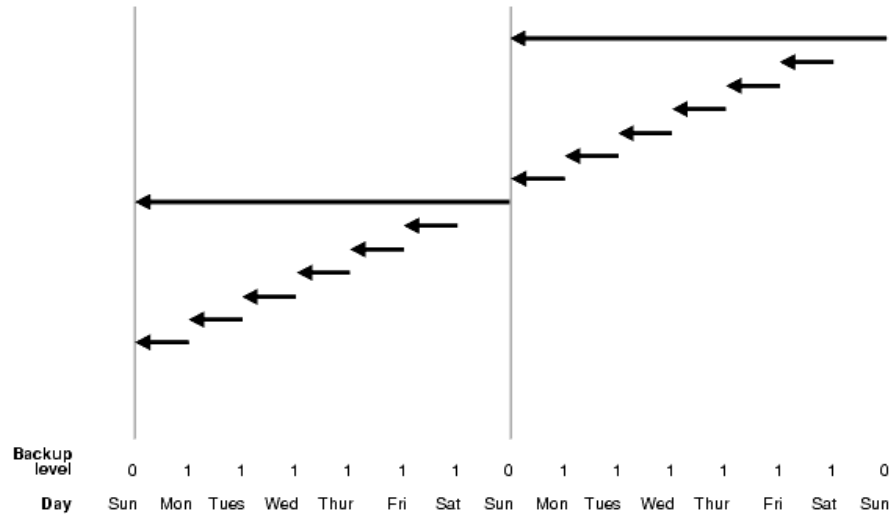
▪ Sao lưu tăng dần mức 0 và mức 1

Mức 0 là nền tảng của các lần sao lưu tiếp theo, nó lưu toàn bộ data blocks của các datafiles, tương đương với sao lưu toàn bộ CSDL.

Mức 1 có thể thuộc vào một trong hai loại: chênh lệch và tích lũy.

▪ Sao lưu tăng dần chênh lệch (differential incremental backup)

Là loại sao lưu mặc định của mức 1, chỉ sao lưu các data blocks có thay đổi từ lần sao lưu tăng dần gần nhất.

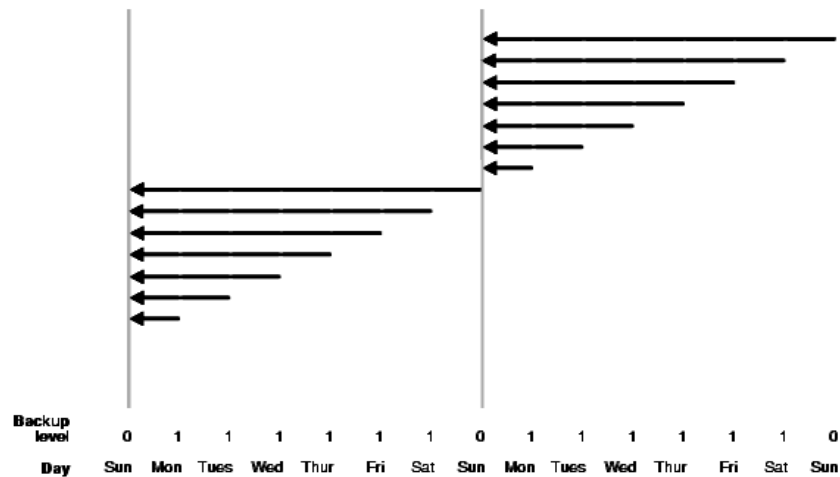


Hình 21. Một chiến lược sao lưu tăng dần chênh lệch

Trong hình trên, một chiến thuật sao lưu được thiết kế theo từng tuần. Ngày chủ nhật có ít GD nên CSDL được sao lưu tăng dần mức 0, sao lưu toàn bộ CSDL. Vào các ngày khác trong tuần, CSDL được sao lưu tăng dần mức 1, nghĩa là thứ hai chỉ lưu lại những thay đổi so với chủ nhật, thứ ba chỉ lưu lại những thay đổi so với thứ hai... và công việc này được lặp lại ở tuần tiếp theo...

▪ Sao lưu tăng dần tích lũy (cumulative incremental backup)

Lưu lại các datablocks có thay đổi kể từ lần sao lưu mức 0 gần nhất. Loại sao lưu này giảm khối lượng công việc phải làm khi hoàn lại (restore) CSDL vì chỉ cần áp dụng một bản sao lưu mức 0 và mức 1. Tuy nhiên, nó làm lại công việc của lần sao lưu cùng mức gần đó, nên nó cần nhiều không gian lưu trữ và thời gian thực hiện hơn.



Hình 22. Một chiến lược sao lưu tăng dần tích lũy

5.1.3 Cú pháp sao lưu

Các lệnh sao lưu phải được thực thi trong chương trình RMAN (Recovery MANager). Để mở chương trình này, vào command console gõ vào lệnh `RMAN target <tên người dùng DBA>`

```
BACKUP FULL tùy_chọn
BACKUP FULL AS (COPY | BACKUPSET) tùy_chọn
```

```
BACKUP INCREMENTAL LEVEL [=] số_nguyên tùy_chọn
BACKUP INCREMENTAL LEVEL [=] số_nguyên AS (COPY | BACKUPSET)
tùy_chọn
BACKUP AS (COPY | BACKUPSET) tùy_chọn
```

```
BACKUP AS (COPY | BACKUPSET) (FULL | INCREMENTAL LEVEL [=]
số_nguyên) tùy_chọn
```

```
Với tùy_chọn ::= { DATAFILE đg_dẫn_datafile [, đg_dẫn_datafile]...
| TABLESPACE ['] tên_tablespace ['] [, ['] tên_tablespace [']]...
| DATABASE
| CURRENT CONTROLFILE [FOR STANDBY]
| SPFILE }
```

Ví dụ:

- Sao lưu CSDL và control file

```
BACKUP DATABASE;
BACKUP CURRENT CONTROLFILE;
```

- Sao lưu các datafiles

```
BACKUP AS BACKUPSET DATAFILE
    'ORACLE_HOME/oradata/trgt/users01.dbf',
    'ORACLE_HOME/oradata/trgt/tools01.dbf';
```

- Sao lưu tất cả datafiles trong CSDL

```
BACKUP AS COPY DATABASE;
```

- Sao lưu các tập tin archive logs trong một khoảng thời gian

```
BACKUP ARCHIVELOG COMPLETION TIME BETWEEN 'SYSDATE-28'
AND 'SYSDATE-7';
```

- Sao lưu tablespace

```
BACKUP TABLESPACE system, users, tools;
```

- Sao lưu controlfile hiện tại

```
BACKUP CURRENT CONTROLFILE TO '/backup/cntrlfile.copy';
```

- Sao lưu tập tin tham số hệ thống

```
BACKUP SPFILE;
```

- Sao lưu toàn bộ CSDL ra backupset

```
BACKUP BACKUPSET ALL;
```

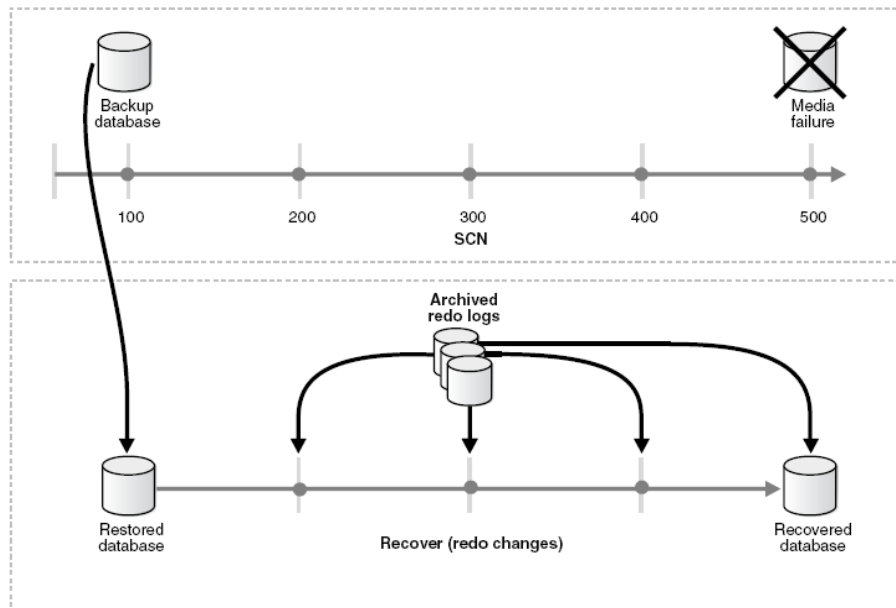
5.2 Phục hồi (recovery)

5.2.1 Giới thiệu

Hoàn lại (restore) bản sao datafile hay control file là tạo lại chúng sao cho Oracle database có thể sử dụng được ngay. Phục hồi (recover) datafile là cập nhật lại dữ liệu bằng cách sử dụng archived redo logs và online redo logs, nghĩa là các thay đổi trên CSDL sau khi việc sao lưu được thực hiện.

Khi có sự cố xảy ra đối với CSDL, nhà quản trị cần hoàn lại CSDL trước tiên, sau đó thực hiện hồi phục các dữ liệu đã thay đổi từ SCN của tập tin sao lưu (đối với sao lưu nhất quán) hoặc SCN cũ nhất của tập tin sao lưu (đối với sao lưu không nhất quán).

Ngoài ra, các lỗi luận lý do người dùng gây ra như xóa nhầm bảng hay xóa nhầm dữ liệu quan trọng. Nhà quản trị có thể sử dụng Oracle Flashback Database và Oracle Flashback Table để khôi phục lại các thông tin này thay vì khôi phục toàn bộ CSDL.



Hình 23. Nguyên lý sao lưu, hoàn lại và phục hồi CSDL

5.2.2 Phân loại phục hồi

5.2.2.1 Phục hồi đầy đủ (complete recovery)

Phục hồi đầy đủ sử dụng dữ liệu redo hoặc các bản sao tăng dần (incremental backup) kết hợp với bản sao CSDL, tablespace hay datafile để cập nhật chúng đến thời điểm gần đây nhất có thay đổi trên dữ liệu. Phục hồi đầy đủ vì ta áp dụng toàn bộ mẫu tin trong archived và online logs trên bản sao. Thông thường, phục hồi đầy đủ được áp dụng khi có sự cố hư hỏng các datafiles và control file.

Các bước thực hiện để phục hồi toàn bộ CSDL:

- Liên kết (mount) CSDL.
- Đảm bảo tất cả datafiles cần phục hồi ở chế độ “trực tuyến”.
- Hoàn lại bản sao toàn bộ CSDL hoặc các tập tin cần phục hồi.
- Áp dụng online và archived redo logs.

Các bước thực hiện để phục hồi tablespace, datafile:

- Đưa tablespace hoặc datafile về chế độ tắt (offline) nếu CSDL còn mở.
- Hoàn lại bản sao của các datafiles cần phục hồi.
- Áp dụng online và archived redo logs.

5.2.2.2 Phục hồi không đầy đủ (incomplete recovery)

Phục hồi không đầy đủ không áp dụng tất cả mẫu tin redo được ghi nhận từ lúc thực hiện sao lưu gần nhất. Phục hồi không đầy đủ toàn bộ CSDL thích hợp với các tình huống sau:

- Sự cố phá hỏng một vài hoặc toàn bộ tập tin online redo logs.
- Người dùng gây ra mất dữ liệu như xóa bảng.
- Thất lạc archived redo log.

- Thất lạc control file và phải sử dụng bản sao control file để mở CSDL.

5.2.3 Cú pháp restore

```
RESTORE đối_tượng_restore;  
đối_tượng_restore ::=  
{ CONTROLFILE [TO 'tên_file']  
  | DATABASE  
  | DATAFILE đg_dẫn_datafile [,đg_dẫn_datafile]...  
  | TABLESPACE ['] tên_tablespace ['] [, ['] tên_tablespace  
[']]...  
  | SPFILE [TO [PFILE] 'tên_file']  
}
```

5.2.4 Cú pháp recover

```
RECOVER đối_tượng_recover;  
đối_tượng_recover::=  
  { DATABASE [cho_đến_khi]  
    | TABLESPACE ['] tên_tablespace ['] [, ['] tên_tablespace  
[']]...  
    | DATAFILE đg_dẫn_datafile [,đg_dẫn_datafile]... }  
cho_đến_khi ::= { UNTIL TIME [=] 'ngày_tháng' | UNTIL SCN [=]  
số_nguyên }
```

Ví dụ:

- Hoàn lại và phục hồi toàn bộ CSDL

```
STARTUP FORCE MOUNT;  
RESTORE DATABASE;  
RECOVER DATABASE;  
ALTER DATABASE OPEN;
```

- Hoàn lại và phục hồi tablespace

```
SQL 'ALTER TABLESPACE users OFFLINE';  
RESTORE TABLESPACE users;  
RECOVER TABLESPACE users;  
SQL 'ALTER TABLESPACE users ONLINE';
```

- Hoàn lại và phục hồi datafile

```
SQL 'ALTER DATABASE DATAFILE 64 OFFLINE';  
RESTORE DATAFILE 64;  
RECOVER DATAFILE 64;  
SQL 'ALTER DATABASE DATAFILE 64 ONLINE';
```


Phần thực hành

BÀI 1. LÀM QUEN VỚI ORACLE

1 Nội dung

- Kết nối với server Oracle.
- Tạo/xóa/mở/tắt CSDL Oracle.
- Quản lý người dùng, quyền và role.
- Import/export và expdp/impdp.

2 Bài tập có hướng dẫn

Câu 1: khởi động các dịch vụ của Oracle

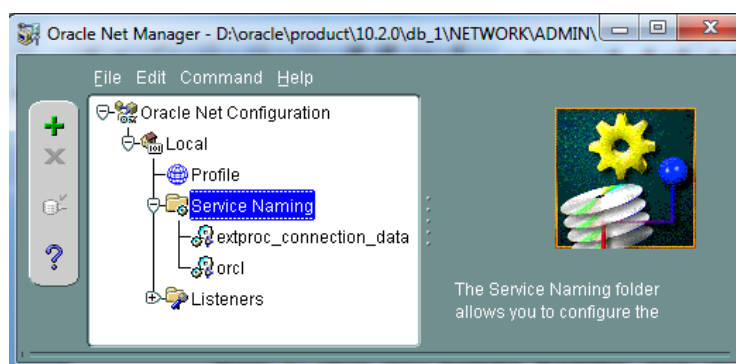
Một số lệnh cơ bản. Các lệnh sau thực hiện trong cửa sổ Command Console của Windows (menu **Start** → **Run**, gõ vào **cmd**):

- Khởi động Listener: `lsnrctl start`
- Khởi động CSDL: `sqlplus sys/hqtcsdl as sysdba`
Tại dấu nhắc `SQL>`, gõ vào `startup;`
Kết thúc trình `sqlplus` bằng lệnh `exit;`
- Khởi động chương trình Oracle Enterprise Manager: `emctl start dbconsole`


Ngoài ra, ta có thể khởi động các dịch vụ này bằng giao diện của trình quản lý các dịch vụ trong Windows (Control Panel → Administrative Tools → Services)

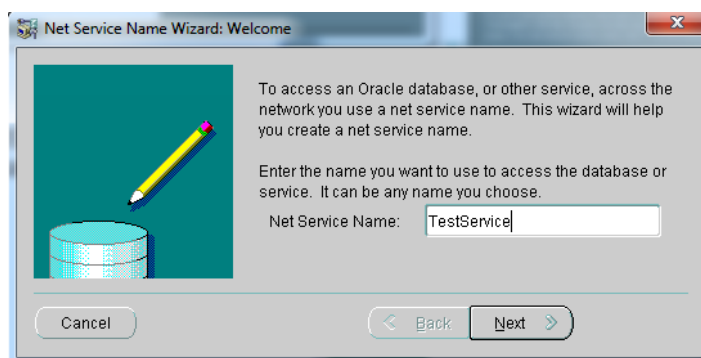
Câu 2: kết nối với CSDL *hqtcsdl* thông qua Oracle Net

Thiết lập kết nối với CSDL *hqtcsdl* thông qua chương trình Oracle Net Manager (Start → Programs → Oracle → Configuration and Migration Tools → Net Manager)



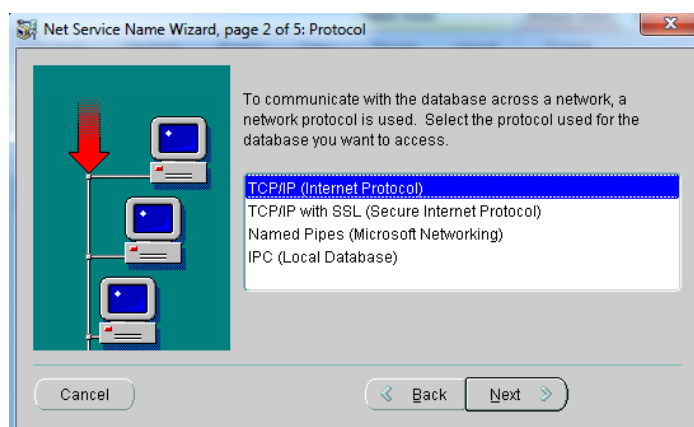
Hình 24. Oracle Net Manager

Bước 1: chọn mục **Service Naming**, kích chuột vào dấu  ở pane bên trái để tạo một net service name với các thông số thực hiện kết nối với server CSDL.



Hình 25. Đặt Net Service Name

Đặt *Net Service Name* là *TestService* (sinh viên có thể đặt tên tùy ý). Chọn **Next**.
Bước 2: chọn Protocol TCP/IP. Chọn **Next**.

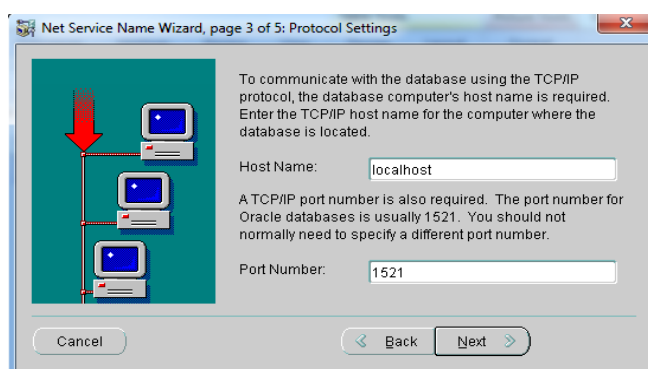


Hình 26. Chọn giao thức mạng được sử dụng

Bước 3: thiết lập các thông số máy chủ như sau:

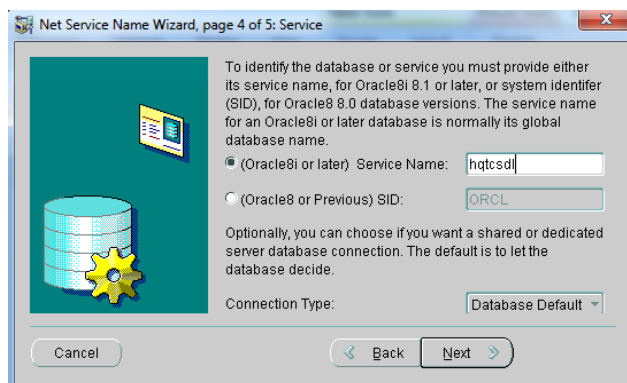
- Host Name: tên server hoặc địa chỉ IP trong mạng, thiết lập *localhost* hoặc *127.0.0.1*
- Port Number: cổng 1521 mặc định.

Chọn **Next**.



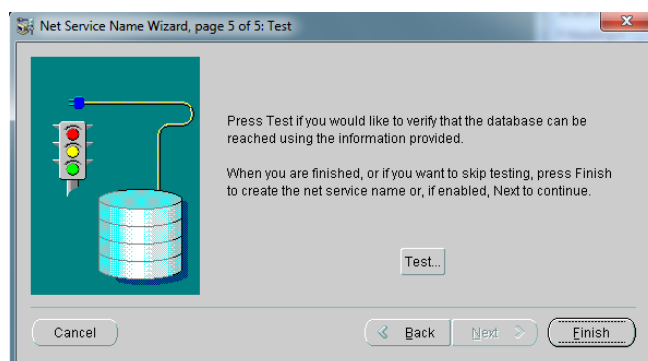
Hình 27. Các thiết lập của giao thức

Bước 4: đặt Service Name: chính là tên CSDL cần truy cập đến (**Global Database Name**). Gõ vào *hqtcSDL*.



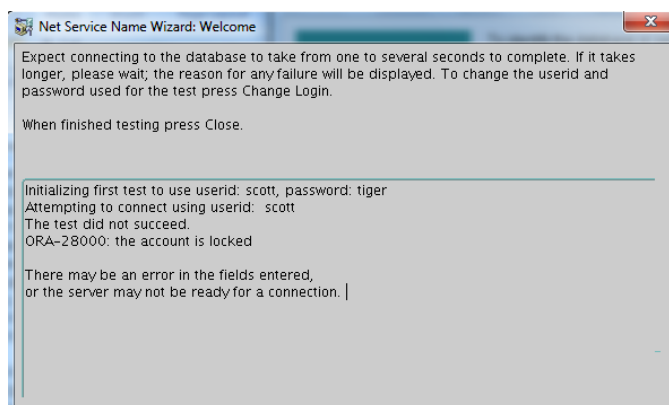
Hình 28. Thiết lập tên service

Bước 5: thực hiện kiểm tra kết nối với CSDL.



Hình 29. Kiểm tra kết nối

Chọn **Test**, nếu kết nối thành công thì xuất hiện bảng thông báo như sau.



Hình 30. Thông báo kết nối thành công

Chọn **Close** và **Finish** để kết thúc thiết lập kết nối.

Bước 6: Kiểm tra kết nối bằng chương trình SQL*Plus. Mở cửa sổ Terminal, gõ vào các lệnh đăng nhập với cú pháp `sqlplus acc/pass@<service name>`, với `acc` và `pass` là tài khoản và mật khẩu của người dùng, **service name** là tên dịch vụ mạng. Ví dụ:

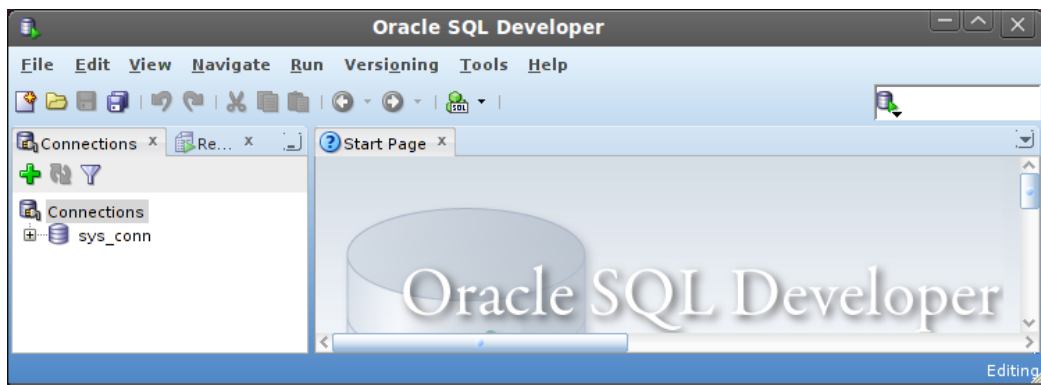
- Đăng nhập bằng người dùng SYS: `sqlplus sys/hqtcSDL@orcl as sysdba;`. Chú ý: `as sysdba` chỉ ra rằng người dùng đăng nhập với vai trò của người dùng quản trị.
- Đăng nhập bằng người dùng SCOTT: `sqlplus scott/tiger@orcl;`

Nếu đăng nhập thành công, thông báo “connected” hiện ra. Ngược lại, cần kiểm tra lại password hoặc các thông số của Net Service Name được sử dụng hoặc trạng thái của tài khoản đăng nhập.


Để kết thúc phiên làm việc, gõ vào `disconnect;` và `exit;`

Câu 3: Dùng chương trình Oracle SQL Developer kết nối với CSDL

Trong khi khởi động `sqldeveloper`, nếu chương trình có hỏi đường dẫn đến thư mục cài đặt J2SE thì chỉ đến thư mục cài đặt Java trong thư mục `C:\Program Files`. Giao diện Oracle SQL Developer như hình sau:



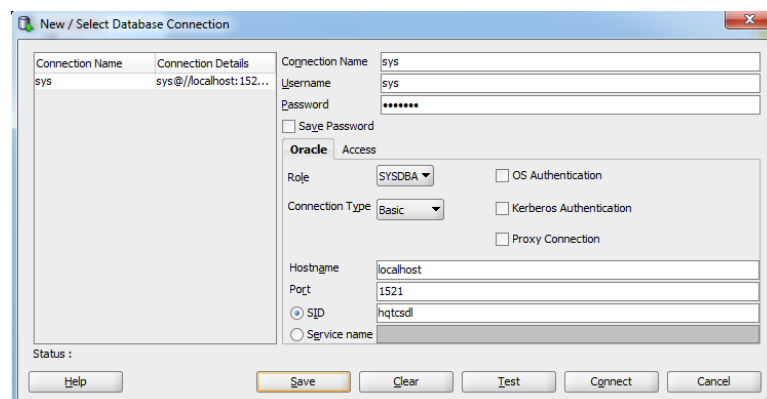
Hình 31. Giao diện SQL Developer

Để tạo kết nối đến CSDL, click vào icon  bên **tab Connections** hoặc vào menu **File** → **New**, chọn Database Connection. Có 2 cách tạo kết nối:

- **Cách 1:** kết nối trực tiếp, tức chọn *Connection type* là basic.

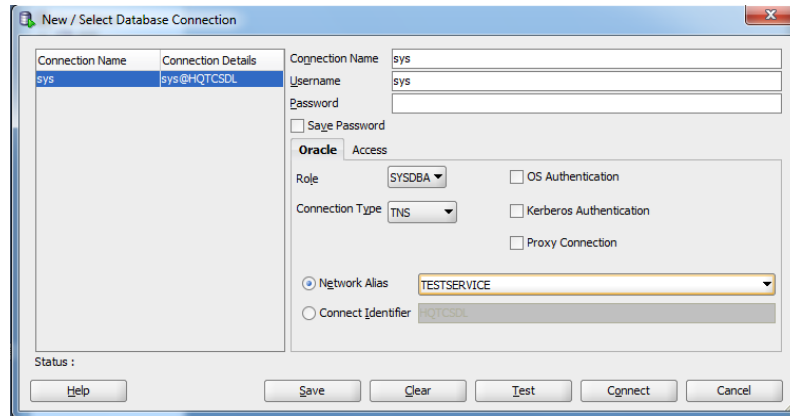
Trong cách kết nối này, các tham số được thiết lập như sau:

- Connection name: tên của kết nối (đặt tùy ý).
- Username, password: của người dùng đăng nhập.
- Role: nếu đăng nhập bằng người dùng SYS, chọn SYSDBA.
- Connection type: basic.
- Hostname: địa chỉ IP của database server. Nhập vào **localhost** hoặc **127.0.0.1**.
- Port mặc định 1521
- SID: tên CSDL cần đăng nhập. Nhập vào **hqtcddl**.



Hình 32. Kết nối trực tiếp

- **Cách 2:** kết nối thông qua Oracle Net, tức chọn *Connection type* là TNS.



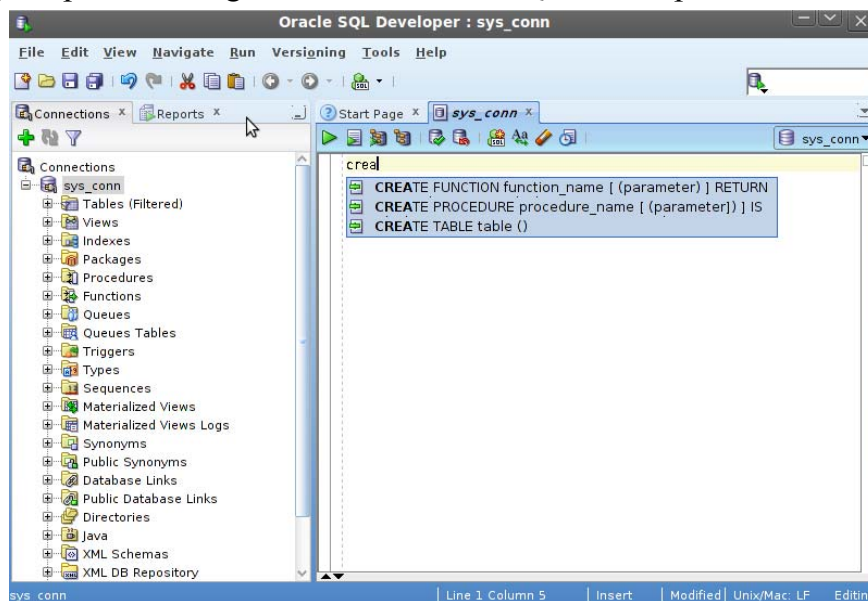
Hình 33. Kết nối thông qua Oracle Net

Trong cách kết nối này, các tham số được thiết lập tương tự cách 1, chỉ khác ở 2 tham số:

- Connection type: TNS.
- Network Alias: chọn *net service name* đã tạo ở các bước trên, tức là TestService.

Trong cả hai cách, các tham số của kết nối có thể được lưu lại để dùng sau này bằng cách nhấn vào nút *Save*. Để kiểm thử, nhấn vào nút *Test*. Để đăng nhập, nhấn vào nút *Connect*.

Khi đăng nhập thành công, cửa sổ chính của SQL Developer xuất hiện như sau:



Hình 34. Cửa sổ chính của Oracle SQL Developer

Giao diện của chương trình được chia thành 2 phần:

- Phần bên trái: hiển thị các schema objects của người dùng đang kết nối.
- Phần bên phải: cửa sổ soạn thảo, hiển thị kết quả, cửa sổ lỗi... của các câu lệnh SQL, PL/SQL.

Ngoài ra, Oracle SQL Developer còn hỗ trợ một số chức năng khác như: giao diện đồ họa thao tác trên các schema objects, trợ giúp về ngôn ngữ SQL và PL/SQL trong module Snippets, quản lý các lệnh SQL đã thực thi với module SQL History, biên dịch và gỡ rối PL/SQL... Và còn nhiều tính năng khác, sinh viên tự tìm hiểu trong quá trình thực hành.

Câu 4: Thực hành tắt và mở dịch vụ CSDL

Giữ kết nối với CSDL bằng chương trình SQL Developer. Mở cửa sổ Terminal, đăng nhập CSDL bằng tài khoản SYS với chương trình SQL*Plus. Thực hiện tắt (shutdown) và mở (startup) CSDL ở các chế độ khác nhau theo các kịch bản sau đây:

Kịch bản 1: tắt CSDL ở chế độ bình thường

SHUTDOWN; -- CSDL có tắt không? Nếu không, hãy giải thích và đưa ra hướng giải quyết.

Kịch bản 2: tắt CSDL ở chế độ tức thì (vẫn giữ kết nối với CSDL bằng SQL Developer)

SHUTDOWN IMMEDIATE; -- Quan sát sự thay đổi của kết nối trong SQL Developer.

Kịch bản 3: tắt CSDL ở chế độ dỡ bỏ (vẫn giữ kết nối với CSDL bằng SQL Developer)

SHUTDOWN ABORT; -- Quan sát sự thay đổi của kết nối trong SQL Developer.

Kịch bản 4: mở CSDL.

STARTUP NOMOUNT; -- Hãy cho biết dung lượng của SGA?

STARTUP; -- Quan sát và giải thích kết quả.

SELECT * FROM DBA_USERS; -- Quan sát và giải thích kết quả.

ALTER DATABASE MOUNT;

SELECT * FROM DBA_USERS; -- Quan sát và giải thích kết quả.

ALTER DATABASE OPEN;

SELECT * FROM DBA_USERS; -- Quan sát và giải thích kết quả.

Câu 5: Tạo người dùng

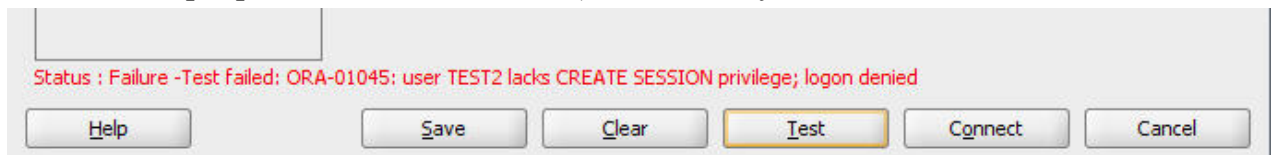
Lưu ý: Để tạo và cấp quyền cho người dùng bạn phải đăng nhập hệ thống với vai trò DBA

Tạo người dùng tương ứng là Test1, mật khẩu giống tên

▪ Tạo user

```
CREATE USER Test1 IDENTIFIED BY Test1;
```

- Thử sử dụng user vừa tạo này để đăng nhập vào hệ thống, bạn sẽ bị thông báo lỗi tương tự như dưới đây, do chúng ta chưa cấp quyền Create Session (quyền cho phép user kết nối vào CSDL) cho user này.



Hình 35. Lỗi đăng nhập do chưa có quyền kết nối đến CSDL

▪ Cấp quyền Create Session cho user Test1

```
GRANT CREATE SESSION TO Test1;
```

- Bây giờ thử đăng nhập lại, OK? Người dùng Test1 được phép kết nối vào CSDL, nhưng việc duy nhất mà user này có thể thực hiện là thay đổi password của chính anh ta, ngoài ra chưa làm được bất cứ việc gì khác.

- Thay đổi mật khẩu cho user này thử xem
`ALTER USER Test1 IDENTIFIED BY pTest1;`
- Đăng nhập lại với mật khẩu mới để kiểm tra. Lúc này, Test1 chỉ có quyền kết nối CSDL.
- Cấp quyền hệ thống cho user này để anh ta được phép tạo bảng và được phép cấp lại quyền này cho những user khác.
`GRANT CREATE TABLE TO Test1 WITH ADMIN OPTION;`
- Đăng nhập với user vừa tạo để thử tạo một bảng, nhập và truy xuất dữ liệu.
`CREATE TABLE Students(SID char(7) PRIMARY KEY,
 SName Varchar2(30));

INSERT INTO Students VALUES('1940000', 'Phung Phung Phi');
INSERT INTO Students VALUES('1940001', 'Tao Tung Thieu');
SELECT * FROM STUDENTS;`

Ồ!!! Tại sao user Test1 lại có quyền INSERT, SELECT trong khi ta chưa cấp quyền nhỉ??? Đơn giản là vì mặc nhiên anh ta sẽ có toàn quyền trên đối tượng mà anh ta vừa tạo (sở hữu - Owner).

Câu 6: Quyền hệ thống

Để kiểm tra kết quả của tùy chọn WITH ADMIN OPTION, Tạo một người dùng khác tên Test2 với Quota 2M để cho phép người dùng này tạo bảng trong schema của anh ta.

- Đăng nhập hệ thống với vai trò của DBA để tạo user Test2
`CREATE USER Test2 IDENTIFIED BY Test2
DEFAULT TABLESPACE USERS QUOTA 2M ON USERS;`
- Gán quyền CREATE SESSION cho Test2
`GRANT CREATE SESSION TO Test2;`
- *Đăng nhập vào hệ thống với tài khoản Test1, cấp quyền hệ thống cho user Test2 để anh ta được phép tạo bảng và được phép cấp lại quyền này cho những user khác.*
`GRANT CREATE TABLE TO Test2 WITH ADMIN OPTION;`
- Đăng nhập với user Test2 để thử tạo một bảng, nhập và truy xuất dữ liệu.
`CREATE TABLE Products(PID char(3) PRIMARY KEY,
 PName Varchar2(30));

INSERT INTO Products VALUES('P01', 'Kem Danh Rang P/S');
INSERT INTO Products VALUES('P02', 'Khan Giay Puppy');
SELECT * FROM PRODUCTS;`
- Thu hồi quyền CREATE TABLE của người dùng Test1. Đăng nhập bằng tài khoản SYS và thực hiện lệnh:
`REVOKE CREATE TABLE FROM Test1;`

- Lần lượt đăng nhập bằng tài khoản Test1 và Test2, tạo một bảng dữ liệu khác. Quan sát kết quả và rút ra kết luận.
- Tạo user Test3 với các quyền như user Test2.

Câu 7: Quyền đối tượng

Cấp quyền đối tượng cho user Test2 truy cập vào bảng student trong schema của Test1.

- Kết thúc câu 2, ta đã tạo user tên Test2 nhưng user này chưa có quyền gì cả, thử đăng nhập lại với user này và truy xuất bảng Students
`SELECT * FROM Test1.Students;`
Bị báo lỗi đúng không? Vì chưa có quyền mà!
- Bây giờ đăng nhập lại với user Test1 và cấp quyền `SELECT, UPDATE` cho user Test2. Chú ý: bạn phải đăng nhập với user là chủ sở hữu của bảng Students, vì chỉ chủ sở hữu có khả năng gán quyền trên các đối tượng của mình.
`GRANT SELECT, UPDATE ON students TO Test2 WITH GRANT OPTION;`
- Đăng nhập lại với user Test2 và truy xuất bảng Students
`SELECT * FROM Test1.Students;`
Được phép rồi!
- Thử thực hiện lệnh sau
`DELETE FROM Test1.Students;`
Chắc chắn là bị lỗi, vì user này chỉ có quyền `SELECT` và `UPDATE` thôi!
- Đăng nhập bằng user Test2, gán quyền `SELECT` và `UPDATE` trên bảng Students cho user Test3.
`GRANT SELECT, UPDATE ON Test1.students TO Test3;`
- Đăng nhập bằng user Test3 và thực hiện câu lệnh truy vấn trên bảng Students
`UPDATE Test1.Students SET sname='bla bla bla' WHERE SID='1940001'`
`SELECT * FROM Test1.Students;`
- Đăng nhập bằng user Test2 thực hiện lệnh sau
`GRANT DELETE ON Test1.students TO Test3;`
Quan sát kết quả và rút ra kết luận.
- Đăng nhập bằng user Test1, thu hồi quyền `SELECT` và `UPDATE` từ Test2
`REVOKE SELECT, UPDATE ON Students FROM Test1;`
Thực hiện các câu truy vấn trên bảng Students với Test2 và Test3, quan sát kết quả và rút ra kết luận.

Câu 8: Tạo role, cấp quyền cho role và thêm người dùng vào role vừa tạo.

Giả sử lớp học có 150 sinh viên thực hành môn Oracle. Các sinh viên này đều có quyền giống nhau là được phép kết nối đến CSDL, tạo bảng, tạo thủ tục, tạo trigger. Chúng ta sẽ thực hiện vấn đề này như sau:

- Đăng nhập bằng người dùng SYS. Tạo các người dùng (thử tạo 2 users)

```
CREATE USER user1 IDENTIFIED BY user1
        DEFAULT TABLESPACE USERS QUOTA 1M ON USERS;
CREATE USER user2 IDENTIFIED BY user2
        DEFAULT TABLESPACE USERS QUOTA 1M ON USERS;
```
- Tạo role có tên là TTOracle

```
CREATE ROLE TTOracle;
```
- Cấp quyền kết nối CSDL, tạo bảng, tạo thủ tục, tạo trigger cho role TTOracle

```
GRANT CREATE SESSION, CREATE TABLE, CREATE PROCEDURE,
        CREATE TRIGGER TO TTOracle;
```
- Cấp quyền cho các sinh viên vào role này (đưa các user vào role)

```
GRANT TTOracle TO user1,user2;
```
- Thử đăng nhập bằng một trong 2 user vừa tạo để tạo bảng xem có được chưa.
- Để thu hồi role, thực hiện câu lệnh sau:

```
REVOKE TTOracle FROM user1;
```

Câu 9: thử nghiệm import và export truyền thống

Bước 1: Tạo lần lượt các người dùng uimp1, uimp2 và uimp3. Gán các quyền cơ bản để các người dùng này có khả năng đăng nhập, tạo bảng dữ liệu và cập nhật dữ liệu. (tham khảo bài 2)

Bước 2: Thực hiện import các đối tượng trong tập tin dump cho trước vào schema của các người dùng vừa tạo ra (đây là các bảng dữ liệu được dùng trong các bài thực hành tiếp theo).

Tạo net service name *xe* trong Net Manager, sau đó mở cửa sổ Command Console. Gõ vào:

```
imp      userid=uimp1@xe      file=D:\csdl1.dmp      fromuser=csdl1
touser=uimp1
```

Nhập vào password tương ứng, quan sát kết quả đầu ra. Nếu thành công, dùng SQL Developer để kiểm tra lại các đối tượng vừa được nhập vào schema của csdl1.

Trong câu lệnh này, tham số *fromuser* chỉ ra tên của schema mà từ đó tập tin dump được export, tham số *touser* chỉ ra tên của schema mà các đối tượng sẽ được import vào CSDL hiện hành.

Tương tự, thực hiện các lệnh import sau:

```
imp      userid=uimp2@xe      file=D:\csdl2.dmp      fromuser=csdl2
touser=uimp2
imp      userid=uimp3@xe      file=D:\csdl3.dmp      fromuser=csdl3
touser=uimp3
```

Bước 3: Thực hiện export các bảng dữ liệu locations và regions của người dùng hr. Sau đó, import các bảng này vào schema của người dùng uimp1 vẫn trong command concole, gõ vào:

```
exp          userid=hr/hr@xe          file=hrTabs.dmp          owner=hr
tables=(locations,regions)
imp userid=uimp1@xe file=D:\hrTabs.dmp fromuser=hr touser=uimp1
```

Câu 10: thử nghiệm import và export với Oracle Data Pump. Ta sử dụng người dùng HR để thực hiện các lệnh impdp và expdp.

Bước 1: thiết lập các cấu hình cần thiết cho Data Pump

Tạo một thư mục trên database server (giả sử D:\backup). Mở cửa sổ Command Console, truy cập chương trình SQLPlus bằng tài khoản SYS bằng lệnh:

```
sqlplus sys as sysdba
```

Nhập mật khẩu, và tiếp tục nhập vào các lệnh sau:

```
SQL> create directory expdir as 'D:\backup';
SQL> grant read, write on directory expdir to hr;
SQL> commit;
```

Sau lệnh này, HR có quyền ghi tập tin dump lên thư mục D:\backup trên database server.

Để xem thông tin các thư mục được tạo trên server, thực hiện câu truy vấn sau:

```
select * from dba_directories;
```

Bước 2: thực hiện export schema của HR

```
expdp hr/hr@xe schemas=hr directory=expdir dumpfile=hr.dmp
```

Bước 3: thực hiện export một vài bảng dữ liệu của hr

```
expdp hr/hr@xe tables=(regions,locations)
        directory=expdir dumpfile=hrTabs.dmp
```

Bước 4: thực hiện import từ các tập tin dump tạo từ các bước trước vào schema uimp1. Gán quyền đọc trên thư mục đối tượng uimp1.

Đăng nhập chương trình SQLPlus bằng tài khoản SYS, thực hiện lệnh:

```
SQL> grant read, write on directory expdir to uimp1;
```

Vào command console, gõ vào lệnh sau:

```
impdp          uimp1          @xe          directory=expdir          dumpfile=hr.dmp
remap_schema=hr:uimp1
```

Kiểm tra lại các thay đổi trong schema uimp1 bằng SQL Developer

Câu 11: thực hiện export ở mức độ CSDL, ta cần sử dụng người dùng SYS hoặc SYSTEM để có đầy đủ quyền thực hiện export toàn bộ CSDL. Mở cửa sổ Terminal, gõ vào:

```
expdp system@xe full=y directory=expdir dumpfile=full%U.dmp
        filesize=100M logfile=fullexp.log
```

Khi kết thúc thành công, các tập tin dump và log sẽ được lưu vào thư mục data_pump_dir (xem chi tiết trong bảng dba_directories). Ngoài ra, độ lớn của tập tin dump tỉ lệ thuận với độ lớn của CSDL, điều này có thể đụng độ với độ lớn tối đa của tập tin trong hệ điều hành. Vì vậy, ta cần giới hạn độ lớn tối đa của tập tin dump bằng

tham số `filesize`, tham số `%U` yêu cầu đặt tên các tập tin dump có phần hậu tố khác nhau.

3 Bài tập tự làm

Câu 1: Thử nghiệm kết nối từ xa

Chọn máy của một bạn trong phòng thực hành. Tạo kết nối đến CSDL trên máy của bạn bằng SQL*Plus và Oracle SQL Developer với tài khoản của người dùng SYS; xem các người dùng có sẵn trên máy đó, tạo 1 người dùng mới với các quyền cần thiết, và tiến hành import dữ liệu từ người dùng scott vào người dùng mới này.

Chú ý: để lấy địa chỉ IP của máy, vào Terminal, gõ lệnh `ipconfig`

Câu 2: Kết nối CSDL bằng SQL Developer

Tài khoản SCOTT và HR mặc định bị khóa khi mới cài đặt Oracle. Hãy kết nối bằng tài khoản sys để kích hoạt lại và thay đổi mật khẩu cho các tài khoản này. Tạo kết nối mới từ 2 người dùng này.

Câu 3: Quan sát các đối tượng CSDL

Quan sát cấu trúc các bảng, kiểu dữ liệu của các cột và sự thông thương giữa các bảng này trong lược đồ của người dùng SCOTT và HR.

Câu 4: Cấp quyền truy cập

Hãy viết các lệnh cấp quyền tương ứng để người dùng SCOTT và HR có thể thêm, sửa, xóa dữ liệu trên các bảng của nhau. Viết một số lệnh thêm, sửa, xóa để kiểm tra quyền xem đã có hiệu lực chưa.

BÀI 2. QUẢN LÝ BẢNG DỮ LIỆU

1 Nội dung

- Quản lý người dùng, quyền và role.
- Tạo/xóa/thay đổi bảng dữ liệu
- Thêm/xóa/thay đổi dữ liệu trong một bảng.
- Tạo các ràng buộc toàn vẹn trên bảng.

2 Bài tập có hướng dẫn

Câu 1: Tạo bảng có sử dụng biểu thức chính quy để kiểm tra miền trị cho dữ liệu.

Viết lệnh tạo bảng NhaXB (**MaNXB**, TenNXB, ThPho, Qgia). Trong đó cột MaNXB phải bắt đầu bằng một chữ hoa, một con số 9, một con số từ 1 đến 9, một con số từ 0 đến 9 hoặc mang 1 trong các giá trị '1389', '0736', '0877', '1622', '1756'. Cột Qgia có trị mặc định là VietNam.

```
CREATE TABLE NhaXB (  
  MaNXB char(4) NOT NULL CONSTRAINT PK_NXB PRIMARY KEY  
  CHECK (MaNXB IN ('1389', '0736', '0877', '1622', '1756')  
    or regexp_like(MaNXB, '[A-Z]9[1-9]\d')),  
  TenNXB varchar2(40) NULL,  
  ThPho varchar2(20) NULL,  
  QGia varchar2(30) DEFAULT 'VietNam' NULL ) ;
```

Thử xen dữ liệu đúng theo mẫu:

```
insert into NhaXB (manxb, tennxb, thpho)  
values('9980', 'NXB Van Hoa 1', 'TP HCM');
```

Câu 2: Tạo người dùng, cấp quyền và import dữ liệu vào người dùng này

- a) Tạo một người dùng có tên là **baohiem** với các tùy chọn về tablespace, quota,... thích hợp

```
create user baohiem identified by bh  
default tablespace users quota 2M on users;
```

- b) Cấp các quyền cần thiết để người dùng này có thể kết nối, tạo bảng và thực hiện import các bảng dữ liệu từ schema **baohiem** (tập tin baohiem.dmp được cung cấp sẵn).

CSDL về **quản lý bảo hiểm** gồm các quan hệ như sau:

1. DONVI (**MaDV**, TenDV, Dchi, Tel): Mỗi đơn vị có một mã phân biệt, một tên phân biệt, địa chỉ và số điện thoại liên lạc. Số ĐT phải nhập theo mẫu (xxx) xxx-xxxx.
2. KH (**MaKH**, HoTen, NamSinh, Phai, MaDV): Mỗi khách hàng có một mã phân biệt, một họ tên, năm sinh, phái (chỉ mang trị 0 hoặc 1) và thuộc một đơn vị nào đó.
3. LOAIBH (**MaLoai**, TenLoai, MucPhi): Mỗi loại bảo hiểm có một mã phân biệt, một tên phân biệt và một mức phí (số tiền phải đóng cho một tháng), phải lớn hơn 0, đơn vị tính là ngàn đồng.

4. THEBH (**MaLoai, MaKH, NgayBD**, ThoiHan): Mỗi thẻ bảo hiểm được xác định duy nhất qua mã loại, mã khách hàng và ngày bắt đầu có hiệu lực. Mỗi thẻ có một thời hạn, chính là số tháng mà khách mua bảo hiểm, giá trị này phải lớn hơn 0.

```
grant create session, create table to qlyBH;  
imp      userid=qlyBH/hqtcsdl      file=d:\qlbh.dmp      fromuser=qlbh  
touser=qlyBH
```

Đăng nhập vào người dùng baohiem và thực hiện tiếp các câu sau:

Câu 3 : Sửa cấu trúc bảng, thêm ràng buộc, cập nhật dữ liệu trong bảng

- a) Hãy viết các lệnh DDL để thiết lập các ràng buộc về khóa chính, khóa ngoại, khóa duy nhất và ràng buộc miền trị như mô tả của từng bảng.

```
alter table donvi add constraint pk_dv primary key (madv)  
                add constraint uk_tendv unique (tendv) ;  
alter table kh add constraint pk_kh primary key (makh)  
                add constraint fk_dv foreign key (madv) references  
donvi (madv)  
add constraint ck_phai check (phai in (0,1)  
add check ( regexp_like ( trim(both ' ' from tel),  
                        '^(\d{3}) \d{3}-\d{4}$' ) ));  
alter table loaibh add constraint pk_loai primary key (maloi)  
add constraint uk_tenloai unique (tenloai)  
add constraint ck_mucphi check(mucphi>0);  
alter table thebh add primary key (maloi, MAKH, ngaybd)  
                add foreign key (maloi) references loaibh(maloi)  
                add foreign key (makh) references kh(makh)  
add constraint ck_thoihan check (thoihan >0);
```

- b) Hãy viết lệnh để thêm vào bảng THEBH cột ConHL (còn hiệu lực) có kiểu số nguyên và chỉ được mang một trong hai giá trị 0 (hết hiệu lực) và 1 (còn hiệu lực). Viết lệnh cập nhật dữ liệu cho cột vừa tạo.

```
alter table thebh add (ConHL int check(conHL in (0,1))) ;  
update thebh set conHL = 0 where ngaykt <= sysdate;  
update thebh set conHL = 1 where ngaykt > sysdate;
```

- c) Hãy viết lệnh để thêm vào bảng THEBH cột NgayKT (ngày kết thúc hiệu lực) kiểu ngày, và có giá trị phải bằng ngày bắt đầu cộng thời hạn. Hãy viết lệnh cập nhật dữ liệu cho cột vừa tạo.

```
alter table thebh add (NgayKT date)  
add check (ngayKT =add_months(ngayBD,thoihan);  
update thebh set ngayKT =add_months(ngayBD,thoihan);
```

- d) Tất cả các khách hàng đã từng mua bảo hiểm y tế của ‘So Dien Luc’ muốn mua tiếp 6 tháng BH y tế này từ hôm nay. Hãy viết lệnh insert để thêm những thẻ BH này một cách tự động.

```
Insert into thebh (maloi, ngaybd, makh, thoihan, ngaykt,  
conhl)  
select distinct a.maloi,sysdate,a.makh, 6, add_months(sysdate,  
6), 1  
from thebh a, kh b, donvi c, loaibh d
```

where a.makh = b.makh and b.madv = c.madv and d.maloai = a.maloai

and tendv = 'SO DIEN LUC' and tenloai = 'BH Y TE';

Câu 4: Tạo bảng mới, cập nhật dữ liệu

- a) Tạo bảng tên THEBH_QUAHAN có cấu trúc giống hết bảng THEBH và dữ liệu bao gồm những thẻ bảo hiểm đã quá hạn của khách hàng.

```
create table THEBH_QUA_HAN as
select * from thebh where conhl = 0;
```

- b) Xóa đi những thẻ bảo hiểm quá hạn trong bảng THEBH

```
Delete from THEBH where conhl = 0;
```

- c) Tạo bảng tên MUCPHI để ghi nhận thông tin lịch sử mỗi khi có sự cập nhật mức phí mới. Bảng này gồm các cột: mã loại BH, mức phí bảo hiểm mới, ngày bắt đầu tính (có giá trị mặc định là ngày hiện tại), người cập nhật (có giá trị mặc định là người dùng hiện tại). Cài đặt ràng buộc cho bảng.

```
Create table MUCPHI (maloai char(2) references loaibh(maloai),
mucphi int check (mucphi>0), ngaybd date default (sysdate),
nguoicn char(20) default (user), primary key (maloai,
ngaybd) );
```

3 Bài tập tự làm

Câu 1: Hãy tạo bảng NhanVien (**MaNV**, TenNV, MaNXB, NgayVao, MaSep). Với MaNV là một chuỗi 9 ký tự theo mẫu 3 ký đầu là ký tự chữ hoa, đến một con số từ 1 đến 9, kế tiếp là 4 con số bất kỳ, và cuối cùng là chữ F hoặc M. MaNXB là khóa ngoại tham chiếu bảng NhaXB, NgayVao có trị mặc định là ngày hiện tại, MaSep chính là MaNV của thủ trưởng trực tiếp, vì vậy nó tham chiếu khóa chính.

Câu 2: Tạo user **banhang** với các quyền cần thiết để đăng nhập CSDL và tạo bảng. Đăng nhập vào người dùng banhang và thực hiện tiếp các câu sau.

Một CSDL **quản lý bán hàng** gồm các bảng sau:

- LoaiHang(**MaLoai**, TenLoai)
- HangHoa(**MaHang**, TenHang, DVT, DonGia, *MaLoai*)
- KhachHang(**MaKH**, TenKH, DiaChi, SoDT, MSThue, SoTaiKhoan)
- HoaDon(**SoHD**, NgayLap, TyLeVAT, TongTriGia, *MaKH*)
- ChiTietHD(**SoHD**, **MaHang**, SoLuong, DonGiaBan)

Câu 3: Viết các lệnh DDL để tạo các bảng dữ liệu được mô tả ở trên. Sinh viên tự đề nghị kiểu dữ liệu cho các cột. Thiết lập đầy đủ các ràng buộc toàn vẹn gồm ràng buộc khó chính, khóa ngoại, khóa duy nhất cho TenLoai, TenHang, MSThue, SoTaiKhoan và ràng buộc miền trị gồm TyleVat >0, SoLuong >0, DonGia>0, DonGiaBan>0 số điện thoại theo mẫu xxx-xxx-xxxx hoặc xxxx-xxx-xxxx. Giá trị mặc định cho NgayLap là ngày hiện tại, TyLeVat là 0.2.

Câu 4: Các cột mã trong CSDL này đều được thiết kế ở chế độ tự tăng, thiết kế một sequence cho các bảng này.

Câu 5: Viết lệnh để thêm cột ThanhTien (thành tiền) vào bảng ChiTietHD, có giá trị phải bằng tích của số lượng và đơn giá bán. Viết lệnh cập nhật giá trị cho cột này.

Câu 6: Viết lệnh tạo bảng GIABAN để ghi nhận thông tin lịch sử mỗi khi có sự cập nhật đơn giá mới. Bảng này gồm các cột: mã hàng, đơn giá mới, ngày bắt đầu tính (có giá trị mặc định là ngày hiện tại), người cập nhật (có giá trị mặc định là người dùng hiện tại). Cài đặt mọi ràng buộc cho bảng.

Câu 7: Hệ thống có các loại người dùng sau:

- Khách hàng:
 - Có quyền `SELECT` trên tất cả các bảng.
 - Có quyền `UPDATE` trên bảng `KhachHang`.
- Bán hàng:
 - Có quyền `SELECT` trên tất cả các bảng.
 - Có quyền `INSERT` và `UPDATE` trên bảng `KhachHang`, `HoaDon` và `ChiTietHD`
- Thủ kho:
 - Có toàn quyền (`ALL`) trên bảng `LoaiHang` và `HangHoa`
- Quản trị:
 - Có toàn quyền trên tất cả các bảng.

Hãy viết lệnh để tạo các role tương ứng các loại người này.

Câu 8: Viết lệnh tạo một vài người dùng ứng với các role trên và kiểm tra thử.

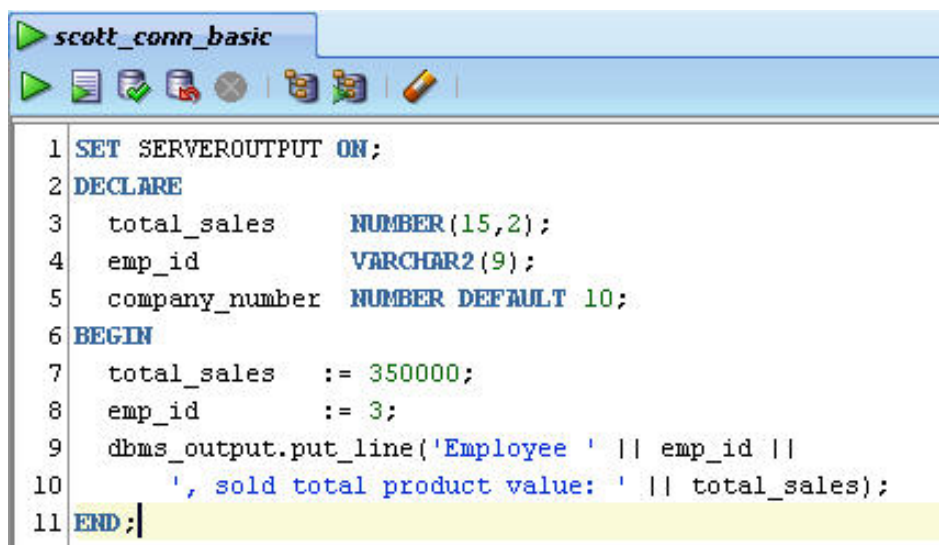
BÀI 3. LẬP TRÌNH PL/SQL

1 Nội dung

- Cấu trúc của một chương trình PL/SQL
- Các kiểu dữ liệu và các mệnh đề thông dụng trong PL/SQL
- Cấu trúc lập trình điều khiển trong PL/SQL
- Sử dụng cursor để xử lý dữ liệu trên kết quả câu truy vấn.

2 Bài tập có hướng dẫn

Câu 1: sử dụng các biến trong PL/SQL bằng cách khai báo, gán trị và in chúng ra màn hình.



Hình 36. Dùng các biến

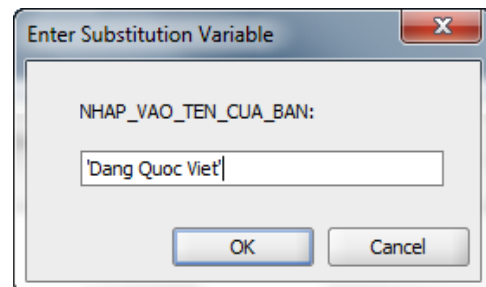
Câu 2: nhận dữ liệu do người dùng nhập vào với SQL Developer

Cú pháp: tên biến> := &chuỗi thông báo>;

Trong đó, chuỗi thông báo không chứa khoảng trắng mà được thay thế bởi dấu “_”. Khi nhập liệu kiểu chuỗi, nếu đặt dấu nháy (') theo dạng ‘&chuỗi thông báo’, thì khi nhập liệu, người dùng không cần gõ vào dấu nháy.

```

set serveroutput on;
declare
  ten varchar2(10);
begin
  ten := &Nhap_vao_ten_cua_ban;
  dbms_output.put_line
    ('chao ban ' || ten);
end;
  
```



Hình 37. Cửa sổ nhập dữ liệu

Khi thực thi chương trình, cửa sổ như hình bên hiện lên và ta nhập dữ liệu vào.

Câu 3: Thử sử dụng kiểu dữ liệu phức

- Tạo biến có kiểu dữ liệu tương tự kiểu dữ liệu trong cột Last_Name, và một biến có kiểu dữ liệu tương tự kiểu dữ liệu trong cột Salary của bảng Employees của schema hr.
- Truy vấn dữ liệu từ bảng liên quan để tìm thông tin của nhân viên có mã số 100 và lưu vào các biến này, sau đó in chúng ra màn hình.

```
SET SERVEROUTPUT ON;
DECLARE
    vEname employees.last_name%TYPE;
    /* kiểu dữ liệu của biến vEname sẽ lấy từ kiểu dữ liệu của
       cột last_name trong bảng employee */
    vSalary employees.salary%TYPE;
BEGIN
    SELECT last_name,salary INTO vEname,vSalary
    FROM employees
    WHERE employee_id=100;
    DBMS_OUTPUT.PUT_LINE('Name: ' || vEname || '.          Salary: ' ||
vSalary);
END;
```

Câu 4: Thử sử dụng các cấu trúc điều khiển

- Tìm nhân viên trong bảng Employees của schema hr với mã số là 120. Nếu tên anh ta là Matthew thì chào thân mật, ngược lại chỉ chào xã giao thôi.

```
SET SERVEROUTPUT ON;
DECLARE
    vEname employees.first_name%TYPE;
BEGIN
    SELECT first_name INTO vEname
    FROM employees
    WHERE employee_id=120;

    IF vEname='Matthew' THEN
        dbms_output.put_line('Hi ' || vEname);
    ELSE
        dbms_output.put_line('Hello' || vEname);
    END IF;
END;
```

- Hiển thị tên vùng của quốc gia có mã là 'CA'.

```
SET SERVEROUTPUT ON;
DECLARE
    vArea VARCHAR2(20) ;
BEGIN
    SELECT region_id INTO vArea
    FROM countries
    WHERE country_id='CA' ;
    CASE vArea
        WHEN 1 THEN vArea :='Europe' ;
        WHEN 2 THEN vArea :='America' ;
```

```
        WHEN 3 THEN vArea := 'Asia' ;
        ELSE vArea := 'Other' ;
    END CASE ;
    dbms_output.put_line('The Area is ' || vArea) ;
END ;
```

- Thử dùng vòng lặp FOR để in xuôi và in ngược

```
SET SERVEROUTPUT ON;
DECLARE
    counter NUMBER ;
BEGIN
    FOR counter IN 1..4          --tăng
    LOOP
        dbms_output.put(counter) ;
    END LOOP ;
    dbms_output.new_line ;
    FOR counter IN REVERSE 1..4  -- giảm
    LOOP
        dbms_output.put(counter) ;
    END LOOP ;
    dbms_output.new_line ;
END;
```

Câu 5: Sử dụng Cursor

Trong schema của Scott, sử dụng 2 bảng :

- Bảng **EMP**(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)

Diễn giải: Lưu trữ thông tin về nhân viên. Gồm có Mã nhân viên, Tên nhân viên, Nghề nghiệp, Mã người quản lý, Ngày gia nhập công ty, Lương, Thưởng, Mã phòng ban.

- Bảng **BONUS**(ENAME, JOB, SAL, COMM)

Diễn giải: Lưu trữ thông tin về tiền thưởng. Gồm có Tên nhân viên, Nghề nghiệp, Lương, Thưởng.

- a. **Yêu cầu:** Các nhân viên đã làm việc trên 25 năm mà có mức lương dưới 2000\$ thì tăng mức thưởng cho họ thêm 500\$, sau đó lưu vào bảng Bonus

Hãy mở cả hai bảng này lên xem có dữ liệu gì trước và sau khi bạn thực hiện đoạn lệnh này.

Ý tưởng:

- Tạo cursor chứa các nhân viên đã làm việc trên 25 năm mà có mức lương dưới 2000\$
- Duyệt từng dòng trong cursor để :
 - Cập nhật lại tiền thưởng trong bảng nhân viên
 - Lưu thông tin vào bảng tiền thưởng.

```
DECLARE
    CURSOR c_Emp IS
    select *
```

```
from emp
where extract(year from sysdate) - extract(year from
hiredate)> 25
and sal <2000;
-- Khai báo biến để chứa dòng dữ liệu của cursor
v_Emp c_Emp%rowtype;
BEGIN
-- Mở cursor
OPEN c_Emp;
-- Dừng vòng lặp để lấy hết dữ liệu
LOOP
-- Lấy một dòng dữ liệu từ cursor
FETCH c_Emp INTO v_Emp;
-- Thoát khỏi vòng lặp nếu đã lấy hết dữ liệu trong
cursor
EXIT WHEN c_Emp%notfound;
-- kiểm tra các giá trị null của cột tiền thưởng
if v_Emp.comm is null then
v_Emp.comm:=500;
else
v_Emp.comm:=v_Emp.comm+500;
end if;
-- Cập nhật lại tiền thưởng trong bảng Emp với mã nhân
viên
-- trong cursor hiện hành
UPDATE Emp
SET comm = v_Emp.comm
WHERE empno = v_Emp.empno;
--Lưu vào bảng Bonus
INSERT INTO BONUS(ename, job, sal, comm)
VALUES (v_Emp.ename, v_Emp.job, v_Emp.sal,
v_Emp.comm);
END LOOP;
COMMIT; -- lưu lại những thay đổi
--Đóng cursor
CLOSE c_Emp;
END;
```

b. Yêu cầu: Giả sử các nhân viên trong bảng EMP khi đã làm việc từ 28 năm trở lên sẽ được cho nghỉ hưu sớm, vì vậy ta cần xóa nhân viên đó trong bảng EMP và chuyển toàn bộ thông tin của họ sang bảng EMP_RETIRE.

Ý tưởng:

- Tạo bảng EMP_RETIRE để chứa các nhân viên nghỉ việc
- Tạo cursor lưu danh sách các nhân viên đã làm việc ≥ 28 năm, sau đó duyệt từng dòng trong cursor để:
 - Thêm dòng này vào bảng EMP_RETIRE
 - Xóa dòng này khỏi bảng nhân viên.

Trước hết tạo bảng EMP_RETIRE có cấu trúc giống bảng EMP

```
CREATE TABLE EMP_RETIRE (  
    EMPNO    NUMBER(4) NOT NULL,  
    ENAME    VARCHAR2(10),  
    JOB      VARCHAR2(9),  
    MGR      NUMBER(4),  
    HIREDATE DATE,  
    SAL      NUMBER(7,2),  
    COMM     NUMBER(7,2),  
    DEPTNO   NUMBER(2) );
```

Sau đó sẽ khai báo cursor để thực hiện:

```
DECLARE  
    -- Tìm các nhân viên có số năm làm việc >=28, khóa bảng  
    -- emp khi cursor này được mở  
    CURSOR cEmpRetire IS  
        Select *  
        From emp  
        where extract(year from sysdate) -  
              extract(year from hiredate) >= 28  
        FOR UPDATE;  
  
BEGIN  
    -- biến emp_rec được dùng trực tiếp để lấy từng dòng dữ liệu  
    -- trong cursor, ta dùng cách này thay cho việc khai báo biến  
    và  
    -- dùng lệnh FETCH  
    FOR emp_rec IN cEmpRetire  
    LOOP  
        -- Lấy dòng hiện hành của cursor để thêm vào bảng retire  
        INSERT INTO Emp_Retire(Empno, EName, Job, Mgr,  
                                Hiredate, Sal, Comm, Deptno)  
        VALUES (emp_rec.Empno, emp_rec.EName, emp_rec.Job,  
emp_rec.Mgr, emp_rec.Hiredate, emp_rec.Sal, emp_rec.Comm,  
emp_rec.Deptno);  
        -- Xóa dữ liệu trong Emp với mẫu tin giống với  
        -- mẫu tin trong cursor hiện hành  
        DELETE FROM Emp  
        WHERE CURRENT OF cEmpRetire;  
    END LOOP;  
END;
```

c. Yêu cầu: Tất cả các nhân viên mà giám đốc (Mgr) của họ tên là 'KING' sẽ được tăng lương thêm 5%.

Ý tưởng:

- Tạo cursor có tham số truyền vào là mã số của Manager
- Tìm nhân viên có tên là 'KING' để lấy mã số lưu vào một biến

- Sau đó duyệt từng dòng trong cursor với giá trị truyền vào là biến vừa tìm được để cập nhật lại mức lương của các nhân viên mà có mã số giám đốc trong cursor.

DECLARE

```
--Khai báo biến manager có kiểu dữ liệu giống với cột MGR
--trong bảng EMP
manager EMP.MGR%TYPE;
--Khi báo cursor có tham số truyền vào là mã số của manager
--và khóa cột SAL trong bảng EMP lại khi cursor này được mở
CURSOR C_Emp(mgr_no number) IS
    SELECT SAL
    FROM EMP
    WHERE MGR = mgr_no
    FOR UPDATE of SAL;
```

BEGIN

```
--Tìm mã số của nhân viên tên là KING lưu vào biến manager
SELECT empno INTO manager
FROM EMP
WHERE EName = 'KING';
--Khai báo không tường minh biến emp_rec trực tiếp trong
--vòng lặp for để lưu giá trị của cursor hiện hành
--ta dùng cách này thay cho việc khai báo tường minh một
```

biến

```
--sau đó dùng lệnh FETCH để chuyển dữ liệu vào biến đó
FOR emp_rec IN C_Emp(manager)
LOOP
    UPDATE scott.EMP
    SET sal = emp_rec.sal * 1.05
    WHERE CURRENT OF C_Emp;
END LOOP;
COMMIT;
```

END;

3 Bài tập tự làm

Câu 1: Quan sát cấu trúc và dữ liệu của các bảng có trong schema SCOTT và dùng PL/SQL để thực hiện các yêu cầu sau:

1. Viết đoạn chương trình tìm kiếm các dòng trong bảng Scott.EMP với biến được đưa từ ngoài vào là 1 biến có kiểu giống cột Job (emp.job%type) và đưa ra thông báo số dòng tìm được cũng như danh sách các dòng.
2. Liệt kê các cột ENAME, HIREDATE, SAL của nhân viên trong bảng Scott.Emp. Với điều kiện EMPNO bằng giá trị được đưa vào, sau đó kiểm tra:
 - Có phải mức lương lớn hơn 1200
 - Tên nhân viên có phải có chứa chữ T
 - Ngày gia nhập cơ quan có phải là tháng 12 (DEC)

Thử với các giá trị 7654, 7369, 7900, 7876.

3. Quan sát bảng Scott.BONUS, thực hiện cập nhật dữ liệu cho bảng này. Nếu nhân viên có chức vụ (job) “Manager” thì có giá trị COMM (phụ cấp) là 0,2.
4. Hiện thị ra màn hình mã, tên, ngày vào làm và lương của tất cả nhân viên theo chức vụ.
5. Dựa vào bảng Scott.SALGRADE, phân loại lương của tất cả nhân viên và hiện thị ra màn hình tên nhân viên theo từng loại.

Câu 2: Quan sát cấu trúc và dữ liệu của các bảng có trong schema HR (Human Resource – quản lý nhân sự) và dùng PL/SQL để thực hiện các yêu cầu sau:

1. Hiện thị ra màn hình tên tất cả các quốc gia phân chia theo châu lục.
2. Hiện thị ra màn hình bảng lương hàng tháng cho tất cả các nhân viên. Biết rằng lương = salary + commission_pct * salary.
3. Các lập trình viên (có mã nghề nghiệp là IT_PROG) đã làm việc trên 15 năm sẽ được tăng lương 10%. Hiện thị ra màn hình tổng số tiền sẽ phát cho các nhân viên này sau khi tăng lương.
4. Các nhân viên làm chung phòng với Tổng giám đốc (có Manager_id là NULL) sẽ được hưởng mức hệ số phụ cấp giống nhau là 0.25. Cập nhật giá trị này vào cột Commission_PCT.
5. Thêm cột *Tình trạng* có kiểu chuỗi ký tự vào bảng Employees. Khi một nhân viên nào đó thôi việc, hãy cập nhật thông tin của nhân viên đó vào bảng Job_History, cập nhật cột Tình trạng với giá trị ‘Fired’.

BÀI 4. HÀM – THỦ TỤC

1 Nội dung

- Tạo và gọi một thủ tục (procedure).
- Tạo và gọi một hàm (function).

2 Bài tập có hướng dẫn

Câu 1: Tạo thủ tục để xóa một nhân viên khỏi danh sách với tham số truyền vào là mã nhân viên

```
CREATE OR REPLACE PROCEDURE Del_Emp (p_EmpNo NUMBER) IS
BEGIN
    DELETE FROM scott.emp WHERE empno = p_EmpNo;
    COMMIT;
END;
```

Tìm xem nhân viên có mã 7654 có trong bảng không :

```
SELECT * FROM Scott.Emp WHERE empno = 7654;
```

Thực thi thủ tục này :

```
EXECUTE Del_Emp(7654);
```

Tìm lại xem nhân viên có mã 7654 còn trong bảng không :

```
SELECT * FROM Scott.Emp WHERE empno = 7654;
```

Câu 2: Viết một hàm để lấy tổng tiền lương của một phòng ban nào đó

```
CREATE OR REPLACE FUNCTION get_dept_salary(dno number) RETURN
NUMBER
IS
    all_sal NUMBER;
BEGIN
    all_sal := 0;
    FOR emp_sal IN
        (SELECT SAL FROM scott.EMP WHERE DEPTNO = dno
         AND SAL IS NOT NULL)
    LOOP
        all_sal := all_sal + emp_sal.sal;
    END LOOP;
    RETURN all_sal;
END get_dept_salary;
```

Gọi thủ hàm này với mã phòng ban là 30

```
SELECT get_dept_salary(30) AS Tsal FROM DUAL;
```

Gọi thủ hàm này với mã phòng ban là 30 và truyền kết quả vào một biến :

```
DECLARE
    result NUMBER;
BEGIN
```

```
result := get_dept_salary(30);
dbms_output.put_line(result);
END;
```

Câu 3: Sử dụng bảng hr.Jobs(Job_id, Job_Title, Min_Salary, Max_Salary): Tạo thủ tục MucLuong nhận vào một Job_id, trả về lương thấp nhất và cao nhất ứng với mã công việc này. Thử gọi thủ tục với các giá trị: IT_PROG, SA_MAN.

```
Create or replace procedure MucLuong (p_jobid jobs.job_id%TYPE,
min_lg OUT jobs.min_salary%type,
max_lg OUT jobs.max_salary%type) IS
begin
select min_salary, max_salary into min_lg, max_lg
from jobs
where JOB_ID = p_jobid;

exception
when no_data_found then
dbms_output.put_line ('Ma cong viec ' || p_jobid ||
' khong tim thay');

when others then
dbms_output.put_line ('Khong biet loi gi');
end;
```

Thử gọi thủ tục:

```
set serveroutput on;
declare
min1 jobs.min_salary%type;
max1 jobs.max_salary%type;
begin
MucLuong('&Nhap_ma_cong_viec',min1, max1);
dbms_output.put_line ('Min = ' || to_char(min1) || ' Max = '
|| to_char(max1));
end;
```

3 Bài tập tự làm

Câu 1: thực hiện các yêu cầu sau

1. Tạo thủ tục TenThang nhận vào 1 ngày tháng, in ra màn hình tên tháng theo tiếng Việt. Thử gọi thủ tục với ngày do người dùng nhập vào.
2. Tạo hàm LayTen nhận vào họ tên của một người. Hàm trả về tên của người đó.

Câu 2: Quan sát cấu trúc và dữ liệu của các bảng có trong schema SCOTT và dùng PL/SQL để thực hiện các yêu cầu sau:

1. Các nhân viên làm việc không tốt sẽ bị trừ tiền thưởng. Viết một thủ tục để giảm tiền thưởng cho nhân viên. Với mã nhân viên và phần trăm giảm được truyền vào từ tham số.

2. Viết một hàm để lấy mã số của nhân viên nào có tiền lương cao nhất.
3. Viết một thủ tục nhận vào mã nhân viên, và in ra họ tên của tất cả nhân viên quản lý trực tiếp cũng như gián tiếp của nhân viên này.

Câu 3: Quan sát cấu trúc và dữ liệu của các bảng có trong schema HR (Human Resource – quản lý nhân sự) và dùng PL/SQL để thực hiện các yêu cầu sau:

1. Viết lệnh truy vấn bảng hr.employees để lấy cột mã nhân viên và họ tên của từng nhân viên có độ dài họ tên nhiều hơn 15 ký tự.
2. Sử dụng bảng hr.employees và hr.jobs: Tạo hàm LuongHopLy nhận vào mã nhân viên, và lương mới, sử dụng thủ tục MucLuong đã tạo ở câu 3 trong phần 2 để kiểm tra xem lương mới của người này có nằm trong khoảng lương ứng với công việc của người đó không. Nếu có, trả về True, ngược lại trả về False (kiểu BOOLEAN). Thử với giá trị (105, 7000), (206, 10000)
3. Sử dụng bảng Dept: Tạo thủ tục ThemPhongBan nhận vào tên phòng, và vị trí phòng để xen vào bảng DEPT với 3 cột:
 - Deptno: Chương trình tự tìm số lớn nhất hiện có + 10
 - Dname: Tên phòng nhận vào
 - Loc: vị trí nhận vàoThử gọi thủ tục với giá trị đầu vào do người dùng nhập.

Câu 4: Quan sát cấu trúc và dữ liệu của các bảng có trong schema csdl5 (sinh viên tự thực hiện import) và dùng PL/SQL để thực hiện các yêu cầu sau:

1. Viết thủ tục GoiTen nhận vào tham số mã giáo viên. Hãy in ra tên của giáo viên đó cùng với danh xưng là 'Thầy' nếu giáo viên là nam, và 'Cô' nếu giáo viên là nữ.
2. Viết hàm TongGioChuan nhận vào tham số mã giáo viên, hàm sẽ trả về tổng số tiết chuẩn mà giáo viên này đã giảng dạy. Biết rằng: Số tiết qui chuẩn = SOTIET * HESO; trong đó HESO là 1 nếu sĩ số lớp thuộc [0..80], là 1.2 nếu sĩ số lớp thuộc (80..120], là 1.4 nếu sĩ số lớp thuộc (120..150], là 1.5 nếu sĩ số lớp từ 150 trở lên.
3. Viết hàm TamUngCN trả về họ tên của giáo viên có tạm ứng nhiều nhất từ trước đến nay.
4. Viết hàm NgayTamUngGanNhat trả về ngày có giáo viên tạm ứng gần đây nhất.
5. Sử dụng hàm TongGioChuan đã tạo ở câu 2, viết thủ tục TongGioTheoCDanh nhận vào tham số tên chức danh, thủ tục in ra họ tên GV cùng với tổng số giờ qui chuẩn của từng giáo viên thuộc chức danh đó.
6. Viết thủ tục TgTamUngTrongTG nhận vào 2 mốc thời gian (từ ngày, đến ngày), thủ tục in ra danh sách gồm họ tên, tổng số tiền tạm ứng của từng GV trong khoảng thời gian đã cho.
7. Sử dụng hàm TongGioChuan đã tạo ở câu 2, viết thủ tục TienGD nhận vào mã giáo viên, trả về tổng giờ chuẩn, số giờ vượt và số tiền giảng dạy của GV đó.
8. Sử dụng thủ tục TienGD vừa tạo ở câu 7, viết thủ tục InBangLuong hiển thị danh sách gồm họ tên, tổng giờ chuẩn đã dạy, số giờ vượt, và số tiền giảng dạy của từng GV.

BÀI 5. TRIGGER

1 Nội dung

- Tạo và sử dụng Trigger.

2 Bài tập có hướng dẫn

Câu 1: Giả sử trong bảng DEPT của lược đồ Scott có thêm cột BUDGET chứa ngân sách tối đa của từng phòng ban dành cho việc trả lương. Xây dựng trigger để kiểm tra rằng khi cập nhật tiền lương thì tổng lương của tất cả các nhân viên trong một phòng ban không được vượt quá ngân sách của nó.

- Thêm cột Budget kiểu number(10) vào bảng DEPT
- Tìm tổng tiền lương của tất cả nhân viên theo từng phòng ban để ước lượng giá trị phải nhập cho cột Budget

```
SELECT deptno, Sum(Sal) As Total_Sal FROM scott.emp
Group By deptno;
```

- Nhập vào giá trị cho cột Budget lớn hơn Total_Sal chút ít
- Tạo trigger trên 2 cột : lương và mã phòng của bảng Emp

```
CREATE OR REPLACE TRIGGER check_budget_EMP
AFTER INSERT OR UPDATE OF SAL, DEPTNO ON Scott.EMP
DECLARE
--Khai báo cursor chứa mã phòng và ngân sách từ bảng DeptT
CURSOR DEPT_CUR IS
    SELECT DEPTNO, BUDGET FROM Scott.DEPT;
--Khai báo biến để lưu mã phòng(DNO), Tổng lương từ ngân
--sách(ALLSAL), và tổng lương từ các nhân viên (DEPT_SAL)
DNO Scott.DEPT.DEPTNO%TYPE;
ALLSAL Scott.DEPT.BUDGET%TYPE;
DEPT_SAL NUMBER;
BEGIN
    OPEN DEPT_CUR;
    LOOP
        --lấy ngân sách của từng phòng
        FETCH DEPT_CUR INTO DNO, ALLSAL;
        EXIT WHEN DEPT_CUR%NOTFOUND;
        --Tính tổng lương của các nhân viên trong phòng đó
        SELECT sum(SAL) INTO DEPT_SAL FROM scott.EMP
        WHERE DEPTNO = DNO;
        --Nếu Tổng lương > Ngân sách thì báo lỗi
        IF DEPT_SAL > ALLSAL then
            raise_application_error(-20325, 'Tổng lương
            trong phong ' || to_char(DNO) || ' da vuot
            qua ngan sach');
        END IF ;
    END LOOP;
```

```
CLOSE DEPT_CUR;
```

```
END;
```

Câu 2: Giả sử để theo dõi việc chỉnh sửa những dữ liệu nhạy cảm (liên quan đến tiền bạc chẳng hạn) của người dùng. Người ta mong muốn với những thông tin đã sửa đổi phải được ghi nhận lại: Ai đã sửa, thời gian nào, giá trị cũ là gì, giá trị mới là gì. Chúng ta sẽ xây dựng trigger để giải quyết vấn đề này. *Thực hiện trên bảng Emp nếu người dùng sửa lương thì thông tin này phải được lưu lại.*

Tạo bảng mới để lưu thông tin như sau :

```
CREATE TABLE Change_Sal_EMP(  
    UserName          VARCHAR2(20),  
    MODIFY_TIME       DATE,  
    EMPNO             NUMBER(4),  
    Old_Sal           NUMBER(7),  
    New_Sal           NUMBER(7)  
);
```

Tạo Trigger

```
CREATE OR REPLACE TRIGGER store_change_sal_EMP  
AFTER UPDATE OF SAL on scott.EMP  
FOR EACH ROW  
BEGIN  
    -- User là một hàm có sẵn trong hệ thống cho biết ai đang kết  
    nối CSDL  
    INSERT INTO Change_Sal_EMP(UserName, MODIFY_TIME, EMPNO,  
                                Old_Sal, New_Sal)  
    VALUES (User, Sysdate, :new.Empno, :old.sal, :new.sal);  
END;
```

Hãy thử cập nhật lương của một nhân viên nào đó xem, sau đó mở bảng Change_Sal_EMP và quan sát kết quả.

Chú ý : Trong lúc cập nhật dữ liệu có thể không thỏa các trigger đã tạo ở các ví dụ trước đây nên việc cập nhật không thực hiện được. Hãy xóa các trigger mà bạn đã tạo ở phần trước trong bảng Emp đi.

Câu 3: Trong schema baohiem (đã tạo ở bài 2), hãy tạo trigger có chức năng ngăn không cho thêm thẻ BH khi khách hàng này đã mua bảo hiểm loại này và thẻ đã mua vẫn còn hiệu lực

```
create trigger Da_co_BH before insert on thebh for each row  
declare  
    sodong int;  
begin  
    select count(*) into sodong from thebh  
    where makh = :new.makh and maloi = :new.maloi  
    and ngaykt<sysdate;  
  
    if sodong >0 then  
        raise_application_error(-20111, 'KH nay hien van con  
        duoc BH loai nay');
```

```
end if;  
end;
```

Thử xem trigger có hiệu lực chưa:

```
insert into thebh (maloi, ngaybd, makh, thoihan, ngaykt, conhl)  
values('YT', sysdate, '006', 6, add_months(sysdate,6), 1);  
insert into thebh (maloi, ngaybd, makh, thoihan, ngaykt, conhl)  
values('YT', sysdate+1, '006', 6, add_months(sysdate+1,6), 1);
```

Câu 4: Trong schema baohiem đã tạo ở bài 2, hãy tạo trigger để chèn dữ liệu vào bảng MUCPHI mỗi khi người dùng cập nhật mức phí hoặc thêm một loại bảo hiểm với phí mới.

```
create or replace trigger CN_Phi after insert or update on  
LoaiBH  
for each row  
declare  
begin  
insert into mucphi values (maloi, mucphi, ngaybd, nguoiicn)  
(:new.maloi, :new.mucphi, sysdate, user);  
end;
```

Thử cập nhật mức phí của một loại bảo hiểm:

```
update loaibh set mucphi=8 where maloi='TS';
```

3 Bài tập tự làm

Câu 1: Tạo trigger tương tự ở câu 2 để có thể theo dõi việc xóa dữ liệu trong bảng Emp.

Câu 2: Trong schema csdl5, viết một trigger theo dõi việc xen dữ liệu vào bảng TAMUNG với quy định: mỗi giáo viên chỉ được tạm ứng tối đa 2 lần và tối đa 3000000 đồng trong một tháng.

Câu 3: Trong schema banhang đã tạo ở bài 2, hãy tạo trigger để chèn dữ liệu vào bảng GIABAN mỗi khi người dùng cập nhật đơn giá mới hoặc thêm mặt hàng với giá mới vào bảng HangHoa.

Câu 4: Trong schema banhang, hãy tạo trigger để cập nhật giá trị cho cột TongTriGia của HoaDon mỗi khi người dùng cập nhật SoLuong hoặc DonGiaBan, hoặc thêm dòng vào trong bảng ChiTietHD.

Câu 5: Trong schema banhang, hãy tạo trigger để kiểm tra việc cập nhật đơn giá của một mặt hàng: giá mới không được cao hơn giá cũ quá 10%.

BÀI 6. QUẢN LÝ GIAO DỊCH, SẠO LƯU VÀ PHỤC HỒI

1 Nội dung

- Các khái niệm về sao lưu và phục hồi trong Oracle
- Các chiến thuật sao lưu và phục hồi CSDL
- Sử dụng RMAN (Recovery MANager) để sao lưu và phục hồi CSDL

2 Bài tập có hướng dẫn

Trong các bài tập sau, chúng ta sẽ tìm hiểu các phương pháp sao lưu và phục hồi luận lý và vật lý.

Câu 1: thử nghiệm sao lưu và phục hồi vật lý với RMAN

Bước 1: đăng nhập RMAN, vào Terminal, gõ vào lệnh:

```
rman target sys
```

Khi đăng nhập thành công, trên màn hình hiện ra dòng RMAN>

Bước 2: backup toàn bộ CSDL (chú ý: lúc này CSDL ở chế độ ARCHIVELOG)

```
RMAN> backup database;
```

Quan sát và ghi nhận thông số của kết quả. Hãy tìm đến thư mục chứa các tập tin backup.

Bước 3: giả lập CSDL bị lỗi, xóa người dùng SCOTT. Đăng nhập SQLPlus, ghi nhận thời gian và gõ vào

```
SQL> drop user scott;
```

Bước 4: phục hồi CSDL, đăng nhập RMAN và thực hiện các lệnh sau

```
RMAN> shutdown immediate;
```

```
RMAN> startup mount;
```

```
RMAN> restore database;
```

```
RMAN> recover database until time "to_date('<date>',  
'<format>')";
```

```
RMAN> alter database open resetlogs;
```

Trong đó, <date> là thời điểm ghi nhận trước khi xóa SCOTT, ví dụ "to_date('27/08/2009 13:00:00', 'DD/MM/YYYY HH24:MI:SS')";

Kiểm tra lại dữ liệu sau khi phục hồi.

Câu 2: thử nghiệm câu lệnh COMMIT và ROLLBACK trong cửa sổ SQLPlus

Đăng nhập vào lược đồ của một người dùng bất kỳ và thực thi câu lệnh sau để tạo bảng:

```
CREATE TABLE test  
(id NUMBER(4) PRIMARY KEY,  
notes VARCHAR(50),  
count NUMBER(4),  
created DATE);
```

Thực hiện tuần tự các lệnh SQL sau

```
INSERT INTO test(id, notes) VALUES(1, 'Test 1');
```

```
INSERT INTO test(id, notes) VALUES(2, 'Test 2');  
COMMIT;
```

Quan sát sự thay đổi dữ liệu trong bảng Test:

```
SELECT * FROM test;
```

Thực hiện tiếp các câu lệnh sau:

```
INSERT INTO test(id, notes) VALUES(3, 'Test 3 rollback');
```

Quan sát kết quả:

```
SELECT * FROM test;
```

Thực hiện cuộn lại:

```
ROLLBACK;
```

Quan sát kết quả:

```
SELECT * FROM test;
```

Câu 3: Thử nghiệm các trường hợp Oracle kết thúc các giao dịch không tường minh

Trường hợp 1: giao dịch kết thúc khi thực hiện thành công một lệnh DDL, thực hiện lần lượt các lệnh sau:

```
INSERT INTO test(id, notes) VALUES(4, 'Test 4');  
INSERT INTO test(id, notes) VALUES(5, 'Test 5');  
CREATE TABLE test_cloned  
    (id NUMBER(4) PRIMARY KEY,  
     notes VARCHAR(50),  
     count NUMBER(4),  
     created DATE);  
ROLLBACK;
```

Sinh viên quan sát dữ liệu trong bảng Test và cho nhận xét.

Trường hợp 2: giao dịch kết thúc khi phiên làm việc của người dùng kết thúc, thực hiện lần lượt các lệnh sau:

```
INSERT INTO test(id, notes) VALUES(4, 'Test 4');  
INSERT INTO test(id, notes) VALUES(5, 'Test 5');
```

Sinh viên tắt kết nối của người dùng hiện tại và đăng nhập lại sau đó, thực hiện lệnh SELECT để xem kết quả trong bảng Test.

Câu 4: Thử nghiệm giao dịch với mức cô lập READ ONLY

Thực hiện các lệnh SQL sau:

```
SET TRANSACTION READ ONLY;
```

```
INSERT INTO test(id, notes) VALUES(6, 'Test 6'); – Quan sát kết quả
```

Sinh viên giữ nguyên phiên làm việc hiện tại, đăng nhập vào lược đồ của một người dùng khác và thực hiện các lệnh sau (chú ý: người dùng này phải có toàn quyền trên bảng Test)

```
DELETE FROM TEST WHERE id = 5;
```

COMMIT;

Quay lại phiên làm việc của người dùng trước và thực hiện các lệnh sau:

SELECT * FROM test; -- Quan sát kết quả

COMMIT;

SELECT * FROM test; -- Quan sát kết quả

Câu 5: Thử nghiệm với các khoá trong Oracle

Sinh viên mở hai phiên làm việc song song (đăng nhập bằng 2 người dùng khác nhau) và thực hiện tuần tự các lệnh sau (theo thứ tự từ trên xuống dưới theo bảng):

Phiên làm việc 1	Phiên làm việc 2
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; UPDATE Test SET count = 1 WHERE id = 1; COMMIT; SELECT * FROM Test; (Sinh viên quan sát kết quả và cho nhận xét) SELECT * FROM Test; (Sinh viên quan sát kết quả và cho nhận xét)	 SET TRANSACTION ISOLATION LEVEL READ COMMITTED; UPDATE Test SET count = 2 WHERE id = 1; (Sinh viên quan sát kết quả và cho nhận xét) (Sinh viên quan sát kết quả và cho nhận xét) COMMIT;

Câu 6: Thử nghiệm với deadlock

Sinh viên mở hai phiên làm việc song song (đăng nhập bằng 2 người dùng khác nhau) và thực hiện tuần tự các lệnh sau (theo thứ tự từ trên xuống dưới theo bảng):

Phiên làm việc 1	Phiên làm việc 2
UPDATE Test SET count = 2 WHERE id = 1; UPDATE Test SET count = 3 WHERE id = 2; (Sinh viên quan sát kết quả và cho nhận xét)	 UPDATE Test SET count = 2 WHERE id = 2; UPDATE Test SET count = 3 WHERE id = 1; (Sinh viên quan sát kết quả và cho nhận xét)

3 Bài tập tự làm

Câu 1: thực hiện các lệnh imp/exp và impdp/expdp

Thực hiện export dữ liệu từ uimp1, uimp2 và uimp3 bằng các tài khoản người dùng tương ứng. Tạo người dùng importer với các quyền đăng nhập và tạo bảng, thực hiện import các dữ liệu trong tập tin dump vừa tạo và schema này.

Câu 2: thực hiện sao lưu và phục hồi CSDL ở chế độ NOARCHIVELOG

Dựa trên các mô tả về sao lưu tăng dần (incremental backup) ở trang 40 và 41. Thử nghiệm sao lưu và phục hồi bằng cả 2 giải thuật: sai khác (differential) và tích lũy (cumulative). Chú ý: trước mỗi lần sao lưu, cần thay đổi vài dữ liệu hoặc đối tượng để đối chiếu kết quả phục hồi.

Khuyến khích sinh viên tìm hiểu các thông tin cần thiết bằng các công cụ tìm kiếm trên Internet để giải bài tập này.

Câu 3: Sinh viên thực hiện các lệnh cần thiết để kiểm chứng tính chính xác của bảng tương thích giữa các khoá trong phần lý thuyết bài 3.

ÔN TẬP

1 Nội dung

Trong bài ôn tập này, tất cả các kiến thức sinh viên đã được tìm hiểu trong các bài trước sẽ được áp dụng vào một CSDL cụ thể. Qua đó, sinh viên sẽ được ôn lại các kiến thức sau:

- Các lệnh DDL liên quan đến bảng dữ liệu, index, sequence...
- Lệnh truy vấn SQL.
- Tạo hàm, thủ tục và trigger với ngôn ngữ PL/SQL.
- Quản lý người dùng, phân quyền và nhóm.
- Quản trị sao lưu và phục hồi CSDL.

2 Bài tập

CSDL Northwind có cấu trúc như sau:

- LoạiHang(**MaLH**, TenLH, MoTa)

Chứa thông tin mã loại hàng, tên và các mô tả chi tiết các danh mục loại hàng.

- KháchHang(**MaKH**, TenCongTy, TenNguoiLH, DanhNghiaLH, DiaChi, ThanhPho, Vung, MaBuuDien, QuocGia, SoDT, Fax)

Lưu trữ thông tin chi tiết của khách hàng gồm mã khách hàng (công ty), tên công ty, tên người liên hệ, danh nghĩa liên hệ, địa chỉ, thành phố, vùng, mã bưu điện, nước, số điện thoại, số fax.

- NhanVien(**MaNV**, TenNV, HoNV, ChucVu, ChucDanhLS, NgaySinh, NgayVaoLam, DiaChi, ThanhPho, Vung, MaBuuDien, QuocGia, DTNha, SoNhanh, GhiChu)

Lưu trữ thông tin chi tiết của nhân viên gồm mã nhân viên, họ, tên, chức vụ, chức danh lịch sự, ngày sinh, ngày vào làm, địa chỉ, thành phố, vùng, mã bưu điện, nước, điện thoại nhà, số điện thoại nhánh, ghi chú.

- ChiTietDatHang(**MaDonDH**, **MaHang**, DonGia, SoLuong, GiamGia)

Thông tin chi tiết đặt hàng gồm mã đơn đặt hàng, mã sản phẩm, đơn giá, số lượng, giảm giá.

- DonDatHang (**MaDonDH**, MaKH, MaNV, NgayDatHang, NgayPhaiGiao, NgayGoiHang, MaCTyVC, GiaVC, TenNguoiNhan, DCDen, TPDen, VungDen, MaBDDen, QuocGiaDen)

Bảng lưu trữ thông tin về Hóa đơn như mã hóa đơn, mã khách hàng đặt, mã nhân viên tiếp nhận, ngày đặt hàng, ngày phải giao, ngày gửi hàng, mã công ty vận chuyển, giá vận chuyển, tên người nhận, địa chỉ đến, thành phố đến, vùng đến, mã bưu điện đến, quốc gia đến.

- HangHoa(**MaHang**, TenHang, MaCtyCC, MaLH, SLTrenDV, DonGia, TonKho, HetSanXuat)

Bảng lưu trữ thông tin chi tiết về sản phẩm gồm: Mã sản phẩm, tên sản phẩm, mã công ty cung cấp, mã loại hàng, số lượng hàng trên 1 đơn vị, đơn giá, số lượng còn tồn kho và tình trạng còn sản xuất (0) hay không (-1).

- Vung(**MaVung**, GiaiThich)

Bảng lưu trữ thông tin về vùng, miền.

- CongTyVanChuyen(**MaCTyVC**, TenCTyVC, DienThoai)

Bảng lưu trữ thông tin các công ty giao hàng.

- CongTyCungCap(**MaCTyCC**, TenCTyCC, TenNguoiLH, DanhNghiaLH, DiaChi, ThanhPho, Vung, MaBuuDien, QuocGia, DienThoai, Fax, TrangChu)

Bảng lưu trữ thông tin về các nhà cung cấp bao gồm mã, tên công ty, tên người liên hệ, danh nghĩa liên hệ, địa chỉ, thành phố, vùng, mã bưu điện, quốc gia, số điện thoại, số fax và trang web của công ty.

Yêu cầu:

1. Tạo người dùng Northwind, thực hiện import các bảng trên vào schema này.
2. Quan sát sự thông thương giữa các bảng, thiết lập các ràng buộc toàn vẹn về khóa ngoại và khóa chính trong các bảng.
3. Sử dụng truy vấn SQL, trả lời các câu hỏi sau:
 - a. Liệt kê thông tin của tất cả khách hàng bao gồm: mã khách hàng, tên, địa chỉ, thành phố đã từng đặt mua các sản phẩm thuộc loại *Beverages*. Sắp xếp theo thứ tự tăng dần của tên khách hàng.
 - b. Hãy cho biết tên sản phẩm, tên loại, và tên các nhà cung cấp cho các sản phẩm được đặt mua nhiều nhất.
 - c. Tìm những khách hàng có đặt mua sản phẩm thuộc cả hai loại *seafood* và *beverages*.
 - d. Tính tổng giá trị bán ra của tất cả các mặt hàng theo từng tháng của từng vùng.
 - e. Với từng khách hàng, hãy cho biết tên và tổng số lượng mua của sản phẩm mà khách hàng đó đã đặt mua nhiều nhất.
4. Sử dụng PL/SQL, sinh viên thực hiện các yêu cầu sau:
 - a. Viết một thủ tục nhận tham số đầu vào là một quốc gia, hiển thị ra màn hình tên, địa chỉ của các khách hàng và số lần đặt hàng của các khách hàng thuộc quốc gia đó.

- b. Tạo một bảng `DaGoi1997`(`MaDonDH`, `MaKH`, `MaNV`, `NgayDH`, `NgayPhaiGiao`, `NgayGoiHang`), thiết lập các ràng buộc về khóa, miền giá trị,... cần thiết. Viết một thủ tục thực hiện xen các đơn đặt hàng được đặt trong năm 1997 vào bảng này.
 - c. Viết một hàm nhận tham số đầu vào là tên một nhân viên, hàm trả về số đơn đặt hàng mà nhân viên này đã tiếp nhận. Viết câu lệnh SQL liệt kê tên của tất cả các nhân viên và số đơn đặt hàng đã tiếp nhận tương ứng (sử dụng hàm vừa tạo).
 - d. Viết một trigger kiểm tra việc xen một dòng dữ liệu vào trong bảng `ChiTietDatHang`. Dòng dữ liệu chỉ được xen vào khi số lượng hàng đặt nhỏ hơn số lượng hàng có trong kho và hàng đó còn được sản xuất, ngược lại xuất một thông báo lỗi.
 - e. Tạo một thủ tục nhận tham số đầu vào là tên của một khách hàng, hiển thị ra màn hình tất cả thông tin về các lần đặt hàng và chi tiết hàng hóa của các lần đặt đó.
5. Giả sử hệ thống gồm các nhóm người dùng sau:
- Khách hàng có quyền:
 - SELECT trên `LoaiHang` và `HangHoa`
 - UPDATE trên `KhachHang`
 - Kế toán có quyền:
 - SELECT trên `DonDatHang`, `ChiTietDatHang` và `DaGoi1997`
 - EXECUTE trên thủ tục ở câu 4a và 4b.
 - Ban giám đốc có toàn quyền trên tất cả các đối tượng trong schema Northwind
- Viết các lệnh tạo các role và gán quyền tương ứng. Tạo một vài user để kiểm tra các role này.

Phụ lục

1 Các lỗi thường gặp

1.1 Lỗi chưa cấp quota cho người dùng

Thông báo lỗi: ORA-01950: no privileges on tablespace 'USERS'

Hướng giải quyết: cấp quota cho người dùng bằng lệnh ALTER USER...

1.2 Lỗi không đủ quyền

Thông báo lỗi: ORA-01031: insufficient privileges

Hướng giải quyết: liên hệ nhà quản trị CSDL để cấp quyền tương ứng cho người dùng

1.3 Lỗi chưa tạo thư mục trên hệ thống cho thư mục đối tượng

Thông báo lỗi: ORA-39002: invalid operation

ORA-39070: Unable to open the log file.

ORA-29283: invalid file operation

ORA-06512: at "SYS.UTL_FILE", line 475

ORA-29283: invalid file operation

Hướng giải quyết: tạo thư mục tương ứng trong lệnh `create directory...`

1.4 Lỗi không đủ quyền trên thư mục đối tượng

Thông báo lỗi: ORA-39002: invalid operation

ORA-39070: Unable to open the log file.

ORA-39087: directory name HR is invalid

Hướng giải quyết: gán quyền read và write cho người dùng bằng lệnh `grant`

1.5 Lỗi chưa tạo hoặc gọi sai tên của net service name

Thông báo lỗi: UDI-00008: operation generated ORACLE error 12154

ORA-12154: TNS:could not resolve the connect identifier specified

Hướng giải quyết: kiểm tra lại net service name trong **net manager**

2 Các hàm xử lý dữ liệu

2.1 Các hàm xử lý chuỗi:

- `LENGTH(<chuỗi>)`: Trả về độ dài chuỗi.
- `INSTR(<chuỗi a>, <chuỗi con b>, <vị trí x>[, <i>])`: Trả về vị trí xuất hiện lần thứ i của chuỗi con b trong chuỗi a, bắt đầu tìm từ vị trí x. Nếu $x < 0$ thì tìm từ phải sang trái.

Ví dụ: `SELECT INSTR('CORPORATE FLOOR', 'OR', 3, 2) FROM DUAL;`
(→14)

`SELECT INSTR('CORPORATE FLOOR', 'OR', -3, 2) FROM DUAL;` (→2)

Chú ý: DUAL là bảng giả (dummy table) trong Oracle, bảng này chỉ có một trường và được sử dụng khi câu truy vấn không cần thiết tham chiếu đến một bảng thực trong cơ sở dữ liệu.

- `SUBSTR(<chuỗi a>, <vị trí bắt đầu x>[, <số ký tự y>])`: Lấy y ký tự bắt đầu từ vị trí x của chuỗi a. Nếu $x < 0$ thì tìm từ phải sang trái. Nếu không có y sẽ lấy đến cuối chuỗi a.

Ví dụ:

`SELECT SUBSTR('ABCDEFGH', 3, 4) "Substring" FROM DUAL;` (→CDEF)

`SELECT SUBSTR('ABCDEFGH', -5, 4) "Substring" FROM DUAL;`
(→CDEF)

- `CONCAT(<chuỗi 1>, <chuỗi 2>)` hay phép toán `<chuỗi 1>||<chuỗi 2>`: ghép chuỗi.
- `LOWER(<chuỗi>)` (hay `UPPER`): Trả về chuỗi ở dạng chữ thường (hay chữ hoa) tương ứng.
- `LTRIM (<chuỗi a>, <chuỗi b>)` (hay `RTRIM`, `TRIM`): Cắt khỏi chuỗi a từ bên trái (hay từ bên phải, hay cả hai bên) những ký tự có trong chuỗi b.

Ví dụ:

`SELECT LTRIM('xyxXxyLAST WORD', 'xy') FROM DUAL;` (→ XxyLAST WORD)

2.2 Các hàm xử lý ngày tháng

- `EXTRACT(YEAR | MONTH | DAY FROM <ngày>)`: Trả về thành phần, ngày, tháng, hoặc năm của một dữ liệu kiểu date.

Ví dụ: `SELECT EXTRACT(YEAR FROM DATE '1998-03-07') FROM DUAL;`
(→1998)

- `ADD_MONTHS(<ngày x>, <số tháng n>)`: Trả về ngày mới sau khi cộng n tháng vào ngày x.

Ví dụ:

`SELECT add_months(sysdate, 3) FROM DUAL;` (→ 20-Apr-2008 17:19:12)

- `MONTHS_BETWEEN(<ngày 1>, <ngày 2>)`: Số tháng giữa 2 ngày.
- `SYSDATE`: Trả về ngày tháng hiện tại.

2.3 Các hàm chuyển kiểu

- `TO_CHAR(<số>)`: Chuyển số về dạng chuỗi
- `TO_CHAR(<ngày>, <chuỗi định dạng >)`: Chuyển ngày về dạng chuỗi

Ví dụ: `SELECT TO_CHAR (SYSDATE, 'MM-DD-YYYY HH24:MI:SS') FROM DUAL;`

(→ 01-20-2008 17:29:05)

- `TO_DATE(<chuỗi dạng ngày tháng>,<định dạng của chuỗi>)`: Chuyển chuỗi về ngày

Ví dụ: `SELECT MONTHS_BETWEEN
(TO_DATE('02-02-1995','MM-DD-YYYY'),
TO_DATE('01-01-1995','MM-DD-YYYY')) FROM DUAL; (→ 1,032)`

Tài liệu tham khảo

Tham khảo cho chương 1 và 2:

Thomas Connolly, Carolyn Begg; *Database Systems – A practical approach to design, implementation, and Management*; Addison Wesley, 4th edition 2005.

Tham khảo cho chương 3:

Christopher Allen; *Oracle Database 10g PL/SQL 101*; Oracle Press 2004

Gavin Powel, Carol McCulloch-Dieter; *Oracle SQL JumpStart with Example*; Elsevier 2005.

Tham khảo về hàm, thủ tục hệ thống của Oracle:

<http://www.techonthenet.com/oracle/functions/index.php>

Cách sử dụng biểu thức chính quy trong Oracle:

http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14251/adfns_regex_p.htm

Cách sử dụng ngoại lệ trong PL/SQL:

<http://plsql-tutorial.com/plsql-exception-handling.htm>

Tham khảo về quản lý giao dịch trong Oracle:

<http://www.tutorialized.com/view/tutorial/>