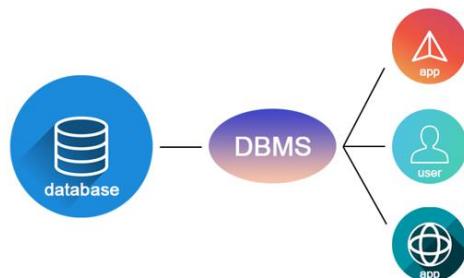


**TRƯỜNG ĐẠI HỌC KỸ THUẬT - CÔNG NGHỆ CẦN THƠ**  
**ThS NGUYỄN VĂN CƯỜNG, ThS NGUYỄN TRUNG KIÊN**



**GIÁO TRÌNH**  
**HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU**



NĂM – 2023

## LỜI NÓI ĐẦU

Ngay từ những ngày đầu ứng dụng của máy tính thì việc quản lý dữ liệu là một trong những nhu cầu rất quan trọng. Nhiều mô hình dữ liệu đã được nghiên cứu như mô hình phân cấp, mô hình mạng,... nhưng đến khi mô hình quan hệ được E. F. Codd đề xuất vào năm 1970 thì cơ sở dữ liệu đã được sang trang mới. Cùng với sự chuẩn hóa của ngôn ngữ truy vấn có cấu trúc (Structured Query Language - SQL) đã làm cho các hệ quản trị cơ sở dữ liệu sử dụng mô hình quan hệ như được tiếp thêm sức mạnh trong ứng dụng. Kết quả là rất nhiều hệ quản trị cơ sở dữ liệu mô hình quan hệ đã ra đời, được ứng dụng rộng rãi như FoxBase, FoxPro, DB2, MySQL, PostgreSQL, Microsoft Access, Microsoft SQL Server, Oracle,... Hiện nay, nhằm đáp ứng nhu cầu lưu trữ và xử lý dữ liệu lớn (Big Data), mô hình NoSQL với hàng loạt hệ quản trị cơ sở dữ liệu đã được xây dựng. Tuy nhiên, các hệ quản trị cơ sở dữ liệu mô hình quan hệ vẫn còn nguyên ý nghĩa trong ứng dụng thực tế.

Hệ quản trị cơ sở dữ liệu là học phần bắt buộc trong chương trình đào tạo các chuyên ngành Công nghệ thông tin và một số ngành khác có liên quan đến lưu trữ, xử lý dữ liệu. Giáo trình Hệ quản trị cơ sở dữ liệu với mục tiêu cung cấp những kiến thức cơ bản về hệ quản trị cơ sở dữ liệu mô hình quan hệ và minh họa cụ thể bằng phần mềm Microsoft SQL Server. Giáo trình được biên soạn hướng đến sinh viên đại học nên các nội dung được trình bày tương đối đơn giản, rõ ràng và nhiều ví dụ minh họa cụ thể.

Giáo trình Hệ quản trị cơ sở dữ liệu là kết quả của sự tổng hợp, chọn lọc các nội dung mà tập thể giảng viên Khoa Công nghệ thông tin, Trường Đại học Kỹ thuật – Công nghệ Cần Thơ đã giảng dạy trong nhiều năm qua. Nhóm biên soạn hy vọng giáo trình này sẽ là tài liệu cho sinh viên và những người tự học nắm rõ các khái niệm của hệ quản trị cơ sở dữ liệu, làm quen và thực hành trên một phần mềm quản trị cơ sở dữ liệu cụ thể.

Giáo trình được chia thành 5 chương như sau:

**Chương 1:** Tổng quan về Hệ quản trị Cơ sở dữ liệu

**Chương 2:** Các biện pháp bảo mật cơ sở dữ liệu

**Chương 3:** SQL và T-SQL

**Chương 4:** Quản lý giao dịch và phục hồi

**Chương 5:** Tối ưu hóa truy vấn

Nhóm biên soạn xin chân thành gửi lời cảm ơn đến Ban Giám hiệu, Lãnh đạo Khoa CNTT và tất cả các đồng nghiệp đã giúp đỡ, tạo điều kiện thuận lợi trong suốt quá trình làm việc.

**Nhóm biên soạn**

## MỤC LỤC

**LỜI NÓI ĐẦU .....** ..... I

**DANH MỤC BẢNG .....** ..... V

**DANH MỤC HÌNH.....** ..... VI

**DANH MỤC TỪ VIẾT TẮT, THUẬT NGỮ .....** ..... VIII

### CHƯƠNG 1

#### TỔNG QUAN VỀ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

1.1 CÁC KHÁI NIÊM CƠ BẢN .....	2
1.1.1 Cơ sở dữ liệu .....	2
1.1.2 Hệ quản trị cơ sở dữ liệu .....	4
1.1.3 Ứng dụng cơ sở dữ liệu .....	4
1.1.4 Các thành phần của môi trường DBMS.....	5
1.1.5 Vai trò con người trong môi trường DBMS .....	8
1.1.6 Ưu điểm và hạn chế của DBMS .....	10
1.2 CÁC CHỨC NĂNG, DỊCH VỤ CỦA DBMS .....	14
1.2.1 Lưu trữ, truy xuất và cập nhật dữ liệu .....	14
1.2.2 Danh mục người dùng có thể truy cập.....	14
1.2.3 Hỗ trợ giao dịch .....	15
1.2.4 Dịch vụ điều khiển đồng thời .....	16
1.2.5 Dịch vụ sao lưu.....	16
1.2.6 Dịch vụ ủy quyền.....	17
1.2.7 Hỗ trợ giao tiếp dữ liệu.....	17
1.2.8 Dịch vụ toàn vẹn.....	17
1.2.9 Dịch vụ thúc đẩy tính độc lập dữ liệu.....	18
1.2.10 Các dịch vụ tiện ích .....	18
1.3 CÁC THÀNH PHẦN CỦA MỘT DBMS.....	18
1.4 KIẾN TRÚC BA MỨC ANSI-SPARC .....	21
1.4.1 Mức bên ngoài .....	23
1.4.2 Mức khái niệm.....	23
1.4.3 Mức bên trong .....	24
1.4.4 Lược đồ, Ánh xạ và Thẻ hiện .....	24
1.4.5 Độc lập dữ liệu .....	26
1.5 KIẾN TRÚC DBMS ĐA NGƯỜI DÙNG .....	27
1.5.1 Xử lý từ xa .....	27
1.5.2 Kiến trúc máy chủ tập tin .....	28
1.5.3 Kiến trúc client-server hai tầng truyền thống .....	29
1.5.4 Kiến trúc client-server ba tầng .....	32
1.5.5 Kiến trúc client-server n tầng .....	33
TÓM TẮT.....	35
CÂU HỎI .....	37

### CHƯƠNG 2

#### CÁC BIỆN PHÁP BẢO MẬT CƠ SỞ DỮ LIỆU

2.1 BẢO MẬT CƠ SỞ DỮ LIỆU VÀ CÁC MÔI ĐE DỌA.....	39
2.2 CÁC BIỆN PHÁP BẢO MẬT .....	42
2.2.1 Ủy quyền .....	43
2.2.2 Điều khiển truy cập .....	44
2.2.3 Khung nhìn .....	45
2.2.4 Sao lưu và Phục hồi .....	46
2.2.5 Tính toàn vẹn .....	47
2.2.6 Mã hóa .....	47
2.2.7 Công nghệ RAID .....	47

## Giáo trình Hệ quản trị Cơ sở dữ liệu

---

2.3 SAO LUU VÀ PHỤC HỒI CƠ SỞ DỮ LIỆU .....	48
2.3.1 Cơ chế sao lưu .....	49
2.3.2 Tập tin nhật ký .....	49
2.3.3 Tạo điểm kiểm tra (Checkpointing).....	51
TÓM TẮT.....	52
CÂU HỎI .....	53

## CHƯƠNG 3

### SQL VÀ T-SQL

3.1 SQL .....	55
3.1.1 Đôi tượng của SQL.....	55
3.1.2 Tâm quan trọng của SQL.....	56
3.1.3 Cách viết câu lệnh SQL .....	56
3.2 CHỈ MỤC .....	57
3.3 KHUNG NHÌN (VIEW).....	58
3.3.1 Tạo khung nhìn (CREATE VIEW) .....	59
3.3.2 Xóa khung nhìn (DROP VIEW).....	62
3.3.3 Phân giải khung nhìn .....	63
3.3.4 Các hạn chế trên khung nhìn .....	64
3.3.5 Khả năng cập nhật khung nhìn .....	64
3.3.6 Tùy chọn WITH CHECK OPTION .....	66
3.3.7 Ưu điểm và hạn chế của khung nhìn .....	67
3.3.8 Cụ thể hóa khung nhìn.....	69
3.4 ĐIỀU KHIÊN TRUY CẬP .....	70
3.4.1 Các khái niệm .....	71
3.4.2 Cấp quyền cho người dùng (GRANT) .....	72
3.4.3 Thu hồi quyền của người dùng (REVOKE) .....	73
3.5 NGÔN NGỮ LẬP TRÌNH SQL .....	75
3.5.1 Biến – Lô – Lệnh GO .....	75
3.5.2 Con trỏ .....	80
3.5.3 Bảng tạm – Biến bảng – Kiểu bảng.....	83
3.5.4 SQL động – Lệnh EXEC – Thủ tục sp_executesql .....	89
3.5.5 Các cấu trúc điều khiển – Xử lý ngoại lệ.....	91
3.6 CHƯƠNG TRÌNH CON .....	98
3.6.1 Hàm người dùng định nghĩa .....	99
3.6.2 Thủ tục lưu trữ.....	101
3.7 TRÌNH KÍCH HOẠT .....	106
3.7.1 Trình kích hoạt DML .....	107
3.7.2 Trình kích hoạt DDL .....	111
TÓM TẮT.....	114
CÂU HỎI .....	116
BÀI TẬP .....	117

## CHƯƠNG 4

### QUẢN LÝ GIAO DỊCH VÀ PHỤC HỒI

4.1 GIAO DỊCH .....	121
4.1.1 Các thuộc tính của giao dịch.....	123
4.1.2 Kiến trúc cơ sở dữ liệu .....	123
4.2 PHỤC HỒI GIAO DỊCH .....	124
4.2.1 Quản lý bộ đệm .....	126
4.2.2 Các kỹ thuật phục hồi .....	127
4.3 ĐIỀU KHIÊN ĐỒNG THỜI .....	130
4.3.1 Nhu cầu điều khiển đồng thời.....	130
4.3.2 Khả năng tuần tự và khả năng khôi phục .....	133
4.3.3 Các kỹ thuật điều khiển đồng thời.....	140
TÓM TẮT.....	150

## CHƯƠNG 5

### TỐI UU HÓA TRUY VÂN

5.1 TỔNG QUAN VỀ XỬ LÝ TRUY VÂN .....	153
5.2 PHÂN RÃ TRUY VÂN .....	156
5.2.1 Phân tích .....	156
5.2.2 Chuẩn hóa .....	157
5.2.3 Phân tích ngữ nghĩa .....	157
5.2.4 Đơn giản hóa .....	159
5.2.5 Tái cấu trúc truy vấn .....	160
5.3 TIẾP CẬN HEURISTIC ĐỂ TỐI UU HÓA TRUY VÂN.....	161
5.3.1 Các quy tắc biến đổi cho các phép toán đại số quan hệ.....	161
5.3.2 Chiến lược xử lý theo phương pháp Heuristic.....	165
5.4 ƯỚC TÍNH CHI PHÍ CÁC PHÉP TOÁN ĐẠI SỐ QUAN HỆ.....	166
5.4.1 Thống kê cơ sở dữ liệu .....	166
5.4.2 Phép Chọn ( $S = \sigma_p(R)$ ).....	167
5.4.3 Phép Nối ( $T = (R \bowtie_F S)$ ).....	168
5.4.4 Phép Chiếu ( $S = \pi_{A1, A2, \dots, Am}(R)$ ).....	169
5.4.5 Các phép toán tập hợp .....	169
TÓM TẮT.....	170
CÂU HỎI .....	171

## PHỤ LỤC

### CÁC CƠ SỞ DỮ LIỆU MẪU

A. CƠ SỞ DỮ LIỆU DREAMHOME .....	172
A.1 Mô tả dữ liệu.....	172
A.2 Giao dịch dữ liệu .....	178
B. CƠ SỞ DỮ LIỆU UNIVERSITY ACCOMMODATION OFFICE .....	180
B.1 Mô tả dữ liệu.....	180
B.2 Giao dịch dữ liệu.....	182
C. CƠ SỞ DỮ LIỆU EASYDRIVE SCHOOL OF MOTORING .....	183
C.1 Mô tả dữ liệu.....	183
C.2 Giao dịch dữ liệu.....	184
TÀI LIỆU THAM KHẢO .....	185

## **DANH MỤC BẢNG**

Tên bảng	Trang
Bảng 1.1 Các ưu điểm của DBMS	10
Bảng 1.2 Các hạn chế của DBMS	13
Bảng 1.3 Tóm tắt các chức năng của client-server	32
Bảng 2.1 Ví dụ về các mối đe dọa	41
Bảng 5.1 Tóm tắt chi phí I/O ước tính của các chiến lược cho phép Chọn	168
Bảng 5.2 Tóm tắt chi phí I/O ước tính của các chiến lược cho phép Nối	168

## DANH MỤC HÌNH

<b>Tên hình</b>	<b>Trang</b>
Hình 1.1 Ví dụ về sơ đồ Quan hệ-Thực thể	3
Hình 1.2 Xử lý cơ sở dữ liệu	5
Hình 1.3 Các thành phần của môi trường DBMS	5
Hình 1.4 Cấu hình phần cứng của ví dụ DreamHome	6
Hình 1.5 Ván đế cập nhật bị mất	16
Hình 1.6 Các thành phần chính của DBMS	19
Hình 1.7 Các thành phần của trình quản lý cơ sở dữ liệu (DM)	21
Hình 1.8 Kiến trúc ba mức ANSI-SPARC	23
Hình 1.9 Sự khác biệt giữa ba mức độ truy tượng	26
Hình 1.10 Độc lập dữ liệu và kiến trúc ba mức ANSI-SPARC	27
Hình 1.11 Kiến trúc xử lý từ xa	28
Hình 1.12 Kiến trúc máy chủ tập tin	29
Hình 1.13 Kiến trúc client-server	30
Hình 1.14 Các hình thái client-server khác	30
Hình 1.15 Kiến trúc client-server hai tầng truyền thông	31
Hình 1.16 Kiến trúc client-server ba tầng	33
Hình 1.17 Kiến trúc bốn tầng	34
Hình 2.1 Tóm tắt các mối đe dọa tiềm ẩn đối với hệ thống máy tính	43
Hình 2.2 Môi trường máy tính đa người dùng tiêu biểu	44
Hình 2.3 Ví dụ một đoạn của tập tin nhật ký	51
Hình 3.1 Ảnh hưởng của REVOKE	74
Hình 4.1 Ví dụ về giao dịch	120
Hình 4.2 Sơ đồ trạng thái của một giao dịch	121
Hình 4.3 Hệ thống xử lý giao dịch của DBMS	123
Hình 4.4 Ví dụ UNDO/REDO	125
Hình 4.5 Ván đế cập nhật bị mất	130
Hình 4.6 Ván đế phụ thuộc không được cam kết	130
Hình 4.7 Ván đế phân tích không nhất quán	131
Hình 4.8 Các lịch trình tương đương	134
Hình 4.9 Hai giao dịch cập nhật đồng thời không khả tuần tự xung đột	135
Hình 4.10 Đồ thị ưu tiên cho Hình 4.9 hiển thị một chu trình	135
Hình 4.11 Lịch trình khả tuần tự khung nhìn nhưng không khả tuần tự xung đột	136

<b>Tên hình</b>	<b>Trang</b>
Hình 4.12 Đồ thị ưu tiên cho lịch trình khả tuần tự khung nhìn ở Hình 4.11	136
Hình 4.13 Đồ thị ưu tiên gắn nhãn cho lịch trình khả tuần tự khung nhìn ở Hình 4.11	138
Hình 4.14 Phiên bản sửa đổi lịch trình trong Hình 4.11 có thêm thao tác đọc	138
Hình 4.15 Đồ thị ưu tiên gắn nhãn cho lịch trình khả tuần tự khung nhìn ở Hình 4.14	139
Hình 4.16 Ngăn chặn sự cố cập nhật bị mất	142
Hình 4.17 Ngăn chặn vấn đề phụ thuộc không được cam kết	142
Hình 4.18 Ngăn chặn vấn đề phân tích không nhất quán	143
Hình 4.19 Deadlock giữa hai giao dịch	144
Hình 4.20 WFG với chu trình hiển thị deadlock giữa hai giao dịch	145
Hình 5.1 Các giai đoạn xử lý truy vấn	155
Hình 5.2 Ví dụ về cây đại số quan hệ	157
Hình 5.3 (a) Đồ thị kết nối quan hệ hiển thị truy vấn được xây dựng không chính xác; (b) Đồ thị kết nối thuộc tính chuẩn hóa hiển thị truy vấn là mâu thuẫn	159
Hình 5.4 Cây đại số quan hệ cho Ví dụ 10.3	165

## **DANH MỤC TỪ VIẾT TẮT, THUẬT NGỮ**

<b>Từ viết tắt, Thuật ngữ</b>	<b>Tiếng Việt</b>
American National Standards Institute - ANSI	Viện Tiêu chuẩn Quốc gia Hoa Kỳ
Application server	Máy chủ ứng dụng
Backus Naur Form - BNF	Ký pháp Backus Naur Form
Big data	Dữ liệu lớn
Buffer manager	Trình quản lý bộ đệm
Business rule	Quy tắc nghiệp vụ
Client-server	Máy chủ - máy khách
Conference on Data Systems Languages - CODASYL	Hội nghị Ngôn ngữ Hệ thống Dữ liệu
Data Administrator - DA	Quản trị viên dữ liệu
Database Administrator - DBA	Quản trị viên cơ sở dữ liệu
Data Base Task Group - DBTG	Nhóm đặc trách cơ sở dữ liệu
Data communication manager - DCM	Trình quản lý truyền thông dữ liệu
Data Definition Language - DDL	Ngôn ngữ Định nghĩa Dữ liệu
Data dictionary	Từ điển dữ liệu
Data independence	Độc lập dữ liệu
Data Manipulation Language - DML	Ngôn ngữ Thao tác Dữ liệu
Database - DB	Cơ sở dữ liệu
Database Administrator - DBA	Quản trị viên cơ sở dữ liệu
Database Application	Ứng dụng cơ sở dữ liệu
Database instance	Thể hiện cơ sở dữ liệu
Database integrity	Toàn vẹn cơ sở dữ liệu
Database Management System - DBMS	Hệ quản trị Cơ sở dữ liệu
Database manager - DM	Trình quản lý cơ sở dữ liệu
Database schema	Lược đồ cơ sở dữ liệu
Database server	Máy chủ cơ sở dữ liệu
Database System	Hệ cơ sở dữ liệu
DDL compiler	Trình biên dịch DDL
Discretionary Access Control - DAC	Điều khiển truy cập tùy quyền
DML preprocessor	Bộ tiền xử lý DML
Entity-Relationship - ER	Sơ đồ Quan hệ-Thực thể
Federal Information Processing Standard - FIPS	Tiêu chuẩn xử lý thông tin liên bang của Mỹ
File server	Máy chủ tập tin
Globally unique identifier - GUID	Mã định danh toàn cục duy nhất
Java 2 Platform, Enterprise Edition - J2EE	Nền tảng Java, phiên bản 2 - Phiên bản công nghiệp

Giáo trình Hệ quản trị Cơ sở dữ liệu

<b>Thuật ngữ, Từ viết tắt</b>	<b>Tiếng Việt</b>
Log file	Tập tin nhật ký
Mandatory Access Control - MAC	Điều khiển truy cập bắt buộc
Metadata	Siêu dữ liệu
OnLine Analytical Processing - OLAP	Xử lý Phân tích trực tuyến
Query optimizer	Trình tối ưu hóa truy vấn
Query processor	Bộ xử lý truy vấn
Recovery manager	Trình quản lý phục hồi
Redundant Array of Independent Disks - RAID	Mảng ổ đĩa độc lập dự phòng
Relational Database Management System - RDBMS	Hệ quản trị cơ sở dữ liệu quan hệ
Scheduler	Trình lập lịch
Standards Planning and Requirements Committee - SPARC	Ủy ban Kế hoạch và Nhu cầu
Stored procedure - SP	Thủ tục lưu trữ
Structured Query Language - SQL	Ngôn ngữ Truy vấn có cấu trúc
System catalog	Danh mục hệ thống
System catalog manager	Trình quản lý danh mục hệ thống
Systems Application Architecture - SAA	Kiến trúc Ứng dụng hệ thống
Table-valued parameter - TVP	Tham số giá trị bảng
Teleprocessing	Xử lý từ xa
Transaction	Giao dịch
Transaction manager	Trình quản lý giao dịch
User-defined function - UDF	Hàm do người dùng định nghĩa
View	Khung nhìn
Wide Area Network - WAN	Mạng diện rộng

## CHƯƠNG 1

### TỔNG QUAN VỀ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Trong chương này, chúng ta sẽ tìm hiểu:

- Tầm quan trọng của hệ cơ sở dữ liệu.
- Một số công dụng chung của hệ cơ sở dữ liệu.
- Ý nghĩa của các thuật ngữ Cơ sở dữ liệu (Database - DB), Hệ quản trị Cơ sở dữ liệu (Database Management System - DBMS).
- Các chức năng điển hình của DBMS.
- Các thành phần chính của DBMS.
- Vai trò con người trong DBMS.
- Ưu điểm và hạn chế của DBMS.

Những nghiên cứu về cơ sở dữ liệu đã có tác động sâu sắc đến nền kinh tế và xã hội, tạo ra một lĩnh vực công nghiệp cực kỳ có giá trị. Hệ cơ sở dữ liệu, được coi là sự phát triển quan trọng nhất trong lĩnh vực kỹ thuật phần mềm, và cơ sở dữ liệu hiện là khung cơ bản của hệ thống thông tin, làm thay đổi cơ bản cách thức hoạt động của nhiều tổ chức. Tầm quan trọng của hệ cơ sở dữ liệu đã tăng lên trong những năm gần đây với những phát triển đáng kể về khả năng phản ứng và thông tin truyền thông, bao gồm sự xuất hiện của Internet, thương mại điện tử, kinh doanh thông minh, truyền thông di động và điện toán đám mây.

Công nghệ cơ sở dữ liệu là một lĩnh vực thú vị và kề từ khi xuất hiện, đã là chất xúc tác cho nhiều phát triển quan trọng trong kỹ thuật phần mềm. Nghiên cứu về cơ sở dữ liệu vẫn chưa kết thúc và vẫn còn nhiều vấn đề cần được giải quyết. Hơn nữa, khi các ứng dụng của hệ cơ sở dữ liệu trở nên phức tạp hơn, chúng ta sẽ phải suy nghĩ lại về nhiều thuật toán hiện đang được sử dụng, chẳng hạn như thuật toán lưu trữ tập tin, truy cập tập tin và tối ưu hóa truy vấn. Các thuật toán nền tảng này đã có những đóng góp đáng kể trong kỹ thuật phần mềm và chắc chắn, sự phát triển của các thuật toán mới sẽ có những tác động tương tự. Trong chương đầu tiên này, chúng ta sẽ tìm hiểu về hệ cơ sở dữ liệu.

## 1.1 CÁC KHÁI NIỆM CƠ BẢN

Để bắt đầu thảo luận về cơ sở dữ liệu, trong phần này chúng ta sẽ tìm hiểu các khái niệm của phương pháp tiếp cận hướng cơ sở dữ liệu. Và trong phạm vi tài liệu này, chúng ta xem Cơ sở dữ liệu (Database - DB) là tập hợp các dữ liệu liên quan, Hệ quản trị Cơ sở dữ liệu (Database Management System - DBMS) là phần mềm quản lý và điều khiển quyền truy cập cơ sở dữ liệu. Ứng dụng cơ sở dữ liệu (Database Application) chỉ đơn giản là chương trình tương tác với cơ sở dữ liệu tại một thời điểm nào đó trong quá trình thực thi. Chúng ta cũng sử dụng thuật ngữ Hệ cơ sở dữ liệu (Database System) như một tập hợp các chương trình ứng dụng tương tác với cơ sở dữ liệu cùng với DBMS và chính cơ sở dữ liệu đó.

Các hệ thống sử dụng cách tiếp cận lưu trữ dựa trên tập tin bộc lộ rất nhiều hạn chế, nhưng hầu hết là do hai yếu tố sau:

1. Định nghĩa của dữ liệu được nhúng trong các chương trình ứng dụng, thay vì được lưu trữ riêng biệt và độc lập.
2. Không có quyền kiểm soát đối với việc truy cập và thao tác dữ liệu ngoài phạm vi được áp đặt bởi các chương trình ứng dụng.

Để việc lưu trữ và truy xuất dữ liệu trở nên hiệu quả hơn, cần phải có một cách tiếp cận mới, đó là hướng cơ sở dữ liệu.

### 1.1.1 Cơ sở dữ liệu

#### Cơ sở dữ liệu

Một tập hợp có thể chia sẻ được của các dữ liệu có liên quan logic và mô tả của nó, được thiết kế để đáp ứng nhu cầu thông tin của một tổ chức.

Cơ sở dữ liệu (database) là một kho dữ liệu duy nhất, có thể được sử dụng đồng thời bởi nhiều bộ phận và người dùng. Thay vì các tập tin không kết nối và dư thừa dữ liệu, tất cả các mục dữ liệu trong cơ sở dữ liệu được tích hợp với số lượng trùng lặp tối thiểu. Cơ sở dữ liệu không còn thuộc sở hữu riêng của bộ phận nào mà là tài nguyên chung của cả tổ chức. Cơ sở dữ liệu không chỉ lưu giữ dữ liệu hoạt động của tổ chức mà còn chứa mô tả về dữ liệu này. Vì lý do đó, cơ sở dữ liệu cũng được định nghĩa là một tập hợp các bản ghi tích hợp tự mô tả. Mô tả của dữ liệu được gọi là danh mục hệ thống (system catalog), hoặc từ điển dữ liệu (data dictionary) hoặc siêu dữ liệu (metadata) - “*dữ liệu về dữ liệu*”. Đây là tính chất tự mô tả của cơ sở dữ liệu, nền tảng của tính độc lập chương trình-dữ liệu.

Cách tiếp cận được thực hiện với các hệ cơ sở dữ liệu, trong đó định nghĩa dữ liệu được tách biệt khỏi các chương trình ứng dụng, tương tự như cách tiếp cận trong phát triển phần mềm hiện đại, cung cấp tách biệt định nghĩa bên trong và định nghĩa bên ngoài của một đối tượng. Người sử dụng đối tượng chỉ nhìn thấy định nghĩa bên ngoài mà không biết (hoặc không cần biết) đối tượng được định nghĩa ra sao và hoạt động như thế nào. Một ưu điểm của cách tiếp cận này, được gọi là trừu tượng hóa dữ liệu (data abstraction), là chúng ta có thể thay đổi định nghĩa bên trong của một đối tượng mà không ảnh hưởng đến người dùng đối tượng, miễn là định nghĩa bên ngoài vẫn giữ nguyên.

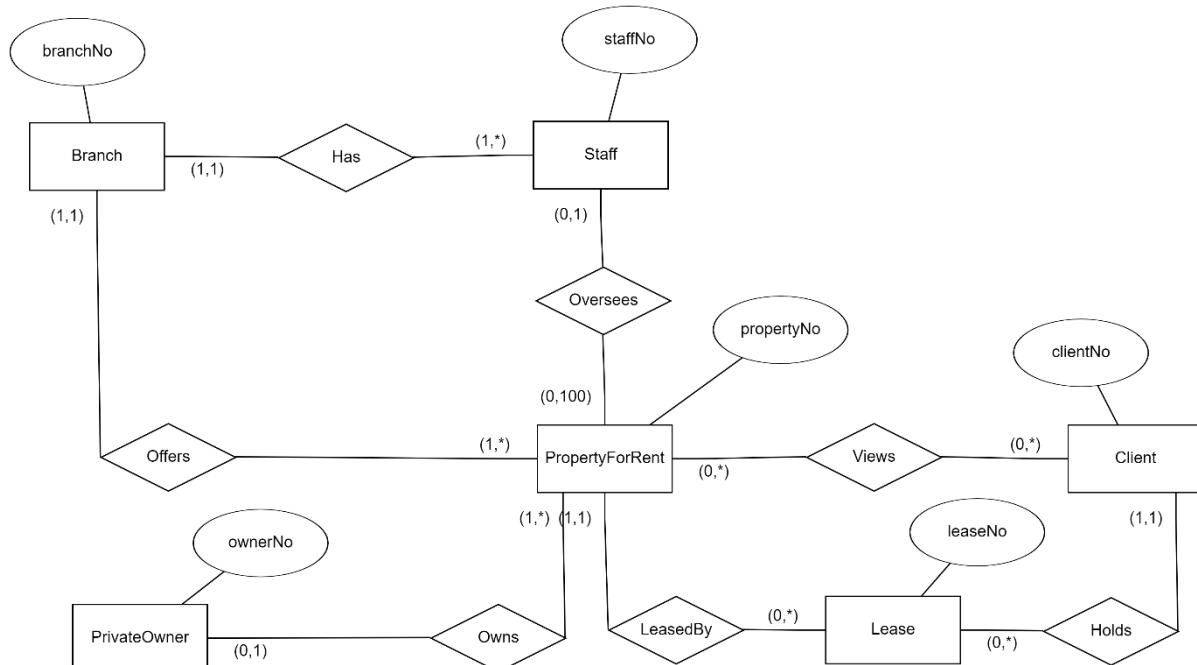
## Giáo trình Hệ quản trị Cơ sở dữ liệu

Theo cách tương tự, cách tiếp cận hướng cơ sở dữ liệu tách cấu trúc của dữ liệu khỏi các chương trình ứng dụng và lưu trữ trong cơ sở dữ liệu. Nếu cấu trúc dữ liệu mới được thêm vào hoặc cấu trúc hiện có được sửa đổi, thì các chương trình ứng dụng sẽ không bị ảnh hưởng, với điều kiện là chúng không phụ thuộc trực tiếp vào những gì đã được sửa đổi. Ví dụ, nếu chúng ta thêm trường mới vào bản ghi hoặc tạo tập tin mới, các ứng dụng hiện có sẽ không bị ảnh hưởng. Tuy nhiên, nếu chúng ta xóa một trường khỏi bản ghi mà chương trình ứng dụng sử dụng, thì chương trình ứng dụng đó sẽ bị ảnh hưởng bởi sự thay đổi này và phải được điều chỉnh cho phù hợp.

Một ý khác trong định nghĩa của cơ sở dữ liệu mà chúng ta nên giải thích là "*liên quan về mặt logic*". Khi phân tích nhu cầu thông tin của một tổ chức, chúng ta có gắng xác định các thực thể, thuộc tính và quan hệ. Thực thể (entity) là một đối tượng riêng biệt (người, địa điểm, sự vật, khái niệm hoặc sự kiện) trong tổ chức sẽ được biểu diễn trong cơ sở dữ liệu. Thuộc tính (attribute) là một đặc điểm mô tả một khía cạnh nào đó của đối tượng mà chúng ta muốn ghi lại và quan hệ (relationship) là sự liên kết giữa các thực thể. Hình 1.1 cho thấy một sơ đồ Quan hệ-Thực thể (Entity-Relationship - ER) cho một bộ phận của ví dụ *DreamHome* (xem phần Phụ lục A – Cơ sở dữ liệu mẫu DreamHome):

- Sáu thực thể (các hình chữ nhật): Branch, Staff, PropertyForRent, Client, PrivateOwner, và Lease;
- Bảy quan hệ (tên hình thoi): Has, Offers, Oversees, Views, Owns, LeasedBy, và Holds;
- Sáu thuộc tính cho sáu thực thể: branchNo, staffNo, propertyNo, clientNo, ownerNo, và leaseNo.

Cơ sở dữ liệu biểu diễn các thực thể, các thuộc tính và các mối quan hệ logic giữa các thực thể. Nói cách khác, cơ sở dữ liệu giữ cho dữ liệu có liên quan về mặt logic.



Hình 1.1 Ví dụ về sơ đồ Quan hệ-Thực thể

### 1.1.2 Hệ quản trị cơ sở dữ liệu

**Hệ quản trị  
Cơ sở dữ liệu**

Một hệ thống phần mềm cho phép người dùng định nghĩa, tạo, duy trì và kiểm soát quyền truy cập cơ sở dữ liệu.

Hệ quản trị cơ sở dữ liệu (DBMS) là phần mềm tương tác với các chương trình ứng dụng của người dùng và cơ sở dữ liệu. Thông thường, một DBMS cung cấp các phương tiện sau:

- Cho phép người dùng định nghĩa cơ sở dữ liệu, thường thông qua Ngôn ngữ Định nghĩa Dữ liệu (Data Definition Language - DDL). DDL cho phép người dùng chỉ định các kiểu dữ liệu, cấu trúc và các ràng buộc đối với dữ liệu sẽ được lưu trữ trong cơ sở dữ liệu.
- Cho phép người dùng chèn, cập nhật, xóa và truy xuất dữ liệu từ cơ sở dữ liệu thông qua Ngôn ngữ Thao tác Dữ liệu (Data Manipulation Language - DML). Việc có một kho lưu trữ trung tâm cho tất cả dữ liệu và các mô tả dữ liệu cho phép DML cung cấp một phương tiện truy vấn chung cho dữ liệu này, được gọi là ngôn ngữ truy vấn (Query Language). Việc cung cấp ngôn ngữ truy vấn làm giảm bớt các vấn đề với các hệ thống dựa trên tập tin, trong đó người dùng phải làm việc với một nhóm truy vấn cố định hoặc có sự gia tăng các chương trình, gây ra các vấn đề lớn về quản lý phần mềm. Ngôn ngữ truy vấn phổ biến nhất là Ngôn ngữ Truy vấn có cấu trúc (Structured Query Language – SQL).
- Cung cấp quyền truy cập có kiểm soát vào cơ sở dữ liệu. Ví dụ, có thể cung cấp:
  - Hệ thống bảo mật, ngăn người dùng không được ủy quyền truy cập cơ sở dữ liệu;
  - Hệ thống toàn vẹn, duy trì tính nhất quán của dữ liệu được lưu trữ;
  - Hệ thống điều khiển đồng thời, cho phép nhiều truy cập diễn ra đồng thời;
  - Hệ thống điều khiển khôi phục, khôi phục cơ sở dữ liệu về trạng thái nhất quán trước đó khi xảy ra sự cố phần cứng hoặc phần mềm;
  - Danh mục người dùng được phép truy cập, chứa các mô tả về dữ liệu trong cơ sở dữ liệu.

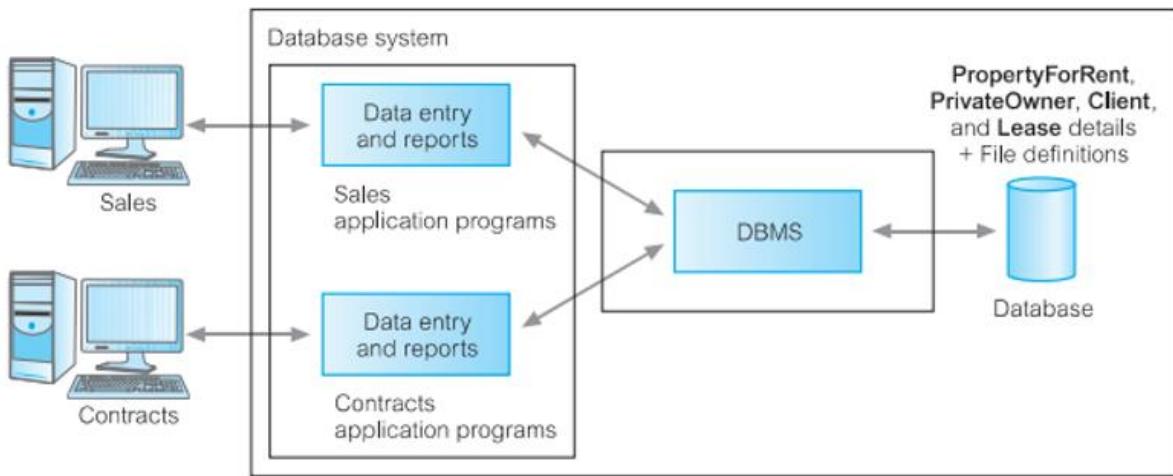
### 1.1.3 Ứng dụng cơ sở dữ liệu

**Ứng dụng  
cơ sở dữ liệu**

Một chương trình máy tính tương tác với cơ sở dữ liệu bằng cách chuyển các yêu cầu thích hợp (thường là một câu lệnh SQL) đến DBMS.

Người dùng tương tác với cơ sở dữ liệu thông qua một số chương trình để tạo và duy trì cơ sở dữ liệu cũng như tạo ra thông tin, được gọi là ứng dụng cơ sở dữ liệu (Database Application), có thể là các ứng dụng thông thường hoặc ngày nay thường là các ứng dụng trực tuyến.

Cách tiếp cận hướng cơ sở dữ liệu được minh họa trong Hình 1.2 cho thấy các Bộ phận Sales và Contracts truy cập cơ sở dữ liệu thông qua DBMS. Tuy nhiên, trái ngược với cách tiếp cận dựa trên tập tin, cấu trúc vật lý và việc lưu trữ dữ liệu trong mô hình này được quản lý bởi DBMS.



**PropertyForRent**(propertyNo, street, city, postcode, type, rooms, rent, ownerNo)

**PrivateOwner**(ownerNo, fName, lName, address, telNo)

**Client**(clientNo, fName, lName, address, telNo, prefType, maxRent)

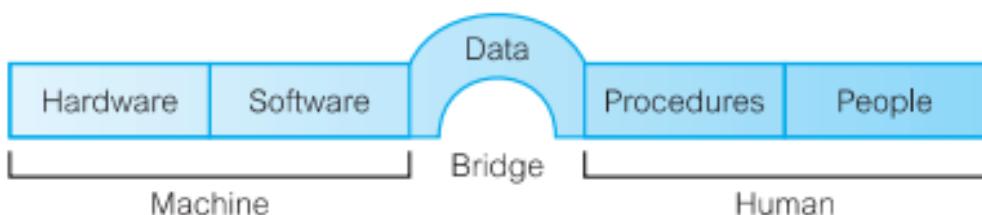
**Lease**(leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentFinish)

### Hình 1.2 Xử lý cơ sở dữ liệu

Các mô tả trên là tổng quát, do trong thực tế, những chức năng được cung cấp bởi các DBMS là khác nhau. Ví dụ, DBMS cho máy tính cá nhân có thể không hỗ trợ truy cập chia sẻ đồng thời và có thể chỉ cung cấp kiểm soát bảo mật, tính toàn vẹn và khôi phục có giới hạn. Ngược lại, các sản phẩm DBMS hiện đại, nhiều người dùng cung cấp tất cả các chức năng đã đề cập và có thể nhiều hơn. Các hệ thống này là những phần mềm cực kỳ phức tạp bao gồm hàng triệu dòng mã do phải cung cấp phần mềm xử lý các yêu cầu có tính chất tổng quát hơn. Hơn nữa, việc sử dụng DBMS ngày nay đòi hỏi một hệ thống cung cấp độ tin cậy gần như hoàn toàn và khả năng sẵn sàng 24/7, ngay cả khi có lỗi phần cứng hoặc phần mềm. DBMS liên tục phát triển và mở rộng để đáp ứng các yêu cầu mới của người dùng. Chẳng hạn, một số ứng dụng hiện nay yêu cầu lưu trữ hình ảnh đồ họa, video, âm thanh, dữ liệu phi cấu trúc, hoặc dữ liệu lớn (Big data)... Để tiếp cận thị trường này, DBMS phải thay đổi, các chức năng mới luôn được cập nhật, bổ sung.

#### 1.1.4 Các thành phần của môi trường DBMS

Chúng ta có thể xác định năm thành phần chính trong môi trường DBMS tiêu biểu: phần cứng, phần mềm, dữ liệu, thủ tục và con người.

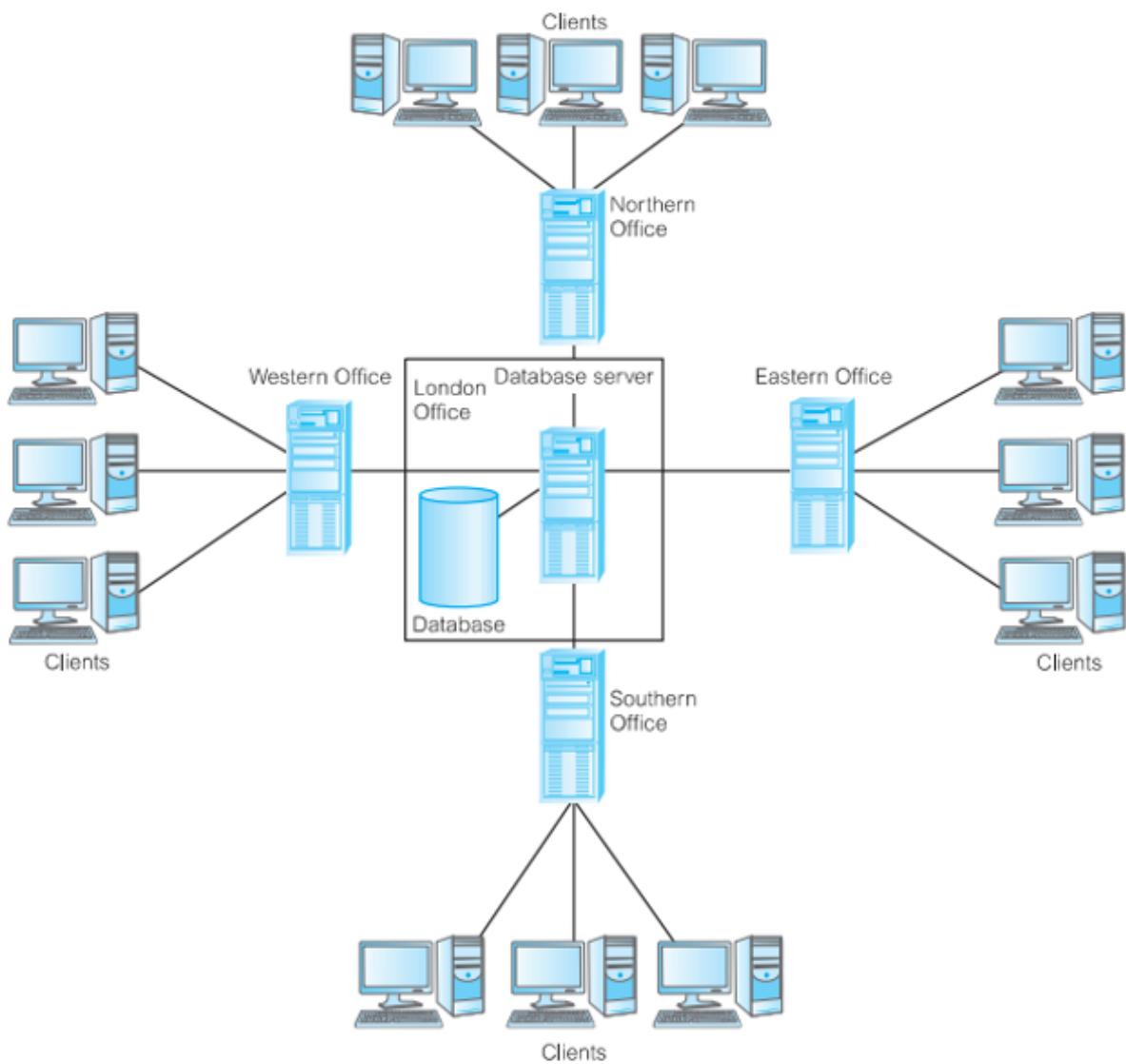


Hình 1.3 Các thành phần của môi trường DBMS

### 1.1.4.1 Phần cứng (Hardware)

Phần cứng cụ thể phụ thuộc vào yêu cầu của tổ chức và DBMS được sử dụng, có thể bao gồm từ một máy tính cá nhân đến một máy tính lớn hoặc một mạng máy tính. Một số DBMS chỉ chạy trên phần cứng hoặc hệ điều hành cụ thể, trong khi một số khác chạy trên nhiều loại phần cứng và các hệ điều hành khác nhau. DBMS yêu cầu một lượng bộ nhớ chính và không gian đĩa tối thiểu để chạy, nhưng cần lưu ý cấu hình tối thiểu này có thể không nhất thiết sẽ cung cấp hiệu suất chấp nhận được.

Cấu hình phần cứng đơn giản cho ví dụ minh họa *DreamHome* được thể hiện trong Hình 1.4, bao gồm một mạng các máy chủ nhỏ, với một máy chủ trung tâm chạy phần phụ trợ của DBMS, tức là một phần của DBMS quản lý và kiểm soát quyền truy cập vào cơ sở dữ liệu. Nó cũng hiển thị một số máy tính tại các vị trí khác nhau đang chạy giao diện người dùng của DBMS, tức là phần của DBMS giao tiếp với người dùng. Đây được gọi là kiến trúc client-server: backend là máy chủ và frontend là các máy khách (xem mục 1.5).



**Hình 1.4** Cấu hình phần cứng của ví dụ *DreamHome*

#### **1.1.4.2 Phần mềm (Software)**

Thành phần phần mềm bao gồm bản thân phần mềm DBMS và các chương trình ứng dụng, cùng với hệ điều hành, bao gồm cả phần mềm mạng nếu DBMS đang được sử dụng qua mạng. Thông thường, các chương trình ứng dụng được viết bằng ngôn ngữ lập trình thẻ hệ thứ ba (3GL), chẳng hạn như C, C++, C#, Java, Visual Basic, COBOL, Fortran, Ada hoặc Pascal hoặc ngôn ngữ thẻ hệ thứ tư (4GL), chẳng hạn như dưới dạng SQL, được nhúng trong ngôn ngữ thẻ hệ thứ ba. DBMS có thể có các công cụ thẻ hệ thứ tư của riêng nó cho phép phát triển nhanh chóng các ứng dụng thông qua việc cung cấp các ngôn ngữ truy vấn phi thủ tục, trình tạo báo cáo, trình tạo biểu mẫu, trình tạo đồ thị và trình tạo ứng dụng. Việc sử dụng các công cụ thẻ hệ thứ tư có thể cải thiện năng suất đáng kể và tạo ra các chương trình ứng dụng dễ bảo trì hơn.

#### **1.1.4.3 Dữ liệu (Data)**

Có lẽ thành phần quan trọng nhất của môi trường DBMS - theo quan điểm của người dùng cuối - là dữ liệu. Trong Hình 1.3, chúng ta quan sát thấy dữ liệu đóng vai trò như một cầu nối giữa các thành phần máy móc và thành phần con người. Cơ sở dữ liệu chứa cả dữ liệu và siêu dữ liệu (metadata), “*dữ liệu về dữ liệu*”. Cấu trúc của cơ sở dữ liệu được gọi là lược đồ cơ sở dữ liệu (database schema). Trong Hình 1.2, lược đồ bao gồm bốn bảng (table), đó là: PropertyForRent, PrivateOwner, Client và Lease. Bảng PropertyForRent có tám trường hoặc thuộc tính (attribute), đó là: propertyNo, street, city, zipCode, type (loại bất động sản), rooms (số lượng phòng), rent (tiền thuê hàng tháng) và ownerNo. Thuộc tính ownerNo mô hình hóa mối quan hệ giữa PropertyForRent và PrivateOwner: nghĩa là chủ sở hữu Sở hữu bất động sản cho thuê, như được mô tả trong sơ đồ ER ở Hình 1.1.

#### **1.1.4.4 Thủ tục (Procedures)**

Các thủ tục để cập nhật đến các hướng dẫn và quy tắc chi phối việc thiết kế và sử dụng cơ sở dữ liệu. Người sử dụng hệ thống và chuyên viên quản trị cơ sở dữ liệu yêu cầu các thủ tục dạng văn bản về cách sử dụng hoặc vận hành hệ thống, có thể bao gồm các hướng dẫn về cách:

- Đăng nhập vào DBMS.
- Sử dụng một chương trình ứng dụng hoặc một DBMS cụ thể.
- Khởi động và dừng DBMS.
- Tạo bản sao lưu của cơ sở dữ liệu.
- Xử lý các lỗi phần cứng hoặc phần mềm. Điều này có thể bao gồm các thủ tục về cách xác định thành phần bị lỗi, cách khắc phục thành phần bị lỗi và sau khi sửa chữa lỗi, cách khôi phục cơ sở dữ liệu.
- Thay đổi cấu trúc của bảng, tổ chức lại cơ sở dữ liệu trên nhiều đĩa, cải thiện hiệu suất hoặc lưu trữ dữ liệu vào bộ nhớ thứ cấp.

#### **1.1.4.5 Con người (People)**

Thành phần cuối cùng là những người liên quan đến hệ thống, sẽ được trình bày ở nội dung ngay sau đây.

#### **1.1.5 Vai trò con người trong môi trường DBMS**

Trong phần này, chúng ta tìm hiểu thành phần thứ năm của môi trường DBMS: con người. Có bốn vai trò con người khác nhau trong môi trường DBMS: quản trị viên dữ liệu và quản trị viên cơ sở dữ liệu, chuyên viên thiết kế cơ sở dữ liệu, nhà phát triển ứng dụng và người dùng cuối.

##### **1.1.5.1 Quản trị viên dữ liệu và quản trị viên cơ sở dữ liệu**

Cơ sở dữ liệu và DBMS là tài nguyên của tổ chức và phải được quản lý như bất kỳ tài nguyên nào khác. Quản trị dữ liệu và quản trị cơ sở dữ liệu là những vai trò thường được liên kết với việc quản lý và kiểm soát một DBMS và dữ liệu bên trong. Quản trị viên dữ liệu (Data Administrator - DA) chịu trách nhiệm quản lý tài nguyên dữ liệu, bao gồm lập kế hoạch cơ sở dữ liệu; phát triển và duy trì các tiêu chuẩn, chính sách và thủ tục; và thiết kế cơ sở dữ liệu khái niệm/logic. DA tham vấn và tư vấn cho các nhà quản lý cấp cao, đảm bảo rằng hướng phát triển cơ sở dữ liệu cuối cùng sẽ hỗ trợ các mục tiêu của tổ chức.

Quản trị viên cơ sở dữ liệu (Database Administrator - DBA) chịu trách nhiệm xây dựng cơ sở dữ liệu vật lý, bao gồm thiết kế và triển khai cơ sở dữ liệu vật lý, kiểm soát bảo mật và tính toàn vẹn, duy trì hệ thống hoạt động và đảm bảo hiệu suất thỏa đáng. Vai trò của DBA được định hướng về mặt kỹ thuật nhiều hơn so với DA, đòi hỏi kiến thức chi tiết về DBMS được sử dụng và môi trường hệ thống; Và có thể, trong một số tổ chức không có sự phân biệt giữa hai vai trò DA và DBA.

##### **1.1.5.2 Chuyên viên thiết kế cơ sở dữ liệu**

Trong các dự án thiết kế cơ sở dữ liệu lớn, chúng ta có thể phân biệt hai nhóm chuyên viên thiết kế: chuyên viên thiết kế cơ sở dữ liệu logic và chuyên viên thiết kế cơ sở dữ liệu vật lý.

Chuyên viên thiết kế cơ sở dữ liệu logic (logical database designer) quan tâm đến việc xác định dữ liệu (nghĩa là các thực thể và thuộc tính), các mối quan hệ giữa dữ liệu và các ràng buộc đối với dữ liệu sẽ được lưu trữ trong cơ sở dữ liệu. Chuyên viên thiết kế cơ sở dữ liệu logic phải có hiểu biết toàn diện và đầy đủ về dữ liệu của tổ chức và bất kỳ ràng buộc nào đối với dữ liệu này (các ràng buộc này đôi khi được gọi là các quy tắc nghiệp vụ - business rule). Những ràng buộc này mô tả các đặc điểm chính của dữ liệu được xem bởi người dùng cuối trong tổ chức. Ví dụ về các ràng buộc trong *DreamHome* là:

- Một nhân viên không thể quản lý hơn 100 bất động sản cho thuê cùng một lúc;
- Một nhân viên không thể xử lý việc cho thuê bất động sản của chính mình;
- Một luật sư không thể hỗ trợ pháp lý đồng thời cho cả người thuê và người cho thuê của cùng một bất động sản.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Để có hiệu quả, chuyên viên thiết kế cơ sở dữ liệu logic phải kết nối tất cả người dùng cơ sở dữ liệu tiềm năng trong quá trình phát triển mô hình dữ liệu và sự tham gia này phải bắt đầu càng sớm càng tốt. Trong tài liệu này, chúng ta tạm chia công việc của chuyên viên thiết kế cơ sở dữ liệu logic thành hai giai đoạn:

- Thiết kế cơ sở dữ liệu khái niệm, độc lập với các chi tiết triển khai, chẳng hạn như DBMS được sử dụng, chương trình ứng dụng, ngôn ngữ lập trình hoặc bất kỳ câu hỏi nào khác;
- Thiết kế cơ sở dữ liệu lôgic, mục tiêu hướng đến một mô hình dữ liệu cụ thể, chẳng hạn như mô hình quan hệ, mạng, phân cấp hoặc hướng đối tượng.

Chuyên viên thiết kế cơ sở dữ liệu vật lý (physical database designer) quyết định cách mà bản thiết kế cơ sở dữ liệu logic được hiện thực về mặt vật lý. Điều này liên quan đến:

- Ánh xạ thiết kế cơ sở dữ liệu logic thành một tập hợp các bảng và các ràng buộc toàn vẹn;
- Lựa chọn cấu trúc lưu trữ cụ thể và phương pháp truy cập dữ liệu để đạt hiệu suất tốt;
- Thiết kế bất kỳ biện pháp bảo mật nào được yêu cầu trên dữ liệu.

Nhiều phần của thiết kế cơ sở dữ liệu vật lý phụ thuộc nhiều vào DBMS được sử dụng và có thể có nhiều hơn một cách triển khai thiết kế. Do đó, chuyên viên thiết kế cơ sở dữ liệu vật lý phải nhận thức đầy đủ về chức năng của DBMS được sử dụng, hiểu những ưu điểm và nhược điểm của từng cách triển khai thay thế. Chuyên viên thiết kế cơ sở dữ liệu vật lý phải có khả năng chọn một chiến lược lưu trữ phù hợp với các tính năng có sẵn của DBMS được sử dụng. Thiết kế cơ sở dữ liệu khái niệm và logic liên quan đến cái gì (*what*), thiết kế cơ sở dữ liệu vật lý quan tâm đến cách thức thực hiện (*how*).

### 1.1.5.3 Nhà phát triển ứng dụng

Khi cơ sở dữ liệu đã được triển khai, các chương trình ứng dụng cung cấp chức năng cần thiết cho người dùng cuối phải được triển khai. Đây là trách nhiệm của các nhà phát triển ứng dụng (application developer). Thông thường, các nhà phát triển ứng dụng làm việc từ một đặc tả do các nhà phân tích hệ thống tạo ra. Mỗi chương trình chứa các câu lệnh yêu cầu DBMS thực hiện một số hoạt động trên cơ sở dữ liệu, bao gồm truy xuất dữ liệu, chèn, cập nhật và xóa dữ liệu. Các chương trình có thể được viết bằng ngôn ngữ lập trình thế hệ thứ ba hoặc thứ tư, như đã thảo luận ở phần trước.

### 1.1.5.4 Người dùng cuối

Người dùng cuối (end user) là “khách hàng” của cơ sở dữ liệu; cơ sở dữ liệu được thiết kế, triển khai và duy trì để phục vụ nhu cầu thông tin của nhóm đối tượng này. Người dùng cuối có thể được phân loại theo cách sử dụng hệ thống:

- Người dùng thông thường (naive user): truy cập cơ sở dữ liệu thông qua các chương trình ứng dụng với mục đích làm cho các hoạt động trở nên đơn giản nhất có thể.

Nhóm người dùng này gọi các hoạt động cơ sở dữ liệu bằng cách nhập các lệnh đơn giản hoặc chọn các tùy chọn từ trình đơn; nghĩa là họ không cần biết bất cứ điều gì về cơ sở dữ liệu hoặc DBMS. Ví dụ, nhân viên thu ngân tại siêu thị sử dụng thiết bị đọc mã vạch để tìm giá của mặt hàng nhờ vào một chương trình ứng dụng có chức năng đọc mã vạch, tra cứu giá của mặt hàng trong cơ sở dữ liệu và hiển thị giá trên màn hình thanh toán.

- Người dùng chuyên sâu (sophisticated user). Nhóm người dùng này quen thuộc với cấu trúc của cơ sở dữ liệu và các tiện ích được cung cấp bởi DBMS. Họ có thể sử dụng ngôn ngữ truy vấn cao như SQL để thực hiện các hoạt động cần thiết, thậm chí có thể tự xây dựng các chương trình ứng dụng đáp ứng cho nhu cầu cá nhân.

### 1.1.6 Ưu điểm và hạn chế của DBMS

Hệ quản trị cơ sở dữ liệu có những lợi thế tiềm năng đầy hứa hẹn, và dĩ nhiên, cũng có những hạn chế.

#### 1.1.6.1 Ưu điểm

Các ưu điểm của hệ quản trị cơ sở dữ liệu được liệt kê trong Bảng 1.1.

Bảng 1.1 Các ưu điểm của DBMS

Kiểm soát dư thừa dữ liệu	Lợi thế kinh tế nhờ quy mô
Tính nhất quán của dữ liệu	Cân bằng các yêu cầu xung đột
Nhiều thông tin hơn từ cùng một lượng dữ liệu	Khả năng truy cập và phản hồi dữ liệu được cải thiện
Chia sẻ dữ liệu	Tăng năng suất
Cải thiện tính toàn vẹn của dữ liệu	Cải thiện khả năng bảo trì
Cải thiện bảo mật	Tăng tính đồng thời
Thực thi các tiêu chuẩn	Dịch vụ sao lưu và dịch vụ phục hồi

- Kiểm soát dư thừa dữ liệu: các hệ thống dựa trên tập tin sẽ lãng phí dung lượng do phải lưu trữ cùng một thông tin trong nhiều tập tin. Ngược lại, cách tiếp cận cơ sở dữ liệu cố gắng loại bỏ sự dư thừa bằng cách tích hợp các tập tin để nhiều bản sao của cùng một dữ liệu không được lưu trữ. Tuy nhiên, cần lưu ý rằng, cách tiếp cận cơ sở dữ liệu không loại bỏ hoàn toàn sự dư thừa, mà kiểm soát lượng dư thừa vốn có trong cơ sở dữ liệu.
- Tính nhất quán của dữ liệu: Bằng cách loại bỏ hoặc kiểm soát sự dư thừa, chúng ta giảm nguy cơ xảy ra sự không nhất quán. Nếu một mục dữ liệu chỉ được lưu trữ một lần trong cơ sở dữ liệu, thì bất kỳ cập nhật nào đối với giá trị của nó chỉ phải được thực hiện một lần và giá trị mới có sẵn ngay lập tức cho tất cả người dùng. Nếu một mục dữ liệu được lưu trữ nhiều lần và hệ thống nhận thức được điều này, hệ thống có thể đảm bảo rằng tất cả các bản sao của mục đó được giữ nhất quán.

- Nhiều thông tin hơn từ cùng một lượng dữ liệu: Với việc tích hợp dữ liệu hoạt động, tổ chức có thể lấy thêm thông tin từ cùng một dữ liệu. Ví dụ, trong hệ thống dựa trên tập tin, bộ phận Contracts không biết ai sở hữu một căn hộ cho thuê. Tương tự, bộ phận Sales không có thông tin về chi tiết hợp đồng thuê. Khi chúng ta tích hợp các tập tin này, bộ phận Contracts có quyền truy cập vào thông tin chi tiết về chủ sở hữu và bộ phận Sales có quyền truy cập vào chi tiết hợp đồng cho thuê. Và đó là lý do vì sao chúng ta có thể có thêm thông tin từ cùng một lượng dữ liệu.
- Chia sẻ dữ liệu: Thông thường, các tập tin được sở hữu bởi những người hoặc bộ phận sử dụng chúng. Ngược lại, cơ sở dữ liệu thuộc về toàn bộ tổ chức và có thể được chia sẻ cho tất cả người dùng được ủy quyền. Hơn nữa, các ứng dụng mới có thể xây dựng dựa trên dữ liệu hiện có trong cơ sở dữ liệu và chỉ thêm dữ liệu hiện không được lưu trữ, thay vì phải xác định lại tất cả các yêu cầu dữ liệu. Các ứng dụng mới cũng có thể dựa vào các chức năng do DBMS cung cấp, chẳng hạn như định nghĩa và thao tác dữ liệu, điều khiển đồng thời và khôi phục, thay vì phải tự phát triển các chức năng này.
- Cải thiện tính toàn vẹn của dữ liệu: Tính toàn vẹn của cơ sở dữ liệu đề cập đến tính hợp lệ và tính nhất quán của dữ liệu được lưu trữ. Tính toàn vẹn thường được thể hiện dưới dạng các ràng buộc, là các quy tắc nhất quán mà cơ sở dữ liệu không được phép vi phạm. Các ràng buộc có thể áp dụng cho các mục dữ liệu trong một bản ghi đơn lẻ hoặc cho các mối quan hệ giữa các bản ghi. Ví dụ, một ràng buộc về tính toàn vẹn có thể quy định rằng lương của một nhân viên không được lớn hơn \$40.000 hoặc số chi nhánh trong hồ sơ nhân viên, biểu diễn cho chi nhánh nơi nhân viên đó làm việc, phải tương ứng với một văn phòng chi nhánh hiện có. Một lần nữa, tích hợp cho phép DBA xác định các ràng buộc toàn vẹn và DBMS sẽ thực thi các ràng buộc này.
- Cải thiện bảo mật: Bảo mật cơ sở dữ liệu là bảo vệ cơ sở dữ liệu khỏi những người dùng không được ủy quyền. Nếu không có các biện pháp bảo mật phù hợp, việc tích hợp làm cho dữ liệu dễ bị tấn công hơn so với các hệ thống dựa trên tập tin. Tuy nhiên, tích hợp cho phép DBA xác định các cơ chế bảo mật cơ sở dữ liệu và DBMS sẽ thực thi các cơ chế này.

Bảo mật có thể ở dạng tên người dùng và mật khẩu để xác định những người được phép sử dụng cơ sở dữ liệu. Quyền truy cập mà người dùng được ủy quyền được phép vào dữ liệu có thể bị hạn chế bởi loại hoạt động (truy xuất, chèn, cập nhật, xóa). Ví dụ, DBA có quyền truy cập vào tất cả dữ liệu trong cơ sở dữ liệu; giám đốc chi nhánh có thể có quyền truy cập vào tất cả dữ liệu liên quan đến văn phòng chi nhánh của mình; và một nhân viên kinh doanh có thể có quyền truy cập vào tất cả dữ liệu liên quan đến bất động sản cho thuê nhưng không có quyền truy cập vào dữ liệu nhạy cảm như chi tiết tiền lương của nhân viên.

- Thực thi các tiêu chuẩn: Một lần nữa, tích hợp cho phép DBA xác định và DBMS thực thi các tiêu chuẩn cần thiết. Những tiêu chuẩn này có thể bao gồm các tiêu chuẩn của bộ phận, tổ chức, quốc gia hoặc quốc tế về những vấn đề như định dạng dữ liệu để tạo điều kiện trao đổi dữ liệu giữa các hệ thống, quy ước đặt tên, tiêu chuẩn tài liệu, quy trình cập nhật và quy tắc truy cập.
- Lợi thế kinh tế nhờ quy mô: Kết hợp tất cả dữ liệu hoạt động của tổ chức vào một cơ sở dữ liệu và tạo một tập hợp các ứng dụng hoạt động trên một nguồn dữ liệu này có thể tiết kiệm chi phí. Trong trường hợp này, ngân sách thường được phân bổ cho mỗi bộ phận để phát triển và duy trì hệ thống dựa trên tập tin có thể được kết hợp lại, có thể dẫn đến tổng chi phí thấp hơn. Ngân sách kết hợp có thể được sử dụng để trang bị cấu hình hệ thống phù hợp hơn với nhu cầu của tổ chức. Điều này có thể bao gồm một máy tính lớn, mạnh mẽ hoặc một mạng các máy tính nhỏ hơn.
- Cân bằng các yêu cầu xung đột: Mỗi người dùng hoặc bộ phận có những nhu cầu có thể mâu thuẫn với nhu cầu của những người dùng, bộ phận khác. Vì cơ sở dữ liệu nằm dưới sự kiểm soát của DBA nên DBA có thể đưa ra quyết định về việc thiết kế và sử dụng cơ sở dữ liệu nhằm cung cấp việc sử dụng tài nguyên tốt nhất cho toàn bộ tổ chức. Những quyết định này sẽ mang lại hiệu quả tối ưu cho những ứng dụng quan trọng, và có thể không tối ưu cho những ứng dụng ít quan trọng hơn.
- Khả năng truy cập và phản hồi dữ liệu được cải thiện: Một lần nữa, do tích hợp, dữ liệu vượt qua ranh giới các bộ phận, cho phép người dùng cuối có thể truy cập trực tiếp. Điều này cung cấp một hệ thống có tiềm năng nhiều chức năng hơn, chẳng hạn như có thể được sử dụng để cung cấp các dịch vụ tốt hơn cho người dùng cuối hoặc khách hàng của tổ chức. Nhiều DBMS cung cấp các ngôn ngữ truy vấn hoặc trình tạo báo cáo cho phép người dùng đặt các câu hỏi đặc biệt và nhận được thông tin cần thiết gần như ngay lập tức tại thiết bị đầu cuối của họ mà không cần phải có những phần mềm để trích xuất thông tin này từ cơ sở dữ liệu. Ví dụ, một giám đốc chi nhánh có thể liệt kê tất cả các căn hộ (flat) có giá cho thuê hàng tháng lớn hơn \$400 bằng cách nhập câu lệnh SQL sau tại một thiết bị đầu cuối:

```
SELECT *  
FROM PropertyForRent  
WHERE type = 'Flat' AND rent > 400;
```

- Tăng năng suất: DBMS cung cấp nhiều chức năng tiêu chuẩn mà lập trình viên thường phải viết trong một ứng dụng dựa trên tập tin. Việc cung cấp các chức năng này cho phép lập trình viên tập trung vào các chức năng đặc thù được người dùng yêu cầu mà không cần phải quan tâm về các chi tiết triển khai ở mức thấp.

- Cải thiện khả năng bảo trì thông qua tính độc lập dữ liệu: Trong các hệ thống dựa trên tập tin, mô tả dữ liệu và logic để truy cập dữ liệu được tích hợp trong mỗi chương trình ứng dụng, làm cho các chương trình phụ thuộc vào dữ liệu. Một thay đổi đối với cấu trúc của dữ liệu - chẳng hạn, tạo địa chỉ 41 ký tự thay vì 40 ký tự hoặc thay đổi cách dữ liệu được lưu trữ trên đĩa - có thể yêu cầu các thay đổi đáng kể đối với các chương trình bị ảnh hưởng. Ngược lại, một DBMS tách các mô tả dữ liệu khỏi các ứng dụng, do đó làm cho các ứng dụng “miễn nhiệm” với các thay đổi trong mô tả dữ liệu (được gọi là độc lập dữ liệu - data independence). Việc cung cấp tính độc lập của dữ liệu giúp đơn giản hóa việc bảo trì ứng dụng cơ sở dữ liệu.
- Tăng tính đồng thời: Trong một số hệ thống dựa trên tập tin, nếu hai hoặc nhiều người dùng được phép truy cập đồng thời vào cùng một tập tin, có thể các quyền truy cập sẽ gây nhiễu lẫn nhau, dẫn đến mất thông tin hoặc thậm chí mất tính toàn vẹn. Đa số DBMS sẽ quản lý quyền truy cập cơ sở dữ liệu đồng thời và đảm bảo rằng các vấn đề như thế không thể xảy ra.
- Dịch vụ sao lưu và phục hồi được cải tiến: Các DBMS hiện đại cung cấp các phương tiện sao lưu và phục hồi hiệu quả giúp giảm thiểu khối lượng công việc cần phải thực hiện sau khi cơ sở dữ liệu phát sinh sự cố.

### 1.1.6.2 Hạn chế

Các hạn chế của phương pháp tiếp cận cơ sở dữ liệu được tóm tắt trong Bảng 1.2.

**Bảng 1.2** Các hạn chế của DBMS

Tính phức tạp	Chi phí chuyển đổi
Kích thước	Hiệu suất
Chi phí của DBMS	Tác động lớn hơn của sự cố
Chi phí phần cứng bổ sung	

- Tính phức tạp: Việc cung cấp đầy đủ các chức năng mà người dùng mong đợi ở một DBMS tốt khiến cho DBMS trở thành một phần mềm cực kỳ phức tạp. Các chuyên viên thiết kế và phát triển cơ sở dữ liệu, quản trị viên dữ liệu và cơ sở dữ liệu và người dùng cuối phải hiểu rõ các chức năng này để tận dụng tối đa chức năng của chúng. Việc không hiểu hệ thống có thể dẫn đến các quyết định thiết kế không tốt, và có thể gây ra hậu quả nghiêm trọng cho tổ chức.
- Kích thước: Độ phức tạp và độ bao quát về chức năng làm cho DBMS trở thành một phần mềm cực kỳ lớn, chiếm nhiều megabyte dung lượng ổ đĩa và yêu cầu một lượng lớn bộ nhớ để hoạt động hiệu quả.
- Chi phí của DBMS: Chi phí của DBMS thay đổi đáng kể, tùy thuộc vào môi trường và chức năng được cung cấp. Ví dụ, một DBMS một người dùng cho một máy tính cá nhân có thể chỉ có giá \$100. Tuy nhiên, một DBMS lớn, đa người dùng có thể cực kỳ đắt, có thể là \$100.000 hoặc thậm chí \$1.000.000.

- Chi phí phần cứng bổ sung: Các yêu cầu lưu trữ đĩa cho DBMS và cơ sở dữ liệu có thể đòi hỏi phải bổ sung dung lượng lưu trữ. Hơn nữa, để đạt được hiệu suất cần thiết, có thể cần phải trang bị một máy tính lớn hơn, thậm chí có thể là một máy tính chỉ dành riêng cho DBMS. Việc trang bị thêm phần cứng dẫn đến chi phí nhiều hơn.
- Chi phí chuyển đổi: Trong một số tình huống, chi phí của DBMS và phần cứng bổ sung có thể tương đối nhỏ so với chi phí chuyển đổi các ứng dụng hiện có sang chạy trên DBMS và phần cứng mới. Chi phí này cũng bao gồm chi phí đào tạo nhân viên sử dụng các hệ thống mới này và có thể thuê chuyên viên để giúp chuyển đổi và vận hành hệ thống. Chi phí này là một trong những lý do chính khiến một số tổ chức cảm thấy bị ràng buộc với hệ thống hiện tại của họ và không thể chuyển sang công nghệ cơ sở dữ liệu hiện đại hơn.
- Hiệu suất: Thông thường, một hệ thống dựa trên tập tin được viết cho một ứng dụng cụ thể, chẳng hạn như lập hóa đơn. Kết quả là, hiệu suất nói chung là rất tốt. Tuy nhiên, DBMS được viết để tổng quát hơn, để phục vụ cho nhiều ứng dụng thay vì chỉ một ứng dụng. Kết quả là một số ứng dụng có thể không đạt được hiệu suất như trước.
- Tác động lớn hơn của sự cố: Việc tập trung các nguồn lực làm tăng tính dễ bị tổn thương của hệ thống. Bởi vì tất cả người dùng và ứng dụng đều dựa vào tính khả dụng của DBMS, sự cố của một số thành phần nhất định trong hệ thống có thể khiến các hoạt động liên quan bị ảnh hưởng hoặc dừng lại.

## 1.2 CÁC CHỨC NĂNG, DỊCH VỤ CỦA DBMS

Trong phần này, chúng ta tìm hiểu các loại chức năng và dịch vụ mà một DBMS tiêu biểu được mong đợi sẽ cung cấp.

### 1.2.1 Lưu trữ, truy xuất và cập nhật dữ liệu

DBMS phải cung cấp cho người dùng khả năng lưu trữ, truy xuất và cập nhật dữ liệu trong cơ sở dữ liệu.

Đây là các chức năng cơ bản của DBMS. Từ những thảo luận trước, rõ ràng với việc cung cấp chức năng này, DBMS đã phải ẩn các chi tiết triển khai vật lý bên trong (chẳng hạn như tổ chức tập tin và cấu trúc lưu trữ) với người dùng.

### 1.2.2 Danh mục người dùng có thể truy cập

DBMS phải cung cấp một danh mục trong đó lưu trữ các mô tả về các mục dữ liệu và những gì mà người dùng có thể truy cập được.

Một tính năng chính của kiến trúc ANSI-SPARC (xem mục 1.4) là phải có danh mục hệ thống (system catalog) tích hợp để chứa dữ liệu về lược đồ, người dùng, ứng dụng,... Danh mục sẽ phải có thể truy cập được đối với người dùng cũng như DBMS.

Danh mục hệ thống, hay từ điển dữ liệu (data dictionary), là một kho thông tin mô tả dữ liệu trong cơ sở dữ liệu; là "dữ liệu về dữ liệu" hoặc siêu dữ liệu (metadata).

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Lượng thông tin và cách thông tin của danh mục hệ thống được sử dụng có thể thay đổi theo từng DBMS khác nhau. Thông thường, danh mục hệ thống sẽ lưu trữ:

- Tên, kiểu và kích thước của các mục dữ liệu;
- Tên của các mối quan hệ;
- Các ràng buộc về tính toàn vẹn đối với dữ liệu;
- Tên người dùng được ủy quyền có quyền truy cập vào dữ liệu;
- Các mục dữ liệu mà mỗi người dùng có thể truy cập và các loại quyền truy cập được phép; ví dụ như chèn, cập nhật, xóa, hoặc đọc;
- Các lược đồ bên ngoài, khái niệm, bên trong và các ánh xạ giữa các lược đồ;
- Thông kê sử dụng, chẳng hạn như tần suất của các giao dịch (transaction) và số lượng truy cập được thực hiện trên các đối tượng của cơ sở dữ liệu.

Danh mục hệ thống là một trong những thành phần nền tảng của DBMS. Nhiều thành phần phần mềm mà chúng ta mô tả trong phần tiếp theo sẽ dựa vào danh mục hệ thống để có thông tin. Một số lợi ích của danh mục hệ thống là:

- Thông tin về dữ liệu có thể được thu thập và lưu trữ tập trung, giúp duy trì quyền kiểm soát dữ liệu dưới dạng tài nguyên.
- Ý nghĩa của dữ liệu có thể được định nghĩa, giúp những người dùng hiểu được mục đích của dữ liệu.
- Giao tiếp được đơn giản hóa, bởi vì các ý nghĩa chính xác đã được lưu trữ. Danh mục hệ thống cũng có thể xác định người dùng hoặc những người dùng nào sở hữu hoặc truy cập dữ liệu.
- Có thể xác định dễ dàng hơn tính dư thừa và không nhất quán khi dữ liệu được tập trung.
- Các thay đổi đối với cơ sở dữ liệu có thể được ghi lại.
- Tác động của một thay đổi có thể được xác định trước khi được thực hiện vì danh mục hệ thống ghi lại từng mục dữ liệu, tất cả các mối quan hệ và tất cả người dùng có liên quan.
- Bảo mật có thể được thực thi.
- Tính toàn vẹn có thể được đảm bảo.
- Thông tin giám sát có thể được cung cấp.

### 1.2.3 Hỗ trợ giao dịch

Một DBMS phải cung cấp một cơ chế đảm bảo rằng tất cả các cập nhật tương ứng với một giao dịch nhất định được thực hiện hoặc không có cập nhật nào được thực hiện.

Giao dịch (transaction) là một dãy tuân tự các hành động, được thực hiện bởi một người dùng hoặc chương trình ứng dụng, truy cập hoặc thay đổi nội dung của cơ sở dữ liệu. Ví dụ, một số giao dịch đơn giản cho *DreamHome* như là thêm một nhân viên mới vào cơ sở dữ liệu, cập nhật mức lương của nhân viên hoặc xóa bất động sản khỏi sổ đăng ký.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Một ví dụ phức tạp hơn là xóa một nhân viên khỏi cơ sở dữ liệu và gán lại các bất động sản mà nhân viên đó quản lý cho một nhân viên khác. Trong trường hợp này, có nhiều thay đổi được thực hiện đối với cơ sở dữ liệu. Nếu giao dịch không thành công trong khi thực hiện, có thể do sự cố máy tính, cơ sở dữ liệu sẽ ở trạng thái không nhất quán: một số thay đổi sẽ được thực hiện và những thay đổi khác thì không. Do đó, những thay đổi đã được thực hiện sẽ phải được hoàn tác để đưa cơ sở dữ liệu trở lại trạng thái nhất quán một lần nữa.

### 1.2.4 Dịch vụ điều khiển đồng thời

Một DBMS phải cung cấp một cơ chế để đảm bảo rằng cơ sở dữ liệu được cập nhật chính xác khi nhiều người dùng cập nhật cơ sở dữ liệu đồng thời.

Một mục tiêu chính trong việc sử dụng DBMS là cho phép nhiều người dùng truy cập đồng thời vào dữ liệu được chia sẻ. Truy cập đồng thời tương đối dễ dàng nếu tất cả người dùng chỉ đọc dữ liệu, vì không có cách nào mà họ có thể xen vào nhau. Tuy nhiên, khi hai hoặc nhiều người dùng truy cập cơ sở dữ liệu đồng thời và ít nhất một trong số họ đang cập nhật dữ liệu, có thể có sự xen vào nhau dẫn đến sự không nhất quán. Ví dụ, hãy xem xét hai giao dịch  $T_1$  và  $T_2$ , đang thực hiện đồng thời, như được minh họa trong Hình 1.5.

Time	$T_1$	$T_2$	$bal_x$
$t_1$		read( $bal_x$ )	100
$t_2$	read( $bal_x$ )	$bal_x = bal_x + 100$	100
$t_3$	$bal_x = bal_x - 10$	write( $bal_x$ )	200
$t_4$	write( $bal_x$ )		90
$t_5$			90

Hình 1.5 Vấn đề cập nhật bị mất

$T_1$  đang rút \$10 từ một tài khoản (có số dư  $bal_x$ ) và  $T_2$  đang chuyển \$100 vào cùng tài khoản đó. Nếu các giao dịch này được thực hiện lần lượt mà không có sự xen kẽ của các hoạt động, số dư cuối sẽ là \$190, bất kể giao dịch nào được thực hiện trước. Tuy nhiên, trong ví dụ này, các giao dịch  $T_1$  và  $T_2$  bắt đầu gần như cùng một lúc và cả hai đều đọc số dư là \$100. Sau đó,  $T_2$  tăng  $bal_x$  từ \$100 lên \$200 và lưu cập nhật vào cơ sở dữ liệu. Trong khi đó, giao dịch  $T_1$  giảm bản sao của  $bal_x$  từ \$10 xuống \$90 và lưu giá trị này vào cơ sở dữ liệu, ghi đè lên cập nhật trước đó của  $T_2$  và do đó tài khoản sẽ “mất” \$100.

### 1.2.5 Dịch vụ sao lưu

DBMS phải cung cấp cơ chế khôi phục cơ sở dữ liệu trong trường hợp cơ sở dữ liệu bị hỏng vì bất kỳ lý do gì.

Khi tìm hiểu về giao dịch, chúng ta đã đề cập rằng nếu giao dịch không thành công, thì cơ sở dữ liệu phải được trả về trạng thái nhất quán. Sự không thành công này có thể là kết quả của sự cố hệ thống, lỗi phrogram, lỗi phần cứng hoặc phần mềm khiến DBMS dừng hoặc có thể là kết quả của việc người dùng phát hiện sai sót trong quá trình giao dịch và hủy bỏ giao dịch trước khi giao dịch hoàn tất. Trong tất cả các trường hợp này, DBMS phải có khả năng đưa cơ sở dữ liệu về trạng thái nhất quán trước đó.

### 1.2.6 Dịch vụ ủy quyền

DBMS phải cung cấp một cơ chế để đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập vào cơ sở dữ liệu.

Không khó để hình dung các trường hợp mà chúng ta muốn ngăn một số dữ liệu được lưu trữ trong cơ sở dữ liệu bị tất cả người dùng nhìn thấy. Ví dụ, chúng ta có thể chỉ muốn các giám đốc chi nhánh xem thông tin liên quan đến tiền lương của nhân viên trong chi nhánh và không cho phép người dùng khác xem dữ liệu này. Ngoài ra, chúng ta có thể muốn bảo vệ cơ sở dữ liệu khỏi bị truy cập trái phép. Thuật ngữ “bảo mật - security” đề cập đến việc bảo vệ cơ sở dữ liệu chống lại sự truy cập trái phép, dù là cố ý hay vô tình. Chúng ta mong đợi DBMS cung cấp các cơ chế để đảm bảo rằng dữ liệu được an toàn.

### 1.2.7 Hỗ trợ giao tiếp dữ liệu

DBMS phải có khả năng tích hợp với phần mềm truyền thông.

Hầu hết người dùng truy cập cơ sở dữ liệu từ các máy trạm. Đôi khi các máy trạm này được kết nối trực tiếp với máy tính lưu trữ DBMS. Trong các trường hợp khác, các máy trạm ở các vị trí xa và giao tiếp với máy tính lưu trữ DBMS qua mạng. Trong cả hai trường hợp, DBMS nhận các yêu cầu dưới dạng thông điệp truyền thông (communications message) và phản hồi theo cách tương tự. Tất cả các quá trình truyền như vậy được xử lý bởi một trình quản lý truyền thông dữ liệu (data communication manager - DCM). Mặc dù DCM không phải là một phần của DBMS, nhưng DBMS cần có khả năng tích hợp được với nhiều loại DCM nếu muốn khả thi về mặt thương mại. Ngay cả các DBMS cho máy tính cá nhân cũng phải có khả năng chạy trên mạng cục bộ để một cơ sở dữ liệu tập trung có thể được thiết lập chia sẻ cho nhiều nhóm người dùng, thay vì có một loạt cơ sở dữ liệu riêng biệt, mỗi cơ sở dữ liệu cho từng nhóm người dùng.

### 1.2.8 Dịch vụ toàn vẹn

DBMS phải cung cấp một phương tiện để đảm bảo rằng dữ liệu trong cơ sở dữ liệu và các thay đổi đối với dữ liệu đều tuân theo các quy tắc nhất định.

Tính toàn vẹn của cơ sở dữ liệu (database integrity) đề cập đến tính đúng đắn và nhất quán của dữ liệu được lưu trữ, có thể được coi là một kiểu bảo vệ cơ sở dữ liệu khác. Mặc dù tính toàn vẹn có liên quan đến bảo mật, nhưng có ý nghĩa rộng hơn: tính toàn vẹn liên quan đến chất lượng của chính dữ liệu. Tính toàn vẹn thường được thể hiện dưới dạng các ràng buộc (constraint), là các quy tắc nhất quán mà cơ sở dữ liệu không được phép vi phạm.

Ví dụ, chúng ta có thể muốn chỉ định một ràng buộc rằng không nhân viên kinh doanh nào có thể quản lý hơn 100 bất động sản cho thuê tại cùng một thời điểm. Do đó, chúng ta muốn DBMS kiểm tra xem ràng buộc này khi gán một bất động sản cho một nhân viên và sẽ ngăn hoạt động gán này xảy ra nếu đã đạt đến giới hạn theo quy định.

### 1.2.9 Dịch vụ thúc đẩy tính độc lập dữ liệu

Một DBMS phải bao gồm các phương tiện hỗ trợ tính độc lập của các chương trình với cấu trúc thực tế của cơ sở dữ liệu.

Tính độc lập dữ liệu thường đạt được thông qua cơ chế khung nhìn (view) hoặc lược đồ con (subschema).

Độc lập dữ liệu vật lý dễ đạt được hơn: thường có một số kiểu thay đổi có thể được thực hiện đối với các đặc tính vật lý của cơ sở dữ liệu mà không ảnh hưởng đến các khung nhìn. Ngược lại, việc độc lập dữ liệu logic hoàn toàn khó đạt được hơn. Việc bổ sung một thực thể, thuộc tính hoặc mối quan hệ mới thường có thể được chấp nhận, nhưng không thể loại bỏ chúng.

### 1.2.10 Các dịch vụ tiện ích

Một DBMS phải cung cấp một tập hợp các dịch vụ tiện ích.

Các chương trình tiện ích giúp DBA quản lý cơ sở dữ liệu một cách hiệu quả. Một số tiện ích hoạt động ở mức bên ngoài và do đó, có thể được tạo ra bởi DBA. Các tiện ích khác hoạt động ở mức bên trong và chỉ có thể được cung cấp bởi nhà cung cấp DBMS.

Ví dụ về các tiện ích ở mức bên trong là:

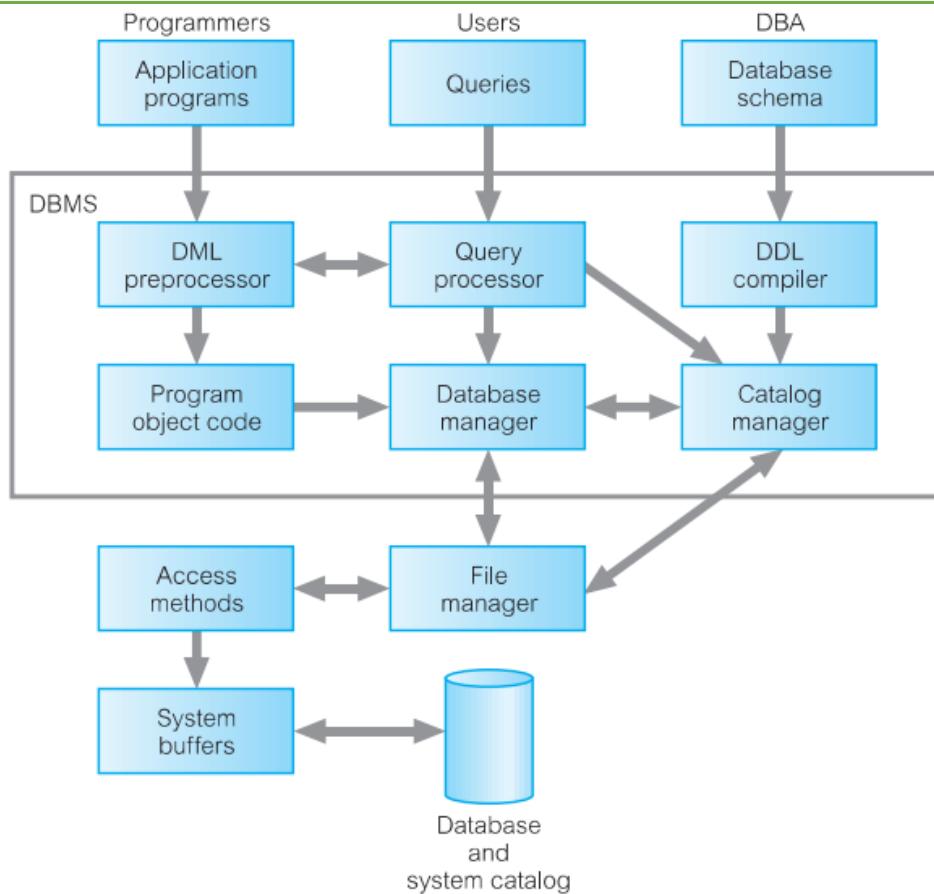
- Cơ chế nhập (import) dữ liệu, để tải cơ sở dữ liệu từ các tập tin phẳng (flat file);
- Cơ chế xuất (export) dữ liệu, để lưu cơ sở dữ liệu xuống các tập tin phẳng;
- Cơ chế giám sát, để giám sát việc sử dụng và vận hành cơ sở dữ liệu;
- Các chương trình phân tích thống kê về hiệu suất hoặc tần suất sử dụng;
- Cơ chế tổ chức lại chỉ mục, để tổ chức lại các chỉ mục;
- Thu gom rác và phân bổ lại không gian đã giải phóng.

## 1.3 CÁC THÀNH PHẦN CỦA MỘT DBMS

DBMS là những phần mềm rất phức tạp và tinh vi nhằm mục đích cung cấp các chức năng, dịch vụ đã được trình bày ở mục 1.2. Không thể khái quát cấu trúc thành phần của một DBMS, vì rất khác nhau giữa các hệ thống. Tuy nhiên, sẽ rất hữu ích khi tìm hiểu các thành phần của hệ quản trị cơ sở dữ liệu và mối quan hệ giữa chúng. Trong phần này, chúng ta sẽ tìm hiểu một kiến trúc khả thi cho một DBMS tiêu biểu.

DBMS được phân chia thành một số thành phần phần mềm (hoặc module), mỗi thành phần trong số đó được chỉ định một hoạt động cụ thể. Như đã nêu trước đây, một số chức năng của DBMS được hỗ trợ bởi hệ điều hành bên dưới. Tuy nhiên, hệ điều hành chỉ cung cấp các dịch vụ cơ bản và DBMS phải được xây dựng trên đó. Do đó, việc thiết kế một DBMS phải tính đến giao diện giữa DBMS và hệ điều hành.

Các thành phần phần mềm chính trong môi trường DBMS được mô tả trong Hình 1.6. Sơ đồ này cho thấy cách DBMS giao tiếp với các thành phần phần mềm khác, chẳng hạn như truy vấn của người dùng và phương thức truy cập (các kỹ thuật quản lý tập tin để lưu trữ và truy xuất bản ghi dữ liệu).



**Hình 1.6** Các thành phần chính của DBMS

Các thành phần có trong Hình 1.6:

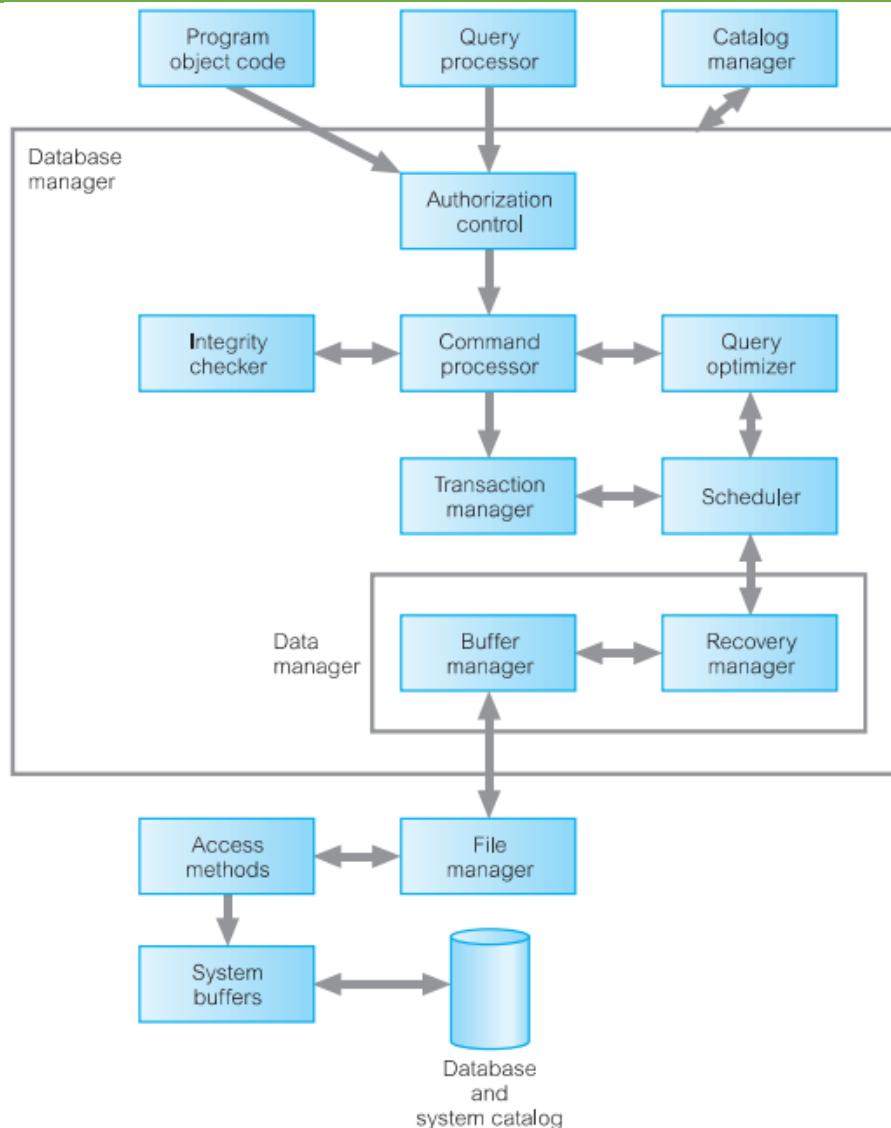
- Bộ xử lý truy vấn (Query processor). Đây là thành phần chính của DBMS, chuyển đổi các truy vấn thành một loạt các lệnh cấp thấp và chuyển đến trình quản lý cơ sở dữ liệu.
- Trình quản lý cơ sở dữ liệu (Database manager - DM). DM giao tiếp với các chương trình ứng dụng và truy vấn do người dùng gửi. DM chấp nhận các truy vấn, kiểm tra các lược đồ bên ngoài và khái niệm để xác định những bản ghi khái niệm được yêu cầu để đáp ứng truy vấn, sau đó thực hiện lời gọi đến trình quản lý tập tin để thực hiện yêu cầu. Các thành phần của DM được thể hiện trong Hình 1.7.
- Trình quản lý tập tin (File manager). Trình quản lý tập tin thao tác với các tập tin lưu trữ bên dưới và quản lý việc phân bổ không gian lưu trữ trên đĩa. Nó thiết lập và duy trì danh sách các cấu trúc và chỉ mục được định nghĩa trong lược đồ bên trong. Nếu các tập tin băm được sử dụng, nó sẽ gọi các hàm băm để tạo địa chỉ bản ghi.

Tuy nhiên, trình quản lý tập tin không trực tiếp quản lý đầu vào và đầu ra vật lý của dữ liệu. Thay vào đó, nó chuyển các yêu cầu tới các phương thức truy cập thích hợp, các phương thức này có thể đọc dữ liệu từ hoặc ghi dữ liệu vào bộ nhớ đệm hệ thống (cache).

- Bộ tiền xử lý DML (DML preprocessor). Mô-đun này chuyển đổi các câu lệnh DML được nhúng trong một chương trình ứng dụng thành các lời gọi hàm tiêu chuẩn trong ngôn ngữ máy chủ. Bộ tiền xử lý DML phải tương tác với bộ xử lý truy vấn để tạo mã thích hợp.
- Trình biên dịch DDL (DDL compiler). Trình biên dịch DDL chuyển đổi các câu lệnh DDL thành một tập hợp các bảng chứa siêu dữ liệu. Các bảng này sau đó được lưu trữ trong danh mục hệ thống trong khi thông tin điều khiển được lưu trữ trong phần đầu tập tin dữ liệu.
- Trình quản lý danh mục (Catalog manager). Trình quản lý danh mục quản lý quyền truy cập và duy trì danh mục hệ thống. Danh mục hệ thống được truy cập bởi hầu hết các thành phần DBMS.

Các thành phần phần mềm chính của trình quản lý cơ sở dữ liệu (DM) như sau:

- Kiểm soát ủy quyền (Authorization control). Mô-đun này xác nhận liệu người dùng có đủ quyền cần thiết để thực hiện các hoạt động đã yêu cầu hay không.
- Bộ xử lý lệnh (Command processor). Khi hệ thống đã xác nhận rằng người dùng có đủ quyền thực hiện hoạt động, quyền kiểm soát sẽ được chuyển cho bộ xử lý lệnh.
- Trình kiểm tra tính toàn vẹn (Integrity checker). Đối với một thao tác làm thay đổi cơ sở dữ liệu, trình kiểm tra tính toàn vẹn sẽ kiểm tra xem thao tác được yêu cầu có thỏa mãn tất cả các ràng buộc toàn vẹn cần thiết hay không (chẳng hạn như các ràng buộc khóa).
- Trình tối ưu hóa truy vấn (Query optimizer). Mô-đun này xác định một chiến lược tối ưu cho việc thực thi truy vấn. Chúng ta sẽ tìm hiểu về tối ưu hóa truy vấn trong Chương 5.
- Trình quản lý giao dịch (Transaction manager). Mô-đun này thực hiện xử lý yêu cầu các hoạt động mà nó nhận được từ các giao dịch.
- Trình lập lịch (Scheduler). Mô-đun này chịu trách nhiệm đảm bảo rằng các hoạt động đồng thời trên cơ sở dữ liệu được tiến hành mà không xung đột với nhau. Nó kiểm soát thứ tự tương đối mà các hoạt động giao dịch được thực hiện.
- Trình quản lý phục hồi (Recovery manager). Mô-đun này đảm bảo rằng cơ sở dữ liệu vẫn ở trạng thái nhất quán khi có lỗi. Nó chịu trách nhiệm về cam kết (commit) và hủy bỏ (abort) giao dịch.
- Trình quản lý bộ đệm (Buffer manager). Mô-đun này chịu trách nhiệm chuyển dữ liệu giữa bộ nhớ chính và bộ nhớ thứ cấp, chẳng hạn như đĩa và băng từ. Trình quản lý khôi phục và trình quản lý bộ đệm đôi khi được gọi chung là trình quản lý dữ liệu (data manager). Trình quản lý bộ đệm đôi khi được gọi là trình quản lý bộ nhớ cache (cache manager).



Hình 1.7 Các thành phần của trình quản lý cơ sở dữ liệu (DM)

#### 1.4 KIẾN TRÚC BA MỨC ANSI-SPARC

Mục đích chính của hệ cơ sở dữ liệu là cung cấp cho người dùng một cách nhìn trùu tượng về dữ liệu, ẩn một số chi tiết nhất định về cách dữ liệu được lưu trữ và thao tác. Do đó, điểm khởi đầu cho việc thiết kế cơ sở dữ liệu phải là một mô tả trùu tượng và tổng quát về các yêu cầu thông tin của tổ chức sẽ được thể hiện trong cơ sở dữ liệu. Trong nội dung này và xuyên suốt giáo trình, chúng ta sử dụng thuật ngữ “tổ chức” một cách lỏng lẻo để chỉ toàn bộ tổ chức hoặc một phần của tổ chức.

Chẳng hạn, trong ví dụ *DreamHome*, chúng ta có thể quan tâm đến việc lập mô hình:

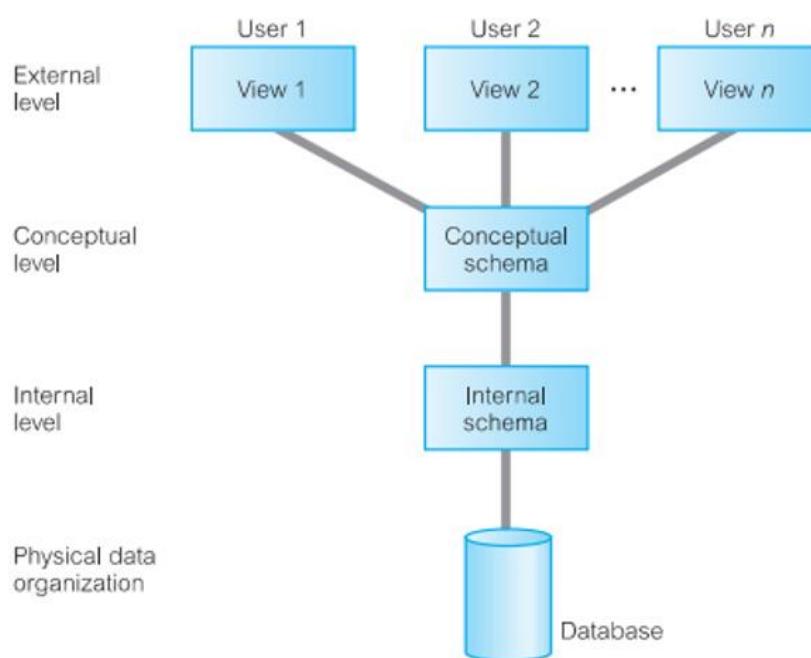
- Các thực thể (entities) trong “thế giới thực” như Staff, PropertyForRent, PrivateOwner, và Client;
- Các thuộc tính (attributes) mô tả đặc điểm hoặc phẩm chất của từng thực thể (mỗi mục Staff có tên, chức vụ và mức lương);
- Các mối quan hệ (relationships) giữa các thực thể này (Staff quản lý PropertyForRent).

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Hơn nữa, bởi vì cơ sở dữ liệu là một tài nguyên được chia sẻ, mỗi người dùng có thể yêu cầu một khung nhìn khác nhau về dữ liệu được lưu trữ trong cơ sở dữ liệu. Để đáp ứng những nhu cầu này, kiến trúc của hầu hết các DBMS thương mại hiện có, ở một mức độ nào đó, dựa trên cái gọi là kiến trúc ANSI-SPARC. Trong nội dung này, chúng ta tìm hiểu về các đặc điểm và chức năng của DBMS theo kiến trúc này.

Một đề xuất ban đầu cho một thuật ngữ tiêu chuẩn và kiến trúc chung cho các hệ quản trị cơ sở dữ liệu đã được đưa ra vào năm 1971 bởi Nhóm đặc trách cơ sở dữ liệu (Data Base Task Group - DBTG) của Hội nghị Ngôn ngữ Hệ thống Dữ liệu (Conference on Data Systems Languages - CODASYL). DBTG nhận thấy sự cần thiết của cách tiếp cận hai mức với một khung nhìn hệ thống được gọi là lược đồ (schema) và các khung nhìn người dùng được gọi là các lược đồ con (subschema). Ủy ban Kế hoạch và Nhu cầu (Standards Planning and Requirements Committee - SPARC) của Viện Tiêu chuẩn Quốc gia Hoa Kỳ (American National Standards Institute - ANSI) đã đưa ra một thuật ngữ và kiến trúc tương tự vào năm 1975. Mặc dù mô hình ANSI-SPARC không trở thành tiêu chuẩn, nhưng vẫn cung cấp cơ sở để hiểu một số chức năng của DBMS.

Đối với mục đích của chúng ta, điểm cơ bản của các báo cáo này và các báo cáo sau đó là việc xác định ba mức trừu tượng, tức là ba mức riêng biệt mà tại đó các mục dữ liệu có thể được mô tả. Các mức tạo thành một kiến trúc ba mức (three-level architecture) bao gồm mức bên ngoài (external level), mức khái niệm (conceptual level) và mức bên trong (internal level), như được mô tả trong Hình 1.8. Ba mức này đôi khi còn được gọi là mức khung nhìn, mức logic và mức vật lý một cách tương ứng. Cách người dùng cảm nhận dữ liệu được gọi là mức bên ngoài. Cách DBMS và hệ điều hành nhận thức dữ liệu là mức bên trong, nơi dữ liệu thực sự được lưu trữ bằng cách sử dụng cấu trúc dữ liệu và tổ chức tập tin. Mức khái niệm cung cấp việc ánh xạ và sự độc lập mong muốn giữa các mức bên ngoài và bên trong.



**Hình 1.8** Kiến trúc ba mức ANSI-SPARC

Mục tiêu của kiến trúc ba mức là tách biệt khung nhìn của mỗi (nhóm) người dùng về cơ sở dữ liệu khỏi cách cơ sở dữ liệu được biểu diễn vật lý vì các lý do sau:

- Mỗi người dùng có thể truy cập vào cùng một dữ liệu, nhưng có một khung nhìn dữ liệu được tùy chỉnh khác nhau. Mỗi người dùng có thể thay đổi cách họ xem dữ liệu và thay đổi này sẽ không ảnh hưởng đến những người dùng khác.
- Người dùng không phải xử lý trực tiếp các chi tiết lưu trữ cơ sở dữ liệu vật lý, chẳng hạn như lập chỉ mục hoặc băm. Nói cách khác, tương tác của người dùng với cơ sở dữ liệu phải độc lập với các quan tâm về bộ nhớ.
- DBA sẽ có thể thay đổi cấu trúc lưu trữ cơ sở dữ liệu mà không ảnh hưởng đến các khung nhìn của người dùng.
- Cấu trúc bên trong của cơ sở dữ liệu không bị ảnh hưởng bởi những thay đổi đối với các khía cạnh vật lý của lưu trữ, chẳng hạn như chuyển đổi sang thiết bị lưu trữ mới.
- DBA sẽ có thể thay đổi cấu trúc khái niệm của cơ sở dữ liệu mà không ảnh hưởng đến tất cả người dùng.

#### 1.4.1 Mức bên ngoài

**Mức bên ngoài** Là khung nhìn cơ sở dữ liệu của người dùng. Mức này mô tả một phần của mức cơ sở dữ liệu có liên quan đến từng người dùng.

Mức bên ngoài (external level) bao gồm một số khung nhìn bên ngoài khác nhau của cơ sở dữ liệu. Mỗi người dùng có một khung nhìn về “thế giới thực” được thể hiện trong một biểu mẫu quen thuộc với người dùng đó. Khung nhìn bên ngoài chỉ bao gồm các thực thể, thuộc tính và mối quan hệ trong “thế giới thực” mà người dùng quan tâm. Các thực thể, thuộc tính hoặc mối quan hệ khác không được quan tâm có thể được trình bày trong cơ sở dữ liệu nhưng người dùng sẽ không biết đến chúng.

Ngoài ra, các khung nhìn khác nhau có thể có các biểu diễn khác nhau của cùng một dữ liệu. Ví dụ, một người dùng có thể xem ngày ở dạng ngày, tháng, năm; trong khi người khác có thể xem ngày dưới dạng năm, tháng, ngày. Một số khung nhìn có thể bao gồm dữ liệu dẫn xuất (derived data) hoặc dữ liệu được tính toán: dữ liệu không thực sự được lưu trữ trong cơ sở dữ liệu, nhưng sẽ được tạo ra khi cần thiết. Trong ví dụ *DreamHome*, chúng ta có thể muốn xem tuổi của một nhân viên. Tuy nhiên, không gì đảm bảo rằng thông tin tuổi được lưu trữ, thay vào đó, ngày sinh nhân viên sẽ được lưu trữ và tuổi sẽ được DBMS tính toán khi được tham chiếu.

#### 1.4.2 Mức khái niệm

**Mức khái niệm** Là khung nhìn chung của cơ sở dữ liệu. Mức này mô tả dữ liệu gì (what) được lưu trữ trong cơ sở dữ liệu và các mối quan hệ giữa các dữ liệu.

Mức khái niệm (conceptual level) chứa cấu trúc logic của toàn bộ cơ sở dữ liệu theo khung nhìn của DBA. Đây là một khung nhìn đầy đủ về các yêu cầu dữ liệu của tổ chức mà không phụ thuộc vào bất kỳ cân nhắc lưu trữ nào.

#### Mức khái niệm thể hiện:

- Tất cả các thực thể, các thuộc tính của chúng và các mối quan hệ của chúng;
- Các ràng buộc về dữ liệu;
- Thông tin ngữ nghĩa về dữ liệu;
- Thông tin bảo mật và toàn vẹn.

Mức khái niệm hỗ trợ mỗi khung nhìn ở mức bên ngoài, trong đó bất kỳ dữ liệu nào có sẵn cho người dùng phải được chứa trong hoặc có thể dẫn xuất từ mức khái niệm. Tuy nhiên, mức này không được chứa bất kỳ chi tiết nào phụ thuộc vào lưu trữ. Ví dụ, mô tả của một thực thể chỉ nên chứa các kiểu dữ liệu của các thuộc tính (như số nguyên, số thực, ký tự) và độ dài của chúng (chẳng hạn như số lượng tối đa của ký số hoặc ký tự), nhưng không chứa bất kỳ cân nhắc lưu trữ nào, chẳng hạn như tổng số byte bị chiếm dụng.

#### 1.4.3 Mức bên trong

**Mức bên trong** Là biểu diễn vật lý của cơ sở dữ liệu trên máy tính. Mức này mô tả cách dữ liệu (how) được lưu trữ trong cơ sở dữ liệu.

Mức bên trong (internal level) bao gồm việc triển khai thực tế của cơ sở dữ liệu để đạt được hiệu suất tối ưu về thời gian chạy và sử dụng không gian lưu trữ. Nó bao gồm các cấu trúc dữ liệu và tổ chức tập tin được sử dụng để lưu trữ dữ liệu trên các thiết bị lưu trữ. Nó giao tiếp với các phương thức truy cập hệ điều hành (kỹ thuật quản lý tập tin để lưu trữ và truy xuất bản ghi dữ liệu) để đặt dữ liệu trên thiết bị lưu trữ, xây dựng chỉ mục, truy xuất dữ liệu,...

Mức bên trong quan tâm đến những vấn đề như:

- Phân bổ không gian lưu trữ cho dữ liệu và chỉ mục;
- Ghi lại các mô tả để lưu trữ (với kích thước được lưu trữ cho các mục dữ liệu);
- Sắp xếp bản ghi;
- Các kỹ thuật nén dữ liệu và mã hóa dữ liệu.

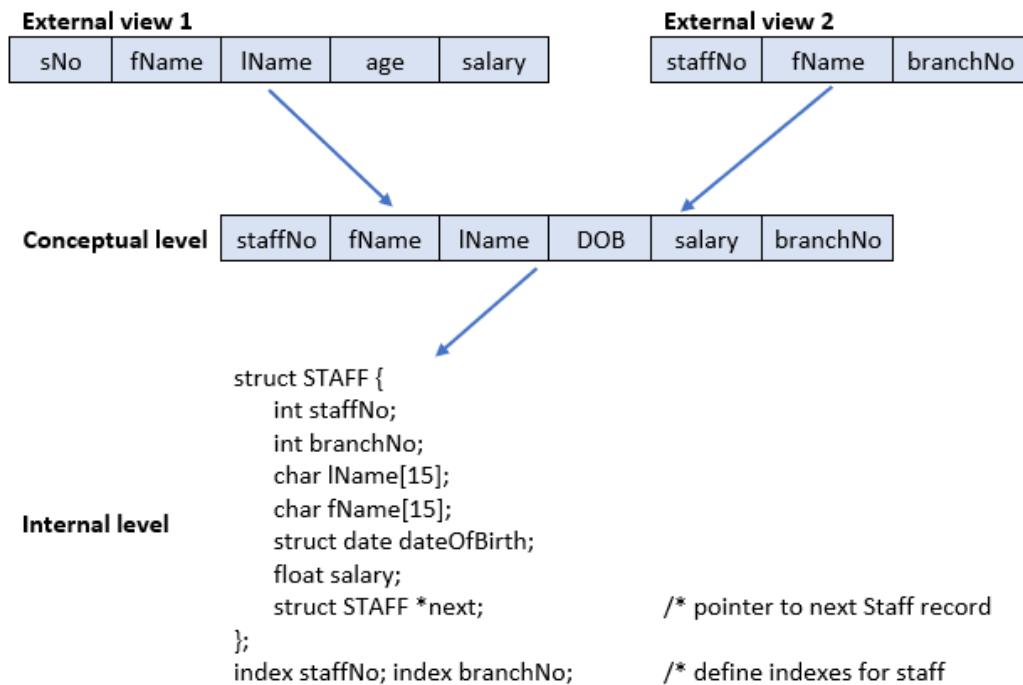
#### 1.4.4 Lược đồ, Ánh xạ và Thể hiện

Mô tả tổng thể của cơ sở dữ liệu được gọi là lược đồ cơ sở dữ liệu (database schema). Có ba loại lược đồ khác nhau trong cơ sở dữ liệu và chúng được xác định theo mức độ trừu tượng của kiến trúc ba mức được minh họa trong Hình 1.8. Ở mức cao nhất, có nhiều lược đồ bên ngoài (external schema), đôi khi còn gọi là lược đồ con (subschema), tương ứng với các khung nhìn khác nhau của dữ liệu. Ở mức khái niệm, có lược đồ khái niệm (conceptual schema), lược đồ này mô tả tất cả các thực thể, thuộc tính và mối quan hệ cùng với các ràng buộc toàn vẹn. Ở mức trừu tượng thấp nhất, có lược đồ bên trong (internal schema), là một mô tả đầy đủ về mô hình bên trong, chứa các định nghĩa về các bản ghi được lưu trữ, các phương thức biểu diễn, các trường dữ liệu, các chỉ mục và cấu trúc lưu trữ được sử dụng. Chỉ có một lược đồ khái niệm và một lược đồ bên trong cho mỗi cơ sở dữ liệu.

DBMS chịu trách nhiệm ánh xạ giữa ba loại lược đồ này và kiểm tra tính nhất quán của các lược đồ; nói cách khác, DBMS phải xác nhận rằng mỗi lược đồ bên ngoài có thể được dẫn xuất từ lược đồ khái niệm và nó phải sử dụng thông tin trong lược đồ khái niệm để ánh xạ giữa mỗi lược đồ bên ngoài và lược đồ bên trong. Lược đồ khái niệm có liên quan đến lược đồ bên trong thông qua một ánh xạ khái niệm/bên trong. Ánh xạ này cho phép DBMS tìm thấy bản ghi thực tế hoặc tổ hợp các bản ghi trong bộ lưu trữ vật lý tạo thành bản ghi lôgic trong lược đồ khái niệm, cùng với bất kỳ ràng buộc nào được thực thi đối với các hoạt động của bản ghi lôgic đó. Nó cũng cho phép giải quyết mọi khác biệt về tên thực thể, tên thuộc tính, thứ tự thuộc tính, kiểu dữ liệu,... Cuối cùng, mỗi lược đồ bên ngoài có liên quan đến lược đồ khái niệm bằng ánh xạ bên ngoài/khai niêm. Ánh xạ này cho phép DBMS ánh xạ các tên trong khung nhìn của người dùng đến phần có liên quan của lược đồ khái niêm.

Ví dụ về các cấp độ khác nhau được thể hiện trong Hình 1.9. Có hai khung nhìn bên ngoài khác nhau về chi tiết nhân viên: một khung nhìn bao gồm mã số nhân viên (sNo), tên (fName), họ (lName), tuổi (age) và lương (salary); một khung nhìn khác bao gồm mã số nhân viên (staffNo), họ (lName) và mã số chi nhánh mà nhân viên làm việc tại (branchNo). Các khung nhìn bên ngoài này được hợp nhất thành một khung nhìn khái niêm. Trong quá trình hợp nhất này, điểm khác biệt chính là trường tuổi đã được thay đổi thành trường ngày sinh, DOB. DBMS duy trì ánh xạ bên ngoài/khai niêm; ví dụ, ánh xạ trường sNo của khung nhìn bên ngoài đầu tiên với trường staffNo của bản ghi khái niêm. Mức khái niêm sau đó được ánh xạ tới mức bên trong, mức này chứa mô tả vật lý về cấu trúc cho bản ghi khái niêm. Ở mức này, chúng ta thấy một định nghĩa về cấu trúc trong một ngôn ngữ cấp cao. Cấu trúc chứa một con trỏ, next, cho phép danh sách các bản ghi nhân viên được liên kết vật lý với nhau để tạo thành một chuỗi. Lưu ý rằng thứ tự của các trường ở mức bên trong khác với thứ tự của các trường ở mức khái niêm.

Điều quan trọng là phải phân biệt giữa mô tả của cơ sở dữ liệu và chính cơ sở dữ liệu đó. Mô tả của cơ sở dữ liệu là lược đồ cơ sở dữ liệu. Lược đồ được chỉ định trong quá trình thiết kế cơ sở dữ liệu và dự kiến sẽ không thay đổi thường xuyên. Tuy nhiên, dữ liệu thực tế trong cơ sở dữ liệu có thể thay đổi thường xuyên; ví dụ, thay đổi mỗi khi chúng ta chèn thông tin chi tiết về một nhân viên mới hoặc một bất động sản mới. Dữ liệu trong cơ sở dữ liệu tại bất kỳ thời điểm cụ thể nào được gọi là một thể hiện cơ sở dữ liệu (database instance). Do đó, có thể có nhiều thể hiện cơ sở dữ liệu tương ứng với cùng một lược đồ cơ sở dữ liệu.

**Hình 1.9** Sự khác biệt giữa ba mức độ trùu tượng

#### 1.4.5 Độc lập dữ liệu

Mục tiêu chính của kiến trúc ba mức là cung cấp sự độc lập dữ liệu (data independence), có nghĩa là các mức trên không bị ảnh hưởng bởi các thay đổi của các mức thấp hơn. Có hai loại độc lập dữ liệu: logic và vật lý.

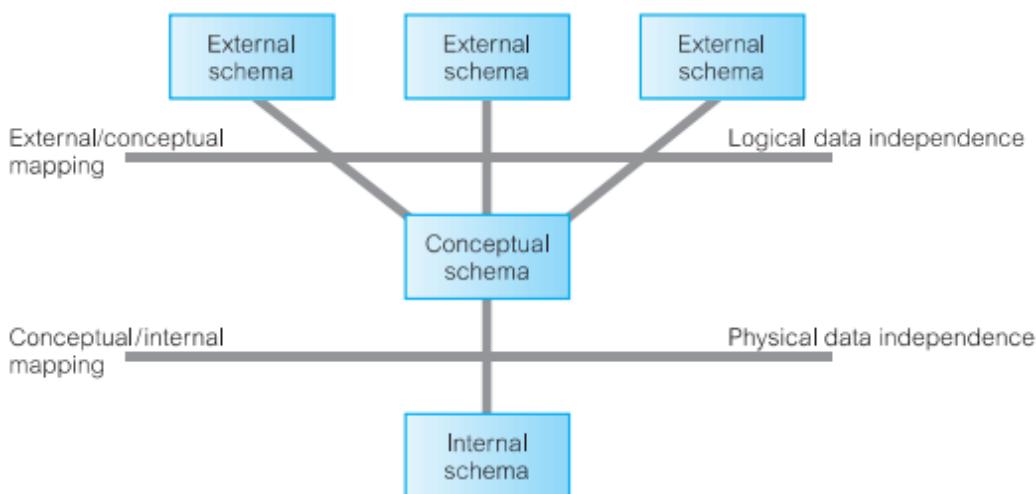
**Độc lập dữ liệu logic** Khả năng không bị ảnh hưởng của các lược đồ bên ngoài đối với những thay đổi trong lược đồ khái niệm.

Các thay đổi đối với lược đồ khái niệm, chẳng hạn như thêm hoặc xóa các thực thể, thuộc tính hoặc mối quan hệ mới, có thể thực hiện được mà không cần phải thay đổi các lược đồ bên ngoài hiện có hoặc phải viết lại các chương trình ứng dụng. Rõ ràng, có những người dùng mà các thay đổi đã được thực hiện cần phải để cho họ biết, nhưng đối với một số người dùng khác thì điều này là không cần thiết.

**Độc lập dữ liệu vật lý** Khả năng không bị ảnh hưởng của lược đồ khái niệm đối với những thay đổi của lược đồ bên trong.

Có thể thực hiện các thay đổi đối với lược đồ bên trong, chẳng hạn như sử dụng các tổ chức tập tin hoặc cấu trúc lưu trữ khác nhau, sử dụng các thiết bị lưu trữ khác nhau, sửa đổi chỉ mục hoặc thuật toán băm mà không cần phải thay đổi lược đồ khái niệm hoặc lược đồ bên ngoài. Hình 1.10 minh họa nơi mà mỗi loại độc lập dữ liệu xảy ra trong mối quan hệ với kiến trúc ba mức.

Ánh xạ hai giai đoạn trong kiến trúc ANSI-SPARC có thể không hiệu quả, nhưng nó cũng cung cấp tính độc lập dữ liệu cao hơn. Tuy nhiên, để ánh xạ hiệu quả hơn, mô hình ANSI-SPARC cho phép ánh xạ trực tiếp các lược đồ bên ngoài vào lược đồ bên trong, do đó bỏ qua lược đồ khái niệm. Việc ánh xạ này tất nhiên làm giảm tính độc lập của dữ liệu, do đó mỗi khi lược đồ bên trong thay đổi, lược đồ bên ngoài và bất kỳ chương trình ứng dụng phụ thuộc nào cũng có thể phải thay đổi.

**Hình 1.10** Độc lập dữ liệu và kiến trúc ba mức ANSI-SPARC

## 1.5 KIẾN TRÚC DBMS ĐA NGƯỜI DÙNG

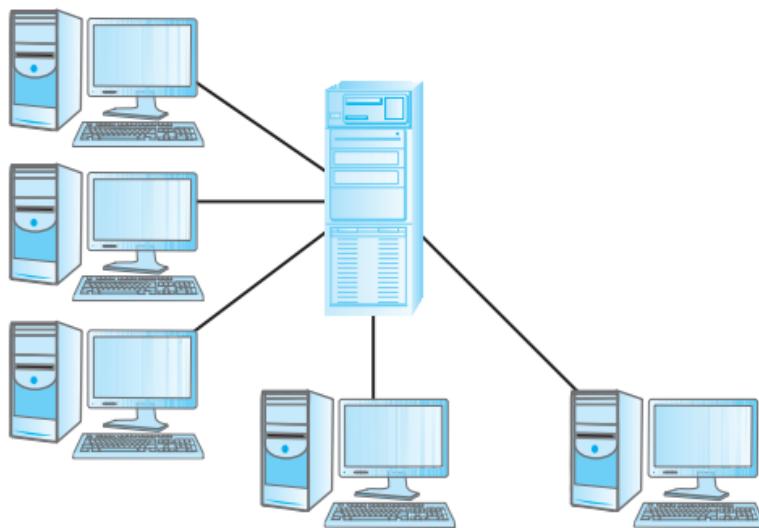
Trong giai đoạn hiện nay, mặc dù đã hiểu rõ hơn về chức năng mà người dùng yêu cầu và các mô hình dữ liệu cơ bản mới đã được đề xuất và triển khai, nhưng mô hình phần mềm được sử dụng để phát triển hệ thống phần mềm nói chung đang có những thay đổi đáng kể. Các nhà cung cấp hệ thống cơ sở dữ liệu đã phải nhận ra những thay đổi này và thích ứng với chúng để đảm bảo rằng hệ thống của họ luôn cập nhật với tư duy mới nhất. Trong phần này, chúng ta xem xét các kiến trúc phổ biến được sử dụng để triển khai hệ quản trị cơ sở dữ liệu đa người dùng: xử lý từ xa, máy chủ tập tin và client-server.

### 1.5.1 Xử lý từ xa

Kiến trúc truyền thống cho các hệ thống đa người dùng là xử lý từ xa (teleprocessing), trong đó có một máy tính với một đơn vị xử lý trung tâm (CPU) duy nhất và một số thiết bị đầu cuối, như được minh họa trong Hình 1.11. Tất cả quá trình xử lý được thực hiện trong ranh giới của cùng một máy tính vật lý. Các thiết bị đầu cuối của người dùng không có khả năng tự xử lý và được kết nối với máy tính trung tâm, chúng gửi các thông điệp qua hệ thống con điều khiển giao tiếp của hệ điều hành đến chương trình ứng dụng của người dùng, chương trình này sẽ sử dụng các dịch vụ của DBMS. Theo cách tương tự, các thông điệp sẽ được truyền trả lại thiết bị đầu cuối của người dùng.

Rõ ràng, kiến trúc này đã đặt một gánh nặng to lớn lên máy tính trung tâm, máy tính này không chỉ phải chạy các chương trình ứng dụng và DBMS mà còn phải thực hiện một lượng lớn công việc thay cho các thiết bị đầu cuối.

Trong những năm gần đây, đã có những tiến bộ đáng kể trong việc phát triển mạng và máy tính cá nhân hiệu suất cao. Hiện nay có một xu hướng đang được hướng đến mạnh mẽ đó là giảm quy mô (downsizing), thay thế các máy tính lớn đắt tiền bằng các mạng của các máy tính cá nhân tiết kiệm chi phí hơn, nhưng vẫn đạt được kết quả tương tự, hoặc thậm chí tốt hơn. Xu hướng này đã làm phát sinh hai kiến trúc tiếp theo: máy chủ tập tin và client-server.

**Hình 1.11** Kiến trúc xử lý từ xa

### 1.5.2 Kiến trúc máy chủ tập tin

#### Máy chủ tập tin

Máy tính được kết nối vào mạng với mục đích chính là cung cấp dung lượng lưu trữ dùng chung cho các tập tin máy tính như tài liệu, bảng tính, hình ảnh và cơ sở dữ liệu.

Trong môi trường máy chủ tập tin (File-Server), quá trình xử lý được phân phối trên mạng, thường là mạng cục bộ (Local Area Network - LAN). Máy chủ tập tin chứa các tập tin sẽ được yêu cầu bởi các ứng dụng và DBMS. Tuy nhiên, các ứng dụng và DBMS chạy trên mỗi máy trạm, yêu cầu các tập tin từ máy chủ tập tin khi cần thiết, như được minh họa trong Hình 1.12. Theo cách này, máy chủ tập tin hoạt động đơn giản như một ổ đĩa cứng dùng chung. DBMS trên mỗi máy trạm gửi yêu cầu đến máy chủ tập tin cho tất cả dữ liệu mà DBMS yêu cầu được lưu trữ trên đĩa. Cách tiếp cận này có thể tạo ra một lượng lưu lượng mạng đáng kể, có thể dẫn đến các vấn đề về hiệu suất.

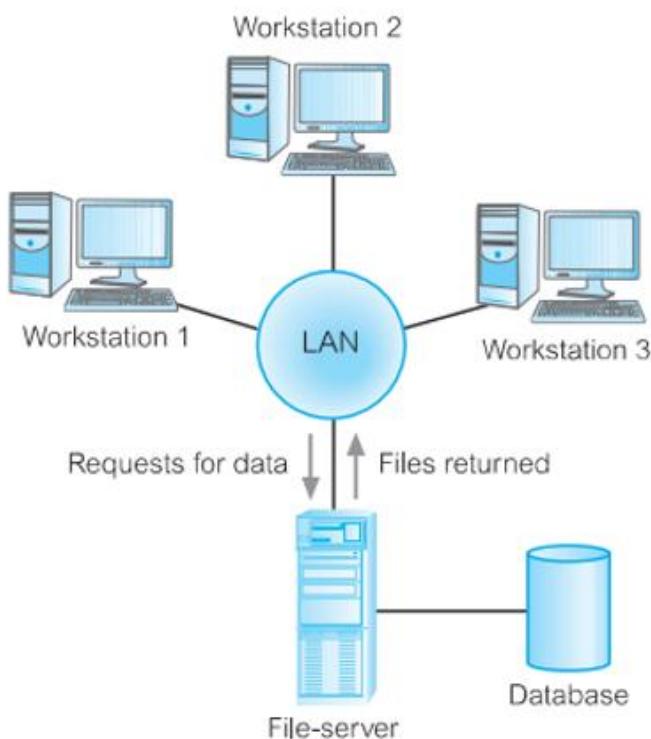
Ví dụ, hãy xem xét một yêu cầu của người dùng: liệt kê danh sách tên của các nhân viên làm việc tại chi nhánh có địa chỉ là 163 Main St. Chúng ta có thể hiện yêu cầu này bằng câu lệnh SQL như sau:

```
SELECT fName, lName
FROM Brand b, Staff s
WHERE b.branchNo = s.branchNo AND b.street = '163 Main St.'
```

Vì máy chủ tập tin không có “kiến thức” về SQL, DBMS phải yêu cầu các tập tin tương ứng với các quan hệ Branch và Staff từ máy chủ tập tin, thay vì chỉ danh sách tên nhân viên thỏa mãn truy vấn.

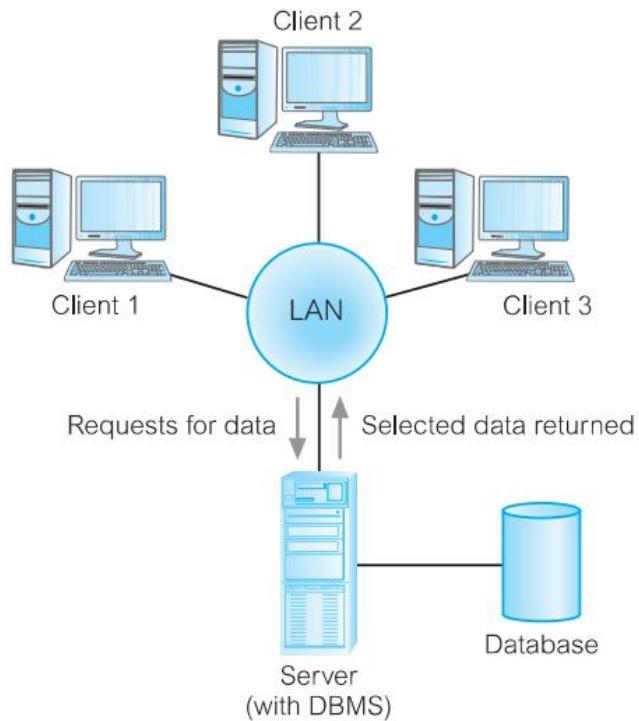
Do đó, kiến trúc máy chủ tập tin có ba hạn chế chính:

- Phát sinh một lượng lớn lưu lượng mạng.
- Một bản sao đầy đủ của DBMS được yêu cầu trên mỗi máy trạm.
- Điều khiển đồng thời, khôi phục và toàn vẹn phức tạp hơn, vì có thể có nhiều DBMS truy cập vào các tập tin giống nhau.

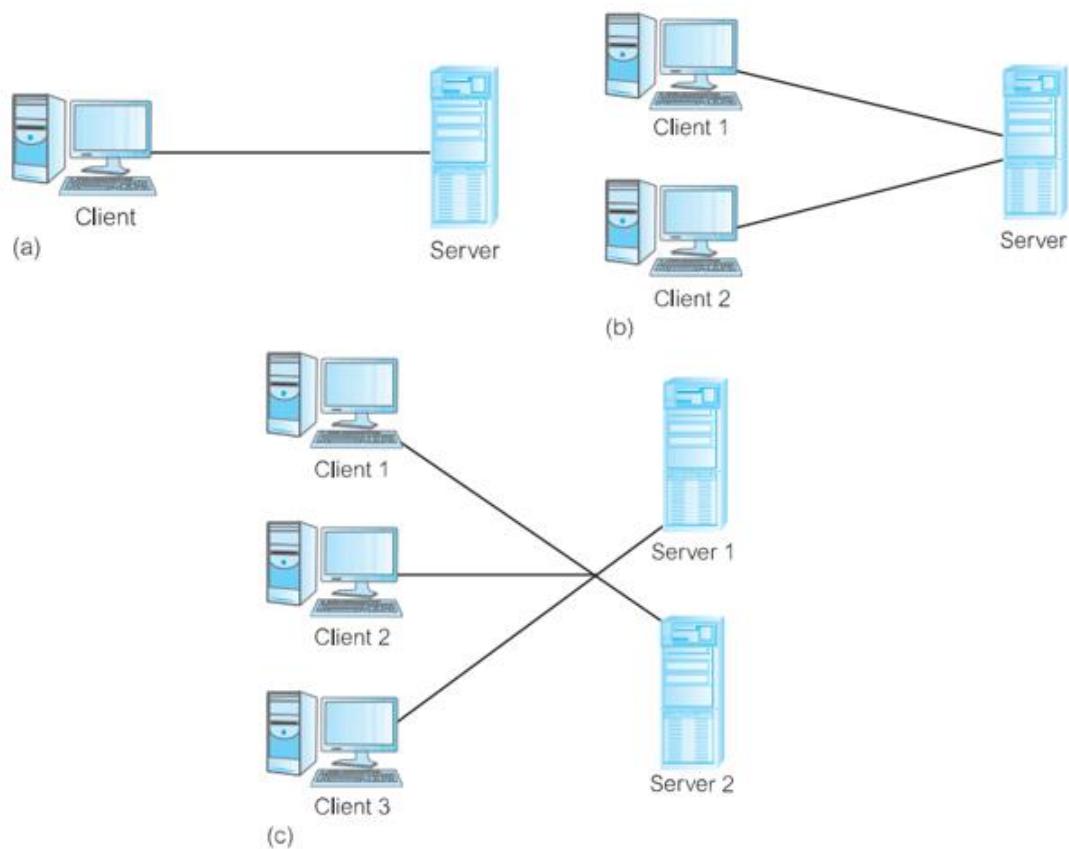
**Hình 1.12** Kiến trúc máy chủ tập tin

### 1.5.3 Kiến trúc client-server hai tầng truyền thông

Để khắc phục những nhược điểm của hai cách tiếp cận đầu tiên và thích ứng với môi trường kinh doanh ngày càng phi tập trung, kiến trúc client-server (máy khách-máy chủ) đã được phát triển. Client-server đề cập đến cách thức mà các thành phần phần mềm tương tác để tạo thành một hệ thống. Như tên cho thấy, có một client, yêu cầu một số tài nguyên và một server, cung cấp tài nguyên. Không có yêu cầu rằng client và server phải nằm trên cùng một máy. Trong thực tế, khá phổ biến khi đặt một server tại một vị trí trong mạng LAN và các client ở các vị trí khác. Hình 1.13 minh họa kiến trúc client-server và Hình 1.14 cho thấy một số kết hợp có thể có của cấu trúc liên kết client-server. Các ứng dụng nghiệp vụ chuyên về dữ liệu bao gồm bốn thành phần chính: cơ sở dữ liệu, logic giao dịch, logic ứng dụng dữ liệu và nghiệp vụ, và giao diện người dùng. Kiến trúc client-server hai tầng truyền thông cung cấp sự phân tách rất cơ bản của các thành phần này. Client (tầng 1) chịu trách nhiệm chính về việc biểu diễn (presentation) dữ liệu đối với người dùng và server (tầng 2) chịu trách nhiệm chính trong việc cung cấp các dịch vụ dữ liệu cho client, như được minh họa trong Hình 1.15. Các dịch vụ hiển thị xử lý các hành động giao diện người dùng và logic ứng dụng dữ liệu và nghiệp vụ chính. Dịch vụ dữ liệu cung cấp logic ứng dụng nghiệp vụ hạn chế, thường là xác thực mà khách hàng không thể thực hiện do thiếu thông tin và quyền truy cập vào dữ liệu được yêu cầu, độc lập với vị trí của nó. Thông thường, client sẽ chạy trên máy tính để bàn của người dùng cuối và tương tác với máy chủ cơ sở dữ liệu tập trung qua mạng.



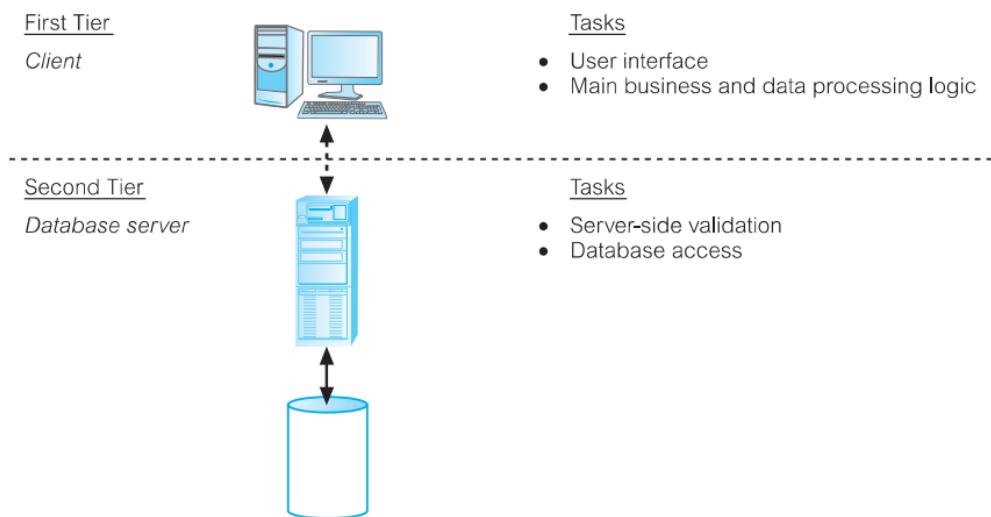
**Hình 1.13** Kiến trúc client-server



**Hình 1.14** Các hình thái client-server khác: (a) một client, một server; (b) nhiều client, một server;  
(c) nhiều client, nhiều server

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Một tương tác điển hình giữa client và server như sau: client nhận yêu cầu của người dùng, kiểm tra cú pháp và tạo các yêu cầu cơ sở dữ liệu bằng SQL hoặc ngôn ngữ cơ sở dữ liệu khác phù hợp với logic ứng dụng. Sau đó, nó truyền thông điệp đến server, chờ phản hồi và định dạng phản hồi cho người dùng. Server chấp nhận và xử lý các yêu cầu cơ sở dữ liệu, sau đó truyền kết quả trả lại client. Quá trình xử lý bao gồm việc kiểm tra ủy quyền, đảm bảo tính toàn vẹn, duy trì danh mục hệ thống và thực hiện xử lý truy vấn và cập nhật. Ngoài ra, server cũng cung cấp khả năng kiểm soát đồng thời và khôi phục. Hoạt động của client và server được tóm tắt trong Bảng 1.3.



**Hình 1.15** Kiến trúc client-server hai tầng truyền thống

Có rất nhiều lợi thế cho kiểu kiến trúc này. Ví dụ:

- Cho phép truy cập rộng hơn vào các cơ sở dữ liệu hiện có.
- Tăng hiệu suất: Nếu client và server nằm trên các máy tính khác nhau, thì các CPU khác nhau có thể xử lý các ứng dụng song song. Việc tối ưu máy chủ cũng sẽ dễ dàng hơn nếu nó chỉ có nhiệm vụ duy nhất là thực hiện xử lý cơ sở dữ liệu.
- Chi phí phần cứng có thể giảm: server chỉ yêu cầu khả năng lưu trữ và xử lý đủ để lưu trữ và quản lý cơ sở dữ liệu.
- Giảm chi phí truyền thông: Các ứng dụng thực hiện một phần hoạt động trên client và chỉ gửi yêu cầu truy cập cơ sở dữ liệu qua mạng, dẫn đến ít dữ liệu được gửi qua mạng hơn.
- Tăng tính nhất quán: Server có thể xử lý các kiểm tra tính toàn vẹn, do đó các ràng buộc được xác định và xác nhận chỉ ở một nơi, thay vì để mỗi chương trình ứng dụng thực hiện kiểm tra riêng của mình.

Một số nhà cung cấp DBMS đã sử dụng kiến trúc này để chỉ ra khả năng của cơ sở dữ liệu phân tán, nghĩa là, một tập hợp nhiều cơ sở dữ liệu liên quan đến nhau về mặt logic được phân phối qua mạng máy tính. Tuy nhiên, mặc dù kiến trúc client-server có thể được sử dụng để cung cấp cơ chế hiện thực cơ sở dữ liệu phân tán, nhưng bản thân nó không cấu thành một DBMS phân tán.

**Bảng 1.3** Tóm tắt các chức năng của client-server

CLIENT	SERVER
Quản lý giao diện người dùng	Chấp nhận và xử lý các yêu cầu cơ sở dữ liệu từ client
Chấp nhận và kiểm tra cú pháp nhập liệu của người dùng	Kiểm tra ủy quyền
Xử lý logic ứng dụng	Đảm bảo các ràng buộc về tính toàn vẹn
Tạo các yêu cầu cơ sở dữ liệu và truyền đến server	Thực hiện xử lý truy vấn/cập nhật và truyền phản hồi đến client
Chuyển phản hồi lại cho người dùng	Duy trì danh mục hệ thống Cung cấp truy cập cơ sở dữ liệu đồng thời Cung cấp điều khiển khôi phục

**1.5.4 Kiến trúc client-server ba tầng**

Nhu cầu về khả năng mở rộng doanh nghiệp đã thách thức mô hình client-server hai tầng truyền thống. Vào giữa những năm 1990, khi các ứng dụng trở nên phức tạp hơn và có khả năng được triển khai cho hàng trăm hoặc hàng nghìn người dùng cuối, phía khách hàng đã đưa ra hai vấn đề phát sinh:

- Một client “béo” yêu cầu tài nguyên đáng kể trên máy tính của khách hàng để chạy hiệu quả. Điều này bao gồm không gian đĩa, RAM và sức mạnh CPU.
- Chi phí quản trị phía client khá đáng kể.

Đến năm 1995, một biến thể mới của mô hình client-server hai tầng truyền thống đã xuất hiện để giải quyết vấn đề về khả năng mở rộng của doanh nghiệp. Kiến trúc mới này đề xuất ba lớp, mỗi lớp có khả năng chạy trên một nền tảng khác nhau:

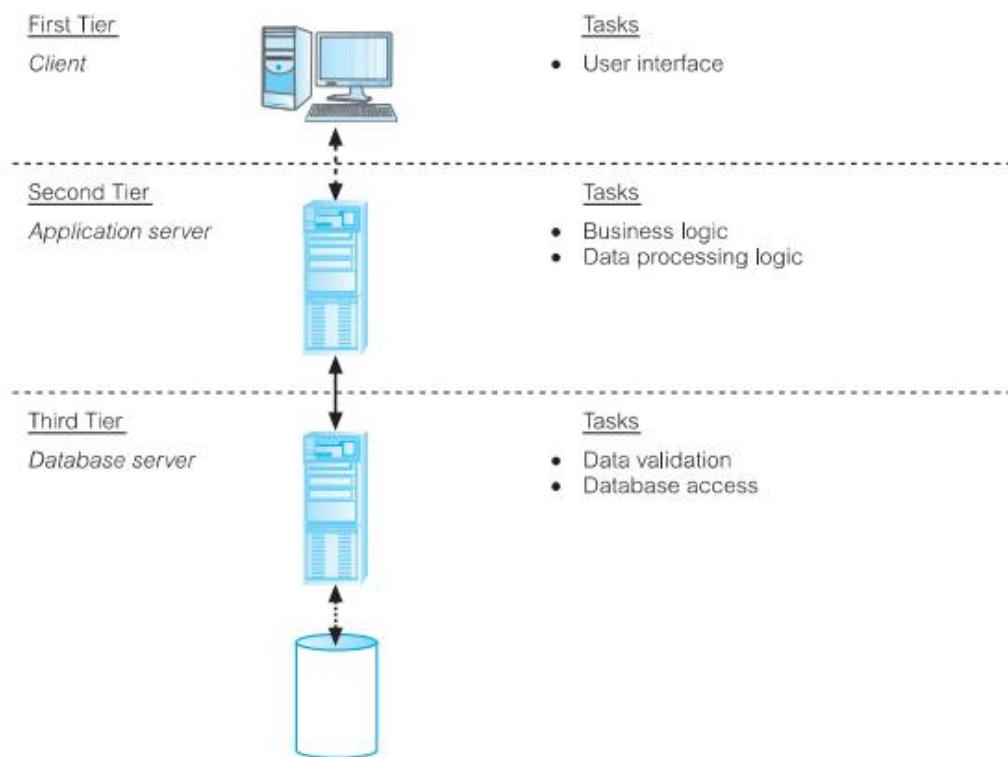
- Lớp giao diện người dùng, chạy trên máy tính của người dùng cuối (client).
- Lớp xử lý dữ liệu và logic nghiệp vụ. Tầng giữa này chạy trên một server và thường được gọi là máy chủ ứng dụng (application server).
- Một DBMS, lưu trữ dữ liệu theo yêu cầu của tầng giữa. Tầng này có thể chạy trên một server riêng biệt được gọi là máy chủ cơ sở dữ liệu (database server).

Trong Hình 1.16, client hiện chỉ chịu trách nhiệm về giao diện người dùng của ứng dụng và có thể thực hiện một số xử lý logic đơn giản, chẳng hạn như xác thực đầu vào, do đó cung cấp một client “gầy”. Logic nghiệp vụ cốt lõi của ứng dụng giờ đây nằm trong lớp riêng, được kết nối vật lý với client và máy chủ cơ sở dữ liệu qua mạng LAN hoặc mạng điện rộng (WAN). Một máy chủ ứng dụng được thiết kế để phục vụ nhiều client.

Thiết kế ba tầng có nhiều ưu điểm hơn so với thiết kế hai tầng/một tầng truyền thống:

- Nhu cầu về phần cứng ít tồn kém hơn vì client “gầy”.
- Việc duy trì ứng dụng được tập trung hóa với việc chuyển logic nghiệp vụ đổi với nhiều người dùng cuối vào một máy chủ ứng dụng duy nhất.
- Mô-đun được bổ sung giúp dễ dàng sửa đổi/thay thế một mức mà không ảnh hưởng đến các mức khác.
- Cân bằng tải dễ dàng hơn với việc tách logic nghiệp vụ cốt lõi khỏi các chức năng cơ sở dữ liệu.

Một ưu điểm nữa là kiến trúc ba tầng rất phù hợp với môi trường Web, với trình duyệt Web đóng vai trò là client “gây” và máy chủ Web hoạt động như máy chủ ứng dụng.



Hình 1.16 Kiến trúc client-server ba tầng

### 1.5.5 Kiến trúc client-server n tầng

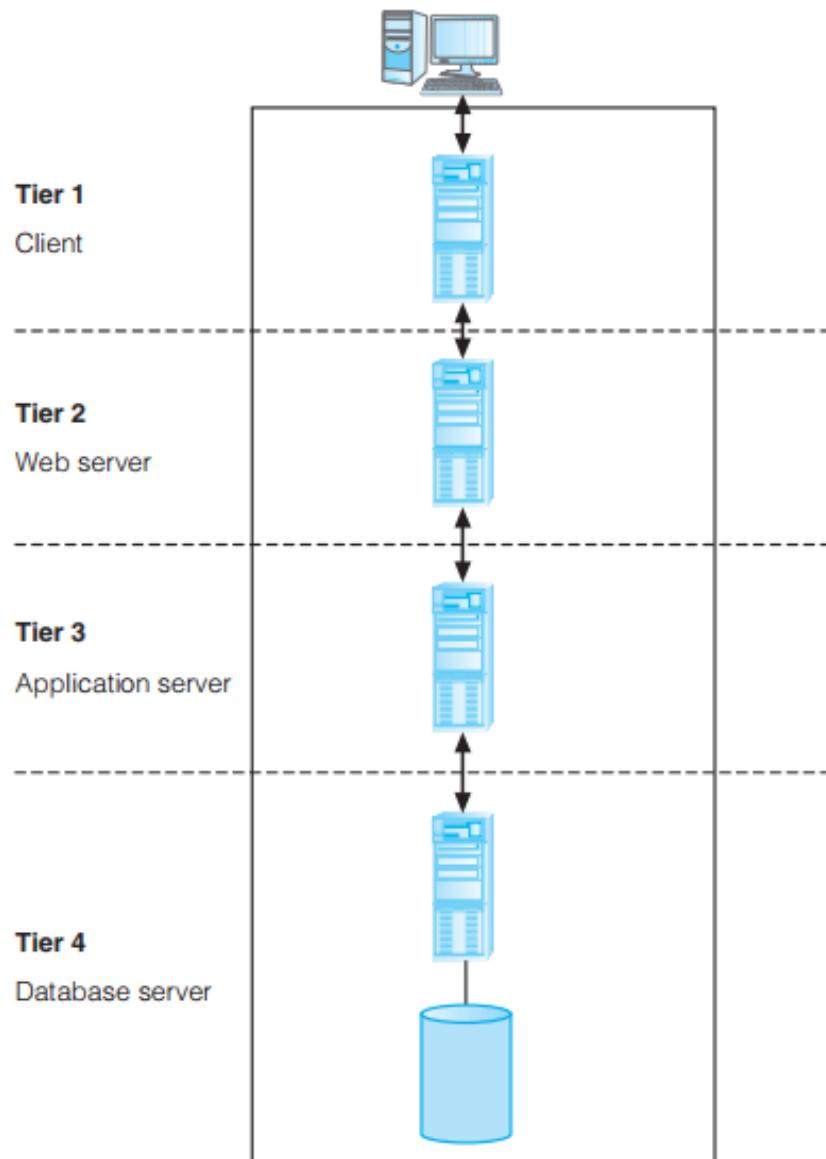
Kiến trúc ba tầng có thể được mở rộng thành n tầng, với các tầng bổ sung cung cấp khả năng mở rộng và linh hoạt hơn. Ví dụ, như được minh họa trong Hình 1.17, tầng giữa của kiến trúc có thể được chia thành hai, một tầng cho máy chủ Web và một tầng khác cho máy chủ ứng dụng.

#### Máy chủ ứng dụng

**Máy chủ ứng dụng** Lưu trữ giao diện lập trình ứng dụng (Application Programming Interface - API), hiển thị logic nghiệp vụ và quy trình nghiệp vụ cho các ứng dụng khác sử dụng.

Máy chủ ứng dụng (application server) phải xử lý một số vấn đề phức tạp:

- Điều khiển đồng thời;
- Quản lý kết nối mạng;
- Cung cấp quyền truy cập vào tất cả các máy chủ cơ sở dữ liệu;
- Lưu trữ tạm thời kết nối cơ sở dữ liệu (connection pooling);
- Hỗ trợ phân cụm;
- Cân bằng tải;
- Sao lưu - phục hồi.



**Hình 1.17** Kiến trúc bốn tầng, tầng giữa được tách thành máy chủ Web và máy chủ ứng dụng. Một số máy chủ ứng dụng phổ biến:

- Java 2 Platform, Enterprise Edition (J2EE), là một đặc tả kỹ thuật cho một nền tảng để lập trình máy chủ bằng ngôn ngữ lập trình Java. Máy chủ ứng dụng J2EE có thể xử lý các giao dịch, bảo mật, khả năng mở rộng, đồng thời và quản lý các thành phần được triển khai cho nó, có nghĩa là các nhà phát triển có thể tập trung nhiều hơn vào logic nghiệp vụ của các thành phần hơn là vào cơ sở hạ tầng và các nhiệm vụ tích hợp.

Một số máy chủ ứng dụng J2EE nổi tiếng là WebLogic Server và Oracle GoldFish Server của Tập đoàn Oracle, JBoss của Red Hat, WebSphere Application Server của IBM và mã nguồn mở Glassfish Application Server.

- .NET Framework là sản phẩm của Microsoft để hỗ trợ sự phát triển của tầng giữa.
- Oracle Application Server cung cấp một tập hợp các dịch vụ để lắp ráp một cơ sở hạ tầng đa năng có thể mở rộng để hỗ trợ kinh doanh điện tử (e-Business).

## TÓM TẮT

- Tiền thân của DBMS là hệ thống dựa trên tập tin, là một tập hợp các chương trình ứng dụng thực hiện các dịch vụ cho người dùng cuối, thường là tạo báo cáo. Mỗi chương trình định nghĩa và quản lý dữ liệu của chính nó. Mặc dù hệ thống dựa trên tập tin là một cải tiến lớn so với hệ thống hồ sơ thủ công, nhưng vẫn có những vấn đề đáng kể, chủ yếu là lượng dữ liệu dư thừa và mối quan hệ phụ thuộc của chương trình - dữ liệu.
- Cách tiếp cận hướng cơ sở dữ liệu xuất hiện để giải quyết các vấn đề với cách tiếp cận dựa trên tập tin. Cơ sở dữ liệu là một tập hợp có thể chia sẻ được của các dữ liệu có liên quan logic và mô tả về tập hợp này, được thiết kế để đáp ứng nhu cầu thông tin của một tổ chức. DBMS là một hệ thống phần mềm cho phép người dùng định nghĩa, tạo, duy trì và kiểm soát quyền truy cập vào cơ sở dữ liệu. Chương trình ứng dụng là một chương trình máy tính tương tác với cơ sở dữ liệu bằng cách đưa ra một yêu cầu thích hợp (thường là một câu lệnh SQL) tới DBMS. Thuật ngữ Hệ cơ sở dữ liệu bao hàm hơn được sử dụng để xác định một tập hợp các chương trình ứng dụng tương tác với cơ sở dữ liệu cùng với DBMS và chính cơ sở dữ liệu.
- Tất cả các truy cập vào cơ sở dữ liệu đều thông qua DBMS. DBMS cung cấp Ngôn ngữ Định nghĩa dữ liệu (Data Definition Language - DDL), cho phép người dùng định nghĩa cơ sở dữ liệu và Ngôn ngữ Thao tác dữ liệu (Data Manipulation Language - DML), cho phép người dùng thêm, cập nhật, xóa và truy xuất dữ liệu.
- DBMS cung cấp quyền truy cập có kiểm soát vào cơ sở dữ liệu, đảm bảo khả năng kiểm soát bảo mật, tính toàn vẹn, đồng thời và khôi phục cũng như danh mục người dùng có thể truy cập.
- Môi trường DBMS bao gồm phần cứng (máy tính), phần mềm (DBMS, hệ điều hành và các chương trình ứng dụng), dữ liệu, thủ tục và con người. Yếu tố con người bao gồm quản trị viên dữ liệu, quản trị viên cơ sở dữ liệu, chuyên viên thiết kế cơ sở dữ liệu, nhà phát triển ứng dụng và người dùng cuối.
- Một số ưu điểm của cách tiếp cận hướng cơ sở dữ liệu bao gồm kiểm soát dư thừa dữ liệu, tính nhất quán của dữ liệu, chia sẻ dữ liệu, cải thiện tính bảo mật và tính toàn vẹn. Một số hạn chế bao gồm sự phức tạp, chi phí, giảm hiệu suất và tác động lớn hơn khi hệ thống phát sinh lỗi.
- Kiến trúc cơ sở dữ liệu ANSI-SPARC sử dụng ba mức trừu tượng: bên ngoài, khái niệm và bên trong. Mức bên ngoài bao gồm các khung nhìn cơ sở dữ liệu của người dùng. Mức khái niệm là khung nhìn chung của cơ sở dữ liệu: đặc tả nội dung thông tin của toàn bộ cơ sở dữ liệu, độc lập với các phương thức lưu trữ. Mức khái niệm biểu diễn tất cả các thực thể, các thuộc tính và mối quan hệ giữa chúng, cũng như các ràng buộc đối với dữ liệu và thông tin bảo mật và toàn vẹn.

Mức bên trong là khung nhìn cơ sở dữ liệu của máy tính: mức này chỉ định cách dữ liệu được biểu diễn, cách các bản ghi được sắp xếp theo trình tự, chỉ mục và các con trỏ,...

- Ánh xạ bên ngoài/khai niệm chuyển đổi các yêu cầu và kết quả giữa mức bên ngoài và mức khai niệm. Ánh xạ khai niệm/bên trong chuyển đổi các yêu cầu và kết quả giữa mức khai niệm và mức bên trong.
- Một lược đồ cơ sở dữ liệu là một mô tả về cấu trúc cơ sở dữ liệu. Có ba loại lược đồ khác nhau trong cơ sở dữ liệu; được xác định theo ba mức của kiến trúc ANSI-SPARC. Tính độc lập về dữ liệu làm cho mỗi mức không bị ảnh hưởng với những thay đổi của các mức thấp hơn. Tính độc lập dữ liệu logic để cập đến khả năng không bị ảnh hưởng của các lược đồ bên ngoài đối với các thay đổi trong lược đồ khai niệm. Tính độc lập dữ liệu vật lý để cập đến khả năng không bị ảnh hưởng của lược đồ khai niệm đối với những thay đổi trong lược đồ bên trong.
- Một ngôn ngữ dữ liệu bao gồm hai phần: Ngôn ngữ Định nghĩa dữ liệu (DDL) và Ngôn ngữ Thao tác dữ liệu (DML). DDL được sử dụng để đặc tả lược đồ cơ sở dữ liệu và DML được sử dụng để đọc và cập nhật cơ sở dữ liệu. Một phần của DML liên quan đến việc truy xuất dữ liệu được gọi là Ngôn ngữ truy vấn (query language).
- Các chức năng và dịch vụ của DBMS đa người dùng bao gồm lưu trữ, truy xuất và cập nhật dữ liệu; một danh mục người dùng có thể truy cập; hỗ trợ giao dịch; dịch vụ điều khiển đồng thời và phục hồi; dịch vụ ủy quyền; hỗ trợ giao tiếp dữ liệu; dịch vụ giám sát; dịch vụ thúc đẩy tính độc lập của dữ liệu; và các dịch vụ tiện ích.
- Kiến trúc client-server để cập đến cách thức mà các thành phần phần mềm tương tác. Một client yêu cầu một số tài nguyên và một server cung cấp tài nguyên đó. Trong mô hình hai tầng, client xử lý giao diện người dùng và logic xử lý nghiệp vụ và server xử lý chức năng cơ sở dữ liệu. Trong môi trường Web, mô hình hai tầng truyền thống đã được thay thế bằng mô hình ba tầng, bao gồm tầng giao diện người dùng (client), tầng xử lý dữ liệu và logic nghiệp vụ (application server) và DBMS (database server), được phân tán trên các máy khác nhau.
- Kiến trúc ba tầng có thể được mở rộng đến n tầng, các tầng bổ sung sẽ được thêm vào nhằm cung cấp tính linh hoạt và khả năng mở rộng hơn.

## CÂU HỎI

1. Trình bày về các thuật ngữ sau:
  - dữ liệu (data)
  - cơ sở dữ liệu (database)
  - hệ quản trị cơ sở dữ liệu (DBMS)
  - chương trình ứng dụng cơ sở dữ liệu (database application)
  - độc lập dữ liệu (data independence)
  - bảo mật (security)
  - tính toàn vẹn (integrity)
2. Mô tả vai trò của hệ quản trị cơ sở dữ liệu (DBMS) trong cách tiếp cận hướng cơ sở dữ liệu.
3. Mô tả năm thành phần của môi trường DBMS và thảo luận về cách chúng liên quan với nhau.
4. Thảo luận về các vai trò của con người trong môi trường cơ sở dữ liệu:
  - quản trị viên dữ liệu (data administrator)
  - quản trị viên cơ sở dữ liệu (database administrator)
  - chuyên viên thiết kế cơ sở dữ liệu logic (logical database designer)
  - chuyên viên thiết kế cơ sở dữ liệu vật lý (physical database designer)
  - nhà phát triển ứng dụng (application developer)
  - người dùng cuối (end-user)
5. Giải thích khái niệm lược đồ cơ sở dữ liệu và trình bày về ba loại lược đồ trong cơ sở dữ liệu.
6. Mô tả các dịch vụ, chức năng được mong đợi được từ một DBMS đa người dùng.
7. Trình bày về chức năng và tầm quan trọng của danh mục hệ thống.
8. Điều khiển đồng thời là gì và tại sao một DBMS cần một phương tiện điều khiển đồng thời?
9. Thuật ngữ kiến trúc client-server có nghĩa là gì và những ưu điểm của cách tiếp cận này là gì? So sánh kiến trúc client-server với hai kiến trúc khác.
10. So sánh và đối chiếu kiến trúc client-server hai tầng cho các DBMS truyền thống với kiến trúc client-server ba tầng. Tại sao kiến trúc thứ hai thích hợp hơn cho Web?

## CHƯƠNG 2

### CÁC BIỆN PHÁP BẢO MẬT CƠ SỞ DỮ LIỆU

Trong chương này, chúng ta sẽ tìm hiểu:

- Phạm vi của bảo mật cơ sở dữ liệu.
- Khái niệm về mối đe dọa.
- Các mối đe dọa có thể ảnh hưởng đến hệ thống cơ sở dữ liệu.
- Các kiểm soát bảo mật dựa trên máy tính.
- Nhu cầu sao lưu và phục hồi cơ sở dữ liệu
- Các phương tiện và chiến lược phục hồi

Dữ liệu là một nguồn tài nguyên quý giá cần phải được kiểm soát và quản lý chặt chẽ, giống như bất kỳ nguồn lực nào của tổ chức. Một phần hoặc toàn bộ dữ liệu của tổ chức có thể có tầm quan trọng chiến lược và do đó cần được giữ an toàn và bảo mật.

Trong Chương 1, chúng ta đã tìm hiểu về môi trường hệ quản trị cơ sở dữ liệu, đặc biệt là các chức năng và dịch vụ điển hình của một hệ quản trị cơ sở dữ liệu tiêu biểu. Các chức năng và dịch vụ này bao gồm dịch vụ ủy quyền, nghĩa là DBMS phải cung cấp một cơ chế để đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập vào cơ sở dữ liệu. Nói cách khác, DBMS phải đảm bảo rằng cơ sở dữ liệu được bảo mật. Thuật ngữ bảo mật đề cập đến việc bảo vệ cơ sở dữ liệu chống lại sự truy cập trái phép, dù là cố ý hay vô tình.

## 2.1 BẢO MẬT CƠ SỞ DỮ LIỆU VÀ CÁC MỐI ĐE DỌA

Trong phần này, chúng ta mô tả phạm vi của bảo mật cơ sở dữ liệu và trình bày lý do các tổ chức phải nghiêm túc xem xét các mối đe dọa tiềm ẩn đối với hệ thống máy tính của họ. Chúng ta cũng xác định phạm vi của các mối đe dọa và hậu quả của chúng trên hệ thống máy tính.

### Bảo mật cơ sở dữ liệu

Các cơ chế bảo vệ cơ sở dữ liệu khỏi các mối đe dọa cố ý hoặc vô ý.

Các cân nhắc về bảo mật không chỉ áp dụng cho dữ liệu được lưu giữ trong cơ sở dữ liệu: vi phạm bảo mật có thể ảnh hưởng đến các phần khác của hệ thống, do đó, có thể ảnh hưởng đến cơ sở dữ liệu. Vì thế, bảo mật cơ sở dữ liệu bao gồm cả các yếu tố phần cứng, phần mềm, con người và dữ liệu. Để thực hiện hiệu quả bảo mật cần có các biện pháp kiểm soát thích hợp, được xác định trong các mục tiêu nhiệm vụ cụ thể của hệ thống. Nhu cầu bảo mật này, mặc dù trước đây thường bị bỏ qua hoặc bị xem nhẹ, nhưng ngày nay các tổ chức ngày càng nhận thức nhiều hơn.

Cơ sở dữ liệu đại diện cho một nguồn lực thiết yếu của tổ chức cần được bảo mật thích hợp bằng cách sử dụng các biện pháp kiểm soát thích hợp. Chúng ta sẽ xem xét bảo mật cơ sở dữ liệu liên quan đến các tình huống sau:

- Trộm cắp và gian lận;
- Mất tính bảo mật;
- Mất sự riêng tư;
- Mất tính toàn vẹn;
- Mất khả năng sẵn sàng.

Những tình huống này đại diện cho các lĩnh vực mà tổ chức cần tìm cách giảm thiểu rủi ro, tức là khả năng xảy ra mất mát hoặc thiệt hại. Trong một số tình huống, các lĩnh vực này có liên quan chặt chẽ với nhau đến mức một hoạt động dẫn đến tổn thất ở lĩnh vực này cũng có thể dẫn đến tổn thất ở lĩnh vực khác. Ngoài ra, các vấn đề như gian lận hoặc mất sự riêng tư có thể phát sinh do hành vi cố ý hoặc vô ý và không nhất thiết dẫn đến bất kỳ thay đổi nào có thể phát hiện được đối với cơ sở dữ liệu hoặc hệ thống máy tính.

Trộm cắp và gian lận không chỉ ảnh hưởng đến môi trường cơ sở dữ liệu mà còn ảnh hưởng đến toàn bộ tổ chức bởi chính những người thực hiện các hoạt động đó, vì thế, nên chú ý đến việc giảm thiểu các cơ hội cho việc này xảy ra. Trộm cắp và gian lận không nhất thiết phải thay đổi dữ liệu, như trường hợp của các hoạt động dẫn đến mất bảo mật hoặc mất sự riêng tư.

Tính bảo mật đề cập đến nhu cầu duy trì tính bí mật của dữ liệu - thường chỉ là dữ liệu quan trọng đối với tổ chức - trong khi sự riêng tư đề cập đến nhu cầu bảo vệ dữ liệu về cá nhân. Ví dụ, vi phạm bảo mật dẫn đến mất tính bí mật có thể dẫn đến mất khả năng cạnh tranh và mất sự riêng tư có thể dẫn đến hành động pháp lý chống lại tổ chức.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Mất tính toàn vẹn của dữ liệu dẫn đến dữ liệu không hợp lệ hoặc bị hỏng, có thể ảnh hưởng nghiêm trọng đến hoạt động của một tổ chức. Nhiều tổ chức hiện đang hướng đến phong cách hoạt động liên tục, gọi là tính khả dụng 24/7. Mất khả năng sẵn sàng có nghĩa là người dùng không thể truy cập dữ liệu hoặc hệ thống hoặc cả hai, điều này có thể ảnh hưởng nghiêm trọng đến hoạt động tài chính của tổ chức. Trong một số trường hợp, các sự kiện khiến hệ thống không khả dụng cũng có thể gây ra hỏng dữ liệu. Bảo mật cơ sở dữ liệu nhằm giảm thiểu tổn thất do các vấn đề có thể dự đoán được gây ra theo cách tiết kiệm chi phí mà không ảnh hưởng đến người dùng.

**Mối đe dọa** Bất kỳ tình huống hoặc sự kiện nào, dù là cố ý hay ngẫu nhiên, có thể ảnh hưởng xấu đến hệ thống và tổ chức.

Mối đe dọa (threat) có thể được gây ra bởi một tình huống hoặc sự kiện liên quan đến một người, một hành động hoặc một hoàn cảnh có khả năng gây tổn hại cho tổ chức. Tác hại có thể là hữu hình, chẳng hạn như mất phần cứng, phần mềm hoặc dữ liệu, hoặc vô hình, chẳng hạn như mất uy tín hoặc niềm tin của khách hàng. Vấn đề mà bất kỳ tổ chức nào cũng phải đối mặt là xác định tất cả các mối đe dọa có thể xảy ra. Do đó, ở mức tối thiểu, một tổ chức nên đầu tư thời gian và công sức vào việc xác định các mối đe dọa nghiêm trọng nhất.

Trong phần trước, chúng ta đã xác định các lĩnh vực mất mát có thể do các hoạt động cố ý hoặc không chủ ý. Mặc dù một số loại mối đe dọa có thể là cố ý hoặc vô ý, tác động vẫn như nhau. Các mối đe dọa có ý liên quan đến yếu tố con người và có thể được thực hiện bởi cả người dùng được ủy quyền và người dùng trái phép, một số người trong số họ có thể là bên ngoài tổ chức.

Bất kỳ mối đe dọa nào cũng phải được xem là một hành vi vi phạm an ninh tiềm ẩn mà nếu thành công, sẽ có tác động nhất định. Bảng 2.1 trình bày các ví dụ về các loại mối đe dọa khác nhau, được liệt kê trong lĩnh vực mà chúng có thể có tác động. Ví dụ: “xem và tiết lộ dữ liệu trái phép” như một mối đe dọa có thể dẫn đến hành vi trộm cắp và gian lận, mất tính bảo mật và mất quyền riêng tư cho tổ chức.

**Bảng 2.1** Ví dụ về các mối đe dọa

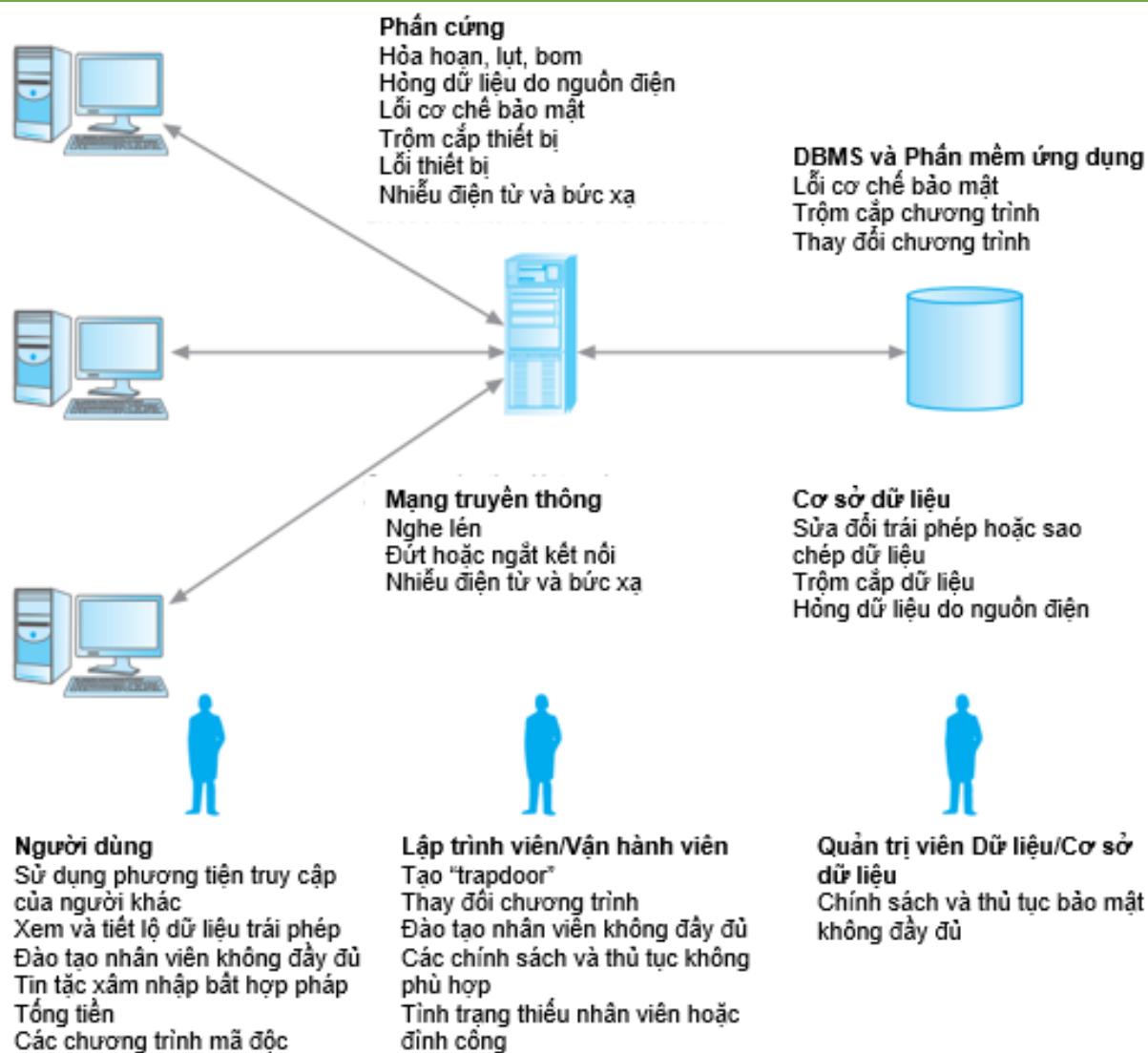
MỐI ĐE DỌA	TRỘM CẤP VÀ GIAN LẬN	MẤT TÍNH BẢO MẬT	MẤT TÍNH RIÊNG TƯ	MẤT TÍNH TOÀN VẸN	MẤT TÍNH SẴN SÀNG
Sử dụng phương tiện truy cập của người khác	✓	✓	✓		
Sửa đổi hoặc sao chép dữ liệu trái phép	✓			✓	
Thay đổi chương trình	✓			✓	✓
Các chính sách và thủ tục không phù hợp dẫn đến dữ liệu đầu ra bí mật và thông thường khó phân tách	✓	✓	✓		
Nghe lén	✓	✓	✓		

## Giáo trình Hệ quản trị Cơ sở dữ liệu

MỐI ĐE DỌA	TRỘM CẤP VÀ GIAN LẬN	MẮT TÍNH BẢO MẬT	MẮT TÍNH RIÊNG TƯ	MẮT TÍNH TOÀN VẸN	MẮT TÍNH SẴN SÀNG
Tin tặc xâm nhập bất hợp pháp	✓	✓	✓		
Tống tiền	✓	✓	✓		
Tạo "trapdoor" vào hệ thống	✓	✓	✓		
Ăn cắp dữ liệu, chương trình và thiết bị	✓	✓	✓		✓
Lỗi cơ chế bảo mật, cấp quyền truy cập cao hơn bình thường		✓	✓	✓	
Tình trạng thiếu nhân viên hoặc đình công				✓	✓
Đào tạo nhân viên không đầy đủ		✓	✓	✓	✓
Xem và tiết lộ dữ liệu trái phép	✓	✓	✓		
Nhiều điện tử và bức xạ				✓	✓
Dữ liệu bị hỏng do mất điện hoặc tăng đột biến				✓	✓
Hỏa hoạn (lỗi nguồn điện, sét đánh, đốt phá), lũ lụt, bom				✓	✓
Tồn hại vật chất đối với thiết bị				✓	✓
Đứt/ngắt cáp kết nối				✓	✓
Các chương trình mã độc				✓	✓

Mức độ thiệt hại mà một tổ chức phải gánh chịu do mối đe dọa gây ra phụ thuộc vào một số yếu tố, chẳng hạn như sự tồn tại của các biện pháp đối phó và kế hoạch dự phòng. Ví dụ, nếu lỗi phần cứng xảy ra làm hỏng bộ nhớ thứ cấp, tất cả hoạt động xử lý phải dừng cho đến khi sự cố được giải quyết. Việc khôi phục sẽ phụ thuộc vào một số yếu tố, bao gồm thời điểm thực hiện các bản sao lưu cuối cùng và thời gian cần thiết để khôi phục hệ thống.

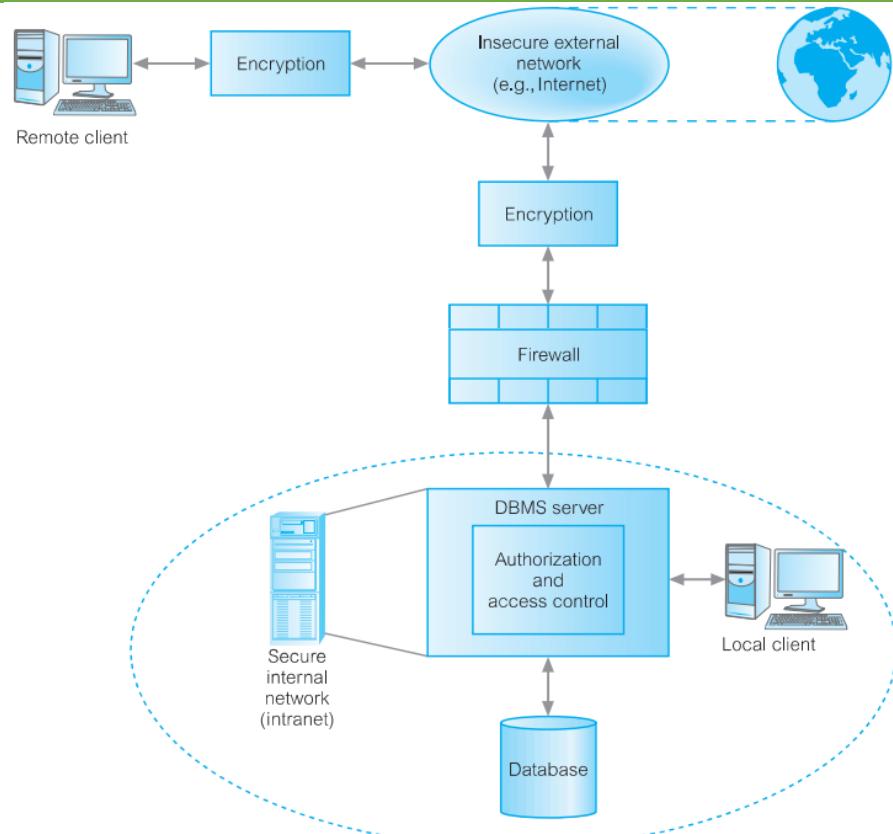
Các chuyên viên của tổ chức cần xác định các loại mối đe dọa có thể phải đối mặt và bắt đầu các kế hoạch, biện pháp đối phó thích hợp, và cũng lưu ý đến chi phí hiện thực giải pháp. Rõ ràng, việc dành thời gian, công sức và tiền bạc đáng kể cho các mối đe dọa tiềm ẩn có thể dẫn đến bất tiện nhỏ như tốn kém về mặt chi phí. Hoạt động kinh doanh của tổ chức cũng có thể ảnh hưởng đến các loại mối đe dọa cần được xem xét, một số loại có thể hiếm. Tuy nhiên, mặc dù “hiếm” cũng cần phải được tính đến, đặc biệt nếu tác động của các mối đe dọa này là đáng kể.



**Hình 2.1** Tóm tắt các mối đe dọa tiềm ẩn đối với hệ thống máy tính

## 2.2 CÁC BIỆN PHÁP BẢO MẬT

Các loại biện pháp đối phó với các mối đe dọa trên hệ thống máy tính bao gồm từ kiểm soát vật lý đến thủ tục hành chính. Mặc dù có sẵn nhiều kiểu kiểm soát dựa trên máy tính, nhưng cần lưu ý rằng, tính bảo mật của DBMS chỉ tốt như của hệ điều hành, do sự liên kết chặt chẽ của chúng. Hình 2.2 biểu diễn một môi trường máy tính đa người dùng điển hình.

**Hình 2.2** Môi trường máy tính đa người dùng tiêu biểu

Trong phần này, chúng ta tập trung vào các biện pháp kiểm soát bảo mật dựa trên máy tính sau đây cho môi trường đa người dùng (một số biện pháp có thể không khả dụng trong môi trường máy tính độc lập):

- Ủy quyền
- Điều khiển truy cập
- Khung nhìn
- Sao lưu và phục hồi
- Tính toàn vẹn
- Mã hóa
- Công nghệ RAID

### 2.2.1 Ủy quyền

**Ủy quyền** Việc cấp quyền hoặc đặc quyền cho phép chủ thẻ có quyền truy cập hợp pháp vào hệ thống hoặc đối tượng của hệ thống.

Kiểm soát ủy quyền có thể được tích hợp trong phần mềm và không chỉ chi phối hệ thống hoặc đối tượng mà một người dùng được chỉ định có thể truy cập, mà còn cả những gì người dùng có thể làm với nó. Quá trình ủy quyền liên quan đến việc xác thực các chủ thẻ yêu cầu quyền truy cập vào các đối tượng, trong đó “chủ thẻ” đại diện cho người dùng hoặc chương trình và “đối tượng” đại diện cho bảng cơ sở dữ liệu, khung nhìn, thủ tục, trình kích hoạt hoặc bất kỳ đối tượng nào khác có thể được tạo trong hệ thống.

<b>Xác thực</b>	Một cơ chế để xác định xem người dùng có phải là người mà họ tuyên bố hay không.
	Quản trị viên hệ thống thường chịu trách nhiệm cho phép người dùng có quyền truy cập vào hệ thống máy tính bằng cách tạo tài khoản người dùng cá nhân. Mỗi người dùng được cấp một định danh duy nhất, được hệ điều hành sử dụng để xác định họ là ai. Được liên kết với mỗi định danh là một mật khẩu, do người dùng chọn và hệ điều hành biết, mật khẩu này phải được cung cấp để cho phép hệ điều hành xác minh (hoặc xác thực - authentication) người dùng là ai.

Quy trình này cho phép sử dụng hệ thống máy tính nhưng không nhất thiết cho phép truy cập vào DBMS hoặc bất kỳ chương trình ứng dụng liên quan nào. Có thể phải thực hiện một thủ tục tương tự, riêng biệt để cấp cho người dùng quyền sử dụng DBMS. Trách nhiệm cho phép sử dụng DBMS thường thuộc về Quản trị viên cơ sở dữ liệu (DBA), người này cũng phải thiết lập các tài khoản người dùng và mật khẩu cá nhân bằng chính DBMS.

Một số DBMS duy trì danh sách các định danh người dùng hợp lệ và mật khẩu được liên kết, có thể khác biệt với danh sách của hệ điều hành. Tuy nhiên, các DBMS khác duy trì một danh sách có các mục được xác thực dựa trên danh sách của hệ điều hành. Điều này ngăn người dùng đăng nhập vào DBMS bằng một tên khác với tên đã đăng nhập vào hệ điều hành.

### 2.2.2 Điều khiển truy cập

Cách điển hình để cung cấp các điều khiển truy cập cho hệ thống cơ sở dữ liệu là dựa trên việc cấp và thu hồi các đặc quyền. Đặc quyền (privilege) cho phép người dùng tạo hoặc truy cập (đọc, ghi hoặc sửa đổi) một số đối tượng cơ sở dữ liệu (chẳng hạn như quan hệ/bảng, khung nhìn hoặc chỉ mục) hoặc thực thi các tiện ích DBMS nhất định. Vì việc cấp quá nhiều đặc quyền không cần thiết có thể ảnh hưởng đến bảo mật, nên một đặc quyền chỉ nên được cấp cho người dùng nếu người dùng đó không thể hoàn thành công việc nếu không có đặc quyền đó. Người dùng tạo một đối tượng cơ sở dữ liệu chẳng hạn như một quan hệ hoặc một khung nhìn sẽ tự động nhận được tất cả các đặc quyền trên đối tượng đó. Sau đó, DBMS theo dõi cách các đặc quyền này được cấp cho những người dùng khác và có thể bị thu hồi, đồng thời đảm bảo rằng chỉ những người dùng có các đặc quyền cần thiết mới có thể truy cập vào một đối tượng.

#### Điều khiển truy cập tùy quyền (Discretionary Access Control - DAC)

Hầu hết các DBMS thương mại cung cấp một cách tiếp cận để quản lý các đặc quyền thông qua sử dụng SQL được gọi là Điều khiển truy cập tùy quyền (Discretionary Access Control - DAC). SQL chuẩn hỗ trợ DAC thông qua các lệnh GRANT và REVOKE. Lệnh GRANT cấp đặc quyền cho người dùng và lệnh REVOKE thu hồi các đặc quyền. Điều khiển truy cập tùy quyền, mặc dù hiệu quả, nhưng có một số điểm yếu nhất định. Đặc biệt, người dùng trái phép có thể lừa người dùng được ủy quyền tiết lộ dữ liệu nhạy cảm.

Rõ ràng, cần có một phương pháp bảo mật bổ sung để loại bỏ những sơ hở như vậy và yêu cầu này được đáp ứng trong một phương pháp được gọi là Điều khiển truy cập bắt buộc (Mandatory Access Control - MAC), mà chúng ta sẽ thảo luận chi tiết tiếp theo. Mặc dù DAC thường được cung cấp bởi hầu hết các DBMS thương mại, nhưng một số khác cũng cung cấp hỗ trợ cho MAC.

### **Điều khiển truy cập bắt buộc (Mandatory Access Control - MAC)**

Điều khiển truy cập bắt buộc (Mandatory Access Control - MAC) dựa trên các chính sách trên toàn hệ thống mà người dùng cá nhân không thể thay đổi. Trong cách tiếp cận này, mỗi đối tượng cơ sở dữ liệu được gán một lớp bảo mật và mỗi người dùng được chỉ định một cấp cho một lớp bảo mật, và các quy tắc được áp đặt đối với việc đọc và ghi các đối tượng cơ sở dữ liệu bởi người dùng. DBMS xác định liệu một người dùng nhất định có thể đọc hoặc ghi một đối tượng nhất định hay không dựa trên các quy tắc nhất định liên quan đến mức độ bảo mật của đối tượng và cấp của người dùng. Các quy tắc này nhằm đảm bảo rằng dữ liệu nhạy cảm không bao giờ có thể được chuyển cho người dùng khác ở cấp không cần thiết. SQL chuẩn không bao gồm hỗ trợ cho MAC.

Mặc dù MAC giải quyết được một điểm yếu chính của DAC, nhưng một nhược điểm lớn là sự cứng nhắc của môi trường MAC. Ví dụ, các chính sách MAC thường được thiết lập bởi quản trị viên hệ thống hoặc cơ sở dữ liệu, và các cơ chế phân loại đôi khi được coi là không linh hoạt.

#### **2.2.3 Khung nhìn**

Người dùng cuối không quá quan tâm đến mức độ phức tạp hay dễ dàng của một tác vụ đối với hệ thống, nên có thể cho rằng DBMS đã làm cho mọi thứ trở nên phức tạp hơn, bởi vì giờ đây họ thấy nhiều dữ liệu hơn mức họ thực sự cần hoặc muốn. Ví dụ, các chi tiết mà Bộ phận Contracts muốn xem đối với một bất động sản cho thuê đã thay đổi trong cách tiếp cận cơ sở dữ liệu, thể hiện trong Hình 1.2. Hiện giờ cơ sở dữ liệu cũng chứa loại bất động sản, số lượng phòng và thông tin chi tiết về chủ sở hữu. Để khắc phục vấn đề này, DBMS cung cấp một cơ chế được gọi là khung nhìn (view), cho phép mỗi loại người dùng có khung nhìn riêng của họ về cơ sở dữ liệu (về bản chất, một khung nhìn là một tập hợp con của cơ sở dữ liệu). Chẳng hạn, chúng ta có thể thiết lập một khung nhìn cho phép Bộ phận Contracts chỉ xem được dữ liệu mà họ muốn xem hoặc được cho phép xem đối với các bất động sản cho thuê.

Ngoài việc giảm độ phức tạp bằng cách cho phép người dùng xem dữ liệu theo cách họ muốn, các khung nhìn có một số lợi ích khác:

- Khung nhìn cung cấp một mức độ bảo mật. Khung nhìn có thể được thiết lập để loại trừ dữ liệu mà một số người dùng không nên xem. Ví dụ, chúng ta có thể tạo khung nhìn cho phép giám đốc chi nhánh và Bộ phận Payroll xem tất cả dữ liệu nhân viên, bao gồm chi tiết tiền lương và tạo khung nhìn thứ hai mà các nhân viên khác sẽ sử dụng mà sẽ không bao gồm chi tiết tiền lương.

- Khung nhìn cung cấp một cơ chế để tùy chỉnh sự xuất hiện của cơ sở dữ liệu. Ví dụ, Bộ phận Contracts có thể muốn gọi trường rent (tiền thuê nhà) bằng cái tên rõ ràng hơn, Monthly Rent (Tiền thuê nhà hàng tháng).
- Một khung nhìn có thể trình bày một bức tranh nhất quán, không thay đổi về cấu trúc của cơ sở dữ liệu, ngay cả khi cơ sở dữ liệu bên dưới bị thay đổi (ví dụ, các trường được thêm vào hoặc bị xóa, các mối quan hệ đã thay đổi, các tập tin bị phân tách, cấu trúc lại hoặc đổi tên). Nếu các trường được thêm vào hoặc xóa khỏi tập tin và các trường này không được khung nhìn yêu cầu, thì khung nhìn không bị ảnh hưởng bởi thay đổi này. Do đó, một khung nhìn giúp cung cấp tính độc lập của chương trình-dữ liệu mà chúng ta đã đề cập trước đây.

Cơ chế khung nhìn cung cấp một cơ chế bảo mật mạnh mẽ và linh hoạt bằng cách ẩn các phần của cơ sở dữ liệu khỏi một số người dùng nhất định. Người dùng không biết về sự tồn tại của bất kỳ thuộc tính hoặc hàng nào bị thiếu trong khung nhìn. Một khung nhìn có thể được định nghĩa trên một số quan hệ với người dùng được cấp đặc quyền thích hợp để sử dụng nó, nhưng không được sử dụng các quan hệ cơ sở. Theo cách này, việc sử dụng một khung nhìn sẽ hạn chế hơn so với việc cấp một số đặc quyền nhất định cho người dùng trên (các) quan hệ cơ sở.

#### 2.2.4 Sao lưu và Phục hồi

##### Sao lưu

Quá trình sao chép định kỳ cơ sở dữ liệu và tập tin nhật ký (và có thể cả các chương trình) sang phương tiện lưu trữ ngoại tuyến.

Một DBMS phải cung cấp các phương tiện dự phòng để hỗ trợ việc khôi phục cơ sở dữ liệu sau khi bị lỗi. Chúng ta nên tạo các bản sao lưu của cơ sở dữ liệu và tập tin nhật ký theo định kỳ và đảm bảo rằng các bản sao được lưu ở một nơi an toàn. Trong trường hợp xảy ra sự cố khiến cơ sở dữ liệu không thể sử dụng được, bản sao lưu và các chi tiết được ghi lại trong tập tin nhật ký được sử dụng để khôi phục cơ sở dữ liệu về trạng thái nhất quán mới nhất có thể.

##### Ghi nhật ký

Quá trình lưu giữ và duy trì một tập tin nhật ký (log file) của tất cả các thay đổi được thực hiện đối với cơ sở dữ liệu để cho phép khôi phục được thực hiện một cách hiệu quả trong trường hợp xảy ra lỗi.

Một DBMS nên cung cấp các phương tiện ghi nhật ký (logging), đôi khi được gọi là journaling, theo dõi trạng thái hiện tại của các giao dịch và các thay đổi cơ sở dữ liệu, để cung cấp hỗ trợ cho các thủ tục khôi phục.

Ưu điểm của ghi nhật ký là trong trường hợp bị lỗi, cơ sở dữ liệu có thể được phục hồi về trạng thái nhất quán đã biết cuối cùng bằng cách sử dụng bản sao lưu của cơ sở dữ liệu và thông tin có trong tập tin nhật ký (log file). Nếu tính năng ghi nhật ký không được bật trên một hệ thống bị lỗi, thì cách duy nhất để khôi phục cơ sở dữ liệu là sử dụng phiên bản cơ sở dữ liệu được sao lưu gần nhất. Tuy nhiên, nếu không có tập tin nhật ký, bất kỳ thay đổi nào được thực hiện sau lần sao lưu cuối cùng của cơ sở dữ liệu sẽ bị mất.

## 2.2.5 Tính toàn vẹn

Các ràng buộc về tính toàn vẹn cũng góp phần duy trì một hệ thống cơ sở dữ liệu an toàn bằng cách ngăn không cho dữ liệu trở nên không hợp lệ và do đó đưa ra các kết quả sai lệch hoặc không chính xác. Các ràng buộc về tính toàn vẹn sẽ được thảo luận chi tiết trong mục 3.1.8 và 3.1.9.

## 2.2.6 Mã hóa

### Mã hóa

Việc biến đổi dữ liệu bằng một thuật toán đặc biệt khiến dữ liệu không thể đọc được bởi bất kỳ chương trình nào nếu không có khóa giải mã.

Nếu một hệ cơ sở dữ liệu lưu giữ dữ liệu đặc biệt nhạy cảm, thì có thể cần phải mã hóa nó như một biện pháp phòng ngừa trước các mối đe dọa từ bên ngoài có thể xảy ra hoặc các nỗ lực truy cập vào nó. DBMS có thể truy cập dữ liệu (sau khi giải mã), mặc dù có sự suy giảm hiệu suất do tốn nhiều thời gian để giải mã. Mã hóa cũng bảo vệ dữ liệu được truyền qua đường truyền thông. Có một số kỹ thuật mã hóa dữ liệu để che giấu thông tin; một số được gọi là "không thể đảo ngược - irreversible" và một số khác gọi là "có thể đảo ngược - reversible". Các kỹ thuật không thể đảo ngược, như tên của nó, không cho phép được biết dữ liệu gốc. Tuy nhiên, dữ liệu có thể được sử dụng để có được thông tin thống kê hợp lệ. Kỹ thuật có thể đảo ngược được sử dụng phổ biến hơn. Để truyền dữ liệu một cách an toàn qua các mạng không an toàn, yêu cầu sử dụng một hệ thống mật mã, bao gồm:

- Một khóa mã hóa (encryption key) để mã hóa dữ liệu (bản rõ - plaintext);
- Một thuật toán mã hóa (encryption algorithm) với khóa mã hóa biến bản rõ thành bản mã (ciphertext);
- Một khóa giải mã (decryption key) để giải mã bản mã;
- Một thuật toán giải mã (decryption algorithm) với khóa giải mã sẽ biến bản mã trở lại thành bản rõ.

## 2.2.7 Công nghệ RAID

Phần cứng mà DBMS đang chạy phải có khả năng chịu lỗi (fault-tolerant), nghĩa là DBMS phải có khả năng tiếp tục hoạt động ngay cả khi một trong các thành phần phần cứng bị lỗi. Điều này cho thấy có các thành phần dự phòng có thể được tích hợp liền mạch vào hệ thống làm việc bất cứ khi nào có một hoặc nhiều thành phần hỏng hóc. Các thành phần phần cứng chính cần có khả năng chịu lỗi bao gồm ổ đĩa, trình điều khiển đĩa, CPU, bộ nguồn và quạt làm mát.

Một giải pháp là sử dụng công nghệ RAID (Redundant Array of Independent Disks). RAID ban đầu là viết tắt của Redundant Array of Inexpensive Disks, nhưng gần đây chữ “I” trong RAID đã trở thành viết tắt cho Independent. RAID hoạt động dựa trên việc có một mảng đĩa lớn bao gồm sự sắp xếp của một số đĩa độc lập được tổ chức để cải thiện độ tin cậy và đồng thời tăng hiệu suất.

Hiệu suất được tăng lên thông qua phân vùng dữ liệu: dữ liệu được phân đoạn thành các phân vùng có kích thước bằng nhau (các đơn vị phân vùng), được phân phối một cách trong suốt trên nhiều đĩa. Điều này tạo ra sự xuất hiện của một đĩa lớn, nhanh, mặc dù dữ liệu thực sự được phân phối trên một số đĩa nhỏ hơn. Kỹ thuật này cải thiện hiệu suất I/O tổng thể bằng cách cho phép nhiều I/O được phục vụ song song. Đồng thời, việc phân vùng dữ liệu cũng cân bằng tải giữa các đĩa.

## 2.3 SAO LUU VÀ PHỤC HỒI CƠ SỞ DỮ LIỆU

**Phục hồi cơ sở dữ liệu** Quá trình khôi phục cơ sở dữ liệu về trạng thái chính xác trong trường hợp bị lỗi.

Ở phần đầu của chương này, chúng ta đã được giới thiệu khái niệm phục hồi cơ sở dữ liệu như một dịch vụ cần được cung cấp bởi DBMS để đảm bảo rằng cơ sở dữ liệu là đáng tin cậy và duy trì trạng thái nhất quán khi có lỗi.

Việc lưu trữ dữ liệu thường bao gồm bốn loại phương tiện khác nhau với mức độ tin cậy tăng dần: bộ nhớ chính, đĩa từ, băng từ và đĩa quang. Bộ nhớ chính là phương tiện lưu trữ khả biến (volatile), thường không thể tồn tại khi hệ thống gặp sự cố. Đĩa từ cung cấp phương tiện lưu trữ bất biến trực tuyến (online nonvolatile). So với bộ nhớ chính, đĩa từ đáng tin cậy hơn và rẻ hơn nhiều, nhưng chậm hơn 3-4 bậc. Băng từ là một phương tiện lưu trữ bất biến ngoại tuyến (offline nonvolatile), đáng tin cậy hơn nhiều so với đĩa từ và khá rẻ, nhưng chậm hơn, chỉ cung cấp khả năng truy cập tuần tự. Đĩa quang đáng tin cậy hơn băng từ, thường rẻ hơn, nhanh hơn và cung cấp khả năng truy cập ngẫu nhiên. Bộ nhớ chính còn được gọi là bộ nhớ sơ cấp (primary storage) và đĩa từ, băng từ, đĩa quang là bộ nhớ thứ cấp (secondary storage). Lưu trữ ổn định nghĩa là thông tin đã được sao chép trong một số phương tiện lưu trữ bất biến (thường là đĩa) với các chế độ độc lập lỗi. Ví dụ, có thể mô phỏng lưu trữ ổn định bằng công nghệ RAID, đảm bảo rằng sự cố của một đĩa, ngay cả trong quá trình truyền dữ liệu, không dẫn đến mất dữ liệu.

Có nhiều dạng lỗi khác nhau có thể ảnh hưởng đến quá trình xử lý cơ sở dữ liệu, mỗi dạng lỗi phải được xử lý theo một cách khác nhau. Một số các nguyên nhân của lỗi là:

- Hệ thống bị treo;
- Lỗi phương tiện;
- Lỗi phần mềm ứng dụng;
- Thiên tai;
- Sự bất cẩn;
- Phá hoại.

Dù nguyên nhân gây ra lỗi là gì, có hai tác động chính chúng ta cần xem xét: mất bộ nhớ chính, bao gồm cả bộ đệm cơ sở dữ liệu và mất bản sao đĩa của cơ sở dữ liệu. Chúng ta sẽ tìm hiểu về các khái niệm và kỹ thuật có thể giảm thiểu những tác động này và cho phép phục hồi sau lỗi.

DBMS phải cung cấp các phương tiện sau để hỗ trợ khôi phục:

- Một cơ chế sao lưu, tạo ra các bản sao lưu định kỳ của cơ sở dữ liệu;
- Các phương tiện ghi nhật ký, theo dõi trạng thái hiện tại của các giao dịch và các thay đổi của cơ sở dữ liệu;
- Một phương tiện tạo điểm kiểm tra (checkpoint);
- Trình quản lý khôi phục, cho phép hệ thống khôi phục cơ sở dữ liệu về trạng thái nhất quán sau khi bị lỗi.

### 2.3.1 Cơ chế sao lưu

DBMS nên cung cấp một cơ chế cho phép các bản sao lưu của cơ sở dữ liệu và tập tin nhật ký (log file) được thực hiện theo các khoảng thời gian đều đặn mà không nhất thiết phải dừng hệ thống. Bản sao lưu của cơ sở dữ liệu có thể được sử dụng trong trường hợp cơ sở dữ liệu bị hỏng hoặc bị phá hủy. Một bản sao lưu có thể là một bản sao hoàn chỉnh của toàn bộ cơ sở dữ liệu hoặc một bản sao lưu gia tăng, chỉ bao gồm các sửa đổi được thực hiện kể từ lần sao lưu hoàn chỉnh hoặc gia tăng cuối cùng.

### 2.3.2 Tập tin nhật ký

Để theo dõi các giao dịch cơ sở dữ liệu, DBMS duy trì một tập tin đặc biệt được gọi là nhật ký (log hoặc journal) chứa thông tin về tất cả các cập nhật cho cơ sở dữ liệu. Tập tin nhật ký (log file) có thể chứa các dữ liệu sau:

- Các bản ghi giao dịch, bao gồm:
  - Định danh giao dịch;
  - Loại bản ghi nhật ký (bắt đầu giao dịch, chèn, cập nhật, xóa, hủy bỏ, cam kết);
  - Định danh của mục dữ liệu bị ảnh hưởng bởi hành động cơ sở dữ liệu (các thao tác chèn, xóa và cập nhật);
  - Hình ảnh trước của mục dữ liệu, nghĩa là giá trị của nó trước khi thay đổi (chỉ các thao tác cập nhật và xóa);
  - Hình ảnh sau của mục dữ liệu, nghĩa là giá trị của nó sau khi thay đổi (chỉ các thao tác chèn và cập nhật);
  - Thông tin quản lý nhật ký, chẳng hạn như một con trỏ đến các bản ghi nhật ký trước đó và tiếp theo cho giao dịch đó (tất cả các hoạt động).
- Bản ghi điểm kiểm tra (checkpoint).

Nhật ký thường được sử dụng cho các mục đích khác với mục đích phục hồi (ví dụ như để theo dõi và đánh giá hiệu suất). Trong trường hợp này, thông tin bổ sung có thể được ghi lại trong tập tin nhật ký (ví dụ như số lần đọc cơ sở dữ liệu, số lần người dùng đăng nhập, đăng xuất,...), nhưng những thông tin này không liên quan đến khôi phục và do đó sẽ được bỏ qua trong nội dung này. Hình 2.3 minh họa một đoạn của tập tin nhật ký cho thấy ba giao dịch đang thực hiện đồng thời T1, T2 và T3. Các cột pPtr và nPtr biểu diễn cho các con trỏ đến các bản ghi nhật ký trước đó và tiếp theo cho mỗi giao dịch.

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

**Hình 2.3** Ví dụ một đoạn của tập tin nhật ký

Do tầm quan trọng trong quá trình khôi phục, tập tin nhật ký giao dịch có thể được bao gồm hai hoặc ba bộ (nghĩa là hai hoặc ba bản sao riêng biệt được duy trì) để nếu một bản bị hỏng, bản khác có thể được sử dụng. Trước đây, các tập tin nhật ký được lưu trữ trên băng từ vì băng từ đáng tin cậy hơn và rẻ hơn đĩa từ. Tuy nhiên, ngày nay các DBMS được mong đợi có thể phục hồi nhanh chóng sau các lỗi nhỏ nên thông thường tập tin nhật ký sẽ được lưu trữ trực tuyến.

Trong một số môi trường có lượng lớn thông tin ghi nhật ký được lưu trữ hàng ngày, rõ ràng không thể giữ tất cả dữ liệu này trực tuyến mọi lúc. Cần có tập tin nhật ký trực tuyến để khôi phục nhanh sau các lỗi nhỏ (ví dụ, khôi phục giao dịch sau khi bị tắc nghẽn). Các lỗi lớn, chẳng hạn như sự cố đầu đọc đĩa, rõ ràng là mất nhiều thời gian hơn để khôi phục và có thể yêu cầu quyền truy cập vào một phần lớn của nhật ký. Trong những trường hợp này, có thể chấp nhận đợi các phần của tập tin nhật ký được đưa trở lại trực tuyến từ thiết bị lưu trữ ngoại tuyến.

Một cách tiếp cận để xử lý vấn đề ngoại tuyến của nhật ký là chia nhật ký trực tuyến thành hai tập tin truy cập ngẫu nhiên riêng biệt. Các bản ghi nhật ký được ghi vào tập tin đầu tiên cho đến khi nó đạt đến ngưỡng, ví dụ như đây 70%. Sau đó, một tập tin nhật ký thứ hai sẽ được mở và tất cả các bản ghi nhật ký cho các giao dịch mới được ghi vào tập tin thứ hai. Các giao dịch cũ tiếp tục sử dụng tập tin đầu tiên cho đến khi chúng hoàn tất, lúc này tập tin đầu tiên được đóng và chuyển sang lưu trữ ngoại tuyến. Điều này đơn giản hóa việc khôi phục một giao dịch riêng lẻ, vì tất cả các bản ghi nhật ký cho giao dịch đó đều được lưu trữ ngoại tuyến hoặc trực tuyến.

Cần lưu ý rằng tập tin nhật ký là một nút cổ chai tiềm ẩn và tốc độ ghi vào tập tin nhật ký có thể rất quan trọng trong việc xác định hiệu suất tổng thể của hệ cơ sở dữ liệu.

### 2.3.3 Tạo điểm kiểm tra (Checkpointing)

Thông tin trong tập tin nhật ký được sử dụng để khôi phục sau sự cố cơ sở dữ liệu. Một khó khăn với giải pháp này là khi lỗi xảy ra, chúng ta có thể không biết được quay lại nhật ký để tìm kiếm bao xa và chúng ta có thể kết thúc bằng việc thực hiện lại các giao dịch đã được ghi vào cơ sở dữ liệu một cách an toàn. Để giới hạn khôi lượng tìm kiếm và xử lý tiếp theo cần thực hiện trên tập tin nhật ký, chúng ta có thể sử dụng một kỹ thuật gọi là tạo điểm kiểm tra (checkpointing).

#### Checkpoint

Điểm đồng bộ giữa cơ sở dữ liệu và tập tin nhật ký giao dịch. Tại đây, tất cả các bộ đệm đều bị buộc ghi vào bộ nhớ thứ cấp.

Các checkpoint được lên lịch theo các khoảng thời gian định trước và liên quan đến các hoạt động sau:

- Ghi tất cả các bản ghi nhật ký trong bộ nhớ chính vào bộ nhớ phụ;
- Ghi các khôi đã sửa đổi trong bộ đệm cơ sở dữ liệu vào bộ nhớ thứ cấp;
- Ghi một bản ghi checkpoint vào tập tin nhật ký. Bản ghi này chứa các định danh của tất cả các giao dịch đang hoạt động tại thời điểm tạo checkpoint.

Nếu các giao dịch được thực hiện tuần tự, khi xảy ra lỗi, chúng ta sẽ kiểm tra tập tin nhật ký để tìm giao dịch cuối cùng được bắt đầu trước checkpoint cuối cùng vì bất kỳ giao dịch nào trước đó đã được cam kết và ghi vào cơ sở dữ liệu. Do đó, chúng ta chỉ cần thực hiện lại giao dịch đã xảy ra tại thời điểm tạo checkpoint đó và bất kỳ giao dịch nào tiếp theo mà cả bản ghi bắt đầu và cam kết đều xuất hiện trong nhật ký. Nếu giao dịch đang hoạt động tại thời điểm bị lỗi, giao dịch phải được hoàn tác nếu nó bắt đầu trước checkpoint cuối cùng. Nếu các giao dịch được thực hiện đồng thời, chúng ta sẽ thực hiện lại tất cả các giao dịch đã cam kết kể từ thời điểm tạo checkpoint và hoàn tác tất cả các giao dịch đang hoạt động tại thời điểm xảy ra sự cố.

Nói chung, tạo checkpoint là một hoạt động tương đối ít tốn chi phí và thường có thể tạo ba hoặc bốn checkpoint trong một giờ.

## TÓM TẮT

- Bảo mật cơ sở dữ liệu (database security) là các cơ chế bảo vệ cơ sở dữ liệu khỏi các mối đe dọa cố ý hoặc vô ý. Bảo mật cơ sở dữ liệu quan tâm đến việc giảm thiểu hoặc tránh xảy ra các tình huống sau: trộm cắp và gian lận, mất tính bảo mật, mất quyền riêng tư, mất tính toàn vẹn và mất tính khả dụng.
- Mối đe dọa (threat) là bất kỳ tình huống hoặc sự kiện nào, dù là cố ý hay vô tình, sẽ gây ra ảnh hưởng tác động xấu đến hệ thống và tổ chức.
- Các kiểm soát bảo mật dựa trên máy tính cho môi trường đa người dùng bao gồm: ủy quyền, điều khiển truy xuất, khung nhìn, sao lưu và phục hồi, tính toàn vẹn, mã hóa và công nghệ RAID.
- Ủy quyền (authorization) là việc cấp quyền hoặc đặc quyền cho phép chủ thể có quyền truy cập hợp lệ vào hệ thống hoặc đối tượng của hệ thống. Xác thực (authentication) là một cơ chế xác định xem người dùng có phải là người mà họ tuyên bố hay không.
- Hầu hết các DBMS thương mại cung cấp một cách tiếp cận được gọi là Điều khiển truy cập tùy quyền (Discretionary Access Control - DAC), quản lý các đặc quyền bằng cách sử dụng SQL. SQL chuẩn hỗ trợ DAC thông qua các lệnh GRANT và REVOKE. Một số DBMS thương mại cũng cung cấp một cách tiếp cận khác để điều khiển truy cập được gọi là Điều khiển truy cập bắt buộc (Mandatory Access Control - MAC), dựa trên các chính sách trên toàn hệ thống mà người dùng cá nhân không thể thay đổi.
- Một khung nhìn (view) là kết quả động của một hoặc nhiều phép toán quan hệ hoạt động trên các quan hệ cơ sở. Khung nhìn là một quan hệ ảo (virtual relation), không thực sự tồn tại trong cơ sở dữ liệu nhưng sẽ được tạo ra tại thời điểm yêu cầu. Khung nhìn cung cấp một cơ chế bảo mật mạnh mẽ và linh hoạt bằng cách ẩn các phần của cơ sở dữ liệu khỏi một số người dùng nhất định.
- Sao lưu (backup) là quá trình định kỳ lấy một bản sao của cơ sở dữ liệu và tập tin nhật ký (và có thể là các chương trình) lưu vào phương tiện lưu trữ ngoại tuyến. Ghi nhật ký (journaling/logging) là quá trình lưu giữ và duy trì một tập tin nhật ký của tất cả các thay đổi được thực hiện đối với cơ sở dữ liệu để cho phép thực hiện khôi phục một cách hiệu quả trong trường hợp có sự cố.
- Các ràng buộc về tính toàn vẹn cũng góp phần duy trì một hệ thống cơ sở dữ liệu an toàn bằng cách ngăn không cho dữ liệu trở nên không hợp lệ, dẫn đến đưa ra các kết quả sai lệch hoặc không chính xác.
- Mã hóa là biến đổi dữ liệu bằng một thuật toán đặc biệt khiến dữ liệu không thể đọc được bởi bất kỳ chương trình nào nếu không có khóa giải mã.

## CÂU HỎI

1. Giải thích mục đích và phạm vi của bảo mật cơ sở dữ liệu.
2. Trình bày các tình huống liên quan đến bảo mật cơ sở dữ liệu:
  - a. Trộm cắp và gian lận
  - b. Mất tính bảo mật (bí mật)
  - c. Mất sự riêng tư
  - d. Mất tính toàn vẹn
  - e. Mất khả năng sẵn sàng
3. Nêu cụ thể một số mối đe dọa (threat) ảnh hưởng đến hệ thống.
4. Trình bày vai trò của người quản trị cơ sở dữ liệu trong việc bảo mật và khôi phục cơ sở dữ liệu.
5. Trình bày các biện pháp kiểm soát bảo mật dựa trên máy tính trong môi trường đa người dùng sau:
  - a. Ủy quyền
  - b. Điều khiển truy cập
  - c. Khung nhìn
  - d. Sao lưu và phục hồi
  - e. Tính toàn vẹn
  - f. Mã hóa
  - g. Công nghệ RAID
6. Vì sao DBMS cần phải cung cấp chức năng sao lưu và phục hồi?
7. Liệt kê các phương tiện và chiến lược phục hồi.
8. Tập tin nhật ký (log file) là gì? Vai trò, ý nghĩa của log file trong việc khôi phục cơ sở dữ liệu?
9. Trình bày về điểm kiểm tra (checkpoint). Vì sao phải sử dụng checkpoint trong việc khôi phục cơ sở dữ liệu?

## CHƯƠNG 3

### SQL VÀ T-SQL

Trong chương này, chúng ta sẽ tìm hiểu:

- Mục đích và ưu điểm của các khung nhìn trong hệ thống quan hệ; Cách tạo, cập nhật và xóa khung nhìn bằng SQL.
- Đối tượng, mục đích và tầm quan trọng của Ngôn ngữ truy vấn có cấu trúc (Structured Query Language - SQL), cách viết một câu lệnh SQL.
- Các kiểu dữ liệu được hỗ trợ bởi SQL chuẩn.
- Cách sử dụng câu lệnh GRANT và REVOKE như một cấp độ bảo mật.
- Cách tạo thủ tục lưu trữ (stored procedure), hàm (function).
- Cách tạo trình kích hoạt (trigger).

Ngày nay, hệ quản trị cơ sở dữ liệu quan hệ (Relational Database Management System - RDBMS) đã trở thành phần mềm xử lý dữ liệu được sử dụng phổ biến. Trong mô hình quan hệ, tất cả dữ liệu được cấu trúc logic trong các quan hệ (bảng). Mỗi quan hệ có một tên và được tạo thành từ các thuộc tính (cột) dữ liệu được đặt tên. Mỗi bộ (hàng) chứa một giá trị cho mỗi thuộc tính. Và cấu trúc logic đơn giản này là một trong những ưu điểm mạnh mẽ của mô hình quan hệ.

Trong chương này, chúng ta sẽ thảo luận về các thuật ngữ và các khái niệm cấu trúc cơ bản của mô hình dữ liệu quan hệ. Tiếp theo, chúng ta sẽ xem xét các ngôn ngữ dữ liệu có thể được sử dụng để cập nhật và truy xuất dữ liệu. Trọng tâm của chương là phần nội dung trình bày các thao tác xử lý dữ liệu thông thường và nâng cao thông qua việc sử dụng ngôn ngữ SQL chuẩn dưới dạng câu lệnh đơn lẻ hoặc một tập các câu lệnh được tích hợp trong một chương trình con (routine).

## 3.1 SQL

### 3.1.1 Đối tượng của SQL

Lý tưởng nhất, một ngôn ngữ cơ sở dữ liệu nên cho phép người dùng:

- Tạo cơ sở dữ liệu và các cấu trúc quan hệ;
- Thực hiện các tác vụ quản lý dữ liệu cơ bản, chẳng hạn như chèn, sửa đổi và xóa dữ liệu khỏi các quan hệ;
- Thực hiện cả truy vấn đơn giản và phức tạp.

Một ngôn ngữ cơ sở dữ liệu phải thực hiện các tác vụ này với nỗ lực tối thiểu của người dùng, đồng thời cấu trúc lệnh và cú pháp phải tương đối dễ học. Cuối cùng, ngôn ngữ phải có tính khả chuyển; nghĩa là, phải tuân theo một số tiêu chuẩn đã được công nhận để người dùng có thể sử dụng cùng một cấu trúc lệnh và cú pháp khi chuyển từ DBMS này sang DBMS khác. SQL đã được đề xuất nhằm đáp ứng các yêu cầu này.

Là một ngôn ngữ dữ liệu, SQL chuẩn gồm hai thành phần chính:

- Ngôn ngữ định nghĩa dữ liệu (Data Definition Language - DDL) để xác định cấu trúc cơ sở dữ liệu và điều khiển quyền truy cập vào dữ liệu;
- Ngôn ngữ thao tác dữ liệu (Data Manipulation Language - DML) để truy xuất và cập nhật dữ liệu.

SQL là một ngôn ngữ tương đối dễ học do:

- SQL là một ngôn ngữ phi thủ tục; người dùng chỉ định những thông tin muốn có, thay vì cách như thế nào để có được thông tin đó.
- Giống như hầu hết các ngôn ngữ hiện đại, SQL về cơ bản là định dạng tự do, nghĩa là các phần của câu lệnh không phải được nhập tại các vị trí cụ thể trên màn hình.
- Cấu trúc lệnh bao gồm các từ tiếng Anh tiêu chuẩn như CREATE TABLE, INSERT, SELECT. Ví dụ:

```

    ▪ CREATE TABLE Staff
      (
        staffNo VARCHAR(5),
        lName VARCHAR(15),
        salary DECIMAL(7, 2)
      );
    ▪ INSERT INTO Staff
      VALUES ('SG16', 'Brown', 8300);
    ▪ SELECT staffNo, lName, salary
      FROM Staff
      WHERE salary > 10000;
  
```

- SQL có thể được sử dụng bởi một loạt các kiểu người dùng như quản trị viên cơ sở dữ liệu, nhân viên quản lý, nhà phát triển ứng dụng và nhiều kiểu người dùng cuối khác.

### 3.1.2 Tầm quan trọng của SQL

SQL là ngôn ngữ cơ sở dữ liệu tiêu chuẩn đầu tiên và duy nhất được chấp nhận rộng rãi. Gần như mọi nhà cung cấp lớn hiện nay đều cung cấp các sản phẩm cơ sở dữ liệu dựa trên SQL hoặc với giao diện SQL. Có một sự đầu tư lớn vào ngôn ngữ SQL của cả nhà cung cấp và người dùng. SQL đã trở thành một phần của các kiến trúc ứng dụng như Kiến trúc Ứng dụng Hệ thống (Systems Application Architecture - SAA) của IBM và là sự lựa chọn chiến lược của nhiều tổ chức lớn và có ảnh hưởng, chẳng hạn như tập đoàn Open Group cho các tiêu chuẩn UNIX. SQL cũng đã trở thành Tiêu chuẩn xử lý thông tin liên bang (Federal Information Processing Standard - FIPS) mà theo đó, sự tuân thủ là bắt buộc đối với tất cả các DBMS được cung cấp cho chính phủ Hoa Kỳ. SQL Access Group, một tập hợp các nhà cung cấp, đã xác định một tập hợp các cải tiến cho SQL hướng đến hỗ trợ khả năng tương tác giữa các hệ thống khác nhau.

SQL được sử dụng trong các tiêu chuẩn khác và thậm chí ảnh hưởng đến sự phát triển của các tiêu chuẩn khác với vai trò như một công cụ định hướng. Sự phát triển của ngôn ngữ được hỗ trợ bởi sự quan tâm học thuật đáng kể, cung cấp cả cơ sở lý thuyết cho ngôn ngữ và các kỹ thuật cần thiết để hiện thực hóa nó thành công. Điều này đặc biệt đúng trong tối ưu hóa truy vấn, phân tán dữ liệu và bảo mật. Hiện nay đã có những triển khai SQL chuyên biệt hướng đến các thị trường mới, chẳng hạn như Xử lý Phân tích Trực tuyến (OnLine Analytical Processing - OLAP).

SQL theo tiêu chuẩn ISO (SQL chuẩn) không sử dụng các thuật ngữ chính thức như quan hệ, thuộc tính và bộ, thay vào đó sử dụng các thuật ngữ bảng, cột và hàng. Trong tài liệu này, khi trình bày về SQL, chúng ta chủ yếu sử dụng thuật ngữ theo chuẩn ISO.

### 3.1.3 Cách viết câu lệnh SQL

Trong phần này, chúng ta mô tả ngắn gọn cấu trúc một câu lệnh SQL và ký hiệu sử dụng để xác định định dạng của các cấu trúc SQL khác nhau. Một câu lệnh SQL bao gồm các từ dành riêng và các từ do người dùng định nghĩa.

- Các từ dành riêng (reserved word) là một phần của ngôn ngữ SQL, mang ý nghĩa cố định.
- Các từ do người dùng định nghĩa (user-defined word) được tạo ra bởi người dùng (theo các quy tắc cú pháp nhất định) và biểu diễn cho tên của các đối tượng cơ sở dữ liệu khác nhau như bảng, cột, khung nhìn, chỉ mục,...

Hầu hết các thành phần của câu lệnh SQL không phân biệt chữ hoa chữ thường, nghĩa là các chữ cái đều có thể được nhập bằng chữ hoa hoặc chữ thường.

Một ngoại lệ quan trọng đối với quy tắc này là dữ liệu ký tự chữ phải được nhập chính xác như nó xuất hiện trong cơ sở dữ liệu. Ví dụ, nếu chúng ta lưu trữ họ của một người là “SMITH” và sau đó tìm kiếm bằng chuỗi “Smith”, thì hàng đó sẽ không được tìm thấy.

Mặc dù SQL là định dạng tự do, nhưng câu lệnh hoặc tập hợp các câu lệnh SQL sẽ dễ đọc hơn nếu sử dụng quy tắc canh lề và xuống dòng. Chẳng hạn:

- Mỗi mệnh đề trong câu lệnh nên bắt đầu trên một dòng mới;
- Đầu mỗi mệnh đề nên thăng hàng với đầu các mệnh đề khác;
- Nếu một mệnh đề có nhiều phần, mỗi phần phải xuất hiện trên một dòng riêng biệt và được thụt lề dưới phần đầu của mệnh đề để thể hiện mối quan hệ.

Trong tài liệu này chúng ta sử dụng dạng mở rộng sau của ký pháp Backus Naur Form (BNF) để định nghĩa các câu lệnh SQL:

- Các chữ hoa được sử dụng để biểu thị các từ dành riêng và phải được viết chính xác như hình minh họa;
- Các chữ thường được sử dụng để biểu diễn cho các từ do người dùng định nghĩa;
- Một thanh dọc (|) cho biết một lựa chọn trong số các lựa chọn thay thế; ví dụ, a | b | c;
- Dấu ngoặc nhọn chỉ ra một phần tử bắt buộc; ví dụ, {a};
- Dấu ngoặc vuông cho biết một phần tử tùy chọn; ví dụ, [a];
- Dấu ba chấm (...) được sử dụng để biểu thị sự lặp lại tùy chọn của một mục không hoặc nhiều lần; ví dụ, {a|b} (, c . . . ) có nghĩa là a hoặc b theo sau bởi không hoặc nhiều lần lặp lại c được phân tách bằng dấu phẩy.

### 3.2 CHỈ MỤC

Chỉ mục (index) là một cấu trúc cung cấp truy cập nhanh vào các hàng của bảng dựa trên các giá trị của một hoặc nhiều cột. Sự hiện diện của một chỉ mục có thể cải thiện đáng kể hiệu suất của một truy vấn. Tuy nhiên, vì hệ thống có thể cập nhật các chỉ mục mỗi khi các bảng bên dưới được cập nhật, nên các chi phí chung bổ sung có thể sẽ phát sinh. Các chỉ mục thường được tạo để đáp ứng các tiêu chí tìm kiếm cụ thể sau khi bảng đã được sử dụng một thời gian và đã tăng kích thước. Câu lệnh tạo chỉ mục không phải là SQL chuẩn. Tuy nhiên, hầu hết các phương ngữ đều hỗ trợ ít nhất các khả năng sau:

```
CREATE [UNIQUE] INDEX IndexName  
ON TableName (columnName [ASC | DESC] [, . . .])
```

Các cột được chỉ định tạo thành khóa chỉ mục và phải được liệt kê theo thứ tự từ chính đến phụ. Chỉ mục chỉ có thể được tạo trên các bảng cơ sở, không thể được tạo trên các khung nhìn. Nếu mệnh đề UNIQUE được sử dụng, tính duy nhất của cột được lập chỉ mục hoặc kết hợp các cột sẽ được DBMS thực thi. Điều này chắc chắn là bắt buộc đối với khóa chính và có thể đối với các cột khác (ví dụ như đối với các khóa ứng viên). Mặc dù có thể tạo chỉ mục bất kỳ lúc nào, nhưng chúng ta có thể gặp sự cố nếu cố gắng tạo chỉ mục duy nhất (unique) trên bảng đã có các bản ghi, vì các giá trị được lưu trữ cho (các) cột được lập chỉ mục đã có thể có sự trùng lặp. Do đó, chúng ta nên tạo các chỉ mục duy nhất, ít nhất là cho các cột khóa chính, khi bảng cơ sở vừa được tạo.

Đối với bảng Staff và PropertyForRent, chúng ta có thể muốn tạo ít nhất các chỉ mục sau:

```
CREATE UNIQUE INDEX StaffNoInd ON Staff (staffNo);
CREATE UNIQUE INDEX PropertyNoInd ON PropertyForRent
(propertyNo);
```

Đối với mỗi cột, có thể chỉ định thứ tự tăng dần (ASC) hoặc giảm dần (DESC), và ASC là mặc định. Ví dụ, nếu chúng ta thực thi lệnh sau:

```
CREATE INDEX RentInd ON PropertyForRent (city, rent);
```

thì một chỉ mục được gọi là RentInd sẽ được tạo cho bảng PropertyForRent. Các hàng trong bảng này sẽ theo được sắp xếp theo thứ tự bảng chữ cái trong cột city và sau đó là rent trong mỗi city.

Nếu chúng ta tạo chỉ mục cho bảng cơ sở và sau đó quyết định rằng nó không còn cần thiết nữa, chúng ta có thể sử dụng câu lệnh DROP INDEX để xóa chỉ mục khỏi cơ sở dữ liệu. DROP INDEX có dạng sau:

```
DROP INDEX IndexName
```

Câu lệnh sau sẽ xóa chỉ mục được tạo trong ví dụ trước:

```
DROP INDEX RentInd;
```

### 3.3 KHUNG NHÌN (VIEW)

#### Khung nhìn

Là kết quả động của một hoặc nhiều phép toán quan hệ thao tác trên các quan hệ cơ sở. Một khung nhìn là một quan hệ ảo, không nhất thiết phải tồn tại trong cơ sở dữ liệu nhưng có thể được tạo ra theo yêu cầu của một người dùng cụ thể, tại thời điểm được yêu cầu.

Đối với người dùng cơ sở dữ liệu, một khung nhìn xuất hiện giống như một bảng cơ sở, với một tập hợp các cột được đặt tên và các hàng dữ liệu. Tuy nhiên, không giống như bảng cơ sở, một khung nhìn không nhất thiết phải tồn tại trong cơ sở dữ liệu dưới dạng một tập giá trị dữ liệu được lưu trữ. Thay vào đó, một khung nhìn được định nghĩa là một truy vấn trên một hoặc nhiều bảng cơ sở hoặc khung nhìn khác. DBMS lưu trữ định nghĩa của khung nhìn trong cơ sở dữ liệu. Khi DBMS gặp tham chiếu đến một khung nhìn, một cách tiếp cận là tra cứu định nghĩa này và chuyển yêu cầu thành một yêu cầu tương đương dựa trên các bảng nguồn của khung nhìn và thực hiện yêu cầu tương đương này. Quá trình hợp nhất này, được gọi là phân giải khung nhìn (view resolution).

Một cách tiếp cận thay thế, được gọi là cụ thể hóa khung nhìn (view materialization), lưu khung nhìn dưới dạng bảng tạm thời trong cơ sở dữ liệu và duy trì tính tức thời của khung nhìn khi các bảng cơ sở bên dưới được cập nhật.

### 3.3.1 Tạo khung nhìn (CREATE VIEW)

Cú pháp lệnh tạo khung nhìn rất đơn giản và dễ hiểu, giống với cú pháp của một bảng, mặc dù có ba tùy chọn thú vị cũng có thể được gán cho khung nhìn.

Định dạng của câu lệnh CREATE VIEW là:

```
CREATE VIEW [schema_name.]view_name  
WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA}  
AS  
    SELECT_statement  
    [WITH CHECK OPTION]
```

Một khung nhìn được định nghĩa bằng cách chỉ định một câu lệnh SELECT. Một tên có thể được tùy chọn được chỉ định cho mỗi cột trong khung nhìn. Nếu danh sách tên cột được chỉ định, danh sách đó phải có cùng số mục với số cột được tạo bởi subselect. Nếu danh sách tên cột bị bỏ qua, mỗi cột trong khung nhìn sẽ lấy tên của cột tương ứng trong câu lệnh SELECT. Danh sách tên cột phải được chỉ định nếu có bất kỳ sự không rõ ràng nào trong tên của một cột. Điều này có thể xảy ra nếu câu lệnh SELECT bao gồm các cột được tính toán và mệnh đề phụ AS không được sử dụng để đặt tên cho các cột đó hoặc nó tạo ra hai cột có tên giống nhau do kết quả của một phép nối.

Tên của khung nhìn (view\_name) có thể được đặt sau tên lược đồ (schema\_name); tuy nhiên, điều này là tùy chọn; nếu chúng ta đang ở đúng cơ sở dữ liệu và đăng nhập bằng ID muốn tạo khung nhìn, tên lược đồ là không bắt buộc.

Tùy chọn ENCRYPTION sẽ mã hóa khung nhìn để khung nhìn được bảo mật và không ai có thể xem mã bên dưới hoặc sửa đổi nội dung của câu lệnh SELECT bên trong. Tuy nhiên, tốt nhất hãy giữ một bản sao lưu nội dung của khung nhìn ở một nơi an toàn trong quá trình phát triển dự phòng trong trường hợp có bất kỳ sửa đổi nào được yêu cầu.

Tùy chọn cuối cùng là VIEW\_METADATA. Khi làm việc với các khung nhìn bằng C#, Java, Excel,... người dùng có thể yêu cầu trả về siêu dữ liệu của khung nhìn. Nếu không có tùy chọn này, siêu dữ liệu sẽ thuộc các bảng cơ sở trong khung nhìn. Thay vào đó, để ẩn thông tin này và hiển thị tên cột từ khung nhìn, hãy tạo khung nhìn với tùy chọn VIEW\_METADATA.

Lệnh SELECT được gọi là truy vấn định nghĩa (defining query) khung nhìn. Nếu WITH CHECK OPTION được chỉ định, SQL đảm bảo rằng nếu một hàng không thỏa mãn mệnh đề WHERE của truy vấn định nghĩa khung nhìn, thì hàng đó sẽ không được thêm vào bảng cơ sở bên dưới của khung nhìn. Cần lưu ý rằng để tạo khung nhìn thành công, chúng ta phải có đặc quyền SELECT trên tất cả các bảng được tham chiếu trong lệnh SELECT và đặc quyền USAGE trên bất kỳ miền nào được sử dụng trong các cột được tham chiếu.

Lưu ý:

- Chúng ta không thể tham chiếu bất kỳ biến tạm thời hoặc bảng tạm thời nào trong một khung nhìn hoặc tạo một bảng mới từ một khung nhìn bằng cách sử dụng mệnh đề INTO. Nói cách khác, không thể có SELECT column INTO new\_table.
- Mã hóa khung nhìn có vẻ như là một ý tưởng hay để ẩn một phần lược đồ cơ sở dữ liệu khỏi những người dùng; tuy nhiên, hãy thận trọng khi sử dụng các khung nhìn được mã hóa và luôn sao lưu nguồn trong một môi trường an toàn và bảo mật. Sử dụng các khung nhìn được mã hóa chỉ khi thực sự cần thiết.
- Ngay cả khi WITH CHECK OPTION được xác định, nếu dữ liệu được sửa đổi trực tiếp trong bảng, nó sẽ không được xác thực đối với bất kỳ khung nhìn nào được định nghĩa trên bảng đó, mặc dù, tất nhiên, dữ liệu sẽ được thêm vào bảng. Ngoài ra, nếu khung nhìn sử dụng TOP thì không thể sử dụng WITH CHECK OPTION.

Mặc dù tất cả các khung nhìn đều được tạo theo cùng một cách, nhưng trên thực tế, các loại khung nhìn khác nhau được sử dụng cho các mục đích khác nhau. Chúng ta sẽ minh họa các loại khung nhìn khác nhau bằng các ví dụ.

### Ví dụ 3.1 Tạo khung nhìn ngang (horizontal view)

*Tạo khung nhìn để người quản lý tại chi nhánh B003 có thể xem thông tin chi tiết chỉ của các nhân viên làm việc tại chi nhánh đó.*

Khung nhìn ngang hạn chế quyền truy cập của người dùng vào các hàng của một hoặc nhiều bảng bên dưới.

```
CREATE VIEW Manager3Staff
AS
SELECT *
FROM Staff
WHERE branchNo = 'B003';
```

Câu lệnh trên tạo ra một khung nhìn được gọi là Manager3Staff có cùng tên cột với bảng Staff nhưng chỉ chứa những hàng có số chi nhánh là B003. (Nói một cách chính xác, cột branchNo là không cần thiết và có thể nên bị loại bỏ ra khỏi định nghĩa của khung nhìn, vì tất cả các hàng đều có branchNo='B003'.) Nếu bây giờ chúng ta thực hiện câu lệnh sau:

```
SELECT * FROM Manager3Staff;
```

sẽ được bảng kết quả như bên dưới. Để đảm bảo rằng người quản lý chi nhánh chỉ có thể nhìn thấy các hàng này, người quản lý không nên được cấp quyền truy cập vào bảng cơ sở Staff. Thay vào đó, người quản lý sẽ được cấp quyền truy cập vào khung nhìn Manager3Staff. Thực tế, điều này cung cấp cho người quản lý chi nhánh một khung nhìn tùy chỉnh của bảng Staff, chỉ hiển thị các nhân viên tại chính chi nhánh đó.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Dữ liệu của khung nhìn Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

### Ví dụ 3.2 Tạo một khung nhìn dọc (vertical view)

*Tạo khung nhìn thông tin chi tiết về nhân viên tại chi nhánh B003 ngoại trừ thông tin về lương, do chỉ người quản lý mới có thể truy cập thông tin chi tiết về lương của nhân viên làm việc tại chi nhánh của họ.*

Khung nhìn dọc hạn chế quyền truy cập của người dùng vào các cột của một hoặc nhiều bảng bên dưới.

```
CREATE VIEW Staff3
AS
SELECT staffNo, fName, lName, position, sex
FROM Staff
WHERE branchNo = 'B003';
```

Lưu ý rằng chúng ta có thể viết lại câu lệnh này sử dụng khung nhìn Manager3Staff thay vì bảng Staff như sau:

```
CREATE VIEW Staff3
AS
SELECT staffNo, fName, lName, position, sex
FROM Manager3Staff;
```

Dù bằng cách nào, chúng ta đã tạo ra một khung nhìn được gọi là Staff3 với các cột giống như bảng Staff, nhưng không bao gồm các cột salary, DOB, và branchNo. Nếu liệt kê khung nhìn này, chúng ta nhận được bảng kết quả như bên dưới. Để đảm bảo rằng chỉ người quản lý chi nhánh mới có thể xem chi tiết lương, nhân viên tại chi nhánh B003 không được cấp quyền truy cập vào bảng cơ sở Staff hoặc khung nhìn Manager3Staff. Thay vào đó, họ sẽ được cấp quyền truy cập vào khung nhìn Staff3, do đó hạn chế được họ truy cập vào dữ liệu lương nhạy cảm.

Khung nhìn dọc thường được sử dụng khi dữ liệu được lưu trữ trong bảng được sử dụng bởi nhiều người dùng hoặc nhóm người dùng khác nhau. Nó cung cấp một “bảng” riêng cho những người dùng này chỉ với các cột họ cần.

Dữ liệu của khung nhìn Staff3.

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

### Ví dụ 3.3 Khung nhìn được gom nhóm và nối

*Tạo khung nhìn về nhân viên quản lý bất động sản cho thuê, bao gồm số chi nhánh họ làm việc tại, số nhân viên và số lượng bất động sản mà họ đang quản lý.*

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS
SELECT s.branchNo, s.staffNo, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo;
```

Ví dụ này minh họa việc sử dụng một lựa chọn con chứa mệnh đề GROUP BY (cung cấp một khung nhìn được gọi là một khung nhìn được gom nhóm - grouped view) và chứa nhiều bảng (cung cấp một khung nhìn được gọi là một khung nhìn được nối - joined view). Một trong những lý do thường xuyên nhất để sử dụng khung nhìn là đơn giản hóa các truy vấn nhiều bảng. Khi một khung nhìn được nối đã được định nghĩa, chúng ta thường có thể sử dụng một truy vấn một bảng đơn giản trên khung nhìn cho các truy vấn mà nếu không sẽ yêu cầu phải nối nhiều bảng. Lưu ý rằng chúng ta phải đặt tên cho các cột trong định nghĩa của khung nhìn do sử dụng hàm tổng hợp COUNT trong lựa chọn con.

Dữ liệu của khung nhìn StaffPropCnt.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SG41	1
B007	SA9	1

#### 3.3.2 Xóa khung nhìn (DROP VIEW)

Khung nhìn bị xóa khỏi cơ sở dữ liệu với câu lệnh DROP VIEW:

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

DROP VIEW khiến định nghĩa của khung nhìn bị xóa khỏi cơ sở dữ liệu. Ví dụ, chúng ta có thể loại bỏ khung nhìn Manager3Staff bằng cách sử dụng câu lệnh sau:

```
DROP VIEW Manager3Staff;
```

Nếu CASCADE được chỉ định, DROP VIEW sẽ xóa tất cả các đối tượng phụ thuộc có liên quan; nói cách khác, tất cả các đối tượng tham chiếu đến khung nhìn. Điều này có nghĩa là DROP VIEW cũng xóa bất kỳ khung nhìn nào được định nghĩa trên khung nhìn đang bị xóa. Nếu RESTRICT được chỉ định và có bất kỳ đối tượng nào khác mà sự tồn tại của chúng phụ thuộc vào khung nhìn đang bị loại bỏ, câu lệnh DROP VIEW sẽ bị từ chối. Cài đặt mặc định là RESTRICT.

### 3.3.3 Phân giải khung nhìn

Sau khi tìm hiểu cách tạo và sử dụng các khung nhìn, bây giờ chúng ta khảo sát kỹ hơn cách xử lý truy vấn trên một khung nhìn. Để minh họa quá trình phân giải khung nhìn (view resolution), hãy xem xét truy vấn sau, truy vấn này đếm số lượng bất động sản được quản lý bởi từng nhân viên tại chi nhánh B003. Truy vấn này thực hiện trên khung nhìn StaffPropCnt của Ví dụ 3.3:

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

Phân giải khung nhìn hợp nhất truy vấn trên với truy vấn định nghĩa khung nhìn StaffPropCnt như sau:

1. Tên cột khung nhìn trong danh sách SELECT được dịch thành tên cột tương ứng của chúng trong truy vấn định nghĩa. Điều này mang lại:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
```

2. Tên khung nhìn trong mệnh đề FROM được thay thế bằng danh sách FROM tương ứng của truy vấn định nghĩa:

```
FROM Staff s, PropertyForRent p
```

3. Mệnh đề WHERE của truy vấn được kết hợp với mệnh đề WHERE của truy vấn định nghĩa bằng cách sử dụng toán tử logic AND, do đó:

```
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
```

4. Các mệnh đề GROUP BY và HAVING được sao chép từ truy vấn định nghĩa. Trong ví dụ này, chúng ta chỉ có mệnh đề GROUP BY:

```
GROUP BY s.branchNo, s.staffNo
```

5. Cuối cùng, mệnh đề ORDER BY được sao chép từ truy vấn người dùng với tên cột khung nhìn được dịch thành tên cột của truy vấn định nghĩa:

```
ORDER BY s.staffNo
```

6. Truy vấn hợp nhất cuối cùng như sau:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Kết quả sau khi phân giải khung nhìn

staffNo	cnt
SG14	1
SG37	2

### 3.3.4 Các hạn chế trên khung nhìn

SQL chuẩn áp đặt một số hạn chế quan trọng đối với việc tạo và sử dụng các khung nhìn, mặc dù có sự khác biệt đáng kể giữa các phương ngữ.

Nếu một cột trong khung nhìn dựa trên một hàm tổng hợp, thì cột đó có thể chỉ xuất hiện trong mệnh đề SELECT và ORDER BY của các truy vấn truy xuất khung nhìn. Cụ thể, một cột như vậy có thể không được sử dụng trong mệnh đề WHERE và có thể không phải là đối số cho một hàm tổng hợp trong bất kỳ truy vấn nào dựa trên khung nhìn. Ví dụ, hãy xem xét khung nhìn StaffPropCnt của Ví dụ 3.3, có một cột cnt dựa trên hàm tổng hợp COUNT. Truy vấn sau sẽ không thành công:

```
SELECT COUNT(cnt)
  FROM StaffPropCnt;
```

bởi vì chúng ta đang sử dụng một hàm tổng hợp trên cột cnt, mà chính cột này lại dựa trên một hàm tổng hợp khác. Tương tự, truy vấn sau cũng sẽ không thành công:

```
SELECT *
  FROM StaffPropCnt
 WHERE cnt > 2;
```

bởi vì chúng ta đang sử dụng cột khung nhìn, cnt, bắt nguồn từ một hàm tổng hợp, ở phía bên trái của mệnh đề WHERE.

Một khung nhìn được gom nhóm có thể không bao giờ được nối với một bảng cơ sở hoặc một khung nhìn khác. Ví dụ, khung nhìn StaffPropCnt là khung nhìn được gom nhóm, vì vậy bất kỳ nỗ lực nào để nối khung nhìn này với một bảng hoặc khung nhìn khác đều không thành công.

### 3.3.5 Khả năng cập nhật khung nhìn

Tất cả các cập nhật cho bảng cơ sở được phản ánh ngay lập tức trong tất cả các khung nhìn dựa trên bảng cơ sở đó. Tương tự, chúng ta có thể mong đợi rằng nếu một khung nhìn được cập nhật, (các) bảng cơ sở cũng sẽ phản ánh sự thay đổi này. Tuy nhiên, hãy xem xét điều gì sẽ xảy ra nếu chúng ta cố gắng chèn một bản ghi vào khung nhìn StaffPropCnt biểu đạt rằng tại chi nhánh B003, nhân viên SG5 quản lý hai bất động sản, bằng cách sử dụng câu lệnh sau:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

Chúng ta phải chèn hai bản ghi vào bảng PropertyForRent cho thấy nhân viên SG5 quản lý bất động sản nào. Tuy nhiên, chúng ta không biết chúng là bất động sản gì; tất cả những gì chúng ta biết là nhân viên này quản lý bất động sản. Nói cách khác, chúng ta không biết các giá trị khóa chính tương ứng cho bảng PropertyForRent. Nếu chúng ta thay đổi định nghĩa của khung nhìn và thay thế số đếm bằng các số của bất động sản thực tế như sau:

```
CREATE VIEW StaffPropList (branchNo, staffNo, propertyNo)
AS
SELECT s.branchNo, s.staffNo, p.propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo;
```

và chúng ta có gắng chèn bản ghi:

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

thì vẫn có vấn đề với việc chèn này, vì chúng ta đã chỉ rõ trong định nghĩa của bảng PropertyForRent rằng tất cả các cột, ngoại trừ postcode và staffNo, là không được phép có giá trị rỗng. Tuy nhiên, vì khung nhìn StaffPropList đã loại bỏ tất cả các cột của bảng PropertyForRent, ngoại trừ số bất động sản, nên chúng ta không có cách nào cung cấp giá trị cho các cột không được rỗng còn lại.

Định nghĩa được đưa ra trong SQL chuẩn là một khung nhìn có thể cập nhật được khi và chỉ khi:

- DISTINCT không được chỉ định; nghĩa là, các hàng trùng lặp không được loại bỏ khỏi kết quả truy vấn.
- Mỗi phần tử trong danh sách SELECT của truy vấn định nghĩa là một tên cột (thay vì một hằng số, biểu thức hoặc hàm tổng hợp) và không có tên cột nào xuất hiện nhiều hơn một lần.
- Mệnh đề FROM chỉ định một bảng duy nhất; nghĩa là, khung nhìn phải có một bảng nguồn duy nhất mà trên đó người dùng có các đặc quyền bắt buộc. Nếu bản thân bảng nguồn là một khung nhìn, thì khung nhìn đó phải thỏa mãn các điều kiện này. Do đó, điều này loại trừ mọi khung nhìn dựa trên nối (JOIN), hợp (UNION), giao (INTERSECT) hoặc hiệu (EXCEPT).
- Mệnh đề WHERE không bao gồm bất kỳ SELECT lồng nào tham chiếu đến bảng trong mệnh đề FROM.
- Không có mệnh đề GROUP BY hoặc HAVING trong truy vấn định nghĩa.

Ngoài ra, tất cả các hàng được thêm vào thông qua khung nhìn không được vi phạm các ràng buộc toàn vẹn của bảng cơ sở. Ví dụ, nếu một hàng mới được thêm vào thông qua một khung nhìn, các cột không có trong khung nhìn được đặt thành null, nhưng điều này không được vi phạm ràng buộc toàn vẹn NOT NULL trong bảng cơ sở.

Khái niệm cơ bản đằng sau những hạn chế này như sau:

**Khung nhìn  
có thể  
cập nhật**

Để một khung nhìn có thể cập nhật được, DBMS phải có khả năng truy vết bất kỳ hàng hoặc cột nào trong khung nhìn về hàng hoặc cột của nó trong bảng nguồn.

### 3.3.6 Tùy chọn WITH CHECK OPTION

Các hàng tồn tại trong một khung nhìn vì chúng thỏa mãn điều kiện WHERE của truy vấn định nghĩa. Nếu một hàng bị thay đổi sao cho không còn thỏa mãn điều kiện này, thì sẽ biến mất khỏi khung nhìn. Tương tự, các hàng mới sẽ xuất hiện trong khung nhìn khi một thao tác chèn hoặc cập nhật trên khung nhìn khiến chúng thỏa mãn điều kiện WHERE.

Nói chung, mệnh đề WITH CHECK OPTION của câu lệnh CREATE VIEW cấm một hàng di chuyển ra khỏi khung nhìn. Các từ khóa tùy chọn LOCAL/CASCDED được áp dụng cho khung nhìn có cấu trúc lan truyền, nghĩa là một khung nhìn có nguồn gốc từ một khung nhìn khác. Nếu LOCAL được chỉ định, thì bất kỳ chèn hoặc cập nhật hàng nào trên khung nhìn này và trên bất kỳ khung nhìn nào được xác định trực tiếp hoặc gián tiếp trên khung nhìn này, không được làm cho hàng đó biến mất khỏi khung nhìn, trừ khi hàng đó cũng biến mất từ khung nhìn/bảng dẫn xuất bên dưới. Nếu CASCDED được chỉ định (mặc định), thì bất kỳ chèn hoặc cập nhật hàng nào trên khung nhìn này và trên bất kỳ khung nhìn nào được định nghĩa trực tiếp hoặc gián tiếp trên khung nhìn này không được làm cho hàng đó biến mất khỏi khung nhìn.

Tính năng này hữu ích đến mức nó có thể làm cho việc thao tác với các khung nhìn hấp dẫn hơn so với thao tác trên các bảng cơ sở. Khi một câu lệnh INSERT hoặc UPDATE trên khung nhìn vi phạm điều kiện WHERE của truy vấn định nghĩa, thao tác sẽ bị từ chối. Hành vi này thực thi các ràng buộc trên cơ sở dữ liệu và giúp bảo toàn tính toàn vẹn của cơ sở dữ liệu.

#### Ví dụ 3.4 WITH CHECK OPTION

Hãy xem xét lại khung nhìn đã được tạo trong ví dụ trước đây:

```
CREATE VIEW Manager3Staff
AS
    SELECT *
    FROM Staff
    WHERE branchNo = 'B003'
    WITH CHECK OPTION;
```

Nếu chúng ta cố gắng cập nhật số chi nhánh của một hàng từ B003 thành B005, ví dụ:

```
UPDATE Manager3Staff
SET branchNo = 'B005'
WHERE staffNo = 'SG37';
```

thì đặc tả của mệnh đề WITH CHECK OPTION trong định nghĩa của khung nhìn sẽ ngăn điều này xảy ra, vì nó sẽ khiến hàng bị xóa khỏi khung nhìn ngang này.

Tương tự, nếu chúng ta cố gắng chèn hàng sau vào khung nhìn:

```
INSERT INTO Manager3Staff
VALUES ('SL15', 'Mary', 'Black', 'Assistant', 'F',
       '1967-06-21', 8000, 'B002');
```

thì đặc tả của WITH CHECK OPTION sẽ ngăn không cho hàng được chèn vào bảng Staff bên dưới và ngay lập tức biến mất khỏi khung nhìn này.

Bây giờ hãy xem xét tình huống mà Manager3Staff không được định nghĩa trực tiếp trên Staff mà trên một khung nhìn khác của Staff:

```
CREATE VIEW LowSalary
AS
SELECT *
FROM Staff
WHERE salary > 9000;
CREATE VIEW HighSalary
AS
SELECT *
FROM LowSalary
WHERE salary > 10000
WITH LOCAL CHECK OPTION;
CREATE VIEW Manager3Staff
AS
SELECT *
FROM HighSalary
WHERE branchNo = 'B003';
```

Nếu bây giờ chúng ta thử lệnh cập nhật sau trên Manager3Staff:

```
UPDATE Manager3Staff
SET salary = 9500 WHERE staffNo = 'SG37';
```

thì cập nhật này sẽ không thành công: mặc dù cập nhật sẽ làm cho hàng biến mất khỏi khung nhìn HighSalary, nhưng sẽ không biến mất khỏi bảng LowSalary mà HighSalary có nguồn gốc từ đó. Tuy nhiên, nếu thay vào đó, cập nhật có gắng thay đổi mức lương thành 8000, thì cập nhật sẽ thành công, vì hàng sẽ không còn là một phần của LowSalary. Ngoài ra, nếu khung nhìn HighSalary đã được chỉ định WITH CASCADED CHECK OPTION, thì việc đặt mức lương thành 9500 hoặc 8000 sẽ bị từ chối, vì hàng sẽ biến mất khỏi HighSalary. Do đó, để đảm bảo rằng những điều bất thường như thế này không phát sinh, mỗi khung nhìn nên được tạo bằng cách sử dụng WITH CASCADED CHECK OPTION.

### 3.3.7 Ưu điểm và hạn chế của khung nhìn

Việc chỉ cấp quyền truy cập của một số người dùng vào khung nhìn có những lợi thế tiềm năng so với việc cho phép người dùng truy cập trực tiếp vào các bảng cơ sở. Thật không may, các khung nhìn trong SQL cũng có những hạn chế.

#### 3.3.7.1 Các ưu điểm

Trong DBMS đa người dùng, các khung nhìn đóng vai trò trung tâm trong việc xác định cấu trúc của cơ sở dữ liệu và thực thi bảo mật và tính toàn vẹn. Những lợi thế chính của các khung nhìn được mô tả tiếp theo.

- Độc lập dữ liệu:** Một khung nhìn có thể trình bày một bức tranh nhất quán, không thay đổi về cấu trúc của cơ sở dữ liệu, ngay cả khi các bảng nguồn bên dưới bị thay đổi (ví dụ, nếu các cột được thêm vào hoặc loại bỏ, các mối quan hệ bị thay đổi, các bảng được tách ra, được cấu trúc lại hoặc được đổi tên).

Nếu các cột được thêm vào hoặc xóa khỏi bảng và các cột này không được khung nhìn yêu cầu, thì định nghĩa của khung nhìn không cần thay đổi. Nếu một bảng hiện có được sắp xếp lại hoặc chia nhỏ, một khung nhìn có thể cần được định nghĩa lại để đảm bảo tính chính xác của dữ liệu.

Trong trường hợp tách một bảng, bảng cũ có thể được tạo lại bằng cách định nghĩa khung nhìn từ việc nối các bảng mới, miễn là việc tách được thực hiện theo cách mà bảng ban đầu có thể được tạo lại. Chúng ta có thể đảm bảo rằng điều này có thể thực hiện được bằng cách đặt khóa chính vào cả hai bảng mới.

Ví dụ, nếu ban đầu bảng Client có dạng sau:

Client (clientNo, fName, lName, telNo, prefType, maxRent, eMail)

chúng ta có thể tách thành hai bảng mới:

ClientDetails (clientNo, fName, lName, telNo, eMail)

ClientRequests (clientNo, prefType, maxRent)

Người dùng vẫn có thể truy cập dữ liệu bằng cách sử dụng cấu trúc bảng cũ, cấu trúc này sẽ được tạo lại bằng cách định nghĩa khung nhìn Client bằng cách nối bảng ClientDetails với bảng ClientRequests, với clientNo là cột chung:

```
CREATE VIEW Client
AS
SELECT cd.clientNo, fName, lName, telNo, prefType,
       maxRent, eMail
  FROM ClientDetails cd, ClientRequests cr
 WHERE cd.clientNo = cr.clientNo;
```

- **Tính tức thời:** Các thay đổi đối với bất kỳ bảng cơ sở nào trong truy vấn định nghĩa sẽ được phản ánh ngay lập tức trong khung nhìn.
- **Cải thiện bảo mật:** Mỗi người dùng chỉ có thể được cấp đặc quyền để truy cập cơ sở dữ liệu thông qua một tập hợp nhỏ các khung nhìn chứa dữ liệu phù hợp với người dùng đó.
- **Giảm độ phức tạp:** Một khung nhìn có thể đơn giản hóa các truy vấn, bằng cách rút dữ liệu từ một số bảng thành một bảng duy nhất, từ đó chuyển các truy vấn nhiều bảng thành các truy vấn một bảng.
- **Tiện lợi:** Các khung nhìn có thể mang lại sự thuận tiện hơn cho người dùng vì người dùng chỉ được hiển thị phần cơ sở dữ liệu mà họ cần xem. Điều này cũng làm giảm độ phức tạp theo quan điểm của người dùng.
- **Tùy biến:** Khung nhìn cung cấp một phương pháp để tùy chỉnh giao diện của cơ sở dữ liệu, để những người dùng khác nhau có thể nhìn thấy cùng các bảng cơ sở nhưng theo những cách khác nhau.
- **Toàn vẹn dữ liệu:** Nếu mệnh đề WITH CHECK OPTION của lệnh CREATE VIEW được sử dụng, đảm bảo rằng không có hàng nào không thỏa mãn mệnh đề WHERE của truy vấn định nghĩa được thêm vào bất kỳ (các) bảng cơ sở.

### 3.3.7.2 Các hạn chế

Mặc dù các khung nhìn cung cấp nhiều lợi ích đáng kể, nhưng cũng tồn tại một số hạn chế nhất định.

- **Hạn chế cập nhật:** Trong mục 3.3.5, chúng ta đã chỉ ra rằng, trong một số trường hợp cụ thể, khung nhìn không thể cập nhật được.
- **Hạn chế cấu trúc:** Cấu trúc của một khung nhìn được xác định tại thời điểm tạo ra nó. Nếu truy vấn định nghĩa có dạng `SELECT * FROM ...`, thì dấu `*` để cập đến các cột của bảng cơ sở hiện diện khi khung nhìn được tạo. Nếu sau đó các cột được thêm vào bảng cơ sở, thì các cột này sẽ không xuất hiện trong khung nhìn, trừ khi khung nhìn bị xóa và tạo lại.
- **Hạn chế hiệu suất:** Có một sự trả giá về hiệu suất khi sử dụng khung nhìn. Trong một số trường hợp, điều này sẽ không đáng kể; trong các trường hợp khác, nó có thể tương đối nghiêm trọng. Ví dụ, một khung nhìn được định nghĩa bởi một truy vấn phức tạp, nhiều bảng thì sẽ có thể phải mất nhiều thời gian để xử lý, vì quá trình phân giải khung nhìn phải nối các bảng lại với nhau mỗi khi khung nhìn được tham chiếu.

### 3.3.8 Cụ thể hóa khung nhìn

Trong mục 3.3.3, chúng ta đã trình bày về một cách tiếp cận để xử lý các truy vấn trên một khung nhìn, trong đó truy vấn được sửa đổi thành truy vấn trên các bảng cơ sở bên dưới. Một bất lợi với cách tiếp cận này là thời gian cần thiết để thực hiện phân giải khung nhìn, đặc biệt nếu khung nhìn được truy cập thường xuyên. Một cách tiếp cận thay thế, được gọi là cụ thể hóa khung nhìn (view materialization), lưu khung nhìn dưới dạng bảng tạm thời trong cơ sở dữ liệu khi khung nhìn được truy vấn lần đầu tiên. Sau đó, các truy vấn trên khung nhìn được cụ thể hóa này có thể nhanh hơn nhiều so với việc tính toán lại khung nhìn cho mỗi lần tham chiếu đến.

Các khung nhìn được cụ thể hóa rất hữu ích trong các ứng dụng mới như kho dữ liệu (data warehouse), máy chủ sao chép (replication server), trực quan hóa dữ liệu (data visualization) và hệ thống di động (mobile system). Kiểm tra ràng buộc toàn vẹn và tối ưu hóa truy vấn cũng có thể được hưởng lợi từ các khung nhìn được cụ thể hóa. Khó khăn với cách tiếp cận này là duy trì tính tức thời của khung nhìn khi (các) bảng cơ sở đang được cập nhật. Quá trình cập nhật một khung nhìn được cụ thể hóa để đáp ứng với những thay đổi đối với dữ liệu cơ bản được gọi là duy trì khung nhìn (view maintenance). Mục đích cơ bản của duy trì khung nhìn là chỉ áp dụng những thay đổi cần thiết cho khung nhìn để duy trì tính tức thời của nó. Để rõ hơn vấn đề, hãy xem xét khung nhìn sau:

```
CREATE VIEW StaffPropRent (staffNo)
AS
SELECT DISTINCT staffNo
FROM PropertyForRent
WHERE branchNo = 'B003' AND rent > 400;
```

staffNo

SG37

SG14

Nếu chúng ta chèn một hàng vào bảng PropertyForRent với rent  $\leq 400$ , thì khung nhìn sẽ không thay đổi. Nếu chúng ta chèn hàng ('PG24', ..., 550, 'CO40', 'SG19', 'B003') vào bảng PropertyForRent, thì hàng đó cũng sẽ xuất hiện trong khung nhìn được cụ thể hóa. Tuy nhiên, nếu chúng ta chèn hàng ('PG54', ..., 450, 'CO89', 'SG37', 'B003') vào bảng PropertyForRent, thì không có hàng mới cần thêm vào khung nhìn được cụ thể hóa, bởi vì đã có hàng cho SG37 rồi. Lưu ý rằng trong ba trường hợp này, chúng ta có thể đưa ra quyết định có chèn hàng vào khung nhìn được cụ thể hóa hay không mà không cần truy cập vào bảng PropertyForRent bên dưới.

Nếu bây giờ chúng ta muốn xóa hàng mới ('PG24', ..., 550, 'CO40', 'SG19', 'B003') khỏi bảng PropertyForRent, thì hàng cũng sẽ bị xóa khỏi khung nhìn được cụ thể hóa. Tuy nhiên, nếu chúng ta muốn xóa hàng mới ('PG54', ..., 450, 'CO89', 'SG37', 'B003') khỏi bảng PropertyForRent, thì hàng tương ứng với SG37 sẽ không bị xóa khỏi khung nhìn được cụ thể hóa, do sự tồn tại của hàng cơ sở bên dưới tương ứng với bất động sản PG21. Trong hai trường hợp này, quyết định xóa hoặc giữ lại hàng trong khung nhìn được cụ thể hóa yêu cầu truy cập vào bảng cơ sở bên dưới PropertyForRent.

### 3.4 ĐIỀU KHIỂN TRUY CẬP

Chúng ta đã nói rằng DBMS phải cung cấp một cơ chế để đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập vào cơ sở dữ liệu (xem mục 2.2.2). Các DBMS hiện đại thường cung cấp một hoặc cả hai cơ chế ủy quyền sau:

- Điều khiển truy cập tùy quyền (Discretionary Access Control - DAC). Mỗi người dùng được cấp các quyền truy cập (hoặc đặc quyền - privilege) thích hợp trên các đối tượng cơ sở dữ liệu cụ thể. Thông thường, người dùng có được các đặc quyền nhất định khi họ tạo một đối tượng và có thể chuyển một số hoặc tất cả các đặc quyền này cho người dùng khác theo quyết định của họ. Mặc dù linh hoạt, nhưng loại cơ chế ủy quyền này có thể bị phá vỡ bởi một người dùng trái phép lừa người dùng được ủy quyền tiết lộ dữ liệu nhạy cảm.
- Điều khiển truy cập bắt buộc (Mandatory Access Control - MAC). Mỗi đối tượng cơ sở dữ liệu được gán một mức phân loại (classification level) nhất định (ví dụ như Top Secret, Secret, Confidential, Unclassified) và mỗi đối tượng (ví dụ như người dùng hoặc chương trình) được cấp một mức quyền hạn (clearance level) đã được thiết kế. Các mức phân loại tạo thành một thứ tự nghiêm ngặt (Top Secret > Secret > Confidential > Unclassified) và một đối tượng cần một mức quyền hạn cần thiết để đọc hoặc ghi một đối tượng cơ sở dữ liệu.

SQL chuẩn chỉ hỗ trợ DAC thông qua các câu lệnh GRANT và REVOKE.

### 3.4.1 Các khái niệm

- **Định danh ủy quyền và Quyền sở hữu**

Định danh ủy quyền (authorization identifier) được sử dụng để thiết lập danh tính của người dùng. Mỗi người dùng cơ sở dữ liệu được DBA chỉ định một định danh ủy quyền. Thông thường, định danh có một mật khẩu được liên kết, vì các lý do bảo mật. Mọi câu lệnh SQL được thực thi bởi DBMS đều được thực hiện thay mặt cho một người dùng cụ thể. Định danh ủy quyền được sử dụng để xác định đối tượng cơ sở dữ liệu nào mà người dùng có thể tham chiếu và những thao tác nào có thể được thực hiện trên các đối tượng đó.

Mỗi đối tượng được tạo trong SQL có một chủ sở hữu. Chủ sở hữu (owner) được xác định bằng định danh ủy quyền được xác định trong mệnh đề AUTHORIZATION của lược đồ chứa đối tượng. Chủ sở hữu ban đầu là người duy nhất có thể biết về sự tồn tại của đối tượng và do đó, có quyền thực hiện bất kỳ thao tác nào trên đối tượng.

- **Đặc quyền**

Đặc quyền (privilege) là những hành động mà người dùng được phép thực hiện trên một bảng cơ sở hoặc khung nhìn nhất định. Các đặc quyền được xác định bởi SQL chuẩn là:

- SELECT - đặc quyền truy xuất dữ liệu từ một bảng;
- INSERT - đặc quyền chèn các hàng mới vào bảng;
- UPDATE - đặc quyền sửa đổi các hàng dữ liệu trong bảng;
- DELETE - đặc quyền xóa các hàng dữ liệu khỏi bảng;
- REFERENCES - đặc quyền tham chiếu đến các cột của một bảng được đặt tên;

Các đặc quyền INSERT và UPDATE có thể bị hạn chế đối với các cột cụ thể của bảng, cho phép thay đổi các cột này nhưng không cho phép thay đổi bất kỳ cột nào khác. Tương tự, đặc quyền REFERENCES có thể bị hạn chế đối với các cột cụ thể của bảng, cho phép các cột này được tham chiếu trong các ràng buộc, chẳng hạn như ràng buộc kiểm tra và ràng buộc khóa ngoại, khi tạo một bảng khác, nhưng không cho phép các cột khác được tham chiếu.

Khi người dùng tạo bảng bằng câu lệnh CREATE TABLE, người đó sẽ tự động trở thành chủ sở hữu của bảng và nhận đầy đủ các đặc quyền đối với bảng. Những người dùng khác ban đầu không có đặc quyền trên bảng mới được tạo. Để họ có thể truy cập vào bảng, chủ sở hữu phải cấp cho họ các đặc quyền cần thiết một cách rõ ràng bằng cách sử dụng câu lệnh GRANT.

Khi người dùng tạo khung nhìn bằng câu lệnh CREATE VIEW, người đó tự động trở thành chủ sở hữu của khung nhìn, nhưng không nhất thiết phải nhận được đầy đủ các đặc quyền trên khung nhìn. Để tạo khung nhìn, người dùng phải có đặc quyền SELECT trên tất cả các bảng tạo nên khung nhìn và đặc quyền REFERENCES trên các cột được đặt tên của khung nhìn.

### 3.4.2 Cấp quyền cho người dùng (GRANT)

Câu lệnh GRANT được sử dụng để cấp các đặc quyền trên các đối tượng cơ sở dữ liệu cho người dùng cụ thể. Thông thường, câu lệnh GRANT được chủ sở hữu của bảng sử dụng để cấp cho người dùng khác quyền truy cập vào dữ liệu. Định dạng của câu lệnh GRANT là:

```
GRANT {PrivilegeList | ALL PRIVILEGES}
ON      ObjectName
TO      {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

PrivilegeList bao gồm một hoặc nhiều đặc quyền sau, được phân tách bằng dấu phẩy:

```
SELECT
DELETE
INSERT      [(columnName [,...])]
UPDATE      [(columnName [,...])]
REFERENCES  [(columnName [,...])]
```

Để thuận tiện, câu lệnh GRANT cho phép sử dụng từ khóa ALL PRIVILEGES để cấp tất cả các đặc quyền cho người dùng thay vì phải chỉ định năm đặc quyền riêng lẻ. Nó cũng cung cấp từ khóa PUBLIC để cho phép cấp quyền truy cập cho tất cả người dùng được ủy quyền hiện tại và tương lai, không chỉ cho những người dùng hiện được biết đến với DBMS. ObjectName có thể là tên của bảng cơ sở, khung nhìn, miền, bộ ký tự,...

Mệnh đề WITH GRANT OPTION cho phép (các) người dùng trong AuthorizationIdList chuyển các đặc quyền mà họ đã được cấp trên đối tượng được đặt tên cho những người dùng khác. Nếu những người dùng này chuyển một đặc quyền khi chỉ định WITH GRANT OPTION, những người dùng nhận được đặc quyền có thể cấp đặc quyền đó cho những người dùng khác. Nếu từ khóa này không được chỉ định, (các) người dùng nhận sẽ không thể chuyển các đặc quyền cho những người dùng khác.

#### Ví dụ 3.5 GRANT tất cả đặc quyền

*Cấp cho người dùng có định danh ủy quyền Manager tất cả các đặc quyền trên bảng Staff.*

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager
WITH GRANT OPTION;
```

Người dùng được xác định là Manager hiện có thể truy xuất các hàng của bảng Staff và cũng có thể chèn, cập nhật và xóa dữ liệu bảng này. Người dùng này cũng có thể tham chiếu bảng Staff và tất cả các cột của Staff trong bất kỳ bảng nào mà người này tạo ra sau đó. Chúng ta cũng chỉ định từ khóa WITH GRANT OPTION, để người dùng Manager có thể chuyển các đặc quyền này cho những người dùng khác.

### Ví dụ 3.6 GRANT các đặc quyền cụ thể

Cấp cho các người dùng có định danh ủy quyền Personnel và Director các đặc quyền SELECT và UPDATE trên cột salary của bảng Staff.

```
GRANT SELECT, UPDATE (salary)  
ON Staff  
TO Personnel, Director;
```

Chúng ta đã bỏ qua từ khóa WITH GRANT OPTION, để người dùng Personnel và Director không thể chuyển các đặc quyền này cho người dùng khác.

### Ví dụ 3.7 GRANT các đặc quyền cụ thể là PUBLIC

Cấp cho tất cả người dùng đặc quyền SELECT trên bảng Branch.

```
GRANT SELECT  
ON Branch  
TO PUBLIC;
```

Việc sử dụng từ khóa PUBLIC có nghĩa là tất cả người dùng (hiện tại và trong tương lai) đều có thể truy xuất tất cả dữ liệu trong bảng Branch.

### 3.4.3 Thu hồi quyền của người dùng (REVOKE)

Câu lệnh REVOKE được sử dụng để thu hồi các đặc quyền đã được cấp bằng câu lệnh GRANT. Câu lệnh REVOKE có thể thu hồi tất cả hoặc một số đặc quyền đã được cấp trước đó cho người dùng. Định dạng của câu lệnh là:

```
REVOKE [GRANT OPTION FOR] {PrivilegeList | ALL PRIVILEGES}  
ON ObjectName  
FROM {AuthorizationIdList | PUBLIC} [RESTRICT | CASCADE]
```

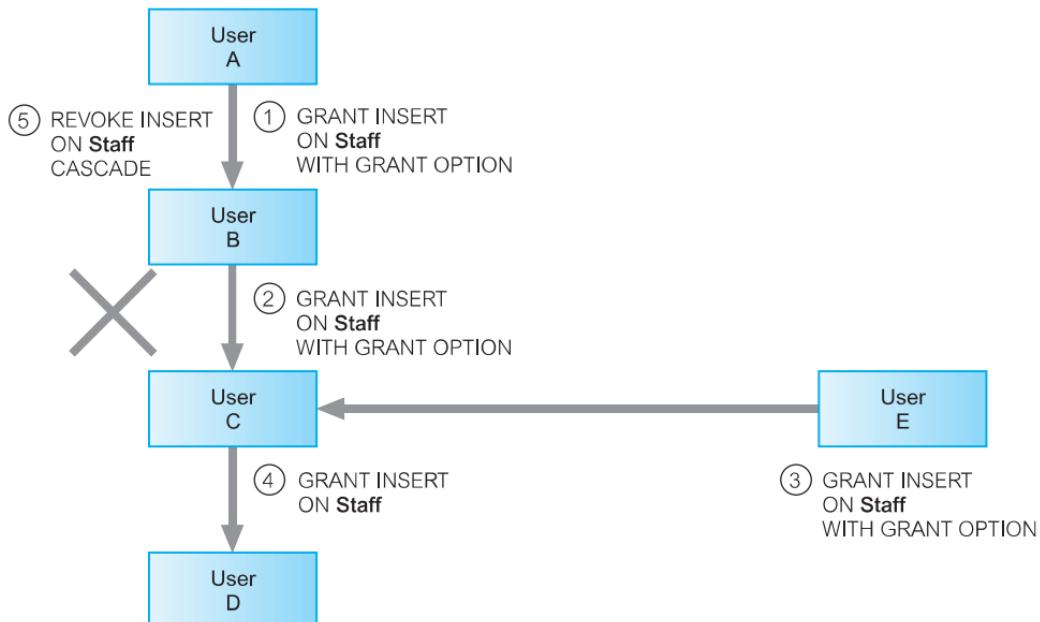
Từ khóa ALL PRIVILEGES để cập đến tất cả các đặc quyền được cấp cho người dùng bởi người dùng thu hồi các đặc quyền. Mệnh đề tùy chọn GRANT OPTION FOR cho phép các đặc quyền được chuyển dựa trên mệnh đề WITH GRANT OPTION của câu lệnh GRANT được thu hồi tách biệt với chính các đặc quyền đó.

Các từ khóa RESTRICT và CASCADE hoạt động chính xác như trong câu lệnh DROP TABLE. Bởi vì các đặc quyền được yêu cầu để tạo các đối tượng nhất định, việc thu hồi một đặc quyền có thể loại bỏ quyền hạn đã cho phép tạo đối tượng (một đối tượng như vậy được cho là bị bỏ rơi). Câu lệnh REVOKE không thành công nếu nó dẫn đến một đối tượng bị bỏ rơi, chẳng hạn như một khung nhìn, trừ khi từ khóa CASCADE đã được chỉ định. Nếu CASCADE được chỉ định, một câu lệnh DROP thích hợp được áp dụng cho bất kỳ khung nhìn, miền, hoặc ràng buộc nào bị bỏ rơi.

Những đặc quyền đã được cấp cho người dùng này bởi những người dùng khác không bị ảnh hưởng bởi câu lệnh REVOKE này. Do đó, nếu người dùng khác đã cấp cho người dùng đặc quyền bị thu hồi, quyền của người dùng khác vẫn cho phép người dùng đó truy cập vào bảng.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Ví dụ, trong Hình 3.5 User A cấp cho User B đặc quyền INSERT trên bảng Staff WITH GRANT OPTION (bước 1). User B chuyển đặc quyền này cho User C (bước 2). Sau đó, User C nhận được đặc quyền tương tự từ User E (bước 3). User C sau đó chuyển đặc quyền cho User D (bước 4). Khi User A thu hồi đặc quyền INSERT từ User B (bước 5), đặc quyền đó không thể bị thu hồi từ User C, vì User C cũng đã nhận được đặc quyền từ User E. Nếu User E không cấp cho User C đặc quyền này, thì việc thu hồi sẽ đã chuyển sang User C và User D.



Hình 3.5 Ảnh hưởng của REVOKE

### Ví dụ 3.8 REVOKE các đặc quyền cụ thể là PUBLIC

*Thu hồi đặc quyền SELECT trên bảng Branch từ tất cả người dùng.*

```
REVOKE SELECT  
ON Branch  
FROM PUBLIC;
```

### Ví dụ 3.9 REVOKE các đặc quyền cụ thể từ người dùng được đặt tên.

*Thu hồi tất cả các đặc quyền đã trao cho Director đối với bảng Staff.*

```
REVOKE ALL PRIVILEGES  
ON Staff  
FROM Director;
```

Điều này tương đương với REVOKE SELECT . . . , vì đây là đặc quyền duy nhất được trao cho Director.

### 3.5 NGÔN NGỮ LẬP TRÌNH SQL

SQL hiện nay là một ngôn ngữ lập trình đầy đủ và chúng ta sẽ tìm hiểu về một số cấu trúc lập trình trong phần này. Các phần mở rộng của SQL chuẩn được gọi là SQL/PSM (Persistent Stored Modules); tuy nhiên, để cho các nội dung trình bày tiếp theo trở nên cụ thể và dễ hiểu hơn, chúng ta chủ yếu minh họa dựa trên ngôn ngữ lập trình T-SQL.

#### T-SQL

Phần mở rộng ngôn ngữ truy vấn SQL chuẩn và thực thi trên hệ quản trị cơ sở dữ liệu SQL Server của Microsoft.

T-SQL (Transact-SQL) có các khái niệm tương tự như các ngôn ngữ lập trình hiện đại, chẳng hạn như khai báo và gán giá trị cho biến, cấu trúc điều khiển, xử lý ngoại lệ và mô-đun hóa. T-SQL là một ngôn ngữ có cấu trúc khối (block-structured): các khối có thể hoàn toàn tách biệt hoặc lồng vào nhau. Các đơn vị cơ bản cấu thành một chương trình T-SQL là các thủ tục, hàm và các khối ẩn danh (không được đặt tên).

#### 3.5.1 Biến – Lô – Lệnh GO

Các biến (variable) được sử dụng để lưu trữ tạm thời các giá trị dữ liệu để sử dụng sau này trong cùng một lô mà chúng đã được khai báo. Một lô (batch) là một hoặc nhiều câu lệnh T-SQL được gửi đến SQL Server để thực hiện như một đơn vị duy nhất.

Sử dụng lệnh DECLARE để khai báo một hoặc nhiều biến và sử dụng lệnh SET để gán một giá trị cho một biến duy nhất. Ví dụ, đoạn mã sau khai báo một biến có tên là @i có kiểu dữ liệu INT và gán cho nó giá trị 10:

```
DECLARE @i AS INT;
SET @i = 10;
```

Ngoài ra, chúng ta có thể khai báo và khởi tạo biến trong cùng một câu lệnh, như sau:

```
DECLARE @i AS INT = 10;
```

Khi gán một giá trị cho một biến vô hướng, giá trị đó phải là kết quả của một biểu thức vô hướng. Biểu thức cũng có thể là một truy vấn vô hướng. Ví dụ, đoạn mã sau khai báo một biến có tên @sName và gán cho nó kết quả của truy vấn trả về tên đầy đủ của nhân viên có staffNo là SL21:

```
DECLARE @sName AS NCHAR(21);
SET @sName =
(
    SELECT fName + ' ' + lName
    FROM dbo.Staff
    WHERE staffNo = 'SL21'
);
SELECT @sName AS staffName;
```

Đoạn mã này trả về kết quả như sau:

staffName

John White

Câu lệnh SET chỉ có thể thao tác với một biến tại một thời điểm, vì vậy, nếu cần gán giá trị cho nhiều biến, chúng ta cần sử dụng nhiều câu lệnh SET. Cách tiếp cận này có thể liên quan đến chi phí không cần thiết khi cần lấy nhiều giá trị thuộc tính từ cùng một hàng. Ví dụ, đoạn mã sau sử dụng hai câu lệnh SET riêng biệt để gán họ và tên của nhân viên có staffNo là SL21 cho hai biến riêng biệt:

```
DECLARE @firstname AS NCHAR(10),
        @lastname AS NCHAR(10);
SET @firstname =
(
    SELECT fName FROM dbo.Staff WHERE staffNo = 'SL21'
);
SET @lastname =
(
    SELECT lName FROM dbo.Staff WHERE staffNo = 'SL21'
);
SELECT @firstname AS firstName,
       @lastname AS lastName;
```

Đoạn mã này trả về kết quả như sau:

firstName	lastName
John	White

T-SQL cũng hỗ trợ câu lệnh SELECT gán không theo tiêu chuẩn được sử dụng để truy vấn dữ liệu và gán nhiều giá trị thu được từ cùng một hàng cho nhiều biến như sau:

```
DECLARE @firstname AS NCHAR(10),
        @lastname AS NCHAR(10);
SELECT @firstname = fName,
       @lastname = lName
FROM dbo.Staff
WHERE staffNo = 'SL21';
SELECT @firstname AS firstName,
       @lastname AS lastName;
```

Tuy nhiên, lưu ý rằng nếu truy vấn có nhiều hơn một hàng đủ điều kiện, mã sẽ không bị lỗi. Việc gán diễn ra trên mỗi hàng đủ điều kiện, và với mỗi hàng được truy cập, các giá trị của hàng hiện tại sẽ ghi đè lên các giá trị đang có trong các biến. Khi phép gán SELECT kết thúc, các giá trị trong các biến là những giá trị từ hàng cuối cùng mà SQL Server đã truy cập. Ví dụ, phép gán SELECT sau đây có hai hàng đủ điều kiện:

```
DECLARE @sName AS NCHAR(21);
SELECT @sName = fName + N' ' + lName
FROM dbo.Staff
WHERE branchNo = 'B005';
SELECT @sName AS staffName;
```

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Thông tin nhân viên kết thúc trong biến sau khi phép gán SELECT kết thúc tùy thuộc vào thứ tự mà SQL Server sẽ truy cập vào các hàng đó - và người dùng không có quyền kiểm soát đối với thứ tự này. Một trong những kết quả nhận được sau khi thực thi đoạn mã trên:

staffName

Julie Lee

Câu lệnh SET an toàn hơn so với phép gán SELECT vì nó yêu cầu sử dụng truy vấn vô hướng để lấy dữ liệu từ một bảng. Hãy nhớ rằng một truy vấn vô hướng không thực hiện thành công nếu nó trả về nhiều hơn một giá trị. Ví dụ, đoạn mã sau không thành công:

```
DECLARE @sName AS NCHAR(21);
SET @sName =
(
    SELECT fName + ' ' + lName
    FROM dbo.Staff
    WHERE branchNo = 'B005'
);
SELECT @sName AS staffName;
```

Vì biến không được gán giá trị nên nó vẫn là NULL, đây là giá trị mặc định cho các biến không được khởi tạo. Đoạn mã trên trả về kết quả sau:

```
Msg 512, Level 16, State 1, Line 2
Subquery returned more than 1 value. This is not permitted
when the subquery follows =, !=, <, <= , >, >= or when the
subquery is used as an expression.
```

(1 row(s) affected)

Lô (batch) là một hoặc nhiều câu lệnh T-SQL được ứng dụng máy khách gửi đến SQL Server để thực thi dưới dạng một đơn vị. Lô trải qua quá trình phân tích cú pháp (kiểm tra cú pháp), phân giải/liên kết (kiểm tra sự tồn tại của các đối tượng và cột được tham chiếu, kiểm tra quyền) và tối ưu hóa dưới dạng một đơn vị.

Đừng nhầm lẫn giữa giao dịch và lô. Một giao dịch là một đơn vị nguyên tử của công việc. Một lô có thể có nhiều giao dịch và một giao dịch có thể được gửi thành nhiều phần dưới dạng nhiều lô. Khi một giao dịch bị hủy (cancel) hoặc khôi phục (roll back), SQL Server sẽ hoàn tác một phần hoạt động đã diễn ra kể từ khi bắt đầu giao dịch, bất kể lô bắt đầu từ đâu.

Các giao diện lập trình ứng dụng máy khách (API) chẳng hạn như ADO.NET cung cấp các phương thức để gửi một lô mã tới SQL Server để thực thi. Các tiện ích SQL Server chẳng hạn như SQL Server Management Studio (SSMS), SQLCMD và OSQI cung cấp một lệnh công cụ máy khách được gọi là GO báo hiệu sự kết thúc của một lô.

Lưu ý rằng lệnh GO là lệnh công cụ máy khách, không phải lệnh máy chủ T-SQL.

Một lô là một tập hợp các lệnh được phân tích cú pháp và thực thi dưới dạng một đơn vị. Nếu quá trình phân tích cú pháp thành công, SQL Server sẽ cố gắng thực hiện lô đó. Trong trường hợp có lỗi cú pháp trong lô, toàn bộ lô sẽ không được gửi tới SQL Server để thực thi. Ví dụ, đoạn mã sau đây có ba lô, lô thứ hai có lỗi cú pháp (*FROM* thay vì *FROM* trong truy vấn thứ hai):

```
-- Valid batch
PRINT 'First batch';
USE DreamHome;
GO
-- Invalid batch
PRINT 'Second batch';
SELECT clientNo FROM dbo.Client;
SELECT branchNo FOM dbo.Branch;
GO
-- Valid batch
PRINT 'Third batch';
SELECT staffNo
FROM dbo.Staff;
```

Vì lô thứ hai có lỗi cú pháp nên toàn bộ lô không được gửi tới SQL Server để thực thi. Lô đầu tiên và lô thứ ba vượt qua xác thực cú pháp và do đó được gửi để thực thi. Mã trên tạo ra kết quả sau đây cho thấy rằng toàn bộ lô thứ hai không được thực thi:

```
First batch
Msg 102, Level 15, State 1, Line 8
Incorrect syntax near 'dbo'.
Third batch
```

(6 row(s) affected)

Một biến là cục bộ đối với lô mà trong đó nó được định nghĩa. Nếu tham chiếu đến một biến được định nghĩa trong lô khác, chúng ta sẽ gặp lỗi cho biết biến đó chưa được định nghĩa. Ví dụ, đoạn mã sau khai báo một biến và xuất ra nội dung của nó trong một lô, sau đó lại cố gắng xuất ra nội dung của biến này từ một lô khác:

```
DECLARE @i AS INT;
SET @i = 10;
-- Succeeds
PRINT @i;
GO
-- Fails
PRINT @i;
```

Tham chiếu đến biến trong câu lệnh PRINT đầu tiên là hợp lệ vì xuất hiện trong cùng lô nơi biến được khai báo, nhưng tham chiếu thứ hai không hợp lệ. Do đó, câu lệnh PRINT đầu tiên trả về giá trị của biến (10), trong khi câu lệnh thứ hai không thành công.

Kết quả được trả về từ mã trên:

10

```
Msg 137, Level 15, State 2, Line 7  
Must declare the scalar variable "@i".
```

Các câu lệnh sau không thể kết hợp với các câu lệnh khác trong cùng một lô: CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE SCHEMA, CREATE TRIGGER, và CREATE VIEW. Ví dụ, đoạn mã sau có câu lệnh DROP sau là câu lệnh CREATE VIEW trong cùng một lô và do đó không hợp lệ:

```
DROP VIEW IF EXISTS dbo.MyView;  
CREATE VIEW MyView  
AS  
SELECT city AS City,  
       COUNT(*) AS numOfProperty  
FROM dbo.PropertyForRent  
GROUP BY City;  
GO
```

Thực thi đoạn mã trên sẽ tạo ra lỗi sau:

```
Msg 111, Level 15, State 1, Line 2  
'CREATE VIEW' must be the first statement in a query batch.
```

Để khắc phục lỗi này, hãy tách các câu lệnh DROP VIEW và CREATE VIEW thành các lô khác nhau bằng cách thêm lệnh GO sau câu lệnh DROP VIEW.

Một lô là một đơn vị để phân giải (còn được gọi là ràng buộc - binding). Điều này có nghĩa là việc kiểm tra sự tồn tại của các đối tượng và cột diễn ra ở cấp độ lô. Hãy nhớ thực tế này khi đang thiết kế phạm vi lô. Khi áp dụng các thay đổi lược đồ cho một đối tượng và cố gắng thao tác đối tượng dữ liệu trong cùng một lô, SQL Server có thể chưa biết về các thay đổi lược đồ và không thực hiện được câu lệnh thao tác dữ liệu, phát sinh một lỗi phân giải. Chúng ta sẽ xem xét vấn đề này thông qua một ví dụ và sau đó đề xuất các phương pháp tốt nhất.

Thực thi đoạn mã sau để tạo một bảng có tên là T1 với một cột có tên là col1:

```
DROP TABLE IF EXISTS dbo.T1;  
CREATE TABLE dbo.T1(col1 INT);
```

Tiếp theo, thêm một cột có tên là col2 vào T1 và truy vấn cột mới trong cùng một lô:

```
ALTER TABLE dbo.T1 ADD col2 INT;  
SELECT col1, col2 FROM dbo.T1;
```

Mặc dù mã có vẻ hoàn toàn hợp lệ, nhưng lô này không thành công trong giai đoạn phân giải với lỗi sau:

```
Msg 207, Level 16, State 1, Line 130  
Invalid column name 'col2'.
```

Tại thời điểm câu lệnh SELECT được phân giải, T1 chỉ có cột col2 và việc tham chiếu đến cột col2 đã gây ra lỗi.

Một phương pháp hay nhất chúng ta có thể làm để tránh những vấn đề như vậy là tách biệt các câu lệnh DDL và DML thành các lô khác nhau, như trong ví dụ sau:

```
ALTER TABLE dbo.T1 ADD col2 INT;
GO
SELECT col1, col2 FROM dbo.T1;
```

Lệnh GO không thực sự là một lệnh T-SQL; nó thực sự là một lệnh được sử dụng bởi các công cụ máy khách của SQL Server, chẳng hạn như SSMS, để biểu thị sự kết thúc của một lô. Lệnh này hỗ trợ một đối số cho biết chúng ta muốn thực hiện lô bao nhiêu lần. Để xem lệnh GO hoạt động như thế nào khi có đối số, trước tiên hãy tạo bảng T1 bằng mã sau:

```
DROP TABLE IF EXISTS dbo.T1;
CREATE TABLE dbo.T1(col1 INT IDENTITY);
```

Cột col1 tự động nhận giá trị từ thuộc tính identity. Tiếp theo, hãy chạy đoạn mã sau để hiển thị kết quả được tạo bởi các câu lệnh DML cho biết có bao nhiêu hàng bị ảnh hưởng:

```
SET NOCOUNT ON;
```

Cuối cùng, hãy chạy đoạn mã sau để định nghĩa một lô với câu lệnh INSERT DEFAULT VALUES và thực hiện lô đó 100 lần:

```
INSERT INTO dbo.T1 DEFAULT VALUES;
GO 100
SELECT * FROM dbo.T1;
```

Truy vấn trả về 100 hàng có giá trị từ 1 đến 100 trong col1.

### 3.5.2 Con trỏ

SQL và T-SQL cũng hỗ trợ một đối tượng được gọi là con trỏ (cursor) có thể sử dụng để xử lý các hàng từ kết quả của một truy vấn tại một thời điểm và theo thứ tự được yêu cầu.

Lưu ý rằng lựa chọn mặc định là sử dụng các truy vấn dựa trên tập hợp; chỉ khi chúng ta có lý do thuyết phục để làm khác thì mới nên cân nhắc sử dụng con trỏ. Khuyến nghị này dựa trên một số yếu tố, bao gồm:

- Đầu tiên và quan trọng nhất, khi sử dụng con trỏ, chúng ta gần như đi ngược lại mô hình quan hệ, mô hình dựa trên lý thuyết tập hợp.
- Thao tác với từng bản ghi được thực hiện bởi con trỏ có phát sinh chi phí hoạt động. Khi so sánh truy vấn dựa trên tập hợp và mã lệnh con trỏ, mã lệnh con trỏ thường chậm hơn nhiều lần so với truy vấn dựa trên tập hợp.
- Với con trỏ, chúng ta đang đưa ra giải pháp tường minh - nói cách khác, phải chịu trách nhiệm xác định cách xử lý dữ liệu (khai báo con trỏ, mở nó, lặp qua các bản ghi con trỏ, đóng con trỏ và giải phóng con trỏ - thu hồi bộ nhớ đã cấp phát). Do đó, các giải pháp con trỏ có xu hướng dài hơn, khó đọc hơn và khó bảo trì hơn so với các giải pháp dựa trên tập hợp.

Đối với hầu hết mọi người, không đơn giản để nghĩ ngay đến các thuật ngữ quan hệ khi bắt đầu học SQL, hầu hết sẽ trực quan hơn khi nghĩ về con trỏ - xử lý một bản ghi tại một thời điểm theo một thứ tự nhất định. Do đó, con trỏ được sử dụng rộng rãi và trong đa số các trường hợp, chúng bị lạm dụng; nghĩa là, chúng được sử dụng ngay cả khi tồn tại nhiều giải pháp dựa trên tập hợp tốt hơn.

Mọi quy tắc đều có ngoại lệ. Một ví dụ là khi cần áp dụng một tác vụ nhất định cho từng hàng từ một số bảng hoặc khung nhìn. Chẳng hạn, chúng ta có thể cần thực thi một số tác vụ quản trị cho từng chỉ mục hoặc bảng trong cơ sở dữ liệu của mình. Trong trường hợp như vậy, sẽ hợp lý khi sử dụng con trỏ để lặp qua từng tên bảng hoặc chỉ mục và thực hiện tác vụ liên quan cho từng tên đó.

Một ví dụ khác về ngữ cảnh nên xem xét sử dụng con trỏ là khi giải pháp dựa trên tập hợp hoạt động không tốt và chúng ta đã nỗ lực tối ưu hóa phương pháp dựa trên tập hợp những vẫn không đạt được hiệu quả như mong muốn. Như đã đề cập, các giải pháp dựa trên tập hợp có xu hướng nhanh hơn nhiều, nhưng trong một số trường hợp ngoại lệ, giải pháp con trỏ sẽ nhanh hơn. Giải pháp duyệt qua vòng lặp, chẳng hạn như giải pháp dựa trên con trỏ, thường là giải pháp tối ưu.

Làm việc với con trỏ thường bao gồm các bước sau:

1. Khai báo con trỏ dựa trên truy vấn.
2. Mở con trỏ.
3. Tìm nạp các giá trị thuộc tính từ bản ghi con trỏ đầu tiên vào các biến.
4. Trong khi chưa đạt đến cuối con trỏ (nghĩa là giá trị của biến hệ thống @@FETCH\_STATUS là 0), hãy lặp qua các bản ghi con trỏ; trong mỗi lần lặp, thực hiện xử lý cần thiết cho hàng hiện tại, sau đó tìm nạp các giá trị thuộc tính từ hàng tiếp theo vào các biến.
5. Đóng con trỏ.
6. Giải phóng con trỏ.

Ví dụ sau sử dụng con trỏ tính tổng số lượt xem bất động sản của từng khách hàng theo mỗi tháng từ bảng Viewing:

```
SET NOCOUNT ON;
DECLARE @Result AS TABLE
(
    clientNo NCHAR(10),
    viewMonth INT,
    viewNum INT,
    viewRun INT,
    PRIMARY KEY (
        clientNo,
        viewMonth
    )
);
```

```
DECLARE @clientNo AS NCHAR(10),
        @prvclientNo AS NCHAR(10),
        @viewMonth AS INT,
        @viewNum AS INT,
        @viewRun AS INT;
DECLARE C CURSOR FAST_FORWARD /* read only, forward only */
FOR
SELECT clientNo,
       MONTH(viewDate),
       COUNT(*)
FROM Viewing
GROUP BY MONTH(viewDate),
         clientNo
ORDER BY clientNo,
         MONTH(viewDate);
OPEN C;
FETCH NEXT FROM C
INTO @clientNo,
      @viewMonth,
      @viewNum;
SELECT @prvclientNo = @clientNo,
       @viewRun = 0;
WHILE @@FETCH_STATUS = 0
BEGIN
    IF @clientNo <> @prvclientNo
        SELECT @prvclientNo = @clientNo,
               @viewRun = 0;
    SET @viewRun = @viewRun + @viewNum;
    INSERT INTO @Result
    VALUES
    (@clientNo, @viewMonth, @viewNum, @viewRun);
    FETCH NEXT FROM C
    INTO @clientNo, @viewMonth, @viewNum;
END;
CLOSE C;
DEALLOCATE C;
SELECT clientNo,
       CONVERT(VARCHAR(7), viewMonth, 121) AS viewMonth,
       viewNum,
       viewRun
FROM @Result
ORDER BY clientNo,
         viewMonth;
```

Đoạn mã trên khai báo một con trỏ dựa trên một truy vấn trả về các hàng từ bảng Viewing được sắp xếp theo số khách hàng và tháng xem bất động sản (giả sử dữ liệu trong bảng Viewing được lưu trữ theo từng năm), đồng thời duyệt qua từng bản ghi một. Đoạn mã này theo dõi tổng số lượt xem hiện tại thông qua một biến có tên @viewRun với giá trị được đặt lại mỗi khi tìm thấy một khách hàng mới. Đối với mỗi hàng, mã tính toán tổng số lượt hiện tại bằng cách thêm số lượt xem của tháng hiện tại (@viewNum) vào @viewRun và chèn một hàng gồm số khách hàng, tháng xem bất động sản, số lượt xem của tháng hiện tại và tổng số lượt xem vào một biến bảng có tên @Result. Khi mã xử lý xong tất cả các bản ghi con trỏ, sẽ truy vấn biến bảng để hiển thị các thông tin tổng hợp.

Kết quả trả về của đoạn mã trên, được hiển thị như sau:

clientNo	viewMonth	viewNum	viewRun
CR56	4	1	1
CR56	5	2	3
CR62	5	1	1
CR76	4	1	1

### 3.5.3 Bảng tạm – Biến bảng – Kiểu bảng

Khi cần lưu trữ tạm thời dữ liệu trong các bảng, trong một số trường hợp nhất định, người dùng có thể không muốn làm việc với các bảng cố định. Giả sử cần dữ liệu chỉ hiển thị với phiên hiện tại hoặc thậm chí chỉ với lô hiện tại. Chẳng hạn, cần lưu trữ dữ liệu tạm thời trong quá trình xử lý dữ liệu, như trong ví dụ về con trỏ ở phần trước. Một trường hợp khác mà người dùng sử dụng bảng tạm là khi họ không có quyền tạo bảng cố định trong cơ sở dữ liệu.

SQL Server hỗ trợ ba kiểu bảng tạm: bảng tạm cục bộ, bảng tạm toàn cục và biến bảng.

#### 3.5.3.1 Bảng tạm cục bộ

Bảng tạm cục bộ được tạo bằng cách đặt tên cho nó với một dấu thăng làm tiền tố, chẳng hạn như #T1. Tất cả ba loại bảng tạm đều được tạo trong cơ sở dữ liệu tempdb.

Một bảng tạm cục bộ chỉ hiển thị với phiên đã tạo ra nó, ở cấp độ tạo và tất cả các cấp độ bên trong ngăn xếp lời gọi (các thủ tục lưu trữ, trình kích hoạt và lô động). Bảng tạm cục bộ bị SQL Server tự động hủy khi mức tạo trong ngăn xếp lời gọi vượt quá phạm vi. Ví dụ, giả sử một thủ tục lưu trữ có tên là Proc1 gọi một thủ tục có tên là Proc2, thủ tục này lại gọi một thủ tục có tên là Proc3, và thủ tục này lại tiếp tục gọi một thủ tục có tên là Proc4. Proc2 tạo một bảng tạm gọi là #T1 trước khi gọi Proc3. Bảng #T1 hiển thị với Proc2, Proc3 và Proc4 nhưng không hiển thị với Proc1 và nó sẽ bị SQL Server tự động hủy khi Proc2 kết thúc. Nếu bảng tạm được tạo trong một lô đặc biệt ở mức lồng ngoài cùng của phiên (nói cách khác, khi giá trị của biến @@NESTLEVEL là 0), thì bảng đó cũng hiển thị với tất cả các lô tiếp theo và chỉ bị hủy tự động bởi SQL Server khi phiên tạo ra nó bị ngắt kết nối.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Một tình huống rõ ràng mà các bảng tạm cục bộ hữu ích là khi chúng ta có một quy trình cần lưu trữ các kết quả trung gian tạm thời - chẳng hạn như trong một vòng lặp - và sau đó truy vấn dữ liệu.

Một tình huống khác là khi chúng ta cần truy cập nhiều lần vào kết quả của một quá trình xử lý tốn kém. Ví dụ, chúng ta cần nối các bảng Orders và OrderDetails để tổng hợp số lượng đơn đặt hàng theo năm đặt hàng và nối hai thể hiện của dữ liệu đã tổng hợp để so sánh tổng số lượng của mỗi năm với năm trước. Các bảng Orders và OrderDetails trong cơ sở dữ liệu mẫu rất nhỏ, nhưng trong các tình huống thực tế, các bảng đó có thể có hàng triệu hàng. Một tùy chọn là sử dụng biểu thức bảng, nhưng hãy nhớ rằng biểu thức bảng là ảo. Hoạt động tốn kém liên quan đến việc quét tất cả dữ liệu, nối các bảng Orders và OrderDetails, cũng như tổng hợp dữ liệu sẽ phải thực hiện hai lần với các biểu thức bảng. Thay vào đó, thật hợp lý khi chỉ thực hiện tất cả các hoạt động tốn kém một lần - lưu trữ kết quả vào một bảng tạm cục bộ - và sau đó nối hai thể hiện của bảng tạm, đặc biệt do kết quả của hoạt động tốn kém là một tập hợp nhỏ - chỉ có một hàng cho mỗi năm đặt hàng.

Đoạn mã sau minh họa tình huống trên bằng cách sử dụng bảng tạm cục bộ:

```
GO
CREATE TABLE #MyOrderTotalsByYear
(
    orderYear INT NOT NULL PRIMARY KEY,
    qty INT NOT NULL
);
INSERT INTO #MyOrderTotalsByYear
(
    orderYear,
    qty
)
SELECT YEAR(O.orderDate) AS orderYear,
       SUM(OD.qty) AS qty
FROM Orders AS O
    INNER JOIN OrderDetails AS OD
        ON OD.orderNo = O.orderNo
    GROUP BY YEAR(orderDate);
SELECT Cur.orderYear,
       Cur.qty AS curYearQty,
       Prv.qty AS prvYearQty
FROM #MyOrderTotalsByYear AS Cur
    LEFT OUTER JOIN #MyOrderTotalsByYear AS Prv
        ON Cur.orderYear = Prv.orderYear + 1;
```

Đoạn mã trên cho ra kết quả sau:

orderYear	curYearQty	prvYearQty
2014	9581	NULL
2015	15489	9581
2016	16247	15489

Để minh họa bảng tạm cục bộ chỉ hiển thị với phiên tạo, chúng ta hãy thử truy cập bảng từ một phiên khác:

```
SELECT orderYear,  
       qty  
  FROM #MyOrderTotalsByYear;
```

sẽ nhận được lỗi sau:

```
Msg 208, Level 16, State 0, Line 1  
      Invalid object name '#MyOrderTotalsByYear'.
```

Khi đã hoàn tất, hãy quay lại phiên ban đầu và hủy bỏ bảng tạm:

```
DROP TABLE IF EXISTS #MyOrderTotalsByYear;
```

Thông thường, chúng ta nên dọn sạch tài nguyên ngay sau khi hoàn thành công việc với chúng.

### 3.5.3.2 Bảng tạm toàn cục

Khi một bảng tạm toàn cục được tạo, nó sẽ hiển thị với tất cả các phiên khác. Các bảng tạm toàn cục bị SQL Server tự động hủy khi phiên tạo bị ngắt kết nối và không có hoạt động nào tham chiếu đến bảng. Bảng tạm toàn cục được tạo ra bằng cách đặt tên cho nó với hai dấu thăng làm tiền tố, chẳng hạn như ##T1.

Bảng tạm toàn cục rất hữu ích khi chúng ta muốn chia sẻ dữ liệu tạm thời với mọi người. Không yêu cầu quyền đặc biệt và mọi người đều có quyền truy cập DDL và DML đầy đủ. Tất nhiên, thực tế là mọi người đều có toàn quyền truy cập có nghĩa là bất kỳ ai cũng có thể thay đổi hoặc thậm chí hủy bỏ bảng, vì vậy hãy cân nhắc các lựa chọn thay thế một cách cẩn thận.

Ví dụ, đoạn mã sau tạo một bảng tạm toàn cục có tên ##Globals với các cột có tên là id và val:

```
CREATE TABLE ##Globals  
(  
    id sysname NOT NULL PRIMARY KEY,  
    val SQL_VARIANT NOT NULL  
);
```

Bảng trong ví dụ này nhằm mô phỏng các biến toàn cục không được hỗ trợ trong T-SQL. Cột id thuộc kiểu dữ liệu sysname (kiểu mà SQL Server sử dụng nội bộ để biểu diễn các mã định danh) và cột val thuộc kiểu dữ liệu SQL\_VARIANT (một kiểu tổng quát có thể lưu trữ bên trong nó một giá trị của hầu hết mọi kiểu cơ sở).

Bất kỳ ai cũng có thể chèn hàng vào bảng. Ví dụ, đoạn mã sau chèn một hàng biểu diễn cho một biến có tên i và khởi tạo với giá trị nguyên 10:

```
INSERT INTO ##Globals
(
    id,
    val
)
VALUES
    (N'i', CAST(10 AS INT));
```

Bất kỳ ai cũng có thể sửa đổi và truy xuất dữ liệu từ bảng. Ví dụ, đoạn mã sau được thực thi từ bất kỳ phiên nào để truy vấn giá trị hiện tại của biến i:

```
SELECT val
FROM ##Globals
WHERE id = N'i';
```

Kết quả của câu lệnh trên là:

```
val
10
```

Nếu muốn tạo một bảng tạm toàn cục mỗi khi SQL Server khởi động và không muốn SQL Server hủy bỏ bảng đó một cách tự động, chúng ta cần tạo bảng từ một thủ tục lưu trữ được đánh dấu là một thủ tục khởi động.

Thực thi đoạn mã sau từ bất kỳ phiên nào để chủ động hủy bảng tạm toàn cục:

```
DROP TABLE IF EXISTS ##Globals;
```

### 3.5.3.3 Biến bảng

Các biến bảng tương tự như các bảng tạm cục bộ nhưng có khác ở một số điểm. Chúng ta khai báo các biến bảng giống như khai báo các biến khác, bằng cách sử dụng câu lệnh DECLARE.

Các biến bảng tồn tại vật lý như một bảng trong cơ sở dữ liệu tempdb, trái ngược với quan niệm sai lầm phổ biến rằng chúng chỉ tồn tại trong bộ nhớ. Giống như các bảng tạm cục bộ, các biến bảng chỉ hiển thị với phiên tạo, nhưng vì chúng là các biến nên chúng có phạm vi hạn chế hơn: chỉ trong lô hiện tại. Các biến bảng không hiển thị đối với các lô bên trong ngăn xếp lòi gọi cũng như các lô tiếp theo trong cùng phiên.

Nếu một giao dịch bị khôi phục, các thay đổi được thực hiện đối với các bảng tạm trong giao dịch đó cũng sẽ được khôi phục; tuy nhiên, những thay đổi được thực hiện đối với các biến bảng bởi các câu lệnh đã hoàn thành trong giao dịch sẽ không được khôi phục. Chỉ những thay đổi được thực hiện bởi câu lệnh không thành công hoặc đã bị chấm dứt trước khi hoàn thành mới được hoàn tác.

Bảng tạm và biến bảng cũng có những khác biệt về tối ưu hóa, nhưng những chủ đề đó nằm ngoài phạm vi của tài liệu này. Hiện tại, chúng ta sẽ chỉ xem xét về mặt hiệu suất, thông thường sẽ hợp lý hơn khi sử dụng các biến bảng với khối lượng dữ liệu nhỏ (chỉ một vài hàng) và sử dụng các bảng tạm cục bộ trong trường hợp ngược lại.

Ví dụ, đoạn mã sau sử dụng biến bảng thay vì bảng tạm cục bộ để so sánh tổng số lượng đặt hàng của từng năm đặt hàng với năm trước đó:

```
DECLARE @MyOrderTotalsByYear TABLE
(
    orderYear INT NOT NULL PRIMARY KEY,
    qty INT NOT NULL
);
INSERT INTO @MyOrderTotalsByYear
(
    orderYear,
    qty
)
SELECT YEAR(O.orderDate) AS orderYear,
       SUM(OD.qty) AS qty
FROM Orders AS O INNER JOIN OrderDetails AS OD
    ON OD.orderNo = O.orderNo
GROUP BY YEAR(orderDate);
SELECT Cur.orderYear,
       Cur.qty AS curYearQty,
       Prv.qty AS prvYearQty
FROM @MyOrderTotalsByYear AS Cur
LEFT OUTER JOIN @MyOrderTotalsByYear AS Prv
    ON Cur.orderYear = Prv.orderYear + 1;
```

Kết quả trả về của đoạn mã trên như sau:

orderYear	curYearQty	prvYearQty
2014	9581	NULL
2015	15489	9581
2016	16247	15489

### 3.5.3.4 Kiểu bảng

Chúng ta có thể sử dụng một kiểu bảng (table type) để lưu giữ định nghĩa bảng dưới dạng một đối tượng trong cơ sở dữ liệu. Sau đó, có thể sử dụng lại nó làm định nghĩa bảng của các biến bảng và tham số đầu vào của các thủ tục lưu trữ và các hàm do người dùng định nghĩa. Các kiểu bảng là bắt buộc đối với các tham số giá trị bảng (table-valued parameter - TVP).

Ví dụ, đoạn mã sau tạo một kiểu bảng có tên là dbo.OrderTotalsByYear:

```
DROP TYPE IF EXISTS dbo.OrderTotalsByYear;
CREATE TYPE dbo.OrderTotalsByYear AS TABLE
(
    orderYear INT NOT NULL PRIMARY KEY,
    qty INT NOT NULL
);
```

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Sau khi kiểu bảng được tạo, bất cứ khi nào cần khai báo một biến bảng dựa trên định nghĩa của kiểu bảng, chúng ta sẽ không cần lặp lại mã - thay vào đó, chỉ cần chỉ định dbo.OrderTotalsByYear làm kiểu của biến, như sau:

```
DECLARE @MyOrderTotalsByYear AS dbo.OrderTotalsByYear;
```

Đoạn mã sau khai báo một biến có tên @MyOrderTotalsByYear của kiểu bảng mới, truy vấn các bảng Orders và OrderDetails để tính tổng số lượng đơn hàng theo năm đặt hàng, lưu trữ kết quả của truy vấn trong biến bảng và truy vấn biến để hiển thị nội dung của nó:

```
DECLARE @MyOrderTotalsByYear AS dbo.OrderTotalsByYear;
INSERT INTO @MyOrderTotalsByYear
(
    orderYear,
    qty
)
SELECT YEAR(O.orderDate) AS orderYear,
       SUM(OD.qty) AS qty
FROM Orders AS O
    INNER JOIN OrderDetails AS OD
        ON OD.orderNo = O.orderNo
GROUP BY YEAR(orderDate);
SELECT orderYear,
       qty
FROM @MyOrderTotalsByYear;
```

Kết quả của đoạn mã trên như sau:

orderYear	qty
2014	9581
2015	15489
2016	16247

Lợi ích của kiểu bảng không chỉ giúp rút ngắn mã lệnh, mà còn có thể sử dụng làm kiểu tham số đầu vào cho các thủ tục lưu trữ và hàm.

### 3.5.4 SQL động – Lệnh EXEC – Thủ tục sp\_executesql

#### 3.5.4.1 SQL động

Với SQL Server, chúng ta có thể xây dựng một lô mã T-SQL dưới dạng một chuỗi ký tự và sau đó thực thi lô này. Khả năng này được gọi là SQL động (dynamic SQL).

SQL Server cung cấp hai cách để thực thi SQL động: sử dụng lệnh EXEC (viết tắt của EXECUTE) và sử dụng thủ tục lưu trữ sp\_executesql. Chúng ta sẽ xem xét sự khác biệt giữa hai kiểu này thông qua các ví dụ cụ thể.

SQL động rất hữu ích cho một số mục đích như sau:

- Tự động hóa các tác vụ quản trị. Ví dụ, truy vấn siêu dữ liệu (metadata), xây dựng và thực thi câu lệnh BACKUP DATABASE cho từng cơ sở dữ liệu trong SQL Server
- Cải thiện hiệu suất của một số tác vụ. Ví dụ, xây dựng các truy vấn đặc biệt được tham số hóa có thể sử dụng lại các kế hoạch thực thi đã lưu trong bộ nhớ cache trước đó
- Xây dựng các phần tử của mã dựa trên truy vấn dữ liệu thực tế. Ví dụ, xây dựng truy vấn PIVOT động khi chúng ta không biết trước phần tử nào sẽ xuất hiện trong mệnh đề IN của toán tử PIVOT

Hãy cực kỳ cẩn thận khi ghép nối đầu vào của người dùng như một phần mã của chúng ta. Tin tức có thể cố gắng đưa vào mã mà chúng ta không có ý định thực thi. Biện pháp tốt nhất có thể thực hiện để chống lại việc này là tránh ghép nối đầu vào của người dùng như một phần mã của chúng ta (thay vào đó, chúng ta có thể sử dụng tham số).

#### 3.5.4.2 Lệnh EXEC

Lệnh EXEC chấp nhận một chuỗi ký tự trong ngoặc đơn làm đầu vào và thực thi lô mã trong chuỗi ký tự đó. EXEC hỗ trợ cả chuỗi ký tự thông thường và Unicode làm đầu vào. Lệnh này cũng có thể được sử dụng để thực thi một thủ tục lưu trữ, như sẽ được trình bày ở phần sau của nội dung này.

Ví dụ sau lưu trữ một chuỗi ký tự với câu lệnh PRINT trong biến @sql và sau đó sử dụng lệnh EXEC để thực thi lô mã được lưu trữ trong biến:

```
DECLARE @sql AS VARCHAR(100);
SET @sql = 'PRINT ''This message was printed by a dynamic
SQL batch.'';'
EXEC (@sql);
```

Lưu ý việc sử dụng hai dấu nháy đơn để biểu thị một dấu nháy đơn trong một chuỗi ký tự trong một chuỗi ký tự khác.

Mã trên trả về kết quả sau:

```
This message was printed by a dynamic SQL batch.
```

### 3.5.4.3 Thủ tục sp\_executesql

Thủ tục lưu trữ sp\_executesql là một công cụ thay thế cho lệnh EXEC để thực thi mã SQL động, an toàn và linh hoạt hơn vì lý do có hỗ trợ các tham số đầu vào và đầu ra. Lưu ý rằng không giống như EXEC, sp\_executesql chỉ hỗ trợ các chuỗi ký tự Unicode làm lô mã đầu vào.

Về mặt bảo mật, các tham số xuất hiện trong mã không thể được coi là một phần của mã - chỉ có thể được coi là toán hạng trong biểu thức. Vì vậy, bằng cách sử dụng các tham số, chúng ta có thể loại bỏ mối đe dọa của việc bị chèn mã không mong muốn vào mã của mình.

Thủ tục lưu trữ sp\_executesql có thể hoạt động tốt hơn EXEC vì hỗ trợ tham số hóa trong việc sử dụng lại các kế hoạch thực thi được lưu trong bộ nhớ cache. Kế hoạch thực thi (execution plan) là kế hoạch xử lý vật lý mà SQL Server tạo ra cho một truy vấn, với tập hợp các hướng dẫn mô tả đối tượng nào sẽ truy cập, theo thứ tự nào, chỉ mục nào sẽ sử dụng, cách truy cập chúng, sử dụng thuật toán nào để kết hợp,... Một trong những yêu cầu để sử dụng lại kế hoạch đã lưu trong bộ nhớ cache trước đó là chuỗi truy vấn phải giống với chuỗi mà kế hoạch đã lưu trong bộ nhớ cache đã được tạo. Cách tốt nhất để tái sử dụng hiệu quả các kế hoạch thực thi truy vấn là sử dụng các thủ tục lưu trữ có tham số. Bằng cách này, ngay cả khi giá trị tham số thay đổi, chuỗi truy vấn vẫn giữ nguyên. Nhưng nếu chúng ta đã quyết định sử dụng mã đặc biệt thay vì các thủ tục lưu trữ, thì ít nhất vẫn có thể làm việc với các tham số nếu sử dụng sp\_executesql và do đó tăng cơ hội sử dụng lại kế hoạch.

Thủ tục sp\_executesql có hai tham số đầu vào và phần gán. Chúng ta chỉ định chuỗi ký tự Unicode chứa lô mã muốn thực thi trong tham số đầu tiên, được gọi là @stmt; cung cấp một chuỗi ký tự Unicode chứa các khai báo của tham số đầu vào và đầu ra trong tham số đầu vào thứ hai, được gọi là @params. Sau đó, chúng ta đặc tả việc gán các tham số đầu vào và đầu ra được phân tách bằng dấu phẩy.

Ví dụ sau đây tạo một lô mã với một truy vấn trên bảng Sales.Orders; sử dụng tham số đầu vào có tên @orderid trong bộ lọc của truy vấn:

```
DECLARE @sql AS NVARCHAR(100);
SET @sql = N'SELECT orderNo, custID, empID, orderDate FROM
    Orders WHERE orderNo = @orderNo;';
EXEC sp_executesql @stmt = @sql,
                    @params = N'@orderNo AS INT',
                    @orderNo = 10248;
```

Mã trên tạo ra kết quả như sau:

orderNo	custID	empID	orderDate
10248	85	5	2014-07-04

Đoạn mã trên gán giá trị 10248 cho tham số đầu vào, nhưng ngay cả khi thực thi lại nó với một giá trị khác, thì chuỗi mã vẫn giữ nguyên. Bằng cách này, chúng ta tăng cơ hội tái sử dụng kế hoạch thực thi đã lưu trong bộ nhớ cache.

### 3.5.5 Các cấu trúc điều khiển – Xử lý ngoại lệ

T-SQL cung cấp các cấu trúc điều khiển luồng thực thi cơ bản như IF ... ELSE, và WHILE.

#### 3.5.5.1 IF... ELSE

IF... ELSE được sử dụng để điều khiển luồng thực thi dựa trên kết quả của một vị từ. Một câu lệnh hoặc khối câu lệnh sẽ được thực thi nếu vị từ là TRUE và tùy chọn một câu lệnh hoặc khối câu lệnh sẽ được thực thi nếu vị ngữ là FALSE hoặc UNKNOWN. Ví dụ, đoạn mã sau kiểm tra xem hôm nay có phải là ngày cuối cùng của năm hay không (nói cách khác, liệu năm của hôm nay có khác năm của ngày mai hay không). Nếu đúng, mã sẽ in một thông báo hôm nay là ngày cuối cùng của năm; nếu không đúng (“ELSE”), mã sẽ in ra thông báo hôm nay không phải là ngày cuối cùng của năm:

```
IF YEAR(SYSDATETIME()) <> YEAR(DATEADD(DAY, 1,
                                             SYSDATETIME()))
    PRINT 'Today is the last day of the year.';
ELSE
    PRINT 'Today is not the last day of the year.';
```

Trong ví dụ này, các câu lệnh PRINT được sử dụng để minh họa phần nào của mã được thực thi và phần nào không, và tất nhiên các câu lệnh khác cũng có thể được sử dụng thay thế.

Hãy nhớ rằng T-SQL sử dụng logic ba giá trị và khối ELSE được kích hoạt khi vị từ là FALSE hoặc UNKNOWN. Trong các trường hợp mà cả FALSE và UNKNOWN đều là kết quả có thể xảy ra của vị từ (ví dụ khi liên quan đến NULL) và cần cách xử lý khác nhau cho từng trường hợp, hãy đảm bảo rằng phải có một cách kiểm tra rõ ràng cho NULL với vị từ IS NULL.

Nếu luồng cần điều khiển liên quan đến nhiều hơn hai trường hợp, chúng ta có thể lồng các IF... ELSE. Ví dụ, đoạn mã tiếp theo sẽ chỉ cho thấy việc xử lý ba trường hợp khác nhau:

- Hôm nay là ngày cuối cùng của năm.
- Hôm nay là ngày cuối cùng của tháng nhưng không phải là ngày cuối cùng của năm.
- Hôm nay không phải là ngày cuối cùng của tháng.

```
IF YEAR(SYSDATETIME()) <> YEAR(DATEADD(DAY, 1,
                                             SYSDATETIME()))
    PRINT 'Today is the last day of the year.';
ELSE IF MONTH(SYSDATETIME()) <> MONTH(DATEADD(DAY, 1,
                                                 SYSDATETIME()))
    PRINT 'Today is the last day of the month but not the
          last day of the year.';
ELSE
    PRINT 'Today is not the last day of the month.';
```

Nếu cần thực thi nhiều hơn một câu lệnh trong phần IF hoặc ELSE, chúng ta cần sử dụng một khối lệnh. Khối lệnh được bao bằng cặp từ khóa BEGIN và END. Ví dụ, đoạn mã minh họa cách thực thi một quy trình nếu đó là ngày đầu tiên của tháng và một quy trình khác nếu không phải:

```
IF DAY(SYSDATETIME()) = 1
BEGIN
    PRINT 'Today is the first day of the month.';
    PRINT 'Starting first-of-month-day process.';
    /* ... process code goes here ... */
    PRINT 'Finished first-of-month-day database process.';

END;

ELSE
BEGIN
    PRINT 'Today is not the first day of the month.';
    PRINT 'Starting non-first-of-month-day process.';
    /* ... process code goes here ... */
    PRINT 'Finished non-first-of-month-day process.';

END;
```

### 3.5.5.2 WHILE

T-SQL cung cấp cấu trúc WHILE để thực thi mã trong một vòng lặp. WHILE sẽ thực thi lặp đi lặp lại một câu lệnh hoặc khối câu lệnh trong khi vị trí được chỉ định sau từ khóa WHILE là TRUE. Khi vị trí là FALSE hoặc UNKNOWN, việc lặp này kết thúc.

T-SQL không cung cấp cấu trúc lặp tích hợp số lần thực thi được xác định trước, nhưng thật dễ dàng mô phỏng cấu trúc này bằng vòng lặp WHILE với một biến. Ví dụ:

```
DECLARE @i AS INT = 1;
WHILE @i <= 5
BEGIN
    PRINT @i;
    SET @i = @i + 1;
END;
```

Đoạn mã trên khai báo một biến số nguyên `@i` đóng vai trò là bộ đếm vòng lặp và được khởi tạo với giá trị 1. Sau đó, đoạn mã này đi vào một vòng lặp thực hiện công việc trong khi bộ đếm nhỏ hơn hoặc bằng 5. Trong mỗi lần lặp, đoạn mã trong phần thân của vòng lặp in giá trị hiện tại của `@i` và sau đó tăng giá trị của `@i` lên 1. Kết quả trả về sau cho thấy rằng vòng lặp đã lặp lại 5 lần:

1  
2  
3  
4  
5

Nếu tại một thời điểm nào đó trong phần thân của vòng lặp, chúng ta muốn thoát ra khỏi vòng lặp và tiếp tục thực hiện câu lệnh sau vòng lặp, hãy sử dụng lệnh BREAK. Ví dụ, đoạn mã sau bị ngắt khỏi vòng lặp nếu giá trị của @i bằng 3:

```
DECLARE @i AS INT = 1;
WHILE @i <= 5
BEGIN
    IF @i = 3
        BREAK;
    PRINT @i;
    SET @i = @i + 1;
END;
```

Đoạn mã trên tạo ra kết quả sau đây cho thấy rằng vòng lặp đã lặp lại hai lần và kết thúc ở đầu lần lặp thứ ba:

```
1
2
```

Tất nhiên, mã này không thực tế lắm; nếu muốn vòng lặp chỉ lặp hai lần, chúng ta chỉ cần chỉ định vị từ `@i <= 2`. Ở đây mục đích chỉ muốn minh họa việc sử dụng lệnh BREAK thông qua một ví dụ đơn giản.

Nếu tại một thời điểm nào đó trong phần nội dung của vòng lặp, chúng ta muốn bỏ qua phần còn lại của hoạt động trong lần lặp hiện tại và đánh giá lại vị từ của vòng lặp, hãy sử dụng lệnh CONTINUE. Ví dụ, đoạn mã sau trình bày cách bỏ qua hoạt động của lần lặp thứ ba của vòng lặp từ điểm mà câu lệnh IF xuất hiện và cho đến khi kết thúc phần thân của vòng lặp:

```
DECLARE @i AS INT = 0;
WHILE @i < 5
BEGIN
    SET @i = @i + 1;
    IF @i = 3
        CONTINUE;
    PRINT @i;
END;
```

Kết quả của đoạn mã trên cho thấy rằng giá trị của `@i` đã được in trong tất cả các lần lặp trừ lần thứ sáu:

```
1
2
4
5
```

Một ví dụ khác về việc sử dụng vòng lặp WHILE, đoạn mã sau tạo một bảng có tên `dbo.Numbers` và điền vào đó 1.000 hàng với các giá trị từ 1 đến 1.000 trong cột `n`:

```
SET NOCOUNT ON;
DROP TABLE IF EXISTS dbo.Numbers;
CREATE TABLE dbo.Numbers
```

```

(
    n INT NOT NULL PRIMARY KEY
);
GO
DECLARE @i AS INT = 1;
WHILE @i <= 1000
BEGIN
    INSERT INTO dbo.Numbers
    (
        n
    )
    VALUES (@i );
    SET @i = @i + 1;
END;

```

### 3.5.5.3 Ngoại lệ trong T-SQL

SQL Server cung cấp công cụ để xử lý ngoại lệ trong mã T-SQL là một cấu trúc được gọi là TRY...CATCH và một tập các hàm trả về các thông tin về lỗi.

Chúng ta làm việc với cấu trúc TRY... .CATCH bằng cách đặt mã T-SQL thông thường vào khối TRY (giữa từ khóa BEGIN TRY và END TRY) và đặt tất cả mã xử lý lỗi vào khối CATCH liền kề (giữa từ khóa BEGIN CATCH và END CATCH). Nếu khối TRY không có lỗi, khối CATCH sẽ bị bỏ qua. Nếu khối TRY có lỗi, quyền điều khiển sẽ được chuyển sang khối CATCH tương ứng.

Thực thi đoạn mã sau để minh họa trường hợp không có lỗi trong khối TRY:

```

BEGIN TRY
    PRINT 10 / 2;
    PRINT 'No error';
END TRY
BEGIN CATCH
    PRINT 'Error';
END CATCH;

```

Tất cả mã trong khối TRY đã hoàn tất thành công; do đó, khối CATCH đã bị bỏ qua. Mã trên tạo ra kết quả sau:

```

5
No error

```

Tiếp theo, thực thi mã tương tự, nhưng lần này chia cho 0. Một lỗi xảy ra:

```

BEGIN TRY
    PRINT 10 / 0;
    PRINT 'No error';
END TRY
BEGIN CATCH
    PRINT 'Error';
END CATCH;

```

Khi lỗi chia cho 0 (*divide by zero*) xảy ra trong câu lệnh PRINT đầu tiên trong khối TRY, quyền điều khiển được chuyển đến khối CATCH tương ứng. Câu lệnh PRINT thứ hai trong khối TRY không được thực hiện. Do đó, mã này tạo ra kết quả sau:

#### Error

Thông thường, xử lý lỗi liên quan đến việc khối CATCH điều tra nguyên nhân gây ra lỗi và thực hiện một quá trình hành động. SQL Server cung cấp thông tin về lỗi thông qua một tập hợp các hàm. Hàm ERROR\_NUMBER trả về một số nguyên là mã lỗi. Khối CATCH thường bao gồm mã luồng kiểm tra mã lỗi để xác định quá trình hành động cần thực hiện. Hàm ERROR\_MESSAGE trả về văn bản thông báo lỗi. Để có danh sách các mã lỗi và thông báo, chúng ta có thể truy vấn khung nhìn danh mục sys.messages. Các hàm ERROR\_SEVERITY và ERROR\_STATE trả về trạng thái và mức độ nghiêm trọng của lỗi. Hàm ERROR\_LINE trả về số của dòng trong mã xảy ra lỗi. Cuối cùng, hàm ERROR\_PROCEDURE trả về tên của thủ tục xảy ra lỗi và trả về NULL nếu lỗi không xảy ra trong một thủ tục.

Để minh họa một trường hợp xử lý lỗi chi tiết hơn bao gồm cả việc sử dụng các hàm báo lỗi, trước tiên hãy thực thi đoạn mã sau tạo một bảng có tên là dbo.Employees:

```
DROP TABLE IF EXISTS dbo.Employees;
CREATE TABLE dbo.Employees
(
    empID INT NOT NULL,
    empName VARCHAR(25) NOT NULL,
    branchID INT NULL,
    CONSTRAINT PK_Employees PRIMARY KEY (empID),
    CONSTRAINT CHK_Employees_empID CHECK (empID > 0),
    CONSTRAINT FK_Employees_Employees FOREIGN KEY (branchID)
        REFERENCES dbo.Employees (empID)
);
```

Đoạn mã sau sẽ chèn một hàng mới vào bảng Employees trong khối TRY và nếu xảy ra lỗi, cho thấy cách xác định lỗi bằng cách kiểm tra hàm ERROR\_NUMBER trong khối CATCH. Mã này sử dụng điều khiển luồng để xác định và xử lý các lỗi mà chúng ta muốn xử lý trong khối CATCH và nếu không xác định được, sẽ ném (throw) ra lỗi không xác định được đó. Đồng thời, đoạn mã này cũng in các giá trị của các hàm báo lỗi khác chỉ để hiển thị thông tin có sẵn khi xảy ra lỗi:

```
BEGIN TRY
    INSERT INTO dbo.Employees (empID, empName, branchID)
    VALUES (1, 'Emp1', NULL);
    -- Also try with empID = 0, 'A', NULL
END TRY
```

```

BEGIN CATCH
    IF ERROR_NUMBER() = 2627
        BEGIN
            PRINT ' Handling PK violation...';
        END;
    ELSE IF ERROR_NUMBER() = 547
        BEGIN
            PRINT ' Handling CHECK/FK constraint violation...';
        END;
    ELSE IF ERROR_NUMBER() = 515
        BEGIN
            PRINT ' Handling NULL violation...';
        END;
    ELSE IF ERROR_NUMBER() = 245
        BEGIN
            PRINT ' Handling conversion error...';
        END;
    ELSE
        BEGIN
            PRINT 'Re-throwing error...';
            THROW;
        END;
    PRINT ' Error Number : ' + CAST(ERROR_NUMBER() AS
        VARCHAR(10));
    PRINT ' Error Message : ' + ERROR_MESSAGE();
    PRINT ' Error Severity : ' + CAST(ERROR_SEVERITY() AS
        VARCHAR(10));
    PRINT ' Error State : ' + CAST(ERROR_STATE() AS
        VARCHAR(10));
    PRINT ' Error Line : ' + CAST(ERROR_LINE() AS
        VARCHAR(10));
    PRINT ' Error Proc : ' + COALESCE(ERROR_PROCEDURE(),
        'Not within proc');

END CATCH;

```

Khi mã này được thực thi lần đầu tiên, hàng mới được chèn thành công vào bảng *Employees* và do đó khối CATCH bị bỏ qua.

Kết quả như sau:

Command(s) completed successfully.

Khi chúng ta thực thi mã trên lần thứ hai, câu lệnh INSERT không thành công, quyền điều khiển được chuyển đến khối CATCH và lỗi vi phạm khóa chính được xác định.

Thông báo lỗi hiển thị như sau:

```
Handling PK violation...
Error      Number : 2627
ErrorMessage : Violation of PRIMARY KEY constraint
'PK_Employees'. Cannot insert duplicate key in object
'dbo.Employees'. The duplicate key value is (1).
Error      Severity : 14
Error      State : 1
Error      Line : 2
Error      Proc : Not within proc
```

Để xem các lỗi khác, hãy thực thi mã với các giá trị 0, 'A' và NULL làm ID nhân viên.

Ở đây, với mục đích minh họa, chúng ta đã sử dụng các câu lệnh PRINT làm hành động khi một lỗi được xác định. Tuy nhiên, xử lý lỗi thường liên quan đến nhiều thứ hơn là chỉ in một thông báo cho biết rằng lỗi đã được xác định.

Lưu ý rằng chúng ta có thể tạo một thủ tục lưu trữ đóng gói mã xử lý lỗi có thể tái sử dụng như sau:

```
DROP PROC IF EXISTS dbo.ErrInsertHandler;
GO
CREATE PROC dbo.ErrInsertHandler
AS
SET NOCOUNT ON;
IF ERROR_NUMBER() = 2627
BEGIN
    PRINT 'Handling PK violation...';
END;
ELSE IF ERROR_NUMBER() = 547
BEGIN
    PRINT 'Handling CHECK/FK constraint violation...';
END;
ELSE IF ERROR_NUMBER() = 515
BEGIN
    PRINT 'Handling NULL violation...';
END;
ELSE IF ERROR_NUMBER() = 245
BEGIN
    PRINT 'Handling conversion error...';
END;
PRINT ' Error Number : ' + CAST(ERROR_NUMBER() AS
VARCHAR(10));
PRINT ' Error Message : ' + ERROR_MESSAGE();
PRINT ' Error Severity: ' + CAST(ERROR_SEVERITY() AS
VARCHAR(10));
```

```

PRINT ' Error State : ' + CAST(ERROR_STATE() AS
      VARCHAR(10));
PRINT ' Error Line : ' + CAST(ERROR_LINE() AS
      VARCHAR(10));
PRINT ' Error Proc: ' + COALESCE(ERROR_PROCEDURE(),
      'Not within proc');
GO

```

Trong khối CATCH, chúng ta kiểm tra xem mã lỗi có phải là một trong những mã muôn xử lý cục bộ hay không. Nếu đúng như vậy, chỉ cần thực hiện thủ tục lưu trữ; nếu không, ném lại lỗi đó:

```

BEGIN TRY
    INSERT INTO dbo.Employees
    (
        empID,
        empName,
        branchID
    )
    VALUES
    (1, 'Emp1', NULL);
END TRY
BEGIN CATCH
    IF ERROR_NUMBER() IN ( 2627, 547, 515, 245 )
        EXEC dbo.ErrInsertHandler;
    ELSE
        THROW;
END CATCH;

```

Bằng cách này, chúng ta có thể duy trì mã xử lý lỗi để có thể tái sử dụng ở một nơi khác.

### 3.6 CHƯƠNG TRÌNH CON

Các chương trình con (routine) là các đối tượng có thể lập trình, đóng gói mã để tính toán kết quả hoặc để thực thi hoạt động. SQL Server hỗ trợ ba loại chương trình con: hàm do người dùng định nghĩa (user-defined function - UDF), thủ tục lưu trữ (stored procedure - SP) và trình kích hoạt (trigger).

Với SQL Server, chúng ta có thể chọn phát triển một chương trình con với T-SQL hoặc với mã Microsoft .NET dựa trên tích hợp CLR trong sản phẩm. Vì trọng tâm của tài liệu này là T-SQL nên các ví dụ ở đây sử dụng T-SQL. Khi tác vụ chủ yếu liên quan đến thao tác dữ liệu, T-SQL thường là lựa chọn tốt hơn. Khi tác vụ thiên về logic lặp, thao tác chuỗi hoặc hoạt động tính toán chuyên sâu, mã .NET thường sẽ là lựa chọn tốt hơn.

### 3.6.1 Hàm người dùng định nghĩa

Mục đích của hàm người dùng định nghĩa (user-defined function - UDF) là đóng gói logic tính toán điều gì đó, có thể dựa trên các tham số đầu vào và trả về một kết quả.

SQL Server hỗ trợ các UDF vô hướng (scalar) và giá trị bảng (table-valued). UDF vô hướng trả về một giá trị duy nhất; UDF giá trị bảng trả về một bảng. Một lợi ích của việc sử dụng UDF là chúng ta có thể kết hợp chúng vào các truy vấn. Các UDF vô hướng có thể xuất hiện ở bất kỳ đâu trong truy vấn nơi mà một biểu thức trả về một giá trị có thể xuất hiện (ví dụ, trong danh sách *SELECT*). UDF giá trị bảng có thể xuất hiện trong mệnh đề *FROM* của truy vấn.

UDF không được phép có bất kỳ tác dụng phụ nào, nghĩa là các UDF không được phép áp dụng bất kỳ thay đổi lược đồ hoặc dữ liệu nào trong cơ sở dữ liệu. Ví dụ, việc gọi hàm RAND để trả về một giá trị ngẫu nhiên hoặc hàm NEWID để trả về một mã định danh toàn cục duy nhất (globally unique identifier - GUID) đều có tác dụng phụ. Bất cứ khi nào hàm RAND được gọi mà không chỉ định một hạt giống, SQL Server sẽ tạo một hạt giống ngẫu nhiên dựa trên lần được gọi trước đó của RAND. Vì lý do này, SQL Server cần lưu trữ thông tin bên trong bất cứ khi nào hàm RAND được gọi. Tương tự, bất cứ khi nào hàm NEWID được gọi, hệ thống cần thiết lập một số thông tin để xem xét trong lần gọi NEWID tiếp theo.

Một hàm vô hướng chỉ có thể trả về một giá trị duy nhất. Hàm GETDATE() mà chúng ta đã quen thuộc, là một hàm vô hướng. Sau đây là cú pháp cơ bản để định nghĩa một hàm vô hướng:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name data_type [ = default ] [ READONLY ] }[ ,...n ] ]
RETURNS return_data_type
[ WITH <function_option> [ ,...n ] ]
[ AS ]
BEGIN
    function_body
RETURN scalar_expression
END
```

Lưu ý rằng không, một hoặc nhiều tham số có thể được truyền vào hàm. Đặt tiền tố cho mỗi tham số trong định nghĩa bằng dấu @ và xác định kiểu dữ liệu. Mọi tham số có thể được thay đổi trong hàm như một phần của quá trình thực thi hàm, trừ khi đặt từ khóa READONLY sau kiểu dữ liệu khi định nghĩa hàm. Có thể gọi một hàm và bỏ qua việc đặc tả một hoặc nhiều tham số của hàm đó. Bất kỳ tham số nào bị bỏ qua phải được định nghĩa với các giá trị mặc định. Trong trường hợp đó, có thể gọi hàm với từ khóa DEFAULT ở vị trí tham số đó.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Mệnh đề RETURNS trong định nghĩa hàm xác định kiểu dữ liệu sẽ được trả về. Tất cả các kiểu dữ liệu, ngoại trừ loại dữ liệu dấu thời gian, đều có thể được trả về.

Chúng ta phải đặt một câu lệnh RETURN khi muốn kết thúc hàm và trả lại điều khiển cho mã gọi.

Hàm giá trị bảng cho phép trả về một bảng dữ liệu thay vì một giá trị đơn lẻ. Chúng ta có thể sử dụng hàm giá trị bảng để thay thế danh sách bảng trong mệnh đề FROM của câu lệnh SELECT. Cú pháp cơ bản cho hàm giá trị bảng như sau:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name parameter_datatype [ =default ] [ READONLY ] } [,...n]
      ]
RETURNS TABLE
[ WITH <function_option> [ ,...n ] ]
[ AS ]
RETURN [ ( ] select_stmt [ ) ]
```

Lưu ý là các hàm luôn đảm bảo tính an toàn. Nếu một lỗi được tạo ra trong một hàm, cho dù là do dữ liệu không hợp lệ được truyền vào hay do lỗi trong logic, hàm sẽ ngừng thực thi tại thời điểm đó và lời gọi hàm sẽ bị hủy. Và hàm cũng không được thay đổi bất kỳ tài nguyên bên ngoài nào chẳng hạn như bảng và không được thực thi các hàm hệ thống làm thay đổi tài nguyên, chẳng hạn như chức năng gửi e-mail.

Ví dụ, đoạn mã sau tạo một UDF có tên là dbo.GetAge trả về tuổi của một người có ngày sinh cụ thể (đối số (@birthDate) tại một ngày cụ thể (đối số @eventDate):

```
DROP FUNCTION IF EXISTS dbo.GetAge;
GO
CREATE FUNCTION dbo.GetAge
(
    @birthDate AS DATE,
    @eventDate AS DATE
)
RETURNS INT
AS
BEGIN
    RETURN DATEDIFF(YEAR, @birthDate, @eventDate)
    - CASE
        WHEN 100 * MONTH(@eventDate) + DAY(@eventDate) <
100 * MONTH(@birthDate) + DAY(@birthDate) THEN
            1
        ELSE
            0
    END;
END;
GO
```

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Hàm tính tuổi là hiệu số tính theo năm giữa năm sinh và năm của ngày cho trước, trừ 1 năm trong trường hợp tháng và ngày cho trước nhỏ hơn tháng và ngày sinh. Biểu thức  $100 * \text{month} + \text{day}$  chỉ đơn giản là một thủ thuật để nối tháng và ngày. Ví dụ, đối với ngày 12 trong tháng 2, biểu thức cho ra số nguyên 212.

Để minh họa việc sử dụng UDF trong một truy vấn, đoạn mã sau sẽ truy vấn bảng Employees và gọi hàm GetAge trong danh sách SELECT để tính tuổi của mỗi nhân viên với thời điểm hiện tại:

```
SELECT empID,
       firstName,
       lastName,
       birthDate,
       dbo.GetAge(birthDate, SYSDATETIME()) AS age
  FROM Employees;
```

Ví dụ, nếu thực thi truy vấn này vào ngày 12 tháng 2 năm 2016, chúng ta sẽ nhận được kết quả sau:

empID	firstName	lastName	birthDate	age
1	Sara	Davis	1968-12-08	47
2	Don	Funk	1972-02-19	43
3	Judy	Lew	1983-08-30	32
4	Yael	Peled	1957-09-19	58
5	Sven	Mortensen	1975-03-04	40
6	Paul	Suurs	1983-07-02	32
7	Russell	King	1980-05-29	35
8	Maria	Cameron	1978-01-09	38
9	Patricia	Doyle	1986-01-27	30

### 3.6.2 Thủ tục lưu trữ

Các thủ tục lưu trữ là các chương trình con đóng gói mã. Chúng có thể có các tham số đầu vào và đầu ra, có thể trả về các tập kết quả của truy vấn và được phép có tác dụng phụ. Chúng ta không chỉ có thể sửa đổi dữ liệu thông qua các thủ tục lưu trữ mà còn có thể thực hiện các thay đổi lược đồ thông qua chúng.

Nói một cách đơn giản nhất, một thủ tục lưu trữ là một tập hợp các câu lệnh và lệnh T-SQL đã biên dịch mà SQL Server có thể truy cập trực tiếp. Mã được đặt trong một thủ tục lưu trữ được thực thi dưới dạng một đơn vị hoặc lô của công việc. Lợi ích của điều này là lưu lượng truy cập mạng được giảm đáng kể, vì nhiều câu lệnh T-SQL không bị buộc phải truyền qua mạng một cách riêng lẻ. Chỉ truyền đi tên và các tham số của thủ tục lưu trữ cần được thực thi. Ngoài việc thực thi các câu lệnh SELECT, UPDATE, INSERT, và DELETE, các thủ tục lưu trữ có thể gọi các thủ tục lưu trữ khác, sử dụng các câu lệnh kiểm soát luồng thực thi, tạo các bảng tạm thời bằng các lệnh khác nhau và thực hiện các hàm tổng hợp hoặc các tính toán khác.

Bất kỳ nhà phát triển nào có quyền truy cập để tạo các đối tượng trong SQL Server đều có thể xây dựng một thủ tục lưu trữ. Ngoài ra còn có hàng trăm thủ tục lưu trữ hệ thống, tất cả đều bắt đầu bằng tiền tố sp\_, trong SQL Server.

Sẽ không hữu ích lầm trong việc xây dựng một thủ tục lưu trữ chỉ để thực thi một tập hợp các câu lệnh T-SQL chỉ một lần; ngược lại, một thủ tục lưu trữ là lý tưởng khi chúng ta muốn thực thi một tập hợp các câu lệnh T-SQL nhiều lần. Lý do để chọn một thủ tục lưu trữ tương tự như những lý do khi chọn một khung nhìn hơn là cho phép người dùng truy cập trực tiếp vào bảng dữ liệu. Các thủ tục lưu trữ cũng mang lại lợi ích; ví dụ: SQL Server sẽ luôn lưu vào bộ đệm chiến lược thực thi của thủ tục lưu trữ và có khả năng tái sử dụng, trong khi các chiến lược thực thi các lệnh SQL được tạo khi thực thi có thể hoặc không được lưu trữ trong bộ đệm. Cách tiếp cận thứ hai có thể dẫn đến làm đầy bộ đệm với nhiều chiến lược thực thi rất giống nhau cho các lô tương tự, chẳng hạn như SQL Server sẽ không khớp với các chiến lược thực thi sử dụng cùng một mã cơ bản nhưng có các giá trị tham số khác nhau.

Các thủ tục lưu trữ cung cấp cho ứng dụng của bạn một giao diện đồng nhất đã được chứng minh để truy cập hoặc thao tác dữ liệu. Điều này có nghĩa là chúng ta đảm bảo được tính toàn vẹn của dữ liệu, thực hiện các sửa đổi hoặc lựa chọn chính xác đối với dữ liệu và đảm bảo rằng người dùng cơ sở dữ liệu không cần biết cấu trúc, bộ cục, mối quan hệ hoặc quy trình liên quan cần thiết để thực hiện một chức năng cụ thể. Chúng ta cũng có thể xác thực mọi dữ liệu đầu vào và đảm bảo rằng dữ liệu được cung cấp cho thủ tục lưu trữ là chính xác.

Giống như các khung nhìn, chúng ta có thể cấp quyền thực thi rất cụ thể cho người dùng các thủ tục lưu trữ (quyền duy nhất có sẵn trên một thủ tục lưu trữ là EXECUTE).

Để ngăn truy cập vào mã nguồn, chúng ta có thể mã hóa các thủ tục lưu trữ, mặc dù điều này thực sự chỉ nên được sử dụng trong những trường hợp cần thiết nhất. Bản thân mã không thực sự được mã hóa; nó chỉ bị xáo trộn, có nghĩa là có thể giải mã nếu cần.

So với việc sử dụng mã lệnh SQL, việc sử dụng thủ tục lưu trữ mang lại nhiều lợi ích:

- Thủ tục lưu trữ đóng gói logic: Nếu cần thay đổi việc triển khai một thủ tục, chúng ta chỉ cần áp dụng thay đổi ở một chỗ bằng cách sử dụng lệnh ALTER PROC và tất cả người dùng của thủ tục đều sẽ sử dụng phiên bản mới đó.
- Các thủ tục giúp kiểm soát an ninh tốt hơn: Chúng ta có thể cấp quyền cho người dùng thực hiện thủ tục mà không cần cấp cho người dùng quyền trực tiếp thực hiện các hoạt động bên dưới. Ví dụ, giả sử muốn cho phép một số người dùng nhất định xóa một khách hàng khỏi cơ sở dữ liệu, nhưng lại không muốn cấp cho họ quyền trực tiếp để xóa các hàng khỏi bảng Customers. Ngoài ra, các thủ tục lưu trữ có tham số có thể giúp ngăn chặn việc chèn mã vào lệnh SQL, đặc biệt là khi chúng thay thế mã SQL được gửi từ ứng dụng khách.
- Chúng ta có thể kết hợp tất cả các mã xử lý lỗi trong một thủ tục, âm thầm thực hiện hành động khắc phục khi có sự cố liên quan.

- Các thủ tục lưu trữ mang lại các lợi ích về hiệu năng: Phần trên chúng ta đã nói về việc sử dụng lại các kế hoạch thực thi đã lưu trong bộ nhớ cache. Các truy vấn trong thủ tục lưu trữ thường được tham số hóa và do đó có nhiều khả năng sử dụng lại các kế hoạch đã lưu trong bộ nhớ cache. Một lợi ích hiệu suất khác của việc sử dụng các thủ tục lưu trữ là giảm lưu lượng mạng. Ứng dụng máy khách chỉ cần gửi tên thủ tục và các đối số của nó tới SQL Server. Máy chủ xử lý tất cả mã của thủ tục và chỉ trả lại kết quả cho người gọi.

Cú pháp câu lệnh CREATE PROCEDURE dùng để tạo một thủ tục lưu trữ cung cấp rất nhiều tùy chọn linh hoạt và mở rộng, có dạng như sau:

```
CREATE PROCEDURE procedure_name
    [ { @parameter_name } datatype [= default_value] [OUTPUT] ] , [ ,...n ]
    [ { WITH [RECOMPILE | ENCRYPTION | RECOMPILE,
        ENCRYPTION | EXECUTE AS clause] } ]
AS
    [BEGIN]
    statements
    [END]
```

Một số thủ tục có thể yêu cầu cung cấp thông tin để thực hiện; điều này đạt được bằng cách truyền vào một tham số. Có thể truyền nhiều hơn một tham số: chỉ cần phân tách các tham số này bằng dấu phẩy. Tham số phải được định nghĩa với tiền tố là ký hiệu @. Không phải tất cả các thủ tục sẽ yêu cầu tham số, vì vậy đây là tùy chọn; tuy nhiên, nếu muốn truyền tham số cho một thủ tục lưu trữ, hãy đặt tên cho tham số, theo sau là kiểu dữ liệu và, nếu cần thiết, độ dài của dữ liệu cần truyền vào. Ví dụ sau đây định nghĩa một tham số có tên L\_Name, với kiểu dữ liệu varchar có độ dài là 50:

```
@L_Name varchar(50)
```

Chúng ta cũng có thể chỉ định một giá trị mặc định cho tham số trong trường hợp người dùng không cung cấp giá trị này tại thời điểm thực thi. Giá trị được chỉ định phải là một giá trị không đổi, như 'DEFAULT' hoặc 24031964 hoặc có thể là NULL. Không thể định nghĩa một biến là giá trị mặc định, vì thủ tục không thể giải quyết điều này khi được biên dịch. Ví dụ, nếu ứng dụng thường được bộ phận tiếp thị sử dụng, nhưng không dành riêng, thì có thể đặt giá trị mặc định của tham số department là 'marketing':

```
@department varchar(50) = 'marketing'
```

Vì vậy, nếu người dùng đến từ bộ phận tiếp thị thì sẽ không cần cung cấp thông tin đầu vào của bộ phận. Tuy nhiên, nếu người dùng đến từ các bộ phận khác, cần cung cấp một giá trị và sẽ được ghi đè lên giá trị mặc định.

Thủ tục lưu trữ cũng có thể trả về một hoặc nhiều giá trị hoặc thậm chí một bảng dữ liệu bằng cách sử dụng tham số để truyền thông tin ra ngoài. Tham số sẽ vẫn được định nghĩa như tham số đầu vào, với một ngoại lệ và một tùy chọn bổ sung.

Lưu ý quan trọng cần ghi nhớ là không thể xác định giá trị mặc định cho tham số này. Nếu chúng ta cố gắng làm như vậy, sẽ không phát sinh lỗi, nhưng định nghĩa này sẽ bị bỏ qua. Tùy chọn bổ sung được yêu cầu là thêm từ khóa OUTPUT vào sau tham số như ví dụ sau:

```
@calc_result varchar(50) OUTPUT
```

Các tham số đầu ra cũng có thể là các tham số đầu vào và do đó chúng có thể được sử dụng để truyền một giá trị vào cũng như trả về một giá trị.

Điều cuối cùng về các tham số cần được thảo luận, và có liên quan đến việc thực thi thủ tục và làm việc với các tham số đã định nghĩa. Khi nói đến việc thực thi một thủ tục lưu trữ có các tham số đầu vào, chúng ta có hai cách để thực thi thủ tục đó.

- Phương pháp đầu tiên là đặt tên cho thủ tục lưu trữ và sau đó truyền các giá trị đầu vào cho các tham số theo thứ tự mà chúng được định nghĩa. Sau đó, SQL Server sẽ lấy từng bộ giá trị được phân cách bằng dấu phẩy và gán nó cho biến đã định nghĩa. Tuy nhiên, điều này đưa ra giả định rằng thứ tự của các tham số không thay đổi và mọi tham số có giá trị mặc định luôn được gán trước một giá trị.
- Phương pháp thứ hai và được ưu tiên, là đặt tên cho tham số và truyền giá trị cho các tên này. Do đó đảm bảo rằng, tại thời điểm thực thi, các tham số đã được đặt tên theo thứ tự nào không quan trọng, bởi vì SQL Server sẽ có thể khớp tham số được định nghĩa với tham số đã được định nghĩa trong thủ tục lưu trữ. Ngoài ra, nếu thủ tục lưu trữ cần được mở rộng, để tương thích ngược, bất kỳ tham số mới nào cũng có thể được định nghĩa bằng các giá trị mặc định, do đó loại bỏ nhu cầu thay đổi mọi mã gọi. Cuối cùng, khi gọi một thủ tục lưu trữ từ mã chặng hạn như .NET, Excel,... mỗi tham số đều phải được đặt tên.

Tiếp theo là hai tùy chọn xác định cách xây dựng thủ tục lưu trữ. Trước hết, một thủ tục lưu trữ khi thực thi lần đầu tiên mà không có kế hoạch thực thi hiện có trong bộ đệm, sẽ được biên dịch thành một kế hoạch thực thi, là một cấu trúc dữ liệu nội bộ trong SQL Server mô tả cách thức thực hiện hoạt động được yêu cầu trong thủ tục lưu trữ. SQL Server lưu trữ mã đã biên dịch cho các lần thực thi tiếp theo, giúp tiết kiệm thời gian và tài nguyên. Nếu có một kế hoạch hiện có trong bộ đệm, thì SQL Server sẽ sử dụng kế hoạch này.

Tuy nhiên, tùy chọn RECOMPILE trên một thủ tục lưu trữ yêu cầu SQL Server rằng mỗi khi thủ tục lưu trữ được thực thi, toàn bộ thủ tục sẽ được biên dịch lại. Thông thường, khi một tham số có thể ảnh hưởng lớn đến số lượng hàng được trả về, chúng ta có thể muốn thêm tùy chọn RECOMPILE vào một thủ tục lưu trữ để buộc trình tối ưu hóa tạo ra kế hoạch tốt nhất cho mỗi lần (nghĩa là muốn tránh sử dụng lại một kế hoạch có thể không rất tốt cho các giá trị tham số nhất định).

Tùy chọn thứ hai là từ khóa ENCRYPTION. Có thể mã hóa - tốt, ít nhất là làm xáo trộn - một thủ tục lưu trữ để không thể xem nội dung của thủ tục lưu trữ một cách dễ dàng. Hãy nhớ rằng ENCRYPTION không bảo mật dữ liệu mà bảo vệ mã nguồn khỏi bị xem và sửa đổi. Cá ENCRYPTION và RECOMPILE đều được đặt sau từ khóa WITH và có thể được sử dụng cùng nhau khi được phân tách bằng dấu phẩy:

```
CREATE PROCEDURE sp_do_nothing @nothing INT  
WITH ENCRYPTION, RECOMPILE  
AS  
SELECT something  
FROM nothing;
```

Từ khóa AS xác định điểm bắt đầu của mã T-SQL, mã này sẽ là phần thân của thủ tục lưu trữ. AS không có chức năng nào khác, nhưng là bắt buộc trong lệnh CREATE PROCEDURE xác định phần cuối của tất cả các định nghĩa biến và tùy chọn tạo thủ tục. Sau khi từ khóa AS được xác định, chúng ta có thể bắt đầu tạo mã T-SQL của mình. Sau đó, có thể bao quanh mã bằng khối BEGIN...END. Chúng ta nên làm điều này để dễ xác định điểm bắt đầu và kết thúc của thủ tục.

Ví dụ đơn giản, đoạn mã sau tạo một thủ tục lưu trữ có tên là GetCustomerOrders. Thủ tục này chấp nhận một ID khách hàng (@custID) và một phạm vi ngày tháng (@fromDate và @toDate) làm đầu vào. Thủ tục trả về các hàng từ bảng Orders biểu diễn cho các đơn đặt hàng được đặt bởi khách hàng trong phạm vi ngày tháng cho trước dưới dạng tập kết quả và số lượng hàng bị ảnh hưởng dưới dạng tham số đầu ra (@numRows):

```
DROP PROC IF EXISTS GetCustomerOrders;  
GO  
CREATE PROC GetCustomerOrders  
    @custID AS INT,  
    @fromDate AS DATETIME = '19000101',  
    @toDate AS DATETIME = '99991231',  
    @numRows AS INT OUTPUT  
AS  
SET NOCOUNT ON;  
SELECT orderID,  
       custID,  
       empID,  
       orderDate  
  FROM Orders  
 WHERE custID = @custID  
       AND orderDate >= @fromDate  
       AND orderDate < @toDate;  
SET @numRows = @@rowcount;  
GO
```

## Giáo trình Hệ quản trị Cơ sở dữ liệu

Khi thực hiện thủ tục, nếu không chỉ định giá trị trong tham số @fromDate, thủ tục sẽ sử dụng giá trị mặc định 19000101 và nếu không chỉ định giá trị trong tham số @toDate, thủ tục sẽ sử dụng giá trị mặc định 99991231. Lưu ý việc sử dụng từ khóa OUTPUT để chỉ ra rằng tham số @numRows là tham số đầu ra. Lệnh SET NOCOUNT ON được sử dụng để chặn các thông báo cho biết có bao nhiêu hàng bị ảnh hưởng bởi các câu lệnh DML, chẳng hạn như câu lệnh SELECT trong thủ tục.

Sau đây là một ví dụ về việc thực thi thủ tục, yêu cầu thông tin về các đơn đặt hàng được đặt bởi khách hàng có ID là 1 trong năm 2015. Đoạn mã gán giá trị của biến cục bộ @rc cho tham số đầu ra @numRows và trả về để hiển thị số lượng các hàng bị ảnh hưởng bởi truy vấn:

```
DECLARE @rc AS INT;
EXEC GetCustomerOrders @custID = 1,
                      @fromDate = '20150101',
                      @toDate = '20160101',
                      @numRows = @rc OUTPUT;
SELECT @rc AS numRows;
```

Mã trên trả về kết quả sau hiển thị có ba đơn đặt hàng đáp ứng yêu cầu:

orderID	custID	empID	orderDate
10643	1	6	2015-08-25
10692	1	4	2015-10-03
10702	1	4	2015-10-13

**numRows**

3

Thực thi lại mã trên, cung cấp ID khách hàng không tồn tại trong bảng Orders (ví dụ, ID khách hàng 100). Chúng ta sẽ nhận được kết quả sau đây cho biết rằng không có đơn đặt hàng nào đáp ứng yêu cầu:

**orderID**      **custID**      **empID**      **orderDate**


**numRows**

0

## 3.7 TRÌNH KÍCH HOẠT

Trình kích hoạt (trigger) là một loại thủ tục lưu trữ đặc biệt - không thể được gọi thực thi một cách rõ ràng. Thay vào đó, phải được đính kèm với một sự kiện. Bất cứ khi nào sự kiện diễn ra, trình kích hoạt sẽ được kích hoạt và mã trong trình kích hoạt sẽ được thực thi. SQL Server hỗ trợ liên kết trình kích hoạt với hai loại sự kiện: sự kiện thao tác dữ liệu (trình kích hoạt DML) chẳng hạn như INSERT, DELETE, hoặc UPDATE và sự kiện định nghĩa dữ liệu (trình kích hoạt DDL) chẳng hạn như CREATE TABLE.

Trình kích hoạt có thể được sử dụng cho nhiều mục đích, bao gồm giám sát (auditing), thực thi các quy tắc toàn vẹn không thể thực thi bằng các ràng buộc và thực thi chính sách.

Các nhà phát triển sẽ chủ yếu sử dụng trình kích hoạt DML; trình kích hoạt DDL chủ yếu được quản trị viên cơ sở dữ liệu sử dụng để đảm bảo rằng một số hành động nhất định không thể xảy ra hoặc sẽ được báo cáo.

Không như thủ tục lưu trữ, chúng ta không thể gọi trình kích hoạt để thực thi, không thể truyền tham số cho trình kích hoạt và cuối cùng, trình kích hoạt không thể trả về giá trị.

### 3.7.1 Trình kích hoạt DML

Trình kích hoạt DML có nhiều công dụng, phổ biến nhất là thực thi quy tắc nghiệp vụ. Ví dụ, khi khách hàng đặt hàng, trình kích hoạt sẽ kiểm tra xem họ có đủ tiền trong tài khoản hay trong kho có đủ hàng không; nếu bất kỳ kiểm tra nào trong số này không thành công, ứng dụng có thể hoàn thành các hành động tiếp theo hoặc gửi lại thông báo lỗi và khôi phục cập nhật.

Trình kích hoạt DML có thể được sử dụng như một hình thức xác thực bổ sung - ví dụ, để thực hiện các kiểm tra phức tạp đối với dữ liệu mà một ràng buộc không thể thực hiện được. Hãy nhớ rằng việc sử dụng các ràng buộc thay vì trình kích hoạt sẽ mang lại hiệu suất tốt hơn, nhưng trình kích hoạt là lựa chọn tốt hơn khi xử lý xác thực dữ liệu phức tạp. Một cách sử dụng khác cho trình kích hoạt DML là thực hiện các thay đổi trong một bảng khác dựa trên những gì sắp xảy ra trong bảng được kích hoạt. Ví dụ, khi chúng ta thêm một đơn đặt hàng, sẽ tạo trình kích hoạt DML để giảm số lượng mặt hàng trong kho. Cuối cùng, trình kích hoạt DML có thể được sử dụng để tạo đường dẫn kiểm tra tự động tạo lịch sử thay đổi cho từng bản ghi.

Chúng ta không thể tạo trình kích hoạt cho SELECT. Điều này là hiển nhiên, vì không có sửa đổi bảng nào xảy ra trong câu lệnh SELECT nên không thể tạo trình kích hoạt như thế.

Trình kích hoạt có thể cập nhật bảng trong các cơ sở dữ liệu khác nếu muốn và trình kích hoạt cũng có thể mở rộng các máy chủ, vì vậy đừng nghĩ rằng phạm vi của trình kích hoạt bị giới hạn ở cơ sở dữ liệu hoặc máy chủ hiện tại.

Trình kích hoạt có thể thực hiện sửa đổi dữ liệu, và do đó, có thể sẽ kích hoạt một trình kích hoạt khác, được gọi là trình kích hoạt lồng nhau (nested trigger). Ví dụ, hãy hình dung chúng ta có Bảng A, bảng này có một trình kích hoạt để thực hiện một sửa đổi trong Bảng B, bảng này lại có một trình kích hoạt để thực hiện một sửa đổi trong Bảng C. Nếu một sửa đổi được thực hiện đối với Bảng A, thì trình kích hoạt của Bảng A sẽ được kích hoạt, sửa đổi dữ liệu trong Bảng B, điều này sẽ kích hoạt trình kích hoạt trong Bảng B, dẫn đến sửa đổi dữ liệu trong Bảng C. Lưu ý rằng hiệu suất sẽ bị ảnh hưởng nghiêm trọng khi sử dụng các trình kích hoạt lồng nhau; vì thế, chỉ sử dụng chúng khi cần thiết.

T-SQL có một câu lệnh sẽ ngăn trình kích hoạt DELETE bị kích hoạt, đó là lệnh TRUNCATE TABLE.

Lưu ý quan trọng phải ghi nhớ khi xây dựng trình kích hoạt là trình kích hoạt sẽ kích hoạt sau mỗi câu lệnh sửa đổi dữ liệu được thực thi, không phải trước khi sửa đổi được thực hiện đối với các hàng trong bảng. Do đó, nếu có một câu lệnh cập nhật nhiều hàng, trình kích hoạt sẽ kích hoạt sau khi hoàn tất tất cả các cập nhật được yêu cầu, không phải khi từng hàng đã được xử lý.

Việc tạo trình kích hoạt thông qua mã T-SQL có thể khá phức tạp nếu sử dụng cú pháp trình kích hoạt đầy đủ. Tuy nhiên, phiên bản rút gọn sau đây sẽ dễ hiểu hơn. Khi tạo trình kích hoạt, chúng ta có thể tạo trình kích hoạt cho một hành động hoặc cho nhiều hành động. Để mở rộng vấn đề này, trình kích hoạt có thể chỉ dành cho việc chèn bản ghi hoặc có thể bao gồm việc chèn và cập nhật bản ghi. Bất kỳ sự kết hợp là có thể.

```
CREATE TRIGGER [schema_name.]trigger_name  
ON {table|view}  
    [WITH ENCRYPTION]  
    {  
        {{FOR {AFTER|INSTEAD OF} {[INSERT] [,] [UPDATE] [,]  
        [DELETE]} }  
        AS  
        [{IF [UPDATE (column)  
        [{AND|OR} UPDATE (column)]]} ]  
        COLUMNS_UPDATE()  
        sql_statements } }
```

- schema\_name.trigger\_name: tên của trình kích hoạt, phải tuân theo các tiêu chuẩn dùng để đặt tên cho các đối tượng trong cơ sở dữ liệu và trình kích hoạt cũng phải thuộc về một lược đồ giống như các đối tượng khác.
- ON {table|view}: tên của bảng hoặc khung nhìn mà trình kích hoạt gắn vào. Mỗi trình kích hoạt chỉ được gắn vào một bảng.
- [WITH ENCRYPTION]: trình kích hoạt có thể được mã hóa mã lệnh bằng cách sử dụng tùy chọn WITH ENCRYPTION.
- {FOR|AFTER|INSTEAD OF}:
  - Trình kích hoạt FOR|AFTER sẽ thực thi mã T-SQL bên trong sau khi dữ liệu trong bảng đã được sửa đổi. Do đó, nếu có bất kỳ ràng buộc nào trên bảng đối với các thay đổi lan truyền, thì bảng sẽ thay đổi và các thay đổi lan truyền này sẽ hoàn tất thành công trước khi trình kích hoạt được kích hoạt.
  - Trình kích hoạt INSTEAD OF: trình kích hoạt được định nghĩa bằng tùy chọn này sẽ thực thi mã T-SQL bên trong thay vì cho phép sửa đổi dữ liệu.

Nói cách khác, nếu một bảng có trình kích hoạt INSTEAD OF gắn với hành động INSERT, thì hoạt động chèn sẽ không thực hiện được.

- {[INSERT] [,] [UPDATE] [,] [DELETE]}: Phân cú pháp này xác định (các) hành động mà trình kích hoạt sẽ thực hiện. Trình kích hoạt có thể kích hoạt một, hai hoặc ba trong số các lệnh này, tùy thuộc vào những gì chúng ta muốn trình kích hoạt thực hiện.
- AS: từ khóa AS xác định điểm bắt đầu của mã kích hoạt, giống như từ khóa AS xác định thời điểm bắt đầu của một thủ tục lưu trữ.
- [{IF UPDATE (column) [{AND|OR} UPDATE (column)}]]: tùy chọn này là một phép thử để kiểm tra xem một cột cụ thể đã được sửa đổi hay chưa. Điều này xảy ra thông qua việc sử dụng từ khóa UPDATE(). Bằng cách đặt tên của cột cần kiểm tra trong dấu ngoặc đơn, giá trị logic TRUE hoặc FALSE sẽ được trả về tùy thuộc vào việc cột đã được cập nhật hay chưa. Việc xóa bản ghi sẽ không đặt kiểm tra UPDATE thành TRUE hoặc FALSE, vì đang xóa một mục và không cập nhật mục đó. INSERT hoặc UPDATE sẽ đặt kiểm tra UPDATE thành giá trị cần thiết.
- COLUMNS\_UPDATE(): Chức năng này tương tự như UPDATE(), nhưng thay vì kiểm tra một cột có tên cụ thể, sẽ kiểm tra nhiều cột trong một lần kiểm tra.
- sql\_statements: Tại thời điểm này, chúng ta viết mã trình kích hoạt giống như thủ tục lưu trữ.

Trong mã của trình kích hoạt, chúng ta có thể truy cập các bảng giả (pseudo table) được gọi là inserted và deleted chứa các hàng bị ảnh hưởng bởi sửa đổi khiến trình kích hoạt bị kích hoạt. Bảng inserted giữ hình ảnh mới của các hàng bị ảnh hưởng trong trường hợp các hành động INSERT và UPDATE. Bảng deleted giữ hình ảnh cũ của các hàng bị ảnh hưởng trong trường hợp các hành động DELETE và UPDATE. Lưu ý là các hành động INSERT, UPDATE và DELETE có thể được gọi bằng các câu lệnh INSERT, UPDATE và DELETE cũng như bằng câu lệnh MERGE. Trong trường hợp trình kích hoạt instead of, các bảng inserted và deleted chứa các hàng được cho là bị ảnh hưởng bởi sửa đổi khiến trình kích hoạt bị kích hoạt.

Ví dụ đơn giản sau về trình kích hoạt after giám sát thao tác chèn vào một bảng. Thực thi đoạn mã sau để tạo một bảng có tên là dbo.T1 trong cơ sở dữ liệu hiện tại và một bảng khác có tên là dbo.T1\_Audit chứa thông tin giám sát hoạt động chèn vào T1:

```
DROP TABLE IF EXISTS dbo.T1_Audit,
                    dbo.T1;

CREATE TABLE dbo.T1
(
    keycol INT NOT NULL PRIMARY KEY,
    datacol VARCHAR(10) NOT NULL
);
```

```

CREATE TABLE dbo.T1_Audit
(
    audit_lsn INT NOT NULL IDENTITY PRIMARY KEY,
    dt DATETIME2(3) NOT NULL
        DEFAULT (SYSDATETIME()),
    login_name sysname NOT NULL
        DEFAULT (ORIGINAL_LOGIN()),
    keycol INT NOT NULL,
    datacol VARCHAR(10) NOT NULL
);

```

Trong bảng giám sát dbo.T1\_Audit, cột audit\_lsn có thuộc tính identity và biểu diễn số tuần tự của nhật ký giám sát. Cột dt biểu diễn ngày và thời gian thực hiện thao tác chèn, sử dụng biểu thức mặc định SYSDATETIME(). Cột login\_name biểu diễn tên của login đã thực hiện thao tác chèn, sử dụng biểu thức mặc định ORIGINAL\_LOGIN().

Tiếp theo, hãy thực thi đoạn mã sau để tạo trình kích hoạt AFTER INSERT trg\_T1\_insert\_audit trên bảng T1 để giám sát các lần chèn:

```

CREATE TRIGGER trg_T1_insert_audit
ON dbo.T1
AFTER INSERT
AS
SET NOCOUNT ON;
INSERT INTO dbo.T1_Audit
(
    keycol,
    datacol
)
SELECT keycol,
       datacol
FROM inserted;
GO

```

Như có thể thấy, trình kích hoạt chỉ cần chèn vào bảng giám sát dbo.T1\_Audit kết quả của một truy vấn trên bảng inserted. Các giá trị của các cột trong bảng giám sát không được liệt kê rõ ràng trong câu lệnh INSERT được tạo bởi các biểu thức mặc định được mô tả trước đó. Để kiểm tra trình kích hoạt, hãy thực thi đoạn mã sau:

```

INSERT INTO dbo.T1 (keycol, datacol) VALUES (10, 'a');
INSERT INTO dbo.T1 (keycol, datacol) VALUES (30, 'x');
INSERT INTO dbo.T1 (keycol, datacol) VALUES (20, 'g');

```

Trình kích hoạt bị kích hoạt sau mỗi câu lệnh. Tiếp theo, truy vấn bảng giám sát:

```

SELECT audit_lsn, dt,
       login_name, keycol, datacol
FROM dbo.T1_Audit;

```

Chúng ta nhận được kết quả sau, chỉ với các giá trị dt và login\_name phản ánh ngày và giờ khi thực hiện các thao tác chèn và thông tin đăng nhập đã sử dụng để kết nối với SQL Server:

<b>audit_lsn</b>	<b>dt</b>	<b>login_name</b>	<b>keycol</b>	<b>datacol</b>
1	2016-02-12 09:04:27.713	K2\Gandalf	10	a
2	2016-02-12 09:04:27.733	K2\Gandalf	30	x
3	2016-02-12 09:04:27.733	K2\Gandalf	20	g

Khi đã hoàn tất, hãy thực thi đoạn mã sau để dọn dẹp:

```
DROP TABLE dbo.T1_Audit,
      dbo.T1;
```

### 3.7.2 Trình kích hoạt DDL

SQL Server hỗ trợ trình kích hoạt DDL (DDL trigger), có thể được sử dụng cho các mục đích như giám sát, thực thi chính sách và quản lý thay đổi. SQL Server hỗ trợ tạo trình kích hoạt DDL ở hai phạm vi, phạm vi cơ sở dữ liệu và phạm vi máy chủ, tùy thuộc vào phạm vi của sự kiện. Cơ sở dữ liệu Azure SQL hiện chỉ hỗ trợ trình kích hoạt DDL ở phạm vi cơ sở dữ liệu.

Chúng ta có thể tạo trình kích hoạt cơ sở dữ liệu (database trigger) cho các sự kiện có phạm vi cơ sở dữ liệu, chẳng hạn như CREATE TABLE; tạo trình kích hoạt (all server trigger) cho các sự kiện có phạm vi máy chủ, chẳng hạn như CREATE DATABASE. SQL Server chỉ hỗ trợ trình kích hoạt DDL after; không hỗ trợ trình kích hoạt DDL instead of.

Trong trình kích hoạt, chúng ta có thể lấy thông tin về sự kiện khiến trình kích hoạt bị kích hoạt bằng cách truy vấn một hàm có tên là EVENTDATA, hàm này trả về thông tin sự kiện dưới dạng một thẻ hiện XML. Chúng ta có thể sử dụng các biểu thức XQuery để trích xuất các thuộc tính sự kiện chẳng hạn như thời gian xảy ra sự kiện, loại sự kiện và tên đăng nhập từ thẻ hiện XML.

Đoạn mã sau tạo bảng dboAuditDDLEvents chứa thông tin giám sát:

```
DROP TABLE IF EXISTS dbo.AuditDDLEvents;
CREATE TABLE dbo.AuditDDLEvents
(
    audit_lsn INT NOT NULL IDENTITY,
    postTime DATETIME2(3) NOT NULL,
    eventType sysname NOT NULL,
    loginName sysname NOT NULL,
    schemaName sysname NOT NULL,
    objectName sysname NOT NULL,
    targetObjectName sysname NULL,
    eventData XML NOT NULL,
    CONSTRAINT PK_AuditDDLEvents
        PRIMARY KEY (audit_lsn)
);
```

Lưu ý rằng bảng có một cột được gọi là eventData có kiểu dữ liệu XML. Ngoài các thuộc tính riêng lẻ mà trình kích hoạt trích xuất từ thông tin sự kiện và lưu trữ trong các thuộc tính riêng lẻ, trình kích hoạt cũng lưu trữ toàn bộ thông tin sự kiện trong cột eventData.

Thực thi đoạn mã sau để tạo trình kích hoạt giám sát trg\_audit\_ddl\_events trên cơ sở dữ liệu bằng cách sử dụng nhóm sự kiện DDL\_DATABASE\_LEVEL\_EVENTS, đại diện cho tất cả các sự kiện DDL ở mức cơ sở dữ liệu:

```
CREATE TRIGGER trg_audit_ddl_events
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
SET NOCOUNT ON;
DECLARE @eventData AS XML = EVENTDATA();
INSERT INTO dbo.AuditDDLEvents (postTime, eventType,
                                loginName, schemaName, objectName,
                                targetObjectName, eventData)
VALUES
(@EventData.value('/EVENT_INSTANCE/PostTime')[1] ,
   'VARCHAR(23)'),
@EventData.value('/EVENT_INSTANCE/EventType')[1] ,
   'sysname'),
@EventData.value('/EVENT_INSTANCE/LoginName')[1] ,
   'sysname'),
@EventData.value('/EVENT_INSTANCE/SchemaName')[1] ,
   'sysname'),
@EventData.value('/EVENT_INSTANCE/ObjectName')[1] ,
   'sysname'),
@EventData.value('/EVENT_INSTANCE/TargetObjectName')[1] ,
   'sysname'),
@EventData);
GO
```

Trước tiên, mã của trình kích hoạt lưu trữ thông tin sự kiện thu được từ hàm EVENTDATA trong biến @EventData. Sau đó, mã sẽ chèn một hàng vào bảng giám sát với các thuộc tính được trích xuất bằng cách sử dụng các biểu thức XQuery bằng phương thức .value từ thông tin sự kiện, cộng với thể hiện XML có thông tin sự kiện đầy đủ.

Để kiểm tra trình kích hoạt, hãy thực thi đoạn mã sau, chứa một vài câu lệnh DDL:

```
CREATE TABLE dbo.T1
(
    col1 INT NOT NULL PRIMARY KEY
);
ALTER TABLE dbo.T1 ADD col2 INT NULL;
ALTER TABLE dbo.T1 ALTER COLUMN col2 INT NOT NULL;
CREATE NONCLUSTERED INDEX idx1 ON dbo.T1 (col2);
```

---

Tiếp theo, thực thi đoạn mã sau để truy vấn bảng giám sát:

```
SELECT *
FROM dbo.AuditDDLEvents;
```

Chúng ta nhận được kết quả sau (được chia ở đây thành hai phần cho mục đích hiển thị), nhưng với các giá trị trong thuộc tính postTime và loginName phản ánh thời gian xảy ra sự kiện và tên đăng nhập:

audit_lsn	postTime	eventType	loginName	
1	2016-02-12 09:06:18.293	CREATE_TABLE	K2\Gandalf	
2	2016-02-12 09:06:18.413	ALTER_TABLE	K2\Gandalf	
3	2016-02-12 09:06:18.423	ALTER_TABLE	K2\Gandalf	
4	2016-02-12 09:06:18.423	CREATE INDEX	K2\Gandalf	
audit_lsn	schemaName	objectName	targetObjectName	eventData
1	dbo	T1	NULL	<EVENT_INSTANCE>..
2	dbo	T1	NULL	<EVENT_INSTANCE>..
3	dbo	T1	NULL	<EVENT_INSTANCE>..
4	dbo	idx1	T1	<EVENT_INSTANCE>..

Khi đã hoàn tất, hãy chạy đoạn mã sau để dọn dẹp:

```
DROP TRIGGER IF EXISTS trg_audit_ddl_events ON DATABASE;
DROP TABLE IF EXISTS dbo.AuditDDLEvents;
```

## TÓM TẮT

- Câu lệnh SELECT là câu lệnh quan trọng nhất trong ngôn ngữ dữ liệu và được sử dụng để diễn đạt một truy vấn. Nó kết hợp ba phép toán đại số quan hệ cơ bản là phép Chọn (Selection), phép Chiếu (Projection) và phép Nối (Join). Mỗi câu lệnh SELECT tạo ra một bảng kết quả bao gồm một hoặc nhiều cột và không hoặc nhiều hàng.
- Mệnh đề SELECT xác định các cột và/hoặc dữ liệu được tính toán để xuất hiện trong bảng kết quả. Tất cả các tên cột xuất hiện trong mệnh đề SELECT phải có trong bảng hoặc khung nhìn tương ứng được liệt kê trong mệnh đề FROM.
- Mệnh đề WHERE chọn các hàng đưa vào bảng kết quả bằng cách áp dụng điều kiện tìm kiếm cho các hàng của (các) bảng được đặt tên. Mệnh đề ORDER BY cho phép bảng kết quả được sắp xếp theo các giá trị trong một hoặc nhiều cột. Mỗi cột có thể được sắp xếp theo thứ tự tăng dần hoặc giảm dần. Nếu được chỉ định, mệnh đề ORDER BY phải là mệnh đề cuối cùng trong câu lệnh SELECT.
- Nếu các cột của bảng kết quả đến từ nhiều bảng thì phải sử dụng phép nối (join), bằng cách chỉ định nhiều bảng trong mệnh đề FROM và thường bao gồm mệnh đề WHERE để chỉ định (các) cột so khớp. SQL chuẩn cho phép các phép nối bên ngoài (outer join) được xác định. Nó cũng cho phép các phép toán tập hợp như Union, Intersection và Difference được sử dụng thông qua các lệnh UNION, INTERSECT và EXCEPT.
- SQL chuẩn cung cấp tám kiểu dữ liệu cơ sở: boolean, ký tự, bit, số chính xác, số gần đúng, ngày giờ, khoảng thời gian và các đối tượng lớn.
- Khung nhìn (view) là một bảng ảo biểu diễn một tập con gồm các cột và/hoặc các hàng và/hoặc các biểu thức cột từ một hoặc nhiều bảng cơ sở hoặc khung nhìn khác. Khung nhìn được tạo bằng cách sử dụng câu lệnh CREATE VIEW dựa trên một truy vấn định nghĩa.
- Các khung nhìn có thể được sử dụng để đơn giản hóa cấu trúc của cơ sở dữ liệu và làm cho các truy vấn dễ viết hơn. Chúng cũng có thể được sử dụng để bảo vệ các cột và/hoặc hàng nhất định không bị truy cập trái phép.
- Phân giải khung nhìn là kết hợp truy vấn trên một khung nhìn với truy vấn định nghĩa khung nhìn tạo ra một truy vấn trên (các) bảng cơ sở bên dưới. Quá trình này được thực hiện mỗi khi DBMS phải xử lý một truy vấn trên một khung nhìn. Một cách tiếp cận thay thế, được gọi là cụ thể hóa khung nhìn, lưu khung nhìn dưới dạng bảng tạm thời trong cơ sở dữ liệu khi khung nhìn được tham chiếu lần đầu tiên. Sau đó, các truy vấn trên khung nhìn được cụ thể hóa này có thể nhanh hơn nhiều so với việc tính toán lại khung nhìn mỗi lần được truy xuất. Một bất lợi với khung nhìn được cụ thể hóa là duy trì tính tức thời của bảng tạm thời.

- Các chương trình con (routine) là các khối T-SQL được đặt tên, có thể nhận các tham số và được gọi. T-SQL có hai loại chương trình con được gọi là thủ tục (procedure) và hàm (function). Các thủ tục và hàm có thể nhận một tập hợp các tham số do chương trình gọi cung cấp và thực hiện một tập hợp các hành động. Cả hai đều có thể sửa đổi và trả về dữ liệu được truyền cho chúng dưới dạng tham số. Sự khác biệt giữa một thủ tục và một hàm là hàm sẽ luôn trả về một giá trị duy nhất cho trình gọi, trong khi thủ tục thì không.
- Trình kích hoạt (trigger) định nghĩa một hành động mà cơ sở dữ liệu sẽ thực hiện khi một sự kiện nào đó xảy ra trong ứng dụng. Một trigger có thể được sử dụng để thực thi một số ràng buộc toàn vẹn tham chiếu, thực thi các ràng buộc toàn vẹn phức tạp hoặc kiểm tra các thay đổi đối với dữ liệu. Trigger dựa trên mô hình Sự kiện-Điều kiện-Hành động (Event-Condition-Action - ECA): sự kiện (event) hoặc các sự kiện kích hoạt quy tắc, điều kiện (condition) xác định xem hành động có nên được thực hiện hay không và hành động (action) sẽ được thực hiện.
- Ưu điểm của trigger bao gồm: loại bỏ mã thừa, đơn giản hóa các sửa đổi, tăng tính bảo mật, cải thiện tính toàn vẹn, cải thiện sức mạnh xử lý và phù hợp tốt với kiến trúc máy khách-máy chủ. Tuy nhiên, trigger cũng có một số hạn chế như: chi phí hiệu suất, hiệu ứng lan truyền, không có khả năng được lập lịch và ít di động hơn.

## CÂU HỎI

1. Mô tả ngắn gọn bốn câu lệnh SQL DML cơ bản và giải thích việc sử dụng chúng.
2. Giải thích tầm quan trọng và ứng dụng của mệnh đề WHERE trong câu lệnh UPDATE và DELETE.
3. Giải thích chức năng của mỗi mệnh đề trong câu lệnh SELECT. Những hạn chế nào được đặt ra đối với các mệnh đề này?
4. Nêu định nghĩa và tầm quan trọng của khung nhìn trong cách tiếp cận cơ sở dữ liệu?
5. Thảo luận về ưu điểm và nhược điểm của các khung nhìn.
6. Những hạn chế nào là cần thiết để đảm bảo một khung nhìn có thể cập nhật được?
7. Khung nhìn được cụ thể hóa là gì và những ưu điểm của việc duy trì khung nhìn được cụ thể hóa là gì?
8. Mô tả sự khác biệt giữa Điều khiển truy cập tùy quyền (DAC) và Điều khiển truy cập bắt buộc (MAC). SQL chuẩn hỗ trợ loại cơ chế điều khiển nào?
9. Mô tả các phần tử điều khiển trong T-SQL. Cho ví dụ minh họa.
10. Thủ tục (procedure) khác với hàm (function) như thế nào?
11. Trình bày sự khác biệt giữa các trigger AFTER và INSTEAD OF. Cho ví dụ minh họa.
12. Trình bày về ưu điểm và hạn chế của trigger.

## BÀI TẬP

### 1.

Sử dụng lược đồ cơ sở dữ liệu Hotel sau:

**Hotel**(hotelNo, hotelName, city)

**Room**(roomNo, hotelNo, type, price)

**Booking**(hotelNo, guestNo, dateFrom, dateTo, roomNo)

**Guest**(guestNo, guestName, guestAddress)

Với:

Hotel chứa thông tin chi tiết về khách sạn và hotelNo là khóa chính;

Room chứa thông tin chi tiết về phòng của từng khách sạn và (roomNo, hotelNo) tạo thành khóa chính;

Booking chứa thông tin chi tiết về đặt phòng và (hotelNo, guestNo, dateFrom) tạo thành khóa chính;

Guest chứa thông tin chi tiết về khách và guestNo là khóa chính.

- a. Liệt kê đầy đủ chi tiết của tất cả các khách sạn.
- b. Liệt kê chi tiết đầy đủ của tất cả các khách sạn ở London.
- c. Liệt kê tên và địa chỉ của tất cả khách sống ở London, được sắp xếp theo thứ tự tên.
- d. Tổng doanh thu mỗi đêm từ tất cả các phòng đôi là bao nhiêu?
- e. Có bao nhiêu khách khác nhau đã đặt phòng cho tháng Tám?
- f. Viết thủ tục liệt kê số lượng phòng trong mỗi khách sạn.
- g. Viết thủ tục thêm 1 dòng vào bảng Hotel.
- h. Viết hàm với tham số là hotelNo và trả về số lượng phòng của khách sạn đó.
- i. Viết hàm cho biết số lượng đặt phòng trung bình của mỗi khách sạn trong tháng Tám là bao nhiêu?
- j. Xây dựng thủ tục hiển thị loại phòng được đặt nhiều nhất của mỗi khách sạn ở London.
- k. Xây dựng hàm cập nhật giá tất cả các phòng tăng 5%.
- l. Xây dựng hàm trả về số lượng khách sạn tại một thành phố cụ thể.
- m. Xây dựng trigger khi thêm mới một dòng vào bảng Booking để thực hiện kiểm tra dateFrom phải nhỏ hơn hoặc bằng dateTo.
- n. Xây dựng trigger khi xóa một dòng trong bảng Room để kiểm tra nếu roomNo đã tồn tại trong Booking thì không thực hiện hoạt động xóa.

## 2.

Sử dụng lược đồ cơ sở dữ liệu Projects sau:

**Employee(empNo, fName, lName, address, DOB, sex, position, deptNo)**

**Department(deptNo, deptName, mgrEmpNo)**

**Project(projNo, projName, deptNo)**

**WorksOn(empNo, projNo, dateWorked, hoursWorked)**

Với:

Employee chứa thông tin chi tiết của nhân viên và empNo là khóa chính.

Department chứa thông tin chi tiết bộ phận và deptNo là khóa chính. mgrEmpNo xác định nhân viên là người quản lý của bộ phận. Chỉ có một người quản lý cho mỗi bộ phận.

Project chứa thông tin chi tiết về các dự án trong từng bộ phận và khóa chính là projNo (không có hai bộ phận nào có thể chạy cùng một dự án).

WorksOn chứa thông tin chi tiết về số giờ làm việc của nhân viên trong mỗi dự án, và (empNo, projNo, dateWorked) tạo thành khóa chính

Hãy thực hiện các truy vấn sau:

- a. Liệt kê tất cả các thông tin chi tiết của các những nhân viên có năm sinh từ 1980-1990.
- b. Liệt kê tất cả người quản lý là nữ theo thứ tự bảng chữ cái của Họ, sau đó đến Tên.
- c. Loại bỏ tất cả các dự án do bộ phận Kế hoạch quản lý.
- d. Giả sử bộ phận Kế hoạch sẽ được hợp nhất với bộ phận CNTT. Cập nhật hồ sơ nhân viên để phản ánh được điều này.
- e. Xây dựng thủ tục trả về tất cả các dự án được quản lý bởi bộ phận CNTT và bộ phận Nhân sự.
- f. Lập báo cáo tổng số giờ làm việc của từng nhân viên nữ, được sắp xếp theo số bộ phận và theo Họ của nhân viên trong mỗi bộ phận.
- g. Xây dựng hàm trả về tuổi của nhân viên với DOB là tham số đầu vào.
- h. Xây dựng trigger khi thêm một dòng vào bảng WorksOn để kiểm tra ràng buộc là mỗi dự án chỉ có thể có tối đa 8 nhân viên tham gia.
- i. Xây dựng trigger khi thêm hoặc cập nhật một dòng trong bảng Employee để đảm bảo ràng buộc là nhân viên phải từ 18 tuổi trở lên.

### 3.

Sử dụng lược đồ cơ sở dữ liệu Library sau:

**Book**(ISBN, title, edition, year)

**BookCopy**(copyNo, ISBN, available)

**Borrower**(borrowerNo, borrowerName, borrowerAddress)

**BookLoan**(copyNo, dateOut, dateDue, borrowerNo)

Với:

Book chứa thông tin chi tiết về các đầu sách trong thư viện và ISBN là khóa chính.

BookCopy chứa thông tin chi tiết về các bản sao sách riêng lẻ trong thư viện và copyNo là khóa chính. ISBN là một khóa ngoại xác định đầu sách.

Borrower chứa thông tin chi tiết về các thành viên thư viện có thể mượn sách và borrowerNo là khóa chính.

BookLoan chứa thông tin chi tiết về các bản sao sách được các thành viên thư viện mượn và (copyNo, dateOut) tạo thành khóa chính. borrowerNo là khóa ngoại xác định người mượn.

Hãy thực hiện các truy vấn sau:

- a. Liệt kê tất cả các tên sách.
- b. Liệt kê tất cả thông tin chi tiết của tất cả người mượn.
- c. Xây dựng view chứa tất cả các tên sách được xuất bản từ năm 2010 đến năm 2014.
- d. Xóa tất cả các sách đã xuất bản trước năm 1950 khỏi cơ sở dữ liệu.
- e. Liệt kê tất cả các tên sách chưa từng được mượn.
- f. Liệt kê tất cả các tên sách có chứa từ ‘database’ và có thể cho mượn.
- g. Xây dựng view chứa tên những người mượn sách đã quá hạn trả.
- h. Xây dựng hàm trả về số lần đã được mượn của một quyển sách với tham số đầu vào là ISBN của quyển sách đó.
- i. Xây dựng thủ tục trả về các tên sách đã được một độc giả thuê mượn.
- j. Xây dựng trigger khi thêm một dòng vào bảng Book thực hiện ràng buộc năm xuất bản không được vượt quá năm hiện tại.
- k. Xây dựng trigger khi xóa một dòng trong Book để tránh trường hợp xóa một quyển sách đang được độc giả mượn.

## CHƯƠNG 4

### QUẢN LÝ GIAO DỊCH VÀ PHỤC HỒI

Trong chương này, chúng ta sẽ tìm hiểu:

- Mục đích của điều khiển đồng thời.
- Chức năng và tầm quan trọng của giao dịch.
- Các thuộc tính của một giao dịch.
- Ý nghĩa của khả năng tuần tự hóa và áp dụng cho điều khiển đồng thời.
- Các khóa có thể được sử dụng như thế nào để đảm bảo khả năng tuần tự hóa.
- Cách thức hoạt động của giao thức khóa hai pha (two-phase locking - 2PL).
- Ý nghĩa của deadlock và cách giải quyết.
- Sử dụng dấu thời gian (timestamp) để đảm bảo khả năng tuần tự hóa.
- Kỹ thuật điều khiển đồng thời lạc quan hoạt động như thế nào.
- Mục đích của tập tin nhật ký giao dịch.
- Mục đích của điểm kiểm tra (checkpoint) trong quá trình ghi nhật ký giao dịch.
- Cách khôi phục dựa trên giao dịch khi cơ sở dữ liệu bị lỗi.

Trong số các chức năng, dịch vụ mà một hệ quản trị cơ sở dữ liệu (DBMS) nên cung cấp, có ba chức năng liên quan chặt chẽ nhau nhằm đảm bảo rằng cơ sở dữ liệu đáng tin cậy và được duy trì ở trạng thái nhất quán: hỗ trợ giao dịch, dịch vụ điều khiển đồng thời và dịch vụ phục hồi. Độ tin cậy và tính nhất quán này phải được duy trì ngay cả khi có xảy ra sự cố, và khi nhiều người dùng truy cập đồng thời cơ sở dữ liệu.

Hầu hết DBMS cho phép nhiều người dùng thực hiện các hoạt động đồng thời trên cơ sở dữ liệu. Và nếu các hoạt động này không được kiểm soát, các truy cập có thể gây nhiễu lẫn nhau và cơ sở dữ liệu có thể trở nên không nhất quán. Để khắc phục điều này, DBMS triển khai một giao thức điều khiển đồng thời (concurrency control) để ngăn các truy cập xen vào nhau.

Phục hồi cơ sở dữ liệu (database recovery) là quá trình khôi phục cơ sở dữ liệu về trạng thái chính xác sau khi bị lỗi. Lỗi có thể là kết quả của sự cố hệ thống do phần cứng, phần mềm, hoặc phương tiện, chẳng hạn như sự cố trầy xước đĩa cứng hoặc lỗi phần mềm ứng dụng, chẳng hạn như lỗi logic trong chương trình đang truy cập cơ sở dữ liệu. Nó cũng có thể là kết quả của việc vô tình hoặc cố ý làm hỏng hoặc phá hủy dữ liệu hoặc phương tiện của người quản trị hệ thống hoặc người dùng. Bất kể nguyên nhân bên dưới của lỗi là gì, DBMS phải có khả năng phục hồi sau lỗi, đưa cơ sở dữ liệu về trạng thái nhất quán trước khi bị lỗi.

## 4.1 GIAO DỊCH

### Giao dịch

Một hành động hoặc một loạt các hành động, được thực hiện bởi một người dùng hoặc chương trình ứng dụng, đọc hoặc cập nhật nội dung của cơ sở dữ liệu

Một giao dịch (transaction) được xem như một đơn vị công việc logic (logical unit of work) trên cơ sở dữ liệu. Nó có thể là toàn bộ một chương trình, một phần của một chương trình, hoặc một câu lệnh đơn lẻ (ví dụ, câu lệnh SQL INSERT hoặc UPDATE) và có thể liên quan đến bất kỳ số lượng hoạt động nào trên cơ sở dữ liệu.

Để minh họa các khái niệm về giao dịch, chúng ta xem xét hai quan hệ của ví dụ *DreamHome*:

**Staff(staffNo, fName, lName, position, sex, DOB, salary, branchNo)**

**PropertyForRent(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)**

Một giao dịch đơn giản đối với cơ sở dữ liệu này là cập nhật mức lương của một nhân viên cụ thể với số nhân viên là x. Ở cấp độ cao, chúng ta có thể viết giao dịch này như trong Hình 4.1(a). Trong chương này, chúng ta biểu thị một thao tác đọc hoặc ghi cơ sở dữ liệu trên một mục dữ liệu x là `read(x)` hoặc `write(x)`. Các ký hiệu bổ sung có thể được thêm vào khi cần thiết; ví dụ, trong Hình 4.1(a), chúng ta đã sử dụng ký hiệu `read(staffNo = x, salary)` để chỉ ra rằng chúng ta muốn đọc mục dữ liệu lương của bộ có khóa chính là x. Trong ví dụ này, chúng ta có một giao dịch (transaction) bao gồm hai toán tử cơ sở dữ liệu (`read` và `write`) và một thao tác phi cơ sở dữ liệu (`salary = salary * 1.1`).

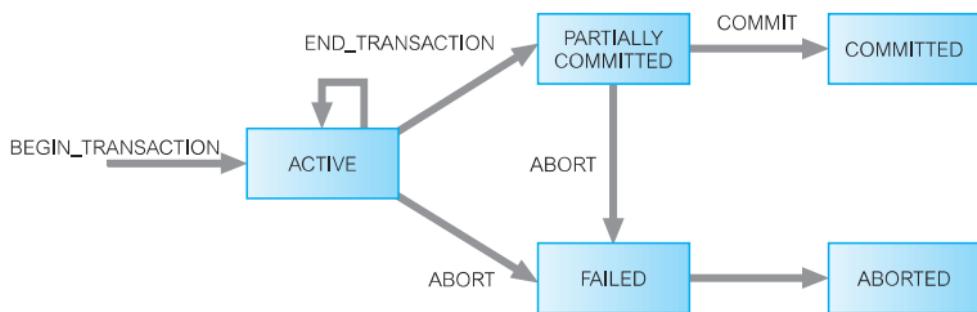
Một giao dịch phức tạp hơn là xóa nhân viên có số nhân viên nhất định là x, như trong Hình 4.1(b). Trong trường hợp này, ngoài việc phải xóa một bộ trong quan hệ Staff, chúng ta cũng cần phải tìm tất cả các bộ trong quan hệ PropertyForRent mà nhân viên này đang quản lý và gán chúng cho một nhân viên khác, `newStaffNo`. Nếu tất cả những cập nhật này không được thực hiện, tính toàn vẹn tham chiếu sẽ bị vi phạm và cơ sở dữ liệu sẽ ở trạng thái không nhất quán (inconsistent state): một hoặc nhiều bất động sản sẽ được quản lý bởi một nhân viên không còn tồn tại trong cơ sở dữ liệu.

<pre> delete(staffNo = x) for all PropertyForRent records, pno begin   read(staffNo = x, salary)   salary = salary * 1.1   write(staffNo = x, salary) end </pre>	<pre> end </pre>
(a)	(b)

Hình 4.1 Ví dụ về giao dịch.

Một giao dịch phải luôn biến đổi cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác, và chúng ta chấp nhận rằng tính nhất quán có thể bị vi phạm trong khi giao dịch đang diễn ra. Ví dụ, trong giao dịch ở Hình 4.1(b), có thể tại thời điểm nào đó một bộ trong PropertyForRent chứa giá trị newStaffNo và các bộ khác vẫn chưa giá trị cũ, x. Tuy nhiên, khi kết thúc giao dịch, tất cả các bộ cần thiết đều phải chứa giá trị newStaffNo.

Một giao dịch có thể có một trong hai kết quả: nếu hoàn tất thành công, giao dịch được cho là đã cam kết (committed) và cơ sở dữ liệu đạt trạng thái nhất quán mới; mặt khác, nếu giao dịch không thực hiện thành công, sẽ bị hủy bỏ (aborted). Nếu một giao dịch bị hủy bỏ, cơ sở dữ liệu phải được khôi phục về trạng thái nhất quán trước khi giao dịch bắt đầu. Một giao dịch như vậy được gọi là không hoàn thành (undone) hoặc hoàn tác (rolled back). Không thể hủy bỏ một giao dịch đã cam kết. Nếu chúng ta cho rằng giao dịch đã cam kết là một sai lầm, chúng ta phải thực hiện một giao dịch bù (compensating transaction) để đảo ngược tác động của nó.



**Hình 4.2** Sơ đồ trạng thái của một giao dịch.

DBMS không có cách cố định nào để biết cập nhật nào được nhóm lại với nhau để tạo thành một giao dịch logic duy nhất. Do đó, nó phải cung cấp một phương pháp cho phép người dùng chỉ ra ranh giới của một giao dịch. Các từ khóa BEGIN TRANSACTION, COMMIT, và ROLLBACK có sẵn trong ngôn ngữ thao tác dữ liệu để phân định các giao dịch. Nếu các từ khóa này không được sử dụng, toàn bộ chương trình thường được xem là một giao dịch duy nhất, và DBMS sẽ tự động thực hiện COMMIT khi chương trình kết thúc đúng cách hoặc ROLLBACK nếu không.

Hình 4.2 cho thấy sơ đồ chuyển đổi trạng thái cho một giao dịch. Lưu ý rằng ngoài các trạng thái rõ ràng là ACTIVE, COMMITTED và ABORTED, có hai trạng thái khác:

- PARTIALLY COMMITTED (Cam kết một phần), xảy ra sau khi câu lệnh cuối cùng đã được thực hiện. Tại thời điểm này, do có thể giao dịch đã vi phạm khả năng tuân tự (xem mục 4.3.2) hoặc đã vi phạm một ràng buộc toàn vẹn nên giao dịch phải bị hủy bỏ. Ngoài ra, hệ thống có thể bị lỗi và bất kỳ dữ liệu nào được cập nhật bởi giao dịch có thể không được ghi lại một cách an toàn trên bộ nhớ thứ cấp. Trong những trường hợp như vậy, giao dịch sẽ chuyển sang trạng thái FAILED và sẽ phải bị hủy bỏ. Nếu giao dịch thành công, sẽ chuyển sang trạng thái COMMITTED.

- FAILED (Thất bại), xảy ra nếu giao dịch không thể được cam kết hoặc giao dịch bị hủy bỏ khi đang ở trạng thái ACTIVE, có thể do người dùng hủy giao dịch hoặc do giao thức điều khiển đồng thời hủy bỏ giao dịch để đảm bảo khả năng tuần tự.

#### 4.1.1 Các thuộc tính của giao dịch

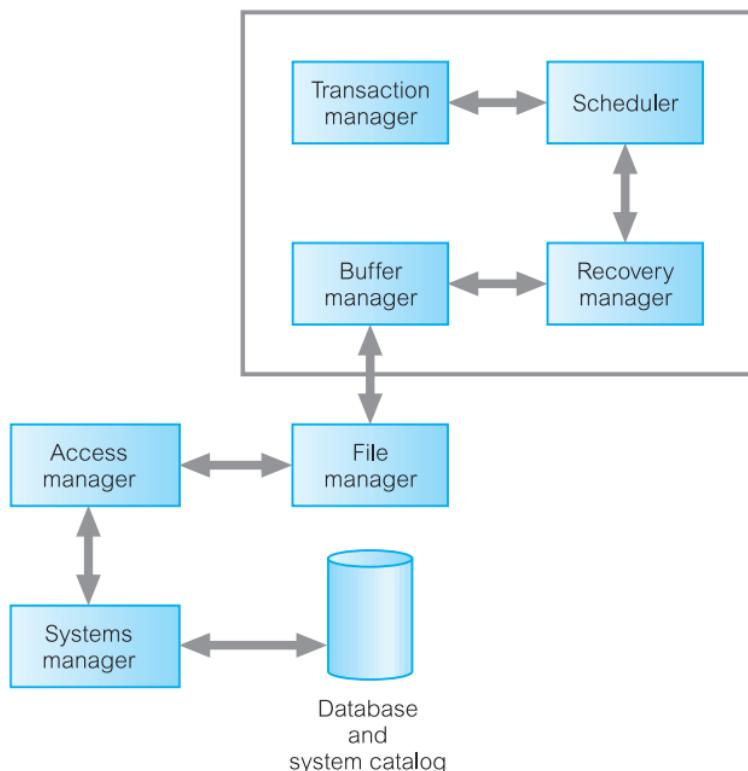
Bốn thuộc tính cơ bản, hay còn gọi là ACID, xác định một giao dịch là:

- Atomicity (Tính nguyên tử). Thuộc tính diễn tả khái niệm "tất cả hoặc không có gì". Giao dịch là một đơn vị không thể phân chia được, hoặc được thực hiện toàn bộ hoặc hoàn toàn không được thực hiện. Hệ thống con phục hồi của DBMS có trách nhiệm đảm bảo tính nguyên tử.
- Consistency (Tính nhất quán). Một giao dịch phải chuyển đổi cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác. Trách nhiệm của cả DBMS và các nhà phát triển ứng dụng là đảm bảo tính nhất quán. DBMS có thể đảm bảo tính nhất quán bằng cách thực thi tất cả các ràng buộc đã được chỉ định trên lược đồ cơ sở dữ liệu, chẳng hạn như các ràng buộc toàn vẹn. Tuy nhiên, bản thân điều này là không đủ để đảm bảo tính nhất quán. Ví dụ, giả sử rằng chúng ta có một giao dịch nhằm chuyển tiền từ tài khoản ngân hàng này sang tài khoản ngân hàng khác và lập trình viên mắc lỗi logic giao dịch, ghi nợ một tài khoản nhưng ghi có sai tài khoản; thì cơ sở dữ liệu sẽ ở trạng thái không nhất quán. Tuy nhiên, DBMS sẽ không chịu trách nhiệm sự không nhất quán này và sẽ không có khả năng phát hiện ra lỗi.
- Isolation (Tính cô lập). Các giao dịch thực hiện độc lập với nhau. Nói cách khác, các tác động một phần của các giao dịch chưa hoàn thành sẽ không thể được nhìn thấy đối với các giao dịch khác. Trách nhiệm của hệ thống con điều khiển đồng thời là đảm bảo sự cô lập.
- Durability (Tính bền vững). Các ảnh hưởng của một giao dịch đã hoàn thành thành công (đã cam kết) được ghi lại vĩnh viễn trong cơ sở dữ liệu và không bị mất vì một lần thất bại tiếp theo. Hệ thống con phục hồi có trách nhiệm đảm bảo độ bền.

#### 4.1.2 Kiến trúc cơ sở dữ liệu

Trong Chương 1, chúng ta đã tìm hiểu một kiến trúc cho một DBMS. Hình 4.3 biểu diễn cho một phần trích xuất từ Hình 1.7 xác định bốn mô-đun cơ sở dữ liệu cấp cao xử lý các giao dịch, điều khiển đồng thời và phục hồi. Trình quản lý giao dịch (transaction manager) điều phối các giao dịch thay mặt cho các chương trình ứng dụng, giao tiếp với trình lập lịch (scheduler), mô-đun chịu trách nhiệm thực hiện một chiến lược cụ thể để điều khiển đồng thời. Trình lập lịch đôi khi được gọi là trình quản lý khóa (lock manager) nếu giao thức điều khiển đồng thời dựa trên khóa. Mục tiêu của trình lập lịch là tối đa hóa sự đồng thời mà không cho phép các giao dịch thực hiện đồng thời ảnh hưởng lẫn nhau và do đó ảnh hưởng đến tính toàn vẹn hoặc tính nhất quán của cơ sở dữ liệu.

Nếu một lỗi xảy ra trong quá trình giao dịch, thì cơ sở dữ liệu có thể không nhất quán. Nhiệm vụ của trình quản lý phục hồi (recovery manager) là đảm bảo rằng cơ sở dữ liệu được khôi phục về trạng thái trước khi bắt đầu giao dịch và đó là trạng thái nhất quán. Cuối cùng, trình quản lý bộ đệm (buffer manager) chịu trách nhiệm chuyển dữ liệu hiệu quả giữa đĩa lưu trữ và bộ nhớ chính.



**Hình 4.3** Hệ thống xử lý giao dịch của DBMS.

## 4.2 PHỤC HỒI GIAO DỊCH

Các giao dịch biểu diễn cho đơn vị phục hồi cơ bản trong một hệ thống cơ sở dữ liệu. Vai trò của trình quản lý khôi phục (recovery manager) là đảm bảo hai trong bốn đặc tính ACID của các giao dịch, đó là tính nguyên tử và tính bền vững, khi có sự cố. Trình quản lý khôi phục phải đảm bảo rằng, khi khôi phục sau lỗi, hoặc tất cả các tác động của một giao dịch nhất định được ghi lại vĩnh viễn trong cơ sở dữ liệu hoặc không có tác động nào trong số chúng. Vấn đề phức tạp bởi thực tế là việc ghi cơ sở dữ liệu không phải là một hành động nguyên tử (một bước), và do đó có thể một giao dịch đã được cam kết nhưng các tác động của nó không được ghi lại vĩnh viễn trong cơ sở dữ liệu đơn giản vì chúng chưa đến được cơ sở dữ liệu.

Hãy xem xét lại ví dụ đầu tiên của chương này, trong đó mức lương của một nhân viên đang được tăng lên, như được thể hiện ở mức cao trong Hình 4.1(a). Để hiện thực thao tác đọc, DBMS thực hiện các bước sau:

- Tìm địa chỉ của khôi đĩa chứa bản ghi có khóa chính giá trị x;
- Chuyển khôi đĩa vào bộ đệm cơ sở dữ liệu trong bộ nhớ chính;
- Sao chép dữ liệu lương từ bộ đệm cơ sở dữ liệu vào biến salary.

Đối với hoạt động ghi, DBMS thực hiện các bước sau:

- Tìm địa chỉ của khối đĩa chứa bản ghi có khóa chính giá trị x;
- Chuyển khối đĩa vào bộ đệm cơ sở dữ liệu trong bộ nhớ chính;
- Sao chép dữ liệu lương từ biến salary vào bộ đệm cơ sở dữ liệu;
- Ghi bộ đệm cơ sở dữ liệu trở lại đĩa.

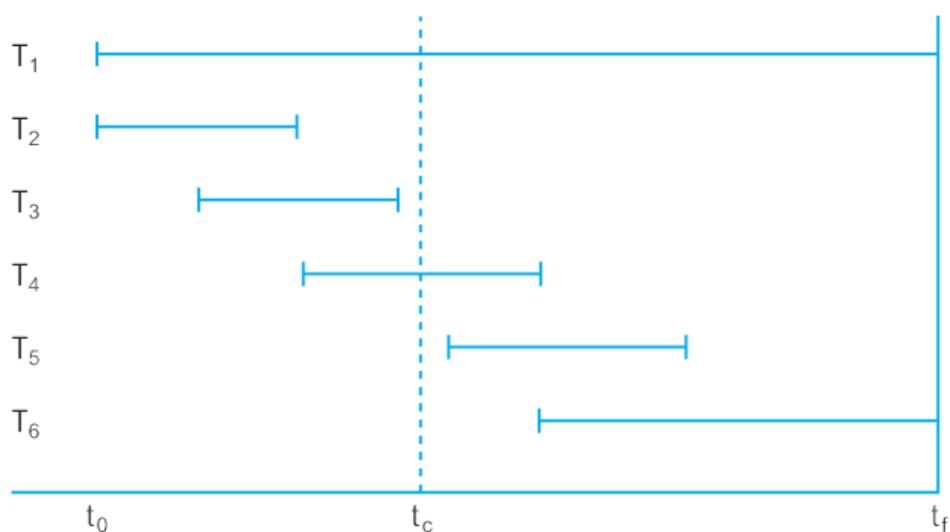
Bộ đệm cơ sở dữ liệu chiếm một vùng trong bộ nhớ chính mà từ đó dữ liệu được truyền đến và đi từ bộ nhớ thứ cấp. Chỉ khi bộ đệm đã được đầy (flush) sang bộ nhớ thứ cấp thì hoạt động cập nhật mới có thể được xem là vĩnh viễn. Việc đầy bộ đệm vào cơ sở dữ liệu có thể được kích hoạt bởi một lệnh cụ thể (ví dụ như cam kết giao dịch) hoặc tự động khi bộ đệm đầy. Việc ghi tường minh bộ đệm vào bộ nhớ thứ cấp được gọi là ghi bắt buộc (force-writing).

Nếu lỗi xảy ra trong quá trình ghi vào bộ đệm và chuyển bộ đệm vào bộ nhớ thứ cấp, trình quản lý khôi phục phải xác định được trạng thái của giao dịch đã thực hiện việc ghi tại thời điểm không thành công. Nếu giao dịch đã cam kết, thì để đảm bảo tính bền vững, trình quản lý khôi phục sẽ phải thực hiện lại (redo) các bản cập nhật của giao dịch đó đối với cơ sở dữ liệu (còn được gọi là rollforward). Mặt khác, nếu giao dịch không được cam kết tại thời điểm thất bại, thì trình quản lý khôi phục sẽ phải hoàn tác (undo) (còn được gọi là rollback) bất kỳ tác động nào của giao dịch đó trên cơ sở dữ liệu để đảm bảo tính nguyên tử của giao dịch. Nếu chỉ có một giao dịch phải được hoàn tác, điều này được gọi là hoàn tác một phần (partial undo). Trình lập lịch có thể kích hoạt hoàn tác một phần khi một giao dịch được khôi phục và khởi động lại do giao thức điều khiển đồng thời, như được mô tả trong phần trước. Một giao dịch cũng có thể bị hủy bỏ đơn phương bởi người dùng hoặc bởi một ngoại lệ phát sinh trong chương trình ứng dụng. Khi tất cả các giao dịch đang hoạt động phải được hoàn tác, điều này được gọi là hoàn tác hoàn toàn (global undo).

#### **Ví dụ 4.1 Sử dụng UNDO/REDO**

Hình 4.4 minh họa một số giao dịch thực thi đồng thời  $T_1, \dots, T_6$ . DBMS bắt đầu tại thời điểm  $t_0$  nhưng không lỗi tại thời điểm  $t_f$ . Chúng ta giả định rằng dữ liệu cho các giao dịch  $T_2$  và  $T_3$  đã được ghi vào bộ nhớ thứ cấp trước khi xảy ra lỗi.

Rõ ràng  $T_1$  và  $T_6$  đã không cam kết tại điểm xảy ra sự cố, do đó khi khởi động lại, trình quản lý khôi phục phải hoàn tác (undo) các giao dịch  $T_1$  và  $T_6$ . Tuy nhiên, có sự không rõ ràng về các thay đổi được thực hiện bởi các giao dịch (đã cam kết) khác,  $T_4$  và  $T_5$ , đã được truyền tới cơ sở dữ liệu trên phương tiện lưu trữ bất biến. Lý do cho sự không chắc chắn này là ở chỗ các bộ đệm cơ sở dữ liệu trên bộ nhớ chính có thể đã được ghi hoặc chưa được ghi vào đĩa. Trong trường hợp không có bất kỳ thông tin nào khác, trình quản lý khôi phục sẽ buộc phải thực hiện lại (redo) các giao dịch  $T_2$ ,  $T_3$ ,  $T_4$  và  $T_5$ .

**Hình 4.4** Ví dụ UNDO/REDO.

#### 4.2.1 Quản lý bộ đệm

Việc quản lý bộ đệm cơ sở dữ liệu đóng một vai trò quan trọng trong quá trình khôi phục và chúng ta thảo luận ngắn gọn về việc quản lý chúng trước khi tiếp tục. Như chúng ta đã đề cập ở đầu chương này, trình quản lý bộ đệm chịu trách nhiệm quản lý hiệu quả các bộ đệm cơ sở dữ liệu được sử dụng để chuyển các trang đến và đi từ bộ nhớ thứ cấp. Điều này liên quan đến việc đọc các trang từ đĩa vào bộ đệm cho đến khi bộ đệm đầy và sau đó sử dụng chiến lược thay thế (replacement strategy) để quyết định (các) bộ đệm nào cần ghi vào đĩa để tạo không gian cho các trang mới cần được đọc từ đĩa. Các chiến lược thay thế ví dụ là FIFO (first-in-first-out) và LRU (least recently used).

Một cách tiếp cận là liên kết hai biến với thông tin quản lý cho mỗi bộ đệm cơ sở dữ liệu: pinCount và dirty, ban đầu được đặt bằng 0 cho mỗi bộ đệm cơ sở dữ liệu. Khi một trang được yêu cầu từ đĩa, trình quản lý bộ đệm sẽ kiểm tra xem trang đó đã nằm trong một trong các bộ đệm cơ sở dữ liệu chưa. Nếu không, trình quản lý bộ đệm sẽ:

1. Sử dụng chiến lược thay thế để chọn một bộ đệm để thay thế (mà chúng ta sẽ gọi là bộ đệm thay thế - replacement buffer) và tăng pinCount của nó. Trang được yêu cầu hiện đã được ghim (pinCount) trong bộ đệm cơ sở dữ liệu và chưa thể ghi lại vào đĩa. Chiến lược thay thế sẽ không chọn một bộ đệm đã được ghim;
2. Nếu biến dirty cho bộ đệm thay thế được đặt, nó sẽ ghi bộ đệm vào đĩa;
3. Đọc trang từ đĩa vào bộ đệm thay thế và đặt lại biến dirty của bộ đệm thành 0.

Nếu cùng một trang được yêu cầu lại, pinCount thích hợp sẽ tăng lên 1. Khi hệ thống thông báo cho trình quản lý bộ đệm rằng nó đã hoàn thành trang, pinCount thích hợp sẽ giảm đi 1. Tại thời điểm này, hệ thống cũng sẽ thông báo cho bộ đệm quản lý nếu nó đã sửa đổi trang và biến dirty được đặt tương ứng. Khi pinCount bằng 0, trang sẽ được bỏ ghim (unpinned) và trang có thể được ghi trở lại vào đĩa nếu nó đã được sửa đổi (nghĩa là nếu biến dirty đã được đặt).

Các thuật ngữ được sử dụng trong khôi phục cơ sở dữ liệu khi các trang được ghi trở lại

đĩa:

- Chính sách steal cho phép trình quản lý bộ đệm ghi bộ đệm vào đĩa trước khi một giao dịch cam kết (bộ đệm được bơ ghim). Nói cách khác, trình quản lý bộ đệm “đánh cắp” một trang từ giao dịch. Chính sách thay thế là no-steal với ý nghĩa ngược lại.
- Chính sách force đảm bảo rằng tất cả các trang được cập nhật bởi một giao dịch sẽ được ghi ngay lập tức vào đĩa khi giao dịch cam kết. Chính sách thay thế là no-force với ý nghĩa ngược lại.

Cách tiếp cận đơn giản nhất từ góc độ triển khai là sử dụng chính sách no-steal, no-force: với no-steal, chúng ta không phải hoàn tác các thay đổi của một giao dịch đã hủy bỏ bởi vì các thay đổi sẽ không được ghi vào đĩa và với no-force, chúng ta không phải thực hiện lại các thay đổi của giao dịch đã cam kết nếu có sự cố tiếp theo vì tất cả các thay đổi sẽ được ghi vào đĩa khi cam kết.

Mặt khác, chính sách steal tránh sự cần thiết phải có một không gian đệm rất lớn để lưu trữ tất cả các trang được cập nhật bởi một tập hợp các giao dịch đồng thời, điều này trong thực tế có thể không khả thi. Hơn nữa, chính sách no-force có lợi thế khác biệt là không phải ghi lại một trang vào đĩa cho một giao dịch sau đó đã được cập nhật bởi một giao dịch đã cam kết trước đó và vẫn có thể nằm trong bộ đệm cơ sở dữ liệu.

#### **4.2.2 Các kỹ thuật phục hồi**

Quy trình khôi phục cụ thể sẽ được sử dụng phụ thuộc vào mức độ thiệt hại đã xảy ra đối với cơ sở dữ liệu. Chúng ta xem xét hai trường hợp:

1. Nếu cơ sở dữ liệu đã bị hỏng nặng - ví dụ, nếu sự cố đầu đĩa đã xảy ra và phá hủy cơ sở dữ liệu - thì cần phải khôi phục bản sao lưu cuối cùng của cơ sở dữ liệu trên một đĩa thay thế và thực hiện lại các thao tác cập nhật của các giao dịch đã cam kết bằng cách sử dụng tập tin nhật ký.
2. Nếu cơ sở dữ liệu không bị hỏng về mặt vật lý nhưng trở nên không nhất quán - ví dụ, nếu hệ thống gặp sự cố trong khi các giao dịch đang thực thi - thì cần phải hoàn tác các thay đổi gây ra sự không nhất quán. Cũng có thể cần thực hiện lại một số giao dịch để đảm bảo rằng các cập nhật mà chúng thực hiện đã đạt đến bộ nhớ thứ cấp. Ở đây, chúng ta không cần sử dụng bản sao lưu nhưng có thể khôi phục cơ sở dữ liệu về trạng thái nhất quán bằng cách sử dụng hình ảnh trước và sau được giữ trong tập tin nhật ký.

Bây giờ chúng ta xem xét hai kỹ thuật để phục hồi từ tình huống sau, đó là trường hợp cơ sở dữ liệu không bị phá hủy nhưng ở trạng thái không nhất quán. Các kỹ thuật, được gọi là cập nhật trì hoãn (deferred update) và cập nhật tức thời (immediate update), khác nhau ở cách các cập nhật được ghi vào bộ nhớ thứ cấp.

### **Kỹ thuật khôi phục bằng cách sử dụng cập nhật trì hoãn**

Sử dụng giao thức khôi phục cập nhật trì hoãn (deferred update), các cập nhật không được ghi vào cơ sở dữ liệu cho đến khi giao dịch đạt đến điểm cam kết. Nếu một giao dịch không thành công trước khi nó đạt đến thời điểm này, nó sẽ không sửa đổi cơ sở dữ liệu và do đó, không cần hoàn tác các thay đổi. Tuy nhiên, có thể cần phải thực hiện lại các cập nhật của các giao dịch đã cam kết vì hiệu lực của chúng có thể chưa đến được cơ sở dữ liệu. Trong trường hợp này, chúng ta sử dụng tập tin nhật ký để bảo vệ khỏi các lỗi hệ thống theo những cách sau:

- Khi một giao dịch bắt đầu, ghi bản ghi bắt đầu giao dịch vào nhật ký.
- Khi bắt kỳ thao tác ghi nào được thực hiện, ghi một bản ghi nhật ký chứa tất cả dữ liệu nhật ký được chỉ định trước đó (không bao gồm hình ảnh trước của cập nhật). Không thực sự ghi cập nhật vào bộ đệm cơ sở dữ liệu hoặc chính cơ sở dữ liệu.
- Khi một giao dịch sắp cam kết, ghi một bản ghi nhật ký cam kết giao dịch, ghi tất cả các bản ghi nhật ký cho giao dịch đó vào đĩa, sau đó cam kết giao dịch. Sử dụng các bản ghi nhật ký để thực hiện các cập nhật thực tế cho cơ sở dữ liệu.
- Nếu một giao dịch bị hủy bỏ, bỏ qua các bản ghi nhật ký cho giao dịch và không thực hiện hoạt động ghi.

Lưu ý rằng chúng ta ghi các bản ghi nhật ký vào đĩa trước khi giao dịch thực sự được cam kết, vì vậy nếu sự cố hệ thống xảy ra trong khi cập nhật cơ sở dữ liệu thực tế đang diễn ra, các bản ghi nhật ký sẽ tồn tại và các bản cập nhật có thể được áp dụng sau này. Trong trường hợp không thành công, chúng ta kiểm tra nhật ký để xác định các giao dịch đang được thực hiện tại thời điểm bị lỗi. Bắt đầu từ mục cuối cùng trong tập tin nhật ký, chúng ta quay lại bản ghi điểm kiểm tra gần đây nhất:

- Bất kỳ giao dịch nào có các bản ghi nhật ký bắt đầu giao dịch và cam kết giao dịch phải được thực hiện lại. Quy trình thực hiện lại thực hiện tất cả các lần ghi vào cơ sở dữ liệu bằng cách sử dụng các bản ghi nhật ký hình ảnh sau đó với các giao dịch, theo thứ tự chúng được ghi vào nhật ký. Nếu việc ghi này đã được thực hiện, trước khi xảy ra lỗi, việc ghi không có ảnh hưởng gì đến mục dữ liệu, vì vậy sẽ không có thiệt hại nào xảy ra nếu chúng ta ghi dữ liệu một lần nữa. Tuy nhiên, phương pháp này đảm bảo rằng chúng ta sẽ cập nhật bất kỳ mục dữ liệu nào không được cập nhật đúng cách trước khi xảy ra lỗi.
- Đối với bất kỳ giao dịch nào có bản ghi nhật ký bắt đầu giao dịch và hủy bỏ giao dịch, chúng ta không làm gì cả, vì không có việc ghi thực tế nào được thực hiện đối với cơ sở dữ liệu, vì vậy các giao dịch này không cần phải hoàn tác.

Nếu sự cố hệ thống thứ hai xảy ra trong quá trình khôi phục, các bản ghi nhật ký sẽ được sử dụng lại để khôi phục cơ sở dữ liệu. Với các thao tác ghi được lưu trong nhật ký, không quan trọng chúng ta thực hiện lại thao tác ghi bao nhiêu lần.

### Kỹ thuật khôi phục bằng cách sử dụng cập nhật tức thời

Sử dụng giao thức khôi phục cập nhật tức thời (immediate update), các cập nhật được áp dụng cho cơ sở dữ liệu khi chúng xảy ra mà không cần đợi đến điểm cam kết. Cũng như việc phải thực hiện lại các cập nhật của các giao dịch đã cam kết sau khi bị lỗi, bây giờ có thể cần phải hoàn tác các ảnh hưởng của các giao dịch không được cam kết tại thời điểm thất bại. Trong trường hợp này, chúng ta sử dụng tập tin nhật ký để bảo vệ khỏi các lỗi hệ thống theo cách sau:

- Khi một giao dịch bắt đầu, ghi bản ghi bắt đầu giao dịch vào nhật ký.
- Khi thao tác ghi được thực hiện, ghi một bản ghi chứa dữ liệu cần thiết vào tập tin nhật ký.
- Khi bản ghi nhật ký được ghi, ghi cập nhật vào bộ đệm cơ sở dữ liệu.
- Các cập nhật cho chính cơ sở dữ liệu đã được ghi khi bộ đệm được ghi vào bộ nhớ thứ cấp.
- Khi giao dịch được cam kết, một bản ghi cam kết giao dịch được ghi vào nhật ký.

Điều cần thiết là các bản ghi nhật ký (hoặc ít nhất là một số phần của chúng) được ghi trước khi ghi tương ứng vào cơ sở dữ liệu. Đây được gọi là giao thức nhật ký được ghi trước (write-ahead log protocol). Nếu các cập nhật được thực hiện cho cơ sở dữ liệu trước và lỗi xảy ra trước khi bản ghi nhật ký được ghi, thì trình quản lý khôi phục sẽ không có cách nào để hoàn tác (hoặc thực hiện lại) hoạt động. Theo giao thức nhật ký được ghi trước, trình quản lý khôi phục có thể giả định một cách an toàn rằng nếu không có bản ghi cam kết giao dịch nào trong tập tin nhật ký cho một giao dịch cụ thể, thì giao dịch đó vẫn hoạt động tại thời điểm thất bại và do đó phải được hoàn tác.

Nếu một giao dịch hủy bỏ, nhật ký có thể được sử dụng để hoàn tác, vì nó chứa tất cả các giá trị cũ cho các trường đã được cập nhật. Vì một giao dịch có thể đã thực hiện một số thay đổi đối với một mục, việc ghi sẽ được hoàn tác theo thứ tự ngược lại. Bất kể việc ghi của giao dịch đã được áp dụng cho chính cơ sở dữ liệu hay chưa, việc ghi các hình ảnh trước đảm bảo rằng cơ sở dữ liệu được khôi phục về trạng thái của nó trước khi bắt đầu giao dịch. Nếu hệ thống bị lỗi, việc khôi phục bao gồm việc sử dụng nhật ký để hoàn tác hoặc thực hiện lại các giao dịch:

- Đối với bất kỳ giao dịch nào mà cả bản ghi bắt đầu giao dịch và bản ghi cam kết giao dịch đều xuất hiện trong nhật ký, chúng ta thực hiện lại bằng cách sử dụng các bản ghi nhật ký để ghi hình ảnh sau của các trường đã được cập nhật, như đã mô tả trước đây.

Lưu ý rằng nếu các giá trị mới đã được ghi vào cơ sở dữ liệu, những hoạt động ghi này sẽ không có tác dụng. Tuy nhiên, bất kỳ hoạt động ghi nào không thực sự đến được cơ sở dữ liệu bây giờ sẽ được thực hiện.

- Đối với bất kỳ giao dịch nào mà nhật ký chứa bản ghi bắt đầu giao dịch nhưng không chứa bản ghi cam kết giao dịch, chúng ta cần hoàn tác giao dịch đó. Lúc này, các bản ghi nhật ký được sử dụng để ghi hình ảnh trước của các trường bị ảnh hưởng và do đó khôi phục cơ sở dữ liệu về trạng thái của nó trước khi bắt đầu giao dịch. Các hoạt động hoàn tác được thực hiện theo thứ tự ngược lại mà chúng đã được ghi vào nhật ký.

## 4.3 ĐIỀU KHIỂN ĐỒNG THỜI

Trong phần này, chúng ta xem xét các vấn đề có thể phát sinh với truy cập đồng thời và các kỹ thuật có thể được sử dụng để tránh những vấn đề này. Chúng ta bắt đầu với định nghĩa của điều khiển đồng thời.

**Điều khiển đồng thời** Quá trình quản lý các hoạt động đồng thời trên cơ sở dữ liệu để chúng không xen vào nhau.

### 4.3.1 Nhu cầu điều khiển đồng thời

Mục tiêu chính trong việc phát triển cơ sở dữ liệu là cho phép nhiều người dùng truy cập đồng thời vào dữ liệu được chia sẻ. Truy cập đồng thời tương đối dễ dàng nếu tất cả người dùng chỉ đọc dữ liệu, vì không có cách nào để họ có thể xen vào nhau. Tuy nhiên, khi hai hoặc nhiều người dùng truy cập cơ sở dữ liệu đồng thời và ít nhất một người đang cập nhật dữ liệu, có thể có sự xen vào nhau dẫn đến sự không nhất quán.

Mục tiêu này tương tự như mục tiêu của hệ thống máy tính đa người dùng, cho phép hai hoặc nhiều chương trình (hoặc giao dịch) thực thi cùng một lúc. Ví dụ, nhiều hệ thống có hệ thống con đầu vào/đầu ra (I/O) có thể xử lý các hoạt động I/O một cách độc lập, trong khi khói xử lý trung tâm chính (CPU) thực hiện các hoạt động khác. Các hệ thống như vậy có thể cho phép hai hoặc nhiều giao dịch thực thi đồng thời. Hệ thống bắt đầu thực hiện giao dịch đầu tiên cho đến khi nó đạt đến hoạt động I/O. Trong khi I/O đang được thực hiện, CPU sẽ tạm dừng giao dịch đầu tiên và thực hiện các lệnh từ giao dịch thứ hai. Khi giao dịch thứ hai đạt đến hoạt động I/O, quyền kiểm soát sau đó quay trở lại giao dịch đầu tiên và hoạt động của nó được tiếp tục từ thời điểm nó bị tạm dừng.

Giao dịch đầu tiên tiếp tục cho đến khi nó lại tiếp cận với một hoạt động I/O khác. Bằng cách này, các hoạt động của hai giao dịch được xen kẽ (interleaved) để thực hiện đồng thời. Ngoài ra, thông lượng (throughput) - khối lượng công việc được hoàn thành trong một khoảng thời gian nhất định - được cải thiện khi CPU đang thực hiện các giao dịch khác thay vì ở trạng thái nhàn rỗi chờ các hoạt động I/O hoàn tất.

Tuy nhiên, mặc dù bản thân hai giao dịch có thể hoàn toàn chính xác, nhưng việc xen kẽ các hoạt động theo cách này có thể tạo ra kết quả không chính xác, do đó ảnh hưởng đến tính toàn vẹn và nhất quán của cơ sở dữ liệu. Chúng ta xem xét ba ví dụ về các vấn đề tiềm ẩn do đồng thời gây ra: vấn đề cập nhật bị mất (lost update problem), vấn đề phụ thuộc không được cam kết (uncommitted dependency problem) và vấn đề phân tích không nhất quán (inconsistent analysis problem).

### Ví dụ 4.2 Vấn đề cập nhật bị mất

Một hoạt động cập nhật dường như đã hoàn thành thành công bởi một người dùng có thể bị ghi đè bởi một người dùng khác. Đây được gọi là vấn đề cập nhật bị mất (lost update problem) và được minh họa trong Hình 4.5, trong đó giao dịch T<sub>1</sub> đang thực hiện đồng thời với giao dịch T<sub>2</sub>. T<sub>1</sub> đang rút \$10 từ một tài khoản có số dư bal<sub>x</sub>, ban đầu là \$100 và T<sub>2</sub> đang gửi \$100 vào cùng một tài khoản. Nếu các giao dịch này được thực hiện tuần tự, nối tiếp nhau mà không có sự xen kẽ của các hoạt động, số dư cuối cùng sẽ là \$190 bất kể giao dịch nào được thực hiện trước.

Giao dịch T<sub>1</sub> và T<sub>2</sub> bắt đầu gần như cùng lúc và cả hai đều đọc số dư là \$100. T<sub>2</sub> tăng bal<sub>x</sub> từ \$100 lên \$200 và lưu trữ bản cập nhật trong cơ sở dữ liệu. Trong khi đó, giao dịch T<sub>1</sub> giảm bản sao của bal<sub>x</sub> từ \$100 xuống \$90 và lưu trữ giá trị này trong cơ sở dữ liệu, ghi đè lên bản cập nhật trước đó, và do đó làm “mất” \$100 đã được thêm vào số dư trước đó.

Việc mất cập nhật của T<sub>2</sub> có thể tránh được bằng cách ngăn không cho T<sub>1</sub> đọc giá trị của bal<sub>x</sub> cho đến sau khi cập nhật của T<sub>2</sub> đã hoàn thành.

Time	T <sub>1</sub>	T <sub>2</sub>	bal <sub>x</sub>
t <sub>1</sub>		begin_transaction	100
t <sub>2</sub>	begin_transaction	read(bal <sub>x</sub> )	100
t <sub>3</sub>	read(bal <sub>x</sub> )	bal <sub>x</sub> = bal <sub>x</sub> + 10	100
t <sub>4</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10	write(bal <sub>x</sub> )	200
t <sub>5</sub>	write(bal <sub>x</sub> )	commit	90
t <sub>6</sub>	commit		90

Hình 4.5 Vấn đề cập nhật bị mất.

### Ví dụ 4.3 Vấn đề phụ thuộc không được cam kết (hoặc dirty read)

Vấn đề phụ thuộc không được cam kết xảy ra khi một giao dịch được phép xem kết quả trung gian của một giao dịch khác trước khi giao dịch đó cam kết. Hình 4.6 cho thấy một ví dụ về một phụ thuộc không được cam kết gây ra lỗi, sử dụng cùng một giá trị ban đầu cho số dư bal<sub>x</sub> như trong ví dụ trước. Ở đây, giao dịch T<sub>4</sub> cập nhật bal<sub>x</sub> thành \$200, nhưng nó hủy giao dịch để bal<sub>x</sub> sẽ được khôi phục về giá trị ban đầu là \$100. Tuy nhiên, tại thời điểm này, giao dịch T<sub>3</sub> đã đọc giá trị mới của bal<sub>x</sub> (\$200) và đang sử dụng giá trị này làm cơ sở cho việc giảm \$10, tạo ra số dư không chính xác mới là \$190, thay vì \$90. Giá trị của bal<sub>x</sub> được đọc bởi T<sub>3</sub> được gọi là dữ liệu không chính xác (dirty data), do đó vấn đề này còn được gọi là dirty read problem.

Time	T <sub>3</sub>	T <sub>4</sub>	bal <sub>x</sub>
t <sub>1</sub>		begin_transaction	100
t <sub>2</sub>		read(bal <sub>x</sub> )	100
t <sub>3</sub>	begin_transaction	bal <sub>x</sub> = bal <sub>x</sub> + 100	100
t <sub>4</sub>	read(bal <sub>x</sub> )	write(bal <sub>x</sub> )	200
t <sub>5</sub>	.....	.....	200
t <sub>6</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10	rollback	100
t <sub>7</sub>	write(bal <sub>x</sub> )		190
t <sub>8</sub>	commit		190

Hình 4.6 Vấn đề phụ thuộc không được cam kết.

Lý do cho việc hoàn tác là không quan trọng; có thể là do giao dịch bị lỗi, có thể ghi có

## Giáo trình Hệ quản trị Cơ sở dữ liệu

vào tài khoản sai. Quan trọng là T<sub>3</sub> giả định rằng bản cập nhật của T<sub>4</sub> đã hoàn tất thành công, mặc dù bản cập nhật sau đó đã được khôi phục lại. Vấn đề này có thể tránh được bằng cách ngăn không cho T<sub>3</sub> đọc bal<sub>x</sub> cho đến khi T<sub>4</sub> đã được cam kết hoặc hủy bỏ.

Hai vấn đề trong các ví dụ này tập trung vào các giao dịch đang cập nhật cơ sở dữ liệu và sự can thiệp của chúng có thể làm hỏng cơ sở dữ liệu. Tuy nhiên, các giao dịch chỉ đọc cơ sở dữ liệu cũng có thể tạo ra kết quả không chính xác nếu chúng được phép đọc một phần kết quả của các giao dịch không hoàn chỉnh đang đồng thời cập nhật cơ sở dữ liệu. Chúng ta sẽ minh họa điều này bằng ví dụ tiếp theo.

### Ví dụ 4.4 Vấn đề phân tích không nhất quán

Vấn đề phân tích không nhất quán xảy ra khi một giao dịch đọc một số giá trị từ cơ sở dữ liệu nhưng giao dịch thứ hai cập nhật một số trong số chúng trong quá trình thực hiện giao dịch đầu tiên. Ví dụ, một giao dịch đang tổng hợp dữ liệu trong cơ sở dữ liệu (ví dụ như tổng các số dư) sẽ nhận được kết quả không chính xác nếu trong khi nó đang thực hiện, các giao dịch khác đang cập nhật cơ sở dữ liệu.

Một ví dụ được minh họa trong Hình 4.7, trong đó giao dịch tính tổng T<sub>6</sub> đang thực hiện đồng thời với giao dịch T<sub>5</sub>. Giao dịch T<sub>6</sub> đang tính tổng số dư của tài khoản x (\$100), tài khoản y (\$50) và tài khoản z (\$25). Tuy nhiên, trong khi chờ đợi, giao dịch T<sub>5</sub> đã chuyển \$10 từ bal<sub>x</sub> sang bal<sub>z</sub>, do đó T<sub>6</sub> bây giờ có kết quả sai (cao hơn \$10). Vấn đề này được tránh bằng cách ngăn giao dịch T<sub>6</sub> đọc bal<sub>x</sub> và bal<sub>z</sub> cho đến khi T<sub>5</sub> hoàn thành cập nhật.

Time	T <sub>5</sub>	T <sub>6</sub>	bal <sub>x</sub>	bal <sub>y</sub>	bal <sub>z</sub>	sum
t <sub>1</sub>		begin_transaction	100	50	25	
t <sub>2</sub>	begin_transaction	sum = 0	100	50	25	0
t <sub>3</sub>	read(bal <sub>x</sub> )	read(bal <sub>x</sub> )	100	50	25	0
t <sub>4</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10	sum = sum + bal <sub>x</sub>	100	50	25	100
t <sub>5</sub>	write(bal <sub>x</sub> )	read(bal <sub>y</sub> )	90	50	25	100
t <sub>6</sub>	read(bal <sub>z</sub> )	sum = sum + bal <sub>y</sub>	90	50	25	150
t <sub>7</sub>	bal <sub>z</sub> = bal <sub>z</sub> + 10		90	50	25	150
t <sub>8</sub>	write(bal <sub>z</sub> )		90	50	35	150
t <sub>9</sub>	commit	read(bal <sub>z</sub> )	90	50	35	150
t <sub>10</sub>		sum = sum + bal <sub>z</sub>	90	50	35	185
t <sub>11</sub>		commit	90	50	35	185

Hình 4.7 Vấn đề phân tích không nhất quán.

Một vấn đề khác có thể xảy ra khi một giao dịch T đọc lại một mục dữ liệu mà nó đã đọc trước đó nhưng ở giữa, một giao dịch khác đã sửa đổi nó. Như vậy, T nhận được hai giá trị khác nhau cho cùng một mục dữ liệu. Điều này đôi khi được gọi là đọc không lặp lại - nonrepeatable (hoặc fuzzy) read. Một vấn đề tương tự có thể xảy ra nếu giao dịch T thực hiện một truy vấn lấy một tập các bộ từ một quan hệ thỏa mãn một vị trí nhất định, thực hiện lại truy vấn sau đó, nhưng phát hiện rằng tập các bộ được truy xuất có chứa một bộ bổ sung (phantom) đã được chèn bởi một giao dịch khác trong thời gian chờ đợi. Điều này đôi khi được gọi là đọc ảo (phantom read).

### 4.3.2 Khả năng tuần tự và khả năng khôi phục

Mục tiêu của giao thức điều khiển đồng thời là lập lịch trình các giao dịch theo cách để tránh bất kỳ sự xen vào nhau của chúng và do đó ngăn chặn các vấn đề được mô tả trong phần trước. Giải pháp đơn giản nhất là chỉ cho phép thực hiện một giao dịch tại một thời điểm: một giao dịch được cam kết (committed) trước khi giao dịch tiếp theo được phép bắt đầu (begin). Tuy nhiên, mục đích của DBMS đa người dùng là tối đa hóa mức độ đồng thời hoặc song song trong hệ thống, để các giao dịch có thể thực hiện song song mà không xen vào nhau. Trong phần này, chúng ta tìm hiểu khả năng tuần tự như một phương tiện giúp xác định cách thực thi của các giao dịch sao cho đảm bảo được tính nhất quán của cơ sở dữ liệu.

**Lịch trình** Một chuỗi các hoạt động bởi một tập hợp các giao dịch đồng thời duy trì thứ tự của các hoạt động trong mỗi giao dịch riêng lẻ.

Một giao dịch là một chuỗi các hoạt động bao gồm các hành động đọc và/hoặc ghi vào cơ sở dữ liệu, sau đó là một hành động cam kết (commit) hoặc hủy bỏ (abort). Một lịch trình S bao gồm một chuỗi các hoạt động từ tập n giao dịch  $T_1, T_2, \dots, T_n$  tuân theo ràng buộc là thứ tự hoạt động cho mỗi giao dịch sẽ được giữ nguyên trong lịch trình. Do đó, đối với mỗi giao dịch  $T_i$  trong lịch trình (schedule) S, thứ tự của các hoạt động trong  $T_i$  phải giống trong lịch trình S.

**Lịch trình tuần tự** Một lịch trình trong đó các hoạt động của mỗi giao dịch được thực hiện liên tục mà không có bất kỳ hoạt động xen kẽ nào từ các giao dịch khác.

Trong một lịch trình tuần tự (serial schedule), các giao dịch được thực hiện theo thứ tự nối tiếp nhau. Ví dụ, nếu chúng ta có hai giao dịch  $T_1$  và  $T_2$ , thứ tự sẽ là  $T_1$ , tiếp theo là  $T_2$  hoặc  $T_2$  sau  $T_1$ . Do đó, trong quá trình thực thi này không có sự xen vào nhau của các giao dịch, bởi vì chỉ có một giao dịch được thực hiện tại một thời điểm bất kỳ. Tuy nhiên, không có gì đảm bảo rằng kết quả của tất cả các lần thực thi tuần tự của một tập các giao dịch nhất định sẽ hoàn toàn giống nhau. Ví dụ, trong ngân hàng, điều quan trọng là liệu tiền lãi được tính trên tài khoản trước khi hay sau khi một khoản tiền lớn được chuyển vào tài khoản đó.

**Lịch trình phi tuần tự** Một lịch trình trong đó các hoạt động từ một tập các giao dịch đồng thời xen kẽ vào nhau.

Các vấn đề được mô tả trong Ví dụ 4.2, 4.3 và 4.4 là do việc quản lý đồng thời không tốt, khiến cơ sở dữ liệu ở trạng thái không nhất quán trong hai ví dụ đầu tiên và hiển thị kết quả sai cho người dùng ở ví dụ thứ ba. Thực thi tuần tự sẽ ngăn chặn việc xảy ra các vấn đề như thế. Bất kể lịch trình tuần tự nào được chọn, việc thực thi tuần tự không bao giờ khiến cơ sở dữ liệu rơi vào trạng thái không nhất quán. Mục tiêu của khả năng tuần tự (serializability) là tìm ra các lịch trình phi tuần tự (nonserial schedule) cho phép các giao dịch thực hiện đồng thời mà không xen vào nhau, tạo ra trạng thái cơ sở dữ liệu giống như được tạo ra bằng thực thi tuần tự.

Nếu một tập các giao dịch được thực hiện đồng thời, chúng ta nói rằng lịch trình (phi tuần tự) là đúng nếu nó tạo ra kết quả giống như một thực thi tuần tự nào đó. Lịch trình như vậy được gọi là khả tuần tự (Serializable). Để ngăn chặn sự mâu thuẫn do các giao dịch xen vào nhau, điều cần thiết là phải đảm bảo khả năng tuần tự của các giao dịch đồng thời. Khi xem xét về khả năng tuần tự, thứ tự của các hoạt động đọc và ghi được cân nhắc như sau:

- Nếu hai giao dịch chỉ đọc một mục dữ liệu, chúng không xung đột và thứ tự không quan trọng.
- Nếu hai giao dịch đọc hoặc ghi các mục dữ liệu hoàn toàn riêng biệt, chúng không xung đột và thứ tự không quan trọng.
- Nếu một giao dịch ghi một mục dữ liệu và một giao dịch khác đọc hoặc ghi cùng một mục dữ liệu, thứ tự thực hiện là rất quan trọng.

Hãy xem xét lịch trình  $S_1$  được trình bày trong Hình 4.8(a) chứa các hoạt động từ hai giao dịch thực hiện đồng thời  $T_7$  và  $T_8$ . Vì thao tác ghi trên  $bal_x$  trong  $T_8$  không xung đột với thao tác đọc tiếp theo trên  $bal_y$  trong  $T_7$ , chúng ta có thể thay đổi thứ tự của các thao tác này để tạo ra lịch trình tương đương  $S_2$  được thể hiện trong Hình 4.8(b). Nếu bây giờ chúng ta cũng thay đổi thứ tự của các hoạt động không xung đột sau đây, chúng ta sẽ tạo ra lịch trình tuần tự tương đương  $S_3$  được hiển thị trong Hình 4.8(c):

- Thay đổi thứ tự  $write(bal_x)$  của  $T_8$  với  $write(bal_y)$  của  $T_7$ .
- Thay đổi thứ tự  $read(bal_x)$  của  $T_8$  với  $read(bal_y)$  của  $T_7$ .
- Thay đổi thứ tự  $read(bal_x)$  của  $T_8$  với  $write(bal_y)$  của  $T_7$ .

Lịch trình  $S_3$  là một lịch trình tuần tự và, vì  $S_1$  và  $S_2$  tương đương với  $S_3$  nên  $S_1$  và  $S_2$  là các lịch trình khả tuần tự (Serializable schedule).

Time	$T_7$	$T_8$
$t_1$	begin_transaction	
$t_2$	read( $bal_x$ )	
$t_3$		write( $bal_x$ )
$t_4$		begin_transaction
$t_5$		read( $bal_x$ )
$t_6$		write( $bal_x$ )
$t_7$	read( $bal_y$ )	
$t_8$		write( $bal_y$ )
$t_9$	commit	
$t_{10}$		read( $bal_y$ )
$t_{11}$		write( $bal_y$ )
$t_{12}$		commit

(a) Lịch trình phi tuần tự  $S_1$

Time	T <sub>7</sub>	T <sub>8</sub>
t <sub>1</sub>	begin_transaction	
t <sub>2</sub>	read( <b>bal<sub>x</sub></b> )	
t <sub>3</sub>	write( <b>bal<sub>x</sub></b> )	
t <sub>4</sub>		begin_transaction
t <sub>5</sub>		read( <b>bal<sub>x</sub></b> )
t <sub>6</sub>	read( <b>bal<sub>y</sub></b> )	
t <sub>7</sub>		write( <b>bal<sub>x</sub></b> )
t <sub>8</sub>	write( <b>bal<sub>y</sub></b> )	
t <sub>9</sub>	commit	
t <sub>10</sub>		read( <b>bal<sub>y</sub></b> )
t <sub>11</sub>		write( <b>bal<sub>y</sub></b> )
t <sub>12</sub>		commit

(b) Lịch trình phi tuần tự S<sub>2</sub> tương đương với S<sub>1</sub>

Time	T <sub>7</sub>	T <sub>8</sub>
t <sub>1</sub>	begin_transaction	
t <sub>2</sub>	read( <b>bal<sub>x</sub></b> )	
t <sub>3</sub>	write( <b>bal<sub>x</sub></b> )	
t <sub>4</sub>	read( <b>bal<sub>y</sub></b> )	
t <sub>5</sub>	write( <b>bal<sub>y</sub></b> )	
t <sub>6</sub>	commit	
t <sub>7</sub>		begin_transaction
t <sub>8</sub>		read( <b>bal<sub>x</sub></b> )
t <sub>9</sub>		write( <b>bal<sub>x</sub></b> )
t <sub>10</sub>		read( <b>bal<sub>y</sub></b> )
t <sub>11</sub>		write( <b>bal<sub>y</sub></b> )
t <sub>12</sub>		commit

(c) Lịch trình tuần tự S<sub>3</sub>, tương đương với S<sub>1</sub> và S<sub>2</sub>

#### Hình 4.8 Các lịch trình tương đương.

Loại khả năng tuần tự này được gọi là khả tuần tự xung đột. Lịch trình khả tuần tự xung đột (conflict serializable schedule) sắp xếp thứ tự bất kỳ hoạt động xung đột nào theo cách giống như một thực thi tuần tự nào đó.

#### Kiểm tra tính khả tuần tự xung đột

Theo quy tắc ghi có ràng buộc (constrained write rule - nghĩa là một giao dịch cập nhật một mục dữ liệu dựa trên giá trị cũ của nó, giá trị này đã được giao dịch đọc trước đó), một đồ thị ưu tiên (precedence graph) có thể được tạo để kiểm tra tính khả tuần tự xung đột. Đối với lịch trình S, đồ thị ưu tiên là đồ thị có hướng G = (N, E) bao gồm một tập các nút N và một tập các cạnh có hướng E, được xây dựng như sau:

- Tạo một nút cho mỗi giao dịch.
- Tạo một cạnh có hướng  $T_i \rightarrow T_j$ , nếu  $T_j$  đọc giá trị của một mục được ghi bởi  $T_i$ .
- Tạo một cạnh có hướng  $T_i \rightarrow T_j$ , nếu  $T_j$  ghi một giá trị vào một mục sau khi mục đó đã được đọc bởi  $T_i$ .
- Tạo một cạnh có hướng  $T_i \rightarrow T_j$ , nếu  $T_j$  ghi một giá trị vào một mục sau khi mục đó đã được ghi bởi  $T_i$ .

Nếu một cạnh  $T_i \rightarrow T_j$  tồn tại trong đồ thị ưu tiên của S, thì trong bất kỳ lịch trình tuần

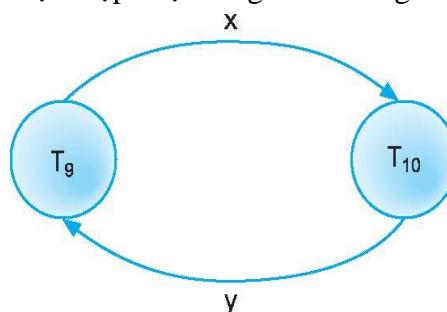
tự S' nào tương đương với S, T<sub>i</sub> phải xuất hiện trước T<sub>j</sub>. Nếu đồ thị ưu tiên chứa một chu trình, thì lịch trình này không khả tuần tự xung đột.

#### Ví dụ 4.5 Lịch trình không khả tuần tự xung đột

Hãy xem xét hai giao dịch được thể hiện trong Hình 4.9. Giao dịch T<sub>9</sub> đang chuyển \$100 từ một tài khoản có số dư bal<sub>x</sub> sang tài khoản khác có số dư bal<sub>y</sub>, trong khi T<sub>10</sub> đang tăng số dư của hai tài khoản này lên 10%. Đồ thị ưu tiên cho lịch trình này, được hiển thị trong Hình 4.10, có một chu trình và do đó không khả tuần tự xung đột.

Time	T <sub>9</sub>	T <sub>10</sub>
t <sub>1</sub>	begin_transaction	
t <sub>2</sub>	read(bal <sub>x</sub> )	
t <sub>3</sub>	bal <sub>x</sub> = bal <sub>x</sub> + 100	
t <sub>4</sub>	write(bal <sub>x</sub> )	begin_transaction
t <sub>5</sub>		read(bal <sub>x</sub> )
t <sub>6</sub>		bal <sub>x</sub> = bal <sub>x</sub> * 1.1
t <sub>7</sub>		write(bal <sub>x</sub> )
t <sub>8</sub>		read(bal <sub>y</sub> )
t <sub>9</sub>		bal <sub>y</sub> = bal <sub>y</sub> * 1.1
t <sub>10</sub>		write(bal <sub>y</sub> )
t <sub>11</sub>	read(bal <sub>y</sub> )	commit
t <sub>12</sub>	bal <sub>y</sub> = bal <sub>y</sub> - 100	
t <sub>13</sub>	write(bal <sub>y</sub> )	
t <sub>14</sub>	commit	

**Hình 4.9** Hai giao dịch cập nhật đồng thời không khả tuần tự xung đột.



**Hình 4.10** Đồ thị ưu tiên cho Hình 4.9 hiển thị một chu trình.

#### Tuần tự khung nhìn

Có một số khả năng tuần tự khác cung cấp các định nghĩa ít nghiêm ngặt hơn về tương đương lịch trình so với định nghĩa được cung cấp bởi khả tuần tự xung đột. Một định nghĩa ít hạn chế hơn được gọi là khả năng tuần tự khung nhìn (view serializability). Hai lịch trình S<sub>1</sub> và S<sub>2</sub> bao gồm các hoạt động giống nhau từ n giao dịch T<sub>1</sub>, T<sub>2</sub>, ..., T<sub>n</sub> được xem là tương đương khung nhìn (view equivalent) nếu thỏa ba điều kiện sau:

- Đối với mỗi mục dữ liệu x, nếu giao dịch T<sub>i</sub> đọc giá trị ban đầu của x trong lịch trình S<sub>1</sub>, thì giao dịch T<sub>i</sub> cũng phải đọc giá trị ban đầu của x trong lịch trình S<sub>2</sub>.
- Đối với mỗi thao tác đọc trên mục dữ liệu x bởi giao dịch T<sub>i</sub> trong lịch trình S<sub>1</sub>, nếu giá trị được đọc bởi T<sub>i</sub> được ghi bởi giao dịch T<sub>j</sub>, thì giao dịch T<sub>i</sub> cũng phải đọc giá trị của x được tạo ra bởi giao dịch T<sub>j</sub> trong lịch trình S<sub>2</sub>.
- Đối với mỗi mục dữ liệu x, nếu thao tác ghi cuối cùng trên x được thực hiện bởi giao dịch T<sub>i</sub> trong lịch trình S<sub>1</sub>, thì giao dịch T<sub>i</sub> cũng phải thực hiện thao tác ghi

cuối cùng trên mục dữ liệu x trong lịch trình S<sub>2</sub>.

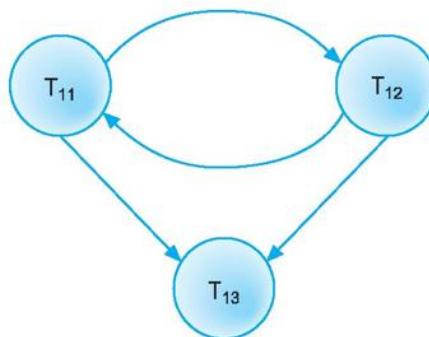
Một lịch trình là khả tuần tự khung nhìn (view serializable) nếu nó tương đương khung nhìn với một lịch trình tuần tự. Mọi lịch trình khả tuần tự xung đột là khả tuần tự khung nhìn, mặc dù điều ngược lại là không đúng. Ví dụ, lịch trình hiển thị trong Hình 4.11 là khả tuần tự khung nhìn nhưng không khả tuần tự xung đột. Trong ví dụ này, các giao dịch T<sub>12</sub> và T<sub>13</sub> không tuân theo quy tắc ghi có ràng buộc; nói cách khác, chúng thực hiện ghi mù (blind write). Điều này cho thấy bất kỳ lịch trình khả tuần tự khung nhìn nào mà không khả tuần tự xung đột đều chứa một hoặc nhiều hành động ghi mù.

Time	T <sub>11</sub>	T <sub>12</sub>	T <sub>13</sub>
t <sub>1</sub>	begin_transaction		
t <sub>2</sub>	read(bal <sub>x</sub> )		
t <sub>3</sub>		begin_transaction	
t <sub>4</sub>		write(bal <sub>x</sub> )	
t <sub>5</sub>		commit	
t <sub>6</sub>	write(bal <sub>x</sub> )		
t <sub>7</sub>	commit		
t <sub>8</sub>			begin_transaction
t <sub>9</sub>			write(bal <sub>x</sub> )
t <sub>10</sub>			commit

Hình 4.11 Lịch trình khả tuần tự khung nhìn nhưng không khả tuần tự xung đột.

### Kiểm tra tính khả tuần tự khung nhìn

Kiểm tra tính khả tuần tự khung nhìn phức tạp hơn nhiều so với kiểm tra tính khả tuần tự xung đột. Trên thực tế đã chứng minh thuật toán kiểm tra khả tuần tự khung nhìn có độ phức tạp là NP-complete; do đó, rất khó có thể tìm ra một thuật toán hiệu quả. Hình 4.12 là đồ thị ưu tiên (khả tuần tự xung đột) tương ứng với lịch trình đã cho trong Hình 4.11. Đồ thị này chứa một chu trình chỉ ra rằng lịch trình này không khả tuần tự xung đột; tuy nhiên, chúng ta biết rằng nó khả tuần tự khung nhìn, vì nó tương đương với lịch trình tuần tự T<sub>11</sub>, tiếp theo là T<sub>12</sub>, sau đó là T<sub>13</sub>. Khi kiểm tra các quy tắc cho đồ thị ưu tiên đã cho trước đó, chúng ta có thể thấy rằng cạnh T<sub>12</sub> → T<sub>11</sub> không nên được chèn vào đồ thị, vì các giá trị của bal<sub>x</sub> được ghi bởi T<sub>11</sub> và T<sub>12</sub> không bao giờ được sử dụng bởi bất kỳ giao dịch nào khác bởi vì các hành động ghi mù.



Hình 4.12 Đồ thị ưu tiên cho lịch trình khả tuần tự khung nhìn ở Hình 4.11.

Do đó, để kiểm tra tính khả tuần tự khung nhìn, chúng ta cần một phương pháp để quyết định xem có nên chèn một cạnh vào đồ thị ưu tiên hay không. Cách tiếp cận mà chúng ta thực hiện là xây dựng một đồ thị ưu tiên gắn nhãn (labeled precedence graph) cho lịch

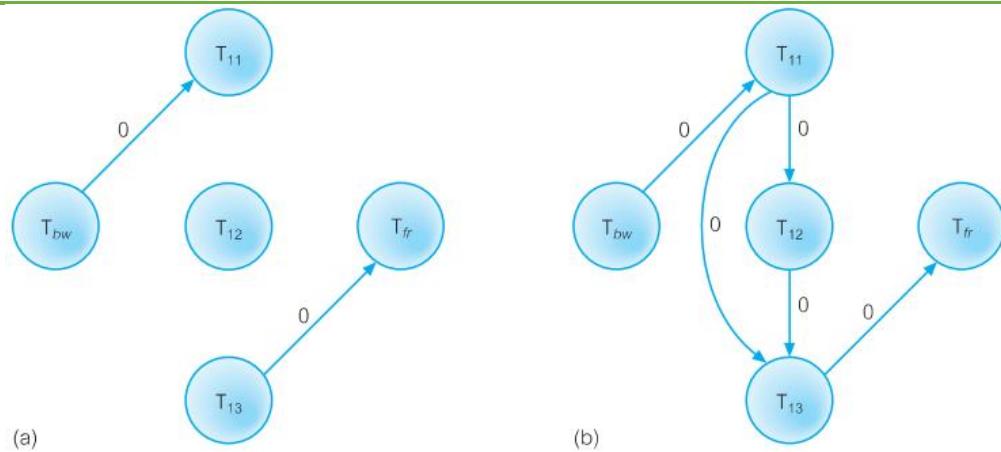
trình như sau:

1. Tạo một nút cho mỗi giao dịch.
2. Tạo một nút có nhãn  $T_{bw}$ .  $T_{bw}$  là một giao dịch giả được chèn vào đầu lịch trình chứa thao tác ghi cho mỗi mục dữ liệu được truy cập trong lịch trình.
3. Tạo một nút có nhãn  $T_{fr}$ .  $T_{fr}$  là một giao dịch giả được thêm vào cuối lịch trình chứa thao tác đọc cho từng mục dữ liệu được truy cập trong lịch trình.
4. Tạo một cạnh có hướng  $T_i \rightarrow^0 T_j$  nếu  $T_j$  đọc giá trị của một mục được ghi bởi  $T_i$ .
5. Loại bỏ tất cả các cạnh có hướng trên giao dịch  $T_i$  mà không có đường dẫn nào từ  $T_i$  đến  $T_{fr}$ .
6. Đối với mỗi mục dữ liệu mà  $T_j$  đọc (đã được ghi bởi  $T_i$ ) và  $T_k$  ghi ( $T_k \neq T_{bw}$ ), thì:
  - a. Nếu  $T_i = T_{bw}$  và  $T_j \neq T_{fr}$ , thì tạo một cạnh có hướng  $T_i \rightarrow^0 T_k$ .
  - b. Nếu  $T_i \neq T_{bw}$  và  $T_j = T_{fr}$ , thì tạo một cạnh có hướng  $T_k \rightarrow^0 T_i$ .
  - c. Nếu  $T_i \neq T_{bw}$  và  $T_j \neq T_{fr}$ , thì tạo một cặp cạnh có hướng  $T_k \rightarrow^x T_i$ , và  $T_j \rightarrow^x T_k$ , trong đó  $x$  là số nguyên dương duy nhất chưa được sử dụng cho việc gắn nhãn cho một cạnh có hướng trước đó. Quy tắc này là một trường hợp tổng quát hơn của hai quy tắc trước, chỉ ra rằng nếu giao dịch  $T_i$  ghi một mục mà  $T_j$  sau đó đọc, sau đó bất kỳ giao dịch nào khác,  $T_k$ , ghi cùng một mục phải được thực hiện trước  $T_i$  hoặc  $T_j$  đã thành công.

Áp dụng năm quy tắc đầu tiên vào lịch trình trong Hình 4.11 sẽ tạo ra đồ thị ưu tiên được hiển thị trong Hình 4.13(a). Áp dụng quy tắc 6(a), chúng ta thêm các cạnh  $T_{11} \rightarrow T_{12}$  và  $T_{11} \rightarrow T_{13}$ , cả hai đều có nhãn là 0; áp dụng quy tắc 6(b), chúng ta thêm các cạnh  $T_{11} \rightarrow T_{13}$  (đã có) và  $T_{12} \rightarrow T_{13}$ , một lần nữa cả hai đều có nhãn 0. Đồ thị cuối cùng được thể hiện trong Hình 4.13(b).

Dựa trên đồ thị ưu tiên gắn nhãn này, kiểm tra tính khả tuần tự khung nhìn như sau:

1. Nếu đồ thị không chứa chu trình, lịch trình là khả tuần tự khung nhìn.
2. Tuy nhiên, sự hiện diện của một chu trình không phải là điều kiện đủ để kết luận rằng lịch trình không khả tuần tự khung nhìn. Các thử nghiệm thực tế dựa trên quan sát cho thấy quy tắc 6(c) tạo ra m cặp cạnh có hướng phân biệt, dẫn đến  $2^m$  đồ thị khác nhau chỉ chứa một cạnh từ mỗi cặp. Nếu bất kỳ đồ thị nào trong số này không chứa chu trình, thì lịch trình tương ứng là khả tuần tự khung nhìn và thứ tự tuần tự hóa được xác định bằng sắp xếp tópô của đồ thị với các giao dịch giả  $T_{bw}$  và  $T_{fr}$  bị loại bỏ.



**Hình 4.13** Đồ thị ưu tiên gắn nhãn cho lịch trình khả tuần tự khung nhìn ở Hình 4.11.

Time	T <sub>11</sub>	T <sub>12</sub>	T <sub>13A</sub>
t <sub>1</sub>	begin_transaction		
t <sub>2</sub>		read( <b>bal</b> <sub>x</sub> )	
t <sub>3</sub>		begin_transaction	
t <sub>4</sub>			write( <b>bal</b> <sub>x</sub> )
t <sub>5</sub>		commit	
t <sub>6</sub>			begin_transaction
t <sub>7</sub>			read( <b>bal</b> <sub>x</sub> )
t <sub>8</sub>			write( <b>bal</b> <sub>x</sub> )
t <sub>9</sub>		commit	
t <sub>10</sub>			write( <b>bal</b> <sub>x</sub> )
t <sub>11</sub>			commit

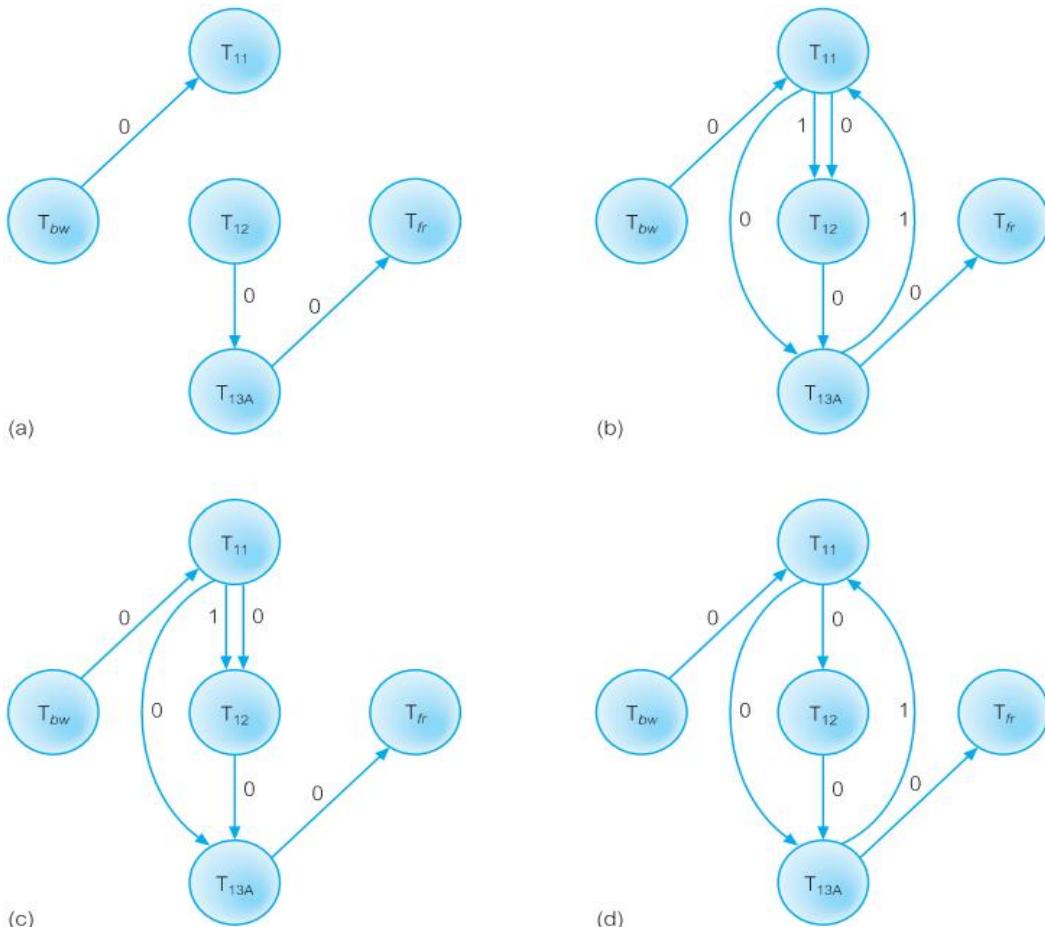
**Hình 4.14** Phiên bản sửa đổi của lịch trình trong Hình 4.11 có thêm thao tác đọc.

Áp dụng các thử nghiệm này vào đồ thị trong Hình 4.13(b), là đồ thị không chứa chu trình, chúng ta kết luận rằng lịch trình này là khả tuần tự khung nhìn. Một ví dụ khác, hãy xem xét biến thể được sửa đổi một chút của lịch trình trong Hình 4.11 được thể hiện trong Hình 4.14 có chứa thêm thao tác đọc trong giao dịch T<sub>13A</sub>. Áp dụng năm quy tắc đầu tiên cho lịch trình này sẽ tạo ra đồ thị ưu tiên được hiển thị trong Hình 4.15(a). Áp dụng quy tắc 6(a), ta thêm các cạnh T<sub>11</sub> → T<sub>12</sub> và T<sub>11</sub> → T<sub>13A</sub>, cả hai đều có nhãn là 0; áp dụng quy tắc 6(b), ta thêm các cạnh T<sub>11</sub> → T<sub>13A</sub> (đã có) và T<sub>12</sub> → T<sub>13A</sub> (lại đã có), cả hai đều có nhãn là 0. Áp dụng quy tắc 6(c), ta thêm cặp cạnh T<sub>11</sub> → T<sub>12</sub> và T<sub>13A</sub> → T<sub>11</sub>, lần này đều có nhãn 1. Đồ thị cuối cùng được thể hiện trong Hình 4.15(b). Từ đó, chúng ta có thể tạo ra hai đồ thị khác nhau chỉ chứa một cạnh, như trong Hình 4.15(c) và 9.14(d). Vì Hình 4.15(c) không chứa chu trình, chúng ta có thể kết luận rằng lịch trình này cũng khả tuần tự khung nhìn (tương ứng với lịch trình tuần tự T<sub>11</sub> → T<sub>12</sub> → T<sub>13A</sub>).

### Khả năng khôi phục

Khả năng tuần tự xác định các lịch trình duy trì tính nhất quán của cơ sở dữ liệu, giả sử rằng không có giao dịch nào trong lịch trình bị lỗi. Một quan điểm khác là kiểm tra khả năng khôi phục của các giao dịch trong một lịch trình. Nếu một giao dịch không thành công, thuộc tính nguyên tử yêu cầu chúng ta hoàn tất các tác động của giao dịch. Ngoài ra, thuộc tính bền vững nói rằng một khi giao dịch được cam kết, các thay đổi của nó không thể được hoàn tất (trừ khi thực thi một giao dịch bù đắp khác).

Hãy xem xét lại hai giao dịch được hiển thị trong Hình 4.9, nhưng thay vì hoạt động cam kết vào cuối giao dịch  $T_9$ , giả sử  $T_9$  quyết định hoàn tác các ảnh hưởng của giao dịch.  $T_{10}$  đã đọc bản cập nhật của  $\text{bal}_x$  do  $T_9$  thực hiện, và bản thân nó cũng đã cập nhật  $\text{bal}_x$  và cam kết thay đổi. Nói một cách chính xác, chúng ta nên hoàn tác giao dịch  $T_{10}$ , vì nó đã sử dụng một giá trị của  $\text{bal}_x$  mà giá trị này đã được hoàn tác. Tuy nhiên, thuộc tính bền vững không cho phép điều này. Nói cách khác, lịch trình này là lịch trình không thể khôi phục (nonrecoverable schedule), không được phép sử dụng. Điều này dẫn đến định nghĩa của lịch trình có thể phục hồi (recoverable schedule).



**Hình 4.15** Đồ thị ưu tiên gắn nhãn cho lịch trình khả tuân tự khung nhìn ở Hình 4.14.

### 4.3.3 Các kỹ thuật điều khiển đồng thời

Hai kỹ thuật điều khiển đồng thời chính cho phép các giao dịch thực hiện song song một cách an toàn với các ràng buộc nhất định được gọi là kỹ thuật khóa (locking) và kỹ thuật dấu thời gian (timestamping).

Kỹ thuật khóa và kỹ thuật dấu thời gian là những cách tiếp cận bi quan (pessimistic) trong đó các giao dịch bị trì hoãn trong trường hợp chúng xung đột với các giao dịch khác vào một thời điểm nào đó trong tương lai. Các cách tiếp cận lạc quan (optimistic), như sẽ thấy ở phần sau, dựa trên tiền đề rằng xung đột là rất hiếm, vì vậy các giao dịch được phép tiến hành không đồng bộ và sẽ kiểm tra xung đột chỉ khi kết thúc, khi giao dịch thực hiện cam kết.

### 4.3.3.1 Các phương pháp khóa

**Khóa** Một thủ tục được sử dụng để điều khiển truy cập đồng thời vào dữ liệu. Khi một giao dịch đang truy cập cơ sở dữ liệu, một khóa có thể từ chối quyền truy cập của các giao dịch khác để ngăn kết quả không chính xác. Phương pháp khóa là cách tiếp cận được sử dụng rộng rãi nhất để đảm bảo khả năng tuần tự của các giao dịch đồng thời. Có một số biến thể, nhưng tất cả đều có chung một đặc điểm cơ bản, đó là giao dịch phải yêu cầu khóa chia sẻ - shared (read) hoặc khóa độc quyền - exclusive (write) trên một mục dữ liệu trước khi thực hiện hành động đọc hoặc ghi cơ sở dữ liệu tương ứng. Khóa (lock) ngăn không cho một giao dịch khác sửa đổi hoặc thậm chí đọc mục dữ liệu đó, trong trường hợp khóa độc quyền.

**Khóa chia sẻ** Nếu một giao dịch trên một mục dữ liệu có khóa chia sẻ, thì nó có thể đọc mục đó nhưng không cập nhật được mục dữ liệu này. Các mục dữ liệu có kích thước khác nhau, từ toàn bộ cơ sở dữ liệu cho đến một trường, đều có thể bị khóa. Kích thước của mục xác định độ mịn (fineness) hoặc độ chi tiết (granularity) của khóa. Khóa thực sự có thể được hiện thực bằng cách thiết lập một bit trong mục dữ liệu để cho biết rằng phần đó của cơ sở dữ liệu đã bị khóa, hoặc bằng cách giữ danh sách các phần bị khóa của cơ sở dữ liệu, hoặc bằng các cách khác.

**Khóa độc quyền** Nếu một giao dịch trên một mục dữ liệu có một khóa độc quyền, thì nó có thể đọc và cập nhật mục đó. Vì các thao tác đọc không thể xung đột, nên cho phép nhiều hơn một giao dịch giữ các khóa chia sẻ đồng thời trên cùng một mục. Ngược lại, một khóa độc quyền cung cấp cho một giao dịch quyền truy cập độc quyền vào mục đó. Do đó, nếu một giao dịch giữ khóa độc quyền trên mục dữ liệu, thì không giao dịch nào khác có thể đọc hoặc cập nhật mục đó. Các khóa được sử dụng theo cách sau:

- Bất kỳ giao dịch nào cần truy cập vào một mục dữ liệu, trước tiên phải khóa mục đó bằng cách yêu cầu khóa chia sẻ cho quyền truy cập chỉ đọc (read-only) hoặc khóa độc quyền cho cả quyền truy cập đọc và ghi.
- Nếu mục dữ liệu chưa bị khóa bởi một giao dịch khác, khóa sẽ được cấp.
- Nếu mục hiện đang bị khóa, DBMS sẽ xác định xem yêu cầu có tương thích với khóa hiện có hay không. Nếu yêu cầu khóa chia sẻ trên một mục đã có khóa chia sẻ trên đó, yêu cầu sẽ được chấp nhận; nếu không, giao dịch này phải đợi cho đến khi khóa hiện tại được giải phóng.
- Một giao dịch sẽ giữ khóa cho đến khi giải phóng khóa một cách tường minh trong khi thực thi hoặc khi kết thúc (hủy bỏ hoặc cam kết). Chỉ khi khóa độc quyền đã được giải phóng thì các tác động của thao tác ghi mới được hiển thị cho các giao dịch khác.

Ngoài các quy tắc này, một số hệ thống cho phép giao dịch yêu cầu một khóa chia sẻ trên một mục dữ liệu và sau đó nâng cấp thành khóa độc quyền. Thực tế, điều này cho phép một giao dịch kiểm tra dữ liệu trước và sau đó quyết định xem liệu có muốn cập nhật dữ liệu đó hay không.

Nếu nâng cấp không được hỗ trợ, giao dịch phải giữ các khóa độc quyền trên tất cả các mục dữ liệu mà nó có thể cập nhật vào một thời điểm nào đó trong quá trình thực hiện giao dịch, do đó có khả năng làm giảm mức độ đồng thời trong hệ thống. Vì lý do tương tự, một số hệ thống cũng cho phép giao dịch yêu cầu một khóa độc quyền và sau đó hạ cấp (downgrade) khóa thành khóa chia sẻ.

Việc sử dụng khóa trong các giao dịch, như đã mô tả trước đây, không đảm bảo khả năng tuần tự của chính các lịch trình.

Để đảm bảo khả năng tuần tự, chúng ta phải tuân theo một giao thức bổ sung liên quan đến việc định vị các hoạt động khóa và mở khóa trong từng giao dịch. Giao thức phổ biến nhất là khóa hai pha (two-phase locking - 2PL).

### Two-phase locking (2PL)

<b>Khóa hai pha (2PL)</b>	Một giao dịch tuân theo giao thức khóa hai pha nếu tất cả các hoạt động khóa được thực hiện trước hoạt động mở khóa đầu tiên trong giao dịch.
-------------------------------	---

Theo các quy tắc của giao thức này, mọi giao dịch có thể được chia thành hai giai đoạn: đầu tiên là giai đoạn phát triển (growing phase), trong đó nó có được tất cả các khóa cần thiết nhưng không thể giải phóng bất kỳ khóa nào, và sau đó là giai đoạn thu hẹp (shrinking phase), trong đó nó giải phóng các khóa của mình nhưng không thể yêu cầu bất kỳ khóa mới nào. Không bắt buộc tất cả các khóa phải được yêu cầu cùng thời điểm. Thông thường, giao dịch yêu cầu một số khóa, thực hiện một số xử lý và tiếp tục yêu cầu các khóa bổ sung nếu cần thiết. Tuy nhiên, nó sẽ không bao giờ giải phóng bất kỳ khóa nào trước khi đạt đến trạng thái không cần khóa mới nữa. Các quy tắc là:

- Một giao dịch phải yêu cầu khóa trên một mục trước khi thao tác trên mục đó. Khóa có thể đọc (chia sẻ) hoặc ghi (độc quyền), tùy thuộc vào nhu cầu giao dịch.
- Một khi giao dịch bắt đầu giải phóng khóa, nó không bao giờ có thể yêu cầu bắt kỳ khóa mới nữa.

Nếu việc nâng cấp khóa được cho phép, việc nâng cấp chỉ có thể diễn ra trong giai đoạn phát triển và có thể yêu cầu giao dịch đó phải chờ cho đến khi giao dịch khác giải phóng khóa chia sẻ trên cùng mục dữ liệu đó. Việc hạ cấp chỉ có thể diễn ra trong giai đoạn thu hẹp. Chúng ta sẽ xem xét cách sử dụng khóa hai pha để giải quyết ba vấn đề được xác định trong mục 9.2.1.

### Ví dụ 4.6 Ngăn chặn sự cố cập nhật bị mất sử dụng 2PL

Giải pháp cho vấn đề cập nhật bị mất được hiển thị trong Hình 4.16. Để ngăn sự cố cập nhật bị mất xảy ra, trước tiên T<sub>2</sub> yêu cầu khóa độc quyền trên bal<sub>x</sub>. Sau đó, nó có thể tiếp tục đọc giá trị của bal<sub>x</sub> từ cơ sở dữ liệu, tăng mục dữ liệu này lên \$100 và ghi giá trị mới trả lại cơ sở dữ liệu. Khi T<sub>1</sub> bắt đầu, nó cũng yêu cầu một khóa độc quyền trên bal<sub>x</sub>. Tuy nhiên, vì mục dữ liệu bal<sub>x</sub> hiện đang bị T<sub>2</sub> khóa độc quyền, yêu cầu không được cấp ngay lập tức và T<sub>1</sub> phải đợi cho đến khi khóa được T<sub>2</sub> giải phóng. Điều này chỉ xảy ra khi cam kết của T<sub>2</sub> đã được hoàn thành.

Time	T <sub>1</sub>	T <sub>2</sub>	bal <sub>x</sub>
t <sub>1</sub>		begin_transaction	100
t <sub>2</sub>	begin_transaction	write_lock(bal <sub>x</sub> )	100
t <sub>3</sub>	write_lock(bal <sub>x</sub> )	read(bal <sub>x</sub> )	100
t <sub>4</sub>	WAIT	bal <sub>x</sub> = bal <sub>x</sub> + 100	100
t <sub>5</sub>	WAIT	write(bal <sub>x</sub> )	200
t <sub>6</sub>	WAIT	commit/unlock(bal <sub>x</sub> )	200
t <sub>7</sub>	read(bal <sub>x</sub> )		200
t <sub>8</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10		200
t <sub>9</sub>	write(bal <sub>x</sub> )		190
t <sub>10</sub>	commit/unlock(bal <sub>x</sub> )		190

**Hình 4.16** Ngăn chặn sự cố cập nhật bị mất.**Ví dụ 4.7** Ngăn chặn vấn đề phụ thuộc không được cam kết sử dụng 2PL

Một giải pháp cho vấn đề phụ thuộc không cam kết được thể hiện trong Hình 4.17. Để ngăn sự cố này xảy ra, trước tiên T<sub>4</sub> yêu cầu một khóa độc quyền trên bal<sub>x</sub>. Sau đó, nó có thể tiếp tục đọc giá trị của bal<sub>x</sub> từ cơ sở dữ liệu, tăng mục dữ liệu này lên \$100 và ghi giá trị mới trở lại cơ sở dữ liệu. Khi quá trình khôi phục (rollback) được thực hiện, các cập nhật của giao dịch T<sub>4</sub> sẽ được hoàn tác và giá trị của bal<sub>x</sub> trong cơ sở dữ liệu được trả về giá trị ban đầu là \$100. Khi T<sub>3</sub> bắt đầu, nó cũng yêu cầu một khóa độc quyền trên bal<sub>x</sub>. Tuy nhiên, vì mục dữ liệu bal<sub>x</sub> hiện đang bị khóa độc quyền bởi T<sub>4</sub>, yêu cầu không được cấp ngay lập tức và T<sub>3</sub> phải đợi cho đến khi khóa được T<sub>4</sub> giải phóng. Điều này chỉ xảy ra khi quá trình khôi phục T<sub>4</sub> đã hoàn tất.

Time	T <sub>3</sub>	T <sub>4</sub>	bal <sub>x</sub>
t <sub>1</sub>		begin_transaction	100
t <sub>2</sub>		write_lock(bal <sub>x</sub> )	100
t <sub>3</sub>		read(bal <sub>x</sub> )	100
t <sub>4</sub>	begin_transaction	bal <sub>x</sub> = bal <sub>x</sub> + 100	100
t <sub>5</sub>	write_lock(bal <sub>x</sub> )	write(bal <sub>x</sub> )	200
t <sub>6</sub>	WAIT	rollback/unlock(bal <sub>x</sub> )	100
t <sub>7</sub>	read(bal <sub>x</sub> )		100
t <sub>8</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10		100
t <sub>9</sub>	write(bal <sub>x</sub> )		90
t <sub>10</sub>	commit/unlock(bal <sub>x</sub> )		90

**Hình 4.17** Ngăn chặn vấn đề phụ thuộc không được cam kết.**Ví dụ 4.8** Ngăn chặn vấn đề phân tích không nhất quán sử dụng 2PL

Giải pháp cho vấn đề phân tích không nhất quán được thể hiện trong Hình 4.18. Để ngăn sự cố này xảy ra, T<sub>5</sub> phải đặt trước các lần đọc của nó bằng các khóa độc quyền và T<sub>6</sub> phải đặt trước các lần đọc của nó bằng các khóa chia sẻ. Do đó, khi T<sub>5</sub> bắt đầu, nó yêu cầu và nhận được một khóa độc quyền trên bal<sub>x</sub>. Lúc này, khi T<sub>6</sub> yêu cầu khóa chia sẻ trên bal<sub>x</sub>, yêu cầu không được cấp ngay lập tức và T<sub>6</sub> phải đợi cho đến khi khóa được giải phóng, đó là lúc T<sub>5</sub> cam kết (commit).

Time	T <sub>5</sub>	T <sub>6</sub>	bal <sub>x</sub>	bal <sub>y</sub>	bal <sub>z</sub>	sum
t <sub>1</sub>		begin_transaction	100	50	25	
t <sub>2</sub>	begin_transaction	sum = 0	100	50	25	0
t <sub>3</sub>	write_lock(bal <sub>x</sub> )		100	50	25	0
t <sub>4</sub>	read(bal <sub>x</sub> )	read_lock(bal <sub>x</sub> )	100	50	25	0
t <sub>5</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10	WAIT	100	50	25	0
t <sub>6</sub>	write(bal <sub>x</sub> )	WAIT	90	50	25	0
t <sub>7</sub>	write_lock(bal <sub>z</sub> )	WAIT	90	50	25	0
t <sub>8</sub>	read(bal <sub>z</sub> )	WAIT	90	50	25	0
t <sub>9</sub>	bal <sub>z</sub> = bal <sub>z</sub> + 10	WAIT	90	50	25	0
t <sub>10</sub>	write(bal <sub>z</sub> )	WAIT	90	50	35	0
t <sub>11</sub>	commit/unlock(bal <sub>x</sub> , bal <sub>z</sub> )	WAIT	90	50	35	0
t <sub>12</sub>		read(bal <sub>x</sub> )	90	50	35	0
t <sub>13</sub>		sum = sum + bal <sub>x</sub>	90	50	35	90
t <sub>14</sub>		read_lock(bal <sub>y</sub> )	90	50	35	90
t <sub>15</sub>		read(bal <sub>y</sub> )	90	50	35	90
t <sub>16</sub>		sum = sum + bal <sub>y</sub>	90	50	35	140
t <sub>17</sub>		read_lock(bal <sub>z</sub> )	90	50	35	140
t <sub>18</sub>		read(bal <sub>z</sub> )	90	50	35	140
t <sub>19</sub>		sum = sum + bal <sub>z</sub>	90	50	35	175
t <sub>20</sub>		commit/unlock(bal <sub>x</sub> , bal <sub>y</sub> , bal <sub>z</sub> )	90	50	35	175

**Hình 4.18** Ngăn chặn vấn đề phân tích không nhất quán.

## Deadlock

### Deadlock

Một deadlock (tắc nghẽn) có thể xảy ra khi hai (hoặc nhiều) giao dịch đang chờ lẫn nhau các khóa bên kia nắm giữ được giải phóng.

Hình 4.19 cho thấy hai giao dịch, T<sub>17</sub> và T<sub>18</sub>, bị tắc nghẽn vì mỗi giao dịch đang chờ giao dịch kia mở khóa trên một mục đang nắm giữ. Tại thời điểm t<sub>2</sub>, giao dịch T<sub>17</sub> yêu cầu và nhận được khóa độc quyền trên mục dữ liệu bal<sub>x</sub> và tại thời điểm t<sub>3</sub>, giao dịch T<sub>18</sub> nhận được khóa độc quyền trên mục dữ liệu baly. Sau đó, tại thời điểm t<sub>6</sub>, T<sub>17</sub> yêu cầu một khóa độc quyền đối với mục dữ liệu baly. Bởi vì T<sub>18</sub> giữ khóa trên baly, giao dịch T<sub>17</sub> sẽ chờ đợi. Trong khi đó, tại thời điểm t<sub>7</sub>, T<sub>18</sub> yêu cầu khóa trên mục bal<sub>x</sub>, được giữ bởi giao dịch T<sub>17</sub>. Cả hai giao dịch đều không thể tiếp tục, bởi vì mỗi giao dịch đang chờ một khóa mà nó không thể lấy được cho đến khi giao dịch kia hoàn thành. Khi deadlock xảy ra, các ứng dụng liên quan không thể tự giải quyết sự cố. Thay vào đó, DBMS phải nhận ra deadlock tồn tại và phá vỡ deadlock theo một cách nào đó.

Thật không may, chỉ có một cách duy nhất để phá vỡ deadlock: hủy bỏ một hoặc nhiều giao dịch. Điều này thường liên quan đến việc hoàn tất cả các thay đổi được thực hiện bởi (các) giao dịch bị hủy bỏ. Trong Hình 4.19, chúng ta có thể quyết định hủy bỏ giao dịch T<sub>18</sub>. Sau khi hoàn tất, các khóa do giao dịch T<sub>18</sub> nắm giữ sẽ được giải phóng và T<sub>17</sub> có thể tiếp tục hoạt động. Deadlock phải trong suốt đối với người dùng, vì vậy DBMS sẽ tự động khởi động lại (các) giao dịch bị hủy bỏ. Tuy nhiên, trên thực tế, DBMS không thể khởi động lại giao dịch bị hủy bỏ vì nó không biết về logic giao dịch, ngay cả khi đã biết về lịch sử giao dịch.

Time	T <sub>17</sub>	T <sub>18</sub>
t <sub>1</sub>	begin_transaction	
t <sub>2</sub>	write_lock(bal <sub>x</sub> )	begin_transaction
t <sub>3</sub>	read(bal <sub>x</sub> )	write_lock(bal <sub>y</sub> )
t <sub>4</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10	read(bal <sub>y</sub> )
t <sub>5</sub>	write(bal <sub>x</sub> )	bal <sub>y</sub> = bal <sub>y</sub> + 100
t <sub>6</sub>	write_lock(bal <sub>y</sub> )	write(bal <sub>y</sub> )
t <sub>7</sub>	WAIT	write_lock(bal <sub>x</sub> )
t <sub>8</sub>	WAIT	WAIT
t <sub>9</sub>	WAIT	WAIT
t <sub>10</sub>	WAIT	WAIT
t <sub>11</sub>	WAIT	WAIT
t <sub>12</sub>	.....	.....

**Hình 4.19** Deadlock giữa hai giao dịch.

Có ba kỹ thuật chung để xử lý deadlock: thời gian chờ (timeout), ngăn chặn deadlock (deadlock prevention), phát hiện deadlock và khôi phục (deadlock detection and recovery). Với thời gian chờ, giao dịch yêu cầu khóa sẽ chờ tối đa trong một khoảng thời gian nhất định. Với ngăn chặn deadlock, DBMS sẽ xem xét trước để xác định xem một giao dịch có gây ra deadlock hay không và không bao giờ cho phép xảy ra deadlock. Với phát hiện và khôi phục deadlock, DBMS cho phép deadlock xảy ra nhưng sẽ nhận ra sự xuất hiện deadlock và phá vỡ nó. Bởi vì việc ngăn chặn deadlock khó hơn là sử dụng thời gian chờ hoặc phát hiện deadlock và khôi phục nên các hệ thống thường tránh giải pháp ngăn chặn deadlock.

### Thời gian chờ

Một cách tiếp cận đơn giản để ngăn chặn deadlock là dựa trên thời gian chờ khóa. Trong cách tiếp cận này, một giao dịch yêu cầu khóa sẽ chỉ chờ trong một khoảng thời gian do hệ thống xác định. Nếu khóa chưa được cấp trong khoảng thời gian này, yêu cầu khóa sẽ hết hiệu lực. Trong trường hợp này, DBMS giả định rằng giao dịch có thể bị deadlock, mặc dù có thể không, và nó sẽ hủy bỏ và tự động khởi động lại giao dịch. Đây là một giải pháp rất đơn giản và thiết thực để ngăn chặn deadlock được nhiều DBMS thương mại sử dụng.

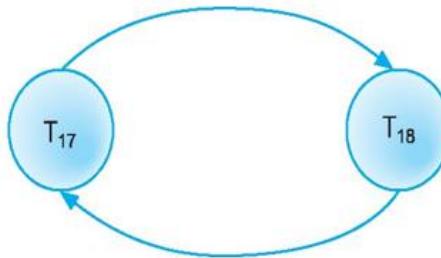
### Ngăn chặn deadlock

Một cách tiếp cận khả thi khác để ngăn chặn deadlock là sắp xếp giao dịch bằng cách sử dụng dấu thời gian giao dịch, chúng ta sẽ thảo luận trong mục 9.2.5. Hai thuật toán đã được đề xuất bởi Rosenkrantz (1978). Một thuật toán, Wait-Die, chỉ cho phép một giao dịch cũ hơn chờ một giao dịch mới hơn, ngược lại giao dịch sẽ bị hủy bỏ (die) và khởi động lại với cùng một dấu thời gian, để cuối cùng nó sẽ trở thành giao dịch hoạt động cũ nhất và sẽ không chết. Thuật toán thứ hai, Wound-Wait, sử dụng cách tiếp cận đối xứng: chỉ cho phép một giao dịch mới hơn chờ một giao dịch cũ hơn. Nếu một giao dịch cũ hơn yêu cầu khóa do một giao dịch mới hơn nắm giữ, giao dịch mới hơn sẽ bị hủy bỏ (wound).

### Phát hiện deadlock và khôi phục

Phát hiện deadlock thường được xử lý bằng cách xây dựng một đồ thị chờ (wait-for graph - WFG) cho thấy các phụ thuộc giao dịch; nghĩa là, giao dịch  $T_i$  phụ thuộc vào  $T_j$  nếu giao dịch  $T_j$  giữ khóa trên một mục dữ liệu mà  $T_i$  đang chờ. WFG là một đồ thị có hướng  $G = (N, E)$  bao gồm tập hợp các nút  $N$  và tập hợp các cạnh có hướng  $E$ , được xây dựng như sau:

- Tạo một nút cho mỗi giao dịch.
- Tạo một cạnh có hướng  $T_i \rightarrow T_j$ , nếu giao dịch  $T_i$  đang chờ để khóa một mục hiện đang bị khóa bởi  $T_j$ .



Dấu thời gian mới phải được gán cho giao dịch được khởi động lại để tránh việc chúng liên tục bị hủy bỏ và khởi động lại. Nếu không có dấu thời gian mới, một giao dịch với dấu thời gian cũ có thể không thể được cam kết do các giao dịch mới hơn đã được cam kết.

Ngoài dấu thời gian cho các giao dịch, còn có dấu thời gian cho các mục dữ liệu. Mỗi mục dữ liệu chứa một `read_timestamp`, dấu thời gian của giao dịch cuối cùng đọc mục và `write_timestamp`, dấu thời gian của giao dịch cuối cùng ghi (cập nhật) mục. Đối với giao dịch T có dấu thời gian  $ts(T)$ , giao thức thứ tự dấu thời gian (timestamp ordering protocol) hoạt động như sau:

1. Giao dịch T đưa ra một `read(x)`.

- Giao dịch T yêu cầu đọc một mục (x) đã được cập nhật bởi một giao dịch mới hơn, tức là,  $ts(T) < write_timestamp(x)$ . Điều này có nghĩa là một giao dịch trước đang cố gắng đọc giá trị của một mục đã được cập nhật bởi một giao dịch sau. Giao dịch trước sẽ đọc được giá trị đã lỗi thời trước đó, có khả năng không nhất quán với giá trị đã cập nhật của mục dữ liệu. Trong tình huống này, giao dịch T phải được hủy bỏ và bắt đầu lại với một dấu thời gian mới.
- Ngược lại, nếu  $ts(T) \geq write_timestamp(x)$  thì thao tác đọc có thể tiếp tục. Chúng ta đặt  $read_timestamp(x) = \max(ts(T), read_timestamp(x))$ .

2. Giao dịch T đưa ra một `write(x)`.

- Giao dịch T yêu cầu ghi một mục (x) có giá trị đã được đọc bởi một giao dịch mới hơn, đó là  $ts(T) < read_timestamp(x)$ . Điều này có nghĩa là giao dịch sau đã sử dụng giá trị hiện tại của mục và sẽ có lỗi nếu cập nhật mục ngay lúc này. Điều này xảy ra khi một giao dịch bị trễ trong quá trình ghi và một giao dịch mới hơn đã đọc giá trị cũ hoặc ghi giá trị mới. Trong trường hợp này, giải pháp duy nhất là roll back giao dịch T và khởi động lại nó bằng cách sử dụng dấu thời gian mới.
- Giao dịch T yêu cầu ghi một mục (x) có giá trị đã được ghi bởi một giao dịch mới hơn, đó là  $ts(T) < write_timestamp(x)$ . Điều này có nghĩa là giao dịch T đang cố gắng ghi một giá trị lỗi thời của mục dữ liệu x. Giao dịch T nên được khôi phục và khởi động lại bằng dấu thời gian mới.
- Ngược lại, thao tác ghi có thể tiến hành. Chúng ta đặt  $write_timestamp(x) = ts(T)$ .

### 4.3.3.3 Các kỹ thuật lạc quan

Trong một số môi trường, rất hiếm khi xảy ra xung đột giữa các giao dịch và việc xử lý bổ sung được yêu cầu bằng các giao thức khóa hoặc dấu thời gian là không cần thiết đối với nhiều giao dịch. Các kỹ thuật lạc quan (optimistic technique) dựa trên giả định rằng xung đột là hiếm và được cho là sẽ hiệu quả hơn nếu cho phép các giao dịch tiến hành mà không gây ra sự chậm trễ để đảm bảo khả năng tuần tự hóa. Khi một giao dịch muốn cam kết, một hoạt động kiểm tra được thực hiện để xác định xem xung đột có xảy ra hay không. Nếu có xung đột, giao dịch phải được khôi phục và bắt đầu lại. Bởi vì tiền đề là xung đột xảy ra rất không thường xuyên, nên việc khôi phục sẽ rất hiếm. Chi phí liên quan đến việc bắt đầu lại một giao dịch có thể là đáng kể, vì nó có nghĩa là thực hiện lại toàn bộ giao dịch. Điều này chỉ có thể được chấp nhận nếu nó xảy ra không thường xuyên, trong trường hợp đó, phần lớn các giao dịch sẽ được xử lý mà không bị chậm trễ. Các kỹ thuật này có khả năng cho phép đồng thời lớn hơn các giao thức truyền thống, vì không cần khóa.

Có hai hoặc ba giai đoạn đối với một giao thức điều khiển đồng thời lạc quan, tùy thuộc vào việc đó là giao dịch chỉ đọc hay giao dịch cập nhật:

- Giai đoạn đọc (Read phase). Giai đoạn này kéo dài từ khi bắt đầu giao dịch cho đến ngay trước khi cam kết. Giao dịch đọc các giá trị của tất cả các mục dữ liệu mà nó cần từ cơ sở dữ liệu và lưu trữ chúng trong các biến cục bộ. Các cập nhật được áp dụng cho bản sao cục bộ của dữ liệu, không áp dụng lên cơ sở dữ liệu.
- Giai đoạn xác thực (Validation phase). Giai đoạn này sau giai đoạn đọc. Xác thực được thực hiện để đảm bảo rằng khả năng tuần tự hóa không bị vi phạm nếu các cập nhật giao dịch được áp dụng lên cơ sở dữ liệu. Đối với giao dịch chỉ đọc, điều này bao gồm việc kiểm tra xem các giá trị dữ liệu đã đọc có còn là giá trị hiện thời của các mục dữ liệu tương ứng hay không. Nếu không có sự xung đột nào xảy ra, giao dịch sẽ được cam kết. Nếu xảy ra xung đột, giao dịch sẽ bị hủy và bắt đầu lại. Đối với một giao dịch có cập nhật, xác thực bao gồm việc xác định xem giao dịch hiện tại có khiến cho cơ sở dữ liệu ở trạng thái nhất quán hay không, với khả năng tuần tự hóa được duy trì. Nếu không, giao dịch sẽ bị hủy và cần được khởi động lại.
- Giai đoạn ghi (Write phase). Giai đoạn này theo sau giai đoạn xác thực thành công cho các giao dịch cập nhật. Trong giai đoạn này, các cập nhật đã được thực hiện đối với bản sao cục bộ được áp dụng cho cơ sở dữ liệu.

Giai đoạn xác thực kiểm tra việc đọc và ghi các giao dịch có thể gây nhiễu. Mỗi giao dịch T được gán một dấu thời gian khi bắt đầu thực hiện, start(T); một khi bắt đầu giai đoạn xác nhận, validation(T); và một ở thời điểm kết thúc, finish(T), bao gồm cả giai đoạn ghi của nó, nếu có.

Để vượt qua kiểm tra xác thực, một trong những điều sau phải đúng:

1. Tất cả các giao dịch S có dấu thời gian trước phải kết thúc trước khi giao dịch T bắt đầu; nghĩa là  $\text{finish}(S) < \text{start}(T)$ .
2. Nếu giao dịch T bắt đầu trước khi giao dịch S trước kết thúc, thì:
  - a. tập hợp các mục dữ liệu được ghi bởi giao dịch trước không phải là những mục được đọc bằng giao dịch hiện tại; và
  - b. giao dịch trước hoàn thành giai đoạn ghi của nó trước khi giao dịch hiện tại bước vào giai đoạn xác nhận, nghĩa là,  $\text{start}(T) < \text{finish}(S) < \text{validation}(T)$ .

Quy tắc 2(a) đảm bảo rằng các hoạt động ghi của một giao dịch trước không được đọc bởi giao dịch hiện tại; quy tắc 2(b) đảm bảo rằng các hoạt động ghi được thực hiện theo trình tự, đảm bảo không có xung đột.

Mặc dù các kỹ thuật lạc quan rất hiệu quả khi có ít xung đột, nhưng chúng có thể dẫn đến việc khôi phục các giao dịch riêng lẻ. Lưu ý rằng quá trình khôi phục chỉ liên quan đến một bản sao cục bộ của dữ liệu, vì vậy không có quá trình khôi phục lan truyền, bởi vì việc ghi chưa thực sự đến cơ sở dữ liệu. Tuy nhiên, nếu giao dịch bị hủy bỏ là một khoảng dài thì thời gian xử lý quý giá sẽ bị mất, vì giao dịch phải được bắt đầu lại. Nếu quá trình rollback xảy ra thường xuyên, thì đó là một dấu hiệu cho thấy rằng phương pháp lạc quan là một lựa chọn tồi để điều khiển đồng thời trong môi trường cụ thể đó.

## TÓM TẮT

- Điều khiển đồng thời (concurrency control) là quá trình quản lý các hoạt động đồng thời trên cơ sở dữ liệu nhằm không để chúng xen vào nhau.
- Phục hồi cơ sở dữ liệu (database recovery) là quá trình khôi phục cơ sở dữ liệu về trạng thái chính xác sau khi bị lỗi.
- Giao dịch (transaction) là một hành động, hoặc một loạt các hành động, được thực hiện bởi một người dùng hoặc chương trình ứng dụng, truy cập hoặc thay đổi nội dung của cơ sở dữ liệu. Giao dịch là một đơn vị công việc (unit of work) hợp lý đưa cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác. Các giao dịch có thể kết thúc thành công (commit) hoặc không thành công (abort). Các giao dịch bị hủy bỏ phải được hoàn tác (rollback).
- Một giao dịch phải có bốn thuộc tính cơ bản (ACID): tính nguyên tử (atomicity), tính nhất quán (consistency), tính cô lập (isolation) và tính bền vững (durability).
- Điều khiển đồng thời là cần thiết khi nhiều người dùng được phép truy cập cơ sở dữ liệu đồng thời. Nếu không có nó, các vấn đề về cập nhật bị mất, phụ thuộc không được xử lý và phân tích không nhất quán có thể phát sinh.
- Thực thi tuần tự nghĩa là thực hiện một giao dịch tại một thời điểm, không có sự đan xen các hoạt động của các giao dịch khác.
- Một lịch trình (schedule) cho thấy trình tự của các hoạt động của các giao dịch. Một lịch trình khả tuần tự (serializable) nếu nó tạo ra kết quả giống như một lịch trình tuần tự nào đó.
- Hai phương pháp đảm bảo khả năng tuần tự của các lịch trình là khóa hai pha (two-phase locking - 2PL) và dấu thời gian (timestamping).
- Tắc nghẽn (deadlock) xảy ra khi một hoặc nhiều giao dịch đang chờ truy cập dữ liệu mà giao dịch khác đã khóa. Cách duy nhất để phá vỡ deadlock là hủy bỏ một hoặc nhiều giao dịch.
- Để thuận lợi cho việc khôi phục, hệ thống duy trì một tập tin nhật ký (log file) chứa các bản ghi giao dịch xác định điểm bắt đầu/kết thúc của các giao dịch và hình ảnh trước và sau của dữ liệu đối với các hoạt động ghi.
- Sử dụng cập nhật trì hoãn (deferred update), đầu tiên các hoạt động ghi chỉ thực hiện trên nhật ký và các bản ghi nhật ký được sử dụng để thực hiện cập nhật thực tế cho cơ sở dữ liệu. Nếu hệ thống bị lỗi, sẽ kiểm tra nhật ký để xác định giao dịch nào cần thực hiện lại, nhưng không cần hoàn tác hoạt động ghi nào.
- Sử dụng cập nhật tức thời (immediate update), một cập nhật có thể được thực hiện trên chính cơ sở dữ liệu bất kỳ lúc nào sau khi bản ghi nhật ký đã được ghi lại. Nhật ký được sử dụng để hoàn tác và thực hiện lại các giao dịch trong trường hợp hệ thống bị lỗi.

## CÂU HỎI

1. Trình bày lý do tại sao có thể xảy ra sự không nhất quán và mất dữ liệu. Tại sao cần phải quản lý các giao dịch đồng thời?
2. Các khía cạnh nhất quán và độ tin cậy của các giao dịch phụ thuộc vào tính ACID của các giao dịch. Trình bày từng thuộc tính này và cách chúng liên quan đến các cơ chế điều khiển đồng thời và khôi phục.
3. Mô tả, với các ví dụ, các loại sự cố có thể xảy ra trong môi trường đa người dùng khi cho phép truy cập đồng thời vào cơ sở dữ liệu.
4. Giải thích các khái niệm về lịch trình tuần tự, không tuần tự và khả tuần tự. Nêu các quy tắc về sự tương đương của các lịch trình.
5. Hệ thống con của trình quản lý giao dịch DBMS đóng vai trò gì trong hệ thống đa người dùng?
6. Trình bày các loại vấn đề có thể xảy ra với các cơ chế điều khiển đồng thời dựa trên khóa và các hành động mà một DBMS có thể thực hiện để ngăn chặn chúng.
7. Dấu thời gian là gì? Các giao thức dựa trên dấu thời gian điều khiển đồng thời khác với các giao thức dựa trên khóa như thế nào?
8. Mô tả giao thức thứ tự dấu thời gian (timestamp ordering protocol) cơ bản để điều khiển đồng thời.
9. Trình bày sự khác biệt giữa điều khiển đồng thời bi quan và lạc quan.
10. Trình bày các loại lỗi có thể xảy ra trong môi trường cơ sở dữ liệu. Giải thích tại sao điều quan trọng đối với DBMS đa người dùng là cung cấp cơ chế khôi phục.
11. Trình bày vì sao tập tin nhật ký là một tính năng cơ bản trong bất kỳ cơ chế khôi phục nào? Ý nghĩa của giao thức nhật ký được ghi trước (write-ahead log protocol) là gì? Các checkpoint ảnh hưởng đến giao thức khôi phục như thế nào?

## CHƯƠNG 5

### TỐI ƯU HÓA TRUY VẤN

Trong chương này, chúng ta sẽ tìm hiểu:

- Các mục tiêu của xử lý truy vấn và tối ưu hóa truy vấn.
- Tối ưu hóa truy vấn tĩnh và động.
- Cách một truy vấn được phân tách và phân tích ngữ nghĩa.
- Cách tạo cây đại số quan hệ để biểu diễn một truy vấn.
- Các quy tắc tương đương cho các phép toán đại số quan hệ.
- Cách áp dụng các quy tắc biến đổi heuristic để cải thiện hiệu quả của truy vấn.
- Các loại thông kê cơ sở dữ liệu cần thiết để ước tính chi phí hoạt động.
- Cách đánh giá chi phí và quy mô của các phép toán đại số quan hệ.

Khi mô hình cơ sở dữ liệu quan hệ lần đầu tiên được đưa ra thương mại, một trong những chỉ trích chính thường là hiệu suất của các truy vấn không thỏa đáng. Kể từ đó, một lượng lớn nghiên cứu đã được dành để phát triển các thuật toán hiệu quả cao để xử lý các truy vấn. Có nhiều cách mà một truy vấn phức tạp có thể được thực hiện và một trong những mục đích của việc xử lý truy vấn là xác định cách nào là hiệu quả nhất về chi phí.

Trong hệ cơ sở dữ liệu phân cấp và mạng thế hệ thứ nhất, ngôn ngữ truy vấn thủ tục cấp thấp thường được nhúng trong ngôn ngữ lập trình cấp cao và lập trình viên có trách nhiệm chọn chiến lược thực thi phù hợp nhất. Ngược lại, với các ngôn ngữ khai báo như SQL, người dùng sẽ chỉ định dữ liệu gì được yêu cầu hơn là cách nó được truy xuất.

Điều này giúp người dùng có trách nhiệm xác định, hoặc thậm chí biết, điều gì tạo nên một chiến lược thực thi tốt và làm cho ngôn ngữ này có thể sử dụng được trên toàn cầu. Ngoài ra, việc trao cho DBMS trách nhiệm lựa chọn chiến lược tốt nhất sẽ ngăn người dùng chọn các chiến lược được cho là không hiệu quả và cho phép DBMS kiểm soát nhiều hơn hiệu suất của hệ thống.

Có hai kỹ thuật chính để tối ưu hóa truy vấn, mặc dù hai kỹ thuật này thường được kết hợp trong thực tế. Kỹ thuật đầu tiên sử dụng các quy tắc heuristic để sắp xếp các phép toán trong một truy vấn. Kỹ thuật thứ hai là so sánh các chiến lược khác nhau dựa trên chi phí tương đối của chúng và chọn một chiến lược giảm thiểu việc sử dụng tài nguyên nhất. Các thao tác truy cập đĩa chậm hơn so với truy cập bộ nhớ, nhưng truy cập đĩa có xu hướng chiếm ưu thế trong quá trình xử lý truy vấn đối với DBMS tập trung, nên đây sẽ là yếu tố chính khi tìm hiểu về kỹ thuật ước tính chi phí.

## 5.1 TỔNG QUAN VỀ XỬ LÝ TRUY VẤN

### Xử lý truy vấn

Các hoạt động liên quan đến phân tích cú pháp, xác thực, tối ưu hóa và thực thi một truy vấn.

Mục tiêu của xử lý truy vấn (query processing) là chuyển đổi một truy vấn được viết bằng ngôn ngữ cấp cao (SQL) thành một chiến lược thực thi đúng và hiệu quả được thể hiện bằng ngôn ngữ cấp thấp (đại số quan hệ) và thực thi chiến lược đó để truy xuất dữ liệu được yêu cầu.

### Tối ưu hóa truy vấn

Hoạt động chọn một chiến lược thực thi hiệu quả để xử lý một truy vấn.

Một khía cạnh quan trọng của xử lý truy vấn là tối ưu hóa truy vấn (query optimization). Vì có nhiều biến đổi tương đương của cùng một truy vấn cấp cao, mục đích của việc tối ưu hóa truy vấn là chọn một biến đổi giảm thiểu việc sử dụng tài nguyên nhất. Nói chung, chúng ta cố gắng giảm thời gian thực thi của truy vấn - là tổng thời gian thực thi của tất cả các phép toán riêng lẻ tạo nên truy vấn. Tuy nhiên, việc sử dụng tài nguyên cũng có thể được xem như thời gian phản hồi của truy vấn, trong trường hợp đó, chúng ta tập trung vào việc tối đa hóa số lượng phép toán song song.

Cả hai phương pháp tối ưu hóa truy vấn đều phụ thuộc vào các thông kê cơ sở dữ liệu để đánh giá đúng các tùy chọn khác nhau có sẵn. Độ chính xác và tính tức thời của những thông kê này có ảnh hưởng đáng kể đến hiệu quả của chiến lược thực hiện đã chọn. Thông kê bao gồm thông tin về các quan hệ, thuộc tính và chỉ mục. Ví dụ, danh mục hệ thống có thể lưu trữ số liệu thông kê về bản số của các quan hệ, số lượng các giá trị riêng biệt cho mỗi thuộc tính và số lượng cấp của một chỉ mục đa cấp. Giữ cho các số liệu thông kê luôn thỏa mãn tính tức thời có thể là một vấn đề. Nếu DBMS cập nhật thông kê mỗi khi một bộ dữ liệu được chèn, cập nhật hoặc xóa, thì điều này sẽ có tác động đáng kể đến hiệu suất trong thời gian cao điểm. Một phương pháp thay thế, và thường được ưu tiên hơn, là cập nhật số liệu thông kê một cách định kỳ; ví dụ, hàng đêm hoặc bất cứ khi nào hệ thống không hoạt động. Một cách tiếp cận khác được một số hệ thống thực hiện là khiến người dùng có trách nhiệm định ra thời điểm các số liệu thông kê sẽ được cập nhật.

Để minh họa về ảnh hưởng của các chiến lược xử lý khác nhau đối với việc sử dụng tài nguyên, chúng ta bắt đầu với một ví dụ.

### Ví dụ 5.1 So sánh các chiến lược xử lý khác nhau

*Liệt kê tất cả các nhân viên Quản lý (Manager) làm việc tại các chi nhánh ở London*

Chúng ta có thể viết truy vấn này bằng SQL dưới dạng:

**SELECT \***

**FROM Staff s, Branch b**

**WHERE s.branchNo = b.branchNo AND**

**(s.position = 'Manager' AND b.city = 'London');**

Ba truy vấn đại số quan hệ tương đương tương với câu lệnh SQL này là:

- (1)  $\sigma_{(position='Manager') \wedge (city = 'London')} \wedge (Staff.branchNo = Branch.branchNo)$  (Staff  $\times$  Branch)
- (2)  $\sigma_{(position='Manager') \wedge (city = 'London')}$  (Staff  $\bowtie_{Staff.branchNo=Branch.branchNo}$  Branch)
- (3)  $(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city = 'London'}(Branch))$

Trong ví dụ này, chúng ta giả định rằng có 1000 bộ trong Staff, 50 bộ trong Branch, 50 Managers (một Manager cho mỗi chi nhánh) và 5 chi nhánh ở London. Chúng ta so sánh ba truy vấn này dựa trên số lượng truy cập đĩa được yêu cầu. Để đơn giản, chúng ta giả định rằng không có chỉ mục hoặc khóa sắp xếp trên cả hai quan hệ và kết quả của bất kỳ hoạt động trung gian nào đều được lưu trữ trên đĩa. Chi phí của lần ghi cuối cùng được bỏ qua, vì nó giống nhau trong mỗi trường hợp. Chúng ta giả định thêm rằng các bộ được truy cập lần lượt (mặc dù trong thực tế, việc truy cập đĩa sẽ dựa trên các khối, thường chứa một số bộ) và bộ nhớ chính đủ lớn để xử lý toàn bộ các quan hệ cho mỗi phép toán đại số quan hệ.

Truy vấn đầu tiên tính tích Đề-các của Staff và Branch, yêu cầu  $(1000 + 50)$  truy cập đĩa để đọc các quan hệ và tạo một quan hệ với  $(1000 * 50)$  bộ. Sau đó, chúng ta phải đọc lại từng bộ này để kiểm tra chúng so với vị trí lựa chọn với chi phí bằng một lần  $(1000 * 50)$  truy cập đĩa khác. Do đó, tổng chi phí là:

$$(1000 + 50) + 2 * (1000 * 50) = 101050 \text{ truy cập đĩa}$$

Truy vấn thứ hai nối Staff với Branch dựa trên số chi nhánh branchNo, truy vấn này một lần nữa yêu cầu  $(1000 + 50)$  truy cập đĩa để đọc từng quan hệ. Chúng ta biết rằng phép Nối của hai quan hệ có 1000 bộ, một bộ cho mỗi nhân viên (một nhân viên chỉ có thể làm việc tại một chi nhánh). Do đó, phép Chọn yêu cầu 1000 truy cập đĩa để đọc kết quả của phép Nối. Do đó, tổng chi phí là:

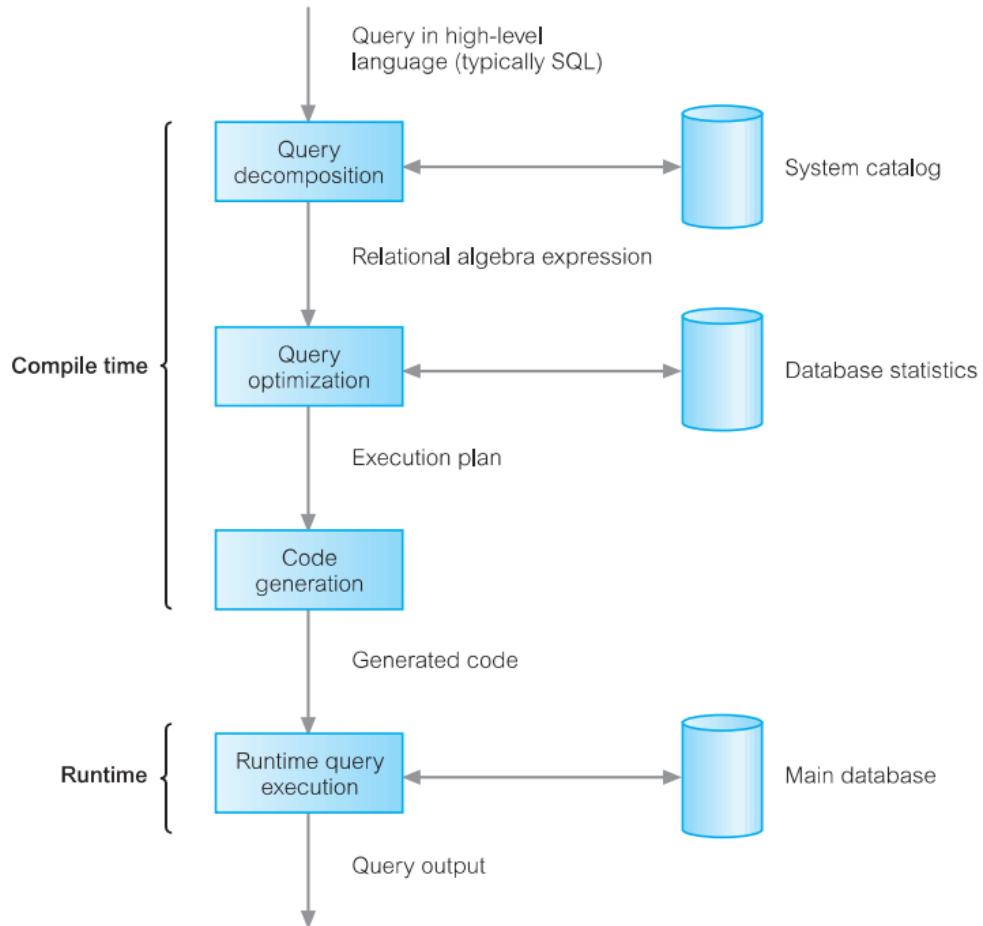
$$2 * 1000 + (1000 + 50) = 3050 \text{ truy cập đĩa}$$

Truy vấn cuối cùng trước tiên đọc từng bộ của Staff để xác định các bộ Manager, thao tác này yêu cầu 1000 truy cập đĩa và tạo ra một quan hệ với 50 bộ. Phép Chọn thứ hai đọc từng bộ của Branch để xác định các chi nhánh ở London, thao tác này yêu cầu 50 truy cập đĩa và tạo ra một quan hệ với 5 bộ. Thao tác cuối cùng là phép Nối các quan hệ Staff và Branch đã giảm, yêu cầu  $(50 + 5)$  truy cập đĩa. Do đó, tổng chi phí là:

$$1000 + 2 * 50 + 5 + (50 + 5) = 1160 \text{ truy cập đĩa}$$

Rõ ràng là chiến lược thứ ba là tốt nhất trong trường hợp này, theo hệ số 87:1. Nếu chúng ta tăng số lượng bộ của Staff lên 10.000 và số chi nhánh lên 500, thì sự cải thiện sẽ là theo hệ số khoảng 870:1. Theo trực giác, chúng ta có thể đã mong đợi điều này, vì phép tích Đề-các và phép Nối đặt hơn nhiều so với phép Chọn và chiến lược thứ ba làm giảm đáng kể kích thước của các quan hệ được kết hợp với nhau. Chúng ta sẽ thấy ngay rằng một trong những chiến lược cơ bản trong xử lý truy vấn là thực hiện các phép toán một ngôi Chọn và Chiếu càng sớm càng tốt, dẫn đến giảm các toán hạng của bất kỳ phép toán nhị phân nào tiếp theo.

Quá trình xử lý truy vấn có thể được chia thành bốn giai đoạn chính: phân rã (bao gồm phân tích cú pháp và xác thực), tối ưu hóa, tạo mã và thực thi, như minh họa trong Hình 5.1. Trong mục 5.2, chúng ta sẽ tìm hiểu ngắn gọn giai đoạn đầu tiên, sự phân rã, trước khi chuyển sang giai đoạn thứ hai, tối ưu hóa truy vấn. Để hoàn thành tổng quan này, chúng ta thảo luận ngắn gọn về thời điểm tối ưu hóa có thể được thực hiện.



**Hình 5.1** Các giai đoạn xử lý truy vấn.

Có hai lựa chọn về thời điểm có thể thực hiện ba giai đoạn đầu tiên của quá trình xử lý truy vấn. Một tùy chọn là thực hiện phân rã và tối ưu hóa mỗi khi truy vấn được chạy. Ưu điểm của tối ưu hóa truy vấn động (dynamic query optimization) xuất phát từ thực tế là tất cả thông tin cần thiết để chọn một chiến lược tối ưu đều được cập nhật. Những bất lợi là hiệu suất của truy vấn bị ảnh hưởng vì truy vấn phải được phân tích cú pháp, xác thực và tối ưu hóa trước khi có thể được thực thi. Hơn nữa, có thể cần phải giảm số lượng các chiến lược thực thi cần phân tích để đạt được chi phí chấp nhận được, điều này có thể dẫn đến việc lựa chọn một chiến lược ít tối ưu hơn.

Tùy chọn thay thế là tối ưu hóa truy vấn tĩnh (static query optimization), trong đó truy vấn được phân tích cú pháp, xác thực và tối ưu hóa chỉ một lần. Cách tiếp cận này tương tự như cách tiếp cận được thực hiện bởi trình biên dịch cho một ngôn ngữ lập trình. Ưu điểm của tối ưu hóa tĩnh là chi phí thời gian chạy được loại bỏ và có thể có nhiều thời gian hơn để đánh giá số lượng lớn hơn các chiến lược thực thi, do đó tăng cơ hội tìm được chiến lược tối ưu hơn.

Đối với các truy vấn được thực hiện nhiều lần, việc dành thêm thời gian để tìm ra phương án tối ưu hơn có thể mang lại lợi ích cao. Những bất lợi này sinh từ thực tế là chiến lược thực thi được chọn là tối ưu khi truy vấn được biên dịch có thể không còn là tối ưu khi truy vấn được chạy. Tuy nhiên, một cách tiếp cận kết hợp có thể được sử dụng để khắc phục nhược điểm này, theo đó, truy vấn sẽ được tối ưu hóa lại nếu hệ thống phát hiện rằng thông kê cơ sở dữ liệu đã thay đổi đáng kể kể từ lần cuối cùng truy vấn được biên dịch. Ngoài ra, hệ thống có thể biên dịch truy vấn cho lần thực thi đầu tiên trong mỗi phiên của DBMS và lưu vào bộ nhớ cache chiến lược tối ưu đó để thực thi cho các lần gọi sau, vì vậy chi phí sẽ được dàn trải trên toàn bộ phiên làm việc.

## 5.2 PHÂN RÃ TRUY VẤN

Phân rã truy vấn là giai đoạn đầu tiên của quá trình xử lý truy vấn. Mục đích của việc phân rã truy vấn là chuyển một truy vấn cấp cao thành một truy vấn đại số quan hệ và kiểm tra xem truy vấn có đúng về mặt cú pháp và ngữ nghĩa hay không. Các giai đoạn điển hình của phân rã truy vấn là phân tích, chuẩn hóa, phân tích ngữ nghĩa, đơn giản hóa và tái cấu trúc truy vấn.

### 5.2.1 Phân tích

Trong giai đoạn này, truy vấn được phân tích từ vựng và cú pháp bằng cách sử dụng các kỹ thuật của trình biên dịch ngôn ngữ lập trình. Ngoài ra, giai đoạn này xác thực rằng các quan hệ và thuộc tính được chỉ định trong truy vấn đã được định nghĩa trong danh mục hệ thống. Nó cũng xác thực rằng bất kỳ hoạt động nào được áp dụng lên các đối tượng cơ sở dữ liệu là phù hợp với kiểu đối tượng đó. Ví dụ, hãy xem xét truy vấn sau:

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

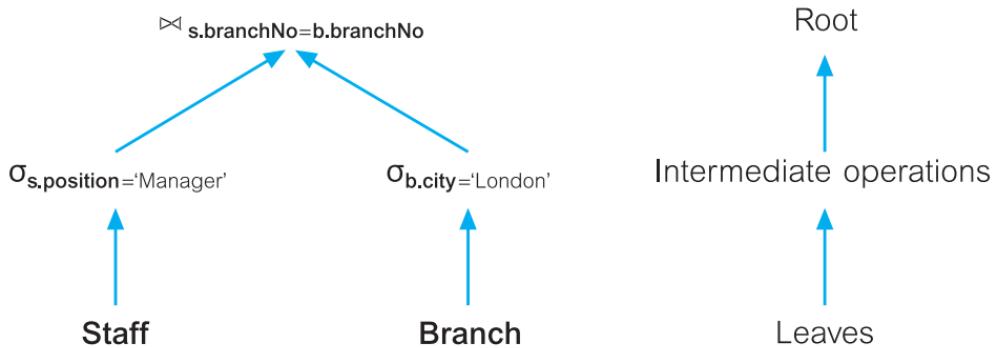
Truy vấn này sẽ bị từ chối vì hai lý do:

1. Trong danh sách chọn, thuộc tính staffNumber không được xác định trong quan hệ Staff (nên là staffNo).
2. Trong mệnh đề WHERE, so sánh “> 10” không tương thích với kiểu dữ liệu của position, vì đây là một chuỗi ký tự.

Khi hoàn thành giai đoạn này, truy vấn cấp cao đã được chuyển đổi thành một biểu diễn bên trong nào đó phù hợp hơn để xử lý. Dạng bên trong này thường được chọn là dạng cây, được xây dựng như sau:

- Một nút lá được tạo cho mỗi quan hệ cơ sở trong truy vấn.
- Một nút không phải nút lá được tạo cho mỗi quan hệ trung gian được tạo ra bởi một phép toán đại số quan hệ.
- Gốc của cây biểu diễn cho kết quả của truy vấn.
- Thứ tự các phép toán hướng từ lá đến gốc.

Hình 5.2 cho thấy một ví dụ về cây truy vấn cho câu lệnh SQL của Ví dụ 10.1. Chúng ta gọi loại cây truy vấn này là cây đại số quan hệ (relational algebra tree).



**Hình 5.2** Ví dụ về cây đại số quan hệ.

### 5.2.2 Chuẩn hóa

Giai đoạn chuẩn hóa của quá trình xử lý truy vấn chuyển truy vấn thành một dạng đã được chuẩn hóa để có thể dễ dàng thao tác hơn. Ví từ (trong SQL, điều kiện WHERE), có thể phức tạp và tùy ý, sẽ được chuyển đổi thành một trong hai dạng bằng cách áp dụng một số quy tắc biến đổi:

- Dạng chuẩn tắc hội (conjunctive normal form): Một hay nhiều tuyển sơ cấp được nối với nhau bằng toán tử  $\wedge$  (AND). Tuyển sơ cấp chứa một hoặc nhiều mệnh đề được kết nối bởi toán tử  $\vee$  (OR). Ví dụ:

$(\text{position} = \text{'Manager'}) \vee (\text{salary} > 20000) \wedge (\text{branchNo} = \text{'B003'})$

Kết quả một phép Chọn hội chỉ chứa những bộ thỏa mãn tất cả các tuyển sơ cấp.

- Dạng chuẩn tắc tuyển (disjunctive normal form): Một hay nhiều hội sơ cấp được nối với nhau bằng toán tử  $\vee$  (OR). Hội sơ cấp chứa một hoặc nhiều mệnh đề được kết nối bởi toán tử  $\wedge$  (AND). Ví dụ, chúng ta có thể viết lại dạng chuẩn tắc hội trên thành:

$(\text{position} = \text{'Manager'}) \wedge (\text{branchNo} = \text{'B003'}) \vee ((\text{salary} > 20000) \wedge (\text{branchNo} = \text{'B003'}))$

Kết quả một phép Chọn tuyển chứa tất cả các bộ thỏa mãn ít nhất một hội sơ cấp.

### 5.2.3 Phân tích ngữ nghĩa

Mục tiêu của phân tích ngữ nghĩa là từ chối các truy vấn đã chuẩn hóa được xây dựng không chính xác hoặc mâu thuẫn. Truy vấn được xây dựng không chính xác nếu các thành phần không góp phần tạo ra kết quả, điều này có thể xảy ra nếu thiếu một số thông số kỹ thuật của phép nối. Một truy vấn mâu thuẫn nếu vị trí của nó không thể được thỏa mãn bởi bất kỳ bộ nào. Ví dụ, ví từ  $(\text{position} = \text{'Manager'}) \wedge (\text{position} = \text{'Assistant'})$  trong quan hệ Staff là mâu thuẫn, vì một nhân viên không thể đồng thời là Manager và Assistant. Do đó, ví từ  $((\text{position} = \text{'Manager'}) \wedge (\text{position} = \text{'Assistant'})) \vee (\text{salary} > 20000)$  có thể được đơn giản hóa thành  $(\text{salary} > 20000)$  bằng cách diễn dịch mệnh đề mâu thuẫn là giá trị boolean FALSE.

Các thuật toán để xác định tính đúng đắn chỉ tồn tại cho tập hợp con các truy vấn không chứa tuyển và phủ định. Đối với những truy vấn này, chúng ta có thể áp dụng các kiểm tra sau:

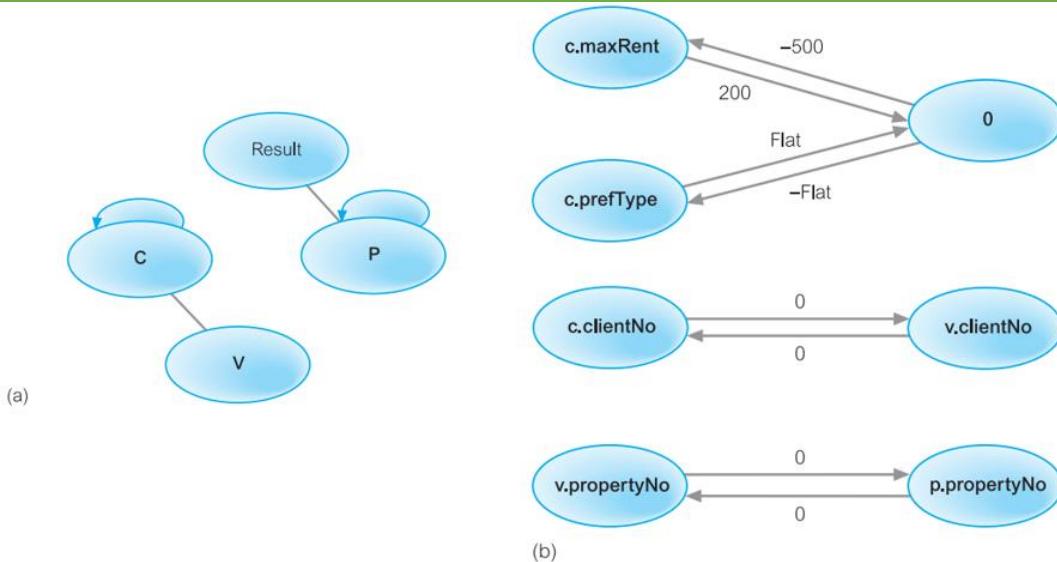
1. Xây dựng đồ thị kết nối quan hệ (relation connection graph). Nếu đồ thị không được kết nối, truy vấn được xây dựng không chính xác. Để xây dựng một đồ thị kết nối quan hệ, chúng ta tạo một nút cho mỗi quan hệ và một nút cho kết quả. Sau đó, chúng ta tạo các cạnh giữa hai nút biểu diễn cho một phép nối và các cạnh giữa các nút đại diện cho nguồn của các phép Chiếu.
2. Xây dựng đồ thị kết nối thuộc tính chuẩn hóa (normalized attribute connection graph). Nếu đồ thị có chu trình mà tổng giá trị là âm, thì truy vấn là mâu thuẫn. Để tạo một đồ thị kết nối thuộc tính chuẩn hóa, chúng ta tạo một nút cho mỗi tham chiếu đến một thuộc tính hoặc hằng số 0. Sau đó, chúng ta tạo một cạnh có hướng giữa các nút biểu diễn cho một nối và một cạnh có hướng giữa một nút thuộc tính và một nút hằng số 0 mà biểu diễn cho một phép Chọn. Tiếp theo, chúng ta tính các cạnh  $a \rightarrow b$  với giá trị  $c$ , nếu nó biểu diễn cho điều kiện không bằng ( $a \leq b + c$ ), và tính các cạnh  $0 \rightarrow a$  với giá trị  $-c$ , nếu nó biểu diễn cho điều kiện không bằng ( $a \geq c$ ).

### Ví dụ 5.2 Kiểm tra tính đúng đắn về ngữ nghĩa

Hãy xem xét truy vấn SQL sau:

```
SELECT p.propertyNo, p.street  
FROM Client c, Viewing v, PropertyForRent p  
WHERE c.clientNo = v.clientNo AND  
      c.maxRent > = 500 AND  
      c.prefType = 'Flat' AND  
      p.ownerNo = 'CO93';
```

Đồ thị kết nối quan hệ trong Hình 5.3(a) không được kết nối đầy đủ, ngụ ý rằng truy vấn không được xây dựng chính xác. Trong trường hợp này, chúng ta đã bỏ qua điều kiện nối ( $v.propertyNo = p.propertyNo$ ) của vị từ.



**Hình 5.3** (a) Đồ thị kết nối quan hệ hiển thị truy vấn được xây dựng không chính xác; (b) Đồ thị kết nối thuộc tính chuẩn hóa hiển thị truy vấn là mâu thuẫn.

Bây giờ hãy xem xét truy vấn sau:

```

SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE c.maxRent > 500 AND
        c.clientNo = v.clientNo AND
        v.propertyNo = p.propertyNo AND
        c.prefType = 'Flat' AND c.maxRent < 200;
    
```

Đồ thị kết nối thuộc tính chuẩn hóa cho truy vấn này được hiển thị trong Hình 5.3(b) có chu trình giữa các nút c.maxRent và 0 với tổng giá trị âm, điều này cho thấy rằng truy vấn là mâu thuẫn. Rõ ràng, chúng ta không thể có một khách hàng có giá thuê tối đa lớn hơn \$500 và dưới \$200.

#### 5.2.4 Đơn giản hóa

Mục tiêu của giai đoạn đơn giản hóa là phát hiện các điều kiện thừa, loại bỏ các biểu thức con thông thường và chuyển đổi truy vấn sang dạng tương đương về mặt ngữ nghĩa nhưng dễ dàng và hiệu quả hơn về mặt tính toán. Thông thường, các hạn chế truy cập, định nghĩa khung nhìn và các ràng buộc toàn vẹn được xem xét ở giai đoạn này. Nếu người dùng không có quyền truy cập thích hợp vào tất cả các thành phần của truy vấn, truy vấn phải bị từ chối. Đơn giản hóa khởi đầu là áp dụng các luật lũy đồng (idempotency rule) nổi tiếng của đại số boolean như:

$$\begin{array}{ll}
 p \wedge p \equiv p & p \vee (p) \equiv p \\
 p \wedge \text{false} \equiv \text{false} & p \vee \text{false} \equiv p \\
 p \wedge \text{true} \equiv p & p \vee \text{true} \equiv \text{true} \\
 p \wedge (\sim p) \equiv \text{false} & p \vee (\sim p) \equiv \text{true} \\
 p \wedge (p \vee q) \equiv p & p \vee (p \wedge q) \equiv p
 \end{array}$$

Ví dụ, hãy xem xét định nghĩa khung nhìn Staff3 sau đây và truy vấn trên khung nhìn này:

**CREATE VIEW Staff3 AS**

**SELECT** staffNo, fName, lName, salary, branchNo

**FROM** Staff

**WHERE** branchNo = 'B003';

**SELECT \***

**FROM** Staff3

**WHERE** (branchNo = 'B003' **AND** salary > 20000);

Như đã thảo luận trong mục 3.3.3, sau khi phân giải khung nhìn, truy vấn này sẽ trở thành:

**SELECT** staffNo, fName, lName, salary, branchNo

**FROM** Staff

**WHERE** (branchNo = 'B003' **AND** salary > 20000) **AND** branchNo = 'B003';

và điều kiện WHERE rút gọn thành (branchNo = 'B003' AND salary > 20000).

Các ràng buộc về tính toàn vẹn cũng có thể được áp dụng để giúp đơn giản hóa các truy vấn. Ví dụ, hãy xem xét ràng buộc về tính toàn vẹn sau đây, ràng buộc này đảm bảo rằng chỉ nhân viên Manager mới có mức lương lớn hơn \$20.000:

**CREATE ASSERTION** OnlyManagerSalaryHigh

**CHECK** ((position <> 'Manager' **AND** salary < 20000)

**OR** (position = 'Manager' **AND** salary > 20000));

và xem xét ảnh hưởng đến truy vấn:

**SELECT \***

**FROM** Staff

**WHERE** (position = 'Manager' **AND** salary < 15000);

Vì từ trong mệnh đề WHERE, tìm kiếm một nhân viên Manager có mức lương dưới \$15.000, là một mâu thuẫn với ràng buộc toàn vẹn nên không thể có bộ nào thỏa mãn vị từ này.

### 5.2.5 Tái cấu trúc truy vấn

Trong giai đoạn cuối cùng của quá trình phân rã truy vấn, truy vấn được cấu trúc lại để cung cấp một triển khai hiệu quả hơn.

## 5.3 TIẾP CẬN HEURISTIC ĐỂ TỐI UU HÓA TRUY VẤN

Trong phần này, chúng ta xem xét cách tiếp cận heuristic để tối ưu hóa truy vấn, sử dụng các quy tắc chuyển đổi để chuyển đổi một biểu thức đại số quan hệ thành một dạng tương đương được cho là hiệu quả hơn. Ví dụ, trong Ví dụ 10.1, chúng ta nhận thấy rằng sẽ hiệu quả hơn khi thực hiện phép Chọn trên một quan hệ trước khi sử dụng quan hệ đó trong một phép Nối, thay vì thực hiện Nối và sau đó là phép Chọn. Chúng ta sẽ trình bày trong mục 5.3.1 quy tắc chuyển đổi cho phép thay đổi thứ tự của các phép Nối và phép Chọn để phép Chọn có thể được thực hiện trước. Sau khi tìm hiểu về các phép biến đổi hợp lệ, trong mục 5.3.2, chúng ta tìm hiểu một tập hợp các quy luật heuristic được cho là tạo ra các chiến lược thực thi “tốt” (không nhất thiết là tối ưu).

### 5.3.1 Các quy tắc biến đổi cho các phép toán đại số quan hệ

Bằng cách áp dụng các quy tắc biến đổi, trình tối ưu hóa có thể biến đổi một biểu thức đại số quan hệ thành một biểu thức tương đương được cho là hiệu quả hơn. Chúng ta sẽ sử dụng các quy tắc này để cấu trúc lại cây đại số quan hệ (chính tắc) được tạo ra trong quá trình phân rã truy vấn. Khi liệt kê các quy tắc này, chúng ta sử dụng ba quan hệ R, S và T, với R được xác định trên các thuộc tính  $A = \{A_1, A_2, \dots, A_n\}$  và S được xác định trên  $B = \{B_1, B_2, \dots, B_n\}$ ; p, q và r biểu thị các vị trí và L, L<sub>1</sub>, L<sub>2</sub>, M, M<sub>1</sub>, M<sub>2</sub> và N biểu thị các tập thuộc tính.

#### (1) Tính phân phối của phép Chọn

$$\sigma_p \wedge p \wedge r(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

Ví dụ:

$$\sigma_{\text{brandNo}='B003' \wedge \text{salary}>15000}(\text{Staff}) = \sigma_{\text{brandNo}='B003'}(\sigma_{\text{salary}>15000}(\text{Staff}))$$

#### (2) Tính giao hoán của phép Chọn

$$\sigma_p(\sigma_p(R)) = \sigma_q(\sigma_p(R))$$

Ví dụ:

$$\sigma_{\text{brandNo}='B003'}(\sigma_{\text{salary}>15000}(\text{Staff})) = \sigma_{\text{salary}>15000}(\sigma_{\text{brandNo}='B003'}(\text{Staff}))$$

#### (3) Tính phân phối của phép Chiếu

Trong một dãy các phép Chiếu, chỉ yêu cầu phép Chiếu cuối cùng

$$\pi_L(\pi_M(\dots \pi_N(R))) = \pi_L(R)$$

Ví dụ:

$$\pi_{\text{lName}}(\pi_{\text{brandNo}, \text{lName}}(\text{Staff})) = \pi_{\text{lName}}(\text{Staff})$$

#### (4) Tính giao hoán của phép Chọn và phép Chiếu.

Nếu vị trí p chỉ liên quan đến các thuộc tính trong danh sách phép Chiếu, thì các phép Chọn và phép Chiếu sẽ hoán chuyển với nhau:

$$\pi_{A_1, \dots, A_m}(\sigma_p(R)) = \sigma_p(\pi_{A_1, \dots, A_m}(R)) \quad \text{với } p \in \{A_1, A_2, \dots, A_m\}$$

Ví dụ:

$$\pi_{\text{fName}, \text{lName}}(\sigma_{\text{lName}='Beech'}(\text{Staff})) = \sigma_{\text{lName}='Beech'}(\pi_{\text{fName}, \text{lName}}(\text{Staff}))$$

**(5) Tính giao hoán của phép Nối Theta (và tích Đè-các).**

$$R \bowtie_p S = S \bowtie_p R$$

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

Vì phép nối Equi và phép nối Natural là các trường hợp đặc biệt của phép nối Theta, nên quy tắc này cũng áp dụng cho các phép nối đã nêu. Ví dụ, sử dụng phép nối Equi với các quan hệ Staff và Branch:

$$\text{Staff} \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} \text{Branch} = \text{Branch} \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} \text{Staff}$$

**(6) Tính giao hoán của phép Chọn và phép Nối Theta (hoặc tích Đè-các).**

Nếu vị tự lựa chọn chỉ liên quan đến các thuộc tính của một trong các quan hệ đang được nối, thì các phép Chọn và phép Nối (hoặc tích Đè-các) sẽ thay đổi như sau:

$$\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$$

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

Ngoài ra, nếu vị tự lựa chọn có dạng  $(p \wedge q)$ , trong đó  $p$  chỉ liên quan đến các thuộc tính của  $R$  và  $q$  chỉ liên quan đến các thuộc tính của  $S$ , thì các phép Chọn và phép Nối Theta sẽ hoán chuyển như sau:

$$\sigma_p \wedge q(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$$

$$\sigma_p \wedge q(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$$

Ví dụ:

$$\begin{aligned} & \sigma_{\text{position} = \text{'Manager'}} \wedge \text{city} = \text{'London'} (\text{Staff} \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} \text{Branch}) = (\sigma_{\text{position} = \text{'Manager'}} \\ & (\text{Staff})) \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} (\sigma_{\text{city} = \text{'London'}}(\text{Branch})) \end{aligned}$$

**(7) Tính giao hoán của phép Chiếu và phép Nối Theta (hoặc tích Đè-các).**

Nếu danh sách chiếu có dạng  $L = L_1 \cup L_2$ , trong đó  $L_1$  chỉ liên quan đến các thuộc tính của  $R$  và  $L_2$  chỉ liên quan đến các thuộc tính của  $S$ , và điều kiện nối chỉ chứa các thuộc tính của  $L$  thì các phép Nối Theta và phép Chiếu sẽ biến đổi như sau:

$$\pi_{L_1 \cup L_2}(R \bowtie_r S) = (\pi_{L_1}(R) \bowtie_r \pi_{L_2}(S))$$

Ví dụ:

$$\begin{aligned} & \Pi_{\text{position}, \text{city}, \text{branchNo}} (\text{Staff} \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} \text{Branch}) = \\ & (\Pi_{\text{position}, \text{branchNo}}(\text{Staff})) \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} (\Pi_{\text{city}, \text{branchNo}}(\text{Branch})) \end{aligned}$$

Nếu điều kiện nối chứa các thuộc tính bổ sung không có trong  $L$ , giả sử các thuộc tính  $M = M_1 \cup M_2$  trong đó  $M_1$  chỉ liên quan đến các thuộc tính của  $R$  và  $M_2$  chỉ liên quan đến các thuộc tính của  $S$ , thì một phép Chiếu cuối cùng được yêu cầu:

$$\Pi_{L_1 \cup L_2}(R \bowtie_r S) = \Pi_{L_1 \cup L_2}(\Pi_{L_1 \cup M_1}(R)) \bowtie_r (\Pi_{L_2 \cup M_2}(S))$$

Ví dụ:

$$\begin{aligned} & \Pi_{\text{position}, \text{city}} (\text{Staff} \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} \text{Branch}) = \\ & \Pi_{\text{position}, \text{city}} (\Pi_{\text{position}, \text{branchNo}}(\text{Staff})) \bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo}} (\Pi_{\text{city}, \text{branchNo}}(\text{Branch})) \end{aligned}$$

**(8) Tính giao hoán của Hợp và Giao (nhưng không áp dụng cho Hiệu).**

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

$$R \cap S = S \cap R$$

**(9) Tính giao hoán của phép Chọn và các phép toán tập hợp (Hợp, Giao, và Hiệu).**

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

$$\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$$

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

**(10) Tính giao hoán của phép Chiếu và phép Hợp.**

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

**(11) Tính kết hợp của phép Nối Theta (và tích Đề-các).**

Tích Đề-các và phép nối Natural luôn kết hợp với nhau:

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

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

Nếu điều kiện nối  $q$  chỉ liên quan đến các thuộc tính từ quan hệ  $S$  và  $T$ , thì phép Nối Theta là kết hợp theo cách sau:

$$(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$$

Ví dụ:

$$(Staff \bowtie_{\text{Staff.staffNo}=\text{PropertyForRent.staffNo}} \text{PropertyForRent}) \bowtie_{\text{ownerNo}=\text{Owner.ownerNo} \wedge \text{Staff.lName}=\text{Owner.lName}} \text{Owner} = \text{Staff} \bowtie_{\text{Staff.staffNo}=\text{PropertyForRent.staffNo} \wedge \text{Staff.lName}=\text{Owner.lName}} (\text{PropertyForRent} \bowtie_{\text{ownerNo}} \text{Owner})$$

Lưu ý rằng trong ví dụ này, chỉ đơn giản "di chuyển dấu ngoặc" thôi sẽ không còn chính xác vì sẽ dẫn đến một tham chiếu không xác định được (Staff.lName) trong điều kiện nối giữa PropertyForRent và Owner:

$$\text{PropertyForRent} \bowtie_{\text{PropertyForRent.ownerNo}=\text{Owner.ownerNo} \wedge \text{Staff.lName}=\text{Owner.lName}} \text{Owner}$$

**(12) Tính kết hợp của Hợp và Giao (nhưng không áp dụng cho Hiệu).**

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

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

**Ví dụ 5.3 Sử dụng các quy tắc biến đổi**

*Đối với những khách thuê tiềm năng đang tìm căn hộ (flat), hãy tìm những bất động sản phù hợp với yêu cầu của họ và thuộc chủ sở hữu CO93.*

Chúng ta có thể viết truy vấn này bằng SQL dưới dạng:

```

SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE c.prefType = 'Flat' AND c.clientNo = v.clientNo AND
      v.propertyNo = p.propertyNo AND c.maxRent >= p.rent AND
      c.prefType = p.type AND p.ownerNo = 'CO93';
    
```

Đối với mục đích của ví dụ này, chúng ta sẽ giả định rằng có ít bất động sản thuộc chủ sở hữu CO93 hơn những người thuê tiềm năng đã chỉ định loại bất động sản ưa thích là căn hộ.

Chuyển SQL sang đại số quan hệ, chúng ta có:

$$\Pi_{p.propertyNo, p.street}(\sigma_{c.prefType='Flat'} \wedge c.clientNo=v.clientNo \wedge v.propertyNo=p.propertyNo \wedge c.maxRent \geq p.rent \wedge c.prefType=p.type \wedge p.ownerNo='CO93' ((c \times v) \times p))$$

Chúng ta có thể biểu diễn truy vấn này dưới dạng cây đại số quan hệ chính tắc được hiển thị trong Hình 5.4(a). Bây giờ chúng ta sử dụng các quy tắc biến đổi sau để cải thiện hiệu quả của chiến lược thực thi:

- Quy tắc 1, để tách hội của các phép Chọn thành các phép Chọn riêng lẻ

Quy tắc 2 và Quy tắc 6, để sắp xếp lại các phép Chọn và sau đó giao hoán các phép Chọn và phép tích Đè-các.

Kết quả của hai bước đầu tiên này được thể hiện trong Hình 5.4(b).

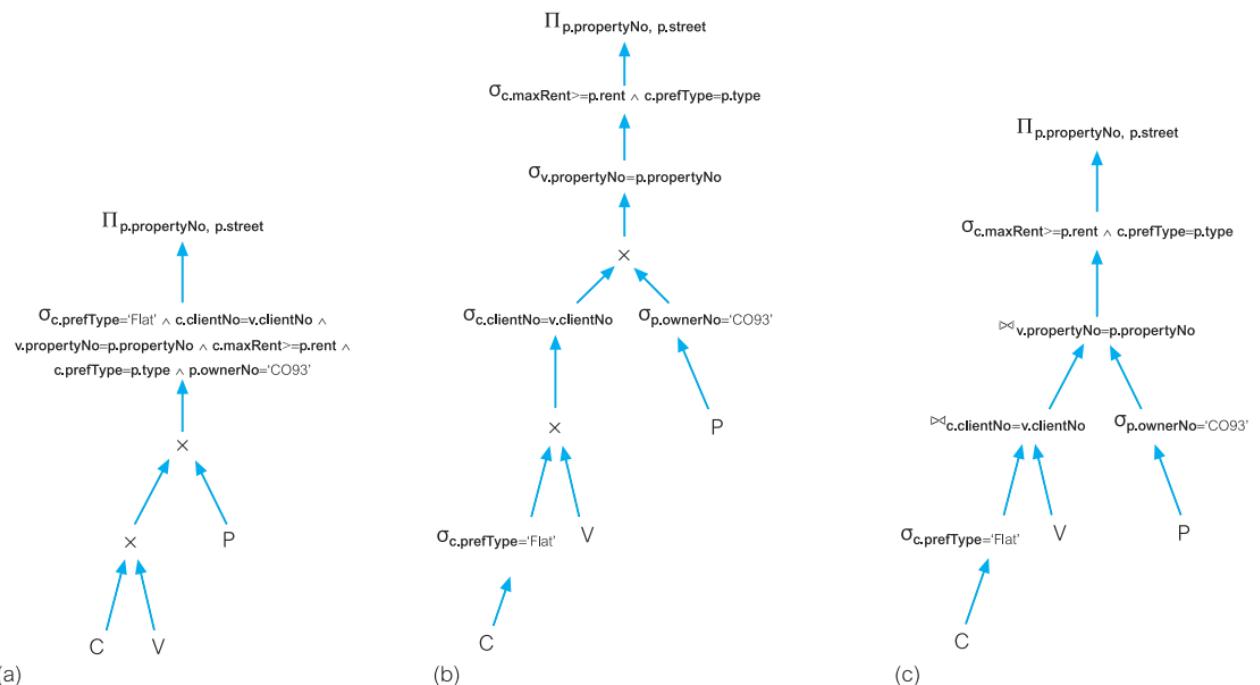
- Chúng ta có thể viết lại một phép Chọn với một vị từ Equijoin và một phép tích Đè-các dưới dạng một phép Nối Equijoin; đó là:

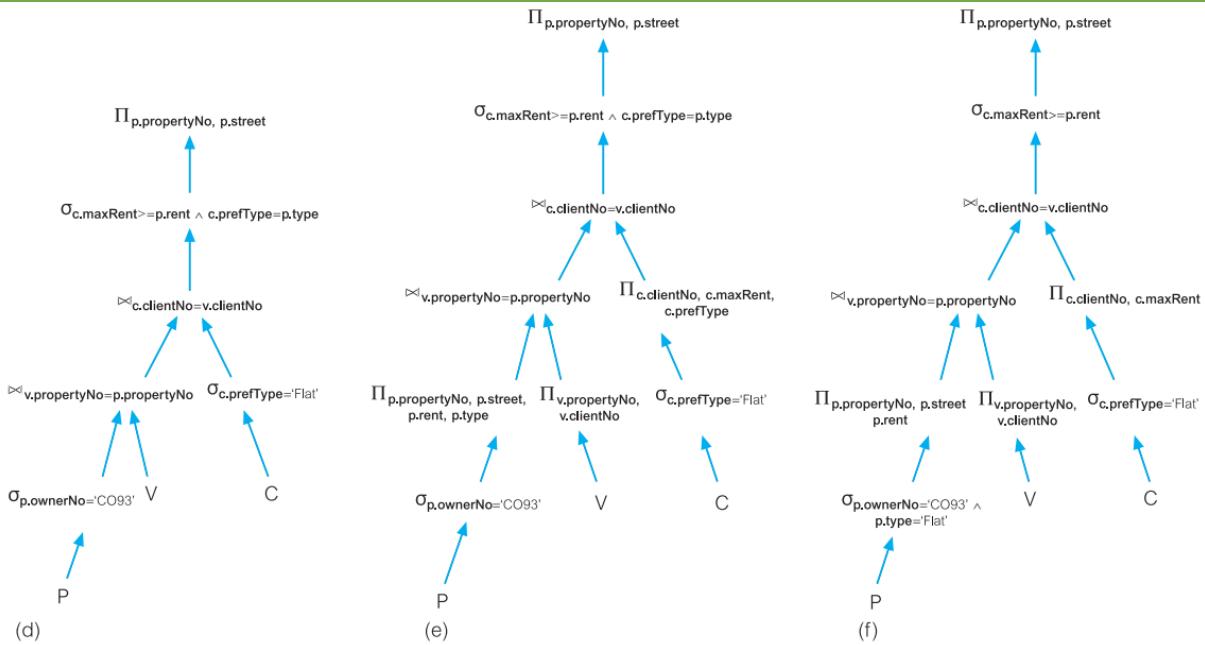
$$\sigma_{R.a=S.b}(R \times S) = R \bowtie_{R.a=S.b} S$$

Kết quả của bước này được thể hiện trong Hình 5.4(c).

- Quy tắc 11, để sắp xếp lại các so sánh bằng, sao cho việc lựa chọn hạn chế hơn, ( $p.ownerNo = 'CO93'$ ) được thực hiện đầu tiên, như thể hiện trong Hình 5.4(d).
- Quy tắc 4 và 7, để di chuyển các phép Chiếu xuống qua các so sánh bằng và tạo các phép Chiếu mới theo yêu cầu. Kết quả của việc áp dụng các quy tắc này được thể hiện trong Hình 5.4(e).

Một tối ưu hóa bổ sung trong ví dụ cụ thể này là phép Chọn ( $c.prefType=p.type$ ) có thể được rút gọn thành ( $p.type = 'Flat'$ ), như chúng ta biết rằng ( $c.prefType = 'Flat'$ ) từ mệnh đề đầu tiên trong vị từ. Sử dụng phép thay thế này, chúng ta đẩy phép Chọn này xuống phía dưới, dẫn đến cây đại số quan hệ rút gọn cuối cùng được thể hiện trong Hình 5.4(f).





**Hình 5.4** Cây đại số quan hệ cho Ví dụ 5.3: (a) cây đại số quan hệ chính tắc; (b) cây đại số quan hệ được hình thành bằng cách đầy các phép Chọn xuống; (c) cây đại số quan hệ được hình thành bằng cách biến đổi các phép Chọn/tích Đè-các thành các phép Nối Equijoin; (d) cây đại số quan hệ được hình thành bằng cách sử dụng tính kết hợp của các Equijoin; (e) cây đại số quan hệ được hình thành bằng cách đầy các phép Chiếu xuống; (f) cây đại số quan hệ rút gọn cuối cùng.

### 5.3.2 Chiến lược xử lý theo phương pháp Heuristic

Nhiều DBMS sử dụng phương pháp heuristic để xác định các chiến lược xử lý truy vấn. Trong phần này, chúng ta xem xét một số phương pháp heuristic tốt có thể được áp dụng trong quá trình xử lý truy vấn.

#### (1) Thực hiện các thao tác Chọn càng sớm càng tốt.

Việc thực hiện phép Chọn làm giảm bản số của quan hệ và giảm quá trình xử lý tiếp theo của quan hệ đó. Do đó, chúng ta nên sử dụng quy tắc 1 để lan truyền (cascade) các phép Chọn và các quy tắc 2, 4, 6 và 9 liên quan đến tính giao hoán giữa phép Chọn với các phép toán đơn phân và nhị phân, để di chuyển các phép Chọn xuống phía dưới của cây càng xa càng tốt. Giữ các vị trí lựa chọn của cùng một quan hệ với nhau.

#### (2) Kết hợp tích Đè-các với một phép Chọn tiếp theo mà vị từ của nó biểu thị một điều kiện nối thành một phép Nối.

Chúng ta đã lưu ý rằng có thể viết lại một phép Chọn với một vị từ Nối Theta và một phép tích Đè-các dưới dạng một phép Nối Theta:

$$\sigma_{R.a \theta S.b}(R \times S) = R \bowtie_{R.a \theta S.b} S$$

#### (3) Sử dụng tính kết hợp của các phép toán nhị phân để sắp xếp lại các nút lá sao cho các nút lá có các phép Chọn hạn chế nhất được thực hiện đầu tiên.

Một lần nữa, nguyên tắc chung của chúng ta là thực hiện giảm càng nhiều càng tốt trước khi thực hiện các phép toán nhị phân. Do đó, nếu chúng ta có hai phép Nối liên tiếp để thực hiện:

$$(R \bowtie_{R.a \theta S.b} S) \bowtie_{S.c \theta T.d} T$$

thì chúng ta nên sử dụng các quy tắc 11 và 12 liên quan đến tính kết hợp của phép nối Theta (và Hợp và Giao) để sắp xếp lại các phép toán sao cho các quan hệ dẫn đến phép nối nhỏ hơn được thực hiện trước, có nghĩa là phép nối thứ hai cũng sẽ dựa trên toán hạng đầu tiên nhỏ hơn.

#### (4) Thực hiện các phép Chiếu càng sớm càng tốt.

Một lần nữa, phép Chiếu làm giảm bản số của quan hệ và giảm quá trình xử lý tiếp theo của quan hệ đó. Do đó, chúng ta nên sử dụng quy tắc 3 để lan truyền các phép Chiếu và các quy tắc 4, 7 và 10 liên quan đến tính giao hoán của phép Chiếu với các phép toán nhị phân, để di chuyển các phép chiếu xuống phía dưới của cây càng xa càng tốt. Giữ các thuộc tính chiếu của cùng một mối quan hệ với nhau.

#### (5) Tính toán các biểu thức chung một lần.

Nếu một biểu thức chung xuất hiện nhiều hơn một lần trong cây và kết quả mà nó tạo ra không quá lớn, hãy lưu trữ kết quả sau khi nó đã được tính toán một lần và sau đó sử dụng lại khi cần thiết. Điều này chỉ có lợi nếu kích thước của kết quả từ biểu thức chung đủ nhỏ để có thể được lưu trữ trong bộ nhớ chính hoặc được truy cập từ bộ nhớ phụ với chi phí thấp hơn so với việc tính toán lại nó. Điều này có thể đặc biệt hữu ích khi truy vấn các khung nhìn, vì cùng một biểu thức phải được sử dụng để tạo khung nhìn mỗi lần được tham chiếu.

### 5.4 UỐC TÍNH CHI PHÍ CÁC PHÉP TOÁN ĐẠI SỐ QUAN HỆ

Một DBMS có thể có nhiều cách khác nhau để hiện thực các phép toán đại số quan hệ. Mục đích của việc tối ưu hóa truy vấn là chọn một truy vấn hiệu quả nhất. Để làm điều này, nó sử dụng các công thức ước tính chi phí cho một số tùy chọn và chọn tùy chọn có chi phí thấp nhất. Trong phần này, chúng ta sẽ tìm hiểu các tùy chọn khác nhau có sẵn để hiện thực các phép toán đại số quan hệ chính. Đối với mỗi tùy chọn, chúng ta cung cấp tổng quan về việc triển khai và đưa ra chi phí ước tính. Vì chi phí chủ yếu trong xử lý truy vấn thường là chi phí truy cập đĩa, chậm hơn so với truy cập bộ nhớ, nên chúng ta tập trung hoàn toàn vào chi phí truy cập đĩa trong các ước tính được cung cấp. Mỗi ước tính biểu diễn cho số lần truy cập khỏi đĩa cần thiết, không bao gồm chi phí ghi quan hệ kết quả.

Nhiều ước tính chi phí dựa trên bản số (cardinality) của quan hệ. Do đó, vì chúng ta cần có thể ước lượng bản số của các quan hệ trung gian, chúng ta cũng chỉ ra một số ước lượng điển hình có thể được suy ra cho các bản số như vậy. Chúng ta bắt đầu phần này bằng cách kiểm tra các loại thống kê mà DBMS sẽ lưu trữ trong danh mục hệ thống để giúp ước tính chi phí.

#### 5.4.1 Thống kê cơ sở dữ liệu

Sự thành công của việc ước tính quy mô và chi phí của các phép toán đại số quan hệ trung gian phụ thuộc vào số lượng và hiện trạng của thông tin thống kê mà DBMS nắm giữ. Thông thường, DBMS sẽ chứa các loại thông tin sau trong danh mục hệ thống.

**Đối với mỗi quan hệ cơ sở R:**

- $nTuples(R)$  - số lượng bộ (bản ghi) trong quan hệ R (nghĩa là bản số của R).
- $bFactor(R)$  - hệ số khói của R (nghĩa là số bộ của R được chứa vừa trong một khói).
- $nBlocks(R)$  - số khói cần thiết để lưu trữ R. Nếu các bộ của R được lưu trữ vật lý cùng nhau, thì:

$$nBlocks(R) = [nTuples(R)/bFactor(R)]$$

ký hiệu  $[x]$  để chỉ ra rằng kết quả của phép tính được làm tròn đến số nguyên nhỏ nhất lớn hơn hoặc bằng chính nó.

**Đối với mỗi thuộc tính A của quan hệ cơ sở R:**

- $nDistinct_A(R)$  - số giá trị khác biệt xuất hiện trong thuộc tính A của quan hệ R.
- $\min_A(R), \max_A(R)$  - các giá trị nhỏ nhất và lớn nhất có thể trong thuộc tính A của quan hệ R.
- $SC_A(R)$  - là số bộ trung bình thỏa mãn điều kiện chọn trên thuộc tính A của quan hệ R. Nếu chúng ta giả sử rằng các giá trị của A được phân phối đồng đều trong R và có tại ít nhất một giá trị thỏa mãn điều kiện, khi đó nếu A là thuộc tính khóa của R thì:

$$SC_A(R) = 1,$$

ngược lại:

$$SC_A(R) = [nTuples(R)/nDistinct_A(R)]$$

**Đối với mỗi chỉ mục đa cấp I trên tập thuộc tính A:**

- $nLevels_A(I)$  - số cấp trong I.
- $nLfBlocks_A(I)$  - số khói lá trong I.

Giữ cho những số liệu thống kê đảm bảo tính tức thời có thể là một vấn đề. Nếu DBMS cập nhật thống kê mỗi khi một bộ dữ liệu được chèn, cập nhật hoặc xóa, thì ở thời điểm cao điểm, sẽ có tác động đáng kể đến hiệu suất. Một phương pháp thay thế và thường được ưu tiên hơn là DBMS cập nhật số liệu thống kê định kỳ, chẳng hạn như hàng đêm hoặc bất cứ khi nào hệ thống không hoạt động. Một cách tiếp cận khác được thực hiện bởi một số hệ thống là người dùng sẽ chỉ định khi nào các số liệu thống kê nên được cập nhật.

**5.4.2 Phép Chọn ( $S = \sigma_p(R)$ )**

Phép Chọn (Selection) trong đại số quan hệ hoạt động trên một quan hệ duy nhất, chẳng hạn như R, và xác định một quan hệ S chỉ chứa các bộ của R thỏa mãn vị trí được chỉ định. Vị trí có thể đơn giản, liên quan đến việc so sánh một thuộc tính của R với một giá trị không đòi hỏi một giá trị thuộc tính khác. Vị trí cũng có thể là hỗn hợp, bao gồm nhiều hơn một điều kiện, với các điều kiện được kết hợp bằng cách sử dụng các liên kết logic  $\wedge$  (AND),  $\vee$  (OR) và  $\sim$  (NOT).

**Bảng 5.1** Tóm tắt chi phí I/O ước tính của các chiến lược cho phép Chọn

Chiến lược	Chi phí
Linear search (unordered file, no index)	$[nBlocks(R)/2]$ với so sánh bằng trên thuộc tính khóa chính, hoặc ngược lại: $nBlocks(R)$
Binary search (ordered file, no index)	$[\log_2(nBlocks(R))]$ với so sánh bằng trên thuộc tính được sắp xếp, hoặc ngược lại: $[\log_2(nBlocks(R))] + [SC_A(R)/bFactory(R)] - 1$
Equality on hash key	1
Equality condition on primary key	$nLevels_A(I) + 1$
Inequality condition on primary key	$nLevels_A(I) + [nBlocks(R)/2]$
Equality condition on clustering (secondary) index;	$nLevels_A(I) + [SC_A(R)/bFactory(R)]$
Equality condition on a non-clustering (secondary) index;	$nLevels_A(I) + [SC_A(R)]$

**5.4.3 Phép Nối ( $T = (R \bowtie_F S)$ )**

Chúng ta đã đề cập ở đầu chương này rằng một trong những mối quan tâm chính khi mô hình quan hệ lần đầu tiên được đưa ra thương mại là hiệu suất của các truy vấn. Đặc biệt, hoạt động được quan tâm nhất là phép Nối (Join), ngoại trừ tích Đề-các, đây là phép toán tốn nhiều thời gian nhất để xử lý và chúng ta phải đảm bảo phép toán này được thực hiện hiệu quả nhất có thể.

Các chiến lược chính để triển khai phép Nối và ước tính chi phí cho từng chiến lược được tóm tắt trong Bảng 5.2:

**Bảng 5.2** Tóm tắt chi phí I/O ước tính của các chiến lược cho phép Nối

Chiến lược	Chi phí
Block nested loop join	$nBlocks(R) + (nBlocks(R) * nBlocks(S))$ nếu chỉ cần một khối bộ đệm cho R và S $nBlocks(R) + [nBlocks(S) * (nBlocks(R)/(nBuffer - 2))]$ nếu cần (nBuffer - 2) khối cho R $nBlocks(R) + nBlocks(S)$ nếu tất cả các khối của R có thể được đọc vào bộ đệm
Indexed nested loop join	Tùy thuộc vào phương thức lập chỉ mục
Sort-merge join	$nBlocks(R)*[\log_2 nBlocks(R)] + nBlocks(S)* [\log_2 nBlocks(S)]$ với các giải thuật sort $nBlocks(R) + nBlocks(S)$ với các giải thuật merge
Hash join	$3(nBlocks(R) + nBlocks(S))$ nếu chỉ mục hash được tổ chức trong bộ nhớ, hoặc ngược lại: $2(nBlocks(R) + nBlocks(S))*[\log_{nBuffer-1} nBlocks(S) - 1] + nBlocks(R) + nBlocks(S)$

#### 5.4.4 Phép Chiếu ( $S = \pi_{A1, A2, \dots, Am}(R)$ )

Phép Chiếu cũng là một phép toán đơn phân xác định một quan hệ S chứa một tập con dọc của một quan hệ R trích xuất các giá trị của các thuộc tính được chỉ định và loại bỏ các bản sao. Do đó, để triển khai phép Chiếu, chúng ta cần các bước sau:

1. loại bỏ các thuộc tính không cần thiết;
2. loại bỏ bất kỳ bộ trùng lặp nào được tạo ra từ bước trên.

Bước thứ hai là bước có vấn đề hơn, mặc dù nó chỉ được yêu cầu nếu các thuộc tính chiếu không bao gồm khóa của quan hệ. Có hai cách tiếp cận chính để loại bỏ các bản sao: sắp xếp và băm.

#### 5.4.5 Các phép toán tập hợp

Các phép toán tập hợp nhị phân Hợp ( $R \cup S$ ), Giao ( $R \cap D S$ ), và Hiệu ( $R - S$ ) chỉ áp dụng cho các quan hệ khả hợp. Chúng ta có thể thực hiện các hoạt động này bằng cách trước tiên sắp xếp cả hai quan hệ trên cùng một thuộc tính và sau đó quét qua từng quan hệ đã sắp xếp một lần để thu được kết quả mong muốn. Trong trường hợp Hợp, chúng ta đặt vào tập kết quả bất kỳ bộ nào xuất hiện trong một trong hai quan hệ ban đầu, loại bỏ các bản sao khi cần thiết. Trong trường hợp Giao, chúng ta đặt vào tập kết quả chỉ các bộ xuất hiện đồng thời trong cả hai quan hệ. Trong trường hợp Hiệu, chúng ta kiểm tra từng bộ của R và chỉ đặt nó vào tập kết quả nếu nó không có trong S. Chi phí ước tính trong mọi trường hợp chỉ đơn giản là:

$$\begin{aligned} nBlocks(R) + nBlocks(S) + nBlocks(R)*[\log_2(nBlocks(R))] \\ + nBlocks(S)*[\log_2(nBlocks(S))] \end{aligned}$$

## TÓM TẮT

- Mục đích của việc xử lý truy vấn là chuyển đổi một truy vấn được viết bằng ngôn ngữ cấp cao, điển hình là SQL, thành một chiến lược thực thi chính xác và hiệu quả được thể hiện bằng ngôn ngữ cấp thấp như đại số quan hệ, và thực thi chiến lược để truy xuất dữ liệu được yêu cầu.
- Vì có nhiều biến đổi tương đương của cùng một truy vấn cấp cao, DBMS phải chọn một biến đổi sử dụng tài nguyên ít nhất. Đây là mục đích của việc tối ưu hóa truy vấn (query optimization).
- Có hai kỹ thuật chính để tối ưu hóa truy vấn, mặc dù hai chiến lược này thường được kết hợp trong thực tế. Kỹ thuật đầu tiên sử dụng các quy tắc heuristic (heuristic rules) để sắp xếp các hoạt động trong một truy vấn. Kỹ thuật còn lại là so sánh các chiến lược khác nhau dựa trên chi phí tương đối của chúng và chọn một chiến lược sử dụng tài nguyên ít nhất.
- Xử lý truy vấn có thể được chia thành bốn giai đoạn chính: phân rã (bao gồm phân tích cú pháp và xác thực), tối ưu hóa, tạo mã và thực thi.
- Phân rã truy vấn (query decomposition) biến đổi một truy vấn cấp cao thành một truy vấn đại số quan hệ và kiểm tra xem truy vấn có đúng về mặt cú pháp và ngữ nghĩa hay không. Các giai đoạn điển hình của phân rã truy vấn là phân tích, chuẩn hóa, phân tích ngữ nghĩa, đơn giản hóa và tái cấu trúc truy vấn. Một cây đại số quan hệ (relational algebra tree) có thể được sử dụng để cung cấp một biểu diễn bên trong của một truy vấn đã biến đổi.
- Tối ưu hóa truy vấn (query optimization) có thể áp dụng các quy tắc biến đổi để chuyển đổi một biểu thức đại số quan hệ thành một biểu thức tương đương được cho là hiệu quả hơn. Các quy tắc biến đổi bao gồm lan truyền phép Chọn, tính giao hoán của các phép toán một ngôi, tính giao hoán của phép nối Theta (và tích Đè-các), tính giao hoán của các phép toán một ngôi và phép nối Theta (và tích Đè-các), và tính kết hợp của phép nối Theta (và tích Đè-các).
- Các quy tắc heuristic (heuristic rules) bao gồm việc thực hiện các phép Chọn và phép Chiếu càng sớm càng tốt; kết hợp tích Đè-các với một phép Chọn tiếp theo mà vị từ biểu thị một điều kiện nối thành một phép Nối; sử dụng tính kết hợp của các phép toán nhị phân để sắp xếp lại các nút lá sao cho các nút lá có các phép Chọn hạn chế nhất được thực hiện đầu tiên.
- Ước tính chi phí (cost estimation) phụ thuộc vào thông tin thống kê có trong danh mục hệ thống. Các số liệu thống kê điển hình bao gồm bản số của mỗi quan hệ cơ sở, số lượng khôi cần thiết để lưu trữ một quan hệ, số lượng các giá trị riêng biệt cho mỗi thuộc tính, số bộ trung bình thỏa mãn điều kiện chọn của mỗi thuộc tính và số lượng cấp trong mỗi chỉ mục đa cấp.

## CÂU HỎI

1. Xử lý truy vấn và tối ưu hóa truy vấn có liên quan với nhau như thế nào?
2. Các giai đoạn điển hình của xử lý truy vấn là gì?
3. Sự khác biệt giữa dạng chuẩn tắc hội và dạng chuẩn tắc tuyển là gì?
4. Nêu các quy tắc biến đổi áp dụng cho:
  - a. phép Chọn
  - b. phép Chiếu
  - c. phép nối Theta
5. Nêu các phương pháp heuristics nên được áp dụng để cải thiện việc xử lý một truy vấn.
6. DBMS nên nắm giữ những loại thông kê nào để có thể lấy được các ước tính của các phép toán đại số quan hệ?
7. Trong những trường hợp nào hệ thống sẽ phải sử dụng đến tìm kiếm tuyến tính khi thực hiện phép toán Lựa chọn?
8. Các chiến lược chính để hiện thực phép Nối là gì?
9. Trình bày sự khác biệt giữa ước tính chi phí và các quy tắc heuristic liên quan đến tối ưu hóa truy vấn.

## PHỤ LỤC

### CÁC CƠ SỞ DỮ LIỆU MẪU

#### A. CƠ SỞ DỮ LIỆU DREAMHOME

##### A.1 Mô tả dữ liệu

###### *Chi nhánh (Branch)*

*DreamHome* có chi nhánh tại các thành phố trên khắp Vương quốc Anh (UK). Mỗi chi nhánh được phân bổ các nhân viên, bao gồm một Trưởng chi nhánh (Manager), người quản lý các hoạt động của nhánh. Dữ liệu mô tả chi nhánh bao gồm số chi nhánh duy nhất, địa chỉ (đường phố, thành phố và mã bưu điện), các số điện thoại (tối đa là ba) và tên của nhân viên hiện đang quản lý chi nhánh (Trưởng chi nhánh). Dữ liệu bổ sung cho mỗi Trưởng chi nhánh, bao gồm ngày đảm nhận vị trí tại chi nhánh và khoản tiền thưởng hàng tháng dựa trên hiệu suất của người đó trong thị trường cho thuê bất động sản.

###### *Nhân viên (Staff)*

Các nhân viên với vai trò Giám sát (Supervisor) chịu trách nhiệm về các hoạt động hàng ngày của một nhóm nhân viên được phân bổ gọi là Trợ lý (Assistant - tối đa 10 người, cùng một lúc). Không phải tất cả các nhân viên đều được chỉ định cho một Giám sát nào đó. Dữ liệu được lưu trữ liên quan đến từng nhân viên bao gồm số nhân viên (duy nhất trên tất cả các chi nhánh), tên (họ và tên), địa chỉ, chức vụ, mức lương, giới tính, ngày sinh (DOB), tên của Giám sát (nếu có) và thông tin chi tiết về chi nhánh mà nhân viên hiện đang làm việc.

###### *Bất động sản cho thuê (PropertyForRent)*

Mỗi chi nhánh cung cấp nhiều loại bất động sản cho thuê. Dữ liệu được lưu trữ cho mỗi bất động sản bao gồm số bất động sản, địa chỉ (đường phố, thành phố, mã bưu điện), loại bất động sản (phòng hoặc căn hộ), số lượng phòng, tiền thuê hàng tháng và thông tin chi tiết về chủ sở hữu của bất động sản đó. Số bất động sản là duy nhất trên tất cả các chi nhánh. Tiền thuê bất động sản hàng tháng được xem xét lại hàng năm. Việc quản lý bất động sản được giao cho một nhân viên bất kỳ khi nào nó được thuê hoặc có nhu cầu cho thuê. Một nhân viên có thể quản lý tối đa 100 bất động sản tại cùng một thời điểm. Khi một bất động sản nào đó có nhu cầu cho thuê, thông tin chi tiết về bất động sản đó sẽ được hiển thị trên trang Web của *DreamHome* và khi cần thiết, sẽ quảng bá trên các tờ báo.

### **Chủ sở hữu Bất động sản (Owner)**

Các thông tin chi tiết về chủ sở hữu bất động sản cũng được lưu trữ. Có hai loại chủ sở hữu chính: tư nhân và doanh nghiệp. Dữ liệu được lưu trữ cho chủ sở hữu cá nhân bao gồm số chủ sở hữu, tên (họ và tên), địa chỉ, số điện thoại, email và mật khẩu. Dữ liệu được lưu trữ cho chủ sở hữu doanh nghiệp bao gồm số chủ sở hữu, tên doanh nghiệp, loại hình doanh nghiệp, địa chỉ, số điện thoại, email, mật khẩu và tên liên hệ.

Mật khẩu sẽ cho phép chủ sở hữu truy cập vào cơ sở dữ liệu *DreamHome* thông qua trang Web.

### **Khách hàng (Client)**

*DreamHome* xem các thành viên của công chúng quan tâm đến việc thuê bất động sản như là khách hàng. Để trở thành khách hàng, trước tiên một người phải đăng ký tại một chi nhánh của *DreamHome*. Dữ liệu được lưu trữ của khách hàng bao gồm số khách hàng, tên (họ và tên, số điện thoại, email, và một số dữ liệu về bất động sản mà họ mong muốn thuê như loại bất động sản ưa thích và giá thuê tối đa mà họ chuẩn bị trả. Cùng được lưu trữ là tên của nhân viên đã xử lý đăng ký, ngày khách hàng tham gia hệ thống và một số thông tin chi tiết về chi nhánh mà khách hàng đã đăng ký. Số khách hàng là duy nhất trên tất cả các chi nhánh của *DreamHome*.

### **Lượt xem bất động sản (Viewing)**

Khách hàng có thể yêu cầu được tham quan bất động sản. Dữ liệu được lưu trữ bao gồm số khách hàng, tên và số điện thoại, số và địa chỉ bất động sản, ngày khách hàng tham quan bất động sản và bất kỳ nhận xét nào của khách hàng về tính phù hợp của bất động sản đó. Khách hàng chỉ có thể xem một bất động sản một lần trong cùng một ngày nhất định.

### **Hợp đồng thuê (Leases)**

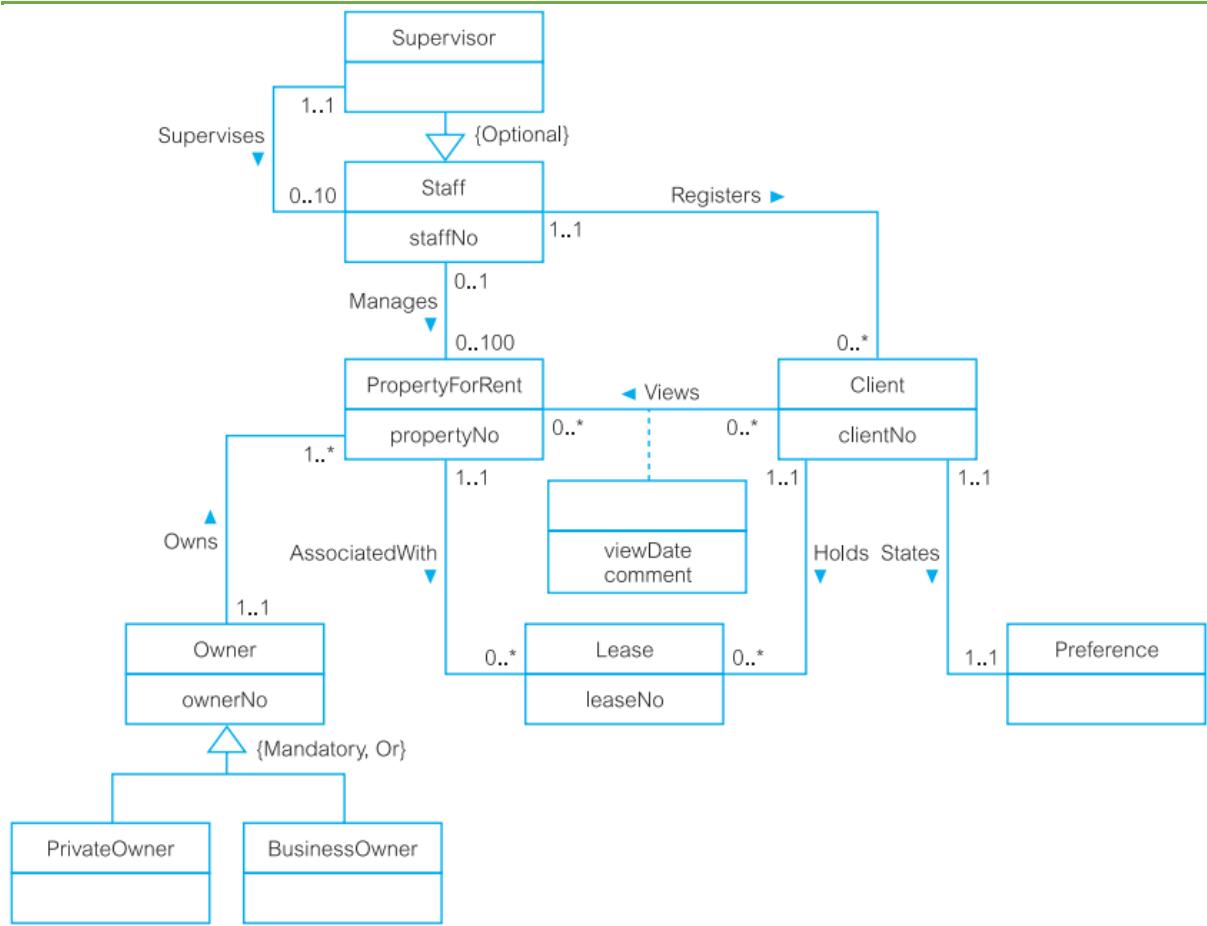
Khi bất động sản được cho thuê, hợp đồng thuê sẽ được ký kết với khách hàng thuê. Dữ liệu được liệt kê chi tiết trên hợp đồng thuê bao gồm số hợp đồng, số khách hàng, tên và địa chỉ khách hàng, số và địa chỉ bất động sản, loại bất động sản và số lượng phòng, tiền thuê hàng tháng, phương thức thanh toán, một dấu hiệu cho biết khoản đặt cọc đã được thanh toán hay chưa (được tính gấp đôi tiền thuê hàng tháng), thời hạn của hợp đồng, ngày bắt đầu và ngày kết thúc thời gian thuê.

Số hợp đồng thuê là duy nhất trên tất cả các chi nhánh của *DreamHome*. Khách hàng có thể ký kết hợp đồng thuê một bất động sản với thời gian tối thiểu ba tháng và tối đa là một năm.

### **Báo (Newspaper)**

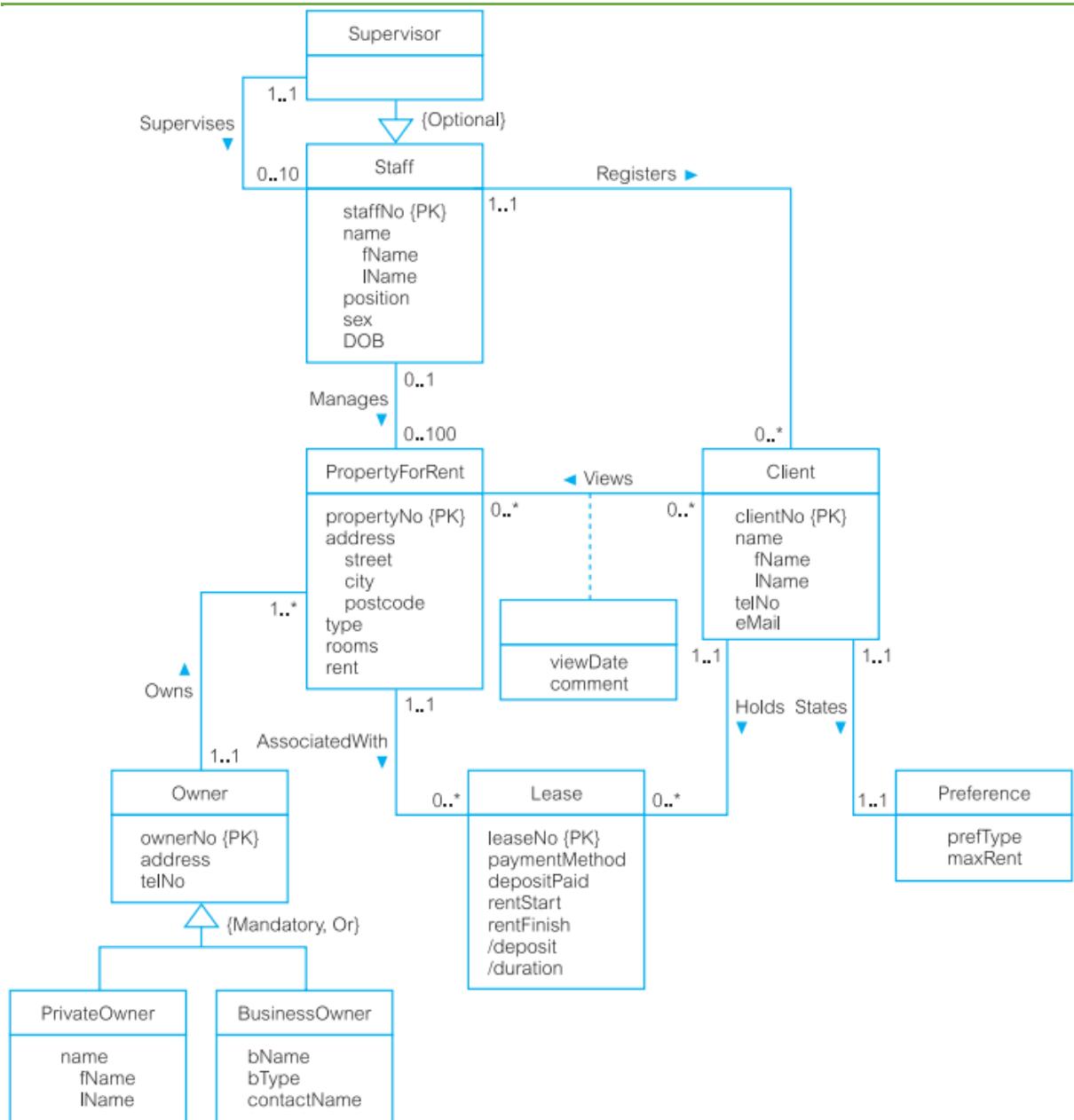
Khi được yêu cầu, thông tin chi tiết về bất động sản cho thuê được quảng bá trên các tờ báo địa phương và quốc gia. Dữ liệu được lưu trữ bao gồm số bất động sản, địa chỉ, loại, số lượng phòng, tiền thuê (theo tháng), ngày được quảng bá, tên tờ báo và chi phí quảng bá. Dữ liệu được lưu trữ cho mỗi tờ báo bao gồm tên tờ báo, địa chỉ, số điện thoại và tên của người đại diện.

## Giáo trình Hệ quản trị Cơ sở dữ liệu



**Hình A.1** Sơ đồ ER của *DreamHome*.

## Giáo trình Hệ quản trị Cơ sở dữ liệu



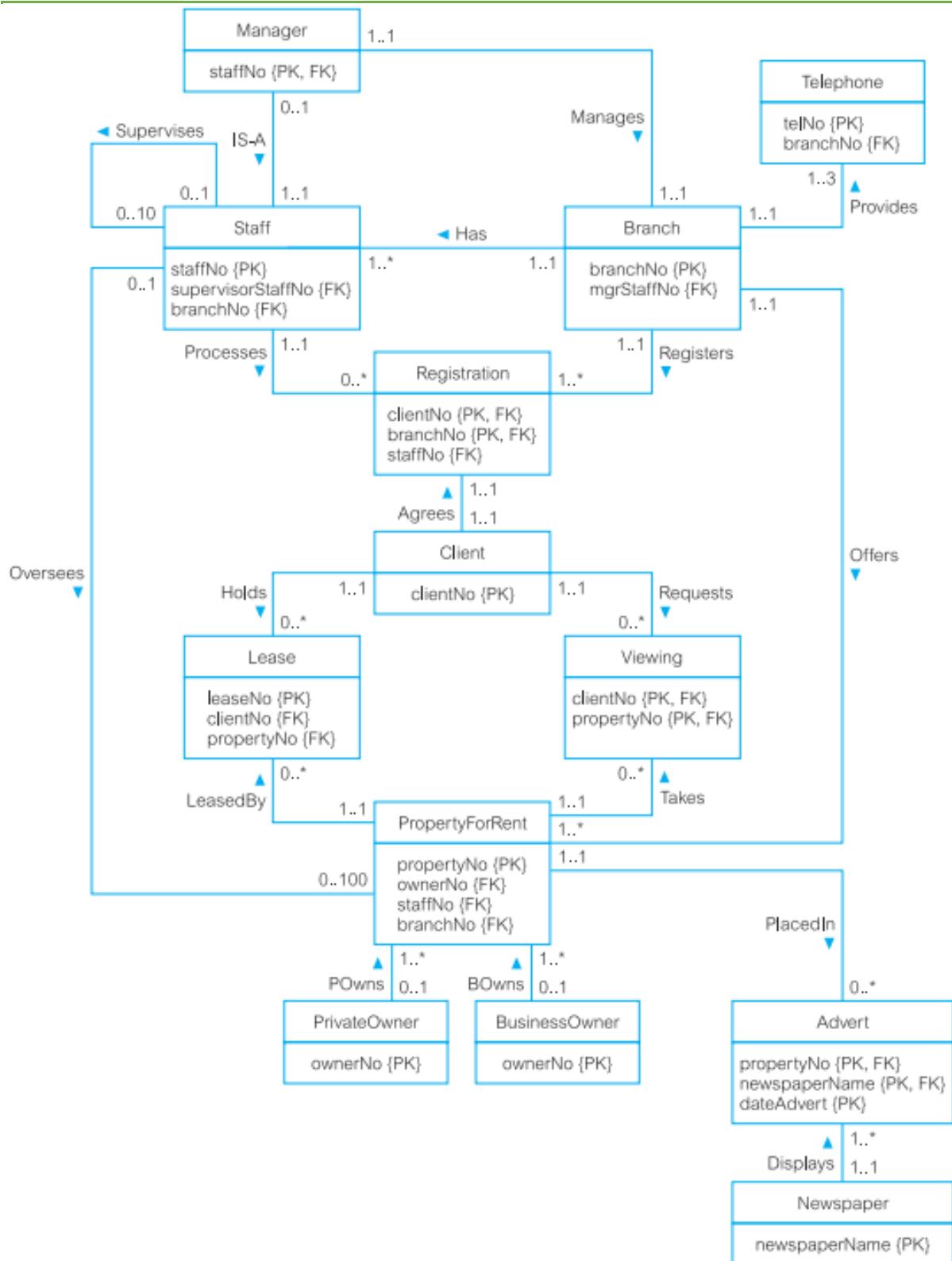
**Hình A.2** Mô hình dữ liệu khái niệm *DreamHome* hiển thị tất cả các thuộc tính.

## Giáo trình Hệ quản trị Cơ sở dữ liệu

<b>Branch</b> (branchNo, street, city, postcode, mgrStaffNo) Primary Key branchNo Alternate Key postcode Foreign Key mgrStaffNo references Manager(staffNo)	<b>Telephone</b> (telNo, branchNo) Primary Key telNo Foreign Key branchNo references Branch(branchNo)
<b>Staff</b> (staffNo, fName, lName, position, sex, DOB, salary, supervisorStaffNo, branchNo) Primary Key staffNo Foreign Key supervisorStaffNo references Staff(staffNo) Foreign Key branchNo references Branch(branchNo)	<b>Manager</b> (staffNo, mgrStartDate, bonus) Primary Key staffNo Foreign Key staffNo references Staff(staffNo)
<b>PrivateOwner</b> (ownerNo, fName, lName, address, telNo) Primary Key ownerNo	<b>BusinessOwner</b> (ownerNo, bName, bType, contactName, address, telNo) Primary Key ownerNo Alternate Key bName Alternate Key telNo
<b>PropertyForRent</b> (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo) Primary Key propertyNo Foreign Key ownerNo references PrivateOwner(ownerNo) and BusinessOwner(ownerNo) Foreign Key staffNo references Staff(staffNo) Foreign Key branchNo references Branch(branchNo)	<b>Viewing</b> (clientNo, propertyNo, dateView, comment) Primary Key clientNo, propertyNo Foreign Key clientNo references Client(clientNo) Foreign Key propertyNo references PropertyForRent(propertyNo)
<b>Client</b> (clientNo, fName, lName, telNo, eMail, prefType, maxRent) Primary Key clientNo Alternate Key eMail	<b>Registration</b> (clientNo, branchNo, staffNo, dateJoined) Primary Key clientNo, branchNo Foreign Key clientNo references Client(clientNo) Foreign Key branchNo references Branch(branchNo) Foreign Key staffNo references Staff(staffNo)
<b>Lease</b> (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo) Primary Key leaseNo Alternate Key propertyNo, rentStart Alternate Key clientNo, rentStart Foreign Key clientNo references Client(clientNo) Foreign Key propertyNo references PropertyForRent(propertyNo) Derived deposit (PropertyForRent.rent*2) Derived duration (rentFinish – rentStart)	<b>Newspaper</b> (newspaperName, address, telNo, contactName) Primary Key newspaperName Alternate Key telNo
<b>Advert</b> (propertyNo, newspaperName, dateAdvert, cost) Primary Key propertyNo, newspaperName, dateAdvert Foreign Key propertyNo references PropertyForRent(propertyNo) Foreign Key newspaperName references Newspaper(newspaperName)	

**Hình A.3** Các quan hệ biểu diễn mô hình dữ liệu logic của *DreamHome*.

## Giáo trình Hệ quản trị Cơ sở dữ liệu



Hình A.4 Sơ đồ quan hệ của *DreamHome*.

## A.2 Giao dịch dữ liệu

### Nhập dữ liệu

1. Nhập thông tin chi tiết một chi nhánh mới (chẳng hạn như chi nhánh B003 ở Glasgow).
2. Nhập thông tin chi tiết một nhân viên mới tại chi nhánh (chẳng hạn như Ann Beech tại chi nhánh B003).
3. Nhập thông tin chi tiết của hợp đồng thuê bất động sản của khách hàng (chẳng hạn như khách hàng Mike Ritchie thuê bất động sản số PG4 từ ngày 10/5/2012 đến ngày 09/5/2013).
4. Nhập thông tin chi tiết của bất động sản được quảng bá trên một tờ báo (chẳng hạn như số bất động sản PG4 được quảng cáo trên tờ Glasgow Daily vào ngày 06/05/2012).
5. Nhập thông tin chi tiết một bất động sản mới và chủ sở hữu (chẳng hạn như chi tiết về số bất động sản PG5 ở Glasgow do Tina Murphy sở hữu).
6. Nhập thông tin chi tiết một khách hàng mới (chẳng hạn như chi tiết về Mike Ritchie).
7. Nhập thông tin chi tiết một lượt tham quan bất động sản của một khách hàng (chẳng hạn như khách hàng Mike Ritchie xem bất động sản PG5 ở Glasgow vào ngày 06/5/2012).

### Cập nhật/Xóa dữ liệu

1. Cập nhật/xóa thông tin chi tiết một chi nhánh.
2. Cập nhật/xóa thông tin chi tiết một nhân viên tại chi nhánh.
3. Cập nhật/xóa thông tin chi tiết của hợp đồng thuê tại một chi nhánh nhất định.
4. Cập nhật/xóa thông tin chi tiết của một quảng bá trên báo tại một chi nhánh nhất định.
5. Cập nhật/xóa thông tin chi tiết của một bất động sản.
6. Cập nhật/xóa thông tin chi tiết của chủ sở hữu bất động sản.
7. Cập nhật/xóa thông tin chi tiết của khách hàng.
8. Cập nhật/xóa thông tin chi tiết một lượt tham quan bất động sản của một khách hàng.

### Truy vấn dữ liệu

Ví dụ về các truy vấn được yêu cầu:

1. Liệt kê chi tiết các chi nhánh trong một thành phố nhất định.
2. Xác định tổng số chi nhánh tại mỗi thành phố.
3. Liệt kê tên, chức vụ và mức lương các nhân viên tại một chi nhánh nhất định, sắp thứ tự theo tên nhân viên.
4. Xác định tổng số nhân viên và tổng tiền lương của họ.

5. Xác định tổng số nhân viên theo từng chức vụ tại các chi nhánh ở Glasgow.
6. Liệt kê tên Trưởng chi nhánh tại mỗi chi nhánh, sắp thứ tự theo địa chỉ chi nhánh.
7. Liệt kê tên các nhân viên được giám sát bởi một Người giám sát nhất định.
8. Liệt kê số bất động sản, địa chỉ, loại và giá thuê hàng tháng của tất cả bất động sản ở Glasgow, được sắp thứ tự theo giá thuê hàng tháng.
9. Liệt kê chi tiết các bất động sản do một nhân viên nhất định quản lý.
10. Xác định tổng số bất động sản được quản lý bởi từng nhân viên tại một chi nhánh nhất định.
11. Liệt kê chi tiết các bất động sản được cung cấp bởi các chủ sở hữu doanh nghiệp tại một chi nhánh nhất định.
12. Xác định tổng số bất động sản theo từng loại tại tất cả các chi nhánh.
13. Xác định thông tin chi tiết chủ sở hữu tư nhân có ít nhất một bất động sản cho thuê.
14. Xác định các căn hộ (flat) có ít nhất ba phòng và giá thuê hàng tháng không cao hơn \$500 ở Aberdeen.
15. Liệt kê số, tên và số điện thoại khách hàng và các quan tâm ưu tiên của họ tại một chi nhánh nhất định.
16. Xác định các bất động sản đã được quảng bá nhiều hơn số lần quảng bá trung bình của tất cả các bất động sản.
17. Liệt kê chi tiết các hợp đồng thuê sẽ hết hạn vào tháng tiếp theo tại một chi nhánh nhất định.
18. Liệt kê tổng số hợp đồng thuê có thời gian thuê dưới một năm tại các chi nhánh ở London.
19. Liệt kê các chi tiết các bất động sản (bao gồm cả tiền đặt cọc thuê) có sẵn để thuê tại chi nhánh nhất định, cùng với các chi tiết của chủ sở hữu.
20. Liệt kê chi tiết các bất động sản được quản lý bởi một nhân viên nhất định tại chi nhánh nhất định.
21. Liệt kê các khách hàng đăng ký tại một chi nhánh nhất định và tên nhân viên đã đăng ký cho khách hàng đó.
22. Xác định các bất động sản ở Glasgow với giá thuê không cao hơn \$450.
23. Xác định tên và số điện thoại của chủ sở hữu một bất động sản nhất định.
24. Liệt kê chi tiết nhận xét của các khách hàng đã tham quan một bất động sản nhất định.
25. Hiển thị tên và số điện thoại của những khách hàng đã tham quan một bất động sản nhất định nhưng không cung cấp nhận xét.
26. Liệt kê chi tiết các bất động sản đã không được thuê trong hơn ba tháng.
27. Tạo danh sách khách hàng có quan tâm ưu tiên phù hợp với một bất động sản

## B. CƠ SỞ DỮ LIỆU UNIVERSITY ACCOMMODATION OFFICE

Giám đốc *University Accommodation Office* có nhu cầu thiết kế một cơ sở dữ liệu để hỗ trợ việc quản lý văn phòng. Giai đoạn thu thập và phân tích yêu cầu của quá trình thiết kế cơ sở dữ liệu đã cung cấp đặc tả yêu cầu dữ liệu sau đây cho cơ sở dữ liệu *University Accommodation Office*, kế tiếp là các ví dụ về các giao dịch dữ liệu cần được cơ sở dữ liệu hỗ trợ.

### B.1 Mô tả dữ liệu

#### Sinh viên (Student)

Dữ liệu được lưu trữ cho mỗi sinh viên chính quy bao gồm: mã số sinh viên, tên (họ và tên), địa chỉ nhà (đường phố, thành phố, mã bưu điện), số điện thoại di động, email, ngày sinh, giới tính, loại sinh viên (ví dụ, đại học năm nhất, sau đại học), quốc tịch, nhu cầu đặc biệt, bất kỳ nhận xét bổ sung nào, tình trạng hiện tại (đã đặt chỗ/chờ), chuyên ngành chính và phụ (nếu có).

Thông tin sinh viên được lưu trữ bao gồm cả những sinh viên hiện đang thuê phòng và những sinh viên trong danh sách chờ. Sinh viên có thể thuê phòng trong ký túc xá hoặc căn hộ dành cho sinh viên.

Khi một sinh viên gia nhập trường đại học, họ được chỉ định cho một nhân viên có vai trò là Cố vấn (Adviser). Cố vấn chịu trách nhiệm theo dõi phúc lợi và tiến trình học tập của học sinh trong suốt thời gian học tại trường đại học. Dữ liệu của Cố vấn bao gồm họ tên, chức vụ, tên khoa, số điện thoại nội bộ, email và số phòng.

#### Ký túc xá (Hall of residence)

Mỗi ký túc xá đều có tên, địa chỉ, số điện thoại và người quản lý giám sát hoạt động của ký túc xá. Các ký túc xá chỉ cung cấp các phòng đơn, có số phòng, số địa điểm và giá thuê hàng tháng.

Số địa điểm xác định duy nhất từng phòng trong tất cả các ký túc xá do Residence Office kiểm soát và được sử dụng khi cho sinh viên thuê phòng.

#### Căn hộ (Student flat)

Residence Office cũng cung cấp các căn hộ cho sinh viên. Các căn hộ này được trang bị đầy đủ tiện nghi và cung cấp chỗ ở trong phòng đơn cho các nhóm ba, bốn hoặc năm sinh viên. Thông tin về các căn hộ dành cho sinh viên bao gồm số căn hộ, địa chỉ và số lượng phòng ngủ đơn có sẵn trong mỗi căn hộ. Số căn hộ xác định duy nhất từng căn hộ.

Mỗi phòng ngủ trong căn hộ đều có giá thuê hàng tháng, số phòng và số địa điểm. Số địa điểm xác định duy nhất từng phòng có sẵn trong tất cả các căn hộ của sinh viên và được sử dụng khi cho sinh viên thuê phòng.

### **Hợp đồng thuê (Leases)**

Sinh viên có thể thuê phòng trong ký túc xá hoặc căn hộ trong nhiều khoảng thời gian khác nhau. Các hợp đồng thuê mới được thương lượng vào đầu mỗi năm học, với thời gian thuê tối thiểu là một học kỳ và thời gian thuê tối đa là một năm, bao gồm học kỳ 1 và học kỳ 2 và học kỳ hè. Mỗi hợp đồng thuê cá nhân giữa sinh viên và Residence Office được xác định duy nhất bằng cách sử dụng một số hợp đồng.

Dữ liệu được lưu trữ của mỗi hợp đồng thuê bao gồm số hợp đồng, thời hạn của hợp đồng thuê (được cho dưới dạng học kỳ), tên sinh viên và mã số sinh viên, số địa điểm, số phòng, chi tiết địa chỉ của ký túc xá hoặc căn hộ và ngày sinh viên muốn nhận phòng, và ngày sinh viên muốn rời khỏi phòng (nếu biết trước).

### **Hóa đơn (Invoice)**

Vào đầu mỗi học kỳ, mỗi sinh viên được gửi một hóa đơn cho thời gian thuê tiếp theo. Mỗi hóa đơn có một số hóa đơn duy nhất.

Dữ liệu được lưu trữ của mỗi hóa đơn bao gồm số hóa đơn, số hợp đồng thuê, học kỳ, thời hạn thanh toán, tên đầy đủ của sinh viên và mã số sinh viên, số địa điểm, số phòng và địa chỉ của ký túc xá hoặc căn hộ. Dữ liệu bổ sung cũng được lưu giữ liên quan đến việc thanh toán hóa đơn bao gồm ngày hóa đơn được thanh toán, phương thức thanh toán (séc, tiền mặt, Visa,...), ngày gửi thông báo đầu tiên và thứ hai (nếu cần).

### **Giám sát nơi cho thuê (Apartment inspection)**

Các căn hộ của sinh viên được nhân viên kiểm tra thường xuyên để đảm bảo rằng chỗ ở được duy trì tốt. Thông tin được ghi lại cho mỗi lần kiểm tra là tên của nhân viên thực hiện việc kiểm tra, ngày kiểm tra, dấu hiệu về việc tài sản được tìm thấy trong tình trạng đạt yêu cầu (có hay không) và bất kỳ nhận xét bổ sung nào.

### **Nhân viên (Staff)**

Một số thông tin cũng được lưu giữ về các nhân viên của Residence Office bao gồm số nhân viên, tên (họ và tên), email, địa chỉ nhà (đường phố, thành phố, mã bưu điện), ngày sinh, giới tính, chức vụ (ví dụ như Quản lý Ký túc xá, Trợ lý Hành chính, Nhân viên tạp vụ) và vị trí (ví dụ như Residence Office hoặc Ký túc xá).

### **Khóa học (Course)**

Residence Office cũng lưu trữ một lượng thông tin hạn chế về các khóa học do trường đại học cung cấp, bao gồm số khóa học, tên khóa học (bao gồm cả năm), người hướng dẫn khóa học, số điện thoại nội bộ, email, số phòng và tên khoa của người hướng dẫn. Mỗi sinh viên cũng được liên kết với một chương trình học duy nhất.

### **Thân nhân (Next-of-kin)**

Bất cứ khi nào có thể, thông tin về người thân của sinh viên sẽ được lưu trữ, bao gồm tên, mối quan hệ, địa chỉ (đường phố, thành phố, mã bưu điện) và số điện thoại liên hệ.

## B.2 Giao dịch dữ liệu

Dưới đây là một số ví dụ về các giao dịch dữ liệu cần được hỗ trợ bởi hệ thống cơ sở dữ liệu University Accommodation Office:

1. Hiển thị một báo cáo liệt kê tên và số điện thoại của Người quản lý cho từng ký túc xá.
2. Hiển thị một báo cáo liệt kê tên và mã số sinh viên của sinh viên với các chi tiết về hợp đồng thuê nhà của họ.
3. Hiển thị chi tiết các hợp đồng thuê bao gồm học kỳ hè.
4. Hiển thị chi tiết về tổng số tiền thuê nhà mà một sinh viên nhất định đã trả.
5. Hiển thị báo cáo về những sinh viên chưa thanh toán hóa đơn trước một ngày nhất định.
6. Hiển thị thông tin chi tiết về việc kiểm tra cẩn hộ nơi tài sản được tìm thấy trong tình trạng không đạt yêu cầu.
7. Hiển thị một báo cáo về tên và mã số sinh viên với số phòng và số nơi ở của họ trong một ký túc xá cụ thể.
8. Hiển thị một báo cáo liệt kê các chi tiết của tất cả các sinh viên hiện đang có trong danh sách chờ đợi để có chỗ ở; đó là; những người không đặt được phòng.
9. Hiển thị tổng số sinh viên trong mỗi loại sinh viên.
10. Trình bày một báo cáo về tên và mã số sinh viên cho tất cả các sinh viên chưa cung cấp thông tin chi tiết về người thân của họ.
11. Hiển thị tên và số điện thoại nội bộ của Cơ quan cho một sinh viên cụ thể.
12. Hiển thị giá thuê tối thiểu, tối đa và trung bình hàng tháng cho các phòng trong các ký túc xá.
13. Hiển thị tổng số vị trí trong mỗi ký túc xá.
14. Hiển thị số nhân viên, tên, tuổi và vị trí hiện tại của tất cả các nhân viên trên 60 tuổi hiện nay.

## C. CƠ SỞ DỮ LIỆU EASYDRIVE SCHOOL OF MOTORING

*EasyDrive School of Motoring* được thành lập tại Glasgow vào năm 1992. Kể từ đó, trường đã phát triển ổn định và hiện có một số văn phòng tại hầu hết các thành phố chính của Scotland. Tuy nhiên, trường hiện nay quá lớn nên ngày càng có nhiều nhân viên hành chính được tuyển dụng để đối phó với số lượng giấy tờ ngày càng nhiều.

Hơn nữa, việc liên lạc và chia sẻ thông tin giữa các văn phòng, ngay cả trong cùng một thành phố, còn chưa hiệu quả. Giám đốc của trường, Dave MacLeod, cảm thấy rằng có quá nhiều sai lầm đang mắc phải và sự thành công của trường sẽ rất ngắn nếu không có giải pháp để khắc phục tình hình. Ông ấy biết rằng cơ sở dữ liệu có thể giúp giải quyết một phần vấn đề và đã tiếp cận bạn và nhóm của bạn để nhờ giúp tạo một hệ thống cơ sở dữ liệu hỗ trợ việc vận hành *EasyDrive School of Motoring*. Giám đốc đã cung cấp mô tả ngắn gọn sau đây về cách hoạt động của trường.

### C.1 Mô tả dữ liệu

Mỗi văn phòng có một Quản lý (người cũng có xu hướng là một Giáo viên cao cấp), một số Giáo viên cao cấp, Giáo viên và nhân viên hành chính. Quản lý chịu trách nhiệm về hoạt động hàng ngày của văn phòng. Trước tiên, khách hàng phải đăng ký tại một văn phòng, bao gồm việc hoàn thành mẫu đơn đăng ký, trong đó ghi lại các chi tiết cá nhân của họ. Trước buổi học đầu tiên, khách hàng được yêu cầu tham gia một cuộc phỏng vấn với Giáo viên để đánh giá nhu cầu của khách hàng và để đảm bảo rằng khách hàng có bằng lái xe tạm thời hợp lệ.

Khách hàng có thể tự do yêu cầu một Giáo viên cụ thể hoặc yêu cầu thay đổi Giáo viên ở bất kỳ giai đoạn nào trong suốt quá trình học lái xe. Sau khi phỏng vấn, buổi học đầu tiên được đặt trước. Khách hàng có thể yêu cầu các bài học riêng lẻ hoặc đặt trước một khối bài học với một khoản phí giảm. Một giờ học cá nhân kéo dài một giờ, bắt đầu và kết thúc tại văn phòng. Một bài học với một Giáo viên cụ thể trên một chiếc ô tô cụ thể tại một thời điểm nhất định. Các bài học có thể bắt đầu sớm nhất là 8 giờ sáng và muộn nhất là 8 giờ tối.

Sau mỗi bài học, Giáo viên ghi lại sự tiến bộ của khách hàng và quãng đường đã sử dụng trong suốt bài học. Trường có một số xe ô tô, được trang bị cho mục đích giảng dạy. Mỗi Giáo viên được phân bổ cho một chiếc xe cụ thể. Cũng như việc giảng dạy, Giáo viên được tự do sử dụng xe ô tô cho mục đích cá nhân. Những chiếc xe được kiểm tra định kỳ để phát hiện lỗi. Sau khi đã sẵn sàng, khách hàng sẽ đăng ký ngày sát hạch. Để có được giấy phép lái xe, khách hàng phải vượt qua cả phần lái xe (thực hành) và phần viết (lý thuyết) của bài sát hạch. Giáo viên có trách nhiệm đảm bảo rằng khách hàng được chuẩn bị tốt nhất cho tất cả các khía cạnh của bài sát hạch. Giáo viên không chịu trách nhiệm về bài sát hạch của khách hàng và không có mặt trong xe trong quá trình sát hạch, nhưng phải có mặt để đưa và đón khách hàng trước và sau khi sát hạch tại Testing Center. Nếu khách hàng không đạt, Giáo viên phải ghi lại lý do không đạt.

## C.2 Giao dịch dữ liệu

Sau đây là một số ví dụ về các truy vấn điển hình mà hệ thống cơ sở dữ liệu cho EasyDrive School of Motoring phải hỗ trợ:

1. Tên và số điện thoại của các Quản lý của từng văn phòng.
2. Địa chỉ đầy đủ của tất cả các văn phòng ở Glasgow.
3. Tên của tất cả các Giáo viên nữ có trụ sở tại văn phòng Bearsden, Glasgow.
4. Tổng số nhân viên tại mỗi văn phòng.
5. Tổng số khách hàng (trong quá khứ và hiện tại) ở mỗi thành phố.
6. Thời gian biểu của các cuộc hẹn cho một Giáo viên nhất định vào tuần tới.
7. Chi tiết các cuộc phỏng vấn được thực hiện bởi một Giáo viên nhất định.
8. Tổng số khách hàng nữ và nam (trước đây và hiện tại) tại văn phòng Bearsden, Glasgow.
9. Số lượng và tên của nhân viên là Giáo viên và trên 55 tuổi.
10. Số đăng ký của các chiếc xe không bị lỗi gì.
11. Số đăng ký của những chiếc xe được sử dụng bởi Giáo viên tại văn phòng Bearsden, Glasgow.
12. Tên của những khách hàng đã vượt qua kỳ thi sát hạch lái xe vào tháng 1 năm 2013.
13. Tên của những khách hàng đã thi sát hạch lái xe nhiều hơn ba lần mà vẫn không đậu.
14. Số đặm trung bình đã lái trong một giờ học kéo dài một giờ,
15. Số lượng nhân viên hành chính đặt tại mỗi văn phòng.

## TÀI LIỆU THAM KHẢO

1. Dương Tuấn Anh, Nguyễn Trung Trực (2020), *Hệ Cơ sở dữ liệu*, NXB Đại học Quốc gia TP. Hồ Chí Minh, Hồ Chí Minh.
2. Đỗ Ngọc Sơn, Phan Văn Viên, Nguyễn Phương Nga (2015), *Giáo trình Hệ quản trị cơ sở dữ liệu*, NXB Khoa học và Kỹ thuật, Hà Nội.
3. G K Gupta (2018), *Database Management Systems*, McGraw-Hill Education, India.
4. Mark L. Gillenson (2011), *Fundamentals of Database Management Systems*, Wiley, United States of America.
5. Thomas M. Connolly, Carolyn E. Begg (2015), *Database Systems A Practical Approach to Design, Implementation, and Management*, Pearson, United States of America.