CHƯƠNG II

SQL

MỤC ĐÍCH

Giới thiệu một hệ CSDL chuẩn, SQL, các thành phần cơ bản của của nó.

YÊU CẦU

Hiểu các thành phần cơ bản của SQL-92

Hiểu và vận dụng phương pháp "dịch" từ câu vấn tin trong ngôn ngữ tự nhiên sang ngôn ngữ SQL và ngược lại

Hiểu và vận dụng cách thêm (xen), xóa dữ liệu

SQL là ngôn ngữ CSDL quan hệ chuẩn, gốc của nó được gọi là Sequel. SQL là viết tắt của **S**tructured **Q**uery **L**anguage. Có nhiều phiên bản của SQL. Phiên bản được trình bày trong giáo trình này là phiên bản chuẩn SQL-92.

SQL có các phần sau:

- ***Ngôn ngữ định nghĩa dữ liệu (DDL)***. **DDL** của SQL cung cấp các lệnh để định nghĩa các sơ đồ quan hệ, xoá các quan hệ, tạo các chỉ mục, sủa đổi các sơ đồ quan hệ
- ***Ngôn ngữ thao tác dữ liệu tương tác (Interactive DML)*****. IDML** bao gồm một ngôn ngữ dựa trên cả đại số quan hệ lẫn phép tính quan hệ bộ. Nó bao hàm các lệnh xen các bộ, xoá các bộ, sửa đổi các bộ trong CSDL
- ***Ngôn ngữ thao tác dữ liệu nhúng (Embedded DML)*****.** Dạng SQL nhúng được thiết kế cho việc sử dụng bên trong các ngôn ngữ lập trình mục đích chung (genaral-purpose programming languages) như PL/I, Cobol, Pascal, Fortran, C.
- ***Đinh nghĩa view***. **DDL SQL** cũng bao hàm các lệnh để định nghĩa các view.
- ***Cấp quyền (Authorization)*****. DDL SQL** bao hàm cả các lệnh để xác định các quyền truy xuất dến các quan hệ và các view
- ***Tính toàn vẹn (Integrity)*****. DDL SQL** chứa các lệnh để xác định các ràng buộc toàn vẹn mà dữ liệu được lưu trữ trong CSDL phải thoả.
- ***Điều khiển giao dịch*****.** SQL chứa các lệnh để xác định bắt đầu và kết thúc giao dịch, cũng cho phép chốt tường minh dữ liệu để điều khiển cạnh tranh

Các ví dụ minh hoạ cho các câu lệnh SQL được thực hiện trên các sơ đồ quan hệ sau:

- **Branch_schema = (Branch_name, Branch_city, Assets)**: Sơ đồ quan hệ chi nhánh nhà băng gồm các thuộc tính Tên chi nhánh (Branch_name), Thành phố (Branch_city), tài sản (Assets)
- **Customer_schema = (Customer_name, Customer_street, Customer_city)**: Sơ đồ quan hệ Khách hàng gồm các thuộc tính Tên khách hàng (Customer_name), phố (Customer_street), thành phố (Customer_city)
- **Loan_schema = (Branch_name, loan_number, amount)**: Sơ đồ quan hệ cho vay gồm các thuộc tính Tên chi nhánh, số cho vay (Loan_number), số lượng (Amount)
- **Borrower_schema = (Customer_name, loan_number)**: Sơ đồ quan hệ người mượn gồm các thuộc tính Tên khách hàng, số cho vay
- **Account_schema = (Branch_name, account_number, balance)**: Sơ đồ quan hệ tài khoản gồm các thuộc tính Tên chi nhánh, số tài khoản (Account_number), số cân đối (Balance: dư nợ/có)
- **Depositor_schema = (Customer_name, account_number)**: Sơ đồ người gửi gồm các thuộc tính Tên khách hàng, số tài khoản

Cấu trúc cơ sở của một biểu thức SQL gồm ba mệnh đề: **SELECT, FROM và WHERE**

- ♦ Mệnh đề **SELECT** tương ứng với phép chiếu trong đại số quan hệ, nó được sử dụng để liệt kê các thuộc tính mong muốn trong kết quả của một câu vấn tin
- ♦ Mệnh đề **FROM** tương ứng với phép tích Đề các , nó nó liệt kê các quan hệ được quét qua trong sự định trị biểu thức
- ♦ Mệnh đề **WHERE** tương ứng với vị từ chọn lọc, nó gồm một vị từ chứa các thuộc tính của các quan hệ xuất hiện sau FROM

Một câu vấn tin kiểu mẫu có dạng:

SELECT A_1, A_2, ..., A_k
FROM R_1, R_2, ..., R_m
WHERE P

trong đó A_i là các thuộc tính (Attribute), R_j là các quan hệ (Relation) và P là một vị từ (Predicate). Nếu thiếu WHERE vị từ P là TRUE.

Kết quả của một câu vấn tin SQL là một quan hệ.

MỆNH ĐỀ SELECT

Ta tìm hiểu mệnh đề SELECT bằng cách xét một vài ví dụ:

"Tìm kiếm tất cả các tên các chi nhánh trong quan hệ cho vay (loan)":

SELECT Branch_name
FROM Loan;

Kết quả là một quan hệ gồm một thuộc tính Tên chi nhánh (Branch_name)

Nếu muốn quan hệ kết quả không chứa các tên chi nhánh trùng nhau:

SELECT **DISTINCT** Branch_name
FROME Loan;

Từ khoá ALL được sử dụng để xác định tường minh rằng các giá trị trùng không bị xoá và nó là mặc nhiên của mệnh đề SELECT.

Ký tự * được dùng để chỉ tất cả các thuộc tính:

SELECT *
FROM Loan;

Sau mệnh đề SELECT cho phép các biểu thức số học gồm các phép toán +, -, *, / trên các hằng hoặc các thuộc tính:

```
SELECT Branch_name, Loan_number, amount * 100
FROM Loan;
```

MỆNH ĐỀ WHERE

"Tìm tất cả các số cho vay ở chi nhánh tên Perryridge với số lượng vay lớn hơn1200$"

```
SELECT Loan_number
FROM Loan
WHERE Branch_name = 'Perryridge' AND Amount > 1200;
```

SQL sử dụng các phép nối logic: **NOT, AND, OR**. Các toán hạng của các phép nối logic có thể là các biểu thức chứa các toán tử so sánh =, >=, <>, <, <=.

Toán tử so sánh **BETWEEN** được dùng để chỉ các giá trị nằm trong một khoảng:

```
SELECT Loan_number
FROM Loan
WHERE Amount  BETWEEN 50000 AND 100000;
```

≈

```
SELECT  Loan_number
FROM Loan
WHERE Amount >= 50000 AND Amount <= 100000;
```

Ta cũng có thể sử dụng toán tử **NOT BETWEEN**.

MỆNH ĐỀ FROM

"Trong tất cả các khách hàng có vay ngân hàng tìm tên và số cho vay của họ"

```
SELECT DISTINCT Customer_name, Borrower.Loan_number
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number;
```

SQL sử dụng cách viết <tên quan hệ >.< tên thuộc tính > để che dấu tính lập lờ trong trường hợp tên thuộc tính trong các sơ đồ quan hệ trùng nhau.

"Tìm các tên và số cho vay của tất cả các khách hàng có vay ở chi nhánh Perryridge"

```
SELECT Customer_name, Borrower.Loan_number
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number AND
        Branch_name ='Perryridge';
```

CÁC PHÉP ĐỔI TÊN

SQL cung cấp một cơ chế đổi tên cả tên quan hệ lẫn tên thuộc tính bằng mệnh đề dạng:

< tên cũ > **AS** < tên mới >

mà nó có thể xuất hiện trong cả mệnh đề SELECT lẫn FROM

```
SELECT DISTINCT Customer_name, Borrower.Loan_number
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number AND
        Branch_name ='Perryridge';
```

Kết quả của câu vấn tin này là một quan hệ hai thuộc tính: Customer_name, Loan_number

Đổi tên thuộc tính của quan hệ kết quả:

```
SELECT Customer_name, Borrower.Loan_number AS Loan_Id
FROM Borrower, Loan
```

```
        WHERE Borrower.Loan_number = Loan.Loan_number AND
              Branch_name =’Perryridge’;
```

CÁC BIẾN BỘ (Tuple Variables)

Các biến bộ được định nghĩa trong mệnh đề FROM thông qua sử dụng mệnh đề AS:

```
SELECT DISTINCT Customer_name, T.Loan_number
FROM Borrower  AS  T, Loan  AS  S
WHERE T.Loan_number = S.Loan_number AND
      Branch_name =’Perryridge’;
```

“Tìm các tên của tất cả các chi nhánh có tài sản lớn hơn ít nhất một chi nhánh ở Brooklyn“

```
SELECT DISTINCT T.branch_name
FROM  Branch AS T, Banch AS S
WHERE T.assets > S.assets AND S.Branch_City = ‘Brooklyn’
```

SQL92 cho phép sử dụng các viết (v_1, v_2, ..., vn) để ký hiệu một n-bộ với các giá trị v_1, v_2, ..., vn. Các toán tử so sánh có thể được sử dụng trên các n-bộ và theo thứ tự tự điển. Ví dụ (a1, b1) <= (a2, b2) là đúng nếu ($a_1 < b_1$) OR (($a_1 = b_1$) AND ($a_2 < b_2$)).

CÁC PHÉP TOÁN TRÊN CHUỖI

Các phép toán thường được dùng nhất trên các chuỗi là phép đối chiếu mẫu sử dụng toán tử **LIKE**. Ta mô tả các mẫu dùng hai ký tự đặc biệt:

- ký tự phần trăm (%): ký tự % tương xứng với ***chuỗi con*** bất kỳ
- ký tự gạch nối (_): ký tự gạch nối tương xứng với ***ký tự*** bất kỳ.
  - ‘Perry%’ tương xứng với bất kỳ chuỗi nào bắt đầu bởi ‘Perry’
  - ‘%idge%’ tương xứng với bất kỳ chuỗi nào chứa ‘idge’ như chuỗi con
  - ‘___’ tương xứng với chuỗi bất kỳ có đúng ba ký tự
  - ‘___%’ tương xứng với chuỗi bất kỳ có ít nhất ba ký tự

"Tìm tên của tất cả các khách hàng tên phố của họ chứa chuỗi con ‘Main’

```
SELECT Customer_name
FROM Customer
WHERE Customer_street LIKE ‘%Main%’
```

Nếu trong chuỗi mẫu có chứa các ký tự % _ \ , để tránh nhầm lẫn ký tự với"dấu hiệu thay thế", SQL sử dụng cách viết: ký tự escape (\) đứng ngay trước ký tự"đặc biệt". Ví dụ nếu chuỗi mẫu là ab%cd được viết là ‘ab\%cd’, chuỗi mẫu là ab_cde được viết là ‘ab_cde’, chuỗi mẫu là ab\cd được viết là ‘ab\\cd’

SQL cho phép đối chiếu không tương xứng bằng cách sử dụng **NOT LIKE**

SQL cũng cho phép các hàm trên chuỗi: nối hai chuỗi (|), trích ra một chuỗi con, tìm độ dài chuỗi, biến đổi một chuỗi chữ thường sang chuỗi chữ hoa và ngược lại ...

THỨ TỰ TRÌNH BÀY CÁC BỘ (dòng)

Mệnh đề ORDER BY tạo ra sự trình bày các dòng kết quả của một câu vấn tin theo một trình tự. Để liết kê theo thứ tự alphabet tất cả các khách hàng có vay ở chi nhánh Perryridge:

```
SELECT  DISTINCT  Customer_name
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number AND
            Branch_name = ‘Perryridge’
ORDER BY Customer_name;
```

Mặc nhiên, mệnh đề ORDER BY liệt kê theo thứ tự tăng, tuy nhiên ta có thể làm liệt kê theo thứ tự **giảm**/**tăng** bằng cách chỉ rõ bởi từ khoá **DESC**/ **ASC**

```
SELECT *
FROM Loan
ORDER BY Amount DESC, Loan_number ASC;
```

CÁC PHÉP TOÁN TẬP HỢP

SQL92 có các phép toán **UNION, INTERSECT, EXCEPT** chúng hoạt động giống như các phép toán hợp, giao, hiệu trong đại số quan hệ. Các quan hệ tham gia vào các phép toán này phải tương thích (có cùng tập các thuộc tính).

- Phép toán **UNION**

"tìm kiếm tất cả các khách hàng có vay, có tài khoản hoặc cả hai ở ngân hàng"

```
(SELECT  Customer_name
 FROM Depositor)
UNION
(SELECT Customer_name
 FROM Borrower);
```

Phép toán hợp UNION tự động loại bỏ các bộ trùng, nếu ta muốn giữ lại các bộ trùng ta phải sử dụng **UNION ALL**

```
(SELECT  Customer_name
 FROM Depositor)
UNION ALL
(SELECT Customer_name
 FROM Borrower);
```

- Phép toán **INTERSECT**

"tìm kiếm tất cả các khách hàng có vay và cả một tài khoản tại ngân hàng"

```
(SELECT DISTINCT Customer_name
 FROM Depositor)
INTERSECT
(SELECT DISTINCT Customer_name
 FROM Borrower);
```

Phép toán INTERESCT tự động loại bỏ các bộ trùng, Để giữ lại các bộ trùng ta sử dụng **INTERSECT ALL**

```
(SELECT  Customer_name
 FROM Depositor)
INTERSECT ALL
(SELECT  Customer_name     FROM Borrower);
```

- Phép toán **EXCEPT**

"Tìm kiếm tất cả các khách hàng có tài khoản nhưng không có vay tại ngân hàng"

```
(SELECT Customer_name
 FROM Depositor)
EXCEPT
(SELECT Customer_name
 FROM Borrower);
```

EXCEPT tự động loại bỏ các bộ trùng, nếu muốn giữ lại các bộ trùng phải dùng **EXCEPT ALL**

```
(SELECT Customer_name
 FROM Depositor)
EXCEPT ALL
```

```
(SELECT Customer_name
 FROM Borrower);
```

CÁC HÀM TÍNH GỘP

SQL có các hàm tính gộp (aggregate functions):

- Tính trung bình (Average): **AVG**()
- Tính min : **MIN**()
- Tính max: **MAX**()
- Tính tổng: **SUM**()
- Đếm: **COUNT**()

Đối số của các hàm AVG và SUM phải là **kiểu dữ liệu số**

"Tìm số cân đối tài khoản trung bình tại chi nhánh Perryridge"

```
SELECT AGV(balace)
FROM Account
WHERE Branch_name = 'Perryridge';
```

SQL sử dụng mệnh để **GROUP BY** vào mục đích nhóm các bộ có cùng giá trị trên các thuộc tính nào đó

"Tìm số cân đối tài khoản trung bình tại mỗi chi nhánh ngân hàng"

```
SELECT Branch_name, AVG(balance)
FROM Account
GROUP BY Branch_name;
```

"Tìm số các người gửi tiền đối với mỗi chi nhánh ngân hàng"

```
SELECT Branch_name, COUNT(DISTINCT Customer_name)
FROM Depositor, Account
WHERE Depositor.Account_number = Account.Acount_number
GROUP BY Branch_name
```

Giả sử ta muốn liệt kê các chi nhánh ngân hàng có số cân đối trung bình lớn hơn 1200$. Điều kiện này không áp dụng trên từng bộ, nó áp dụng trên từng nhóm. Để thực hiện được điều này ta sử dụng mệnh đề **HAVING** của SQL

```
SELECT Branch_name, AVG(balance)
FROM Account
GROUP BY Branch_name
HAVING AGV(Balance) > 1200$;
```

Vị từ trong mệnh đề HAVING được áp dụng sau khi tạo nhóm, như vậy hàm AVG có thể được sử dụng

"Tìm số cân đối đối với tất cả các tài khoản"

```
SELECT AVG(Balance)   FROM Account;
```

"Đếm số bộ trong quan hệ Customer"

```
SELECT Count(*)     FROM Customer;
```

SQL không cho phép sử dụng **DISTINCT** với **COUNT(*)**, nhưng cho phép sử dụng DISTINCT với MIN và MAX.

Nếu WHERE và HAVING có trong cùng một câu vấn tin, vị từ sau WHERE được áp dụng trước. Các bộ thoả mãn vị từ WHERE được xếp vào trong nhóm bởi GROUP BY, mệnh đề HAVING (nếu có) khi đó được áp dụng trên mỗi nhóm. Các nhóm không thoả mãn mệnh đề HAVING sẽ bị xoá bỏ.

"Tìm số cân đối trung bình đối với mỗi khách hàng sống ở Harrison và có ít nhất ba tài khoản"

```
SELECT Depositor.Customer_name, AVG(Balance)
```

```
FROM Depositor, Account, Customer
WHERE Depositor.Account_number = Account.Account_number AND
      Depositor.Customer_name = Customer.Customer_name AND
      Customer.city ='Harrison'
GROUP BY Depositor.Customer_name
HAVING COUNT(DISTINT Depositor.Account_number) >= 3;
```

CÁC GIÁ TRỊ NULL

SQL cho phép sử dụng các giá trị null để chỉ sự vắng mặt thông tin tạm thời về giá trị của một thuộc tính. Ta có thể sử dụng từ khoá đặc biệt **null** trong vị từ để thử một giá trị null.

"Tìm tìm tất cả các số vay trong quan hệ Loan với giá trị Amount là null"

```
SELECT Loan_number
FROM Loan
WHERE Amount is null
```

Vị từ **not null** thử các giá trị không rỗng

Sử dụng giá trị null trong các biểu thức số học và các biểu thức so sánh gây ra một số phiền phức. Kết quả của một biểu thức số học là null nếu một giá trị input bất kỳ là null. Kết quả của một biểu thức so sánh chứa một giá trị null có thể được xem là false. SQL92 xử lý kết quả của một phép so sánh như vậy như là một giá trị **unknown**, là một giá trị không là **true** mà cũng không là **false**. SQL92 cũng cho phép thử kết quả của một phép so sánh là unknown hay không. Tuy nhiên, trong hầu khắp các trường hợp, unknown được xử lý hoàn toàn giống như false.

Sự tồn tại của các giá trị null cũng làm phức tạp việc sử lý các toán tử tính gộp. Giả sử một vài bộ trong quan hệ Loan có các giá trị null trên trường Amount. Ta xét câu vấn tin sau:

```
SELECT SUM(Amount)
FROM LOAN
```

Các giá trị được lấy tổng trong câu vấn tin bao hàm cả các trị null. Thay vì tổng là null, SQL chuẩn thực hiện phép tính tổng bằng cách bỏ qua các giá trị input là null.

Nói chung, các hàm tính gộp tuân theo các quy tắc sau khi xử lý các giá trị null: Tất cả các hàm tính gộp ngoại trừ COUNT(*) bỏ qua các giá trị input null. Khi các giá trị nul bị bỏ qua, tập các giá trị input có thể là rõng. COUNT() của một tập rỗng được định nghĩa là 0. Tất cả các hàm tính gộp khác trả lại giá trị null khi áp dụng trên tập hợp input rỗng.

CÁC CÂU VẤN TIN CON LỒNG NHAU (Nested Subqueries)

SQL cung cấp một cơ chế lòng nhau của các câu vấn tin con. Một câu vấn tin con là một biểu thức SELECT-FROM-WHERE được lồng trong một caau vấn tin khác. Các câu vấn tin con thường được sử dụng để thử quan hệ thành viên tập hợp, so sánh tập hợp và bản số tập hợp.

QUAN HỆ THÀNH VIÊN TẬP HỢP (Set relationship)

SQL đưa vào các phép tính quan hệ các phép toán cho phép thử các bộ có thuộc một quan hệ nào đó hay không. Liên từ **IN** thử quan hệ thành viên này. Liên từ NOT IN thử quan hệ không là thành viên.

"Tìm tất cả các khách hàng có cả vay lẫn một tài khoản tại ngân hàng"

Ta đã sử dụng INTERSECTION để viết câu vấn tin này. Ta có thể viết câu vấn tin này bằng các sử dụng IN như sau:

```
SELECT DISTINCT Customer_name
```

```
FROM Borrower
WHERE Customer_name IN (        SELECT Customer_name
                                FROM Depositor)
```

Ví dụ này thử quan hệ thành viên trong một quan hệ một thuộc tính. SQL92 cho phép thử quan hệ thành viên trên một quan hệ bất kỳ.

"Tìm tất cả các khách hàng có cả vay lãn một tài khoản ở chi nhánh Perryridge"

Ta có thể viết câu truy vấn như sau:

```
SELECT DISTINCT Customer_name
FROM Borrower, Loan
WHERE       Borrower. Loan_number = Loan.Loan_number AND
            Branch_name = 'Perryridge'   AND
            (Branch_name. Customer_name IN
                  (SELECT Branch_name, Customer_name
                   FROM Depositor, Account
                   WHERE       Depositor.Account_number   =
                               Account.Account_number )
```

"Tìm tất cả các khách hàng có vay ngân hàng nhưng không có tài khoản tại ngân hàng"

```
SELECT DISTINCT Customer_name
FROM borrower
WHERE Customer_name NOT IN (  SELECT Customer_name
                              FROM Depositor)
```

Các phép toán IN và NOT IN cũng có thể được sử dụng trên các tập hợp liệt kê:

```
SELECT DISTINCT Customer_name
FROM borrower
WHERE Customer_name NOT IN ('Smith', 'Jone')
```

SO SÁNH TẬP HỢP (Set Comparision)

"Tìm tên của tất cả các chi nhánh có tài sản lớn hơn ít nhất một chi nhánh đóng tại Brooklyn"

```
SELECT DISTINCT Branch_name
FROM Branch AS T, Branch AS S
WHERE T.assets > S.assets AND S.branch_city = 'Brooklyn'
```

Ta có thể viết lại câu vấn tin này bằng cách sử dụng mệnh đề"lớn hơn ít nhất một"trong SQL

- **SOME** :

```
SELECT Branch_name
FROM Branch
WHERE Assets > SOME (    SELECT Assets
                         FROM Branch
                         WHERE Branch_city ='Brooklyn')
```

Câu vấn tin con

```
( SELECT Assets
  FROM Branch
  WHERE Branch_city ='Brooklyn')
```

sinh ra tập tất cả các Assets của tất cả các chi nhánh đóng tại Brooklyn. So sánh > SOME trong mệnh đề WHERE nhận giá trị đúng nếu giá trị Assets của bộ được xét lớn hơn ít nhất một trong các giá trị của tập hợp này.

SQL cũng có cho phép các so sánh **< SOME, >= SOME, <= SOME, = SOME, <> SOME**

- **ALL**

"Tìm tất cả các tên của các chi nhánh có tài sản lớn hơn tài sản của bất kỳ chi nhánh nào đóng tại Brooklyn"

```
SELECT Branch_name
FROM Branch
WHERE Assets > ALL (    SELECT Assets
                        FROM Branch
                        WHERE Branch_citty = 'Brooklyn')
```

SQL cũng cho phép các phép so sánh: **< ALL, <= ALL, > ALL, >= ALL, = ALL, <> ALL**.

"Tìm chi nhánh có số cân đối trung bình lớn nhất"

SQL không cho phép hợp thành các hàm tính gộp, như vậy MAX(AVG (...)) là không được phép. Do vậy, ta phải sử dụng câu vấn tin con như sau:

```
SELECT Branch_name
FROM Account
GROUP BY Branch_name
HAVING AVG (Balance) >= ALL ( SELECT AVG (balance)
                              FROM Account
                              GROUP BY Branch_name)
```

THỬ CÁC QUAN HỆ RỖNG

"tìm tất cả các khách hàng có cả vay lẫn tài khoản ở ngân hàng"

```
SELECT Customer_name
FROM Borrower
WHERE  EXISTS ( SELECT *
                FROM Depositor
                WHERE Depositor.Customer_name = Borrower.Customer_name)
```

Cấu trúc **EXISTS** trả lại giá trị true nếu quan hệ kết quả của câu vấn tin con không rỗng. SQL cũng cho phép sử dụng cấu trúc **NOT EXISTS** để kiểm tra tính không rỗng của một quan hệ.

"Tìm tất cả các khách hàng có tài khoản tại mỗi chi nhánh đóng tại Brooklyn"

```
SELECT DISTINCT S.Customer_name
FROM Depositor  AS S
WHERE NOT EXISTS      ( ( SELECT Branch_name
                          FROM Branch
                          WHERE Branch_city = 'Brooklyn')
                        EXCEPT
                          ( SELECT R.branch_name
                            FROM Depositor AS T, Account AS R
                            WHERE    T.Acoount_number = R.Account_number
                            AND  S.Customer_name = T.Customer_name) )
```

THỬ KHÔNG CÓ CÁC BỘ TRÙNG

SQL đưa vào cấu trúc **UNIQUE** để kiểm tra việc có bộ trùng trong quan hệ kết quả của một câu vấn tin con.

"Tìm tất cả khách hàng chỉ có một tài khoản ở chi nhánh Perryridge"

```
SELECT T.Customer_name
FROM Depositor AS T
WHERE UNIQUE ( SELECT   R.Customer_name
               FROM Account, Depositor AS  R
               WHERE T.Customer_name = R.Customer_name AND
                     R.Account_number    =    Account.Acount_number
                     AND   Account.Branch_name = 'Perryridge')
```

Ta có thể thử sự tồn tại của các bộ trùng trong một vấn tin con bằng cách sử dụng cấu trúc **NOT UNIQUE**

"Tìm tất cả các khách hàng có ít nhất hai tài khoản ở chi nhánh Perryridge"

```
SELECT DISTINCT T.Customer_name
FROM  Account, Depositor AS T
WHERE NOT UNIQUE ( SELECT R.Customer_name
                   FROM Account, Depositor AS  R
                   WHERE T.Customer_name=R.Customer_name
                AND   R.Account_number = Account.Account_number
                AND   Account.Branch_name = 'Perryridge')
```

UNIQUE trả lại giá rị false khi và chỉ khi quan hệ có hai bộ trùng nhau. Nếu hai bộ t_1, t_2 có ít nhất một trường null, phép so sánh $t_1 = t_2$ cho kết quả false. Do vậy UNIQUE có thể trả về giá trị true trong khi quan hệ có nhiều bộ trùng nhau nhưng chứa trường giá trị null !

QUAN HỆ DẪN XUẤT

SQL92 cho phép một biểu thức vấn tin con được dùng trong mệnh đề FROM. Nếu biểu thức như vậy được sử dụng, quan hệ kết quả phải được cho một cái tên và các thuộc tính có thể được đặt tên lại (bằng mệnh đề AS)

Ví dụ câu vấn tin con:

```
(SELECT  Branch_name, AVG(Balance)
  FROM Account
  GROUP BY Branch_name)
AS result (Branch_name, Avg_balace)
```

Sinh ra quan hệ gồm tên của tất cả các chi nhánh, và số cân đối trung bình tương ứng. Quan hệ này được đặt tên là result với hai thuộc tính Branch_name và Avg_balance.

"Tìm số cân đối tài sản trung bình của các chi nhánh tại đó số cân đối tài khoản trung bình lớn hơn 1200$"

```
SELECT Branch_name, avg_balance
FROM  ( SELECT  Branch_name, AVG(Balance)
          FROM Account
        GROUP BY Branch_name)
        AS result (Branch_name, Avg_balace)
WHERE avg_balance > 1200
```

VIEWS

Trong SQL, để định nghĩa view ta sử dụng lệnh **CREATE VIEW**. Một view phải có một tên.

CREATE VIEW < tên view > AS < Biểu thức vấn tin >

"Tạo một view gồm các tên chi nhánh, tên của các khách hàng có hoặc một tài khoản hoặc vay ở chi nhánh này"

Giả sử ta muốn đặt tên cho view này là All_customer.

```
CREATE VIEW All_customer        AS
(       SELECT Branch_name, Customer_name
        FROM Depositor, Account
        WHERE Depositor.Account_number = Account.Account_number )
UNION
(       SELECT Branch_name, Customer_name
        FROM Borrower, Loan
        WHERE Borrower.Loan_number = Loan.Loan_number)
```

Tên thuộc tính của một view có thể xác định một cách tường minh như sau:

```
CREATE VIEW Branch_total_loan (Branch_name, Total_loan)      AS
        ( SELECT Branch_name, sum(Amount)
          FROM Loan
          GROUP BY Branch_name)
```

Một view là một quan hệ, nó có thể tham gia vào các câu vấn tin với vai trò của một quan hệ.

```
SELECT Customer_name
FROM All_customer
WHERE Branch_name = 'Perryridge'
```

Một câu vấn tin phức tạp sẽ dễ hiểu hơn, dễ viết hơn nếu ta cấu trúc nó bằng cách phân tích nó thành các view nhỏ hơn và sau đó tổ hợp lại.

Định nghĩa view được giữ trong CSDL đến tận khi một lệnh DROP VIEW < tên view > được gọi.

Trong chuẩn SQL 3 hiện đang được phát triển bao hàm một đề nghị hỗ trợ những view tạm không được lưu trong CSDL.

SỬA ĐỔI CƠ SỞ DỮ LIỆU

DELETE
INSERT
UPDATE

XÓA (Delete)

Ta chỉ có thể xoá nguyên vẹn một bộ trong một quan hệ, không thể xoá các giá trị của các thuộc tính. Biểu thức xoá trong SQL là:

```
DELETE FROM  r
[WHERE  P]
```

Trong đó p là một vị từ và r là một quan hệ.

Lệnh DELETE duyệt qua tất cả các bộ t trong quan hệ r, nếu P(t) là true, DELETE xoá t khỏi r. Nếu không có mệnh đề WHERE, tất cả các bộ trong r bị xoá.

Lệnh DELETE chỉ hoạt động trên một quan hệ.

- DELETE FROM Loan = Xoá tất cả các bộ của quan hệ Loan
- DELETE FROM Depositor WHERE Customer_name = 'Smith'
- DELETE FROM Loan
  WHERE Amount BETWEEN 1300 AND 1500
- DELETE FROM Account
  WHERE Branch_name IN (SELECT Branch_name
  FROM Branch
  WHERE Branch_city = 'Brooklyn')
- DELETE FROM Account
  WHERE Balance < (SELECT AVG(Balance)
  FROM Account)

XEN (Insert)

Để xen dữ liệu vào một quan hệ, ta xác định một bộ cần xen hoặc viết một câu vấn tin kết quả của nó là một tập các bộ cần xen. Các giá trị thuộc tính của bộ cần xen phải thuộc vào miền giá trị của thuộc tính và số thành phần của bộ phải bằng với ngôi của quan hệ.

"Xen vào quan hệ Account một bộ có số tài khoản là A-9732, số cân đối là 1200$ và tài khoản này được mở ở chi nhánh Perryridge"

```
INSERT INTO Account
VALUES ('Perryridge', 'A-9732', 1200);
```

Trong ví dụ này thự tự các giá trị thuộc tính cần xen trùng khớp với thứ tự các thuộc tính trong sơ đồ quan hệ. SQL cho phép chỉ rõ các thuộc tính và các giá trị tương ứng cần xen:

```
INSERT INTO Account (Branch_name, Account_number, Balance)
VALUES ('Perryridge', 'A-9732', 1200);

INSERT INTO Account (Account_number, Balance, Branch_name)
VALUES ('A-9732', 1200, 'Perryridge');
```

"Cấp cho tất cả các khách hàng vay ở chi nhánh Perryridge một tài khoản với số cân đối là 200$ như một quà tặng sử dụng số vay như số tài khoản"

```
INSERT INTO Account
        SELECT Branch_name, Loan_number, 200
        FROM Loan
        WHERE Branch_name = 'Perryridge'
INSERT INTO Depositor
        SELECT Customer_name, Loan_number
        FROM Borrower, Loan
        WHERE Borrower.Loan_number = Loan.Loan_number AND
                Branch_name = 'Perryridge'
```

CẬP NHẬT (Update)

Câu lệnh UPDATE cho phép thay đổi giá trị thuộc tính của các bộ

"Thêm lãi hàng năm vào số cân đối với tỷ lệ lãi suất 5%"

```
UPDATE Account
SET Balance = Balance*1.05
```

Giả sử các tài khoản có số cân đối > 10000$ được hưởng lãi suất 6%, các tài khoản có số cân đối nhỏ hơn hoặc bằng 10000 được hưởng lãi suất 5%

```
UPDATE Account
SET Balance = Balance*1.06
WHERE Balance > 10000
UPDATE Account
SET Balance = Balance*1.05
WHERE Balance <= 10000
```

SQL92 đưa vào cấu trúc CASE như sau:

```
CASE
        WHEN P1 THEN Result1
        WHEN P2 THEN Result2
        ...
        WHEN Pn THEN Resultn
        ELSE  Result0
END
```

trong đó P_i là các vị từ, $Result_i$ là các kết quả trả về của hoạt động CASE tương ứng với vị từ P_i đầu tiên thỏa mãn. Nếu không vị từ P_i nào thỏa mãn CASE trả về $Result_0$.

Với cấu trúc CASE như vậy ta có thể viết lại yêu cầu trên như sau:

```
UPDATE Account
SET Balance = CASE
        WHEN Balance > 10000 THEN Balance*1.06
        ELSE Balance*1.05
END
```

"Trả 5% lãi cho các tài khoản có số cân đối lớn hơn số cân đối trung bình"

```
UPDATE Account
```

```
SET Balance = Balance*1.05
WHERE Balance >  SELECT AVG(Balance)
                 FROM Account
```

CÁC QUAN HỆ NỐI

SQL92 cung cấp nhiều cơ chế cho nối các quan hệ bao hàm nối có điều kiện và nối tự nhiên cũng như các dạng của nối ngoài.

```
Loan INNER JOIN Borrower
ON Loan.Loan_number = Borrower.Loan_number
```

Nối quan hệ Loan và quan hệ Borrower với điều kiên:

```
Loan.Loan_number = Borrower.Loan_number
```

Quan hệ kết quả có các thuộc tính của quan hệ Loan và các thuộc tính của quan hệ Borrower (như vậy thuộc tính Loan_number xuất hiện 2 lần trong quan hệ kết quả).

Để đổi tên quan hệ (kết quả) và các thuộc tính, ta sử dụng mệnh đề **AS**

```
Loan INNER JOIN Borrower
ON Loan.Loan_number = Borrower.Loan_number
AS  LB(Branch, Loan_number, Amount, Cust, Cust_Loan_number)
Loan LEFT OUTER JOIN Borrower
ON Loan.Loan_number = Borrower.Loan_number
```

Phép nối ngoài trái được tính như sau: Đầu tiên tính kết quả của nối trong INNER JOIN. Sau đó đối với mỗi bộ t của quan hệ trái (Loan) không tương xứng với bộ nào trong quan hệ bên phải (borrower) khi đó thêm vào kết quả bộ r gồm các giá trị thuộc tính trái là các giá trị thuộc tính của t, các thuộc tính còn lại (phải) được đặt là null.

```
Loan NATURAL INNER JOIN Borrower
```

Là nối tự nhiên của quan hệ Loan và quan hệ Borrower (thuộc tính trùng tên là Loan_number).

NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU (DDL)

DDL SQL cho phép đặc tả:

- Sơ đồ cho mỗi quan hệ
- Miền giá trị kết hợp với mỗi thuộc tính
- các ràng buộc toàn vẹn
- tập các chỉ mục được duy trì cho mỗi quan hệ
- thông tin về an toàn và quyền cho mỗi quan hệ
- cấu trúc lưu trữ vật lý của mỗi quan hệ trên đĩa

CÁC KIỂU MIỀN TRONG SQL

SQL-92 hỗ trợ nhiều kiểu miền trong đó bao hàm các kiểu sau:

- **char(n) / charater:** chuỗi ký tự dộ dài cố định, với độ dài n được xác định bởi người dùng
- **vachar(n) / character varying (n):** chuỗi ký tự độ dài thay đổi, với độ dài tối đa được xác dịnh bởi người dung là n
- **int / integer:** tập hữu hạn các số nguyên
- **smallint:** tập con của tập các số nguyên **int**
- **numeric(p, d):** số thực dấu chấm tĩnh gồm p chữ số (kể cả dấu) và d trong p chữ số là các chữ số phần thập phân
- **real, double precision:** số thực dấu chấm động và số thực dấu chấm động chính xác kép
- **float(n):** số thực dấu chấm động với độ chính xác được xác định bởi người dùng ít nhất là n chữ số thập phân

- **date:** kiểu năm tháng ngày (YYYY, MM, DD)
- **time:** kiểu thời gian (HH, MM, SS)

SQL-92 cho phép định nghĩa miền với cú pháp:

CREATE DOMAIN < tên miền > < Type >

Ví dụ: CREATE DOMAIN hoten char(30);

Sau khi đã định nghĩa miền với tên hoten ta có thể sử dụng nó để định nghĩa kiểu của các thuộc tính

ĐỊNH NGHĨA SƠ ĐỒ TRONG SQL.

Lệnh **CREATE TABLE** với cú pháp

CREATE TABLE < tên bảng > (
< Thuộc tính 1 > < miền giá trị thuộc tính 1 > ,
...
< Thuộc tính n > < miền giá trị thuộc tính n> ,
< ràng buộc toàn vẹn 1 > ,
...
< ràng buộc toàn vẹn k >)

Các ràng buộc toàn vẹn cho phép bao gồm:

primary key ($A_{i_1}, A_{i_2}, ..., A_{i_m}$ **)**

và

check(P)

Đặc tả **primary key** chỉ ra rằng các thuộc tính $A_{i_1}, A_{i_2}, ..., A_{i_m}$ tạo nên khoá chính của quan hệ. Mệnh đề **check** xác định một vị từ P mà mỗi bộ trong quan hệ phải thoả mãn.

Ví dụ:

```
CREATE TABLE customer (
        customer_name       CHAR(20) not null,
        customer_street     CHAR(30),
        customer_city       CHAR(30),
        PRIMARY KEY(customer_name));
CREATE TABLE branch (
        branch_name         CHAR(15) not null,
        branch_city         CHAR(30),
        assets              INTEGER,
        PRIMARY KEY (branch_name),
        CHECK (assets >= 0));
CREATE TABLE account (
        account_number      CHAR(10) not null,
        branch_name         CHAR(15),
        balance             INTEGER,
        PRIMARY KEY (account_number),
        CHECK       (balance >= 0));
CREATE TABLE depositor (
        customer_name       CHAR(20) not null,
        account_number      CHAR(10) not null,
        PRIMARY KEY (customer_name, account_namber));
```

Giá trị **null** là giá trị hợp lệ cho mọi kiểu trong SQL. Các thuộc tính được khai báo là primary key đòi hỏi phải là ***not null*** và ***duy nhất***. do vậy các khai báo not null trong ví dụ trên là dư (trong SQL-92).

```
CREATE TABLE student (
```

```
name            CHAR(15) not null,
student_ID      CHAR(10) not null,
degree_level    CHAR(15) not null,
PRIMARY KEY (student_ID),
CHECK (degree_level  IN  ('Bachelors', 'Masters', 'Doctorats'));
```

- Xoá một quan hệ khỏi CSDL sử dụng lệnh **Drop table** với cú pháp:
  DROP TABLE < tên bảng >
- Thêm thuộc tính vào bảng đang tồn tại sử dụng lệnh **Alter table** với cú pháp:
  ALTER TABLE < tên bảng > ADD < thuộc tính > < miền giá trị >
- Xoá bỏ một thuộc tính khỏi bảng đang tồn tại sử dụng lệnh **Alter table** với cú pháp:
  ALTER TABLE < Tên bảng > DROP < tên thuộc tính >

SQL NHÚNG (Embedded SQL)

Một ngôn ngữ trong đó các vấn tin SQL được nhúng gọi là ngôn ngữ chủ *(host language)*, cấu trúc SQL cho phép trong ngôn ngữ chủ tạo nên SQL nhúng. Chương trình được viết trong ngôn ngữ chủ có thể sử dụng cú pháp SQL nhúng để truy xuất và cập nhật dữ liệu được lưu trữ trong CSDL.

BÀI TẬP CHƯƠNG II

II.1. Xét CSDL bảo hiểm sau:

person(<u>ss#</u>, name, address): Số bảo hiểm ss# sở hữu bởi người tên name ở địa chỉ address

car(<u>license</u>, <u>year</u>, model): Xe hơi số dăng ký license, sản xuất năm year, nhãn hiệu Model

accident(<u>date</u>, <u>driver</u>, damage_amount): tai nạn xảy ra ngày date, do người lái driver, mức hư hại damage_amount

owns(<u>ss#</u>, <u>license</u>): người mang số bảo hiểm ss# sở hữu chiếc xe mang số đăng ký license

log(<u>license</u>, <u>date</u>, <u>driver</u>): ghi sổ chiếc xe mang số đăng ký license, bị tai nạn ngày do người lái driver

các thuộc tính được gạch dưới là các primary key. Viết trong SQL các câu vấn tin sau:

1. Tìm tổng số người xe của họ gặp tai nạn năm 2001
2. Tìm số các tai nạn trong đó xe của"John"liên quan tới
3. Thêm khách hàng mới: ss# =”A-12345”, name ="David”, address ="35 Chevre Road”, license ="109283”, year =”2002”, model ="FORD LASER"vào CSDL
4. xoá các thông tin lien quan dến xe model "MAZDA"của"John Smith”
5. Thêm thông tin tai nạn cho chiếc xe"TOYOTA"của khách hàng mang số bảo hiểm số"A-84626”

II.2. Xét CSDL nhân viên:

employee (E_name, street, city): Nhân viên có tên E_name, cư trú tại phố street, trong thành phố city

works (E_name, C_name, salary): Nhân viên tên E_name làm việc cho công ty C_name với mức lương salary

copany (C_name, city): Công ty tên C_name đóng tại thành phố city

manages(E_name, M_name): Nhân viên E_name dưới sự quản lý của nhân viên M_name

Viết trong SQL các câu vấn tin sau:

1. Tìm tên của tất cả các nhân viên làm việc cho First Bank
2. Tìm tên và thành phố cư trú của các nhân viên làm việc cho First Bank
3. Tìm tên, phố, thành phố cư trú làm việc cho First Bank hưởng mức lương > 10000$
4. Tìm tất cả các nhân viên trong CSDL sống trong cùng thành phố với công ty mang họ làm việc cho
5. Tìm tất cả các nhân viên sông trong cùng thành phố, cùng phố với người quản lý của họ
6. Tìm trong CSDL các nhân viên không làm việc cho First Bank
7. Tìm trong CSDL, các nhân viên hưởng mức lương cao hơn mọi nhân viên của Small Bank
8. Giả sử một công ty có thể đóng trong một vaì thành phố. Tìm tất cả các công ty đóng trong mỗi thành phố trong đó Small Bank đóng.
9. Tìm tất cả các nhân viên hưởng múc lương cao hơn mức lương trung bình của công ty họ làm việc
10. Tìm công ty có nhiều nhân viên nhất
11. Tìm công ty có tổng số tiền trả lương nhỏ nhất
12. Tìm tất cả các công ty có mức lương trung bình cao hơn mức luong trung bình của công ty First Bank
13. Thay đổi thành phố cư trú của nhân viên"Jones"thành NewTown
14. Nâng lương cho tất cả các nhân viên của First Bank lên 10%
15. nâng lương cho các nhà quản lý của công ty First Bank lên 10%
16. Xoá tất cả các thông tin liên quan tới cong ty Bad Bank