TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN - TIN HỌC

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

(*Bài giảng tóm tắt*)

NGƯỜI BIÊN SOẠN

TẠ THỊ THU PHƯỢNG

Đà Lạt 2009

MỤC LỤC

Chương 1
TỔNG QUAN VỀ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

I. Giới thiệu

Thông tin là nguồn tài nguyên quý giá của một tổ chức. Các phần mềm máy tính là những công cụ hiệu quả để xử lý thông tin và hệ quản trị cơ sở dữ liệu là công cụ phổ biến cho phép lưu trữ và rút trích thông tin một cách hiệu quả.

Hệ quản trị cơ sở dữ liệu quan hệ là hệ quản trị cơ sở dữ liệu phổ biến nhất hiện nay và được hỗ trợ bởi nhiều nhà cung cấp phần mềm. Tính hiệu quả của các ứng dụng phụ thuộc vào chất lượng của việc tổ chức dữ liệu. Những cải tiến trong kỹ thuật và xử lý cơ sở dữ liệu đưa đến các cơ hội sử dụng thông tin một cách linh hoạt và hiệu quả khi dữ liệu được tổ chức và lưu trữ trong các cấu trúc quan hệ. Hệ quản trị cơ sở dữ liệu là một thành công trong lĩnh vực thương mại.

Mục tiêu của hệ quản trị cơ sở dữ liệu.

Hệ quản trị cơ sở dữ liệu phải đảm bảo các mục tiêu sau: dữ liệu sẵn dùng (*data availability*), tính toàn vẹn dữ liệu (*data integrity*), an toàn dữ liệu (*data secutity*), và độc lập dữ liệu (*data independency*).

- Dữ liệu sẵn dùng (*data availability*): dữ liệu được tổ chức sao cho mọi người dùng có thể truy cập dễ dàng theo chức năng và nhiệm vụ của họ.
- Tính toàn vẹn dữ liệu (*data integrity*): dữ liệu lưu trữ trong cơ sở dữ liệu là đúng đắn, đáng tin cậy.
- An toàn dữ liệu (*data secutity*): Chỉ những người dùng được phép mới có thể truy cập dữ liệu. Nếu nhiều người dùng truy cập chung một mục dữ liệu cùng lúc thì hệ quản trị cơ sở dữ liệu không cho phép họ thực hiện những thay đổi gây mâu thuẫn dữ liệu.
- Độc lập dữ liệu (*data independency*): hệ quản trị cơ sở dữ liệu phải cho phép tất cả mọi người dùng được phép lưu trữ, cập nhật và rút trích dữ liệu hiệu quả mà không cần nắm chi tiết về cấu trúc của cơ sở dữ liệu được biểu diễn và cài đặt.

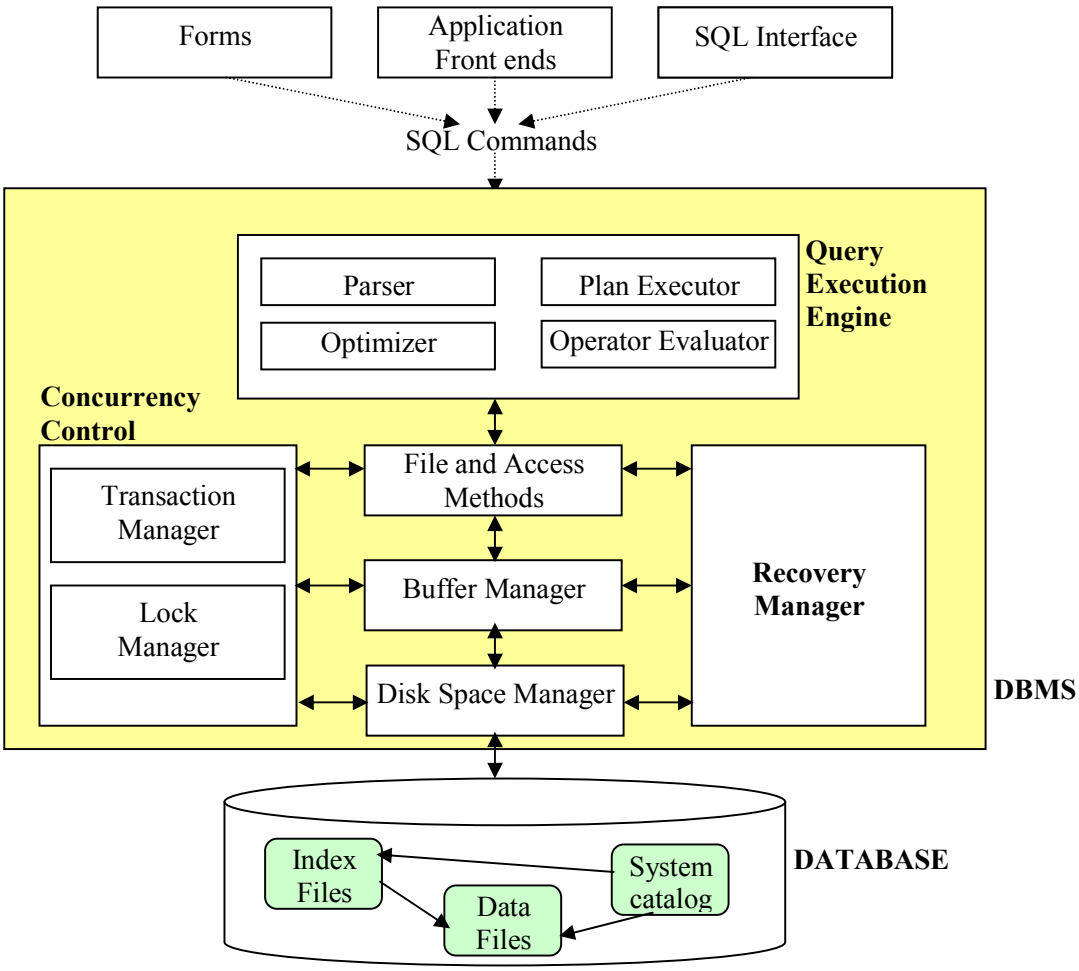
Quá trình phát triển của hệ quản trị cơ sở dữ liệu.

Quá trình phát triển của DBMS như sau:

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

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

II. Cấu trúc và thành phần của hệ quản trị cơ sở dữ liệu.



Hình 1.1 Kiến trúc của DBMS

Kiến trúc của hệ quản trị cơ sở dữ liệu gồm 2 thành phần chức năng:

- Bộ quản lý lưu trữ (Storage manager).
- Bộ Xử lý truy vấn (Query Processor).

1. Bộ quản lý lưu trữ

Bộ quản lý lưu trữ có nhiệm vụ lưu trữ, rút trích và cập nhật dữ liệu vào cơ sở dữ liệu. Bộ quản lý lưu trữ gồm có các đơn vị sau:

- Kiểm tra chứng thực và toàn vẹn.
- Quản lý giao dịch .
- Quản lý file.

– Quản lý vùng đệm.

Quản lý giao dịch (Transaction management)

Thông thường, một số thao tác trên cơ sở dữ liệu tạo thành một đơn vị logic công việc. Ta hãy xét ví dụ chuyển khoản, trong đó một số tiền x được chuyển từ tài khoản A (A:=A-x) sang một tài khoản B (B:=B+x). Một yếu tố cần thiết là cả hai thao tác này hoặc cùng xảy ra hoặc không hoạt động nào xảy ra cả. Việc chuyển khoản phải xảy ra trong tính toàn thể của nó hoặc không. Yêu cầu ***toàn thể-hoặc-không*** này được gọi là tính nguyên tố (atomicity). Một yếu tố cần thiết khác là sự thực hiện việc chuyển khoản bảo toàn tính nhất quán của cơ sở dữ liệu: *giá trị của tổng A + B phải được bảo toàn*. Yêu cầu về tính chính xác này được gọi là tính nhất quán (consistency). Cuối cùng, sau khi thực hiện thành công hoạt động chuyển khoản, các giá trị của các tài khoản A và B phải bền vững cho dù có thể có sự cố hệ thống. Yêu cầu về tính bền vững này được gọi là tính lâu bền (durability).

Một giao dịch là một tập các hoạt động thực hiện chỉ một chức năng logic trong một ứng dụng cơ sở dữ liệu. Mỗi giao dịch là một đơn vị mang cả tính nguyên tố lẫn tính nhất quán. Như vậy, các giao dịch phải không được vi phạm bất kỳ ràng buộc nhất quán nào: ***Nếu cơ sở dữ liệu là nhất quán khi một giao dịch khởi động thì nó cũng phải là nhất quán khi giao dịch kết thúc thành công***. Tuy nhiên, trong khi đang thực hiện giao dịch, phải cho phép sự không nhất quán tạm thời. Sự không nhất quán tạm thời này tuy là cần thiết nhưng lại có thể dẫn đến các khó khăn nếu xảy ra sự cố.

Trách nhiệm của người lập trình là xác định đúng đắn các giao dịch sao cho bảo toàn tính nhất quán của cơ sở dữ liệu.

Đảm bảo tính nguyên tố và tính lâu bền là trách nhiệm của hệ cơ sở dữ liệu nói chung và của ***thành phần quản trị giao dịch (transaction-management component)*** nói riêng. Nếu không có sự cố, tất cả giao dịch hoàn tất thành công và tính nguyên tố được hoàn thành dễ dàng. Tuy nhiên, do sự hiện diện của các sự cố, một giao dịch có thể không hoàn tất thành công sự thực hiện của nó. Nếu tính nguyên tố được đảm bảo, một giao dịch thất bại không gây ảnh hưởng đến trạng thái của cơ sở dữ liệu. Như vậy, cơ sở dữ liệu phải được hoàn lại trạng thái của nó trước khi giao dịch bắt đầu. Hệ quản trị cơ sở dữ liệu phải có trách nhiệm phát hiện sự cố hệ thống và trả lại cơ sở dữ liệu về trạng thái trước khi xảy ra sự cố.

Khi một số giao dịch tương tranh cập nhật cơ sở dữ liệu, tính nhất quán của dữ liệu có thể không được bảo toàn, ngay cả khi mỗi giao dịch là chính xác. ***Bộ quản trị điều khiển tương tranh (concurency-control manager)*** có trách nhiệm điều khiển các tương tác giữa các giao dịch đồng thời để đảm bảo tính thống nhất của CSDL.

***Thành phần Kiểm tra chứng thực và toàn vẹn* (*Authorization and Integrity Manager*)**

Kiểm tra ràng buộc toàn vẹn và quyền truy cập dữ liệu của người dùng cơ sở dữ liệu.

***Thành phần quản lý giao dịch* (*Transaction manager*)**

Thành phần này đảm bảo rằng cơ sở dữ liệu luôn ở trạng thái nhất quán. Nó quản lý việc thực thi các yêu cầu thao tác dữ liệu và đảm bảo các truy cập dữ liệu đồng thời không dẫn đến mâu thuẫn.

Thành phần quản lý file (File manager): quản lý việc cấp phát không gian trên đĩa. Các file được dùng để chứa tập các dữ liệu tương tự nhau. Hệ quản lý file quản lý các file độc lập, giúp đỡ nhập và lấy các mẩu tin. Thành phần quản lý file thiết lập và duy trì danh sách các cấu trúc và chỉ mục được định nghĩa trong lược đồ trong. Thành phần quản lý file có thể:

- Tạo file.
- Xóa file.
- Cập nhật mẩu tin trong file.
- Lấy một mẩu tin từ một file.

***Thành phần quản lý vùng đệm* (*Buffer Manager*)**: có trách nhiệm chuyển dữ liệu từ đĩa lưu trữ vào bộ nhớ chính theo yêu cầu của chương trình.

2. **Bộ xử lý truy vấn (Query Processor)**

Thực hiện câu truy vấn nhận được từ người dùng qua các giai đoạn phân tích (parser), tối ưu hóa câu hỏi (query optimizer), lập kế hoạch thực hiện (plan executor) và thực hiện tính toán (operator evaluator).

Chương 2
XÂY DỰNG, QUẢN LÝ VÀ KHAI THÁC CƠ SỞ DỮ LIỆU

I. Khái niệm cơ sở dữ liệu

- Ở mức logic, một cơ sở dữ liệu (CSDL) gồm:
  - Các bảng (tables) chứa dữ liệu có cấu trúc và các ràng buộc (constraint) định nghĩa trên các bảng.
  - Các khung nhìn (view).
  - Các thủ tục/ hàm.
  - Các vai trò (role) và người dùng (user).
  - …
- Ở mức lưu trữ vật lý, một database của SQL Server được lưu trữ bởi 3 loại tập tin:
  - Tập tin dữ liệu (data file) gồm có:
    - ✓ 1 tập tin dữ liệu chính (primary data file), thường có phần mở rộng "mdf": chứa các dữ liệu khởi đầu của database.
    - ✓ 0-n tập tin dữ liệu thứ cấp (secondary data file), thường có phần mở rộng "ndf": chứa các dữ liệu không lưu trữ hết trong tập tin dữ liệu chính.
  - Tập tin nhật ký giao tác (transaction log file) gồm có 1-n tập tin nhật ký, thường có phần mở rộng "ldf": *chứa các thông tin về nhật ký giao tác, dùng để phục hồi database sau khi xảy ra sự cố*.

II. Tạo cơ sở dữ liệu

1. Cú pháp lệnh tạo CSDL

Create Database *database_name*
[On [Primary]
{ *file_spec* [,…n] }
]
[Log on
{ *file_spec* [,…n] }
]

với

file_spec :: = (Name = *logical_file_name*,
Filename = '*os_file_name* '

[, Size = *size*]
[, Maxsize = { *max_size* | Unlimited }]
[, Filegrowth = *growth_increment*])

Mặc định, các tập tin dữ liệu và log được lưu trong thư mục MSSQL\ Data của thư mục cài đặt SQL Server.

Ví dụ

- Ví dụ 1: tạo CSDL QLSinhVien theo các quy định mặc định của SQL Server

```
Create Database QLSinhVien
```

- Ví dụ 2: tạo CSDL QLSinhVien với khai báo tên file logic, thư mục lưu tập tin dữ liệu chính, kích thước, …

```
Create Database QLSinhVien
On
(       Name = QLSV_Data
        Filename  = 'C:\ ...\ QLSV_Data.mdf ',
        Size = 1,
        Filegrowth = 10% )
```

- Ví dụ 3

```
Create Database QLSinhVien
On
(       Name = QLSV_Data1,
        Filename  = 'C:\ ...\ QLSV_Data.mdf ',
        Size = 1,
        Maxsize = 10 MB,
        Filegrowth = 1 MB   ) ,
(       Name = QLSV_Data2 ,
        Filename = 'C:\...\QLSV_Data1.ndf' )
Log on
(       Name = QLSV_Log,
        Filename = 'D:\...\QLSV_Log.ldf')
```

2. Xoá một CSDL đã tồn tại

```
Drop Database database_name
```

3. Thay đổi một CSDL

```
Alter Database database_name ….
```

Dùng để:

– Thêm/xoá/thay đổi các tập tin.
– Thay đổi các tùy chọn cho CSDL.

III. Kiểu dữ liệu

SQL Server cung cấp các kiểu dữ liệu:

1. Số

– Số nguyên: bit, tinyint, smallint, int, bigint.

- Số thực
  - ✓Floating point:
    - *float(n)*
    - *real = float(24)*
  - ✓Fixed point
    - *Decimal(p,s)*
    - *Numeric(p,s)*

2. Chuỗi

- ✓char(n): chuỗi có độ dài cố định.
- ✓ nchar(n): chuỗi (theo mã Unicode) có độ dài cố định.
- ✓varchar(n): chuỗi có độ dài thay đổi.
- ✓nvarchar(n): chuỗi (theo mã Unicode) có độ dài thay đổi.
- ✓text: kiểu dữ liệu cho phép chứa chuỗi có kích thước hơn 8KB.
- ✓ntext: kiểu dữ liệu cho phép chứa chuỗi (theo mã Unicode) có kích thước hơn 8KB.

3. Ngày giờ

- ✓Datetime.
- ✓Smalldatetime

4. *Kiểu người dùng tự định nghĩa*

a. Định nghĩa một kiểu dữ liệu:

sp_addtype *type_name*, *system_type* [, '*null_type*'][, '*owner*']

Ví dụ: định nghĩa kiểu dữ liệu Code là kiểu chuỗi gồm 10 ký tự cho phép để trống

Exec sp_addtype Code, char(10), 'NULL'

b. Xóa một kiểu dữ liệu người dùng định nghĩa:

sp_droptype '*type_name*'

IV. Tạo và quản lý bảng

1. Tạo bảng

- Xác định các cột (các thuộc tính) của bảng.
- Xác định khóa chính.
- Xác định các thuộc tính null/ not null.
- Xác định thuộc tính identity (nếu có) (phải là kiểu số nguyên).

Lưu ý:

- *Luôn tạo khóa chính cho một bảng.*
- *Ràng buộc khóa ngoại nên được tạo sau khi đã tạo xong tất cả các bảng liên quan.*

a. Cú pháp lệnh tạo bảng

Create table *Table_name*
(
{ *Column_name* *Data_type* [null | not null]
[default *default_value*]
[identity [(*seed*, *increment*)]]
} [,…n]
[, constraint *constraint_name* primary key (*Column_name* [,…n])]
)

Ví dụ: Tạo bảng học sinh có khóa chính là (STT, Lop)

```
Create table HOCSINH
(       STT         tinyint not null,
        Lop         char(5) not null  default  '11A1' ,
        HoTen       nvarchar(30) not null,
        NgaySinh    datetime not null,
        DiaChi      nvarchar(100),
        constraint  pk_HS primary key (STT, Lop)
)
```

b. Thay đổi cấu trúc bảng / xóa bảng

✓ Thay đổi cấu trúc bảng là thực hiện:

- Thêm/ xoá/ cập nhật kiểu dữ liệu của một cột (column).
- Thêm/ xoá/ kiểm tra/ không kiểm tra ràng buộc (constraint).
- Cho phép/ không cho phép trigger hoạt động.

Cú pháp: Alter table <tên_bảng>

…

✓ Xóa bảng: xoá dữ liệu và cấu trúc của bảng

Cú pháp: Drop table <tên_bảng>

Ví dụ

- Thêm thuộc tính DanToc vào bảng HOCSINH:

```
Alter table HOCSINH
        Add DanToc nvarchar(20) null default 'Kinh'
```

- Sửa kiểu dữ liệu của thuộc tính NgaySinh thành kiểu SmallDatetime**:**

```
Alter table HOCSINH
        Alter column NgaySinh SmallDatetime not null
```

2. Quản lý bảng

- Các tên bảng, tên ràng buộc không được trùng nhau trong cùng một database.

- Tên các cột trong cùng một bảng không được trùng nhau.
- Thông tin về các bảng, các ràng buộc được lưu trong *bảng hệ thống* ***sysobjects***

  Ví dụ: đọc thông tin về các bảng trong database hiện hành:

  Select * from sysobjects where type = 'U'
- Một số thủ tục SQL Server cung cấp để quản lý bảng và cấu trúc bảng:
  - sp_databases
  - sp_tables ['*table_name*'] [, '*owner*'][,'*database_name*'][, " '*type*' "]

    Ví dụ: Exec sp_tables null, null, null, " 'TABLE' "
  - sp_help [*object_name*]
    - ✓ sp_help cho biết các thông tin về đối tượng bất kỳ trong database (đối tượng có chứa trong sysobjects).

      Ví dụ: Exec sp_help HOCSINH
  - sp_columns *object* [, *owner*] [, *database*] [,*column*]
  - sp_helpconstraint '*table_name*'
  - …

3. Cài đặt ràng buộc toàn vẹn đơn giản

SQL Server cung cấp sẵn cơ chế để kiểm tra các loại ràng buộc toàn vẹn (RBTV) sau:

- Khóa chính (primary key constraint).
- Khóa ngoại (foreign key constraint).
- Giá trị duy nhất (unique constraint).
- Check constraint (Kiểm tra ràng buộc miền giá trị).

Có thể khai báo ràng buộc trong lúc tạo bảng hoặc khi bảng đã tồn tại. *Thông thường nên khai báo ràng buộc toàn vẹn trước khi nhập dữ liệu.*

a. Khai báo ràng buộc trong lúc tạo bảng

Cú pháp:

```
Create table Table_name
(   …
        [, constraint Constraint_name
                    {  primary key (Column_name [,…n])
                     | unique ( Column_name [,…n])
                       | check ( logical_expression ) }
        ] […n]
)
```

Ví dụ

```
Create table SinhVien
(
        MaSV        char(10) not null,
        HoTen       nvarchar(30) not null,
        Nam         tinyint,
        CMND        char(10),
        Khoa        char(5),
        constraint pk_SV primary key (MaSV),
        constraint u_CMND unique (CMND),
        constraint chk_Nam check (Nam > 0 and Nam <= 4)
)
```

b. Khai báo ràng buộc trên bảng đã tồn tại

Cú pháp:

```
Alter table table_name
        [with  check| with nocheck] Add
                { constraint  constraint_name
                        {   primary key ( column_name [,…n] )
                          | unique ( column_name [,…n] )
                          | check ( logical_expression )
                          | foreign key ( column_name [,…n] )
                              references ref_table ( ref_column [,…n] )
                                      [ on delete {cascade| no action} ]
                                      [ on update {cascade| no action} ]
                } [,…n]
```

Ví dụ

```
/* giả sử đã tồn tại bảng KHOA( MaKhoa, …) */
Alter table SINHVIEN
        with check add
          constraint u_CMND unique (CMND),
          constraint chk_Nam check (Nam in (1, 2, 3, 4) ),
          constraint fk_SV_maKhoa foreign key (Khoa),
                references KHOA(MaKhoa)
```

c. Kiểm tra / không kiểm tra ràng buộc

Cú pháp:

```
        Alter table Table_name
                {Check| Nocheck} constraint { All | constraint_name [,…n] }
```

Ví dụ:

```
        alter table SINHVIEN
                nocheck constraint u_CMND, chk_Nam
```

d. Xoá ràng buộc

Cú pháp:

Alter table *table_name*

Drop { [constraint] *constraint_name* } [,…n]

Ví dụ:

Alter table SINHVIEN

drop constraint u_CMND, chk_Nam

e. Rule

- Rule là một qui định chung được tạo ra trong một CSDL.
- Một rule có thể được áp dụng cho nhiều thuộc tính của nhiều bảng khác nhau, hoặc cho các kiểu dữ liệu người dùng định nghĩa trong database.

Tạo rule

Cú pháp:

Create rule *rule_name*
as *logical_expression*

(trong đó "logical_expression" phải chứa một biến. Biến này tương ứng với đối tượng sẽ được áp dụng rule).

Ví dụ:

create rule r_SoDuong
as @value >0

Kết buộc/ gỡ kết buộc rule

Kết buộc rule: dùng thủ tục:

sp_bindrule '*rule_name*', '*object*', ['futureonly']

trong đó:

- ✓ Tùy chọn *futureonly* chỉ dùng khi kết buộc rule với kiểu dữ liệu người dùng định nghĩa, có nghĩa các cột thuộc kiểu dữ liệu này trước đó không bị ảnh hưởng bởi rule.

Ví dụ: sp_bindrule 'r_SoDuong', 'SinhVien.Nam'

- ✓ Rule mới kết buộc sẽ ngầm gỡ rule cũ trên đối tượng.

Gỡ kết buộc

sp_unbindrule '*object*', ['futureonly']

Ví dụ: sp_unbindrule 'SinhVien.Nam'

Xoá rule

Cú pháp: Drop rule {*rule_name*} [,…n]

Lưu ý: *Chỉ xóa được rule khi nó không còn kết buộc với đối tượng nào.*

V. Các thao tác trên dữ liệu

Chú ý khi thêm/ xóa/ cập nhật dữ liệu:

– Dữ liệu nhập phải phù hợp với kiểu dữ liệu.

– Đảm bảo các ràng buộc toàn vẹn.

– Định dạng giá trị kiểu chuỗi unicode, kiểu datetime.

– Nhập giá trị rỗng (Null).

1. Các dạng lệnh insert

– *Thêm từng dòng dữ liệu vào bảng*

Insert [into] *Table_name*[(*column_name*[,…n])]
values (*value* [,…n])

– *Thêm 0-n dòng dữ liệu từ bảng khác/ từ kết quả của một câu truy vấn*

Insert [into] *Table_name*
Select_statement

Lưu ý: *trong câu select, ta có thể đọc dữ liệu từ các bảng trong database khác. Khi đó, tên bảng được viết đầy đủ như sau:*

Database_name.Owner.Table_name

Ví dụ: select * from QLSinhVien.dbo.SinhVien

2. Lệnh cập nhật dữ liệu

update *table_name*
set *column_name*_1= value1,…, *column_name_m= value_m*
[where *conditional_expression*]

3. Lệnh xoá dữ liệu

delete [from] *table_name*
[where *conditional_expression*]

VI. Truy vấn dữ liệu

1. Câu truy vấn tổng quát

Cú pháp tổng quát của câu truy vấn dữ liệu:

SELECT [tính chất] <danh sách các thuộc tính_1>
FROM <danh sách các table hoặc query/view [as alias] >
[**WHERE** <điều kiện_1>]
[**GROUP BY** <danh sách các thuộc tính_2>]
[**HAVING** <điều kiện_2>]
[**ORDER BY** <danh sách các thuộc tính_3> [ASC | DESC]]

trong đó:

– ***Tính chất*** là một trong các từ khóa: **ALL** (chọn ra tất cả các dòng trong bảng), **DISTINCT** (loại bỏ các dòng trùng lắp thông tin), **TOP <n>** (chọn n dòng đầu tiên thỏa mãn điều kiện).

- ***Danh sách các thuộc tính_1***: tên các thuộc tính cho biết thông tin cần lấy.

Chú ý:

- ✓ Các thuộc tính cách nhau bởi dấu ‘,’.
- ✓ Nếu lấy tất cả các thuộc tính của 1 bảng R thì dùng: R.*
- ✓ Nếu sau FROM chỉ có 1 bảng và lấy tất cả các cột của bảng đó thì dùng select *.
- ✓ Nếu tồn tại 1 thuộc tính sau select xuất hiện ở 2 bảng sau FROM thì phải chỉ định rõ thuộc tính đó thuộc bảng nào.

- ***Danh sách các table/query/view***: các bảng, câu truy vấn, hoặc khung nhìn chứa thông tin cần lấy. Khi tìm kiếm thông tin trên nhiều hơn 2 bảng/truy vấn thì phải kết các bảng lại với nhau (có thể đặt điều kiện kết đặt sau where hoặc đặt trong mệnh đề **From… join/ left join/ right join/full join … on …**).
- ***Alias***: bí danh (tên tắt) của bảng dùng cho các bảng có tên quá dài, hoặc một bảng được dùng nhiều lần trong mệnh đề from của câu truy vấn.
- ***điều kiện_1***: là điều kiện để lọc dữ liệu (chọn các bộ thoả điều kiện).
- ***danh sách các thuộc tính_2***: dữ liệu sẽ được gom nhóm theo các cột này, độ ưu tiên tính từ trái sang.
- ***điều kiện_2***: điều kiện lọc các nhóm theo một tiêu chí đại diện cho cả nhóm.
- ***danh sách các thuộc tính_3***:sắp xếp dữ liệu theo cột nào, thứ tự là tăng (ASC) hoặc giảm (DESC). Mặc định là dữ liệu được sắp theo thứ tự tăng dần. Việc sắp xếp được thực hiện theo thứ tự ưu tiên từ trái qua phải.

Lưu ý:

- ➢ *Nếu câu truy vấn không có mệnh đề Group By thì cũng không có mệnh đề Having.*
- ➢ *Nếu câu truy vấn có chứa mệnh đề Group By thì* ***Danh sách các thuộc tính_1*** *chỉ chứa các thuộc tính hoặc biểu thức liên quan đến các thuộc tính trong* ***danh sách các thuộc tính_2*** *và các hàm gộp (max, min, avg, sum, count).*

2. Các hàm thường dùng

- Các hàm gộp (Aggregate functions): max, min, sum, avg, count
- Các hàm thời gian.
- Các hàm toán học.
- Các hàm xử lý chuỗi.
- ….

(*Sinh viên có thể tra cứu theo từ khóa trong Books Online*).

VII. Tạo và sử dụng khung nhìn (View)

1. Khái niệm khung nhìn:

Khung nhìn (View) là một bảng ảo, có cấu trúc như một bảng, khung nhìn không lưu trữ dữ liệu mà dữ liệu của nó được tạo ra khi sử dụng, khung nhìn là đối tượng thuộc CSDL. Khung nhìn được tạo ra từ câu lệnh truy vấn dữ liệu (lệnh Select), truy vấn từ một hoặc nhiều bảng dữ liệu.

2. Sử dụng khung nhìn

- Khung nhìn được sử dụng khai thác dữ liệu như một bảng dữ liệu, có thể được chia sẻ bởi nhiều người dùng, an toàn trong khai thác.
- Có thể thực hiện truy vấn dữ liệu trên cấu trúc của khung nhìn.
- Các khung nhìn được tạo từ nhiều bảng hoặc trong khung nhìn có chứa từ khóa DISTINCT, hàm gộp, mệnh đề **group by** đều không cho phép cập nhật dữ liệu từ khung nhìn vào các bảng gốc trong cơ sở dữ liệu.

Cú pháp tạo khung nhìn:

Create View *view_name*

As *Select_statement*

VIII. Tạo và sử dụng chỉ mục (Index)

Chỉ mục (Index) là một phần quan trọng đối với CSDL, đặc biệt là cơ sở dữ liệu lớn. Chỉ mục được thiết lập từ một hoặc nhiều cột dữ liệu của bảng dữ liệu. Các giá trị của Chỉ mục sẽ được sắp xếp và lưu trữ theo một danh sách (bảng khác). Mỗi giá trị chỉ mục là duy nhất trong danh sách và nó sẽ liên kết đến giá trị trong bảng dữ liệu (liên kết dạng con trỏ). Việc lưu trữ dữ liệu của bảng có khóa chỉ mục được thực hiện theo cấu trúc B-Cây nhằm tăng tốc độ truy xuất dữ liệu đối với ổ đĩa (thiết bị thứ cấp).

Khi tìm kiếm một giá trị trong cột dữ liệu, mà cột này tham gia tạo Chỉ mục, đầu tiên câu lệnh xác định vị trí của giá trị nằm trong Chỉ mục bằng phép duyệt cây, sau đó thực hiện tìm theo liên kết đến bản ghi chứa giá trị tương ứng với khóa trong bảng.

1. Lựa chọn chỉ mục

- Không có chỉ mục, hệ quản trị CSDL thực hiện truy vấn bằng cách duyệt qua từng dòng trong bảng.
- Cài đặt các chỉ mục cho bảng giúp truy vấn thông tin nhanh hơn (tìm kiếm trên B-Cây).
- Khóa chính và các ràng buộc unique hiển nhiên là các chỉ mục của bảng.
- Cơ sở để chọn cài đặt chỉ mục: dựa vào các nhu cầu truy vấn thực hiện *thường xuyên* trên CSDL.

- Nên cài đặt chỉ mục cho các trường hợp sau:
  - Trường hợp 1: Có nhu cầu truy vấn thường xuyên các bộ của bảng Q theo một số (tập) thuộc tính nào đó.

    Ví dụ: GiaoDich(MãGD, …,NgàyGD): Có nhu cầu truy xuất *thường xuyên* các bộ của giao dịch trong một ngày hoặc trong một khoảng thời gian nhất định: cài đặt chỉ mục trên thuộc tính NgayGD của quan hệ GiaoDich.
  - Trường hợp 2: tập thuộc tính tham gia vào phép kết của một câu truy vấn xảy ra thường xuyên.

    Ví dụ: cho 2 lược đồ quan hệ:

    HocSinh(STT, Lop, HoTen,…)

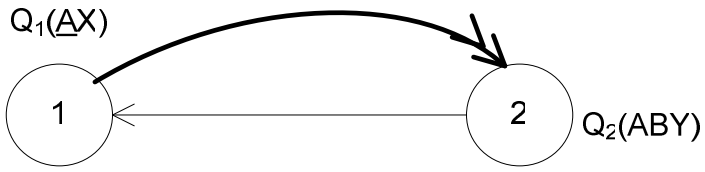
    KetQua(STT, Lop, Mon, Diem)

    Thường xuyên có nhu cầu truy vấn: cho biết kết quả học tập của một học sinh. Câu lệnh truy vấn như sau:

```
select hs.STT, hs.Lop, hs.HoTen, kq.Mon, kq.Diem
from HocSinh hs join KetQua kq on hs.STT = kq.STT
        and hs.Lop = kq.Lop
```

    → Cài đặt chỉ mục (STT, Lop) cho quan hệ KetQua

    Tổng quát: trên mô hình quan hệ, xác định các *con đường truy xuất thường xuyên:*



    → Từ một bộ của Q_1(một giá trị cụ thể a của A) có nhu cầu truy xuất thường xuyên các bộ của Q_2 tương ứng (tìm kiếm các bộ của Q_2 với A = a): khai báo chỉ mục (A) cho Q_2.

Lưu ý: *một chỉ mục (AB) khác với hai chỉ mục (A) và (B).*

2. Các loại chỉ mục

Có hai loại chỉ mục:

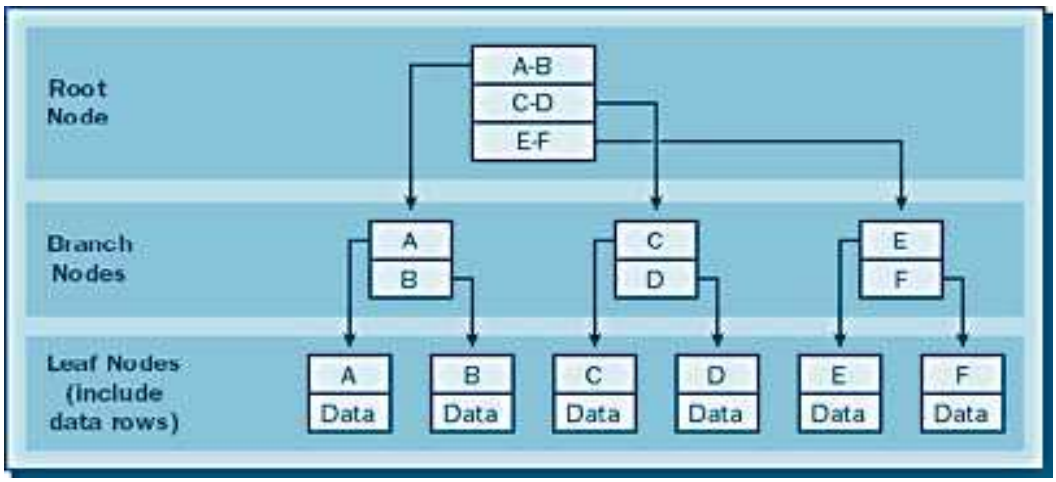
- Clustered index
- Nonclustered index

Clustered index:

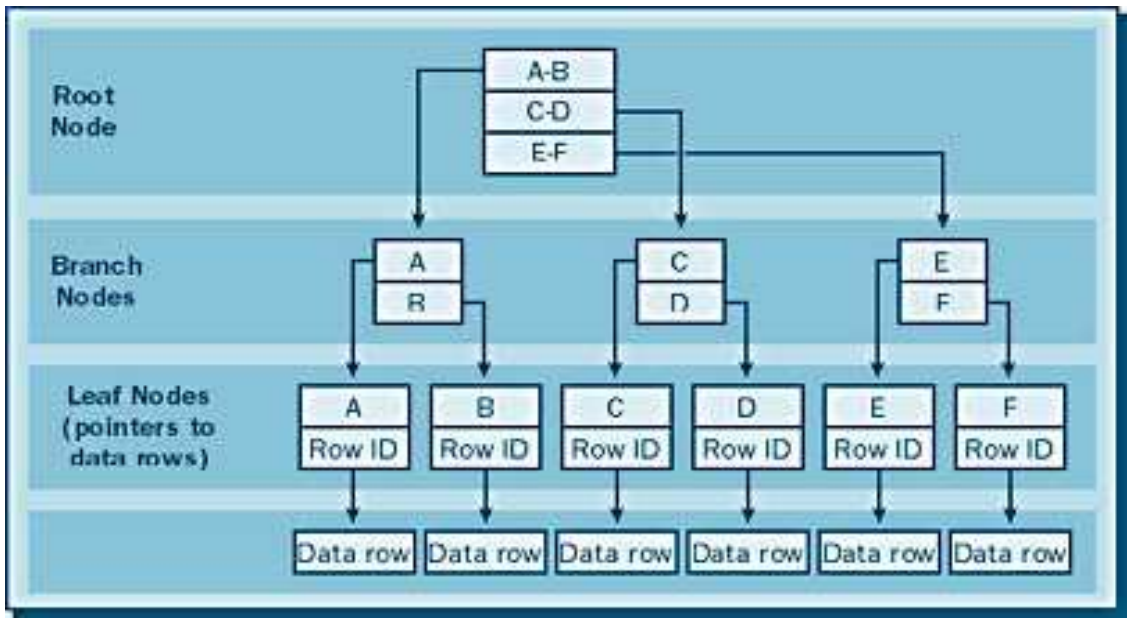
- Dữ liệu thật sự được sắp xếp vật lý theo chỉ mục (thật sự nằm ở nút lá của cây).
- Mỗi bảng chỉ có thể có một clustered chỉ mục, thường là khóa chính.

Nonclustered index:

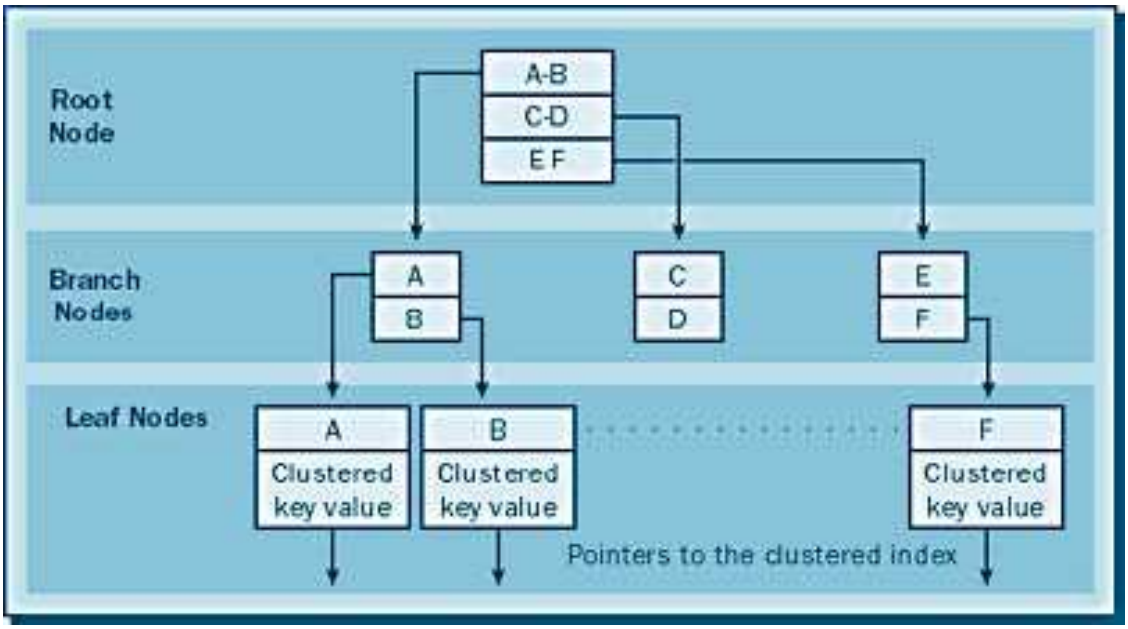
- Chỉ mục logic, dữ liệu thật sự không được sắp xếp vật lý theo chỉ mục.
- Nút lá là con trỏ trỏ đến vị trí của bộ dữ liệu, hoặc trỏ đến giá trị của clustered chỉ mục (trong trường hợp bảng có clustered index).



– Không có clustered index:



– Có clustered index



- Một số cân nhắc khi chọn chỉ mục:
  - Sử dụng nhiều chỉ mục tăng tốc độ truy vấn, nhưng làm giảm hiệu quả của các thao tác thêm/xoá/cập nhật dữ liệu.
  - Không nên tạo chỉ mục trên các bảng quá nhỏ (vài trăm dòng).
  - Chỉ nên chọn chỉ mục mà mỗi giá trị của nó tương ứng với một số ít bộ. Nếu mỗi giá trị chỉ mục ứng với trên 20% số lượng bộ trong bảng, thực hiện truy vấn bình thường bằng cách duyệt qua các dòng trong bảng sẽ hiệu quả hơn.
  - Các giá trị chỉ mục phải phân bố đều các bộ trong bảng.
  - Cố gắng dùng các chỉ mục với số thuộc tính ít (chiếm ít không gian và cần ít chi phí duy trì hơn chỉ mục với số thuộc tính lớn).
  - Clustered index phải nhỏ (số thuộc tính ít, kích thước nhỏ), vì các chỉ mục nonclustered đều phải gắn kết tới nó.

3. Cài đặt chỉ mục với SQL Server

Một số qui định:

1. Một bảng có tối đa 249 nonclustered chỉ mục (bao gồm cả những chỉ mục ngầm định khi khai báo khóa chính và chỉ mục).
2. Kích thước tối đa của một chỉ mục (tổng kích thước các thuộc tính tham gia vào chỉ mục) không quá 900 bytes.
3. Mặc định: chỉ mục clustered được khai báo ngầm định cùng với khai báo khóa chính, các trường hợp khác là nonclustered (tất nhiên có thể chỉ định khác đi).

Cú pháp khai báo chỉ mục:

Create [Unique][Cluster| Nonclustered] Chỉ mục *chỉ mục_name*

On {*table* | *view* } (*column* [Asc | Desc] [,...*n*])

Ví dụ:

Create nonclustered chỉ mục idx_STTHS_Lop

On KETQUA (STTHS, Lop)

Cú pháp xóa chỉ mục:

Drop Chỉ mục *table_name* (*chỉ mục_name*)

Ví dụ:

Drop Chỉ mục KETQUA(idx_STTHS_Lop)

IX. Chuyển đổi dữ liệu với các ứng dụng khác

(xem các tài liệu hướng dẫn thực hành SQL Server kèm theo)

Chương 3
T-SQL NÂNG CAO

I. Khai báo và sử dụng biến

1. Biến cục bộ

- Là một đối tượng có thể chứa giá trị thuộc một kiểu dữ liệu nhất định, tên biến bắt đầu bằng một ký tự @.
- Biến cục bộ có giá trị trong một *query batch* hoặc trong một thủ tục thường trú (stored procedure) hoặc hàm (function).
- Khai báo biến cục bộ bằng lệnh declare: cung cấp tên biến và kiểu dữ liệu:

Declare *tên_biến Kiểu_dữ_liệu*

Ví dụ:

Declare @MaSinhVien *char(10)*
Declare @HoTen *nvarchar(30)*
Declare @Sum *float*, @Count *int*

- Để gán giá trị cho một biến cục bộ dùng lệnh ***set***. Giá trị gán cho biến phải phù hợp với kiểu dữ liệu của biến.

Set *tên_biến = giá_trị*
Set *tên_biến = tên_biến*
Set *tên_biến* = b*iểu_thức*
Set *tên_biến = kết_quả_truy_vấn*

Ví dụ:

Set @MaLop = 'TH2001'
Set @SoSV = (select count (*) from SinhVien)
Set @MaLop = 'TH'+Year(@NgayTuyenSinh)

Đưa kết quả truy vấn vào biến:

SV(MaSV: int; HoTen: nvarchar(30), Tuoi int)
Select @Var1 = HoTen, @Var1 = Tuoi from SV
where MaSV = 1

Lưu ý: *nếu câu truy vấn trả về nhiều dòng, các biến chỉ nhận giá trị tương ứng của dòng đầu tiên.*

2. Biến toàn cục

- Là các biến hệ thống do SQL Server cung cấp, tên biến bắt đầu bằng 2 ký tự @
- SQL tự cập nhật giá trị cho các biến này, người sử dụng không thể gán giá trị trực tiếp.
- **Một số biến hệ thống thường dùng**
  - **@@error:** thông báo mã lỗi, nếu @@error = 0: thao tác thực hiện thành công.

- **@@rowcount:** cho biết số dòng bị ảnh hưởng bởi lệnh cuối (insert, update, delete).
- **@@trancount:** cho biết số giao dịch đang hoạt động trên kết nối hiện tại.
- **@fetch_status:** cho biết thao tác lấy dữ liệu từ cursor có thành công không.

II. Cấu trúc điều khiển

1. Lệnh If…else

- Chức năng: xét điều kiện để quyết định những lệnh T-SQL nào sẽ được thực hiện
- Cú pháp:

```
If  biểu_thức_điều kiện
        Lệnh| Khối_lệnh
[Else Lệnh| Khối_lệnh]
```

Khối lệnh là một hoặc nhiều lệnh nằm trong cặp từ khóa **begin…end**

Ví dụ: xét 2 lược đồ quan hệ (LĐQH)

HocPhan(MaHP, TenHP, SiSo)

DangKy(MaSV, MaHP)

Viết lệnh để thêm một đăng ký mới cho sinh viên có mã số 001 vào học phần HP01 (giả sử học phần này đã tồn tại trong bảng HocPhan). Lời giải như sau:

```
Declare @SiSo int
select @SiSo = SiSo from HocPhan where MaHP= 'HP01'
if @SiSo < 50
Begin
    insert into DANG_KY(MaSV, MaHP)
            values('001', 'HP01')
    print N'Đăng ký thành công'
End
Else
    print N'Học phần đã đủ SV'
```

2. Lệnh While

- Chức năng: thực hiện lặp lại một đoạn lệnh T-SQL khi điều kiện còn đúng.
- Cú pháp:

While *biểu_thức_điều_kiện*

Lệnh| Khối lệnh

– Có thể sử dụng *Break* và *Continue* trong khối lệnh của while

- ✓ Break: thoát khỏi vòng while hiện hành.
- ✓ Continue : trở lại đầu vòng while, bỏ qua các lệnh sau đó.

Ví dụ: xét lược đồ quan hệ SinhVien(MaSV: int, HoTen: nvarchar(30))

Viết lệnh xác định một mã sinh viên mới theo qui định: mã sinh viên tăng dần, nếu có chỗ trống thì mã mới xác định sẽ chèn vào chỗ trống đó. Chẳng hạn, nếu trong bảng sinhvien đã có các mã sinh viên 1, 2, 3, 7 → mã sinh viên mới là 4.

Giải:

```
Declare @STT int
Set @STT = 1
While exists(select * from SV where MaSV = @STT)
        set @STT = @STT+1
Insert into SV(MaSV, HoTen) values(@STT, 'Nguyen Van A')
```

3. Lệnh Case

– Chức năng: kiểm tra một dãy các điều kiện và trả về kết quả phù hợp với điều kiện đúng. Lệnh case được sử dụng như một hàm trong câu select.

– Cú pháp: Có hai dạng:

- Dạng 1 (simple case):

```
Case Biểu_thức_đầu_vào
        When Giá_trị  then kết_quả
        [...n]
        [ Else kết_quả_khác]
End
```

- Dạng 2 (searched case):

```
Case
        When biểu_thức_điều kiện  then kết_quả
        [...n]
        [ Else kết_quả_khác]
End
```

Ví dụ: xét LĐQH NHAN_VIEN(MaNV, HoTen, NgaySinh, CapBac,Phai)

Cho biết những nhân viên đến tuổi nghỉ hưu biết rằng tuổi về hưu của nam là 60, của nữ là 55).

Giải:

```
select * from NHAN_VIEN
        where datediff(yy, NgaySinh, getdate()) > =
            Case Phai
                when 'Nam' then 60
                when 'Nu' then 55
            End
```

Cho biết mã NV, họ tên và loại nhân viên (cấp bậc <=3:bình thường, cấp bậc = null: chưa xếp loại, còn lại: cấp cao).

Giải:

```
Select MaNV, HoTen, 'Loai' =  Case
        when CapBac<=3 then 'Binh Thuong'
        when CapBac is null then 'Chua xep loai'
        else 'Cap Cao'    End
From NhanVien
```

III. Thủ tục thường trú (Stored Procedures)

1. Khái niệm

Thủ tục thường trú (Stored Procedures - SP) chứa các lệnh T_SQL. Tương tự như một thủ tục trong các ngôn ngữ lập trình, SP trong SQL Server có thể truyền tham số, có tính tái sử dụng. Các thủ tục này được dịch và lưu trữ thành một đối tượng trong CSDL.

Ý nghĩa:

- Tính tái sử dụng, tính uyển chuyển nhờ hệ thống tham số.
- Khi biên dịch SP, các lệnh trong của nó được tối ưu hóa nó sao cho thực thi hiệu quả nhất. Kết quả tối ưu hóa được lưu bền vững. Khi gọi thực thi thủ tục không cần biên dịch và tối ưu hóa lại → lời gọi thủ tục tiết kiệm thời gian và tài nguyên hơn khối lệnh tương đương thân thủ tục.
- Trong ứng dụng triển khai theo môi trường client/server, client gửi lời gọi SP lên server thì chiếm đường truyền ít hơn rất nhiều lần so với việc gửi khối lệnh tương đương trong thân thủ tục → Giảm khối lượng thông tin trao đổi khi ứng dụng gửi yêu cầu thực hiện công việc về cho server do đó tránh nghẽn đường truyền, giảm trì trệ.

- Đóng gói chỉ các thao tác cho phép trên CSDL vào các SP và quy định truy xuất dữ liệu phải thông qua SP. Ngoài ra còn có thể phân quyền trên SP → Hỗ trợ tốt hơn cho việc đảm bảo an toàn (security) cho CSDL.
- SP giúp cho việc kết xuất báo biểu bằng Crystal Report trở nên đơn giản và hiệu quả hơn rất nhiều so với việc kết xuất dữ liệu trực tiếp từ các bảng và khung nhìn.

2. Khai báo và sử dụng thủ tục

Cú pháp khai báo:

Create {proc | procedure} *procedure_name*
{*Parameter_name DataType* [=*default*] [output] }[,…n]
As
{ khối lệnh }
Go

Lưu ý:

- ✓ *Tên tham số đặt theo qui tắc như tên biến cục bộ.*
- ✓ *Giá trị trả về của SP dùng một (hay một số) tham số* output.

Ví dụ:

- *Xây dựng SP cho biết danh sách sinh viên của một lớp có mã cho trước*

```
Create proc DS_Lop @MaLop varchar(10)
As
        Select SV.MaSV, SV.HoVaTen, SV.NgaySinh
        From SinhVien SV where SV.Lop = @MaLop
Go
```

- *Xây dựng SP tính toán giá trị cho đơn hàng có mã cho trước với quan hệ DonHang như sau:*
  DonHang(Ma, SoLuong, DonGia, ThueSuat, ChietKhau, ThanhTien)

```
Create proc TongTien @MaDH varchar(10)
As
Declare @ThanhTien float
Declare @TienThue float
Declare @TienChietKhau float
Declare @DonGia float,@SoLuong int
Set @SoLuong = (select SoLuong from DonHang where Ma = @MaDH)
```

```
Set @DonGia = (select DonGia from DonHang where Ma = @MaDH)
Set @TienThue = (select ThueSuat from DonHang where Ma = @MaDH)
Set @TienChietKhau = (select ChietKhau from DonHang
                        where Ma = @MaDH)
Set @ThanhTien = @DonGia*@SoLuong
Set @TienThue = @ThanhTien*@TienThue/100
Set @ThanhTien = @ThanhTien + @TienThue
Set @TienChietKhau = @ThanhTien*@TienChietKhau/100
Set @ThanhTien = @ThanhTien - @TienChietKhau
Update DonHang set ThanhTien = @ThanhTien where Ma = @MaDH
Go
```

– *Viết thủ tục thêm một đăng ký của sinh viên vào một học phần (tổng quát ví dụ trong phần If ...else)*

```
Create procedure usp_ThemDangKy
    @MaSV char(5), @MaHP char(5),
    @SiSo int = 0 output
As
select @SiSo = SiSo from HocPhan where MaHP= @MaHP
if @SiSo < 50
 Begin
        insert into DANG_KY(MaSV, MaHP)
                values(@MaSV, @MaHP)
        set @SiSo = @SiSo+1
        return 1
    End
 return 0
Go
```

- *Xây dựng SP tính điểm trung bình và xếp loại cho sinh viên thuộc lớp cho trước. Giả sử có các quan hệ như sau:*

SinhVien (MaSV, HoTen, DTB, XepLoai, Lop)

MonHoc (MaMH, TenMH)

KetQua (MaMH, MaSV, LanThi, Diem)

trong đó:

- Điểm thi chỉ tính lần thi sau cùng.
- Xếp loại: Xuất sắc [9, 10], Giỏi [8, 8.9], Khá [7, 7.9], Trung bình [5.0, 6.9], Yếu [0,4.9].
- Kết quả xuất dạng tham số output, không ghi xuống CSDL.

Giải

```
Create proc XepLoaiSV @MaSV varchar(10), @DTB float out put,
                              @XL nvarchar(20) out put
As
        Set @DTB = (Select avg(Diem) from KetQua Kq
                        Where MaSV = @MaSV
                                and  not exists (select * from KetQua Kq1
                                                        where Kq1.MaSV = @MaSV
                                                                and Kq1.MaMH=Kq.MaMH
                                                                and Kq1.LanThi > Kq.LanThi))
        If @DTB >= 9
                Set @XL = N'Xuất sắc'
        Else if @DTB >= 8
                Set @XL = N'Giỏi'
        Else if @DTB >= 7
                Set @XL = N'Khá'
        Else if @DTB >= 5
                Set @XL = N'Trung bình'
        Else
                Set @XL = N'Yếu'
Go
```

Cú pháp gọi thực hiện thủ tục:

EXEC| EXECUTE

{ [*@return_status* =] *procedure_name*

{ [*@parameter _name* =] *value* [OUTPUT] } [,...*n*]

Lưu ý:

- *Khi gọi thực hiện SP, dùng từ khóa Exec và cần truyền đủ tham số với kiểu dữ liệu phù hợp và thứ tự chính xác như khai báo trong định nghĩa SP.*
- *Có thể truyền giá trị cho tham số đầu vào (input) là một hằng hoặc một biến đã gán giá trị, không truyền được một biểu thức.*
- *Để nhận được giá trị kết quả (thông qua tham số đầu ra), cần truyền vào một biến và có từ khóa output.*

Ví dụ:

- Gọi thủ tục usp_ThemDangKy:

```
Exec usp_ThemDangKy  '001', 'HP01'
```

hoặc

```
Exec usp_ThemDangKy @MaHP = 'HP01', @MaSV = '001'
```

- Gọi thủ tục usp_ThemDangKy có nhận kết quả đầu ra:

```
Declare @SiSo int
Exec usp_ThemDangKy '001','HP01', @SiSo output
Print  @SiSo
```

- Gọi thủ tục usp_ThemDangKy có nhận kết quả đầu ra và kết quả trả về từ thủ tục :

```
Declare @SiSo int, @KetQua int
Exec @KetQua = usp_ThemDangKy '001','HP01', @SiSo output
```

- Gọi thực hiện thủ tục xếp loại sinh viên:

```
Declare@MaSinhVien varchar(10)
Declare@DiemTB varchar(10)
Declare@XepLoai varchar(10)
Set@MaSinhVien = '0712345'
Exec XepLoaiSV @MaSinhVien,@DiemTB out put,@XepLoai out put
Exec XepLoaiSV '0713478', @DiemTB out put,@XepLoai out put
```

Sửa thủ tục

Thay từ khóa Create trong lệnh tạo thủ tục bằng từ khóa Alter.

Xóa thủ tục

```
Drop {procedure|proc} procedure_name
```

Ví dụ: Drop procedure usp_ThemDangKy

3. Stored procedure hệ thống

- Là những thủ tục do SQL Server cung cấp sẵn để thực hiện các công việc: quản lý CSDL, quản lý người dùng, cấu hình CSDL,…
- Các thủ tục này có tên bắt đầu bằng "sp_" → Khi xây dựng thủ tục, tránh đặt tên thủ tục bắt đầu với "sp_".

IV. Kiểu dữ liệu cursor

1. Khái niệm Cursor

- Là một cấu trúc dữ liệu ánh xạ đến một tập các dòng dữ liệu kết quả của một câu truy vấn (select).

- Cho phép duyệt tuần tự qua tập các dòng dữ liệu và đọc giá trị từng dòng.
- Thể hiện của cursor là 1 biến, nhưng tên biến này không bắt đầu bằng '@'.
- Vị trí hiện hành của cursor có thể được dùng như điều kiện trong mệnh đề where của lệnh update hoặc delete: cho phép cập nhật/xoá dữ liệu (dữ liệu thật sự trong CSDL) tương ứng với vị trí hiện hành của cursor.

2. Khai báo và sử dụng Cursor

Khai báo Cursor

Có thể sử dụng cú pháp chuẩn SQL 92 hoặc cú pháp T_SQL mở rộng.

– Cú pháp SQL 92 chuẩn:

Declare *cursor_name* [Insensitive] [Scroll] Cursor

For *select_statement*

[For {Read only| Update [of *column_name* [,…n]] }]

– Cú pháp T_SQL mở rộng

Declare *cursor_name* Cursor

[Local | Global]

[Forward_only| Scroll]

[Static| Dynamic]

[Read_only]

For select_statement

[For Update [of column_name [,…n]]]

Lưu ý: *Tên cursor trong các cách khai báo không bắt đầu bằng ký tự "@".*

Ý nghĩa các tham số tùy chọn trong khai báo:

- *Insensitive/ static:* nội dung của cursor không thay đổi trong suốt thời gian tồn tại, trong trường hợp này cursor chỉ là read only.
- *Dynamic:* trong thời gian tồn tại, nội dung của cursor có thể thay đổi nếu dữ liệu trong các bảng liên quan có thay đổi.
- *Local:* cursor cục bộ, chỉ có thể sử dụng trong phạm vi một khối (query batch) hoặc một thủ tục/ hàm.
- *Global:* cursor toàn cục, có thể sử dụng trong một thủ tục/hàm hay một query batch bất kỳ hoặc đến khi bị hủy một cách tường minh.
- *Forward_only:* cursor chỉ có thể duyệt một chiều từ đầu đến cuối.
- *Scroll:* có thể duyệt lên xuống cursor tùy ý (duyệt theo đa chiều).

- *Read only:* chỉ có thể đọc từ cursor, không thể sử dụng cursor để update dữ liệu trong các bảng liên quan (ngược lại với "for update…").

Mặc định khi khai báo cursor nếu không chỉ ra các tùy chọn thì cursor có các tính chất:

- Global
- Forward_only
- Read only hay "for update" tùy thuộc vào câu truy vấn
- Dynamic

Duyệt cursor

Dùng lệnh Fetch để duyệt tuần tự qua cursor theo cú pháp:

Fetch

[[Next| Prior| First| Last| Absolute n| Relative n]

From] Tên_cursor

[Into Tên_biến [,…n]]

- Mặc định: fetch next.
- Đối với cursor dạng forward_only, chỉ có thể fetch next.
- Biến hệ thống @@fetch_status cho biết lệnh fetch vừa thực hiện có thành công hay không, giá trị của biến này cơ sở để biết đã duyệt đến cuối cursor hay chưa.

Quy trình sử dụng Cursor

- Khai báo cursor.
- "Mở" cursor bằng lệnh *Open*

  Open *tên_cursor*

- Khai báo các biến tạm để chứa phần tử hiện hành (đang được xử lý) của cursor:
  - ✓ Các biến tạm phải cùng kiểu dữ liệu với các trường tương ứng của phần tử trong cursor.
  - ✓ Có n trường trong phần tử của cursor thì phải có đủ n biến tạm tương ứng.
- Fetch (next,…) cursor để chuyển đến vị trí phù hợp:
  - ✓ Có thể đưa các giá trị của dòng hiện hành vào các biến thông qua mệnh đề into của lệnh fetch.
  - ✓ Nếu không có mệnh đề into, các giá trị của dòng hiện hành sẽ được hiển thị ra cửa sổ kết quả (result pane) sau lệnh fetch.
  - ✓ Có thể sử dụng vị trí hiện tại như là điều kiện cho mệnh đề where của câu delete/ update (nếu cursor không là read_only).

– Lặp lại việc duyệt và sử dụng cursor, có thể sử dụng biến @@fetch_status để biết đã duyệt qua hết cursor hay chưa. **@@FETCH_STATUS = 0 : lấy dữ liệu thành công, @@FETCH_STATUS < 0 : không lấy được dữ liệu.**

– Đóng cursor bằng lệnh Close

Close *Tên_cursor*

Lưu ý: *Sau khi đóng, vẫn có thể mở lại nếu cursor chưa bị hủy.*

– Hủy cursor bằng lệnh deallocate

Deallocate *Tên_cursor*

Ví dụ: xét hai LĐQH

SINHVIEN (MaSV, HoTen, MaKhoa)

KHOA(MaKhoa, TenKhoa)

- Duyệt và đọc giá trị từ cursor

Cập nhật lại giá trị MaSV = Viết tắt tên Khoa + MaSV hiện tại cho tất cả sinh viên:

```
declare cur_DSKhoa cursor
for select MaKhoa, TenKhoa from Khoa
open cur_DSKhoa
declare @MaKhoa int,
        @TenKhoa varchar(30), @TenTat varchar(5)
fetch next from cur_DSKhoa into @MaKhoa, @TenKhoa
while @@fetch_status = 0
begin
    -- xác định tên tắt của Khoa dựa vào @TenKhoa…
    update   SinhVien set MaSV = @TenTat+MaSV
        Where MaKhoa = @MaKhoa
    fetch next from cur_DSKhoa into @MaKhoa, @TenKhoa
end
Close  cur_DSKhoa
Deallocate cur_DSKhoa
```

- Dùng cursor để xác định dòng cập nhật

```
declare cur_DSKhoa cursor scroll
for select MaKhoa, TenKhoa from Khoa
open cur_DSKhoa
fetch absolute 2 from cur_DSKhoa
if (@@fetch_status = 0)
        update Khoa
                set TenKhoa = 'aaa'
                where current of cur_DSKhoa
Close  cur_DSKhoa
Deallocate cur_DSKhoa
```

3. Biến cursor

- Ta có thể khai báo một biến kiểu cursor và gán cho nó tham chiếu đến một cursor đang tồn tại.
- Biến cursor có thể được xem như là con trỏ cursor.
- Biến cursor là một biến cục bộ.
- Biến cursor sau khi gán giá trị được sử dụng như một cursor thông thường.

Ví dụ :

```
Declare @cur_var cursor
set @cur_var = my_cur      -- my_cur là một cursor đang tồn tại
```

hoặc:

```
Declare @cur_var cursor
set @cur_var = cursor for select_statement
```

Kết hợp cursor với stored procedure

Xây dựng SP tính điểm trung bình và xếp loại cho sinh viên thuộc lớp cho trước. Giả sử có các quan hệ như sau:

SinhVien (MaSV, HoTen, DTB, XepLoai, Lop)

MonHoc (MaMH, TenMH)

KetQua (MaMH, MaSV, LanThi, Diem)

Biết rằng

- ✓ Điểm thi chỉ tính lần thi sau cùng
- ✓ Xếp loại: Xuất sắc [9, 10], Giỏi [8, 8.9], Khá [7, 7.9], Trung bình [5.0, 6.9], Yếu [0, 4.9].
- ✓ Kết quả ghi xuống CSDL, đồng thời xuất ra tổng số sinh viên xếp loại giỏi của lớp đó.

- **Phân tích ví dụ:**

- Lớp cần xét có nhiều sinh viên, từng sinh viên cần được xử lý thông qua 3 bước:
  - ✓ Tính điểm trung bình cho sinh viên, điểm trung bình phải là điểm của lần thi sau cùng. Có thể tái sử dụng thủ tục XepLoaiSVLop.
  - ✓ Dựa vào điểm trung bình của sinh viên để xác định xếp loại.
  - ✓ Cập nhật điểm và xếp loại vào bảng sinh viên.
- Mọi sinh viên đều lặp lại 3 bước trên.

Từ phân tích trên ta thấy:

- ✓ Cần xử lý nhiều phần tử (các sinh viên).
- ✓ Mỗi phần tử xử lý tương đối phức tạp (truy vấn, tính toán, gọi thủ tục khác, điều kiện rẽ nhánh, cập nhật dữ liệu, …).
- ✓ Cách xử lý các phần tử là như nhau.

⇒ Sử dụng cursor là thích hợp

- ✓ Cursor chứa các sinh viên của lớp cần xét, chỉ cần chứa mã sinh viên là được.

- **Xây dựng thủ tục**

```
Create procedure  XepLoaiSVLop
    @Lop nvarchar(10), @SoSVGioi int out
As
    Declare @DTB float
    Declare @XepLoai nvarchar(20)
    Declare @MaSV nvarchar(10)
    Declare cur_SV cursor
    For  (select MaSV from SinhVien where Lop=@Lop)
    Open cur_SV
    Fetch Next from cur_SV into @MaSV
    While  @@FETCH_STATUS = 0
    Begin
            Exec XepLoaiSV @MaSV, @DTB output, @XepLoai output
            Update SinhVien set DTB = @DTB, XepLoai=@XepLoai
            Where MaSV = @MaSV
            Fetch Next from cur_SV into @MaSV
    End
    Close cur_SV
    Deallocate cur_SV
```

```
Set @SoSVGioi = (select count(*) from sinhvien
                 where lop = @Lop and XepLoai = N'Giỏi')
Go
```

V. Hàm người dùng (User Defined Functions)

1. Khái niệm hàm người dùng

- Giống stored procedure:
  - mã lệnh có thể tái sử dụng.
  - Chấp nhận các tham số input.
  - Biên dịch một lần và từ đó có thể gọi khi cần.
- Khác stored procedure:
  - Chấp nhận nhiều kiểu giá trị trả về (chỉ một giá trị trả về).
  - Không chấp nhận tham số output.
  - Khác về cách gọi thực hiện.
- Có thể xem hàm người dùng thuộc về 3 loại tùy theo giá trị trả về của nó:
  - Giá trị trả về là kiểu dữ liệu cơ sở (int, varchar, float, datetime…).
  - Giá trị trả về là Table có được từ một câu truy vấn.
  - Giá trị trả về là table mà dữ liệu có được nhờ tích lũy dần sau một chuỗi thao tác xử lý và insert.

2. Khai báo và sử dụng

Khai báo hàm người dùng

Loại 1: Giá trị trả về là kiểu dữ liệu cơ sở

```
Create function func_name
( {parameter_name  DataType [= default ] } [,…n])
returns DataType
As
Begin
...
Return {value | variable | expression}
End
```

Ví dụ:

```
Create function SoLonNhat
(@a int,@b int,@c int) return int
As
Begin
        declare @max int
        set @max = @a
        if @b > max  set  @max = @b
        if @c > max  set  @max = @c
        return @max
End
```

Loại 2: Giá trị trả về là một bảng có được từ một câu truy vấn

```
Create function func_name
        ( {parameter_name  DataType [= default ] } [,…n])
        returns Table
As
Return [ ( ]select_statement [ ) ]
Go
```

Ví dụ: Viết hàm in danh sách các mặt hàng của một mã đơn hàng cho trước

```
Create function DanhSachMatHang
( @MaDonHang varchar(10) ) returns Table
As
Return
(Select MH.TenHang,MH.DonGia
From ChiTietDH CT, MatHang MH
Where CT.MaDH = @MaDonHang
and CT.MaMH = MH.MaMH)
Go
```

Loại 3: Giá trị trả về là một bảng mà dữ liệu có được nhờ tích lũy dần sau một chuỗi thao tác xử lý và insert.

```
Create function func_name
( {parameter_name  DataType [= default ] } [,…n])
returns TempTab_name Table(Table_definition)
As
Begin
...
Return
End
Go
```

Ví dụ:

```
Create function DanhSachLop ()
returns @DS  Table(@MaLop varchar(10),@SoSV int)
As
--các xử lý insert dữ liệu vào bảng DS
return
Go
```

Lưu ý: *Trong thân hàm không được sử dụng các hàm hệ thống bất định (Built-in nondeterministic functions), bao gồm :*

- GETDATE
- GETUTCDATE
- NEWID
- RAND
- TEXTPTR
- @@TOTAL_ERRORS, @@CPU_BUSY, @@TOTAL_READ, @@IDLE, @@TOTAL_WRITE, @@CONNECTIONS …

Sử dụng hàm người dùng

Các hàm người dùng được sử dụng trong câu truy vấn, trong biểu thức… phù hợp kiểu dữ liệu trả về của nó.

Ví dụ:

- Select dbo.SoLonNhat(3,5,7)
- Select * from DanhSachLop()

Lưu ý:

- *Nếu dùng giá trị mặc định của tham số, phải dùng từ khóa default.*

– *Khi gọi hàm loại 1 (trả về giá trị cơ bản), phải có tên owner của hàm đi kèm (ví dụ dbo.SoLonNhat(5,8,-10)).*

Thay đổi hàm người dùng

Thay từ khóa create trong các lệnh tạo hàm bằng từ khóa alter

Xóa hàm người dùng

Drop function *tên_hàm_cần _xóa*

Ví dụ:

Drop function DanhSachMatHang

3. Các hàm hệ thống

Ngoài các hàm do người dùng định nghĩa, SQL Server còn cung cấp các hàm xây dựng sẵn của hệ thống. Các hàm này cung cấp tiện ích như xử lý chuỗi, xử lý thời gian, xử lý số học…

Sinh viên tìm hiểu thêm về các hàm này trong Books on-line và các tài liệu tham khảo.

- Để tạo hàm hệ thống cần tiến hành theo các bước sau:
  - ✓ Tạo hàm trong cơ sở dữ liệu **Master**
  - ✓ Tên hàm bắt đầu bởi **fn*_functionName***
  - ✓ Thay đổi chủ nhân của hàm bằng thủ tục sp_changeobjectowner như sau:

    EXEC sp_changeobjectowner ‘fn_*functionName*’ , ‘system_function_schema’

Ví dụ: *Tạo hàm hệ thống thực chuyển đổi một biến kiểu ngày tháng sang kiểu chuỗi.*

```
--Tạo hàm fn_doingay
        create function fn_doingay(@ngay datetime)
        returns char(10)
        as
        begin
                return convert(nchar(10),@ngay,103)
        end
--Thay đổi chủ nhân của hàm
        EXEC sp_changeobjectowner 'fn_doingay' , 'system_function_schema'
```

Sau lệnh này hàm fn_doingay có thể dùng được cho CSDL bất kỳ.

```
        Select manv, hoten, fn_doingay(ngaysinh)
        From nhanvien
```

VI. Triggers và cài đặt ràng buộc dữ liệu

1. Giới thiệu

- Trigger là một loại stored procedure đặc biệt có các đặc điểm sau:

- Tự động thực hiện khi có thao tác insert, delete hoặc update trên dữ liệu.
- Thường dùng để kiểm tra các ràng buộc toàn vẹn của CSDL hoặc các qui tắc nghiệp vụ.
- Một trigger được định nghĩa trên một bảng, nhưng các xử lý trong trigger có thể sử dụng nhiều bảng khác.

• Xử lý của trigger thường cần sử dụng đến hai bảng tạm:

- **Inserted**: chứa các dòng vừa mới được thao tác insert/ update thêm vào bảng.
- **Deleted**: chứa các dòng vừa mới bị xóa khỏi bảng bởi thao tác update/delete.

Lưu ý: *update = delete dòng chứa giá trị cũ + insert dòng chứa giá trị mới*

- Inserted và deleted là các bảng trong bộ nhớ chính:
  - Cục bộ cho mỗi trigger.
  - Có cấu trúc giống như bảng (table) mà trigger định nghĩa trên đó
  - Chỉ tồn tại trong thời gian trigger đang xử lý.
- Nếu thao tác insert/ delete/ update thực hiện trên nhiều dòng, trigger cũng chỉ được gọi một lần →Bảng inserted/ deleted có thể chứa nhiều dòng.

2. Sử dụng Trigger

Khai báo trigger

- Cú pháp:

```
Create trigger tên_trigger
On {tên_bảng|tên_view}
{For| After| Instead of } { [delete] [,] [insert] [,] [update] }
As
                { các lệnh T-sql }
Go
```

rong đó:

For | After:

- Trigger được gọi thực hiện sau khi thao tác delete/ insert/ update tương ứng đã được thực hiện thành công:
  - ✓ Các dòng mới được thêm chứa đồng thời trong bảng dữ liệu và bảng inserted.
  - ✓ Các dòng bị xoá chỉ nằm trong bảng deleted (đã bị xoá khỏi bảng dữ liệu).
- Có thể xử lý quay lui thao tác đã thực hiện bằng lệnh *rollback transaction.*

Instead of:

- Trigger được gọi thực hiện thay cho thao tác delete/ insert/ update tương ứng:

- ✓ Các dòng mới được thêm chỉ chứa trong bảng inserted.
- ✓ Các dòng bị chỉ định xoá nằm đồng thời trong bảng deleted và bảng dữ liệu (dữ liệu không bị xoá).

– Trigger *Instead of* thường được dùng để xử lý cập nhật trên khung nhìn.

Lưu ý:

– Lệnh tạo trigger phải là lệnh đầu tiên trong một *query batch.*

– Trên một bảng có thể định nghĩa nhiều trigger *for/after* cho mỗi thao tác nhưng chỉ có thể định nghĩa một trigger *instead of* cho mỗi thao tác.

– Không thể định nghĩa trigger *instead of update/ delete* trên bảng có cài đặt khóa ngoại dạng *update cascade/ delete cascade.*

– Trong thân trigger, có thể sử dụng hàm *Update(tên_cột)* để kiểm tra xem việc cập nhật được thực hiện trên cột nào.

Update(*tên_cột*) = *true* : có thực hiện cập nhật trên cột *tên_cột*

Sử dụng trigger cài đặt một số loại ràng buộc

Ví dụ 1- Ràng buộc liên thuộc tính – liên quan hệ

Cho CSDL:

DatHang(MaPDH, NgayDH,…)

GiaoHang(MaPGH, MaPDH, NgayGH,…)

Ràng buộc: Ngày giao hàng không thể nhỏ hơn ngày đặt hàng tương ứng

Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
DatHang	-	-	+ (NgayDH)
GiaoHang	+	-	+ (NgayGH, MaPDH)

→Cần cài đặt trigger cho thao tác sửa trên bảng DatHang, và thêm/sửa trên bảng Giaohang

Trigger cho thao tác thêm và sửa trên giao hàng:

```
Create trigger tr_GH_ins_upd_NgayGH
On GIAOHANG  for insert, update
As
if  update(MaPDH) or update (NgayGH)
        if exists(select * from inserted i, DatHang d
                where i.MaPDH = d.MaPDH
```

```
                    and i.NgayGH<d.NgayDH)
        begin
            raiserror (N'Ngày GH không thể nhỏ hơn ngày ĐH',0,1)
            rollback tran
        end
go
```

Bài tập: Trigger cho thao tác sửa trên đặt hàng.

Ví dụ 2 – Ràng buộc toàn vẹn liên bộ

Xét LĐQH: KetQua(MASV, MAMH, LANTHI, DIEM)

Tạo trigger kiểm tra RBTV: "Sinh viên chỉ được thi tối đa 2 lần cho một môn học"

- Xác định bảng tầm ảnh hưởng:

	Insert	Delete	Update
KetQua	+	-	+ (MASV, MAMH)

- Cài đặt trigger dựa trên bảng tầm ảnh hưởng:

```
-- Tao trigger ung voi thao tac insert tren bang KetQua
Create trigger trg_KetQua_insert
on KetQua
for insert
as
    declare @SoLanThi int
    select @SoLanThi=count(*)
    from KetQua K, inserted I
    where I.MaSV=K.MaSV and I.MaMH = K.MaMH
    if @SoLanThi > 2
    begin
            raiserror('So lan thi phai <= 2', 0,1)
            rollback transaction
    end
```

Bài tập: Tạo trigger ứng với thao tac Update trên bảng KetQua.

Ví dụ 3: Trigger cho việc thực hiện một thao tác cập nhật dữ liệu nào đó.

Cho quan hệ

CHI_TIET_HOA_DON(**MaHD,STT**, MaMH, SoLuong, DonGia, ThanhTien).

- Viết trigger thực hiện cập nhật giá trị của ThanhTien khi thêm một chi tiết hóa đơn

```
CREATE TRIGGER trgTINHTIEN_CTHD
ON CHI_TIET_HOA_DON
FOR INSERT ,UPDATE
AS
BEGIN
    Update CHI_TIET_HOA_DON
    Set ThanhTien=SoLuong*DonGia
    Where exists (SELECT *
                  FROM INSERTED I
                  WHERE CHI_TIET_HOA_DON.MAHD = I.MAHD and
                        CHI_TIET_HOA_DON.STT = I.STT)
END
```

Chương 4
BẢO MẬT VÀ AN TOÀN DỮ LIỆU

I. Bảo mật trong hệ quản trị cơ sở dữ liệu

1. Khái niệm cơ bản về bảo mật

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

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

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

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

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

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

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

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

2. Lựa chọn bảo mật

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

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

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

- SQL Server cung cấp các chức năng hay các thủ tục tạo mới và quản trị người dùng CSDL SQL Server như sau:
  - ✓ Sử dụng thủ tục sp_addlogin.
  - ✓ Sử dụng công cụ Enterprise Manager.

- Sử dụng thủ tục sp_addlogin

  Cú pháp

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

trong đó các tham số có ý nghĩa như sau:

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

Ví dụ: tạo người dùng có tên ‘nam’, mật khẩu ‘123’, cơ sở dữ liệu mặc định ‘QLKyNang’

Exec sp_addlogin ‘nam’, ‘123’, ‘QLKyNang’

- Thay đổi mật khẩu

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

3. Quyền người dùng và quản trị quyền người dùng

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

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

Cấp phát quyền truy cập vào CSDL

Cú pháp:

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

Các tham số:

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

Loại bỏ quyền truy cập vào CSDL db_name của người dùng

Use *db_name*
Go

sp_revokedbaccess [@loginame =] '*login*'

Cấp phát quyền thực thi trên cơ sở dữ liệu

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

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

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

Thủ tục cấp quyền GRANT

GRANT ALL | < PERMISSION> [, …n]

ON

<table hoặc view name> [(<column name> [, … n])]

| <stored hoặc extension procedure name>

TO

<login or role name> [, … n]

[WITH GRANT OPTION]

[AS <role name>]

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

Từ chối quyền truy vấn và xử lý dữ liệu

DENY ALL | < PERMISSION> [, …n]

ON

<table hoặc view name> [(<column name> [, … n])]

| <stored hoặc extension procedure name>

TO

<login or role name> [, … n]

[Cascade]

Loại bỏ quyền truy vấn và xử lý dữ liệu

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

Quyền tạo đối tượng trong CSDL

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

Các quyền tạo các đối tượng như sau:

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

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

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

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

Vai trò trên SQL Server:

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

Vai trò trên CSDL:

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

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

- *Tạo một role*

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

  Trong đó:

@rolename: tên role mới

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

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

- *Thêm người sử dụng vào Role*

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

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

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

II. Bản sao dữ liệu

Phần này sẽ giới thiệu kỹ thuật làm giảm lưu lượng dữ liệu giao dịch với SQL Server khi đã cấu hình nhiều Server trên mạng.

1. Giới thiệu về nhân bản dữ liệu

Nhân bản dữ liệu (Replication) là công cụ được sử dụng để sao chép một hoặc nhiều CSDL đến một hoặc nhiều server (SQL Server) khác. Các Server được đặt trong mạng máy tính nội bộ (LAN). Người khai thác có thể thực hiện truy cập đến CSDL có trong Server chứa dữ liệu được nhân bản. Dữ liệu giữa các máy được thực hiện đồng bộ với nhau theo lịch hoặc theo sự kiện, khi có yêu cầu. Nhân bản dữ liệu có những ưu điểm sau:

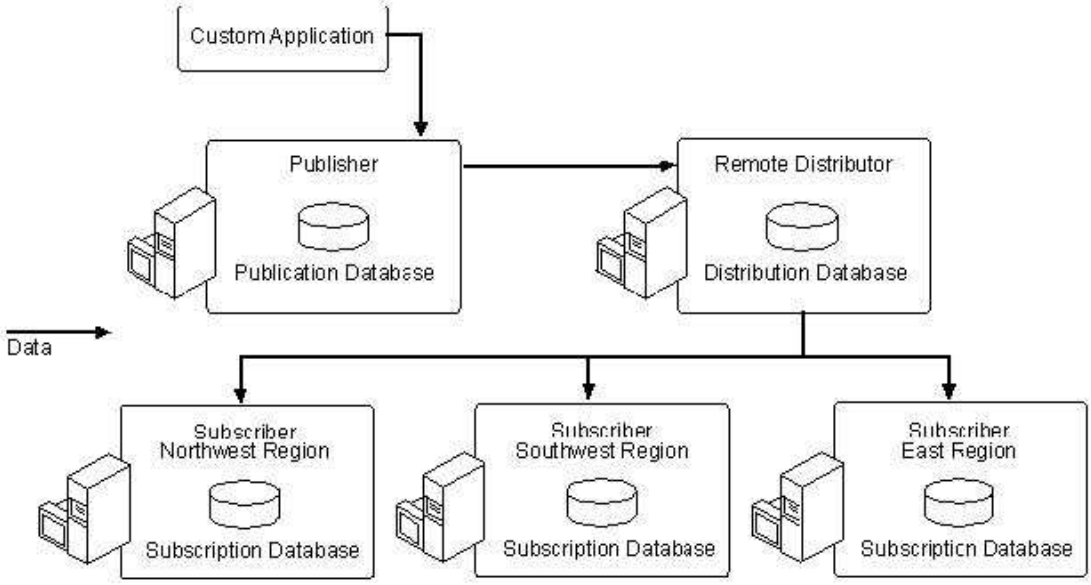
- Dữ liệu được lưu trữ ở nhiều nơi, hiệu quả trong việc có nhiều ứng dụng cùng truy cập, khai thác.
- Thích hợp với các ứng dụng phân tích dữ liệu của hệ thống xử lý giao dịch trực tuyến (OLTP) trong kho dữ liệu (Data warehouse).
- Có thể khai thác dữ liệu khi không kết nối đến Server.
- Giảm thiểu xung đột do số lượng lớn các giao dịch trên mạng.
- Là một giải pháp an toàn khi Server bị lỗi hoặc bảo dưỡng.

Mô hình nhân bản.

Dịch vụ nhân bản dữ liệu gồm các thành phần cơ bản sau: Publisher, Distributor, Subscribers, Publications, Articles, Subscriptions.

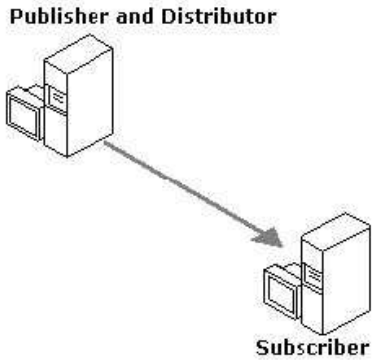
- *Publisher*: là server cung cấp dữ liệu nhân bản cho các server khác. Một publisher có thể thiết lập nhiều bộ dữ liệu nhân bản (gọi là publication).

- *Distributor*: là server quản lý các thông tin nhân bản, lưu trữ dữ liệu trong các giao dịch thực hiện nhận và chuyển dữ liệu từ Publisher đến các Subscriber. Remote distributor là server tách rời khỏi publisher và được cấu hình là distributor. Local distributor là một server được cấu hình là Publisher và Distributor.
- *Subscriber*: Là server nhận dữ liệu nhân bản. Subscriber gắn liền với publication (là máy chủ nhận dữ liệu nhân bản của một bộ dữ liệu cấu hình nhân bản).
- *Article*: Là một bảng, tập dữ liệu hoặc đối tượng của CSDL cấu hình để nhân bản.
- *Publication*: Là một tập gồm một hoặc nhiều article.
- *Subscription*: Là một giao dịch yêu cầu bản sao bộ dữ liệu hoặc các đối tượng của CSDL thực hiện nhân bản. Trong mỗi giao dịch publisher thực hiện đẩy (push subscription) dữ liệu, subscriber thực hiện kéo (pull subscription).

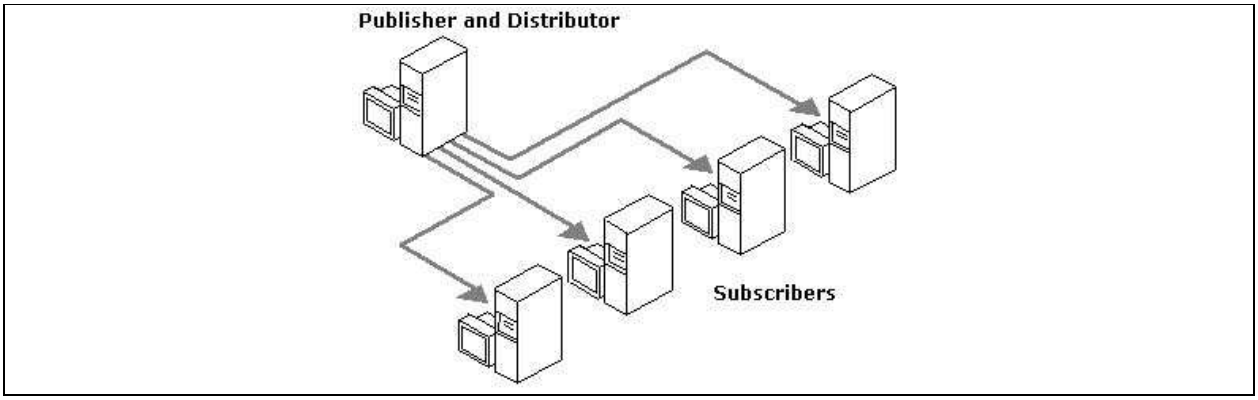


Nhân bản dữ liệu được thực hiện theo những mô hình cơ bản sau:

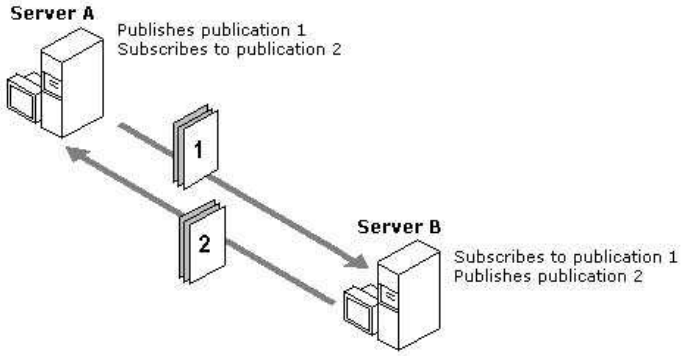
- *Central Publisher*: Là mô hình Publisher và Distributor thiết lập trên một máy. Gồm các mô hình sau:
  - ✓ Một Publisher và một Subscriber:



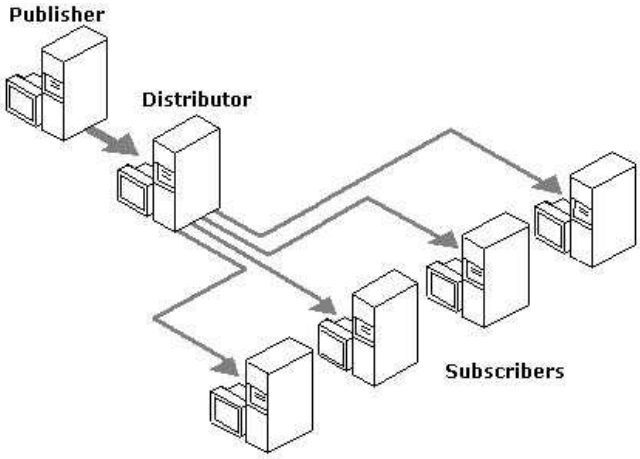
✓ Một Publisher và nhiều Subscriber.



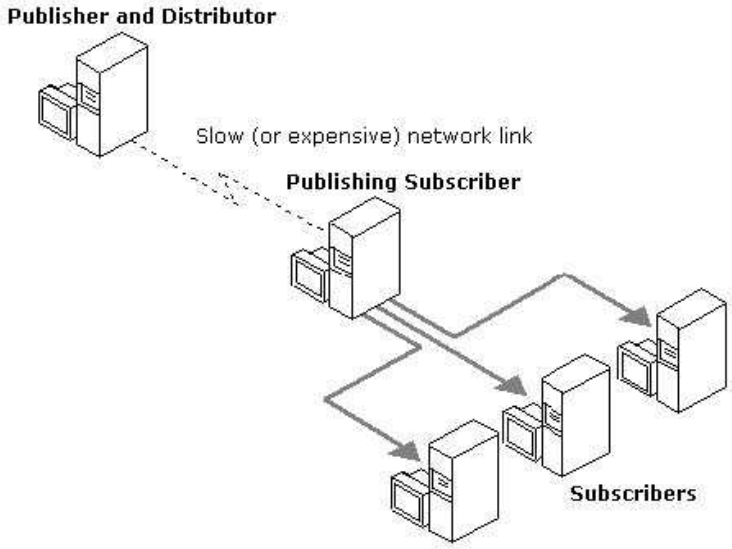
✓ Publisher và Subscriber được thiết lập trên một máy:



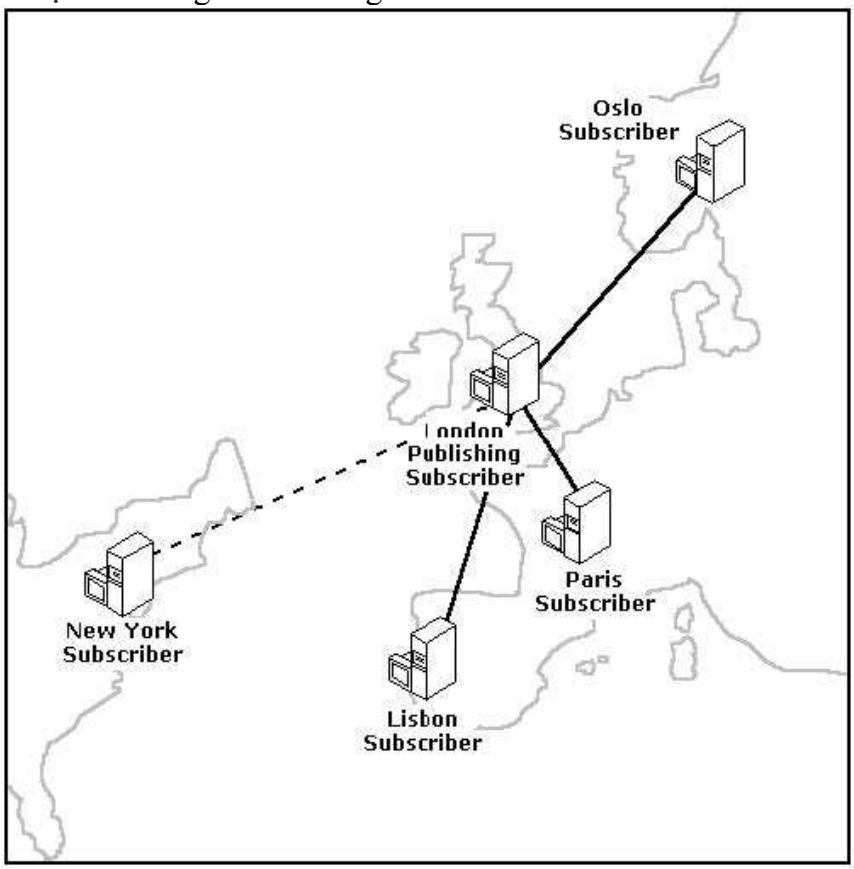
– *Publisher và Distributor không thiết lập trên một máy*:



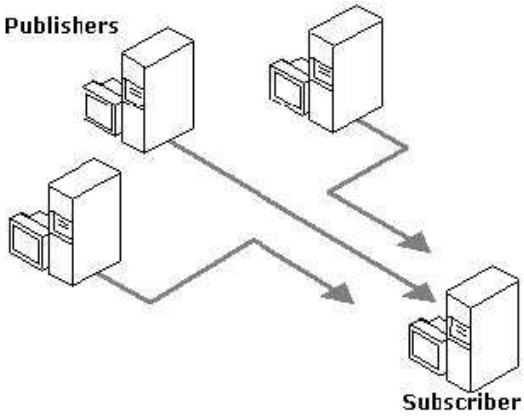
– *Republisher*: Là mô hình Publisher xuất bản dữ liệu đến Subscriber, sau đó Subscriber được thiết lập là Publisher xuất bản dữ liệu đến Subscriber khác.



Đường truyền giữa hai máy được thiết lập là Publisher có thể tốc độ thấp, phù hợp với vị trí xa nhau. Ví dụ mô hình giữa các vùng cách xa nhau:



+ *Central Subscriber*: Là mô hình Subscriber thiết lập nhận dữ liệu xuất bản từ nhiều Publisher.

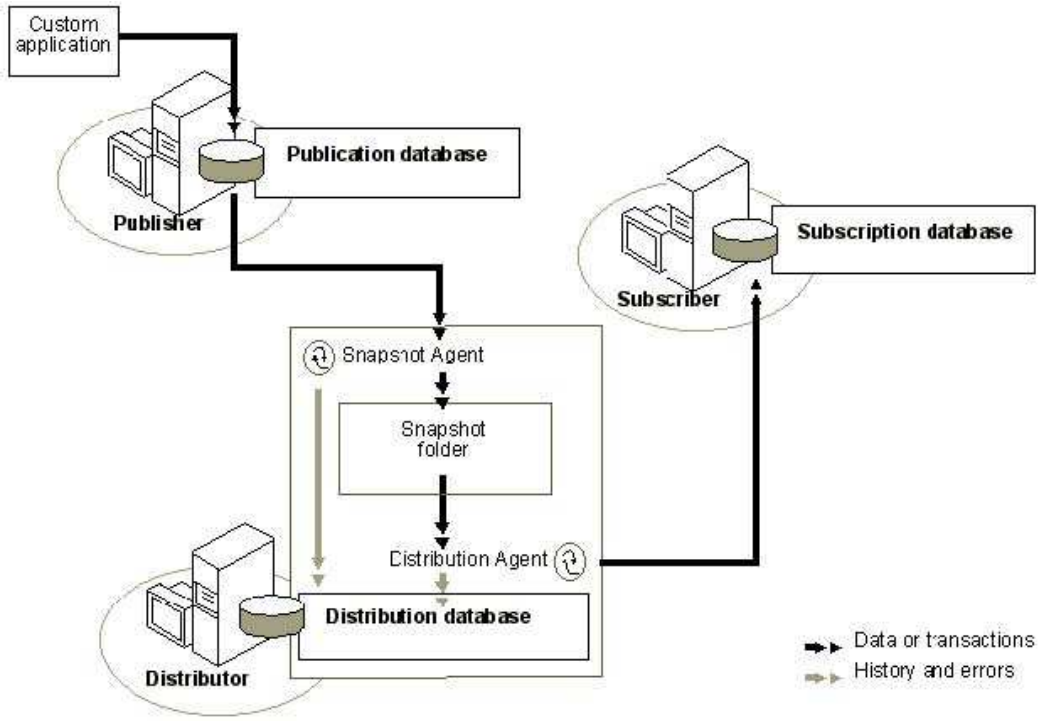


Những kiểu nhân bản dữ liệu.

Có 3 kiểu nhân bản dữ liệu: Snaphot, Transaction, Merge.

Snapshot replication: là kiểu nhân bản thực hiện sao chép, phân tán dữ liệu hoặc các đối tượng của CSDL tại một thời điểm. Snapshot thường được sử dụng cho những tình huống sau:

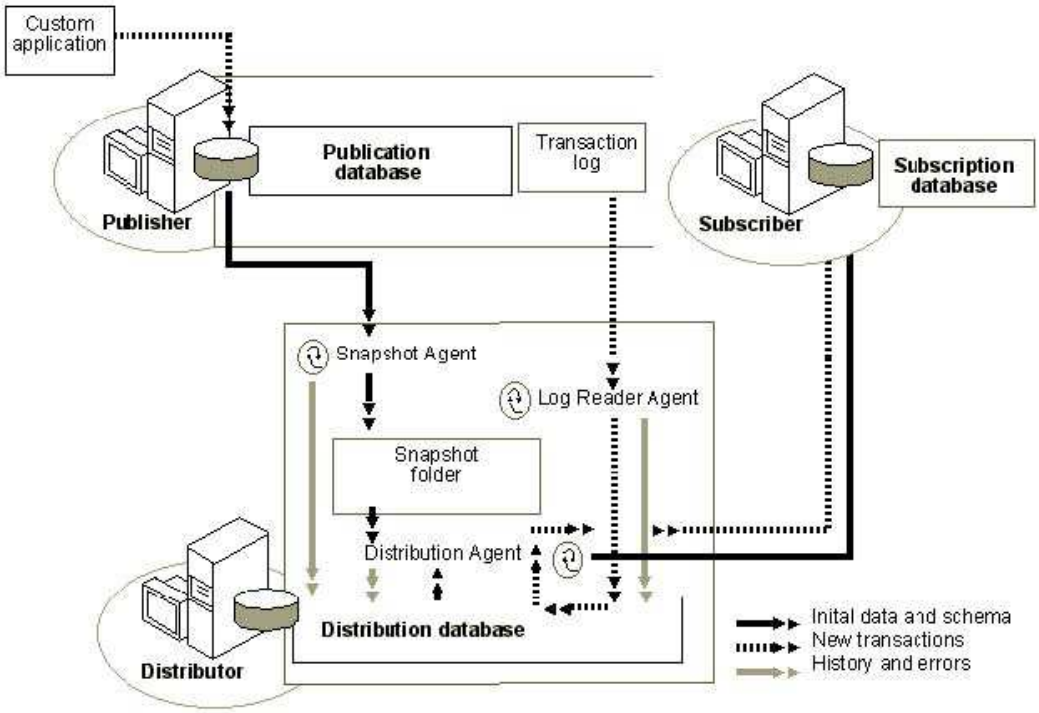
- ✓ Dữ liệu thường là tĩnh, ít thay đổi.
- ✓ Nhân bản số lượng dữ liệu nhỏ.



Transaction replication: là kiểu nhân bản mà bắt đầu bằng nhân bản Snapshot, sau đó sẽ thực hiện nhân giao dịch dữ liệu theo các sự kiện insert, update, delete và những thay đổi liên quan đến thực hiện thủ tục, khung nhìn chỉ mục. Nhân bản kiểu này cho phép thực hiện lọc dữ liệu tại Publisher, cho phép người sử dụng sửa đổi dữ liệu nhân bản tại

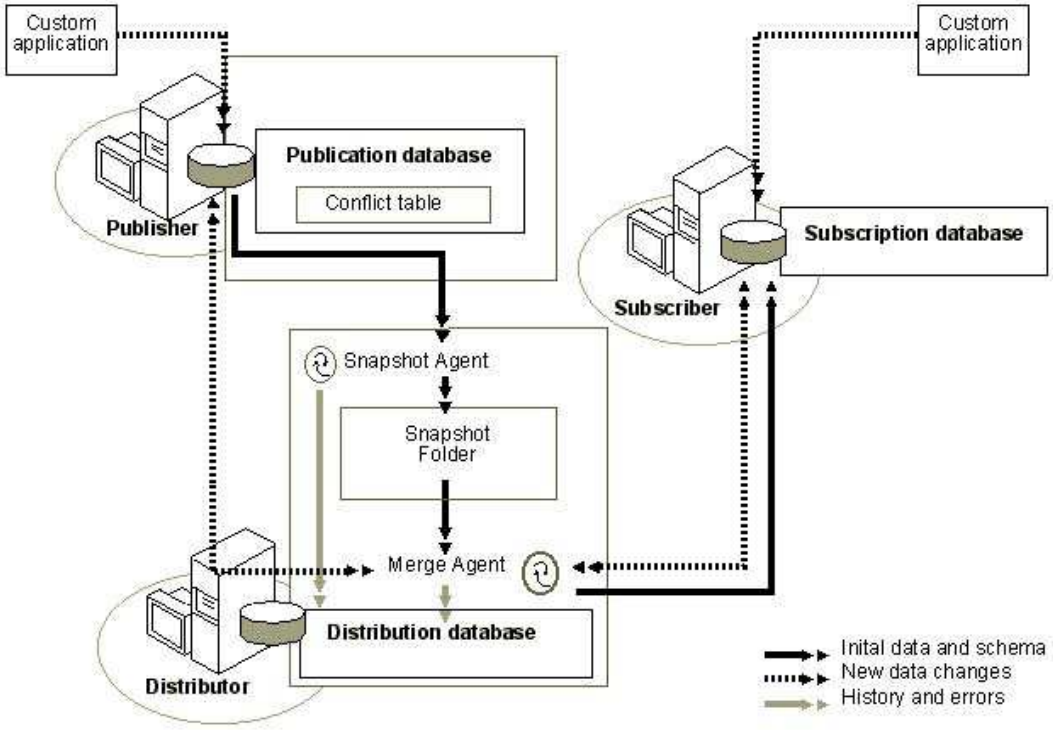
subscriber và chuyển dữ liệu đã sửa đổi đến Publisher hoặc Subscriber khác, dữ liệu sửa đổi này có thể coi là dữ liệu được xuất bản. Nhân bản kiểu này được thực hiện khi:

- ✓ Muốn sửa đổi dữ liệu được xuất bản chuyển đến Subscriber, thời gian thực hiện theo giây, hoặc tức thời.
- ✓ Cần giao dịch trên toàn bộ hệ thống nhân bản dữ liệu (dữ liệu có thể chuyển đến tất cả các Subscriber hoặc không chuyển đến Subscriber nào).
- ✓ Subscriber thường xuyên kết nối với Publisher.



Merge replication: là kiểu nhân bản dữ liệu cho phép thực hiện nhân bản sửa đổi dữ liệu trên nhiều Subscriber, có thể kết nối (online) hoặc không kết nối (offline) đến Publisher. Dữ liệu sẽ được đồng bộ theo lịch hoặc theo yêu cầu, dữ liệu cập nhật có thời điểm sau sẽ được chấp nhận. Kiểu nhân bản này thực hiện khi:

- ✓ Nhiều Subscriber có nhu cầu cập nhật dữ liệu và chuyển dữ liệu cập nhật đến Publisher hoặc Subscriber khác.
- ✓ Subscriber yêu cầu nhận hoặc chuyển dữ liệu khi offline, đồng bộ dữ liệu với các Subscriber và Publisher sau.

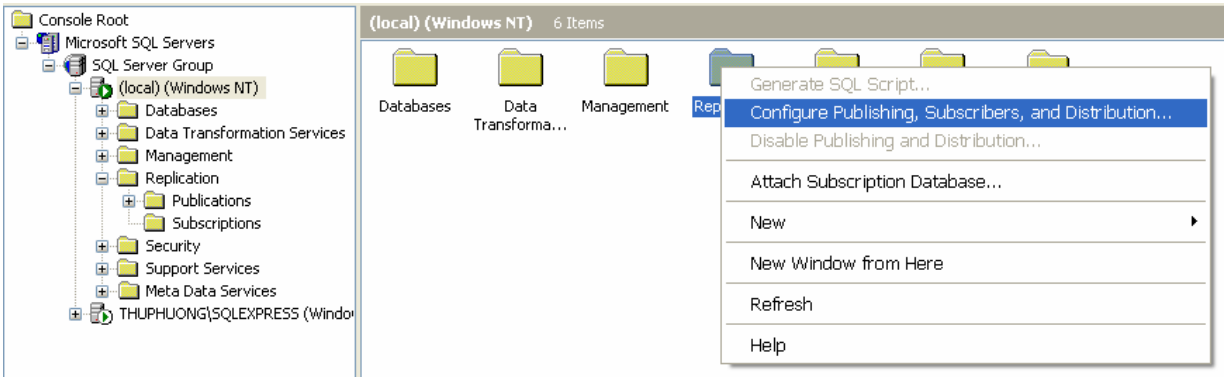


2. *Cấu hình Publisher và Distributor.*

Trước khi thực hiện cấu hình các máy thành Publisher hay Distributor ta phải thực hiện chạy dịch vụ SQL Server Agent trong chức năng Service manager. Các bước cấu hình như sau:

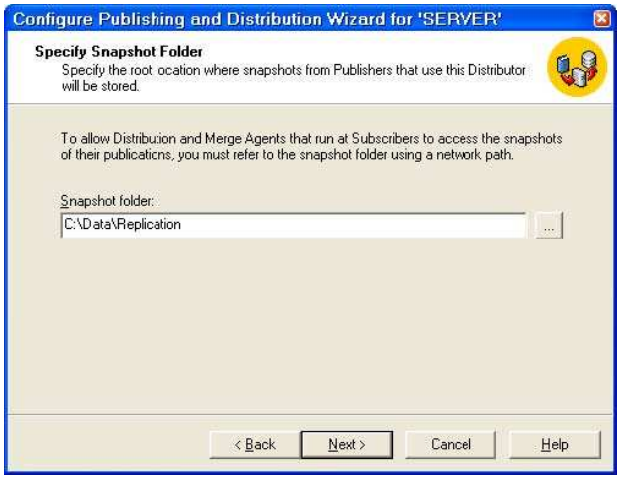
- Chọn Server cần cấu hình -> Replication

- Nhấp phải chuột -> Configure Publishing Subscription and Distribution...



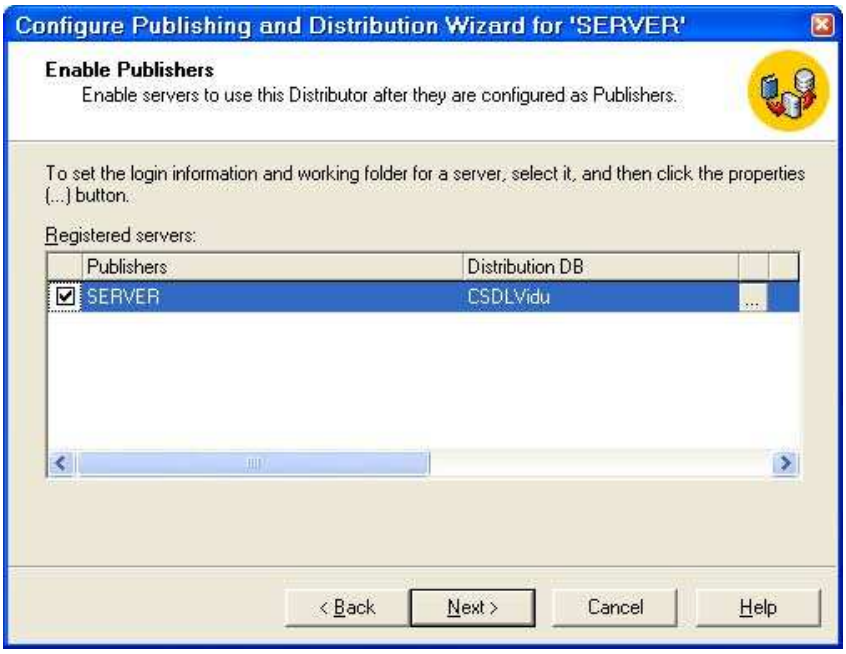
- Thực hiện theo các bước:

+ Chọn thư mục Snapshot

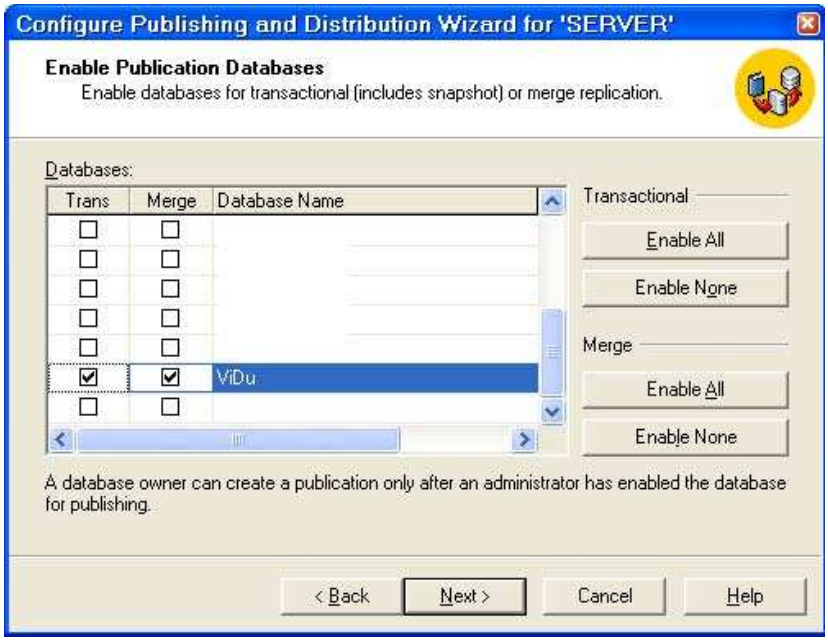


- Đặt tên CSDL của Distribution.

- Chọn Server cấu hình thành Publisher.



- Chọn CSDL tham gia nhân bản, kiểu nhân bản.



- Chọn Server được cấu hình là Subscriber của Publisher đang thiết lập.
- Kết thúc.

Tạo Publication

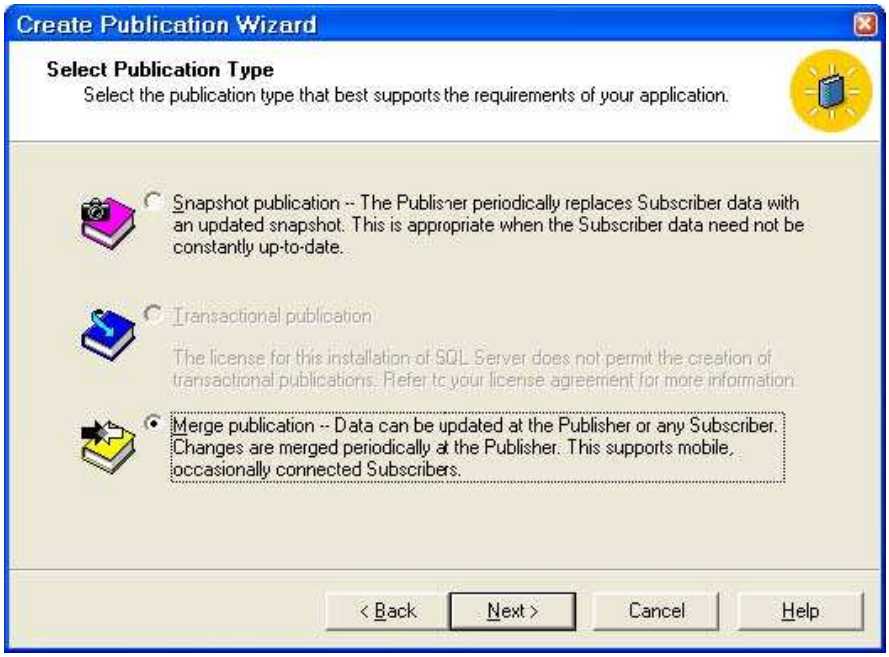
Bước này sẽ thực hiện tạo Publication, cách thực hiện như sau:

+ Chọn Publication trong Replication của Publisher.

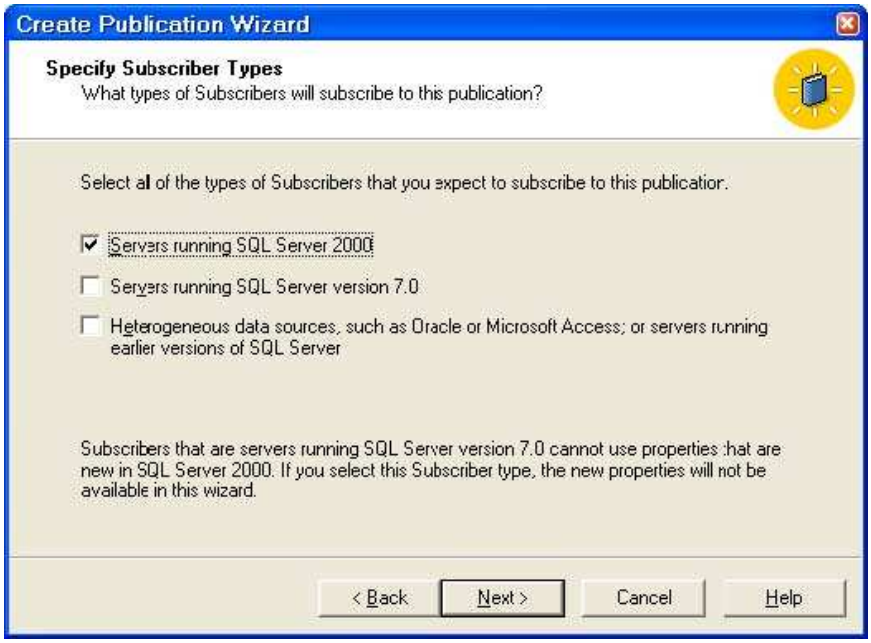
+ Nhấn phải chuột → chọn New Publication...

+ Thực hiện theo các bước:

- Chọn CSDL cần xuất bản dữ liệu hoặc đối tượng.
- Chọn kiểu nhân bản (trong ví dụ này thực hiện kiểu Merge).

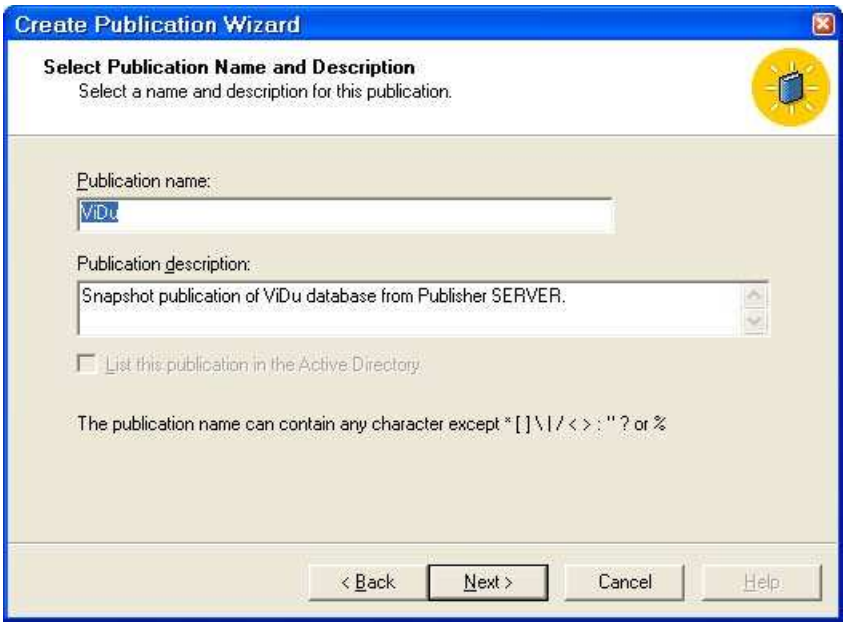


- Chọn phiên bản SQL Server của Subscriber.



- Chọn Article tham gia Publication.

- Đặt tên cho Publication.

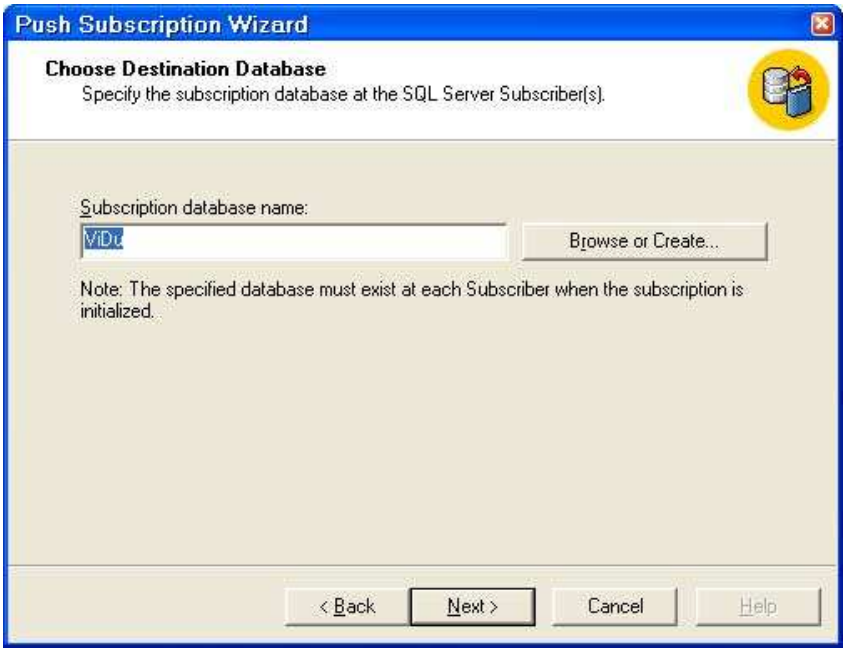


- Kết thúc.

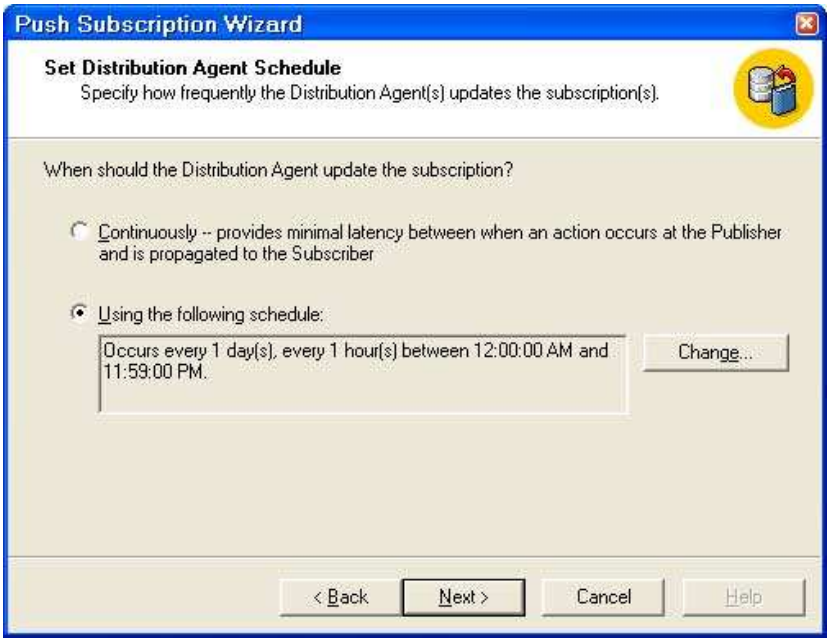
***Tạo Push Subscription*.**

Bước này thực hiện tạo thủ tục đẩy (push) từ Publisher (Distributor trong ví dụ này) đến Subscriber, được thực hiện trên Publisher. Các bước thực hiện như sau:

- Chọn Publication của Publisher -> Nhấn phải chuột -> Push new Subscription...
- Chọn Subscriber.
- Chọn CSDL trên Subscriber nếu đã có, nếu chưa có thực hiện chọn chức năng tạo mới.



- Chọn lịch thực hiện đồng bộ dữ liệu.
- Kết thúc. Sau khi thiết lập xong trên Subscriber sẽ có CSDL theo tên đã tạo.



Tạo Pull Subscription

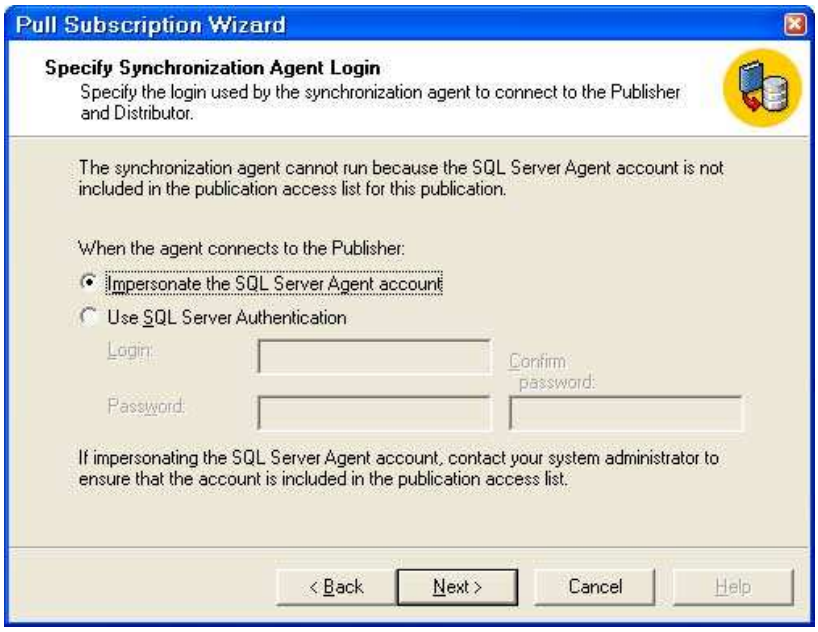
Bước này thực hiện tạo công cụ kéo dữ liệu nhân bản từ Publisher về Subscriber, được thực hiện trên Subscriber.

- Chọn Subscription của Subscriber → Nhấn phải chuột → New Pull Supcription...

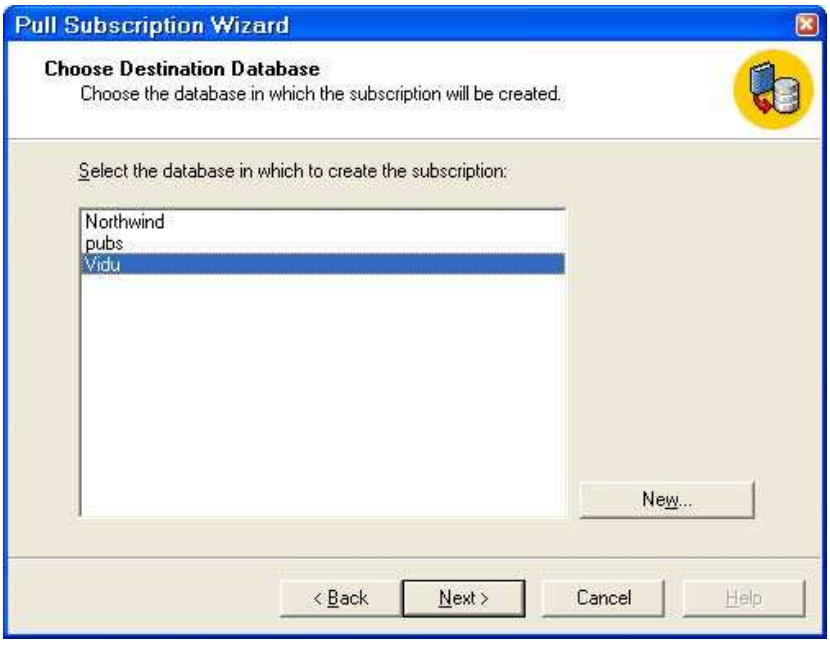
- Thực hiện theo các bước:

  + Chọn Publication.

  + Chọn Agent tham gia kết nối Publisher.



- Chọn CSDL đích.



- Thực hiện tiếp các bước và kết thúc. Nếu đã tạo Push Subscription với một CSDL sẽ không được tạo Pull Subscription với CSDL đó.

Thực hiện đồng bộ dữ liệu

Sau khi thiết lập theo các mô hình nhân bản xong, có thể thực hiện đồng bộ dữ liệu bằng cách:

- Thực hiện theo lịch.

- Theo yêu cầu: Chọn Subscription (Push hoặc Pull) → Nhấn phải chuột → Start Synchronizing.

Sau khi thực hiện xong dữ liệu sẽ được đồng bộ giữa Publisher và Subscriber. Ngoài thực hiện theo công cụ ta có thể tìm hiểu thực hiện nhân bản theo câu lệnh T-SQL hoặc Stored Procedure.

III. Sao lưu và khôi phục dữ liệu

1. Lý do phải sao lưu và khôi phục dữ liệu

Trong quá trình thực hiện quản trị CSDL SQL Server thì một số nguyên nhân sau đây bắt buộc bạn phải xem xét đến kỹ thuật sao lưu và khôi phục dữ liệu:

- ✓ Ổ đĩa bị hỏng (chứa các tập tin CSDL).
- ✓ Server bị hỏng.
- ✓ Nguyên nhân bên ngoài (thiên nhiên, hỏa hoạn, mất cắp,...).
- ✓ User vô tình xóa dữ liệu.
- ✓ Bị vô tình hay cố ý làm thông tin sai lệch.
- ✓ Bị hack.

2. Các loại sao lưu dữ liệu

Sao lưu (Backup) dữ liệu trong SQL Server gồm các loại sau:

- ✓ Full Database Backups: sao chép toàn bộ CSDL (các tập tin bao gồm các bảng, khung nhìn, các đối tượng khác).
- ✓ Differential Database Backups: sao chép những dữ liệu thay đổi trong Data file kể từ lần full backup gần nhất.
- ✓ File or file group backups: sao chép một file đơn hay file group.
- ✓ Differential File or File Group Backups: thực hiện như Differential Database nhưng copy phần dữ liệu thay đổi của file đơn hoặc file group.
- ✓ Transaction log backups: Ghi nhận tất cả các transaction chứa trong transaction log file kể từ lần transaction log backup gần nhất. Với loại sao lưu này ta có thể khôi phục dữ liệu tại một thời điểm.

3. Các mô hình khôi phục dữ liệu

- ✓ Full Recovery model: Là mô hình phục hồi toàn bộ hoạt động giao dịch của dữ liệu (Insert, Update, Delete, hoạt động bởi lệnh bcp, bulk insert). Với mô hình này ta có thể phục hồi dữ liệu tại một thời điểm trong quá khứ đã được lưu trong transaction log file.
- ✓ Bulk-Logged Recovery Model: Mô hình này được thực thi cho các thao tác bcp, bulk insert, create chỉ mục, writetext, updatetext, các hoạt động này chỉ nhật ký sự kiện vào log để biết mà không sao lưu toàn bộ dữ liệu, chi tiết như trong full recover. Các sự kiện Insert, Update, Delete vẫn được nhật ký và khôi phục bình thường.
- ✓ Simple Recovery Model: Với mô hình này bạn chỉ phục hồi lại thời điểm sao lưu gần nhất mà không theo thời điểm khác trong quá khứ.
- ✓ Cách đặt mô hình khôi phục:

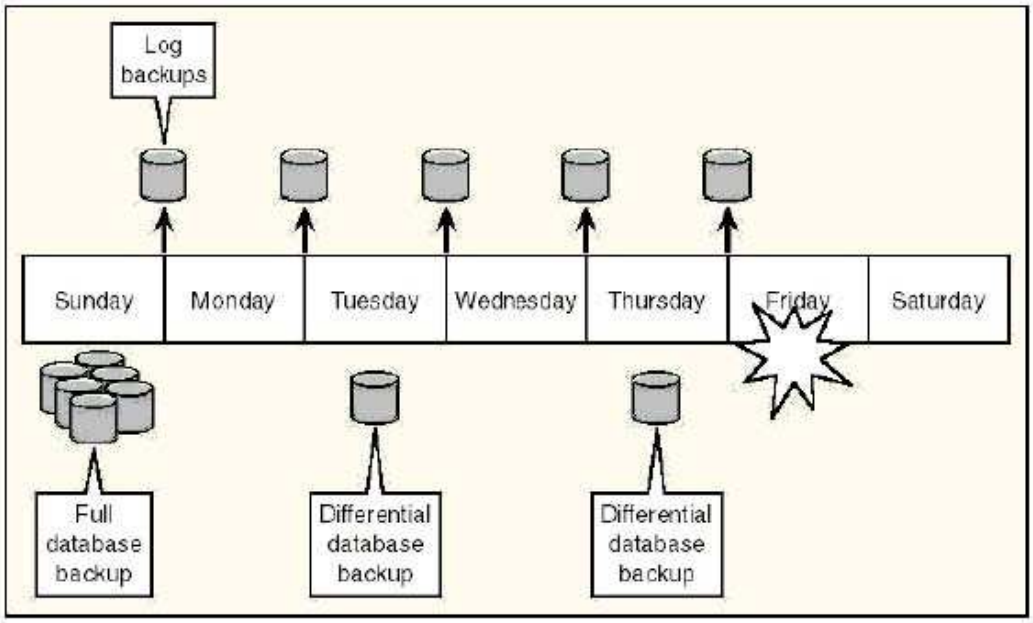
  - Chọn CSDL.

  - Nhấn nút phải chuột → Properties → Options → Recovery → chọn Model.

Xét ví dụ sau: Giả sử ta có một CSDL được backup theo chiến lược như hình vẽ bên dưới

Nhìn hình ta thấy CSDL được lập lịch Full Database Backup vào ngày chủ nhật, Differential Database Backup vào ngày thứ ba và thứ năm, còn Log Database Backup vào 5 ngày trong tuần, ngày thứ sáu có sự cố với CSDL, data file bị hỏng. Vấn đề đặt ra là phải phục hồi dữ liệu và CSDL hoạt động bình thường. Ta phải làm các bước sau:

- ✓ Thực hiện Backup log file (giả sử log file không bị hỏng).
- ✓ Khôi phục Full Database của ngày chủ nhật.
- ✓ Phục hồi Differential Database của ngày thứ năm.
- ✓ Khôi phục Transaction log backup ngày thứ năm.



4. Sao lưu cơ sở dữ liệu (Backup Database)

Trước khi xem xét kỹ thuật sao lưu CSDL, ta thống nhất một số thuật ngữ bằng tiếng Anh như sau:

- ✓ Backup: Là quá trình copy toàn bộ hoặc một phần database, transaction log, file, file group thành lập một backup set được chứa trong backup media (disk hoặc tape) bằng cách sử dụng một backup device (tape drive name hoặc physical filename).
- ✓ Backup Device: Một file vật lý hoặc một drive tape.
- ✓ Backup file: Một file chứa Backup set.
- ✓ Backup media: là Disk hoặc tape.
- ✓ Backup set: Một bộ backup một lần backup đơn chứa trên backup media.
- ✓ Các bước thực hiện backup như sau:
  - Chọn CSDL cần backup.
  - Nhấn phải chuột → All Tasks → Backup Database…
  - Nhập các tham số, lựa chọn kiểu.

5. Khôi phục dữ liệu (Restore Database)

Là chức năng thực hiện khôi phục dữ liệu đã sao lưu, tùy theo chiến lược backup mà người quản trị có thể phục hồi đến thời điểm nào, thu được bộ dữ liệu trong quá khứ như thế nào. Khôi phục dữ liệu được thực hiện theo thứ tự backup, thông tin này được lưu trữ trong msdb. Các bước thực hiện như sau:

- Chọn mục Databases → Nhấn nút phải chuột → All Tasks → Restore Database…
- Nhập tham số, chọn mô hình khôi phục.

IV. Quản lý giao dịch

1. Các khái niệm

Một giao dịch là một đơn vị thực hiện chương trình truy xuất và có thể cập nhật nhiều mục dữ liệu. Một giao dịch thường là kết quả của sự thực hiện một chương trình người dùng được viết trong một ngôn ngữ thao tác dữ liệu mức cao hoặc một ngôn ngữ lập trình (SQL, COBOL, PASCAL ...), và được phân cách bởi các câu lệnh (hoặc các lời gọi hàm) có dạng **begin transaction** và **end transaction**. Giao dịch bao gồm tất cả các hoạt động được thực hiện giữa begin và end transaction.

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

- **Tính nguyên tử (Atomicity)**. Hoặc toàn bộ các hoạt động của giao dịch được phản ánh đúng đắn trong CSDL hoặc không có gì cả.
- **Tính nhất quán (consistency)**. Sự thực hiện của một giao dịch là cô lập (Không

có giao dịch khác thực hiện đồng thời) để bảo tồn tính nhất quán của CSDL.

- **Tính cô lập (Isolation)**. Cho dù nhiều giao dịch có thể thực hiện đồng thời, hệ thống phải đảm bảo rằng đối với mỗi cặp giao dịch T_i, T_j , hoặc T_j kết thúc thực hiện trước khi T_i khởi động hoặc T_j bắt đầu sự thực hiện sau khi T_i kết thúc. Như vậy mỗi giao dịch không cần biết đến các giao dịch khác đang thực hiện đồng thời trong hệ thống.
- **Tính bền vững (Durability)**. Sau một giao dịch hoàn thành, các thay đổi đã được tạo ra đối với CSDL vẫn còn ngay cả khi xảy ra sự cố hệ thống.

Các tính chất này thường được gọi là các tính chất ACID (Các chữ cái đầu của bốn tính chất). Ta xét một ví dụ: một hệ thống ngân hàng gồm một số tài khoản và một tập các giao dịch truy xuất và cập nhật các tài khoản. Tại thời điểm hiện tại, ta giả thiết rằng CSDL nằm trên đĩa, nhưng một vài phần của nó đang nằm tạm thời trong bộ nhớ. Các truy xuất CSDL được thực hiện bởi hai hoạt động sau:

- **READ(X)**: chuyển mục dữ liệu X từ CSDL đến buffer của giao dịch thực hiện hoạt động **READ** này.
- **WRITE(X)**: chuyển mục dữ liệu X từ buffer của giao dịch thực hiện **WRITE** đến CSDL.

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

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

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

Ta xem xét mỗi yêu cầu trong ACID:

- **Tính nhất quán**: Đòi hỏi nhất quán ở đây là tổng của A và B là không thay đổi bởi sự thực hiện giao dịch. Nếu không có yêu cầu nhất quán, tiền có thể được tạo ra hay mất bởi giao dịch. Dễ dàng kiểm nghiệm rằng nếu CSDL nhất quán trước một thực hiện giao dịch, nó vẫn nhất quán sau khi thực hiện giao dịch. Đảm bảo tính nhất quán

cho một giao dịch là trách nhiệm của người lập trình ứng dụng người đã viết ra giao dịch. Nhiệm vụ này có thể được làm cho dễ dàng bởi kiểm thử tự động các ràng buộc toàn vẹn.

- **Tính nguyên tử**: giả sử rằng ngay trước khi thực hiện giao dịch T_i, giá trị của các tài khoản A và B tương ứng là 1000 và 2000. Giả sử rằng trong khi thực hiện giao dịch T_i, một sự cố xảy ra cản trở T_i hoàn tất thành công sự thực hiện của nó. Ta cũng giả sử rằng sự cố xảy ra sau khi hoạt động **WRITE**(A) đã được thực hiện, nhưng trước khi hoạt động **WRITE**(B) được thực hiện. Trong trường hợp này giá trị của tài khoản A và B là 950 và 2000. Ta đã mất 50$. Tổng A+B không còn được bảo toàn. Như vậy, kết quả của sự cố là trạng thái của hệ thống không còn phản ánh trạng thái của thế giới thực mà CSDL được giả thiết nắm giữ. Ta gọi trạng thái như vậy là trạng thái không nhất quán. Ta phải đảm bảo rằng tính không nhất quán này không xuất hiện trong một hệ CSDL. Tuy nhiên, ở tại một vài thời điểm, hệ thống có thể ở trong trạng thái không nhất quán. Ví dụ giao dịch T_i, trong quá trình thực hiện cũng tồn tại thời điểm tại đó giá trị của tài khoản A là 950 và tài khoản B là 2000 – một trạng thái không nhất quán. Trạng thái này được thay thế bởi trạng thái nhất quán khi giao dịch đã hoàn tất. Như vậy, nếu giao dịch không bao giờ khởi động hoặc được đảm bảo sẽ hoàn tất, trạng thái không nhất quán sẽ không bao giờ xảy ra. Đó chính là lý do có yêu cầu về tính nguyên tử: Nếu tính chất nguyên tử được đảm bảo, ***tất cả các hành động của giao dịch được phản ánh trong CSDL hoặc không có gì cả***. ý tưởng cơ sở để đảm bảo tính nguyên tử là như sau: hệ CSDL lưu vết (trên đĩa) các giá trị cũ của bất kỳ dữ liệu nào trên đó giao dịch đang thực hiện viết, nếu giao dịch không hoàn tất, giá trị cũ được khôi phục để đặt trạng thái của hệ thống trở lại trạng thái trước khi giao dịch diễn ra. Đảm bảo tính nguyên tử là trách nhiệm của hệ CSDL, và được quản lý bởi một thành phần được gọi là thành phần quản trị giao dịch (transaction-management component).

- **Tính bền vững**: tính chất bền vững đảm bảo rằng mỗi khi một giao dịch hoàn tất, tất cả các cập nhật đã thực hiện trên cơ sở dữ liệu vẫn còn đó, ngay cả khi xảy ra sự cố hệ thống sau khi giao dịch đã hoàn tất. Ta giả sử một sự cố hệ thống có thể gây ra việc mất dữ liệu trong bộ nhớ chính, nhưng dữ liệu trên đĩa thì không mất. Có thể đảm bảo tính bền vững bởi việc đảm bảo hoặc ***các cập nhật được thực hiện bởi giao dịch đã được viết lên đĩa trước khi giao dịch kết thúc*** hoặc ***thông tin về sự cập nhật được thực hiện bởi giao dịch và được viết lên đĩa đủ cho phép CSDL xây dựng lại các cập nhật khi hệ CSDL được khởi động lại sau sự cố***. Đảm bảo tính bền vững là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị phục hồi (recovery-management component). Hai thành phần quản trị giao dịch và quản trị phục hồi quan hệ mật thiết với nhau.

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

Trạng thái giao dịch

Nếu không có sự cố, tất cả các giao dịch đều hoàn tất thành công. Tuy nhiên, một giao dịch trong thực tế có thể không thể hoàn tất sự thực hiện của nó. Giao dịch như vậy được gọi là bị bỏ dở. Nếu ta đảm bảo được tính nguyên tử, một giao dịch bị bỏ dở không được phép làm ảnh hưởng tới trạng thái của CSDL. Như vậy, bất kỳ thay đổi nào từ giao dịch bị bỏ dở này đều phải bị hủy bỏ. Mỗi khi các thay đổi do giao dịch bị bỏ dở bị hủy bỏ, ta nói rằng giao dịch bị cuộn lại (rolled back). Khi một giao dịch hoàn tất một cách thành công sự thực hiện của nó được gọi là được bàn giao (committed). Một giao dịch được bàn giao, các lệnh cập nhật sẽ biến đổi CSDL sang một trạng thái nhất quán mới và nó là bền vững ngay cả khi có sự cố. Mỗi khi một giao dịch là được bàn giao, ta không thể hủy bỏ các hiệu quả của nó bằng cách bỏ dở nó. Cách duy nhất để hủy bỏ các hiệu quả của một giao dịch được bàn giao là thực hiện một giao dịch bù (compensating transaction), nhưng không phải luôn luôn có thể tạo ra một giao dịch bù. Do vậy trách nhiệm viết và thực hiện một giao dịch bù thuộc về người sử dụng và không được quản lý bởi hệ CSDL.

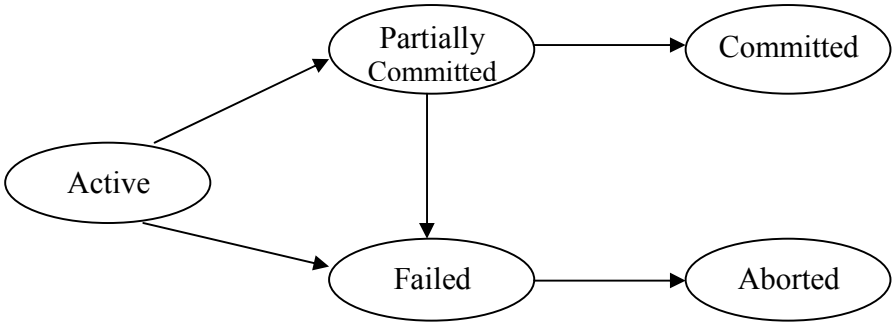
Một giao dịch phải ở trong một trong các trạng thái sau:

- **Hoạt động (Active)**: trạng thái khởi đầu; giao dịch giữ trong trạng thái này trong khi nó đang thực hiện.
- **Được bàn giao bộ phận (Partially Committed):** sau khi lệnh cuối cùng được thực hiện.

- **Thất bại (Failed):** sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- **Bỏ dở (Aborted):** sau khi giao dịch đã bị cuộn lại và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao dịch.
- **Được bàn giao (Committed):** sau khi hoàn thành thành công giao dịch.

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

Biểu đồ trạng thái tương ứng với một giao dịch như sau:



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

- ***Khởi động lại giao dịch***, dùng lựa chọn này chỉ nếu giao dịch bị bỏ dở là do lỗi phần cứng hoặc phần mềm nào đó không liên quan đến logic bên trong của giao dịch. Giao dịch được khởi động lại được xem là một giao dịch mới.
- ***Huỷ giao dịch*** thường được tiến hành hoặc do lỗi logic bên trong giao dịch, lỗi này cần được chỉnh sửa bởi viết lại chương trình ứng dụng hoặc do đầu vào xấu hoặc do dữ liệu mong muốn không tìm thấy trong CSDL.

Ta phải thận trọng khi thực hiện thao tác viết ngoài khả quan sát (observable external **Write** - như viết ra terminal hay máy in). Mỗi khi một viết như vậy xảy ra, nó không thể bị xoá do nó có thể phải giao tiếp với bên ngoài hệ CSDL. Hầu hết các hệ thống cho phép các thao tác viết như thế xảy ra chỉ khi giao dịch đã di vào trạng thái committed. Một cách để thực thi một sơ đồ như vậy là cho hệ CSDL lưu trữ tạm thời bất kỳ giá trị nào kết hợp với các thao tác viết ngoài như vậy trong lưu trữ không hay thay đổi và thực hiện các thao tác viết hiện tại chỉ sau khi giao dịch đã đi vào trạng thái committed. Nếu hệ thống thất bại sau khi giao dịch đi vào trạng thái committed nhưng trước khi hoàn tất các tháo tác viết ngoài, hệ CSDL sẽ làm các thao tác viết ngoài này (sử dụng dữ liệu trong lưu trữ không hay thay đổi) khi hệ thống khởi động lại.

Trong một số ứng dụng, có thể muốn cho phép giao dịch hoạt động trình bày dữ liệu cho người sử dụng, đặc biệt là các giao dịch kéo dài trong vài phút hay vài giờ. Ta không thể cho phép xuất ra dữ liệu khả quan sát như vậy trừ phi ta buộc phải làm tổn hại tính nguyên tử giao dịch. Hầu hết các hệ thống giao dịch hiện hành đảm bảo tính nguyên tử và do vậy cấm dạng trao đổi với người dùng này.

Thực thi tính nguyên tử và tính bền vững

Thành phần quản trị phục hồi của một hệ CSDL hỗ trợ tính nguyên tử và tính bền vững. Trước tiên ta xét một sơ đồ đơn giản (song cực kỳ thiếu hiệu quả). Sơ đồ này giả thiết rằng chỉ một giao dịch là hoạt động tại một thời điểm và được dựa trên việc tạo bản sao của CSDL được gọi là các bản sao khuất (shadow copies). Sơ đồ giả thiết rằng CSDL chỉ là một file trên đĩa. Một con trỏ được gọi là db_pointer được duy trì trên đĩa, nó trỏ tới bản sao hiện hành của CSDL.

Trong sơ đồ CSDL khuất (shadow-database), một giao dịch muốn cập nhật CSDL, đầu tiên tạo ra một bản sao đầy đủ của CSDL. Tất cả các thao tác cập nhật được làm trên bản sao này, không đụng tới bản gốc. Nếu tại một thời điểm bất kỳ giao dịch bị bỏ dở, bản sao mới bị xoá. Bản sao cũ của CSDL không bị ảnh hưởng. Nếu giao dịch hoàn tất, nó được được bàn giao như sau: đầu tiên, hỏi hệ điều hành để đảm bảo rằng tất cả các trang của bản sao mới đã được viết lên đĩa (flush). Sau khi flush con trỏ db_pointer được cập nhật để trỏ đến bản sao mới; bản sao mới trở thành bản sao hiện hành của CSDL. Bản sao cũ bị xoá đi. Giao dịch được gọi là đã được được bàn giao tại thời điểm sự cập nhật con trỏ db_pointer được ghi lên đĩa. Ta xét kỹ thuật này quản lý sự cố giao dịch và sự cố hệ thống ra sao? Trước tiên, ta xét sự cố giao dịch. Nếu giao dịch thất bại tại thời điểm bất kỳ trước khi con trỏ db_pointer được cập nhật, nội dung cũ của CSDL không bị ảnh hưởng. Ta có thể bỏ dở giao dịch bởi xoá bản sao mới. Mỗi khi giao dịch được được bàn giao, tất cả các cập nhật mà nó đã thực hiện là ở trong CSDL được trỏ bởi db_pointer. Như vậy, hoặc tất cả các cập nhật của giao dịch đã được phản ánh hoặc không kết quả nào được phản ánh,

bất chấp tới sự cố giao dịch. Bây giờ ta xét sự cố hệ thống. Giả sử sự cố hệ thống xảy ra tại thời điểm bất kỳ trước khi db_pointer đã được cập nhật được viết lên đĩa. Khi đó, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và như vậy sẽ thấy nội dung gốc của CSDL – không kết quả nào của giao dịch được nhìn thấy trên CSDL. Bây giờ lại giả sử rằng sự cố hệ thống xảy ra sau khi db_pointer đã được cập nhật lên đĩa. Trước khi con trỏ được cập nhật, tất cả các trang được cập nhật của bản sao mới đã được viết lên đĩa. Từ giả thiết file trên đĩa không bị hư hại do sự cố hệ thống. Do vậy, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và sẽ thấy nội dung của CSDL sau tất cả các cập nhật đã thực hiện bởi giao dịch. Sự thực thi này phụ thuộc vào việc viết lên db_pointer, việc viết này phải là nguyên tử, có nghĩa là hoặc tất cả các byte của nó được viết hoặc không byte nào được viết. Nếu chỉ một số byte của con trỏ được cập nhật bởi việc viết nhưng các byte khác thì không thì con trỏ trở thành vô nghĩa và cả bản cũ lẫn bản mới của CSDL có thể tìm thấy khi hệ thống khởi động lại. May mắn thay, hệ thống đĩa cung cấp các cập nhật nguyên tử toàn bộ khối đĩa hoặc ít nhất là một sector đĩa. Như vậy hệ thống đĩa đảm bảo việc cập nhật con trỏ db_pointer là nguyên tử. Tính nguyên tử và tính bền vững của giao dịch được đảm bảo bởi việc thực thi bản sao bóng của thành phần quản trị phục hồi. Sự thực thi này cực kỳ thiếu hiệu quả trong ngữ cảnh CSDL lớn, do sự thực hiện một giao dịch đòi hỏi phải sao toàn bộ CSDL. Hơn nữa sự thực thi này không cho phép các giao dịch thực hiện đồng thời với các giao dịch khác. Phương pháp thực thi tính nguyên tử và tính lâu bền mạnh hơn và đỡ tốn kém hơn được trình bày trong chương *hệ thống phục hồi*.

Các thực hiện tương tranh

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

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

có nghĩa là hiệu suất sử dụng bộ xử lý và đĩa tăng lên.

- Có thể có sự trộn lẫn các giao dịch đang chạy trong hệ thống, cái thì dài cái thì ngắn. Nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến trước hoàn tất, mà điều đó dẫn đến một sự trì hoãn không lường trước được trong việc chạy một giao dịch. Nếu các giao dịch đang hoạt động trên các phần khác nhau của CSDL, sẽ tốt hơn nếu ta cho chúng chạy đồng thời, chia sẻ các chu kỳ CPU và truy xuất đĩa giữa chúng. Thực hiện tương tranh làm giảm sự trì hoãn không lường trước trong việc chạy các giao dịch, đồng thời làm giảm thời gian đáp ứng trung bình: *Thời gian để một giao dịch được hoàn tất sau khi đã được đệ trình*.

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

Xét hệ thống ngân hàng đơn giản, nó có một số tài khoản và có một tập hợp các giao dịch, chúng truy xuất, cập nhật các tài khoản này. Giả sử T_1 và T_2 là hai giao dịch chuyển khoản từ một tài khoản sang một tài khoản khác. Giao dịch T_1 chuyển 50$ từ tài khoản A sang tài khoản B. Giao dịch T_2 chuyển 10% số dư từ tài khoản A sang tài khoản B, và được xác định như sau:

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

Giả sử giá trị hiện tại của A và B tương ứng là 1000$ và 2000$. Giả sử rằng hai giao dịch này được thực hiện theo trình tự:

Trường hợp 1: thực hiện xong giao dịch T_1 rồi đến giao dịch T_2

Trường hợp 2: thực hiện xong giao dịch T_2 rồi đến giao dịch T_1

T_1	T_2
Read(A); A:=A-50; **Write**(A); **Read**(B); B:=B+50; **Write**(B);	
	Read(A); Temp:=A*0.1; A:=A-temp; **Write**(A); **Read**(B); B:=B+temp; **Write**(B);
Thời lịch 1: Giá trị sau cùng của A là 855, B là 2145, tổng 2 tài khoản (A+B) là không đổi	

T_1	T_2
	Read(A); Temp:=A*0.1; A:=A-temp; **Write**(A); **Read**(B); B:=B+temp; **Write**(B);
Read(A); A:=A-50; **Write**(A); **Read**(B); B:=B+50; **Write**(B);	
Thời lịch 2: Giá trị sau cùng của A là 850, B là 2150, tổng 2 tài khoản (A+B) là không đổi	

Thời lịch (schedule): là một dãy các thao tác (lệnh) của các giao tác được sắp xếp theo trình tự thời gian.

Một thời lịch đối với một tập các giao dịch phải bao gồm tất cả các chỉ thị của các giao dich này và phải bảo tồn thứ tự các chỉ thị xuất hiện trong mỗi một giao dịch. Ví dụ, đối với giao dịch T_1, chỉ thị **Write**(A) phải xuất hiện trước chỉ thị **Read**(B), trong bất kỳ thời lịch hợp lệ nào. Các thời lịch 1 và Thời lịch 2 là những thời lịch tuần tự.

Thời lịch tuần tự gồm một dãy các chỉ thị từ các giao dịch, trong đó các chỉ thị thuộc về một giao dịch xuất hiện cùng nhau trong thời lịch. Như vậy, đối với một tập n giao dịch, có n! thời lịch tuần tự hợp lệ khác nhau.

Khi một số giao dịch được thực hiện đồng thời, thời lịch tương ứng không nhất thiết là tuần tự. Nếu hai giao dịch đang chạy đồng thời, hệ điều hành có thể thực hiện một giao dịch trong một khoảng ngắn thời gian, sau đó chuyển đổi ngữ cảnh, thực hiện giao dịch thứ hai một khoảng thời gian sau đó lại chuyển sang thực hiện giao dịch thứ nhất một khoảng và cứ như vậy (hệ thống chia sẻ thời gian). Thời lịch 3 và Thời lịch 4 là các ví dụ.

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

Thời lịch 3

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

Thời lịch 4

Tuy nhiên, không phải tất cả các thực hiện tương tranh đều cho ra một trạng thái đúng. Ví dụ xét thời lịch 4:

Sau khi thực hiện giao dịch này, ta đạt tới trạng thái trong đó giá trị cuối của A và B tương ứng là 950$ và 2100$. Trạng thái này là một trạng thái không nhất quán (A+B trước khi thực hiện giao dịch là 3000$ nhưng sau khi giao dịch là 3050$). Như vậy, nếu giao phó việc điều khiển thực hiện tương tranh cho hệ điều hành, sẽ có thể dẫn tới các trạng thái không nhất quán. Nhiệm vụ của hệ CSDL là đảm bảo rằng một thời lịch được phép thực hiện tương tranh sẽ đưa CSDL sang một trạng thái nhất quán. Thành phần của hệ CSDL thực hiện nhiệm vụ này được gọi là thành phần điều khiển tương tranh (concurrency-control component). Ta có thể đảm bảo sự nhất quán của CSDL với thực hiện tương tranh bằng cách nắm chắc rằng một thời lịch được thực hiện có cùng hiệu quả như một thời lịch tuần tự.

Tính khả tuần tự (Serializability)

Hệ CSDL phải điều khiển sự thực hiện tương tranh các giao dịch để đảm bảo rằng trạng thái CSDL giữ nguyên ở trạng thái nhất quán. Trước khi ta kiểm tra hệ CSDL có thể thực hiện nhiệm vụ này như thế nào, đầu tiên ta phải hiểu các thời lịch nào sẽ đảm bảo tính nhất quán và các thời lịch nào không. Vì các giao dịch là các chương trình, nên thật khó xác định các hoạt động chính xác được thực hiện bởi một giao dịch là hoạt động gì và

những hoạt động nào của các giao dịch tác động lẫn nhau. Vì lý do này, ta sẽ không giải thích kiểu hoạt động mà một giao dịch có thể thực hiện trên một mục dữ liệu. Thay vào đó, ta chỉ xét hai hoạt động: **Read** và **Write**. Ta cũng giả thiết rằng giữa một chỉ thị **Read(Q)** và một chỉ thị **Write(Q)** trên một mục dữ liệu **Q,** một giao dịch có thể thực hiện một dãy tuỳ ý các hoạt động trên bản sao của Q được lưu trú trong buffer cục bộ của giao dịch. Vì vậy ta sẽ chỉ nêu các chỉ thị **Read** và **Write** trong thời lịch, nếu biểu diễn với quy ước như vậy của thời lịch 3 sẽ là:

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

Tuần tự xung đột (Conflict Serializability)

Xét thời lịch S trong đó có hai chỉ thị liên tiếp I_i và I_j của các giao dịch T_i , T_j tương ứng ($i \neq j$). Nếu I_i và I_j tham khảo đến các mục dữ liệu khác nhau, ta có thể đổi chỗ I_i và I_j mà không làm ảnh hưởng đến kết quả của bất kỳ chỉ thị nào trong thời lịch. Tuy nhiên, nếu I_i và I_j tham khảo cùng một mục dữ liệu Q, khi đó thứ tự của hai bước này có thể rất quan trọng. Do ta đang thực hiện chỉ các chỉ thị **Read** và **Write**, nên ta có bốn trường hợp cần phải xét sau:

1. I_i = **Read**(Q); I_j = **Read**(Q): Thứ tự của I_i và I_j không gây ra vấn đề nào, do T_i và T_j đọc cùng một giá trị Q bất kể đến thứ tự giữa I_i và I_j.
2. I_i = **Read**(Q); I_j = **Write**(Q): thứ tự thực hiện của I_i và I_j là quan trọng.
3. I_i = **Write**(Q); I_j = **Read**(Q): thứ tự thực hiện của I_i và I_j là quan trọng.
4. I_i = **Write**(Q); I_j = **Write**(Q): Cả hai chỉ thị là hoạt động **Write**, thứ tự của hai chỉ thị này không ảnh hưởng đến cả hai giao dịch T_i và T_j . Tuy nhiên, giá trị nhận được bởi chỉ thị **Read** kế trong S sẽ bị ảnh hưởng do kết quả phụ thuộc vào chỉ thị **Write** được thực hiện sau cùng trong hai chỉ thị **Write** này. Nếu không còn chỉ thị **Write** nào sau I_i và I_j trong S, thứ tự của thứ tự thực hiện của I_i và I_j sẽ ảnh hưởng trực tiếp đến giá trị cuối của Q trong trạng thái CSDL kết quả (của thời lịch S).

Như vậy chỉ trong trường hợp cả I_i và I_j là các chỉ thị **Read**, thứ tự thực hiện của hai chỉ thị này (trong S) là không gây ra vấn đề.

*Ta nói I_i và I_j xung đột nếu các hoạt động này nằm trong các giao dịch khác nhau, tiến hành trên cùng một mục dữ liệu và có ít nhất một hoạt động là **Write**.*

Ví dụ, trong thời lịch schedule - 3: Chỉ thị **Write(A)** trong T_1 xung đột với **Read(A)** trong T_2. Tuy nhiên, chỉ thị **Write(A)** trong T_2 không xung đột với chỉ thị **Read(B)** trong T_1 do các chỉ thị này truy xuất các mục dữ liệu khác nhau.

I_i và I_j là hai chỉ thị liên tiếp trong thời lịch S. Nếu I_i và I_j là các chỉ thị của các giao dịch khác nhau và không xung đột, khi đó ta có thể đổi thứ tự của chúng mà không làm ảnh hưởng gì đến kết quả xử lý và như vậy ta nhận được một thời lịch mới S' tương đương với S. Chẳng hạn, do chỉ thị **Write(A)** của T_2 không xung đột với chỉ thị **Read(B)** của T_1, ta có thể đổi chỗ các chỉ thị này để được một thời lịch tương đương – thời lịch 5 dưới đây

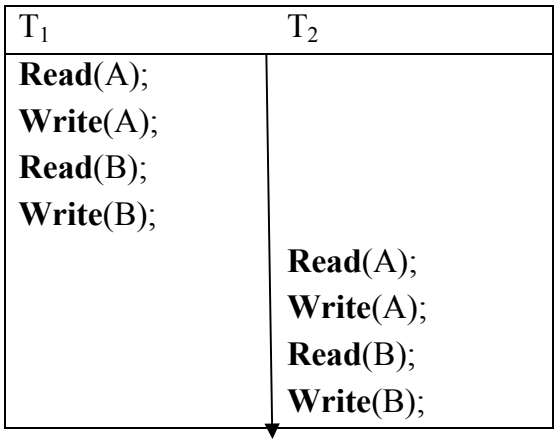
T_1	T_2
Read(A);	
Write(A);	
	Read(A);
Read(B);	
	Write(A);
Write(B);	
	Read(B);
	Write(B);

Thời lịch 5

Ta tiếp tục đổi chỗ các chỉ thị không xung đột như sau:

- Đổi chỗ chỉ thị **Read(B)** của T_1 với chỉ thị **Read(A)** của T_2
- Đổi chỗ chỉ thị **Write(B)** của T_1 với chỉ thị **Write(A)** của T_2
- Đổi chỗ chỉ thị **Write(B)** của T_1 với chỉ thị **Read(A)** của T_2

Kết quả cuối cùng của các bước đổi chỗ này là một thời lịch mới (thời lịch 6 –thời lịch tuần tự) tương đương với thời lịch ban đầu (thời lịch 3):



Thời lịch 6

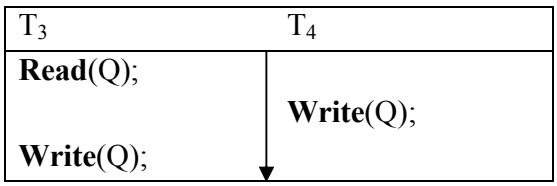
Sự tương đương này cho ta thấy: bất chấp trạng thái hệ thống ban đầu, thời lịch-3 sẽ sinh ra cùng trạng thái cuối như một thời lịch tuần tự nào đó.

***Tương đương xung đột (conflict equivalent)**:Nếu một thời lịch S có thể biến đổi thành một thời lịch S' bởi một dãy các thao tác đổi chỗ các chỉ thị không xung đột, ta nói S và S' là tương đương xung đột.*

Ví dụ, trong các thời lịch đã được nêu ở trên, ta thấy thời lịch-1 tương đương xung đột với thời lịch-3.

Khả tuần tự xung đột (conflict serializable): Một thời lịch S là khả tuần tự xung đột (conflict serializable) nếu nó tương đương xung đột với một thời lịch tuần tự.

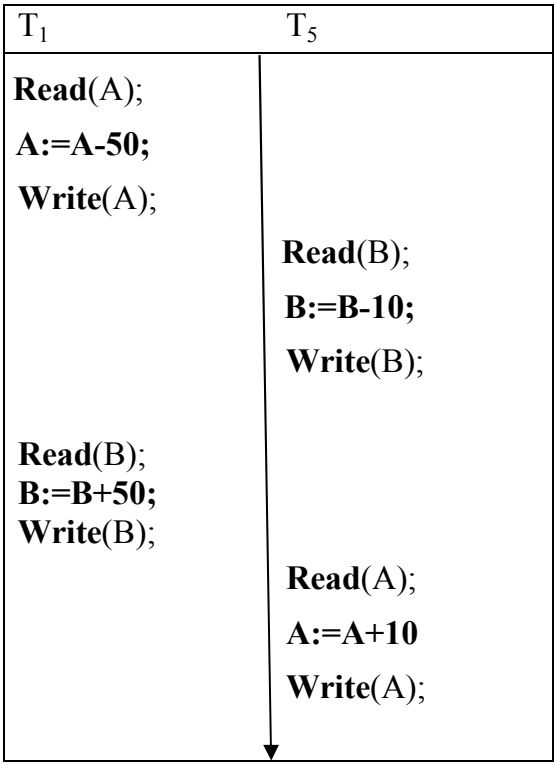
Ví dụ, thời lịch 3 là khả tuần tự xung đột còn thời lịch 7 dưới đây không tương đương xung đột với một thời lịch tuần tự nào do vậy nó không là khả tuần tự xung đột:



Thời lịch 7

Có thể có hai thời lịch sinh ra cùng kết quả, nhưng không tương đương xung đột. Ví dụ, giao dịch T_5 chuyển 10$ từ tài khoản B sang tài khoản A. Ta xét thời lịch 8 như dưới đây, thời lịch này không tương đương xung đột với thời lịch tuần tự < T_1, T_5 > do trong thời lịch 8 chỉ thị **Write(B)** của T_5 xung đột với chỉ thị **Read(B)** của T_1 như vậy ta không thể di chuyển tất cả các chỉ thị của T_1 về trước các chỉ thị của T_5 bởi việc hoán đổi liên tiếp các chỉ thị không xung đột. Tuy nhiên, các giá trị sau cùng của tài khoản A và B sau khi thực hiện thời lịch 8 hoặc sau khi thực hiện thời lịch tuần tự <T_1, T_5 > là như nhau A là

960 và B là 2040 tương ứng. Qua ví dụ này ta thấy cần thiết phải phân tích cả sự tính toán được thực hiện bởi các giao dịch mà không chỉ các hoạt động **Read** và **Write.** Tuy nhiên sự phân tích như vậy sẽ nặng nề và phải trả một giá tính toán cao hơn.



Schedule - 8

Tuần tự View (View Serializability)

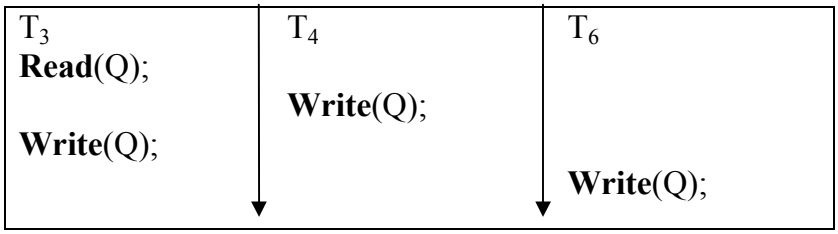
Xét hai thời lịch S và S', trong đó cùng một tập hợp các giao dịch tham gia vào cả hai thời lịch. Các thời lịch S và S' được gọi là *tương đương view* nếu ba điều kiện sau được thỏa mãn:

1. Đối với mỗi mục dữ liệu Q, nếu giao dịch T_i đọc giá trị khởi đầu của Q trong thời lịch S, thì giao dịch T_i phải cũng đọc giá trị khởi đầu của Q trong thời lịch S'.
2. Đối với mỗi mục dữ liệu Q, nếu giao dịch T_i thực hiện **Read(Q)** trong thời lịch S và giá trị đó được sản sinh ra bởi giao dịch T_j thì T_i cũng phải đọc giá trị của Q được sinh ra bởi giao dịch T_j trong S'.
3. Đối với mỗi mục dữ liệu Q, giao dịch thực hiện hoạt động **Write(Q)** sau cùng trong thời lịch S, phải thực hiện hoạt động **Write(Q)** sau cùng trong thời lịch S'.

Điều kiện 1 và 2 đảm bảo mỗi giao dịch đọc cùng các giá trị trong cả hai thời lịch và do vậy thực hiện cùng tính toán. Điều kiện 3 đi cặp với các điều kiện 1 và 2 đảm bảo cả hai thời lịch cho ra kết quả là trạng thái cuối cùng của hệ thống như nhau. Trong các ví dụ trước, thời lịch-1 là không tương đương view với thời lịch 2 do trong thời lịch-1, giá trị

của tài khoản A được đọc bởi giao dịch T_2 được sinh ra bởi T_1, trong khi điều này không xảy ra trong thời lịch-2. Thời lịch-1 tương đương view với thời lịch-3 vì các giá trị của các tài khoản A và B được đọc bởi T_2 được sinh ra bởi T_1 trong cả hai thời lịch.

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



Thời lịch 9

Nó tương đương view với thời lịch tuần tự < T_3, T_4, T_6 > do chỉ thị **Read(Q)** đọc giá trị khởi đầu của Q trong cả hai thời lịch và T_6 thực hiện **Write** sau cùng trong cả hai thời lịch như vậy thời lịch 9 khả tuần tự view.

Mỗi thời lịch khả tuần tự xung đột là khả tuần tự view, nhưng có những thời lịch khả tuần tự view không khả tuần tự xung đột (ví dụ thời lịch 9).

Trong thời lịch 9 các giao dịch T_4 và T_6 thực hiện các hoạt động **Write(Q)** mà không thực hiện hoạt động **Read(Q)**, Các Write dạng này được gọi là các Write mờ (blind write). Các Write mờ xuất hiện trong bất kỳ thời lịch khả tuần tự view không khả tuần tự xung đột.

Tính khả phục hồi (Recoverability)

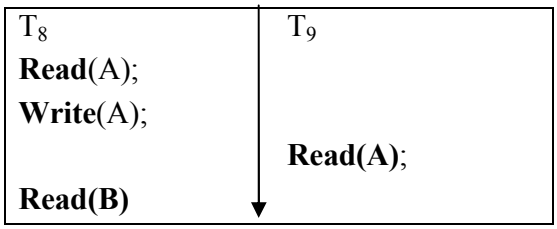
Ta đã nghiên cứu các thời lịch có thể chấp nhận dưới quan điểm sự nhất quán của CSDL với giả thiết không có giao dịch nào thất bại. Ta sẽ xét hiệu quả của thất bại giao dịch trong thực hiện tương tranh.

Nếu giao dịch T_i thất bại vì lý do nào đó, ta cần hủy bỏ hiệu quả của giao dịch này để đảm bảo tính nguyên tử của giao dịch. Trong hệ thống cho phép thực hiện tương tranh, cũng cần thiết đảm bảo rằng bất kỳ giao dịch nào phụ thuộc vào T_i cũng phải bị bỏ. Để thực hiện sự chắc chắn này, ta cần bố trí các hạn chế trên kiểu thời lịch được phép trong hệ thống.

Thời lịch khả phục hồi (Recoverable Schedule)

Xét thời lịch 10 trong đó T_9 là một giao dịch chỉ thực hiện một chỉ thị **Read(A)**. Giả sử hệ thống cho phép T_9 bàn giao ngay sau khi thực hiện chỉ thị **Read(A).** Như vậy T_9 bàn

giao trước T_8. Giả sử T_8 thất bại trước khi bàn giao, vì T_9 đã đọc giá trị của mục giữ liệu A được viết bởi T_8 , ta phải bỏ dở T_9 để đảm bảo tính nguyên tử giao dịch. Tuy nhiên T_9 đã được bàn giao và không thể bỏ dở được. Ta có tình huống trong đó không thể khôi phục đúng sau thất bại của T_8.



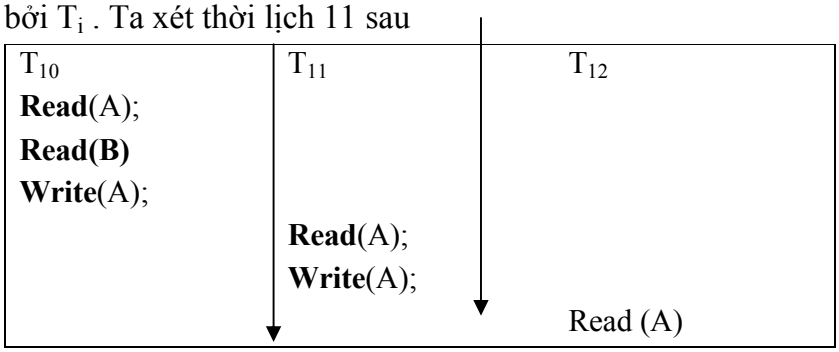
Thời lịch 10

Thời lịch 10 là một ví dụ về thời lịch không phục hồi được và không được phép. Hầu hết các hệ CSDL đòi hỏi tất cả các thời lịch phải phục hồi được.

Một thời lịch khả phục hồi là thời lịch trong đó, đối với mỗi cặp giao dịch T_i , T_j , nếu T_j đọc mục dữ liệu được viết bởi T_i thì hoạt động bàn giao của T_j phải xảy ra sau hoạt động bàn giao của T_i .

Thời lịch cascadeless (Cascadeless Schedule)

Ngay cả khi thời lịch là khả phục hồi, để phục hồi đúng sau thất bại của một giao dịch T_i ta phải cuộn lại một vài giao dịch. Tình huống như thế xảy ra khi các giao dịch đọc dữ liệu được viết bởi T_i . Ta xét thời lịch 11 sau



Thời lịch 11

Giao dịch T_{10} viết một giá trị được đọc bởi T_{11}. Giao dịch T_{12} đọc một giá trị được viết bởi T_{11}. Giả sử rằng tại điểm này T_{10} thất bại. T_{10} phải cuộn lại, do T_{11} phụ thuộc vào T_{10} nên T_{11} cũng phải cuộn lại và cũng như vậy với T_{12}. Hiện tượng trong đó một giao dịch thất bại kéo theo một dãy các giao dịch phải cuộn lại được gọi là sự cuộn lại hàng loạt (cascading rollback).

Cuộn lại hàng loạt dẫn đến việc huỷ bỏ một khối lượng công việc đáng kể. Phải hạn chế các thời lịch để việc cuộn lại hàng loạt không thể xảy ra. Các thời lịch như vậy được gọi là các thời lịch cascadeless.

Một thời lịch cascadeless là một thời lịch trong đó mỗi cặp giao dịch T_i , T_j nếu T_j đọc một mục dữ liệu được viết trước đó bởi T_i, hoạt động bàn giao của T_i phải xuất hiện trước hoạt động đọc của T_j . Một thời lịch cascadeless là khả phục hồi.

Thực thi cô lập (Implementation of Isolation)

Có nhiều sơ đồ điều khiển tương tranh có thể được sử dụng để đảm bảo các tính chất một thời lịch phải có (nhằm giữ CSDL ở trạng thái nhất quán, cho phép quản lý các giao dịch...), ngay cả khi nhiều giao dịch thực hiện tương tranh, chỉ các thời lịch có thể chấp nhận được sinh ra, bất kể hệ điều hành chia sẻ thời gian tài nguyên như thế nào giữa các giao dịch.

Ví dụ ta xét một sơ đồ điều khiển tương tranh sau: một giao dịch được cấp một khóa (lock) trên toàn bộ CSDL trước khi nó khởi động và tháo khóa khi nó đã bàn giao. Trong khi giao dịch này giữ khóa, không giao dịch nào khác được cấp khóa và như vậy phải chờ đến tận khi khóa được tháo. Trong cơ chế khóa, chỉ một giao dịch được thực hiện tại một thời điểm và như vậy chỉ thời lịch tuần tự được sinh ra. Sơ đồ điều khiển tương tranh này cho ra một hiệu năng tương tranh thấp. Ta nói nó cung cấp một bậc tương tranh thấp (poor degree of concurrency).

Mục đích của các sơ đồ điều khiển tương tranh là cung cấp một bậc cạnh tranh cao trong khi vẫn đảm bảo các thời lịch được sinh ra là khả tuần tự xung đột hoặc khả tuần tự view và cascadeless.

2. Thuật toán kiểm tra tính khả tuần tự

Thuật toán kiểm tra xung đột tương đương (Tính khả tuần tự xung đột)

Cho S là một thời lịch. Ta dùng một đồ thị định hướng, được gọi là đồ thị trình tự (precedence graph), biểu diễn từ S=(V, E). Trong đó:

V là tập các đỉnh và E là tập các cung.

Tập các đỉnh V bao gồm tất cả các giao dịch tham gia vào thời lịch.

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

(i). T_i thực hiện **Write(Q)** trước T_j thực hiện **Read(Q).**

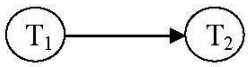
(ii). T_i thực hiện **Read(Q)** trước khi T_j thực hiện **Write(Q).**

(iii). T_i thực hiện **Write(Q)** trước khi T_j thực hiện **Write(Q).**

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

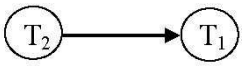
Ví dụ:

Thời lịch 1 được biểu diễn bằng đồ thị



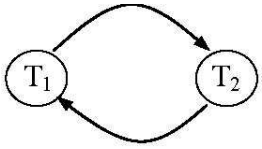
vì tất cả các chỉ thị của T_1 được thực hiện trước chỉ thị đầu tiên của T_2.

Thời lịch 2 được biểu diễn bằng đồ thị



vì tất cả các chỉ thị của T_2 được thực hiện trước chỉ thị đầu tiên của T_1.

Thời lịch 4 được biểu diễn bằng đồ thị chứa các cung $T_1 \rightarrow T_2$ vì T_1 thực hiện **Read(A)** trước T_2 thực hiện **Write(A).** Nó cũng chứa cung $T_2 \rightarrow T_1$ vì T_2 thực hiện **Read(B)** trước khi T_1 thực hiện **Write(B):**



Nếu đồ thị trình tự đối với S có chu trình, khi đó thời lịch S không là khả tuần tự xung đột. *Như vậy bài toán kiểm tra tính khả tuần tự xung đột của thời lịch S đưa về bài toán kiểm tra chu trình trên đồ thị trình tự biểu diễn của S.*

Thuật toán kiểm tra khả tuần tự view

Không có thuật toán nào hiệu quả để kiểm tra tính khả tuần tự view của một thời lịch. Một cách tiếp cận là cải tiến thuật toán kiểm tra xung đột tương đương để kiểm tra tính khả tuần tự view.

Giả sử S là thời lịch gồm các giao dịch $\{T_1 , T_2 , ... , T_n\}$. T_b và T_f là hai giao dịch giả sao cho: T_b bắt nguồn lệnh **Write(Q)** đối với mỗi mục dữ liệu Q được truy xuất trong S, T_f bắt nguồn lệnh **Read(Q)** đối với mỗi mục dữ liệu Q được truy xuất trong S. Ta xây dựng thời lịch mới S' từ S bằng cách xen T_b ở bắt đầu của S và T_f ở cuối của S. Biểu diễn S' bằng đồ thị trình tự có gán nhãn trên cung được xây dựng dựa trên các quy tắc:

1. Gán nhãn 0 cho cung $T_i \rightarrow T_j$, nếu T_j đọc giá trị của mục dữ liệu Q được viết bởi T_i
2. Xóa tất cả các cung liên quan tới các giao dịch không dùng. Một giao dịch T_i được gọi là không dùng nếu không có con đường nào trong đồ thị trình tự dẫn từ T_i đến T_f .
3. Đối với mỗi mục dữ liệu Q sao cho T_j ***đọc giá trị của Q được viết (Write) bởi*** T_i và T_k thực hiện **Write(Q),** $T_k \neq T_b$ tiến hành các bước sau :
    a. Nếu $T_i = T_b$ và $T_j \neq T_f$, gán nhãn 0 cho cung $T_j \rightarrow T_k$ trong đồ thị trình tự gán

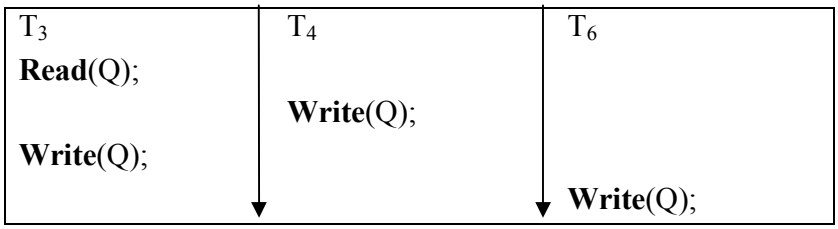
nhãn

b. Nếu $T_i \neq T_b$ và $T_j = T_f$ gán nhãn 0 cho cung $T_k \rightarrow T_i$ trong đồ thị trình tự gán nhãn

c. Nếu $T_i \neq T_b$ và $T_j \neq T_f$ gán nhãn p cho cả hai cung $T_k \rightarrow T_i$ và $T_j \rightarrow T_k$ trong đồ thị trình tự gán nhãn, trong đó p là một số nguyên duy nhất > 0 mà chưa được sử dụng trước đó để gán nhãn cung.

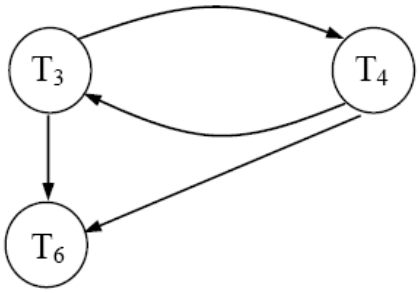
Quy tắc 3c phản ánh rằng nếu T_i viết mục dữ liệu được đọc bởi T_j thì một giao dịch T_k viết cùng mục dữ liệu này phải hoặc đi trước T_i hoặc đi sau T_j. Quy tắc 3a và 3b là trường hợp đặc biệt là kết quả của sự kiện T_b và T_f cần thiết là các giao dịch đầu tiên và cuối cùng tương ứng.

Ví dụ: xét thời lịch 9

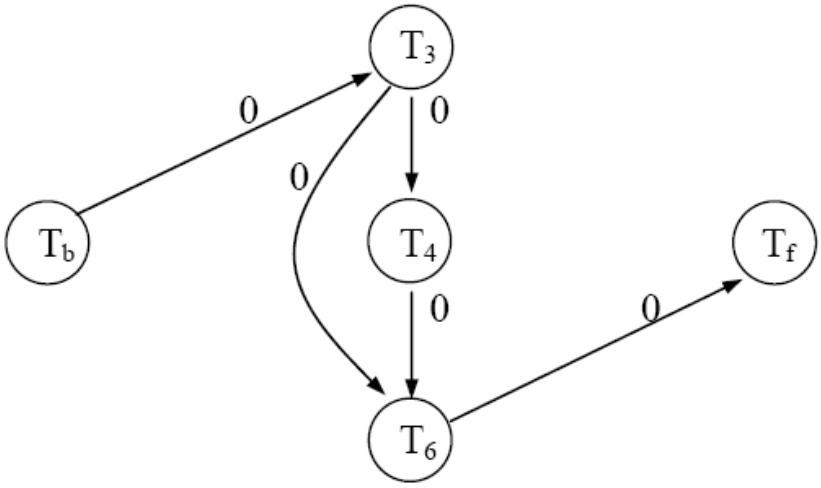


Thời lịch 9

Biểu diễn đồ thị của S như sau:



Và đồ thị trình tự có gán nhãn biểu diễn S’ như sau:



Nếu đồ thị trình tự gán nhãn không chứa chu trình, thời lịch tương ứng là khả tuần tự view, như vậy thời lịch 9 là khả tuần tự view. Tuy nhiên, nếu đồ thị S' chứa chu trình, điều kiện này không kéo theo thời lịch tương ứng không là khả tuần tự view. Bây giờ ta giả sử rằng có n cặp cung tách biệt, *đó là do ta đã áp dụng n lần quy tắc 3c trong sự xây dựng đồ thị trình tự.* Khi đó có 2^n đồ thị khác nhau, mỗi một đồ thị chứa đúng một cung trong mỗi cặp. Nếu một đồ thị nào đó trong các đồ thị này là phi chu trình, khi đó thời lịch tương ứng là khả tuần tự view. Thuật toán này đòi hỏi một phép kiểm thử vét cạn các đồ thị riêng biệt, và như vậy thuộc về lớp bài toán NP-đầy đủ.

3. Điều khiển tương tranh dựa trên khóa

Một phương pháp để đảm bảo tính khả tuần tự là yêu cầu việc truy xuất đến mục dữ liệu được tiến hành theo kiểu loại trừ tương hỗ; có nghĩa là trong khi một giao dịch đang truy xuất một mục dữ liệu, không một giao dịch nào khác có thể sửa đổi mục dữ liệu này. Phương pháp chung nhất được dùng để thực thi yêu cầu này là cho phép một giao dịch truy xuất một mục dữ liệu chỉ nếu nó đang giữ khóa trên mục dữ liệu đó.

Khóa (Lock)

Có nhiều phương thức khóa mục dữ liệu. Ta hạn chế việc nghiên cứu trên hai phương thức:

1. **Shared:** Nếu một giao dịch T_i nhận được một khóa ở phương thức shared (ký hiệu là S) trên mục Q, khi đó T_i có thể đọc, nhưng không được viết Q.
2. **Exclusive.** Nếu một giao dịch T_i nhận được một khóa ở phương thức Exclusive (ký hiệu là X), khi đó T_i có thể cả đọc lẫn viết Q.

Hàm tương thích

Mỗi giao dịch đòi hỏi một khóa ở một phương thức thích hợp trên mục dữ liệu Q, phụ thuộc vào kiểu hoạt động mà nó sẽ thực hiện trên Q. Giả sử một giao dịch T_i đòi hỏi một khóa phương thức A trên mục Q mà trên nó giao dịch T_j ($T_j \neq T_i$) hiện đang giữ một khóa phương thức B. Nếu giao dịch T_i có thể được cấp một khóa trên Q ngay, bất chấp sự hiện diện của khóa phương thức B, khi đó ta nói phương thức A tương thích với phương thức B. Hàm tương thích giữa hai phương thức khóa được biểu diễn bởi một ma trận. Quan hệ cho bởi ma trận **comp** sau:

	S	X
S	True	False
X	False	False

Comp(A, B) = True có nghĩa là các phương thức A và B tương thích.

Các thủ tục cấp và thu hồi khóa như sau:

- **lock-S(Q):** yêu cầu một khóa **shared** trên mục dữ liệu Q.
- **lock-X(Q):** yêu cầu một khóa **exclusive** trên mục dữ liệu Q.
- **unlock(Q):** thu hồi khóa trên mục dữ liệu Q

Để truy xuất một mục dữ liệu, giao dịch T_i đầu tiên phải khóa mục này. Nếu mục này đã bị khóa bởi một giao dịch khác ở phương thức không tương thích, bộ điều khiển tương tranh sẽ không cấp khóa cho đến tận khi tất cả các khóa không tương thích bị giữ bởi các giao dịch khác được tháo. Như vậy T_i phải chờ đến tận khi tất cả các khóa không tương thích bị giữ bởi các giao dịch khác được giải phóng.

Giao dịch T_i có thể tháo khóa một mục dữ liệu mà nó đã khóa trước đây. Một giao dịch cần thiết phải giữ một khóa trên một mục dữ liệu chừng nào mà nó còn truy xuất mục này. Hơn nữa, đối với một giao dịch việc tháo khóa ngay sau truy xuất cuối cùng đến mục dữ liệu không luôn luôn là điều mong muốn vì như vậy tính khả tuần tự có thể không được đảm bảo. Để minh hoạ cho tình huống này, ta xét ví dụ sau: A và B là hai tài khoản có thể được truy xuất bởi các giao dịch T_1 và T_2 . Giao dịch T_1 chuyển 50$ từ tài khoản B sang tài khoản A và được xác định như sau:

T_1 : **Lock-X(B);**
Read(B);
B:=B-50;
Write(B);
Unlock(B);
Lock-X(A);
Read(A);

A:=A+50;
Write(A);
Unlock(A);

Giao dịch T_2 hiển thị tổng số lượng tiền trong các tài khoản A và B (A + B) và được xác định như sau:

T_2 : **Lock-S(A);**
Read(A);
Unlock(A);
Lock-S(B);
Read(B);
Unlock(B);
Display(A+B);

Giả sử giá trị của tài khoản A và B tương ứng là 100$ và 200$. Nếu hai giao dịch này thực hiện tuần tự, hoặc theo thứ tự T_1, T_2 hoặc theo thứ tự T_2 , T_1 , và khi dó T_2 sẽ hiển thị giá trị 300$. Tuy nhiên nếu các giao dịch này thực hiện tương tranh, giả sử theo thời lịch schedule-1, trong trường hợp như vậy giao dịch T_2 sẽ hiển thị giá trị 250$, một kết quả không đúng vì giao dịch T_1 đã tháo khóa mục B quá sớm và T_2 đã tham khảo một trạng thái không nhất quán.

Thời lịch 12 cho thấy các hoạt động được thực hiện bởi các giao dịch cũng như các thời điểm khi các khóa được cấp bởi bộ quản trị điều khiển tương tranh. Giao dịch đưa ra một yêu cầu khóa không thể thực hiện hành động kế của mình đến tận khi khóa được cấp bởi bộ quản trị điều khiển tương tranh; do đó, khóa phải được cấp trong khoảng thời gian giữa hoạt động yêu cầu khóa và hành động sau của giao dịch. Sau này ta sẽ luôn giả thiết khóa được cấp cho giao dịch ngay trước hành động kế và như vậy ta có thể bỏ qua cột bộ quản trị điều khiển tương tranh trong bảng.

T_1	T_2	Bộ quản trị điều khiển tương tranh
Lock-X(B)		
		Grant-X(B,T_1)
Read(B)		
B:=B-50		
Write(B)		
Unlock(B)		
	Lock-S(A)	
		Grant-S(A,T_2)
	Read(A)	

	Unlock(A)	
	Lock-S(B)	
		Grant-S(B,T_2)
	Read(B)	
	Unlock(B)	
	Display(A+B)	
Lock-X(A)		
		Grant-X(A,T_1)
Read(A)		
A:=A+50		
Write(A)		
Unlock(A)		

Thời lịch 12

Bây giờ giả sử rằng tháo khóa bị làm trễ đến cuối giao dịch. Giao dịch T_3 tương ứng với T_1 với tháo khóa bị làm trễ được định nghĩa như sau:

T_3 : **Lock-X(B);**

Read(B);

B:=B-50;

Write(B);

Lock-X(A);

Read(A);

A:=A+50;

Write(A);

Unlock(B);

Unlock(A);

Giao dịch T_4 tương ứng với T_2 với tháo khóa bị làm trễ được xác định như sau:

T_4 : **Lock-S(A);**

Read(A);

Lock-S(B);

Read(B);

Display(A+B);

Unlock(A);

Unlock(B);

Các thời lịch có thể trên T_3 và T_4 không để cho T_4 hiển thị trạng thái không nhất quán.

Tuy nhiên, sử dụng khóa có thể dẫn đến một tình huống không mong đợi. Ta xét thời lịch 13 gồm một phần công việc trên T_3 và T_4 sau:

T_3	T_4
Lock-X(B)	
Read(B)	
B:=B-50	
Write(B)	
	Lock-S(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

Thời lịch 13

Do T_3 giữ một khóa phương thức Exclusive trên B, nên yêu cầu một khóa phương thức shared của T_4 trên B phải chờ đến khi T_3 tháo khóa. Cũng vậy, T_3 yêu cầu một khóa Exclusive trên A trong khi T_4 đang giữ một khóa shared trên nó và như vậy phải chờ. Ta gặp phải tình huống trong đó T_3 chờ đợi T_4 đồng thời T_4 chờ đợi T_3, một sự chờ đợi vòng tròn, và như vậy không giao dịch nào có thể tiến triển. Tình huống này được gọi là deadlock (khóa chết). Khi tình huống khóa chết xảy ra hệ thống buộc phải cuộn lại một trong các giao dịch. Mỗi khi một giao dịch bị cuộn lại, các mục dữ liệu bị khóa bởi giao dịch phải được tháo khóa và nó trở nên sẵn có cho giao dịch khác, như vậy các giao dịch này có thể tiếp tục được sự thực hiện của nó.

Nếu ta không sử dụng khóa hoặc tháo khóa mục dữ liệu ngay khi có thể sau đọc hoặc viết mục, ta có thể rơi vào trạng thái không nhất quán. Mặt khác, nếu ta không tháo khóa một mục dữ liệu trước khi yêu cầu một khóa trên một mục dữ liệu khác, deadlock có thể xảy ra. Có các phương pháp tránh deadlock trong một số tình huống, tuy nhiên nói chung deadlock là khó tránh khi sử dụng khóa nếu ta muốn tránh trạng thái không nhất quán. Deadlock được ưa thích hơn trạng thái không nhất quán vì chúng có thể điều khiển được bằng cách cuộn lại các giao dịch trong khi đó trạng thái không nhất quán có thể dẫn đến các vấn đề thực tế mà hệ CSDL không thể điều khiển.

Nghi thức khóa (locking protocol)

Xét $\{ T_0 , T_1 , ..., T_n \}$ một tập các giao dịch tham gia vào thời lịch S. Ta nói T_i đi trước T_j trong S, và được viết là $T_i \rightarrow T_j$, nếu tồn tại một mục dữ liệu Q sao cho T_i giữ

khóa phương thức A trên Q , T_j giữ khóa phương thức B trên Q muộn hơn và **comp(A,B)** = false. Nếu $T_i \rightarrow T_j$, thì T_i sẽ xuất hiện trước T_j trong bất kỳ thời lịch tuần tự nào.

Ta nói một thời lịch S là hợp lệ dưới một nghi thức khóa nếu S là một thời lịch tuân thủ các quy tắc của nghi thức khóa đó. Ta nói rằng một nghi thức khóa đảm bảo tính khả tuần tự xung đột nếu và chỉ nếu đối với tất cả các thời lịch hợp lệ, quan hệ → kết hợp là phi chu trình.

Cấp khóa

Khi một giao dịch T_i yêu cầu một khóa trên một mục dữ liệu Q ở phương thức M, khóa sẽ được cấp nếu các điều kiện sau được thỏa mãn:

(i) Không có giao dịch khác đang giữ một khóa trên Q ở phương thức xung đột với M.

(ii) Không có một giao dịch nào đang chờ được cấp một khóa trên M và đã đưa ra yêu cầu về khóa trước T_i.

Nghi thức khóa hai kỳ (Two-phase locking protocol)

Nghi thức khóa hai kỳ là một nghi thức đảm bảo tính khả tuần tự. Nghi thức này yêu cầu mỗi một giao dịch phát ra yêu cầu khóa và tháo khóa qua hai giai đoạn:

1. **Giai đoạn tăng trưởng (Growing phase):** một giao dịch có thể nhận được các khóa, nhưng nó không thể tháo bất kỳ khóa nào.
2. **Giai đoạn thu hẹp (Shrinking phase):** một giao dịch có thể tháo các khóa nhưng không thể nhận được một khóa mới nào.

Khởi đầu, một giao dịch ở giai đoạn tăng trưởng. Giao dịch được cấp các khóa cần thiết. Mỗi khi giao dịch tháo một khóa, nó đi vào giai đoạn thu hẹp và nó không thể phát ra bất kỳ một yêu cầu khóa nào nữa. Các giao dịch T_3 và T_4 là hai kỳ. Các giao dịch T_1 và T_2 không là hai kỳ. Nghi thức khóa hai kỳ đảm bảo tính khả tuần tự xung đột, nhưng không đảm bảo tránh được deadlock và việc cuộn lại hàng loạt. Cuộn lại hàng loạt có thể tránh được bởi nghi thức khóa hai kỳ nghiêm ngặt.

Nghi thức khóa hai kỳ nghiêm ngặt (**Strict Two-Phase Locking**) gồm hai luật:

1. Một giao dịch T muốn đọc (sửa) mục dữ liệu Q nó phải yêu cầu cấp khóa Shared (Exclusive) trên Q.
2. Tất cả các khóa do một giao dịch đang nắm giữ được giải phóng khi giao dịch được bàn giao.

Một sự tinh chế nghi thức khóa hai kỳ cơ sở dựa trên việc cho phép chuyển đổi khóa: nâng cấp một khóa shared sang exclusive và hạ cấp một khóa exclusive thành khóa shared. Chuyển đổi khóa không thể cho phép một cách tuỳ tiện, nâng cấp chỉ được phép

diễn ra trong giai đoạn tăng trưởng, còn hạ cấp chỉ được diễn ra trong giai đoạn thu hẹp. Một giao dịch thử nâng cấp một khóa trên một mục dữ liệu Q có thể phải chờ. Nghi thức khóa hai kỳ với chuyển đổi khóa cho phép chỉ sinh ra các thời lịch khả tuần tự xung đột. Nếu các khóa exclusive được giữ đến tận khi bàn giao, các thời lịch sẽ là cascadeless.

Các thủ tục liên quan được sử dụng trong nghi thức khóa 2 kỳ nghiêm ngặt:

- **lock-S(Q):** yêu cầu một khóa **shared** trên mục dữ liệu Q.
- **lock-X(Q):** yêu cầu một khóa **exclusive** trên mục dữ liệu Q.
- **unlock(Q):** thu hồi (giải phóng, tháo) khóa trên mục dữ liệu Q.
- **Upgrade(Q)**: chuyển đổi khóa từ phương thức shared (S) sang phương thức exclusive (X) trên mục dữ liệu Q, Upgrade chỉ được phép xảy ra trong giai đoạn tăng trưởng (giai đoạn xin khóa).
- **Downgrade(Q)**: chuyển đổi khóa từ phương thức exclusive sang phương thức shared trên mục dữ liệu Q, Downgrade chỉ được phép xảy ra trong giai đoạn thu hẹp (giai đoạn tháo khóa).

Ta xét một ví dụ: Các giao dịch T_8 và T_9 được nêu trong ví dụ chỉ được trình bày bởi các hoạt động ý nghĩa là **Read** và **Write.**

T_8 : **Read(A_1);**
Read(A_2);
...
Read(A_n);
Write(A_1).

T_9 : **Read(A_1);**
Read(A_2);
Display($A_1 + A_2$).

Nếu ta sử dụng nghi thức khóa hai kỳ, khi đó T_8 phải khóa A_1 ở phương thức exclusive. Bởi vậy, sự thực hiện tương tranh của hai giao dịch trở thành thực hiện tuần tự. Ta thấy rằng T_8 cần một khóa exclusive trên A_1 chỉ ở cuối sự thực hiện của nó, khi nó write(A_1). Như vậy, T_8 có thể khởi động khóa A_1 ở phương thức shared và đổi khóa này sang phương thức exclusive sau này. Như vậy ta có thể nhận được tính tương tranh cao hơn, vì như vậy T_8 và T_9 có thể truy xuất đến A_1 và A_2 đồng thời.

T_8	T_9
Lock-S(A_1)	
	Lock-S(A_2)

Lock-S(A_2)	
	Lock-S(A_2)
Lock-S(A_3)	
…	
	Unlock(A_1)
	Unlock(A_2)
UpGrade(A_1)	

Chú ý rằng một giao dịch thử cập nhật một khóa trên một mục dữ liệu Q có thể buộc phải chờ. Việc chờ bắt buộc này xảy ra khi Q đang bị khóa bởi giao dịch khác ở phương thức shared.

Nghi thức khóa hai kỳ với chuyển đổi khóa chỉ sinh ra các thời lịch khả tuần tự xung đột, các giao dịch có thể được tuần tự hoá bởi các điểm khóa của chúng. Hơn nữa, nếu các khóa exclusive được giữ đến tận khi kết thúc giao dịch, thời lịch sẽ là cascadeless.

Quản lý khóa

Trong hệ quản trị CSDL, các khóa của các giao dịch được quản lý bởi bộ phận quản lý khóa (lock manager) thông qua bảng khóa (lock table). Mỗi mục của bảng khóa tương ứng với một đối tượng (trang, mẫu tin, …) chứa các thông tin: số lượng giao dịch đang giữ khóa trên đối tượng này (số lượng giao dịch có thể > 1 nếu đối tượng được khóa theo phương thức Shared), phương thức khóa (Shared hay Exclusive), con trỏ đến hàng đợi yêu cầu khóa trên đối tượng đó.

Khi một giao dịch T_i cần một khóa trên một mục dữ liệu Q, nó gửi một yêu cầu cấp khóa đến bộ quản lý khóa, yêu cầu được xử lý như sau:

1. Nếu yêu cầu một khóa Shared, hàng đợi các yêu cầu của mục Q là rỗng và mục dữ liệu Q không bị khóa theo phương thức Exclusive thì bộ phận quản lý khóa cấp khóa cho giao dịch T_i và cập nhật lại thông tin trên bảng khóa.
2. Nếu yêu cầu một khóa Exclusive và không có giao dịch nào đang giữ khóa trên Q thì bộ phận quản lý khóa cấp khóa cho giao dịch T_i và cập nhật lại thông tin trên bảng khóa.
3. Ngược lại, thêm yêu cầu này vào hàng đợi khóa của mục dữ liệu Q và giao dịch T_i tạm thời ngưng.

Khi giao dịch được bàn giao hoặc bị bỏ dở, nó phải tháo tất cả các khóa nó đang nắm giữ. Khi một khóa được giải phóng, bộ quản lý khóa cập nhật lại bảng khóa và xem xét cấp phát khóa cho các yêu cầu khóa đang ở trong hàng đợi tương ứng.

Quản lý deadlock

Một hệ thống ở trạng thái deadlock nếu tồn tại một tập hợp các giao dịch sao cho mỗi giao dịch trong tập hợp đang chờ một giao dịch khác trong tập hợp. Chính xác hơn, tồn tại một tập các giao dịch { T_0 , T_1 , ..., T_n } sao cho T_0 đang chờ một mục dữ liệu được giữ bởi T_1 , T_1 đang chờ một mục dữ liệu đang bị chiếm bởi T_2 , ..., T_{n-1} đang chờ một mục dữ liệu được giữ bởi T_n và T_n đang chờ một mục T_0 đang chiếm. Không một giao dịch nào có thể tiến triển được trong tình huống như vậy. Một cách xử lý tình huống này là cuộn lại một vài giao dịch tham gia vào deadlock.

Có hai phương pháp chính giải quyết vấn đề deadlock: ngăn ngừa deadlock, phát hiện deadlock và khôi phục. Nghi thức ngăn ngừa deadlock đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Sơ đồ phát hiện deadlock và khôi phục (deadlock-detection and deadlock-recovery scheme) cho phép hệ thống đi vào trạng thái deadlock và sau đó cố gắng khôi phục. Cả hai phương pháp đều có thể dẫn đến việc cuộn lại giao dịch. Phòng ngừa deadlock thường được sử dụng nếu xác suất hệ thống đi vào deadlock cao, phát hiện và khôi phục hiệu quả hơn trong các trường hợp còn lại.

Phòng ngừa deadlock (Deadlock prevention)

Một cách phòng ngừa deadlock là sử dụng thứ tự ưu tiên và cuộn lại quá trình. Với thứ tự ưu tiên, một giao dịch T_2 yêu cầu một khóa bị giữ bởi giao dịch T_1 , khóa đã cấp cho T_1 có thể bị lấy lại và cấp cho T_2 , T_1 bị cuộn lại. Để điều khiển ưu tiên, ta gán một nhãn thời gian duy nhất cho mỗi giao dịch. Hệ thống sử dụng các nhãn thời gian này để quyết định một giao dịch phải chờ hay cuộn lại. Khóa vẫn được sử dụng để điều khiển tương tranh. Nếu một giao dịch bị cuộn lại, nó vẫn giữ nhãn thời gian cũ của nó khi tái khởi động. Hai sơ đồ phòng ngừa deadlock sử dụng nhãn thời gian khác nhau được đề nghị:

1. Sơ đồ **Wait-Die** dựa trên kỹ thuật không ưu tiên. Khi giao dịch T_i yêu cầu một mục dữ liệu bị chiếm bởi T_j , T_i được phép chờ chỉ nếu nó có nhãn thời gian nhỏ hơn của T_j nếu không T_i bị cuộn lại (die).
2. Sơ đồ **Wound-Wait** dựa trên kỹ thuật ưu tiên. Khi giao dịch T_i yêu cầu một mục dữ liệu hiện đang bị giữ bởi T_j , T_i được phép chờ chỉ nếu nó có nhãn thời gian lớn hơn của T_j , nếu không T_j bị cuộn lại (Wounded).

Một điều quan trọng là phải đảm bảo rằng, mỗi khi giao dịch bị cuộn lại, nó không bị "chết đói" (starvation) có nghĩa là nó sẽ không bị cuộn lại lần nữa và được phép tiến triển.

Cả hai sơ đồ **Wound-Wait** và **Wait-Die** đều tránh được sự chết đói: tại một thời điểm, có một giao dịch với nhãn thời gian nhỏ nhất. Giao dịch này không thể bị yêu cầu cuộn lại trong cả hai sơ đồ. Do nhãn thời gian luôn tăng và do các giao dịch không được gán nhãn

thời gian mới khi chúng bị cuộn lại, một giao dịch bị cuộn lại sẽ có nhãn thời gian nhỏ nhất (vào thời gian sau) và sẽ không bị cuộn lại lần nữa.

Tuy nhiên, có những khác nhau lớn trong cách thức hoạt động của hai sơ đồ:

- Trong sơ đồ **Wait-Die**, một giao dịch già hơn phải chờ một giao dịch trẻ hơn giải phóng mục dữ liệu. Như vậy, giao dịch già hơn có xu hướng bị chờ nhiều hơn. Ngược lại, trong sơ đồ **Wound-Wait,** một giao dịch già hơn không bao giờ phải chờ một giao dịch trẻ hơn.
- Trong sơ đồ **Wait-Die,** nếu một giao dịch T_i chết và bị cuộn lại vì nó đòi hỏi một mục dữ liệu bị giữ bởi giao dịch T_j , khi đó T_i có thể phải tái phát ra cùng dãy các yêu cầu khi nó khởi động lại. Nếu mục dữ liệu vẫn bị chiếm bởi T_j , T_i bị chết lần nữa. Như vậy, T_i có thể bị chết vài lần trước khi nhận được mục dữ liệu cần thiết. Trong sơ đồ **Wound-Wait,** giao dịch T_i bị thương và bị cuộn lại do T_j yêu cầu mục dữ liệu nó chiếm giữ. Khi T_i khởi động lại, và yêu cầu mục dữ liệu, bây giờ, đang bị T_j giữ, T_i chờ. Như vậy, có ít cuộn lại hơn trong sơ đồ **Wound-Wait.**

Một vấn đề nổi trội đối với cả hai sơ đồ là có những cuộn lại không cần thiết vẫn xảy ra.

Sơ đồ dựa trên timeout

Một cách tiếp cận khác để quản lý deadlock được dựa trên lock timeout. Trong cách tiếp cận này, một giao dịch đã yêu cầu một khóa phải chờ nhiều nhất một khoảng thời gian xác định. Nếu khóa không được cấp trong khoảng thời gian này, giao dịch được gọi là mãn kỳ (time out), giao dịch tự cuộn lại và khởi động lại. Nếu có một deadlock, một hoặc một vài giao dịch dính líu đến deadlock sẽ time out và cuộn lại, để các giao dịch khác tiến triển. Sơ đồ này nằm trung gian giữa phòng ngừa deadlock và phát hiện và khôi phục deadlock.

Sơ đồ timeout dễ thực thi và hoạt động tốt nếu giao dịch ngắn và nếu sự chờ đợi lâu là do deadlock. Tuy nhiên, khó quyết định được khoảng thời gian timeout. Sơ đồ này cũng có thể đưa đến sự chết đói.

Phát hiện deadlock và khôi phục

Nếu một hệ thống không dùng nghi thức phòng ngừa deadlock, khi đó sơ đồ phát hiện và khôi phục phải được sử dụng. Một giải thuật kiểm tra trạng thái của hệ thống được gọi theo một chu kỳ để xác định xem deadlock có xảy ra hay không. Nếu có, hệ thống phải khôi phục lại từ deadlock, muốn vậy hệ thống phải:

- Duy trì thông tin về sự cấp phát hiện hành các mục dữ liệu cho các giao dịch cũng như các yêu cầu mục dữ liệu chưa được giải quyết.
- Cung cấp một thuật toán sử dụng các thông tin này để xác định hệ thống đã đi vào

trạng thái deadlock chưa.

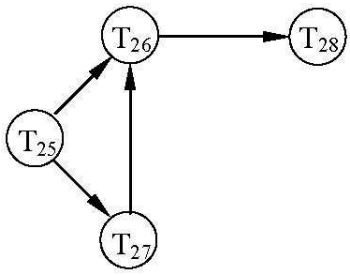
- Phục hồi từ deadlock khi phát hiện được deadlock đã xảy ra.

Phát hiện deadlock

Deadlock có thể mô tả chính xác bằng đồ thị định hướng được gọi là đồ thị chờ (wait for graph). Đồ thị này gồm một cặp G = < V, E >, trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh gồm tất cả các giao dịch trong hệ thống. Mỗi phần tử của E là một cặp $T_i \rightarrow T_j$, nó chỉ ra rằng T_i chờ T_j giải phóng một mục dữ liệu nó cần.

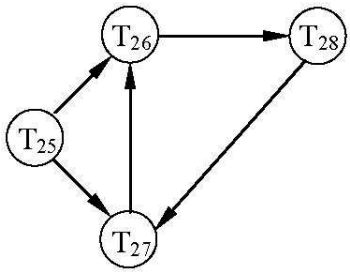
Khi giao dịch T_i yêu cầu một mục dữ liệu đang bị giữ bởi giao dịch T_j khi đó cung $T_i \rightarrow T_j$ được thêm vào đồ thị. Cạnh này bị xoá đi chỉ khi giao dịch T_j không còn giữ mục dữ liệu nào mà T_i cần.

Deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa một chu trình. Mỗi giao dịch tham gia vào chu trình này được gọi là bị deadlock. Để phát hiện deadlock, hệ thống phải duy trì đồ thị chờ và gọi theo một chu kỳ thủ tục tìm kiếm chu trình. Ta xét ví dụ sau:



Đồ thị chờ (phi chu trình)

Do đồ thị không có chu trình nên hệ thống không ở trong trạng thái deadlock. Bây giờ, giả sử T_{28} yêu cầu một mục dữ liệu được giữ bởi T_{27} , cung $T_{28} \rightarrow T_{27}$ được thêm vào đồ thị, điều này dẫn đến tồn tại một chu trình $T_{26} \rightarrow T_{28} \rightarrow T_{27} \rightarrow T_{26}$ có nghĩa là hệ thống rơi vào tình trạng deadlock và T_{26} , T_{27} , T_{28} bị deadlock.



Vấn đề đặt ra là khi nào thì chạy thủ tục phát hiện? câu trả lời phụ thuộc hai yêu tố sau:

1. Deadlock thường xảy ra hay không?
2. Bao nhiêu giao dịch sẽ bị ảnh hưởng bởi deadlock?

Nếu deadlock thường xảy ra, việc chạy thủ tục phát hiện diễn ra thường xuyên hơn. Các mục dữ liệu được cấp cho các giao dịch bị deadlock sẽ không sẵn dùng cho các giao dịch khác đến khi deadlock bị phá vỡ. Hơn nữa, số chu trình trong đồ thị có thể tăng lên. Trong trường hợp xấu nhất, ta phải gọi thủ tục phát hiện mỗi khi có một yêu cầu cấp phát không được cấp ngay.

Khôi phục từ deadlock

Khi thuật toán phát hiện xác định được sự tồn tại của deadlock, hệ thống phải khôi phục từ deadlock. Giải pháp chung nhất là cuộn lại một vài giao dịch để phá vỡ deadlock. Ba việc cần phải làm là:

1. **Chọn nạn nhân**. Đã cho một tập các giao dịch bị deadlock, ta phải xác định giao dịch nào phải cuộn lại để phá vỡ deadlock. Ta sẽ cuộn lại các giao dịch sao cho giá phải trả là tối thiểu. Nhiều nhân tố xác định giá của cuộn lại:

   a. Giao dịch đã tính toán được bao lâu và bao lâu nữa.

   b. Giao dịch đã sử dụng bao nhiêu mục dữ liệu.

   c. Giao dịch cần bao nhiêu mục dữ liệu nữa để hoàn tất.

   d. Bao nhiêu giao dịch bị cuộn lại.

2. **Cuộn lại (Rollback).** Mỗi khi ta đã quyết định được giao dịch nào phải bị cuộn lại, ta phải xác định giao dịch này bị cuộn lại bao xa. Giải pháp đơn giản nhất là cuộn lại toàn bộ: bỏ dở giao dịch và bắt đầu lại nó. Tuy nhiên, sẽ là hiệu quả hơn nếu chỉ cuộn lại giao dịch đủ xa như cần thiết để phá vỡ deadlock. Nhưng phương pháp này đòi hỏi hệ thống phải duy trì các thông tin bổ sung về trạng thái của tất cả các giao dịch đang chạy.
3. **Sự chết đói (Starvation).** Trong một hệ thống trong đó việc chọn nạn nhân dựa trên các nhân tố giá, có thể xảy ra là một giao dịch luôn là nạn nhân của việc chọn này và kết quả là giao dịch này không bao giờ có thể hoàn thành. Tình huống này được gọi là sự chết đói. Phải đảm bảo việc chọn nạn nhân không đưa đến chết đói. Một giải pháp xem số lần bị cuộn lại của một giao dịch như một nhân tố về giá.

Chương 5
LẬP TRÌNH CƠ SỞ DỮ LIỆU

I. Lập trình với ADO.NET

1. Giới thiệu

1.1. DataSet và DataTable:

- DataSet là trung tâm của kiến trúc ADO.NET, mục tiêu là hỗ trợ hiệu quả thao tác trên dữ liệu từ nhiều nguồn, cũng như tương tác dữ liệu trong mô hình ứng dụng nhiều lớp (multiple tier).
- Có thể xem DataSet như là một cấu trúc dữ liệu để lưu trữ dữ liệu trong bộ nhớ chính. DataSet chứa một tập các đối tượng DataTable (có cấu trúc logic tương tự như một bảng trong CSDL), các ràng buộc trên chúng và cả mối quan hệ giữa các bảng này.
- Một đối tượng DataTable (hoặc một view của nó thuộc lớp DataView), có thể được kết buộc với các control như ComboBox, DataList, DataGrid,…
- Các lớp đối tượng DataSet, DataTable, DataView, DataColumn,… nằm trong namespace System.Data.

1.2. Data Provider :

- Trong namespace *System.Data* có 3 namespace tương ứng với 3 loại Data Provider: Data Provider for SQL Server (*System.Data.SqlClient*), Data Provider for ODBC (*System.Data.Odbc*) và Data Provider for OLE DB (*System.Data.OleDb*).
- Ở đây ta sử dụng hệ quản trị SQL Server, nên sẽ sử dụng trực tiếp Data Provider for SQL Server (tất nhiên ta cũng có thể thông qua Provider for ODBC hoặc OLE DB để thao tác với CSDL SQL Server).
- Các lớp đối tượng chính để kết nối và thao tác với CSDL là trong namespace System.Data.SqlClient là: SqlConnection, SqlCommand, SqlDataReader, SqlDataAdapter. (Tương tự đối với hai Provider còn lại).

1.2.1. SqlConnection :

Một đối tượng thuộc lớp này thể hiện một kết nối đến CSDL. Các thông số để kết nối được chỉ định trong Connection String

Ví dụ:

//dùng Window Authentication

ConnectionString = "Data Source = .; Initial Catalog = Northwind; Integrated Security = SSPI"

hoặc :

ConnectionString= "Data Source = .; Initial Catalog = Northwind; User ID = ws01; Password="

(dấu "." chỉ local host)

1.2.2. *SqlCommand* : Một đối tượng thuộc lớp này thể hiện một lệnh thực thi trên hệ quản trị CSDL. Có thể thiết lập thuộc tính CommandType của đối tượng Command để chỉ ra lệnh được khai báo ở dạng text hay là tên stored procedure.

1.2.3. *SqlDataReader :* là lớp đối tượng dùng để đọc kết quả truy vấn được từ CSDL. Có thể xem SqlDataReader là một RecordSet chỉ có thể đọc và đọc tuần tự một chiều.

1.2.4. *SqlDataAdapter :* một đối tượng Data Adapter có thể xem như một cầu nối giữa DataSet và CSDL, để chuyển dữ liệu từ CSDL vào DataSet và cập nhật những thay đổi trên DataSet trở lại vào CSDL.

1.3. Trình tự thao tác CSDL với ADO.Net :

- Tạo lập và thiết lập các thông số cho đối tượng Connection (nếu chưa thiết lập trước đó).
- Mở kết nối bằng phương thức Open của đối tượng Connection.
- Thực hiện các công việc đọc/ghi với CSDL vừa kết nối tới.
- Đóng kết nối.

Lưu ý:

- ✓ *Không phải luôn luôn mở và đóng kết nối mỗi khi thực hiện một lệnh, có thể mở kết nối một lần và thực hiện nhiều lệnh trước khi đóng nó.*
- ✓ *Luôn sử dụng try và catch để bắt các lỗi phát sinh từ CSDL khi thực hiện các lệnh mở kết nối hoặc thực thi lệnh trong đối tượng command, nếu có lỗi hiển thị thông điệp lỗi dễ hiểu cho người sử dụng (NSD).*

2. Đọc dữ liệu

Trong phần này chỉ mô tả một số điểm cần lưu ý, sinh viên tự tìm hiểu cách thực hiện cụ thể trong các ví dụ và ebook được cung cấp hoặc MSDN.

2.1. Data Reader

DataReader là cách tốt (tiết kiệm tài nguyên - bộ nhớ) để đọc dữ liệu trong trường hợp chỉ cần lấy dữ liệu để hiển thị, không cần thao tác phức tạp hay thao tác trên nhiều tập

dữ liệu (không cần đến datatable hay dataset).

2.2. DataAdapter

- Như đã nói ở trên, ta có thể sử dụng DataAdapter như là một cầu nối để lấy dữ liệu từ CSDL vào Dataset. Ta có thể định nghĩa hoặc không định nghĩa trước bảng (đối tượng DataTable) và cấu trúc của bảng sẽ nhận dữ liệu. Nếu lệnh trả về n tập dữ liệu, n bảng tương ứng sẽ được tạo ra trong dataset để chứa các tập dữ liệu này.
- Để lấy dữ liệu, ta sử dụng phương thức Fill của đối tượng DataAdapter. Phương thức này có nhiều hàm quá tải (overload), hỗ trợ nhiều cách truyền tham số (DataSet, DataTable, DataSet và tên DataTable,…).
- Phương thức Fill có thể được gọi mà không cần mở kết nối trước, trong trường hợp này, kết nối tương ứng sẽ được mở và đóng lại ngay sau khi thực hiện xong việc lấy dữ liệu.

  Lưu ý: *Trong trường hợp ta muốn dữ liệu đưa vào đối tượng DataTable thỏa ràng buộc khóa chính (tự động loại bỏ dữ liệu trùng trên khóa chính nếu có), ta có thể thiết lập thuộc tính MissingSchemaAction của đối tượng DataAdapter là AddWithKey. Tuy nhiên, đặt lựa chọn này sẽ làm chậm đáng kể quá trình đọc dữ liệu, thay vào đó, nếu có thể ta nên định nghĩa trước cấu trúc và khóa chính của đối tượng DataTable này, rồi đọc bình thường, tác dụng cũng sẽ tương tự.*

2.3. ExecuteScalar :

Trong trường hợp câu truy vấn chỉ trả về một giá trị, ta sử dụng phương thức ExecuteScalar của đối tượng command để thực thi truy vấn và nhận giá trị trả về.

Lưu ý: *Trong trường hợp ta gọi thực thi một thủ tục (với CommandType là stored procedure), giá trị của các tham số output mà ta khai báo trong Parameters của command sẽ được cập nhật tương ứng.*

3. Ghi dữ liệu

3.1. ExecuteNonQuery

Cách thông dụng nhất để ghi dữ liệu là đưa trực tiếp các lệnh cập nhật dữ liệu (Insert, Update, Delete, lệnh tạo các đối tượng trong CSDL) (và hầu hết các lệnh T-SQL khác), hoặc tên của thủ tục thường trú thực hiện các công việc này vào một đối tượng Command, sau đó gọi phương thức ExecuteNonQuery của đối tượng Command để thực hiện.

3.2. DataAdapter

- DataAdapter thực hiện cập nhật CSDL theo cách ánh xạ những thay đổi trên

DataSet vào CSDL (thành các thao tác insert, update, delete tương ứng).

- Cơ sở của việc ánh xạ này là việc quản lý tình trạng của các dòng (DataRow) trong DataTable, thể hiện qua thuộc tính RowState. Khi một dòng trong DataTable được thêm, xóa, cập nhật, nó sẽ có trạng thái tương ứng là Inserted, Deleted, và Updated, sau khi phương thức AcceptChanges được gọi, nó trở lại trạng thái UnChanged.
- Để thực hiện cập nhật ta gán InsertCommand, DeleteCommand và UpdateCommand cho đối tượng DataAdapter, sau đó gọi phương thức Update của nó. Khi đó, tất cả những thay đổi trên DataSet sẽ được ánh xạ về CSDL theo cách mà ta chỉ định trong các Commands (xem thêm ví dụ).
- Phương thức Update của DataAdapter cũng có nhiều hàm quá tải, cho phép ta lựa chọn Update một dòng, một table hay toàn bộ DataSet.

II. Thiết kế chức năng đọc/ ghi dữ liệu

1. Thiết kế chức năng ghi dữ liệu:

- Giao diện đáp ứng yêu cầu chung: dễ sử dụng (dễ hiểu, gợi nhớ, không "bẫy" người sử dụng), có tính thẩm mỹ, tính tiện dụng (cho phép người sử dụng thao tác nhanh: sắp xếp các mục hợp lý, hỗ trợ di chuyển bằng phím tab, hỗ trợ phím tắt,…).
- Kiểm tra chặt chẽ các ràng buộc toàn vẹn, đảm bảo thao tác thêm/cập nhật sau khi thực hiện xong không gây ra mâu thuẫn trong CSDL.
- Cung cấp cách thức nhập liệu phù hợp nhất với nghiệp vụ thực tế.

  Ví dụ: nếu thực tế NSD nhập liệu cho một tập đối tượng cùng lúc, mỗi đối tượng có ít thuộc tính và xử lý đơn giản thì nên nhập liệu bằng lưới (grid). Nếu đối tượng có nhiều thuộc tính hoặc xử lý phức tạp thì có thể nhập riêng từng đối tượng, nhưng nên hiển thị song song một lưới chứa danh sách các đối tượng đã nhập để NSD có thể kiểm tra lại khi cần.
- Lựa chọn cách xử lý để giảm thiểu thời gian làm việc của NSD (tất nhiên vẫn phải đảm bảo tính an toàn và đúng đắn): thời điểm kiểm tra ràng buộc toàn vẹn (xem mục 2), ghi nhận dữ liệu một lần hay sau mỗi lần NSD nhập xong một đối tượng, thời điểm mở và đóng kết nối với CSDL,…

2. Kiểm tra ràng buộc toàn vẹn

Khi xây dựng chức năng nhập liệu (cũng như cập nhật dữ liệu), phải đảm bảo rằng các ràng buộc toàn vẹn không bị vi phạm. Tuy nhiên, cần phải lưu ý cân nhắc xem kiểm tra

ràng buộc toàn vẹn ở mức nào, ở thời điểm nào, và dưới hình thức nào là hợp lý.

Một số nguyên tắc:

- Thiết kế giao diện sao cho có thể hạn chế lỗi của người sử dụng, ví dụ: sử dụng ComboBox, Check box,… để đảm bảo ràng buộc tham chiếu và ràng buộc miền giá trị rời rạc. Tuy nhiên, cũng cần cân nhắc kỹ vì giao diện quá "cứng", quá nghiêm ngặt sẽ cản trở và làm chậm thao tác của NSD.
- Trong các trường hợp có thể, cố gắng xử lý kiểm tra ràng buộc toàn vẹn ở mức trên (tầng giao diện, kế đến là tầng nghiệp vụ), phản hồi ngay và rõ ràng cho NSD nếu có lỗi sai. Nếu để chương trình đẩy dữ liệu xuống CSDL, sau khi nhận báo lỗi từ CSDL mới phản hồi cho NSD thì sẽ mất nhiều thời gian.
  - ✓ Các trường hợp thông thường có thể kiểm tra ràng buộc toàn vẹn ở tầng giao diện hoặc nghiệp vụ: ràng buộc đơn giản như miền giá trị, liên thuộc tính trên một quan hệ,…, ràng buộc phức tạp hơn (liên bộ, liên thuộc tính) nhưng các dữ liệu liên quan cần thiết để kiểm tra nó đã được chương trình đọc sẵn trước đó.
  - ✓ Nếu việc kiểm tra ràng buộc cần các dữ liệu liên quan khác chưa được chương trình đọc sẵn trước đó, kiểm tra ở CSDL trong đa số trường hợp sẽ hiệu quả hơn là đọc các dữ liệu đó lên để kiểm tra ở tầng trên.
- Nếu các ràng buộc của ứng dụng có các tham số đặt trong bảng tham số ở CSDL hoặc trong tập tin, nên đọc các tham số này lên một lần và sử dụng lại cho các lần nhập liệu, thay vì phải đọc lại từ CSDL hay tập tin mỗi khi nhập một đối tượng liên quan.
- Nếu các đối tượng trong CSDL được quản lý bằng mã, chương trình nên có cơ chế tự động phát sinh các mã này, để tránh gây ra các vi phạm trên ràng buộc khóa chính (Trừ những trường hợp nghiệp vụ thực tế đòi hỏi mã cho NSD ghi).

3. Thiết kế chức năng đọc dữ liệu:

Có thể chia thành hai dạng chức năng đọc dữ liệu chính: đọc dữ liệu lên Form và đọc dữ liệu lên báo biểu (Report). Dữ liệu đọc lên Form có thể được thay đổi và cập nhật xuống CSDL, dữ liệu đọc lên báo biểu chỉ nhằm mục đích hiển thị thông tin.

Khi đọc dữ liệu lên Form, ta sử dụng các đối tượng dữ liệu của ngôn ngữ lập trình (DataSet, DataTable, DataReader…) để lưu trữ tạm dữ liệu trong quá trình hiển thị. Ta có thể sao chép dữ liệu từ các đối tượng dữ liệu vào các đối tượng hiển thị như Textbox, DataGridView, Combobox,… trên form, hoặc kết buộc trực tiếp các đối tượng dữ liệu với đối tượng hiển thị. Với cách thứ hai, khi dữ liệu được cập nhật: khi người sử dụng thay đổi dữ liệu trên các đối tượng hiển thị, dữ liệu trong các đối tượng dữ liệu cũng sẽ thay

đổi theo.

Khi đọc dữ liệu lên report, ngoại trừ những trường hợp phức tạp mà dữ liệu được lấy từ nhiều nguồn: từ CSDL, từ file, từ form khác,…, thông thường ta đưa dữ liệu trực tiếp lên report mà không thông qua các đối tượng dữ liệu để giảm bớt một khoản chi phí trung gian. Các môi trường thiết kế report (Crystal report,…) thường hỗ trợ riêng phương thức đọc dữ liệu, không phụ thuộc vào các cách thức đọc dữ liệu của môi trường lập trình, dù rằng trong một số trường hợp vẫn hỗ trợ tích hợp (như Crystal report for .NET, trong đó dataset của .NET được Crystal report xem như một nguồn dữ liệu).

Tuy nhiên, cho dù đọc dữ liệu lên form hay lên report, người thiết kế vẫn phải chú ý đến những nguyên tắc cơ bản sau:

- Hạn chế đọc dữ liệu nhiều lần từ CSDL: nếu có thể, đọc các dữ liệu cần thiết một lần thay vì đọc thành nhiều lần từ CSDL để giảm chi phí thiết lập kết nối với dữ liệu. Hơn nữa, ở phía CSDL, việc đọc 1 lần n dòng dữ liệu sẽ nhanh hơn đọc k lần, mỗi lần n/k dòng.
    - ✓ Ví dụ: trên form có một lưới liệt kê danh sách các lớp trong trường theo từng khối. Mỗi lần NSD chọn lại khối trong một combobox, danh sách các lớp trong lưới được lọc lại theo khối đó. Vậy ta nên đọc một lần tất cả các lớp trong trường từ CSDL vào một đối tượng dữ liệu (DataTable chẳng hạn), sau đó tùy yêu cầu mà hiển thị phần dữ liệu phù hợp, hay mỗi lần NSD chọn khối ta lại đọc lại từ CSDL? Câu trả lời cho tình huống này và những tình huống tương tự là: nếu tất cả dữ liệu không quá lớn (vài trăm dòng trở xuống), ta nên đọc tất cả lên ứng dụng một lần.
- Giảm thiểu lượng dữ liệu chuyển từ CSDL lên ứng dụng: đọc đúng dữ liệu cần thiết, không đọc thừa.
- Dữ liệu phải được hiển thị theo định dạng thân thiện với người sử dụng. Trong CSDL, để tối ưu hoá lưu trữ và truy xuất, dữ liệu có thể ở dạng mà người sử dụng không hiểu hoặc không cần (ví dụ những mã đối tượng được phát sinh thêm để phục vụ việc lưu trữ, định dạng ngày giờ, …). Người sử dụng không cần biết dữ liệu bên dưới được lưu trữ như thế nào, cấu trúc ra sao, họ chỉ cần thấy được những thông tin được hiển thị và sắp xếp theo cách quen thuộc và tiện lợi nhất cho nghiệp vụ của họ.

4. Lưu ý chung:

- Nếu xử lý có thể mất nhiều thời gian, nên có phản hồi để NSD biết rằng chương trình vẫn đang làm việc (progress bar, waiting message,…).

- Nên sử dụng thủ tục thường trú thay vì viết các lệnh SQL trực tiếp trong mã nguồn chương trình, vì các thủ tục đã được biên dịch trước nên thực hiện nhanh hơn.
- Dùng try…catch để bắt các ngoại lệ (exception) có thể xảy ra, nhất là khi thực hiện các thao tác đóng/mở kết nối và đọc/ghi trên CSDL và thông báo lỗi theo cách mà NSD có thể hiểu được.
- Thiết kế chương trình theo mô hình 3 lớp, thiết kế các module nhỏ gọn, rõ ràng để dễ kiểm tra, bảo trì, và tăng khả năng tái sử dụng.

III. Tạo báo biểu với Crystal Report

1. Giới thiệu

Crystal Report là một phần mềm hỗ trợ lập báo biểu từ đơn giản đến phức tạp. Hiện nay ngoài các phiên bản Crystal Report riêng (Standard, Professional, Developer, Advanced), còn có một phiên bản đặc biệt tích hợp với Visual Studio.NET.

Ngoài việc cung cấp môi trường để thiết kế báo biểu, Crystal Report 9 phiên bản Developer và Advanced có hỗ trợ Report Creation API, cho phép người lập trình ứng dụng có thể tạo lập/ thay đổi cấu trúc/ nội dung report lúc runtime.

Phiên bản Crystal Report tích hợp với .NET có hỗ trợ Run Time Object Model, cho phép thực hiện một số thay đổi trên report khi chương trình thực hiện (truyền tham số, thay đổi các thông số để đăng nhập database, thay đổi kích thước và vị trí các đối tượng), ta cũng có thể gán các đối tượng dữ liệu của .NET (ví dụ DataTable) làm data source cho report. Cửa sổ thiết kế report được tích hợp vào môi trường phát triển ứng dụng .NET.

Crystal Report for .NET không hoàn toàn là một phần của Crystal Report (xét phiên bản 9), nó có một số tính năng hỗ trợ riêng cho .NET, đồng thời Crystal Report 9 cũng có một số tính năng riêng mà Crystal Report for .NET không hỗ trợ. Cửa sổ thiết kế report của Crystal Report 9 cũng đầy đủ và dễ dùng hơn. Ta có thể thiết kế một report bằng Crytal Report 9, sau đó đưa vào ứng dụng .NET.

2. Xây dựng một report cơ bản

2.1. *Các phần (session) của report*:

Một báo biểu (report) gồm có những phần chính như sau :

- Report header: phần thông tin đầu tiên của báo biểu. Một báo biểu thường sẽ gồm nhiều trang, report header là phần xuất hiện chỉ một lần ở trang đầu tiên của toàn báo biểu. Ví dụ như báo cáo có tiêu đề **"**Báo cáo thu chi tháng 12 năm 2008**"** thì tiêu đề sẽ được đặt trong phần report header
- Page header: phần hiển thị thông tin xuất hiện ở đầu mỗi trang của báo biểu. Ví

dụ như một báo cáo dạng bảng có nhiều cột như: STT, Tên, Địa chỉ, SĐT… thì đầu mỗi trang cần lặp lại những tên cột để người đọc nhận biết dễ dàng ý nghĩa mỗi cột. Khi đó các tiêu đề cột được đặt vào phần page header.

- Details: phần hiển thị thông tin chi tiết của báo biểu. Một báo biểu thường bao gồm nhiều mục với vai trò như nhau tương ứng với các mẩu tin (record) của database mà báo biểu sử dụng. Phần details sẽ liệt kê những mẩu tin đó. Ví dụ báo cáo thu chi thì các mục thu chi sẽ được liệt kê trong phần details.
- Báo biểu footer: phần hiển thị thông tin xuất hiện chỉ một lần ở cuối báo biểu. Ví dụ: các thông tin như tổng số (grand total), người lập báo cáo là ai, tại đâu, vào ngày nào, sẽ được đặt ở báo biểu footer.
- Page footer: phần thông tin xuất hiện cuối mỗi trang. Ví dụ số trang được đặt ở page footer.

2.2. *Các loại đối tượng trong report*

Các đối tượng trong báo biểu được quản lý qua cửa sổ Field explorer (View → Field Explorer). Các đối tượng này có thể được chọn để đưa vào hiển thị trong báo biểu. Gồm có:

- Database fields: Các trường thuộc dạng CSDL (có thể là table, stored procedure, SQL command). Thông thường các trường trong mục này sẽ được hiển thị trong phần detail của báo biểu.
- Formula fields: các trường tạo thành từ việc thiết lập các công thức. Ta có thể tạo mới một đối tượng formula bằng cách sử dụng Formula Editor hoặc Formula Expert. Formula được viết bằng cú pháp Crystal hoặc Basic. Có hai loại:
  - ✓ Formula Field : Là các trường mà dữ liệu có được nhờ tính toán trên các trường khác.

    Ví dụ: Khi làm một report báo cáo hóa đơn bán hàng, giả sử CSDL chỉ lưu trữ giá và số lượng của mặt hàng mua trong hóa đơn mà không lưu trữ thành tiền, khi đó ta có thể tạo một Formula field "Thành tiền" được tính bằng công thức:

    Thành tiền = Giá * Số lượng.

    Khi đó ta có thể tạo report với cột thành tiền (mặc dù không được lưu trong database).
  - ✓ Selection Formula: Là các công thức để chọn dữ liệu, gồm có chọn bộ (tương tự các điều kiện chọn trong mệnh đề where) hay chọn nhóm (tương tự điều kiện chọn trong mệnh đề having). Selection formula có thể được xây dựng trong cửa

sổ Formula Editor/ Formula Expert hoặc Select Expert.

- SQL Expression field: SQL Expression cũng là trường mà dữ liệu của nó được tính toán từ những trường khác (ví dụ count, sum,… hay một công thức tính toán bất kỳ). Tuy nhiên, khác với Formula field, SQL Exppression được gửi về xử lý ở CSDL (và do đó phải được viết bằng cú pháp SQL), kết quả được đưa trả về báo biểu qua SQL Expression field.
- Parameter fields: các trường tham số cho report. Đây có thể là một tham số ta tự khai báo, hoặc Crystal Report sẽ tự động thêm vào khi ta đưa một thủ tục thường trú có tham số vào trong database field. Lưu ý, khi chạy báo biểu trong Crystal, những trường tham số sẽ được hỏi giá trị, ta cần nhập vào ngay trong Crystal để hiển thị tạm thời.
- Group Name Field : Các thuộc dùng để gom nhóm dữ liệu trong report.

  Ví dụ : Ta muốn hiển thị danh sách học sinh theo từng lớp, khi đó thuộc tính Lớp sẽ là một đối tượng trong Group Name Field (Thêm một nhóm bằng cách chọn Insert Group hoặc chọn Group Expert từ menu database).
- Running Total Field : trường chứa giá trị tổng hợp (aggregate) : max, min, sum, count,…
- Special fields: các trường đặt biệt có sẵn của Crystal như số trang, ngày hiện tại… Thông thường những trường này sẽ được hiển thị trong những phần header, footer.

Ngoài ra, ta có thể đưa vào report những kiểu đối tượng khác như text(label), hình ảnh, đường kẻ, biểu đồ,…

2.3. *Xây dựng report:*

a. Thiết kế report : xác định các thông tin cần hiển thị, cách bố trí, tổ chức thông tin.

Một số tiêu chí khi thiết kế report :

- Đáp ứng mục tiêu nghiệp vụ, phù hợp với người sử dụng (thông tin kết xuất phải là thông tin mà người sử dụng có thể hiểu được, khớp với nghiệp vụ thực tế).
- Số lượng vừa đủ, sắp xếp, gom nhóm hợp lý, tránh làm NSD bị rối mắt vì báo biểu dày đặc dữ liệu.
- Trình bày dữ liệu đúng vị trí.
- Trình bày dữ liệu đúng lúc cần (ví dụ hiển thị lại tên các cột dữ liệu khi sang trang mới).

b. Xây dựng report theo bản thiết kế: Xem hướng dẫn chi tiết trong ebook và tài liệu kỹ thuật của Crystal Report.

Một số lưu ý khi xây dựng report :

- Nếu dữ liệu được hiển thị trong báo biểu sẽ thay đổi trong các lần báo biểu được hiển thị, bỏ lựa chọn Save Data with Report (File → Report Options)
- Khi sử dụng Formula, các tính toán, chọn dòng/nhóm sẽ được thực hiện bởi Crystal Report sau khi đã đọc toàn bộ dữ liệu liên quan từ CSDL lên. Điều này có thể làm nặng đường truyền và tăng chi phí đọc ghi nếu dữ liệu liên quan lớn. Ngoài ra, một hệ quản trị CSDL luôn có cơ chế điều chỉnh để cải thiện tốc độ xử lý trong trường hợp dữ liệu lớn, vì vậy, việc thực hiện xử lý chọn trên dòng/nhóm hay các tính toán mà dữ liệu kết quả nhỏ hơn dữ liệu trung gian (count/sum, max,…) được thực hiện ở hệ quản trị CSDL sẽ tối ưu hơn. Do đó, ta nên chuyển các công thức thành SQL Command/Expression hay thủ tục thường trú bất cứ khi nào có thể.
- Nên tận dụng stored prcedure vì các ích lợi của nó so với việc thực thi trực tiếp các câu lệnh SQL (hỗ trợ bảo mật, được biên dịch trước, tái sử dụng và dễ bảo trì).

2.4. *Xem trước report:*

Ta có thể xem trước report bằng cách nhấn F5 (Refresh Report Data) và chuyển qua tab preview.

3. Hiển thị report trong một ứng dụng .NET

3.1. *Crystal Report Viewer :*

Để hiển thị Crystal report trên .NET Windows form, ta sử dụng control CrystalReportViewer (có thể kéo thả control này từ toolbox vào form).

3.2. *Kết buộc báo biểu vào Report Viewer :*

Giả sử ta đã sử dụng một control CrystalReportViewer tên là rptViewer. Ta có thể sử dụng một trong những cách sau để đưa một Crystal report vào ứng dụng và kết buộc vào viewer để hiển thị :

a. Kết buộc dang "Untyped report": Giả sử ta có sẵn một report tên MyReport rên đĩa C.

- ✓ Kết buộc bằng tên: Gán tên và đường dẫn đến báo biểu cho thuộc tính ReportSource của rptviewer.

rptViewer.ReportSource = "C:\\ MyReport.rpt"

✓ Kết buộc report object :

- Project → Add reference. Chọn CrystalDecisions.CrystalReports.Engine
- Khai báo đối tượng thuộc lớp ReportDocument, tải báo biểu vào đối tượng này và kết buộc vào rptViewer

```
(using CrystalDecisions.CrystalReports.Engine;)
                    ReportDocument oRpt=null;
oRpt = new ReportDocument();
oRpt.Load("c:\\MyReport.rpt");
rptViewer.ReportSource = oRpt;
```

b. Kết buộc dạng "strongly-typed report":

- Nếu tạo báo biểu ngay trong .NET : Project → Add New Item.
- Nếu đã có sẵn report: Project →Add existing Item. Tìm và chọn report muốn đưa vào.

Với cách này, trong project sẽ xuất hiện một lớp đối tượng mới tương ứng với báo biểu mới thêm vào (lớp đối tượng này kế thừa từ lớp ReportClass, và ReportClass kế thừa ReportDocument). Một đối tượng thuộc lớp này, ngoài các thuộc tính và phương thức kế thừa từ lớp ReportDocument, còn có một số thuộc tính thể hiện các thông tin riêng của từng report: các session, tham số,… Tuy nhiên các thuộc tính này phần lớn là read only. Để thay đổi một số thông tin trên báo biểu (ví dụ truyền tham số), ta vẫn chủ yếu sử dụng các phương thức và thuộc tính của lớp ReportDocument.

Giả sử ta đã thêm một strongly-typed report và project và có được lớp rptMyReport tương ứng, ta kết buộc report vào viewer như sau:

```
rptMyReport report = new rptMyReport();
rptViewer.ReportSource = report;
```

3.3. *Thay đổi thông tin kết nối đến nguồn dữ liệu:*

Giả sử ta có các chuỗi ServerName, DatabaseName, UserID, Password lưu thông tin để kết nối đến data source cho report, ta gán các thông tin này cho report như sau :

```
(using CrystalDesisions.Shared)
```

```
TableLogOnInfo Info;
 for (int i=0;i<report.Database.Tables.Count;i++)
{

  Info = report.Database.Tables[i].LogOnInfo;
  Info.ConnectionInfo.ServerName = ServerName;  // ”.” nếu là localhost
  Info.ConnectionInfo.DatabaseName =DatabaseName;
  Info.ConnectionInfo.UserID = UserID;
  report.Database.Tables[i].ApplyLogOnInfo(Info);

  /*sửa lại location của table (có dạng Database.Owner.TenTable/storedProcedure) cho
  khớp với tên database và owner mới */
  string location = report.Database.Tables[i].Location;
  location = location.Substring(location.LastChỉ mụcOf(".")+1);
  report.Database.Tables[i].Location = location;//report sẽ tự điền vào database và owner
  mới
  }
```

Lưu ý :

- Nếu kết nối sử dụng Windows Authentication (Integrated Security), ta để trống UserID.
- Trong đoạn lệnh trên, nên đặt lệnh report.Database.Tables[i].Location = location; trong try…catch để bắt Exception có thể xảy ra nếu các thông tin kết nối được gán không hợp lệ:

```
try
{
        report.Database.Tables[i].Location = location;
}
catch (EngineException ex)
{
    //báo lỗi
    //hiển thị dialog yêu cầu NSD nhập lại các thông số để kết nối
```

}

3.4. *Truyền giá trị cho tham số*

Giả sử trong report có tham số @MaNganh, dùng để đọc danh sách sinh viên của một ngành cụ thể. Trong chương trình ta truyền giá trị cho tham số này như sau :

```
(string MaNganh = "CNTT";)
  ParameterDiscreteValue ParamValue = new ParameterDiscreteValue();
  ParamValue.Value = MaNganh;
  ParameterValues values = new ParameterValues();
  values.Add(ParamValue);
  report.DataDefinition.ParameterFields["@MaNganh"].ApplyCurrentValues(values);
  rptViewer.ReportSource = report;
```

(Lưu ý : Lệnh rptViewer.ReportSource = report; phải được thực hiện **sau cùng**, sau khi đã cấu hình cho đối tượng report (thay đổi thông số kết nối data source, truyền giá trị tham số,…)).

BÀI TẬP QUẢN LÝ SINH VIÊN

Cho CSDL như sau:

Khoa : Mỗi khoa có một mã khoa để quản lý, một tên khoa và ghi nhận năm thành lập khoa.

maKhoa	tenKhoa	namThanhLap
varchar(10)	nvarchar(100)	int

KhoaHoc : Mỗi khóa học có một mã để quản lý, năm bắt đầu khoá học và năm kết thúc khóa học.

MaKhoaHoc	namBatDau	namKetThuc
varchar(10)	int	int

SinhVien : Mỗi sinh viên có một mã để quản lý và thuộc về một lớp nào đó (xác định bởi maLop).

MaSV	hoTen	namSinh	danToc	maLop
varchar(10)	nvarchar(100)	int	nvarchar(20)	varchar(10)

ChuongTrinh : Mỗi chương trình có một mã để quản lý và một tên chương trình.

MaCT	tenChuongTrinh
varchar(10)	nvarchar(100)

MonHoc : Mỗi môn học có một mã để quản lý và thuộc về một khoa nào đó (xác định bởi maKhoa)

MaMH	tenMonHoc	maKhoa
varchar(10)	nvarchar(100)	varchar(10)

KetQua : Mỗi kết quả thi ghi nhận điểm của một sinh viên làm bài thi cho 1 môn học nào đó ở một lần thi cụ thể. (1 sinh viên có thể thi 1 môn nào đó trên 1 lần)

MaSV	MaMH	lanThi	diem
varchar(10)	varchar(10)	Int	float

GiangKhoa: Mỗi dòng trong bảng này cho biết một môn học được giảng dạy tại một khoa nào đó trong một chương trình nào đó. Trong chương trình này, môn học đó được quy định số tiết lý thuyết và thực hành cụ thể và tương đương với bao nhiêu tín chỉ. (soTinChi bao gồm cả tín chỉ lý thuyết lẫn thực hành).

maCT	maKhoa	maMH	namHoc	hocKy	soTietLyThuyet	soTietThucHanh	soTinChi
varchar(10)	varchar(10)	varchar(10)	int	int	int	Int	int

Lop : Một lớp có một mã lớp để quản lý, thuộc về một khoa nào đó và mở ra cho một khóa học nhất định, trong một chương trình nhất định. Số thứ tự được đánh tăng dần cho các lớp cùng khoá học, cùng khoa và cùng chương trình.

MaLop	maKhoaHoc	maKhoa	maCT	soThuTu
varchar(10)	varchar(10)	varchar(10)	varchar(10)	int

Yêu cầu:

1. Hãy cài đặt CSDL trên
2. Hãy cài đặt khóa chính và khóa ngoại của CSDL trên.
3. Nhập các bộ dữ liệu sau :

Khoa

MaKhoa	tenKhoa	namThanhLap
CNTT	Công nghệ thông tin	1995
VL	Vật Lý	1970

Khóa học:

MaKhoaHoc	namBatDau	namKetThuc
K2002	2002	2006
K2003	2003	2007
K2004	2004	2008

SinhVien

MaSV	hoTen	namSinh	danToc	maLop
0212001	Nguyễn Vĩnh An	1984	Kinh	TH2002/01
0212002	Nguyên Thanh Bình	1985	Kinh	TH2002/01
0212003	Nguyễn Thanh Cường	1984	Kinh	TH2002/02
0212004	Nguyễn Quốc Duy	1983	Kinh	TH2002/02
0311001	Phan Tuấn Anh	1985	Kinh	VL2003/01
0311002	Huỳnh Thanh Sang	1984	Kinh	VL2003/01

ChuongTrinh

MaCT	tenChuongTrinh
CQ	Chính Qui

MonHoc

MaMH	tenMonHoc	maKhoa
THT01	Toán Cao cấp A1	CNTT
VLT01	Toán cao cấp A1	VL
THT02	Toán rời rạc	CNTT
THCS01	Cấu trúc dữ liệu 1	CNTT
THCS02	Hệ điều hành	CNTT

KetQua

maSV	maMH	lanThi	diem
0212001	THT01	1	4
0212001	THT01	2	7
0212002	THT01	1	8
0212003	THT01	1	6
0212004	THT01	1	9
0212001	THT02	1	8
0212002	THT02	1	5.5
0212003	THT02	1	4
0212003	THT02	2	6
0212001	THCS01	1	6.5
0212002	THCS01	1	4
0212003	THCS01	1	7

GiangKhoa

maCT	maKhoa	maMH	namHoc	hocKy	soTietLyThuyet	soTietThucHanh	soTinChi
CQ	CNTT	THT01	2003	1	60	30	5
CQ	CNTT	THT02	2003	2	45	30	4
CQ	CNTT	THCS01	2004	1	45	30	4

Lop

MaLop	maKhoaHoc	maKhoa	maCT	soThuTu
TH2002/01	K2002	CNTT	CQ	1
TH2002/02	K2002	CNTT	CQ	2
VL2003/01	K2003	VL	CQ	1

4. Viết các câu truy vấn sau :
   - 4.1. Danh sách các sinh viên khoa "Công nghệ Thông tin" khoá 2002-2006
   - 4.2. Cho biết các sinh viên (MSSV, họ tên ,năm sinh) của các sinh viên học sớm hơn tuổi qui định (theo tuổi qui định thi sinh viên đủ 18 tuổi khi bắt đầu khóa học)
   - 4.3. Cho biết sinh viên khoa CNTT, khoá 2002-2006 chưa học môn cấu trúc dữ liệu 1
   - 4.4. Cho biết sinh viên thi không đậu (Diem <5) môn cấu trúc dữ liệu 1 nhưng chưa thi lại.
   - 4.5. Với mỗi lớp thuộc khoa CNTT, cho biết mã lớp, mã khóa học, tên chương trình và số sinh viên thuộc lớp đó
   - 4.6. Cho biết điểm trung bình của sinh viên có mã số 0212003 (điểm trung bình chỉ tính trên lần thi sau cùng của sinh viên)
5. Hãy viết các function sau :
   - 5.1. Với 1 mã sinh viên và 1 mã khoa, kiểm tra xem sinh viên có thuộc khoa này không (trả về đúng hoặc sai)
   - 5.2. Tính điểm thi sau cùng của một sinh viên trong một môn học cụ thể
   - 5.3. Tính điểm trung bình của một sinh viên (chú ý : điểm trung bình được tính dựa trên lần thi sau cùng), sử dụng function 5.2 đã viết
   - 5.4. Nhập vào 1 sinh viên và 1 môn học, trả về các điểm thi của sinh viên này trong các lần thi của môn học đó.
   - 5.5. Nhập vào 1 sinh viên, trả về danh sách các môn học mà sinh viên này phải học.

6. Hãy viết các Stored Procedure sau:
   - 6.1. In danh sách các sinh viên của 1 lớp học
   - 6.2. Nhập vào 2 sinh viên, 1 môn học, tìm xem sinh viên nào có điểm thi môn học đó lần đầu tiên là cao hơn.
   - 6.3. Nhập vào 1 môn học và 1 mã sv, kiểm tra xem sinh viên có đậu môn này trong lần thi đầu tiên không, nếu đậu thì xuất ra là "Đậu", không thì xuất ra "Không đậu"
   - 6.4. Nhập vào 1 khoa, in danh sách các sinh viên (mã sinh viên, họ tên, ngày sinh) thuộc khoa này.
   - 6.5. Nhập vào 1 sinh viên và 1 môn học, in điểm thi của sinh viên này của các lần thi môn học đó.
     Ví dụ: Lần 1 : 10
     Lần 2: 8
   - 6.6. Nhập vào 1 sinh viên, in ra các môn học mà sinh viên này phải học.
   - 6.7. Nhập vào 1 môn học, in danh sách các sinh viên đậu môn này trong lần thi đầu tiên.
   - 6.8. In điểm các môn học của sinh viên có mã số là maSinhVien được nhập vào.
     (Chú ý: điểm của môn học là điểm thi của lần thi sau cùng)

6.8.1. Chỉ in các môn đã có điểm
6.8.2. Các môn chưa có điểm thì ghi điểm là null
6.8.3. Các môn chưa có điểm thì ghi điểm là <chưa có điểm>

Thêm 1 quan hệ

XepLoai

maSV	diemTrungBinh	ketQua	hocLuc

6.9. *Đưa dữ liệu vào bảng xếp loại. Sử dụng function 5.3 đã viết ở trên*
Qui định : ketQua của sinh viên là "Đạt' nếu *diemTrungBinh* (chỉ tính các môn đã có điểm) của sinh viên đó lớn hơn hoặc bằng 5 và không quá 2 môn dưới 4 điểm, ngược lại thì kết quả là không đạt
Đối với những sinh viên có *ketQua* là "Đạt' thì *hocLuc* được xếp loại như sau:
- *diemTrungBinh* >= 8 thì *hocLuc* là "Giỏi"
- 7 < = *diemTrungBinh* < 8 thì *hocLuc* là "Khá"
- Còn lại là "Trung bình"

6.10. Với các sinh viên có tham gia đầy đủ các môn học của khoa, chương trình mà sinh viên đang theo học, hãy in ra điểm trung bình cho các sinh viên này.
(Chú ý: Điểm trung bình được tính dựa trên điểm thi lần sau cùng). Sử dụng function 5.3 đã viết ở trên

7. Hãy cài đặt các ràng buộc toàn vẹn sau (bằng check constraint, unique constraint, rule hoặc trigger):

Miền giá trị

7.1. ChuongTrinh.ma chỉ có thể là 'CQ' hoặc 'CD' hoặc 'TC'
7.2. Chỉ có 2 học kỳ là 'HK1' và 'HK2'
7.3. Số tiết lý thuyết (GiangKhoa.soTietLyThuyet) tối đa là 120
7.4. Số tiết thực hành (GiangKhoa.soTietThucHanh) tối đa là 60
7.5. Số tín chỉ (GiangKhoa.soTinChi) của một môn học tối đa là 6
7.6. Điểm thi (KetQua.diem) được chấm theo thang điểm 10 và chính xác đến 0.5 (làm bằng 2 cách: kiểm tra và báo lỗi nếu không đúng qui định; tự động làm tròn nếu không đúng qui định về độ chính xác)

Liên thuộc tính trên 1 quan hệ

7.7. Năm kết thúc khóa học phải lớn hơn hoặc bằng năm bắt đầu
7.8. Số tiết lý thuyết của mỗi giảng khóa không nhỏ hơn số tiết thực hành

Liên bộ trên 1 quan hệ

7.9. Tên chương trình phải phân biệt.
7.10. Tên khoa phải phân biệt
7.11. Tên môn học phải duy nhất
7.12. Sinh viên chỉ được thi tối đa 2 lần cho một môn học
7.13. Liên thuộc tính trên nhiều quan hệ
7.14. Năm bắt đầu khóa học của một lớp không thể nhỏ hơn năm thành lập của khoa quản lý lớp đó
7.15. Sinh viên chỉ có thể dự thi các môn học có trong chương trình và thuộc về khoa mà sinh viên đó đang theo học

Tổng hợp

7.16. Hãy bổ sung vào quan hệ LOP thuộc tính SISO và kiểm tra sĩ số của một lớp phải bằng số lượng sinh viên đang theo học lớp đó

BÀI TẬP LẬP TRÌNH CƠ SỞ DỮ LIỆU

Cho cơ sở dữ liệu sau (có tên tập tin cơ sở dữ liệu **QLThuVien**):

- **NhaXuatBan(MANXB, TenNXB)**: *Mỗi nhà xuất bản có một mã số (MANXB) để phân biệt và tên nhà xuất bản (TenNXB)*
- **TheLoai(MaTL, TenTL):** *Sách được phân loại theo thể loại. Mỗi thể loại có mã thể loại (MaTL), tên thể loại (TenTL).*
- **Sach(MaSach, TuaDe, MANXB, TacGia, SoLuong, NgayNhap, MaTL):** *Mỗi cuốn sách có một mã số để phân biệt (MaSach), tên sách (TuaDe), do một nhà xuất bản xuất bản (MANXB), tác giả (TacGia), số lượng bản hiện có trong thư viện (SoLuong) và ngày nhập sách (NgayNhap) và thuộc về một thể loại.*
- **BanDoc(MaThe , TenBanDoc, DiaChi, SoDT)**: *Mỗi bạn đọc có một số thẻ để phân biệt (MaThe), họ tên bạn đọc (TenBanDoc), địa chỉ (DiaChi) và số điện thoại (SoDT).*
- **MuonSach(MaThe, MaSach , NgayMuon, NgayTra)**: *Một bạn đọc có thể mượn nhiều cuốn sách, với mỗi cuốn sách người ta ghi nhận ngày mượn (NgayMuon) và ngày trả (NgayTra).*

Ghi chú: *các field có gạch dưới là khoá của lược đồ quan hệ tương ứng.*

Dữ liệu mẫu cho các table như sau:

NhaXuatBan

MANXB	TenNXB
N001	Giáo dục
N002	Khoa học kỹ thuật
N003	Thống kê

BanDoc

MaThe	TenBanDoc	DiaChi	SoDT
050001	Trần Xuân	17 Yersin	858936
050002	Lê Nam	5 Hai Bà Trưng	845623
060001	Nguyễn Năm	10 Lý Tự Trọng	823456
060002	Trần Hùng	20 Trần Phú	841256

Sach

MaSach	TuaDe	MANXB	TacGia	SoLuong	NgayNhap	MaTL
TH0001	Sử dụng Corel Draw	N002	Đậu Quang Tuấn	3	08/09/2005	TH
TH0002	Lập trình mạng	N003	Phạm Vĩnh Hưng	2	03/12/2003	TH
TH0003	Thiết kế mạng chuyên nghiệp	N002	Phạm Vĩnh Hưng	5	04/05/2003	TH
TH0004	Thực hành mạng	N003	Trần Quang	3	06/05/2004	TH
TH0005	3D Studio kỹ xảo hoạt hình T1	N001	Trương Bình	2	05/02/2004	TH
TH0006	3D Studio kỹ xảo hoạt hình T2	N001	Trương Bình	3	05/06/2004	TH
TH0007	Giáo trình Access 2000	N001	Thiện Tâm	5	11/12/2005	TH

MuonSach

MaThe	MaSach	NgayMuon	NgayTra
050001	TH0006	12/12/2006	01/03/2007
050001	TH0007	12/12/2006	
050002	TH0001	08/03/2006	15/04/2007
050002	TH0004	04/03/2007	
050002	TH0002	04/03/2007	04/04/2007
050002	TH0003	02/04/2007	15/04/2007
060002	TH0001	08/04/2007	
060002	TH0007	15/03/2007	15/04/2007

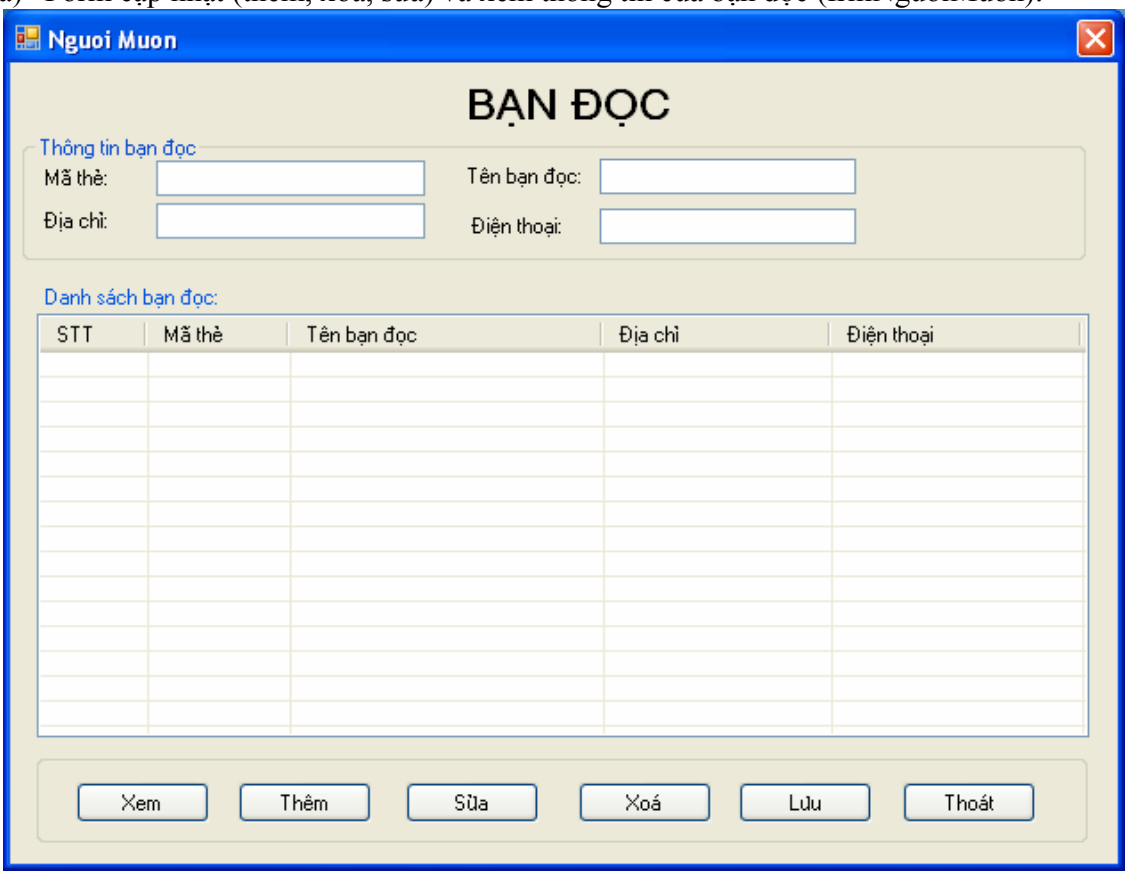
TheLoai

MATL	TENTL
TH	Tin học
HH	Hoá học
KT	Kinh tế
TN	Toán học

1) Tạo các table và thiết lập mối quan hệ (relationship) giữa các table. Căn cứ vào dữ liệu mẫu để chọn kiểu dữ liệu cho phù hợp cho các field trong các bảng.
2) Cài đặt các ràng buộc sau:
   RB1. Số lượng sách >=0
   RB2. Mã thẻ gồm 6 ký tự, được tạo theo quy tắc: hai chữ cuối của năm tạo thẻ ghép với số thứ tự của thẻ trong năm đó. (ví dụ: 050001 trong đó 05 là năm 2005, 1 là số thứ tự của thẻ trong năm 2005) (gợi ý: tạo hàm sinh mã thẻ)
   RB3. Mã sách gồm 6 ký tự, được tạo theo quy tắc: mã thể loại ghép với số thứ tự của cuốn sách trong thể loại đó (gợi ý: tạo hàm sinh mã sách).
   RB4. Mỗi đọc giả không được giữ quá ba quyển sách.
   RB5. Đọc giả không được phép mượn lại cuốn sách mà họ đang nợ.
   RB6. Số lượng trong bảng sách sẽ được thay đổi tuỳ theo thao tác cho bạn đọc mượn, nhận sách trả của bạn đọc hay nhập thêm sách.

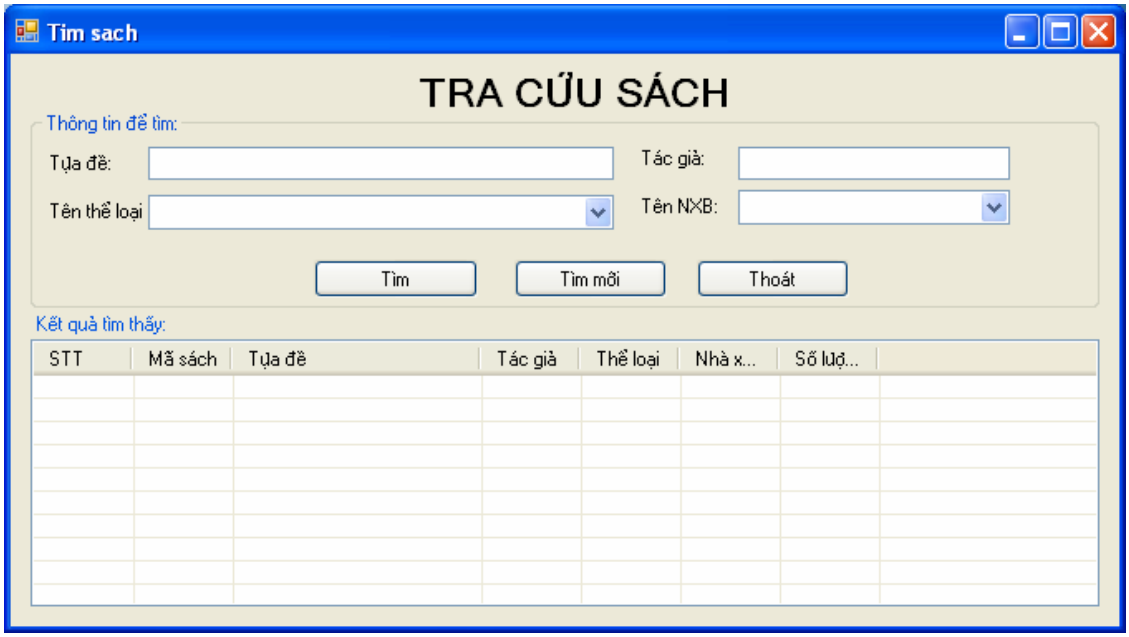
   Các ràng buộc trên khi bị vi phạm sẽ hiện ra thông báo bằng tiếng Việt.
3) Nhập dữ liệu cho các bảng.
4) Viết Stored Procedure CapNhatSach (X, ThaoTac) thực hiện cập nhật số lượng của cuốn sách có mã số X tăng hay giảm 1 đơn vị tuỳ theo thao tác cho nhận trả sách hay cho mượn sách, trong đó SoLuong luôn luôn thoả điều kiện >=0. Nếu ThaoTac=1 là cho mượn sách, ThaoTac = 2 là nhận sách trả. X và ThaoTac là 2 tham số input.
5) Phân tích và xây dựng các thủ tục thường trú, các hàm cần thiết.
6) Thiết kế, phân tích xử lý và lập trình cho các form sau (*lưu ý đảm bảo các ràng buộc toàn vẹn dữ liệu*).

   a) Form cập nhật (thêm, xoá, sửa) và xem thông tin của bạn đọc (frmNguoiMuon):

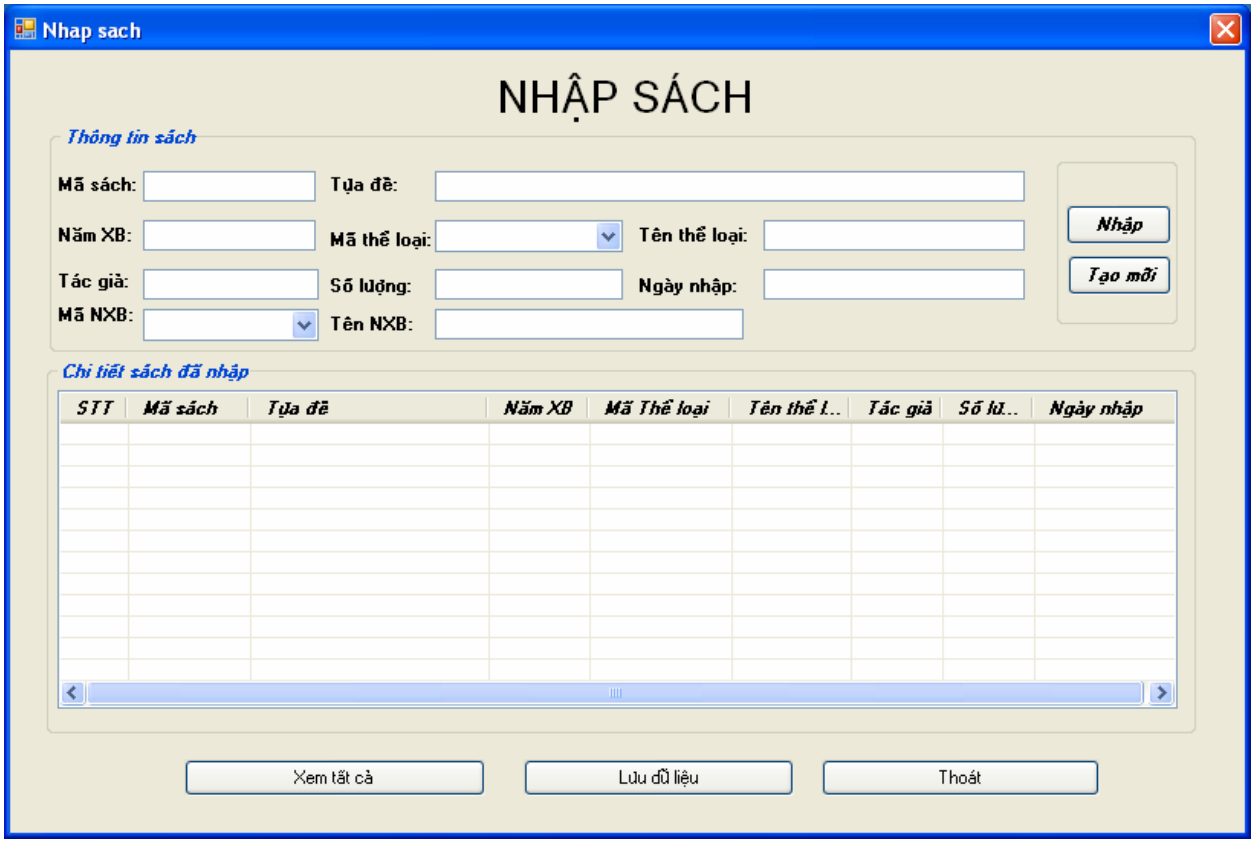


b) Tương tự thiết kế các form cập nhật thông tin thể loại (frmTheLoai) và form cập nhật thông tin nhà xuất bản (frmNhaXuatBan).

c) Form tra cứu sách (frmTraCuu):



d) Form nhập thông tin sách (frmSach):



e) Form xử lý mượn/trả sách:

f) Thiết kế form chính (frmManHinhChinh) với bố trí menu phù hợp để gọi sử dụng các form trên.

7) Tương tự như câu 6, áp dụng thiết kế chương trình theo mô hình 3 lớp.

TÀI LIỆU THAM KHẢO

Tiếng việt:

[1] Phạm Hữu Khang, *Quản trị SQL Server 2000*, NXB Thống kê, 2005.

[2] Dương Quang Thiện, *SQL Server 2000 Lập trình T-SQL*, NXB Văn hoá Sài Gòn, 2007.

[3] Vũ Tuyết Trinh, SQL Server 2008 (Slide bài giảng).

Tiếng Anh:

[4] Bill Hamilton, *ADO.NET Cookbook,* O'Reilly, 2003 (Ebook).

[5] Ramakrishnan, R. and Gehrke, J., *Database Management Systems*, Third Edition, McGraw Hill, 2003.

[6] Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of database systems,* Addison Wesley - 4th edition, 2004.

[7] Sumathi, S. and Esakkirajan, S., *Fundamentals of Relational Database Management Systems,* Springer-Verlag, 2007.