Mục lục

1

Transaction và Lock

I. Giao tác (Transaction)

1. Khái niệm

- **Giao tác là 1 tập hợp các thao tác có thứ tự truy xuất dữ liệu trên CSDL thành 1 đơn vị công việc logic (*xem là 1 thao tác nguyên tố*), chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.**

2. Các tính chất của giao tác: ACID

a. Atomic – Tính nguyên tố

- **1 giao tác là 1 đơn vị xử lý không thể chia nhỏ được nữa; hoặc là tất cả các thao tác trong giao tác được thực hiện (được ghi nhận chắc chắn), hoặc là không có thao tác nào được ghi nhận kết quả.**
- ***Nếu chia nhỏ giao tác thành các thao tác thì sẽ không đảm bảo tính nhất quán của CSDL.***

b. Consistency – Tính nhất quán

- **Giao tác chuyển CSDL từ tình trạng nhất quán này sang tình trạng nhất quán khác.**

c. Isolation – Tính cô lập (hay Independence – Tính độc lập)

- **Các giao tác xử lý đồng thời phải độc lập với những thay đổi được thực hiện bởi các giao tác chưa hoàn tất khác; những thay đổi này chưa hình thành nên 1 trạng thái nhất quán của CSDL.**

d. Durability – Tính lâu dài, bền vững

- **Tất cả những thay đổi lên CSDL mà giao tác thực hiện cho đến khi được xác nhận hoàn tất (commit) thì phải được ghi nhận chắc chắn lên CSDL.**
- ***Có trường hợp sau khi đã commit transaction nhưng những thay đổi chưa thật sự được ghi nhận lên lưu trữ vật lý của CSDL (còn lưu tạm thời trong buffer). Nếu xảy ra sự cố thì sau đó, Hệ quản trị CSDL phải có khả năng phục hồi CSDL, ghi nhận chắc chắn những thay đổi mà giao tác đã thực hiện trên CSDL.***

3. Các lệnh T-SQL đặc trưng của giao tác

BEGIN TRANSACTION	**Bắt đầu giao tác.**
COMMIT TRANSACTION	**Kết thúc giao tác thành công.**
ROLLBACK TRANSACTION	**Kết thúc giao tác không thành công, những thao tác làm ảnh hưởng CSDL đã được thực hiện trước đó được undo, CSDL được trả về tình trạng trước khi thực hiện giao tác.**
SAVE TRANSACTION	**Đánh dấu vị trí để định vị khi cần rollback.**

4. Các vấn đề thường gặp khi xây dựng giao tác

a. Kiểm tra lỗi khi thực hiện giao tác

- **Một số lỗi thường gặp sau khi thực hiện 1 câu lệnh trong giao tác:**
  - **Không có quyền truy cập trên 1 đối tượng (table, stored procedure,…)**
  - **Vi phạm ràng buộc toàn vẹn.**
  - **Update hay Insert 1 dòng dữ liệu đã có trong table.**
  - **Deadlock.**
  - **…**
- **SQL Server trả giá trị lỗi về trong biến toàn cục @@ERROR.**
  @@ERROR= 0: không xảy ra lỗi
  @@ERROR≠ 0: xảy ra lỗi với mã lỗi là @@ERROR
- **Giao tác không thể tự động rollback khi gặp những lỗi phát sinh trong quá trình thực hiện 1 câu lệnh thành phần trong giao tác.**
- **Cần kiểm tra giá trị của biến @@ERROR sau mỗi câu lệnh thành phần trong giao tác và cần xử lý những lỗi (nếu có): yêu cầu giao tác rollback 1 cách tường minh bằng lệnh ROLLBACK TRAN[SACTION]**

Giao tác không kiểm tra lỗi	Giao tác có kiểm tra lỗi
BEGIN TRAN	BEGIN TRAN
EXEC sp_StoredProc	EXEC sp_StoredProc
	IF (@@ERROR <> 0) **ROLLBACK TRAN**
COMMIT TRAN	COMMIT TRAN

b. @@ROWCOUNT và @@ERROR

- **Biến toàn cục @@ROWCOUNT chứa số lượng những dòng dữ liệu được tìm thấy (ví dụ như khi thực hiện lệnh SELECT hay UPDATE). Đây chính là số lượng những dòng dữ liệu thỏa mãn điều kiện trong mệnh đề WHERE.**
- **Nếu không tìm thấy 1 dòng dữ liệu nào thỏa mãn yêu cầu truy vấn (thỏa mãn điều kiện trong mệnh đề WHERE) thì @@ERROR vẫn bằng 0**

(nghĩa là không xảy ra lỗi). Do đó, trong trường hợp này cần kiểm tra giá trị @@ROWCOUNT.

- **Ví dụ: xét giao tác T như sau**

```
BEGIN TRAN
UPDATE TABLE TAB1
SET    VALUE = 100
WHERE  IDX   = 5
COMMIT TRAN
```

Giả sử ta có:

TAB1

IDX	VALUE
1	20
2	50

Rõ ràng là không có dòng nào trên bảng TAB1 thỏa mãn điều kiện *IDX* = 5 nên @@ROWCOUNT = 0 nhưng không có lỗi xảy ra và @@ERROR vẫn bằng 0

c. Transaction lồng nhau

- **Lệnh COMMIT TRANSACTION ở tầng ngoài cùng mới thật sự commit giao tác.**
- **Lệnh ROLLBACK TRANSACTION bất kỳ trong giao tác sẽ làm rollback toàn bộ giao tác.**
- **Giao tác không được lồng nhau quá 32 tầng.**

II. Lock và việc xử lý đồng thời (concurrency)

1. Những vấn đề của việc xử lý đồng thời

a. Đọc dữ liệu chưa commit (Dirty Reads)

b. Thao tác đọc không thể lặp lại (Unrepeatable Reads)

c. Phantom

d. Mất dữ liệu cập nhật (Lost Updates)

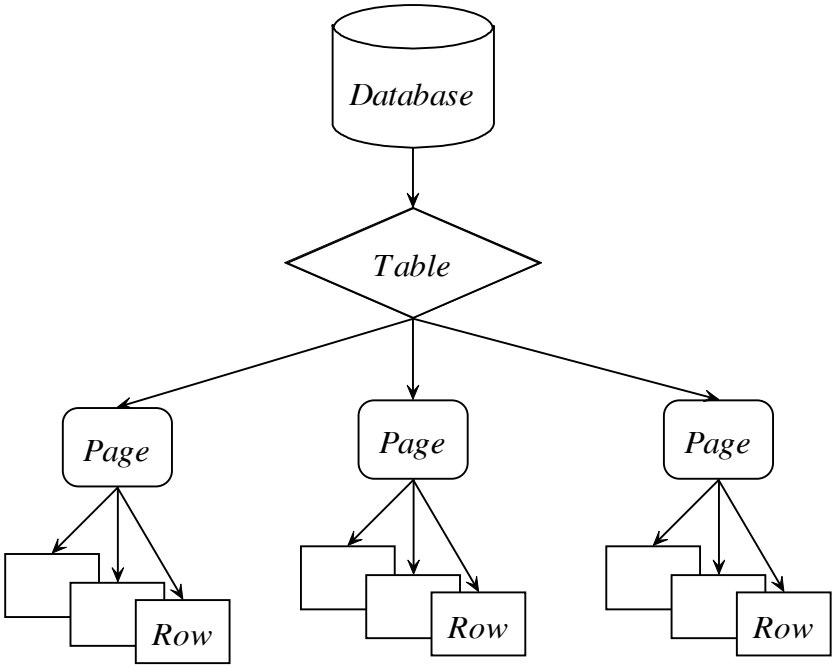
2. Những loại tài nguyên có thể được khóa (lockable resources)

- **Trong SQL Server, có 6 loại tài nguyên có thể được khóa, bao gồm:**

Tài nguyên	Ý nghĩa
Database	- **Khóa trên toàn bộ cơ sở dữ liệu** - **Chỉ nên áp dụng khi tiến hành thay đổi trên lược đồ của CSDL.**
Table	- **Khóa trên 1 bảng trong cơ sở dữ liệu. Toàn bộ các đối tượng trong bảng này, bao gồm tất cả các dòng và tất cả các khóa trong các chỉ mục trong bảng, đều bị khóa.**
Extent	- **Khóa trên 1 extend (= 8 trang).**
Page	- **Khóa trên 1 trang. Tất cả dữ liệu và các khóa chỉ mục trong**

	trang này đều bị khóa.
Key	**- Khóa trên 1 hay 1 số khóa (key) trong 1 chỉ mục.**
Row	**- Được đưa vào SQL Server từ version 7.0.** **- Khóa trên 1 dòng dữ liệu trong 1 bảng.**

- **Khi khóa trên 1 đơn vị dữ liệu ở cấp cao hơn thì các đơn vị dữ liệu con bên trong cũng bị khóa.**
- **Khi 1 đơn vị dữ liệu con bị khóa thì các đơn vị dữ liệu ở các cấp cao hơn sẽ bị khóa bằng khóa Intent tương ứng.**

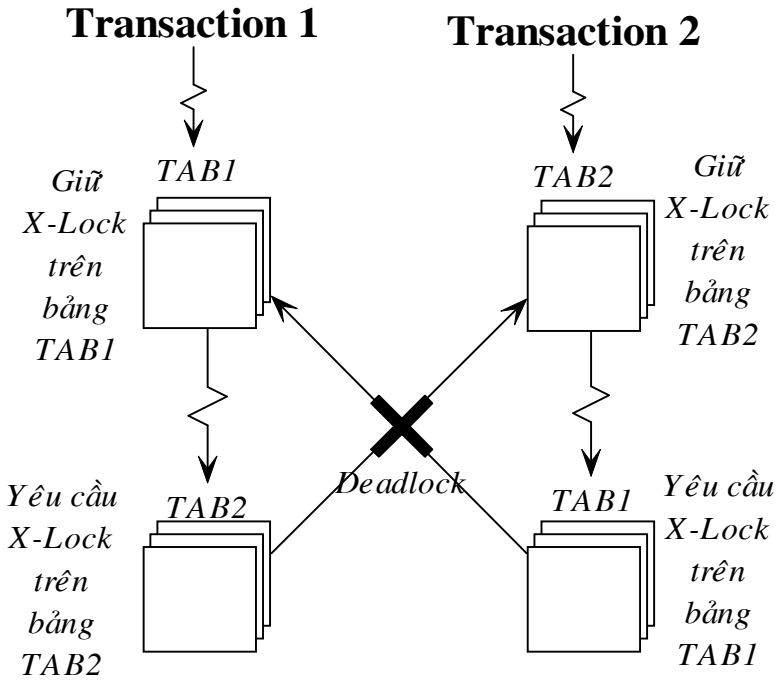


3. Hiện tượng Deadlock

- **Deadlock là tình trạng trong đó những giao tác có liên quan không thể thực hiện tiếp các thao tác của nó mà phải chờ nhau mãi.**
- **Hiện tượng deadlock thường xảy ra trong 2 tình huống sau:**

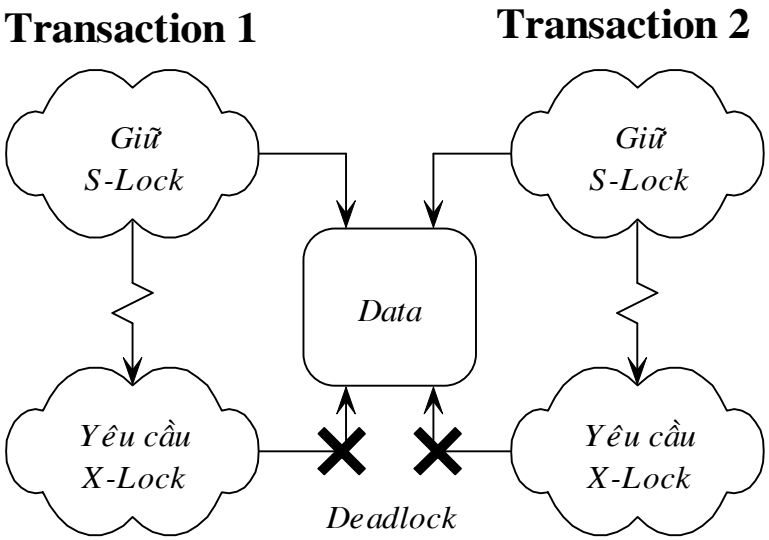
a. Cycle Deadlock

- **Ví dụ:**
  - **Giao tác T1 giữ X-Lock trên bảng TAB1**
  - **Giao tác T2 giữ X-Lock trên bảng TAB2**
  - **Giao tác T1 yêu cầu X-Lock trên bảng TAB2 => T1 chờ T2**
  - **Giao tác T2 yêu cầu X-Lock trên bảng TAB1 => T2 chờ T1**



b. Conversion Deadlock

- **Ví dụ:**
  - **Giao tác T1 và T2 cùng giữ S-Lock trên 1 tài nguyên R**
  - **Giao tác T1 yêu cầu X-Lock trên R => T1 chờ T2**
  - **Giao tác T2 yêu cầu X-Lock trên R => T2 chờ T1**



4. Các phương thức khóa (lock modes)

a. *Shared Locks (S)*

- **Shared Lock ⇔ Read Lock**
- **Khi đọc 1 đơn vị dữ liệu, SQL Server tự động thiết lập Shared Lock trên đơn vị dữ liệu đó (trừ trường hợp sử dụng No Lock)**
- **Shared Lock có thể được thiết lập trên 1 bảng, 1 trang, 1 khóa hay trên 1 dòng dữ liệu.**
- **Nhiều giao tác có thể đồng thời giữ Shared Lock trên cùng 1 đơn vị dữ liệu.**
- **Không thể thiết lập Exclusive Lock trên đơn vị dữ liệu đang có Shared Lock.**
- **Shared Lock thường được giải phóng ngay sau khi sử dụng xong dữ liệu được đọc, trừ khi có yêu cầu giữ shared lock cho đến hết giao tác.**

b. *Exclusive Locks (X)*

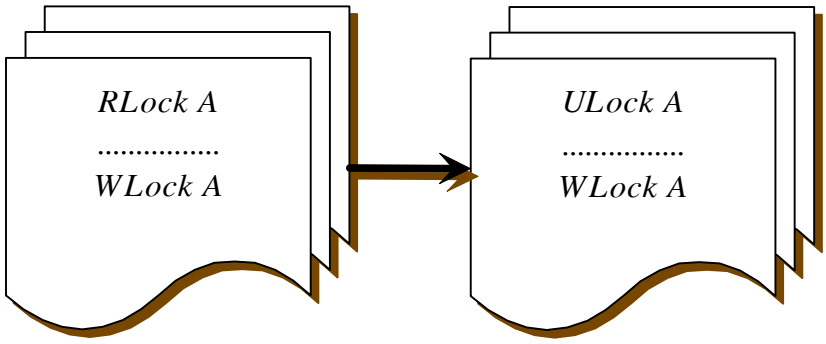
- **Exclusive Lock ⇔ Write Lock**
- **Khi thực hiện thao tác ghi (insert, update, delete) trên 1 đơn vị dữ liệu, SQL Server tự động thiết lập Exclusive Lock trên đơn vị dữ liệu đó.**
- **Exclusive Lock luôn được giữ đến hết giao tác.**
- **Tại 1 thời điểm, chỉ có tối đa 1 giao tác được quyền giữ Exclusive Lock trên 1 đơn vị dữ liệu.**

c. *Update Locks (U)*

- **Update Lock = Intent-to-update Lock**
- **Update Lock sử dụng khi đọc dữ liệu với dự định ghi trở lại trên đơn vị dữ liệu này.**
- **Update Lock là chế độ khóa trung gian giữa Shared Lock và Exclusive Lock.**

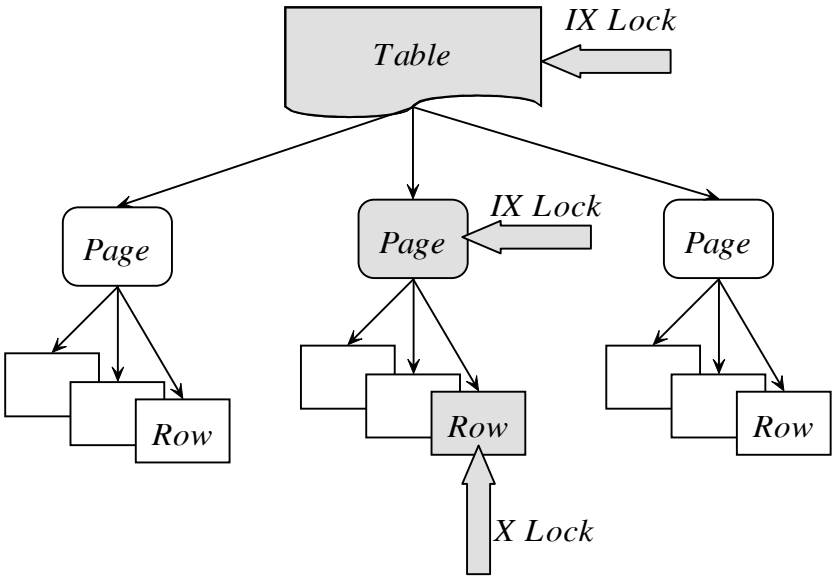
Shared Lock	Update Lock
Tương thích với Shared Lock	Tương thích với Shared Lock
Sử dụng trong việc đọc dữ liệu	Sử dụng trong việc đọc dữ liệu
Tại 1 thời điểm có thể có nhiều Shared Lock trên cùng1 đơn vị dữ liệu	Tại 1 thời điểm, có tối đa 1 Update Lock trên 1 đơn vị dữ liệu

- **Update Lock không ngăn cản việc thiết lập các Shared Lock khác trên cùng 1 đơn vị dữ liệu => Update Lock tương thích với Shared Lock**
- **Khi thực hiện thao tác ghi lên 1 đơn vị dữ liệu thì bắt buộc Update Lock phải tự động chuyển đổi thành Exclusive Lock**
- **Update Lock giúp tránh hiện tượng deadlock khi có yêu cầu chuyển từ Shared Lock lên Exclusive Lock trên 1 đơn vị dữ liệu nào đó (Do tại 1 thời điểm chỉ có tối đa 1 Update Lock trên 1 đơn vị dữ liệu)**



d. Intent Locks

- **Intent Lock không phải là 1 chế độ khóa riêng biệt mà được sử dụng kết hợp với các chế độ khóa khác:**
  - **Shared Lock => Intent Shared Lock (IS)**
  - **Update Lock => Intent Update Lock (IU)**
  - **Exclusive Lock => Intent Exclusive Lock (IX)**
- **Intent Lock chỉ áp dụng trên table and và page**
- **Intent Lock được SQL Server tự động thiết lập, không thể được yêu cầu thiết lập Intent Lock một cách tường minh.**
- **SQL Server cho phép lock trên các đơn vị dữ liệu ở cấp độ khác nhau (table, page, row,...) => Cần 1 cơ chế cho việc kiểm tra 1 đơn vị dữ liệu thành phần của 1 đơn vị dữ liệu có đang bị khóa hay không**
- **Khi 1 đơn vị dữ liệu thành phần bị khóa, các đơn vị dữ liệu ở cấp cao hơn cũng sẽ bị khóa bằng Intent Lock tương ứng**
- **Ví dụ: 1 giao tác giữ Exclusive Lock trên 1 dòng dữ liệu thì cũng giữ Intent-Exclusive Lock trên trang chứa dòng này và bảng dữ liệu tương ứng**



e. Các loại khóa đặc biệt

- **Schema Stability Locks (Sch-S)**
  - **Dùng cho table**
  - **Cho biết đang có 1 lệnh truy vấn có sử dụng đến bảng này đang được compile, không cho phép thay đổi cấu trúc bảng.**
  - **Tương thích với tất cả các loại lock khác (ngoại trừ Sch-M)**
- **Schema Modification Locks (Sch-M)**
  - **Dùng cho table**
  - **Cho biết cấu trúc 1 bảng đang được thay đổi**
- **Bulk Update Locks (BU)**
  - **Dùng khi thực hiện thao tác chép dữ liệu hàng loạt vào 1 bảng**

f. Sự tương thích giữa các phương thức khóa

	IS	S	U	IX	SIX	X	Sch-S	Sch-M	BU
IS	Yes	Yes	Yes	Yes	Yes		Yes		
S	Yes	Yes	Yes				Yes		
U	Yes	Yes					Yes		
IX	Yes						Yes		
SIX	Yes						Yes		
X							Yes		
Sch-S	Yes	Yes	Yes	Yes	Yes	Yes	Yes		Yes
Sch-M									
BU							Yes		Yes

- **Khi 1 yêu cầu lock của giao tác T1 chưa thể được đáp ứng, giao tác T1 phải chờ.**
- **Yêu cầu lock trên 1 đơn vị dữ liệu được đáp ứng theo nguyên tắc FIFO, giao tác nào yêu cầu lock trước sẽ được đáp ứng trước => Giải quyết được hiện tượng livelock (hay lock-starvation)**

5. Chiến lược sử dụng các phương thức khóa

SERIALIZABLE/ HOLDLOCK	➢ **Sau khi được thiết lập bằng 1 lệnh trong giao tác, khóa sẽ tồn tại cho đến khi chấm dứt giao tác.** ➢ **Không thể thêm dữ liệu mới vào nếu dữ liệu này thỏa điều kiện trong mệnh đề Where của lệnh tạo lập khóa này.** ➢ **Tương tự như sử dụng Isolation Level là Serializable**
READUNCOMMITTED/ NOLOCK	➢ **Thao tác thực hiện sẽ rất nhanh nhưng có thể gặp**

	phải các vấn đề trong xử lý đồng thời
READCOMMITTED	➢ **Đây là chế độ mặc định.** ➢ **Chỉ đọc những dữ liệu đã được commit** ➢ **Thiết lập shared lock trên đơn vị dữ liệu cần đọc**
REPEATABLEREAD	➢ **Tương tự SERIALIZABLE/HOLDLOCK nhưng vẫn có thể thêm dữ liệu mới vào CSDL.**
READPAST	➢ **Chỉ có thể sử dụng trong lệnh Select và chỉ áp dụng trên khóa của dòng dữ liệu (row-lock). Những dòng bị khóa sẽ được bỏ qua.**
ROWLOCK	
TABLOCK	➢ **Khóa 1 bảng trong CSDL.** ➢ **Các thao tác cập nhật của những giao tác khác không thể thực hiện trên bảng này.**
TABLOCKX	➢ **Tương tự TABLOCK nhưng mọi thao tác trên bảng này của các giao tác khác đều không thực hiện được.**
UPDLOCK	➢ **Sử dụng Updatelock thay vì Shared lock.**

6. Các mức độ cô lập (Isolation Levels)

- **Mức độ cô lập của 1 giao tác quy định mức độ nhạy cảm của 1 giao tác đối với những sự thay đổi trên CSDL do các giao tác khác tạo ra**
  => Mức độ cô lập của giao tác quy định thời gian giữ lock: lock có cần được giữ cho đến hết giao tác để ngăn những sự thay đổi trên CSDL do các giao tác khác tạo ra hay không
- **Các mức độ cô lập được SQL Server 7.0 hỗ trợ:**
  - **Read Uncommitted**
  - **Read Committed (default)**
  - **Repeatable Read**
  - **Serializable**
- **Mỗi transaction đều có 1 trong 4 mức độ cô lập nêu trên.**
- **Lệnh T-SQL thiết lập mức độ cô lập:**
  `SET TRANSACTION ISOLATION LEVEL <READ COMMITTED|READ UNCOMMITTED|REPEATABLE READ|SERIALIZABLE>`
- **Việc thay đổi mức độ cô lập chỉ có tác dụng trong giao tác đang xét và các giao tác tiếp theo trong cùng 1 connection**
  => Việc thay đổi mức độ cô lập của 1 giao tác không ảnh hưởng đến các giao tác thuộc các connection khác

a. Read Uncommitted

- **Đặc điểm:**
  - **Không thiết lập Shared Lock trên những đơn vị dữ liệu cần đọc**
  - **Không bị ảnh hưởng bởi những lock của các giao tác khác trên những đơn vị dữ liệu cần đọc**

- **Không phải chờ khi đọc dữ liệu (kể cả khi dữ liệu đang bị lock bởi giao tác khác)**
- **Các vấn đề gặp phải khi xử lý đồng thời:**
  - **Dirty Reads**
  - **Unrepeatable Reads**
  - **Phantoms**
  - **Lost Updates**

➢ **Ưu điểm:**
- **Tốc độ xử lý rất nhanh**
- **Không cản trở những giao tác khác thực hiện việc cập nhật dữ liệu**

➢ **Khuyết điểm:**
- **Có khả năng xảy ra mọi vấn đề trong việc xử lý đồng thời, đặc biệt là vấn đề Dirty Reads**

➢ **Nhận xét:**
- **Chỉ nên dùng để đọc dữ liệu trong trường hợp cần 1 cái nhìn tổng quan về CSDL, ví dụ như tạo những báo cáo về tình hình chung.**
- **Không dùng trong trường hợp cần đọc những số liệu chính xác hay tiến hành cập nhật trên cơ sở dữ liệu.**

b. Read Committed

➢ **Đặc điểm:**
- **Đây là mức độ cô lập mặc định của SQL Server**
- **Giải quyết vấn đề Dirty Reads**
- **Tạo Shared Lock trên đơn vị dữ liệu được đọc, Shared Lock được giải phóng ngay sau khi đọc xong dữ liệu**
- **Tạo Exclusive Lock trên đơn vị dữ liệu được ghi, Exclusive Lock được giữ cho đến hết giao tác**

➢ **Ưu điểm:**
- **Giải quyết vấn đề Dirty Reads**
- **Shared Lock được giải phóng ngay, không cần phải giữ cho đến hết giao tác nên không ngăn cản thao tác cập nhật của các giao tác khác.**

➢ **Khuyết điểm:**
- **Chưa giải quyết được vấn đề Unrepeatable Reads, Phantoms, Lost Updates**
- **Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.**

c. Repeatable Read

➢ **Đặc điểm:**

- **Repeatable Read = Read Committed+Giải quyết Unrepeatable Reads**
- **Giải quyết vấn đề Dirty Reads và Unrepeatable Reads**
- **Tạo Shared Lock trên đơn vị dữ liệu được đọc, Shared Lock được giữ cho đến hết giao tác => Không cho phép các giao tác khác cập nhật, thay đổi giá trị trên đơn vị dữ liệu này.**
- **Tạo Exclusive Lock trên đơn vị dữ liệu được ghi, Exclusive Lock được giữ cho đến hết giao tác.**

➢ **Ưu điểm:**

- **Giải quyết vấn đề Dirty Reads và Unrepeatable Reads**

➢ **Khuyết điểm:**

- **Chưa giải quyết được vấn đề Phantoms và Lost Updates**
- **Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.**
- **Các giao tác khác không được phép cập nhật trên những đơn vị dữ liệu đang bị giữ Shared Lock.**
- **Vẫn cho phép Insert những dòng dữ liệu thỏa mãn điều kiện thiết lập những Shared Lock => Phantoms**

d. Serializable

➢ **Đặc điểm:**

- **Serializable = Repeatable Read + Giải quyết Phantoms**
- **Giải quyết vấn đề Dirty Reads, Unrepeatable Reads và Phantoms**
- **Tạo Shared Lock trên đơn vị dữ liệu được đọc, Shared Lock được giữ cho đến hết giao tác => Không cho phép các giao tác khác cập nhật, thay đổi giá trị trên đơn vị dữ liệu này.**
- **Không cho phép Insert những dòng dữ liệu thỏa mãn điều kiện thiết lập những Shared Lock (sử dụng khóa trên 1 miền giá trị-Key Range Lock)**
- **Tạo Exclusive Lock trên đơn vị dữ liệu được ghi, Exclusive Lock được giữ cho đến hết giao tác.**

➢ **Ưu điểm:**

- **Giải quyết vấn đề Dirty Reads, Unrepeatable Reads và Phantoms**

➢ **Khuyết điểm:**

- **Chưa giải quyết được vấn đề Lost Updates**
- **Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.**

III. Ví dụ

1. Giả lập nhiều giao tác đồng thời trên SQL Server:

- **Vào SQL Query Analyzer, tạo 2 connection trên 2 cửa sổ riêng biệt trong Query Analyzer, mỗi connection ứng với 1 giao tác.**
- **Trong mỗi giao tác, sử dụng lệnh** `WAITFOR DELAY` **để yêu cầu 1 giao tác tạm dừng xử lý. Cú pháp:**

```
WAITFOR DELAY 'hh:mm:ss'
```

2. Ví dụ 1

- **Mục đích: So sánh mức độ cô lập Read UnCommitted và Read Committed**
- **Giả sử ban đầu bảng SINHVIEN không có sinh viên nào tên là 'Minh'**

a. Trường hợp 1

T1	T2
BEGIN TRAN UPDATE *SINHVIEN* SET *TEN* = 'Minh' WAITFOR DELAY *'00:00:20'* ***ROLLBACK TRAN***	 ***BEGIN TRAN*** SET TRANSACTION ISOLATION LEVEL *READ UNCOMMITTED* SELECT * FROM *SINHVIEN* WHERE *TEN* = 'Minh' ***COMMIT TRAN***

- **T2 sẽ đưa ra tất cả các dòng của bảng SINHVIEN**

b. Trường hợp 2

T1	T2
BEGIN TRAN UPDATE *SINHVIEN* SET *TEN* = 'Minh' WAITFOR DELAY *'00:00:20'* ***ROLLBACK TRAN***	 ***BEGIN TRAN*** SET TRANSACTION ISOLATION LEVEL *READ COMMITTED* SELECT * FROM *SINHVIEN* WHERE *TEN* = 'Minh' ***COMMIT TRAN***

- **T2 sẽ không đưa ra bất kỳ dòng nào của bảng SINHVIEN**

3. Ví dụ 2

- **Mục đích: So sánh mức độ cô lập Read Committed và Repeatable Read**

a. Trường hợp 1

T1	T2
BEGIN TRAN SET TRANSACTION ISOLATION LEVEL *READ COMMITTED* SELECT *TEN* FROM *SINHVIEN* WAITFOR DELAY *'00:00:20'* SELECT *TEN* FROM *SINHVIEN* ***COMMIT TRAN***	***BEGIN TRAN*** UPDATE *SINHVIEN* SET *TEN*= 'Minh' ***COMMIT TRAN***

- **Kết quả của 2 câu lệnh SELECT của T1 là khác nhau. T2 không cần chờ T1 thực hiện xong mới thực hiện được lệnh Update**

b. Trường hợp 2

T1	T2
BEGIN TRAN SET TRANSACTION ISOLATION LEVEL *REPEATABLE READ* SELECT *TEN* FROM *SINHVIEN* WAITFOR DELAY *'00:00:20'* SELECT *TEN* FROM *SINHVIEN* ***COMMIT TRAN***	***BEGIN TRAN*** UPDATE *SINHVIEN* SET *TEN*= ' Minh' ***COMMIT TRAN***

- **Kết quả của 2 câu lệnh SELECT của T1 là như nhau. Trên thực tế, T2 phải chờ T1 thực hiện xong mới thực hiện được lệnh Update**

4. Ví dụ 3

- **Mục đích: So sánh mức độ cô lập Repeatable Read và Serializable**

a. Trường hợp 1

T1	T2

BEGIN TRAN SET TRANSACTION ISOLATION LEVEL *REPEATABLE READ* SELECT *TEN* FROM *SINHVIEN* WAITFOR DELAY *'00:00:20'* SELECT *TEN* FROM *SINHVIEN* ***COMMIT TRAN***	***BEGIN TRAN*** INSERT *SINHVIEN* VALUES ('**Tuyết**',…) COMMIT TRAN

- **Kết quả của 2 câu lệnh SELECT của T1 là khác nhau. T2 không phải chờ T1 thực hiện xong mới thực hiện được lệnh Insert**

b. Trường hợp 2

T1	T2
BEGIN TRAN SET TRANSACTION ISOLATION LEVEL *SERIALIZABLE* SELECT *TEN* FROM *SINHVIEN* WAITFOR DELAY *'00:00:20'* SELECT *TEN* FROM *SINHVIEN* ***COMMIT TRAN***	***BEGIN TRAN*** INSERT *SINHVIEN* VALUES ('**Tuyết**',…) ***COMMIT TRAN***

- **Kết quả của 2 câu lệnh SELECT của T1 là như nhau. Trên thực tế, T2 phải chờ T1 thực hiện xong mới thực hiện được lệnh Insert**

IV. Các bước xây dựng Transaction

- **Bước 1: Xây dựng các câu lệnh chính yếu của transaction.**
  - **Mục tiêu:**
    - **Xây dựng chính xác giải thuật của giao tác**
    - **Tạm thời không đề cập đến vấn đề:**
      - **Vấn đề kiểm tra lỗi (kiểm tra giá trị biến** @@ERROR)
      - **Cạnh tranh truy xuất**
      - **Vấn đề Lock**
      - **Vấn đề Isolation Level**
  - **Ví dụ:**

```
DECLARE @SISOHT    INT,
        @SISOMAX   INT
SELECT  @SISOHT = SISOHT
FROM    LOP
```

```
WHERE   MALOP   = @MALOP
UPDATE  TABLE LOP
SET     SISOHT  = SISOHT+1
WHERE   MALOP   = @MALOP
```

- **Bước 2: Bổ sung các lệnh kiểm tra lỗi**
  - **Mục tiêu:**
    - **Kiểm tra các lỗi phát sinh trong quá trình thực hiện giao tác.**
  - **Sau mỗi câu lệnh Transact-SQL, bổ sung dòng lệnh kiểm tra lỗi và yêu cầu** `ROLLBACK TRANSACTION` **nếu có lỗi**
  - **Ví dụ:**

```
DECLARE @SISOHT    INT,
        @SISOMAX   INT
SELECT  @SISOHT = SISOHT
FROM    LOP
WHERE   MALOP   = @MALOP
→ IF (@@ERROR<>0)
→    ROLLBACK TRANSACTION
UPDATE  TABLE LOP
SET     SISOHT  = SISOHT+1
WHERE   MALOP   = @MALOP
→ IF (@@ERROR<>0)
→    ROLLBACK TRANSACTION
```

- **Bước 3: Xác định chiến lược sử dụng Lock và Isolation Level**
  - **Mục tiêu:**
    - **Bảo đảm giao tác đang được xây dựng không gặp phải các vấn đề trong truy xuất đồng thời (concurrency)**
    - **Chưa cần quan tâm đến hiệu quả của hệ thống**
  - **Một số gợi ý:**
    - **Nếu không cần đọc chính xác dữ liệu mà chỉ cần 1 cái nhìn tổng quan về thông tin trong CSDL thì nên sử dụng** `READ UNCOMMITTED` **hay** `NO LOCK`
    - **Thông thường, sử dụng chế độ mặc định của SQL Server là** `READ COMMITTED`
    - **Nếu không muốn nội dung của 1 đơn vị dữ liệu bị thay đổi trong suốt quá trình diễn ra giao tác thì sử dụng** `REPEATABLE READ`
    - **Nếu không muốn xuất hiện những dòng dữ liệu phantoms thì sử dụng** `SERIALIZABLE`
    - **Nếu dự kiến khóa trên rất nhiều dòng dữ liệu trong 1 bảng thì nên sử dụng** `TABLOCK` **hay** `TABLOCKX`
    - **Nếu dự kiến cập nhật 1 đơn vị dữ liệu sau khi đọc nội dung đơn vị dữ liệu này thì nên sử dụng** `UPDLOCK`
  - **Ví dụ:**

```
DECLARE @SISOHT    INT,
        @SISOMAX   INT
→ SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

```
   SELECT  @SISOHT = SISOHT
→  FROM    LOP WITH (UPDLOCK)
   WHERE   MALOP   = @MALOP
   IF (@@ERROR<>0)
      ROLLBACK TRANSACTION
   UPDATE  TABLE LOP
   SET     SISOHT  = SISOHT+1
   WHERE   MALOP   = @MALOP
   IF (@@ERROR<>0)
      ROLLBACK TRANSACTION
```

- **Bước 4: Tối ưu hóa giao tác**
  - **Mục tiêu:**
    - **Tối ưu hóa giao tác (trong vấn đề sử dụng lock và isolation level)**
    - **Vẫn đảm bảo giao tác không gặp phải các vấn đề của truy xuất đồng thời**
  - **Một số gợi ý:**
    - **Nên sử dụng mức độ cô lập mặc định** (READ COMMITTED)
    - **Chỉ giữ shared lock trong trường hợp thật sự cần thiết**
    - **Thiết lập các exclusive lock ở gần cuối giao tác**
  - **Ví dụ:**

```
DECLARE @SISOHT    INT,
        @SISOMAX   INT
→ SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT  @SISOHT = SISOHT
FROM    LOP WITH (UPDLOCK, HOLDLOCK)
WHERE   MALOP   = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRANSACTION
UPDATE  TABLE LOP
SET     SISOHT  = SISOHT+1
WHERE   MALOP   = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRANSACTION
```

2

Liên kết giữa các ngôn ngữ lập trình với SQL Server – Lập trình với giao tác

Việc lập trình giao tác có thể được thực hiện theo 2 cách

- **Sử dụng các công cụ được cung cấp bởi môi trường lập trình để xây dựng và quản lý các giao tác. Ví dụ như sử dụng hàm BeginTrans, CommitTrans, hay RollbackTrans.**
- **Xây dựng sẵn các stored procedure trên SQL Server và gọi thực hiện các stored procedure này từ các ứng dụng được xây dựng bằng các môi trường lập trình khác.**

I. Stored Procedure

- **Stored procedure đã được dịch sẵn (precompiled) và kế hoạch thực hiện đã được tối ưu hóa được lưu trong vùng đệm dành cho procedure.**
- **Sử dụng stored procedure giúp chia sẻ logic chương trình (sharing application logic).**
- **Việc thực hiện giao tác thông qua việc gọi stored procedure sẽ dễ dàng hơn**
- **Giao tác có thể được lưu trữ sẵn dưới dạng các stored procedure để sử dụng trong các ứng dụng.**

II. Visual Basic

1. Sơ lược về các thành phần của ADO :

- **Đối tượng Connection :**

  Dùng để thực hiện các công việc : cấu hình một kết nối, thiết lập và hủy bỏ các phiên làm việc (session) với nguồn dữ liệu, thi hành một query.

- **Đối tượng Command :**

  Dùng để thực hiện các công việc : thực hiện một lệnh trên cơ sở dữ liệu. Ngoài ra còn có thể mở mới một kết nối, nếu nó tham chiếu tới một đối tượng Connection thì nó thi hành lệnh trên kết nối này, nếu nó được xác định với một chuỗi mô tả kết nối thì nó sẽ mở một kết nối mới.

- **Đối tượng Recordset :**

  Dùng để thực hiện các công việc : cho phép thêm, xóa, cập nhật và cuộn qua các record trong recordset.

- **Bộ Errors và đối tượng Error :**

  Bộ Errors chứa các lỗi do trình cung cấp chỉ định, gây ra bởi một thao tác đơn. Mỗi đối tượng Error chỉ định một lỗi trong bộ Errors. Ta có thể dùng vòng lặp để duyệt qua toàn bộ lỗi có trong bộ Errors.

- **Đối tượng Parameter :**

  Bộ Parameters và đối tượng Parameter cung cấp thông tin và dữ liệu cho đối tượng Command. Chúng được sử dụng khi câu truy vấn trong đối tượng Command cần tham số.

2. Thực hiện Transaction trên Visual Basic :

Việc thực hiện transaction trên Visual Basic có thể được làm bằng cách tạo một store procedure chứa transaction sẵn trên SQL Server và Visual Basic chỉ việc truyền tham số và gọi store procedure này. Hoặc cũng có thể tạo transaction bằng cách thực hiện từng câu lệnh SQL trên Visual Basic.

a. Thực hiện bằng cách gọi stored procedure :

- **Việc thực hiện store procedure trên Visual Basic được thực hiện bằng đối tượng Command của ADO.**
- **Ví dụ :**

  'Giả sử đã mở một connection (cn) và store procedure tên 'sp_DANGKY_HOCPHAN có tham số vào là MASV và MALOP

```
Dim cmd As New ADODB.Command
Dim param As Parameter

' Set up a command object for the stored procedure.
Set cmd.ActiveConnection = cn
cmd.CommandText = "sp_DANGKY_HOCPHAN"
cmd.CommandType = adCmdStoredProc

' Set up a return parameter.
Set param = cmd.CreateParameter("Return", adInteger,
                                adParamReturnValue)
cmd.Parameters.Append param

' Set up an input parameter.
Set param = cmd.CreateParameter("MASV", adChar,
                                adParamInput, 10)
cmd.Parameters.Append param
param.Value = cboSV.Text
Set param = cmd.CreateParameter("MALOP", adChar,
                                adParamInput, 10)
cmd.Parameters.Append param
param.Value = cboLop.Text
```

```
' Set up an output parameter.
Set param = cmd.CreateParameter("Output", adInteger,
                                adParamOutput)
cmd.Parameters.Append param

' Execute command.
cmd.Execute
```

b. Thực hiện bằng các lệnh từ Visual Basic

- **Một Transaction sẽ là tập hợp các câu lệnh SQL. Do đó việc thực hiện một transaction chính là thực hiện từng câu lệnh SQL.**
- **Ví dụ thực hiện một lệnh SQL trên Visual Basic :**

```
'Giả sử đã mở một connection (cn)
Dim rs As New ADODB.Recordset
Dim cmd As String
cmd =    'Lệnh SQL
Set rs = cn.Execute(cmd)
```

III. Visual C++ và ADO

- **Cần phải có thư viện MSADO15.DLL. Để import thư viện vào thực hiện câu lệnh sau :**

```
#import "MSADO15.DLL" rename_namespace("ADOCG")
rename("EOF", "EndOfFile")
using namespace ADOCG;
#include "icrsint.h"
```

- **Visual C++ hỗ trợ kết nối cơ sở dữ liệu bằng ADO. Các thành phần ADO trên Visual C++ cũng giống như trên Visual Basic. Nhưng việc thực hiện có một vài điểm cần lưu ý do môi trường lập trình CSDL trên Visual C++ phức tạp hơn trên Visual Basic.**

1. Mở một kết nối

- **Trước khi khởi tạo một kết nối CSDL bằng ADO tên Visual C++ thì ta phải khởi tạo mội trường COM bằng lệnh :** `::CoInitialize(NULL)`**. Sau đó :**

```
_ConnectionPtr pCon;
pCon.CreateInstance(__uuidof(Connection));
pCon->Open("DSN=Biblio ;UID=adimin; PWD=;");
Hay :
pCon->Open("driver=;Database=;UID=sa;PWD=;" );
```

2. Thực hiện truy vấn

a. Thực hiện bằng đối tượng command :

```
_CommandPtr pCmd;
```

```
pCmd.CreateInstance(__uuidof(Command));
pCmd->ActiveConnection = pConn;
pCmd->CommandText = "Select * from Addresses";
_RecordsetPtr pRs;
pRs = pCmd->Execute();
```

b. Thực hiện bằng đối tượng connection :

```
_RecordsetPtr pRs();
pRs.CreateInstance(__uuidof(Recordset));
pRs->Open
("Select FirstName, LastName, Age from Employees", pCon,
adOpenForwardOnly, adLockReadOnly, adCmdUnknown);
```

3. Kết buộc dữ liệu

- **Vì dữ liệu khi lấy từ cơ sở dữ liệu đưa vào Visual C++ sẽ mang kiểu chung là variant. Do đó để có thử hiểu được các dữ liệu đó ta phải thực hiện một quá trình trung gian để kết buộc dữ liệu thành dữ liệu cần thiết.**

a. Thực hiện bằng cách chuyển đổi từ biến variant :

```
_variant_t    vFirstName
FieldPtr   pfldFirstName;

pfldFirstName = pRs->Fields->GetItem(0);
vFirstName = pfldFirstName->Value;
WideCharToMultiByte(CP_ACP,  0,  vFirstName.bstrVal,  -1,
m_szFirstName, sizeof(m_szFirstName), NULL, NULL);
```

b. Dùng cấu trúc để kết buộc dữ liệu :

- **Ta có thể dùng một cấu trúc được khai báo trước để lấy trực tiếp dữ liệu, khỏi qua quá trình chuyển đổi mà quá trình này Visual C sẽ tự làm. Điểm bất lợi của phương pháp này là cấu trúc dữ liệu từ recordset phải được biết trước. Nhưng ưu điểm của phương pháp này là rất dễ sử dụng.**
- **Ví dụ :**
  - **Đầu tiên phải khai báo một lớp thừa kế từ CADORecordBinding.**

```
class CCustomRs : public CADORecordBinding {
BEGIN_ADO_BINDING(CCustomRs)
   ADO_FIXED_LENGTH_ENTRY(1, adInteger, m_lAddressID,
   lAddressIDStatus, ^ FALSE)
   ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar,
   m_szFirstName,
   ^ sizeof(m_szFirstName), lFirstNameStatus, TRUE)
   ADO_FIXED_LENGTH_ENTRY(3, adDate, m_dtBirthdate,
   lBirthdateStatus, ^ TRUE)
```

```
    ADO_FIXED_LENGTH_ENTRY(4, adBoolean, m_bSendCard,
    lSendCardStatus, ^TRUE)
  END_ADO_BINDING()
  public:
   LONG m_lAddressID;
   ULONG lAddressIDStatus;
   CHAR m_szFirstName[51];
   ULONG lFirstNameStatus;
   DATE m_dtBirthdate;
   ULONG lBirthdateStatus;
   VARIANT_BOOL m_bSendCard;
   ULONG lSendCardStatus;
 };
```

- Sau đó trong thân chương trình khi muốn lấy một recordset ta làm như sau :
  - Khai báo recordset :

```
CCustomRs m_rsRecSet;
```

  - Tạo một con trỏ IADORecordBinding :

```
IADORecordBinding *picRs = NULL;
```

  - Khi lấy recordset từ _Recordset pRs ta thêm các lệnh :

```
if (FAILED(pRs->QueryInterface(__uuidof(IADORecordBinding),
(LPVOID*)&picRs)))
   _com_issue_error(E_NOINTERFACE);
picRs->BindToRecordset(&m_rsRecSet);
```

4. Các lệnh về Transaction

```
pCon->BeginTrans();
pCon->CommitTrans();
pCon->Rollback();
```

5. Đóng Recordset và đóng kết nối

```
pRs->Close();
pCon->Close();
```

IV. Visual C++ và DB-Library

1. Giới thiệu DB-Library

- **Lập trình sử dụng thư viện DB-Library trong môi trường Visual C++ bao gồm các thao tác cơ bản sau:**
  1. **Tạo kết nối với Microsoft® SQL Server™**
  2. **Đưa câu lệnh Transact-SQL vào vùng đệm (buffer) rồi gửi câu lệnh này cho SQL Server.**

3. **Xử lý kết quả nhận được (nếu có) sau khi yêu cầu SQL Server thực hiện câu lệnh SQL. Có thể chuyển đổi những giá trị của mỗi dòng dữ liệu nhận được (từng field trong mỗi record) về những kiểu dữ liệu chuẩn trong ngôn ngữ C để thuận tiện trong việc thao tác**
4. **Xử lý các lỗi được DB-Library trả về và các message của SQL-Server**
5. **Chấm dứt kết nối đã được tạo với SQL Server**

2. Các tập tin cần thiết của DB-Library

Directory	File	Description
\Mssql7\DevTools\Include	Sqldb.h	DB-Library function prototypes.
	Sqlfront.h	DB-Library type và macro definitions.
	Sqlca.h	SQLCA header
	Sqlda.h	SQLDA header
\Mssql7\DevTools\Lib	Bldblib.lib	Borland large-model DB-Library static library for Microsoft® MS-DOS®.
	Bmdblib.lib	Borland medium-model DB-Library static library for MS-DOS.
	Msdblib3.lib	DB-Library import library for Microsoft Windows®.
	Ntwdblib.lib	DB-Library import library for Microsoft Win32®.
	Rldblib.lib	Large-model DB-Library static library for MS-DOS.
	Rmdblib.lib	Medium-model DB-Library static library for MS-DOS.
	Caw32.lib	Thư viện SQLCA library cho Windows NT và Windows 95/98
	Sqlakw32.lib	Thư viện dành cho Windows NT và Windows 95/98
	Ntwdblib.lib	Thư viện DB-Library sử dụng để giao tiếp với SQL Server
	W3dblib.lib	Old DB-Library import library for Windows.
\Mssql7\Binn	Nsqlprep.exe	32-bit precompiler for Microsoft® Windows NT® and Microsoft Windows® 95/98

	Sqlaiw32.dll	**Thư viện dành cho Windows NT và Windows 95/98**
	Sqlakw32.dll	**Thư viện dành cho Windows NT và Windows 95/98**

3. Các lệnh đặc trưng

a. Hàm dberrhandle và dbmsghandle

- **Syntax:**

```
DBERRHANDLE_PROC dberrhandle (DBERRHANDLE_PROC handler);
DBMSGHANDLE_PROC dbmsghandle (DBMSGHANDLE_PROC handler);
```

- **Hàm dberrhandle cho phép ứng dụng khai báo 1 hàm xử lý các lỗi (error-handling function).**
- **Hàm dbmsghandle cho phép ứng dụng khai báo 1 hàm xử lý các thông điệp (message-handling function)**
- **Chương trình ứng dụng *nên* khai báo hàm xử lý lỗi và xử lý thông điệp riêng**
- **Hàm xử lý lỗi/thông điệp của ứng dụng được tự động gọi mỗi khi ứng dụng gặp lỗi của DB-Library**
- **Ví dụ:**

```
int err_handler (PDBPROCESS dbproc, INT severity,
                 INT dberr, INT oserr, LPCSTR dberrstr,
                 LPCSTR oserrstr)
{
   printf ("DB-Library Error %i: %s\n", dberr, dberrstr);
   if (oserr != DBNOERR)
   {
      printf ("Operating System Error %i: %s\n",
               oserr, oserrstr);
   }
   return (INT_CANCEL);
}
int msg_handler (PDBPROCESS dbproc, DBINT msgno, INT msgstate,
                 INT severity, LPCSTR msgtext, LPCSTR server,
                 LPCSTR procedure, DBUSMALLINT line)
{
    printf ("SQL Server Message %ld: %s\n", msgno, msgtext);
    return (0);
}
```

b. Hàm dbinit

- **Syntax:**

```
LPCSTR dbinit ( void );
```

- **Khởi động thư viện DB-Library.**
- **Được gọi khi bắt đầu chương trình**
- **Kết quả trả về:**
  - **Chuỗi ký tự lưu version của thư viện (thành công)**
  - **NULL (thất bại)**

c. Hàm dblogin

- **Syntax:**

```
PLOGINREC dblogin ( void );
```

- **Thư viện DB-Library sử dụng cấu trúc LOGINREC trong việc tạo kết nối với SQL Server**
- **Hàm dblogin cần được gọi khi bắt đầu sử dụng 1 biến cấu trúc LOGINREC**
- **Ví dụ:**

```
LOGINREC*  loginrec;
loginrec = dblogin();
```

d. Hàm dbopen

- **Syntax:**

```
PDBPROCESS dbopen (PLOGINREC login, LPCSTR servername);
```

- **Các hàm thao tác trên cấu trúc LOGINREC**
  - DBSETLUSER**: Cung cấp UserID**
  - DBSETLPWD **: Cung cấp Password**
  - DBSETLAPP**: Cung cấp tên của ứng dụng (tên của ứng dụng sẽ được liệt kê trong bảng** `sysprocess`**)**
- **Ví dụ:**

```
LOGINREC*  loginrec;
loginrec = dblogin();
DBSETLUSER(loginrec,  "user");
DBSETLPWD  (loginrec, "my_password");
DBSETLAPP  (loginrec, "my_program");
```

- **Hàm dbopen thực hiện việc kết nối với SQL Server**
- **Kết quả trả về là 1 con trỏ đến đối tượng DBPROCESS. Có thể xem 1 đối tượng DBPROCESS đại diện cho 1 connection.**
- **Ví dụ:**

```
PDBPROCESS    dbproc;
dbproc = dbopen(loginrec, “my_server”);
```

e. Hàm dbcmd

- **Syntax:**

```
RETCODE dbcmd(PDBPROCESS dbproc, LPCSTR cmdstring);
```

- **Đưa tiếp tục 1 chuỗi ký tự vào vùng đệm câu lệnh SQL**
- **DB-Library cho phép đưa 1 lần nhiều câu lệnh đến SQL-Server (xử lý command batch)**
- **Kết quả của hàm là** SUCCESS **hay** FAIL
- **Ví dụ:**

```
dbcmd(dbproc, " SELECT *”);
dbcmd(dbproc, " FROM AUTHORS”);
dbcmd(dbproc, " WHERE state = 'CA' ");
```

f. Hàm dbsqlexec

- **Syntax:**

```
RETCODE dbsqlexec ( PDBPROCESS dbproc );
```

- **Kết quả của hàm là** SUCCESS **hay** FAIL
- **Thực hiện (các) câu lệnh trong vùng đệm. Thư viện SQL-Server gửi toàn bộ nội dung trong buffer đến SQL Server, yêu cầu SQL Server phân tích và thực hiện các lệnh này**

- **Khi thực hiện hàm này, chương trình ứng dụng sẽ chờ cho đến khi SQL Server thực hiện xong tất cả các lệnh trong buffer. Để tránh điều này, có thể sử dụng hàm dbsettime để đặt thời gian time-out, hay có thể sử dụng các hàm dbsqlsend, dbdataready, và dbsqlok (thay vì sử dụng hàm dbsqlexec).**

g. Hàm dbresults

- **Syntax:**

```
RETCODE dbresults ( PDBPROCESS dbproc );
```

- **Kết quả của hàm là** NO_MORE_RESULTS, NO_MORE_RPC_RESULTS, SUCCESS **hay** FAIL.
- **Tương ứng với mỗi câu lệnh SQL trong nhóm các lệnh gửi đến cho SQL Server, cần gọi 1 lệnh dbresults, cho dù câu lệnh SQL này không phát sinh kết quả trả về.**
- **Nên gọi dbresults cho đến khi kết quả trả về của hàm là** NO_MORE_RESULTS.

h. Hàm dbbind

- **Syntax:**

```
RETCODE dbbind
        (PDBPROCESS dbproc, INT column,
         INT vartype, DBINT varlen, LPBYTE varaddr );
```

- **Kết buộc dữ liệu trong 1 cột của các record kết quả nhận được với các biến trong chương trình**

Var Type	Var Addr	SQL Server data type of *column*
CHARBIND	DBCHAR	SQLCHAR, SQLVARCHAR, SQLTEXT
STRINGBIND	DBCHAR	SQLCHAR, SQLVARCHAR, SQLTEXT
NTBSTRINGBIND	DBCHAR	SQLCHAR, SQLVARCHAR, SQLTEXT
VARYCHARBIND	DBVARYCHAR	SQLCHAR, SQLVARCHAR, SQLTEXT
BINARYBIND	DBBINARY	SQLBINARY, SQLVARBINARY, SQLIMAGE
VARYBINBIND	DBVARYBIN	SQLBINARY, SQLVARBINARY, SQLIMAGE
TINYBIND	DBTINYINT	SQLINT1, SQLINTN
SMALLBIND	DBSMALLINT	SQLINT2, SQLINTN
INTBIND	DBINT	SQLINT4, SQLINTN
FLT4BIND	DBFLT4	SQLFLT4, SQLFLTN
FLT8BIND	DBFLT8	SQLFLT8, SQLFLTN
BITBIND	DBBIT	SQLBIT
SMALLMONEYBIND	DBMONEY4	SQLMONEY4, SQLMONEYN
MONEYBIND	DBMONEY	SQLMONEY, SQLMONEYN
DECIMALBIND	DBDECIMAL	SQLDECIMAL
NUMERICBIND	DBNUMERIC	SQLNUMERIC
SRCDECIMALBIND	DBDECIMAL	SQLDECIMAL
SRCNUMERICBIND	DBNUMERIC	SQLNUMERIC
SMALLDATETIBIND	DBDATETIM4	SQLDATETIM4, SQLDATETIMN
DATETIMEBIND	DBDATETIME	SQLDATETIME, SQLDATETIMN

i. Hàm dbnextrow

- **Syntax:**

```
STATUS dbnextrow ( PDBPROCESS dbproc );
```

- **Đọc 1 dòng dữ liệu (1 record) và đưa các giá trị của từng field vào các biến của chương trình (được xác định bằng các hàm dbbind trước đó)**
- **Khi tất cả các dòng dữ liệu đã được xử lý hết, hàm dbnextrow trả về giá trị** NO_MORE_ROWS
- **Ví dụ:**

```
DBINT   xvariable;
DBCHAR  yvariable[10];

// Đưa câu lệnh Transact-SQL vào buffer
dbcmd(dbproc, "SELECT x = 100, y = 'hello'");

// Gửi lệnh SQL đến SQL Server, yêu cầu thực hiện
dbsqlexec(dbproc);

// Chuẩn bị thao tác trên dữ liệu kết quả nhận được
dbresults(dbproc);
// Kết buộc mỗi cột trong record với các biến của chương trình ứng dụng
dbbind(dbproc, 1, INTBIND, (DBINT) 0, (BYTE *)&xvariable);
dbbind(dbproc, 2, STRINGBIND, (DBINT) 0, yvariable);

// Xử lý từng record
while (dbnextrow(dbproc) != NO_MORE_ROWS)
{
  //   Đoạn code bằng ngôn ngữ C thao tác trên các field của từng record
}
```

j. Hàm dbclose

```
RETCODE dbclose ( PDBPROCESS dbproc );
```

- **Chấm dứt kết nối dbproc**

k. Hàm dbexit

```
void dbexit ( void );
```

- **Chấm dứt tất cả các kết nối được tạo ra trong chương trình và giải phóng tất cả các đối tượng DBPROCESS.**
- **Thường được gọi trước khi chấm dứt chương trình**

4. Ví dụ

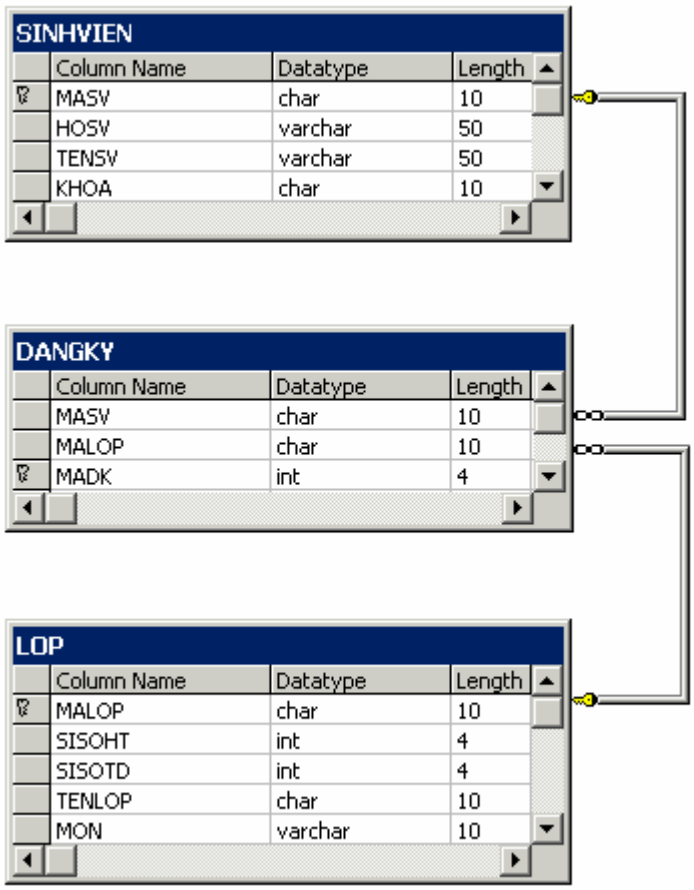
```
#define DBNTWIN32
#include <stdio.h>
#include <windows.h>
#include <sqlfront.h>
#include <sqldb.h>

// Forward declarations of the error handler and message handler.

int err_handler(PDBPROCESS, INT, INT, INT, LPCSTR, LPCSTR);
int msg_handler(PDBPROCESS, DBINT, INT, INT, LPCSTR, LPCSTR,
                LPCSTR, DBUSMALLINT);

void main(void)
```

```
{
    PDBPROCESS  dbproc;    // The connection with SQL Server.
    PLOGINREC   login;     // The login information.
    DBCHAR      name[100];
    DBCHAR      city[100];

    // Install user-supplied error- and message-handling functions.
    dberrhandle (err_handler);
    dbmsghandle (msg_handler);

    // Initialize DB-Library.
    dbinit ();

    // Get a LOGINREC.
    login = dblogin ();
    DBSETLUSER (login, "my_login");
    DBSETLPWD (login, "my_password");
    DBSETLAPP (login, "example");

    // Get a DBPROCESS structure for communication with SQL Server.
    dbproc = dbopen (login, "my_server");

    // Retrieve some columns from the authors table in the
    // pubs database.

    // First, put the command into the command buffer.
    dbcmd(dbproc, "SELECT au_lname, city FROM pubs..authors");
    dbcmd(dbproc, " WHERE state = 'CA' ");

    // Send the command to SQL Server and start execution.
    dbsqlexec (dbproc);

    // Process the results.
    if (dbresults (dbproc) == SUCCEED)
    {
        // Bind column to program variables.
        dbbind (dbproc, 1, NTBSTRINGBIND, 0, name);
        dbbind (dbproc, 2, NTBSTRINGBIND, 0, city);

        // Retrieve and print the result rows.
        while (dbnextrow (dbproc) != NO_MORE_ROWS)
        {
            printf ("%s from %s\n", name, city);
        }
    }

    // Close the connection to SQL Server.
    dbexit ();

    return (0);
}
```

3

Ứng dụng minh họa

I. Sơ đồ logic (VOPC – View of Participating Classes)



II. Các giao tác chính

STT	Mã số	Giao tác	Đối tượng liên quan
1	T1	Thêm một đăng ký (lớp mà sinh viên đăng ký phải còn chỗ và khi thêm phải tăng sĩ số sinh viên trong lớp).	SINHVIEN, LOP, DANGKY
2	T2	Rút một đăng ký (khi rút một đăng ký phải giảm sĩ số sinh viên trong lớp).	SINHVIEN, LOP, DANGKY

1. Giao tác "Đăng ký Học phần"

Nếu (yêu cầu đăng ký chưa có trong CSDL)
  Nếu (sĩ số hiện tại của lớp < sĩ số tối đa của lớp)
    Thêm 1 đăng ký vào CSDL
    Tăng sĩ số hiện tại của lớp
  Ngược lại
    Thông báo hết chỗ
Ngược lại
  Thông báo yêu cầu đăng ký đã có trong CSDL

a. Bước 1:

```
BEGIN TRAN
SELECT  @SISOHT = SISOHT,  @SISOTD = SISOTD
FROM  LOP
WHERE MALOP = @MALOP
IF (@@ROWCOUNT=0)
   ROLLBACK TRAN
IF (@SISOHT>=@SISOTD)
   ROLLBACK TRAN
INSERT INTO DANGKY (MASV, MALOP)
VALUES (@MASV, @MALOP)
SET     SISOHT  = SISOHT+1
WHERE   MALOP   = @MALOP
COMMIT TRAN
```

b. Bước 2:

```
BEGIN TRAN
SELECT  @SISOHT = SISOHT,  @SISOTD = SISOTD
FROM  LOP
WHERE MALOP = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRAN
IF (@@ROWCOUNT=0)
   ROLLBACK TRAN
IF (@SISOHT>=@SISOTD)
   ROLLBACK TRAN
INSERT INTO DANGKY (MASV, MALOP)
VALUES (@MASV, @MALOP)
IF (@@ERROR<>0)
   ROLLBACK TRAN
SET     SISOHT  = SISOHT+1
WHERE   MALOP   = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRAN
COMMIT TRAN
```

c. Bước 3-4:

```
BEGIN TRAN
SELECT  @SISOHT = SISOHT,  @SISOTD = SISOTD
```

```
FROM LOP  WITH (UPDLOCK, REPEATABLEREAD)
WHERE MALOP = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRAN
IF (@@ROWCOUNT=0)
   ROLLBACK TRAN
IF (@SISOHT>=@SISOTD)
   ROLLBACK TRAN
INSERT INTO DANGKY (MASV, MALOP)
VALUES (@MASV, @MALOP)
IF (@@ERROR<>0)
   ROLLBACK TRAN
SET     SISOHT  = SISOHT+1
WHERE   MALOP   = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRAN
COMMIT TRAN
```

2. Giao tác "Hủy đăng ký Học phần"

Nếu (tìm thấy yêu cầu đăng ký cần xoá trong CSDL)

Xóa 1 đăng ký cần thiết khỏi CSDL

Giảm sĩ số hiện tại của lớp

Ngược lại

Thông báo không tìm thấy yêu cầu đăng ký cần xoá trong CSDL

a. Bước 1:

```
BEGIN TRAN
DELETE  FROM DANGKY
WHERE   (@MALOP = MALOP) AND (@MASV = MASV)
IF (@@ROWCOUNT=0)
   ROLLBACK TRAN
UPDATE  LOP
SET     SISOHT = SISOHT-1
WHERE   MALOP = @MALOP
COMMIT TRAN
```

b. Bước 2:

```
BEGIN TRAN
DELETE  FROM DANGKY
WHERE   (@MALOP = MALOP) AND (@MASV = MASV)
IF (@@ROWCOUNT=0)
   ROLLBACK TRAN
IF (@@ERROR<>0)
   ROLLBACK TRAN
UPDATE  LOP
SET     SISOHT = SISOHT-1
WHERE   MALOP = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRAN
COMMIT TRAN
```

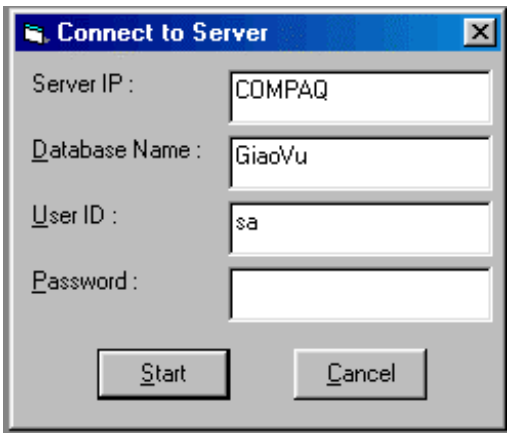
c. Bước 3-4:

```
BEGIN TRAN
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
DELETE  FROM  DANGKY
WHERE   (@MALOP = MALOP) AND (@MASV = MASV)
IF (@@ROWCOUNT=0)
   ROLLBACK TRAN
IF (@@ERROR<>0)
   ROLLBACK TRAN
UPDATE  LOP
SET     SISOHT = SISOHT-1
WHERE   MALOP = @MALOP
IF (@@ERROR<>0)
   ROLLBACK TRAN
COMMIT TRAN
```

III. Cài đặt:

1. Xây dựng giao diện Login

a. Giao diện:



b. Thủ tục kết nối cơ sở dữ liệu trên VB (Sử dụng ADO):

```
Public fMainForm As FrmMain
Private fLogin As frmLogin
Public cn As ADODB.Connection

Sub Main()
   On Error GoTo LoginError
   Set fLogin = New frmLogin
   fLogin.Show vbModal

   If fLogin.bOK = True Then
      Dim sCnString As String

      sCnString = fLogin.sCnString
```

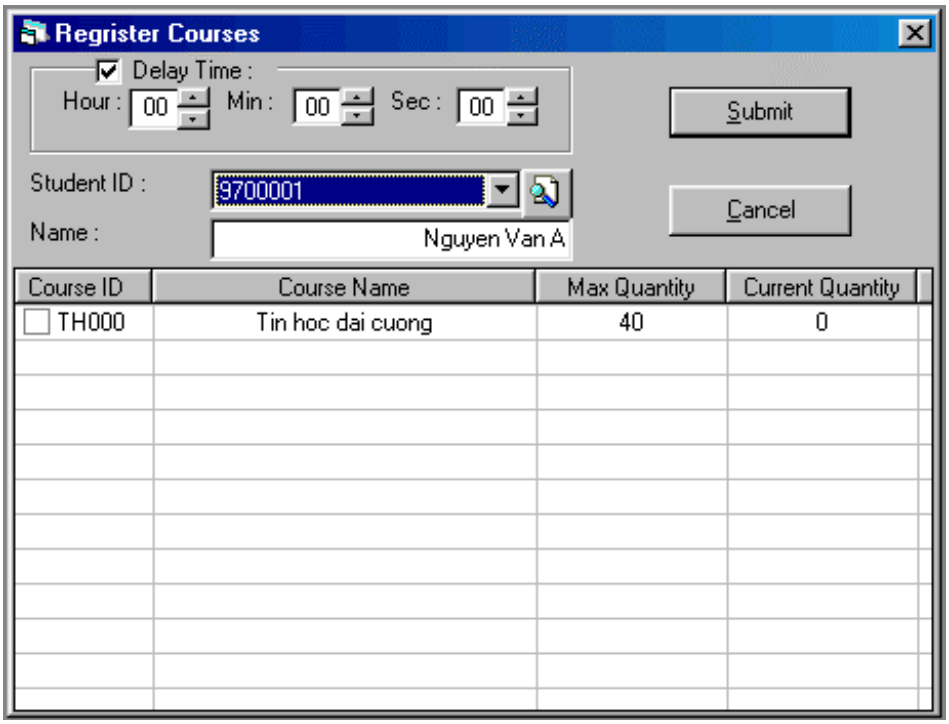
```
        Set cn = New ADODB.Connection
        cn.ConnectionString = sCnString
        cn.ConnectionTimeout = 5
        cn.Open

        Unload fLogin
        Set fMainForm = New FrmMain
        fMainForm.Show
      Else
        Unload fLogin
      End If
      Exit Sub
    LoginError:
      MsgBox "Khong the login vao database. Ket thuc chuong trinh.",
    vbCritical, "Error"
      Unload fLogin
    End Sub
```

c. Thủ tục kết nối cơ sở dữ liệu trên VC++ (Sử dụng DB-Library):

```
int err_handler(PDBPROCESS, INT, INT, INT, LPCSTR, LPCSTR);
int msg_handler(PDBPROCESS, DBINT, INT, INT, LPCSTR, LPCSTR,
                LPCSTR, DBUSMALLINT);

BOOL       Initialization()
{
    // Initialize DB-Library.
    if (dbinit () == NULL)
        return FALSE;
    // Install user-supplied error- and message-handling functions.
    dberrhandle (err_handler);
    dbmsghandle (msg_handler);
    return TRUE;
}

PDBPROCESS CreateConnection()
{
    PDBPROCESS  dbproc;    // The connection with SQL Server.
    PLOGINREC   login;     // The login information.


    // Get a LOGINREC.
    login = dblogin ();
    DBSETLUSER (login, "my_login");
    DBSETLPWD (login,  "my_password");
    DBSETLAPP (login,  "example");

    // Get a DBPROCESS structure for communication with SQL Server.
    dbproc = dbopen (login, "my_server");
    return dbproc;
}
```

2. Xây dựng giao diện đăng ký

a. Giao diện :



Sinh viên sau khi nhập Mã sinh viên, List sẽ tự động hiện các môn sinh viên có thể đăng ký. Khi nhấn Submit thì các môn được sinh viên đánh dấu sẽ lần lượt thực hiện đăng ký.

Giờ trì hoãn chính là thời gian ngắt giữa chừng một đăng ký. Mục đích là để một giao tác đăng ký khác chen vào và sự tranh chấp giữa các giao tác sẽ xuất hiện.

b. Thủ tục gọi Transaction từ VB (Sử dụng ADO):

```
Private Sub DangKyHocPhan(MaSinhVien As String, MaLopHoc As String, WaitTime As String)
    Dim cmd As New ADODB.Command
    Dim param As Parameter

    ' Set up a command object for the stored procedure.
    Set cmd.ActiveConnection = cn
    cmd.CommandText = "sp_DANGKY_HOCPHAN"
    cmd.CommandType = adCmdStoredProc

    ' Set up a return parameter.
```

```
        Set param = cmd.CreateParameter("Return", adInteger,
    adParamReturnValue)
        cmd.Parameters.Append param
        ' Set up an input parameter.
        Set param = cmd.CreateParameter("MASV", adChar,
    adParamInput, 10)
        cmd.Parameters.Append param
        param.Value =MaSinhVien

        Set param = cmd.CreateParameter("MALOP", adChar,
      adParamInput, 10)
        cmd.Parameters.Append param
        param.Value = MaLopHoc

        Set param = cmd.CreateParameter("TIMEWAIT", adChar,
      adParamInput, 10)
        cmd.Parameters.Append param
        param.Value = WaitTime

        ' Set up an output parameter.
        Set param = cmd.CreateParameter("Output", adInteger,
    adParamOutput)
        cmd.Parameters.Append param

        ' Execute command.
        cmd.Execute
      End Sub
```

c. Thủ tục gọi Transaction từ VC++(Sử dụng DB-Library):

```
BOOL DangKyHocPhan
     (CString MaSinhVien, CString MaLopHoc,  CString WaitTime)
{
   CString SqlCommand;
   dbuse(dbproc, _T("COURSE"));
   dbcmd(dbproc, "EXEC sp_DANGKY_HOCPHAN ");
   dbcmd(dbproc, MaSinhVien);
   dbcmd(dbproc, MaLopHoc);
   dbcmd(dbproc, WaitTime);

   if (dbsqlexec (dbproc))
   {
      dbresults(dbproc);
      dbcancel(dbproc);
      return TRUE;
   }
   return FALSE;
}
```

Tài liệu tham khảo

1. **Database Systems Handbook**, *Paul J.Fortier, Ph.D.*, McGraw-Hill, 1997

2. **MCSE TestPrep – SQL Server 6.5 – Design & Implementation**, *Owen Williams, Deanna Townsend, Arun Singhal, Vilas Ekbote, Corby Jordan, Christian Plazas*, New Riders, 1998

3. **Microsoft® Visual Basic 6.0 và Lập trình Cơ sở Dữ liệu, *Nguyễn thị Ngọc Mai*, NXB Giáo Dục, 2000**

4. **Inside Distributed COM**, *Guy Eddon, Henry Eddon*, Microsoft Press, 1998

5. **Teach Yourself Microsoft SQL Server 6.5 in 21 Days**, *Rick Sawtell, Richard Waymire, Lance Mortensen, Kalen Delaney, Roberta Bragg, Darren Brinksneader*, SAMS, 1998

6. **Special Edition: Using SQL Server 6.5**, *Bob Branchek, Peter Hazlehurst, Stephen Wynkoop, Scott L. Warner*, QUE, 1998

7. **ADO and SQL Server Developer's Guide**, *Joyce Chen, Richard Patterson*, 1998

8. **Microsoft® MSDN 2000**