# GIAO TÁC PHÂN TÁN

## I. GIAO TÁC TRÊN CƠ SỞ DỮ LIỆU TẬP TRUNG:

### 1. Biến @@TRANCOUNT: trả về số giao tác đang hoạt động trên kết nối hiện tại

**Kiểu trả về :** integer

Lệnh **BEGIN TRANSACTION** sẽ tăng @@TRANCOUNT thêm 1. ROLLBACK TRANSACTION sẽ cho @@TRANCOUNT về 0. Tuy nhiên, nếu ta dùng ROLLBACK TRANSACTION *savepoint_name* thì sẽ không ảnh hưởng đến biến @@TRANCOUNT. Lệnh COMMIT TRANSACTION hoặc COMMIT WORK sẽ giảm bớt 1 trên biến @@TRANCOUNT .

**Ví dụ:**

```
Create Proc sp_UpperName
  @ten varchar(50)
AS
    BEGIN TRANSACTION
    UPDATE nhanvien SET ten = upper(ten)
        WHERE ten = @ten
    IF @@ROWCOUNT = 2
      COMMIT TRAN

    IF @@TRANCOUNT > 0
    BEGIN                    -- update
      ROLLBACK TRAN
      PRINT 'Lenh da bi huy'
    END
```

Giải thích chức năng của sp_UpperName ?

2. SAVE TRANSACTION

Ghi lại 1 vị trí trong giao tác

**Syntax**

SAVE TRANSACTION { *savepoint_name* | *@savepoint_variable* }

**Arguments**

*savepoint_name:* là tên được gán cho vị trí mà ta muốn ghi lại. Tên của savepoint giống như tên của id (tối đa là 32 ký tự)
***@****savepoint_variable :* biến chứa tên của savepoint. Biến này chỉ thuộc về các kiểu **char**, **varchar**, **nchar**, **or nvarchar** .

The savepoint định nghĩa 1 vị trí trong giao tác để ta có thể hủy 1 phần lệnh trong giao tác .
**SAVE TRANSACTION không hỗ trợ trong môi trường *distributed transactions* .**

**Examples**

This example changes the royalty (tiền bản quyền) split for the two authors of *The Gourmet Microwave*. Because the database would be inconsistent between the two updates, they must be grouped into a user-defined transaction.

```
BEGIN TRANSACTION royaltychange
  UPDATE titleauthor
    SET royaltyper = 65
          WHERE royaltyper = 75
          AND title = 'The Gourmet Microwave'

  UPDATE titleauthor
    SET royaltyper = 35
        WHERE royaltyper = 25
          AND title = 'The Gourmet Microwave'

SAVE TRANSACTION percentchanged

/*
After having updated the royaltyper entries for the two authors, the
user inserts the savepoint percentchanged, and then determines how a
10-percent increase in the book's price would affect the authors' royalty earnings.  */
UPDATE titles
  SET price = price * 1.1
    WHERE title = 'The Gourmet Microwave'

SELECT (price * royalty * ytd_sales) * royaltyper
```

```
  FROM titles, titleauthor
    WHERE title = 'The Gourmet Microwave'
      AND titles.title_id = titleauthor.title_id

/* The transaction is rolled back to the savepoint with the ROLLBACK TRANSACTION
statement.
*/
 ROLLBACK percentchanged

COMMIT TRANSACTION
/* End of royaltychange. */
```

### 3. BEGIN TRANSACTION WITH MARK

Đánh dấu điểm bắt đầu của 1 giao tác cục bộ. BEGIN TRANSACTION sẽ làm tăng @@TRANCOUNT lên 1.

**Syntax**

```
BEGIN TRANSACTION [ transaction_name | @tran_name_variable
   [ WITH MARK [ 'description' ] ] ]
```

**Arguments**

*transaction_name*
Is the name assigned to the transaction. *transaction_name* must conform to the rules for identifiers but identifiers longer than 32 characters are not allowed. Use transaction names only on the outermost pair of nested BEGIN...COMMIT or BEGIN...ROLLBACK statements.
***@****tran_name_variable*
Is the name of a user-defined variable containing a valid transaction name. The variable must be declared with a **char**, **varchar**, **nchar**, or **nvarchar** data type.
WITH MARK ['*description*']
Chỉ ra  transaction được đánh dấu trong file nhật ký. *description* là 1 chuỗi mô tả ý nghĩa vị trí đánh dấu.
Nếu WITH MARK được dùng, tên giao tác phải có trong câu lệnh. WITH MARK cho phép phục hồi giao tác tới vị trí này

**Remarks**

BEGIN TRANSACTION represents a point at which the data referenced by a connection is logically and physically consistent. Nếu phát hiện lỗi, tất cả data đã thay đổi sẽ quay trở lại trạng thái ban đầu.
BEGIN TRANSACTION starts a local transaction for the connection issuing the statement. Depending on the current transaction isolation level settings, many resources acquired to support the Transact-SQL statements issued by the connection are locked by

the transaction until it is completed with either a COMMIT TRANSACTION or ROLLBACK TRANSACTION statement.

Although BEGIN TRANSACTION starts a local transaction, it is not recorded in the transaction log until the application subsequently performs an action that must be recorded in the log, such as executing an INSERT, UPDATE, or DELETE statement. An application can perform actions such as acquiring locks to protect the transaction isolation level of SELECT statements, but nothing is recorded in the log until the application performs a modification action.
Naming multiple transactions in a series of nested transactions with a transaction name has little effect on the transaction. Only the first (outermost) transaction name is registered with the system. A rollback to any other name (other than a valid savepoint name) generates an error. None of the statements executed before the rollback are in fact rolled back at the time this error occurs. The statements are rolled back only when the outer transaction is rolled back.
BEGIN TRANSACTION starts a local transaction. The local transaction is escalated to a distributed transaction if the following actions are performed before it is committed or rolled back:

- An INSERT, DELETE, or UPDATE statement is executed that references a remote table on a linked server. The INSERT, UPDATE, or DELETE statement fails if the OLE DB provider used to access the linked server does not support the **ITransactionJoin** interface.

- A call is made to a remote stored procedure when the REMOTE_PROC_TRANSACTIONS option is set to ON.

The local copy of SQL Server becomes the transaction controller and uses MS DTC to manage the distributed transaction.

**Marked Transactions**

The WITH MARK option causes the transaction name to be placed in the transaction log. When restoring a database to an earlier state, the marked transaction can be used in place of a date and time. For more information, see Restoring a Database to a Prior State, Recovering to a Named Transaction, and RESTORE.
Additionally, transaction log marks are necessary if you need to recover a set of related databases to a logically consistent state. Marks can be placed in the transaction logs of the related databases by a distributed transaction. Recovering the set of related databases to these marks results in a set of databases that are transactionally consistent. Placement of marks in related databases requires special procedures. For more information, see Backup and Recovery of Related Databases.
The mark is placed in the transaction log only if the database is updated by the marked transaction. Transactions that do not modify data are not marked.
BEGIN TRAN *new_name* WITH MARK can be nested within an already existing transaction that is not marked. Upon doing so, *new_name* becomes the mark name for the

transaction, despite the name that the transaction may already have been given. In the following example, M2 is the name of the mark.

```
BEGIN TRAN T1
  UPDATE table1 ...
  BEGIN TRAN M2 WITH MARK
    UPDATE table2 ...
    SELECT * from table1
    COMMIT TRAN M2
    UPDATE table3 ...
COMMIT TRAN T1
```

Attempting to mark a transaction that is already marked results in a warning (not error) message:

```
BEGIN TRAN T1 WITH MARK
UPDATE table1 ...
BEGIN TRAN M2 WITH MARK

Server: Msg 3920, Level 16, State 1, Line 3
WITH MARK option only applies to the first BEGIN TRAN WITH MARK.
The option is ignored.
```

## II. GIAO TÁC PHÂN TÁN :

### 1. Dịch vụ MS DTC

The Microsoft Distributed Transaction Coordinator (MS DTC) là 1 trình quản lý, điều phối các giao tác phân tán, nó cho phép các ứng dụng của client thao tác lên dữ liệu của các data sources trong 1 giao tác phân tán.

*The MS DTC service điều phối sự đúng đắn của 1 giao tác phân tán, nó bảo đảm rằng hoặc là tất cả các cập nhật dữ liệu trên tất cả các servers là được thực hiện, hoặc trong trường hợp có lỗi thì xem như chưa thực hiện thao tác gì trên giao tác đó.*

Lưu ý: Nếu dịch vụ MSDTC không hoạt động, Cách khắc phục như sau:
+ Vào CMD và gõ lệnh msdtc.exe -install
+ Sau đó start service tên Distributed Transaction Coordinator
Nếu không thể khởi động được service thì làm như sau:
+ Vào CMD gõ lệnh msdtc -resetlog
+ sau đó gõ tiếp lệnh net start msdtc

### 2. Cú pháp : BEGIN DISTRIBUTED TRANSACTION

Khởi đầu của distributed transaction được quản lý bởi Microsoft Distributed Transaction Coordinator (MS DTC).

**BEGIN DISTRIBUTED { TRAN | TRANSACTION }**
**[ *transaction_name* | *@tran_name_variable* ] ;**

**Arguments**

*transaction_name*
là tên giao tác do user định nghĩa; tên giao tác phải tuân thủ qui tắc đặt tên cho danh hiệu và phải <= 32 ký tự.

*@tran_name_variable*
là tên của 1 biến chứa tên của giao tác. Biến phải thuộc 1 trong các kiểu **char**, **varchar**, **nchar**, hoặc **nvarchar** .

The instance of the SQL Server Database Engine thực thi lệnh BEGIN DISTRIBUTED TRANSACTION là transaction originator và điều khiển việc hoàn thành của transaction. Khi lệnh COMMIT TRANSACTION hoặc ROLLBACK TRANSACTION được thực thi trong session, instance điều khiển sẽ yêu cầu MS DTC hoàn tất distributed transaction trên các instances có liên quan.

Transaction-level snapshot isolation does not support distributed transactions

The primary way remote instances of the Database Engine are enlisted in a distributed transaction is when a session already enlisted in the distributed transaction executes a distributed query referencing a linked server.

For example, if BEGIN DISTRIBUTED TRANSACTION is issued on ServerA, the session calls a stored procedure on ServerB and another stored procedure on ServerC. The stored procedure on ServerC executes a distributed query against ServerD, and then all four computers are involved in the distributed transaction. The instance of the Database Engine on ServerA is the originating controlling instance for the transaction.

**Permissions** : Requires membership in the **public** role.

## Examples

Ví dụ sau sẽ giảm bớt 1 đơn vị của số lượng vật tư có mã 'TL01' thuộc bảng CT_PHATSINH trong 2 phân mảnh. Trong bảng này, ta đã thiết lập 1 Check Constraint SOLUONG > 0. Như vậy, giả sử trong Server hiện tại ta có SOLUONG vật tư 'TL01' là 3, còn trong SERVER tại LINK1 là 2 thì giao tác phân tán sẽ COMMIT,nghĩa là trong Server hiện tại ta sẽ có SOLUONG vật tư 'TL01' là 2, còn trong SERVER tại LINK1 là 1.

Nhưng nếu thực thi giao tác này thêm 1 lần nữa thì hệ thống sẽ roll back toàn bộ giao tác, nghĩa là trong Server hiện tại ta vẫn có SOLUONG vật tư 'TL01' là 2, còn trong SERVER tại LINK1 thì vẫn là 1. Lúc này, ta sẽ nhận 1 thông báo lỗi:

```
Msg 547, Level 16, State 1, Line 1
The UPDATE statement conflicted with the CHECK constraint "CK_SOLUONG". The conflict occurred in
database "QL_VATTU", table "dbo.CT_PHATSINH", column 'SOLUONG'.
```

```
  SET XACT_ABORT ON
  BEGIN DISTRIBUTED TRANSACTION;
  -- Update SOLUONG from local instance.
    UPDATE CT_PHATSINH
      SET SOLUONG = SOLUONG -1
      WHERE MAVT = 'TL01';
  -- Update SOLUONG from remote instance.
    UPDATE LINK1.QL_VATTU.DBO.CT_PHATSINH
      SET SOLUONG = SOLUONG -1
      WHERE MAVT = 'TL01';

    COMMIT TRANSACTION;
  GO
```

Ví dụ 2: Giả sử ta đã tạo Stored Proc sau trên Server Publisher, và đã được đẩy qua các Subscription:

```
CREATE PROCEDURE  SP_UPDATE_SOLUONG
  @MAVT VARCHAR (5), @SOLG INT
AS
BEGIN
UPDATE CT_PHATSINH
    SET SOLUONG = SOLUONG -@SOLG
    WHERE MAVT = @MAVT;
```

```
END
```

Ta thực hiện lại ví dụ 1, nhưng theo dạng RPC thì kết quả vẫn như cũ

```
SET XACT_ABORT ON
BEGIN DISTRIBUTED TRANSACTION;

  EXEC SP_UPDATE_SOLUONG 'TL01', 1

  EXEC LINK1.QL_VATTU.dbo.SP_UPDATE_SOLUONG 'TL01', 1

COMMIT TRANSACTION;
```

**Locking**

Microsoft® SQL Server™ uses locking to ensure transactional integrity and database consistency. Locking prevents users from reading data being changed by other users, and prevents multiple users from changing the same data at the same time. If locking is not used, data within the database may become logically incorrect, and queries executed against that data may produce unexpected results.

Although SQL Server enforces locking automatically, you can design applications that are more efficient by understanding and customizing locking in your applications.

**Customizing the Lock Time-out**

When Microsoft® SQL Server™ cannot grant a lock to a transaction on a resource because another transaction already owns a conflicting lock on that resource, the first transaction becomes blocked waiting on that resource. If this causes a deadlock, SQL Server terminates one of the participating transactions (with no time-out involved).

If there is no deadlock, the transaction requesting the lock is blocked until the other transaction releases the lock. By default, there is no mandatory time-out period, and no way to test if a resource is locked before locking it, except to attempt to access the data (and potentially get blocked indefinitely).

> **Note** The **sp_who** system stored procedure can be used to determine if a process is being blocked, and who is blocking it.

The LOCK_TIMEOUT setting allows an application to set a maximum time that a statement waits on a blocked resource. When a statement has waited longer than the LOCK_TIMEOUT setting, the blocked statement is canceled automatically, and error message 1222 "Lock request time-out period exceeded" is returned to the application.

However, any transaction containing the statement is not rolled back or canceled by SQL Server. Therefore, the application must have an error handler that can trap error message 1222. If an application does not trap the error, it can proceed unaware that an individual statement within a transaction has been canceled, and errors can occur because statements later in the transaction may depend on the statement that was never executed.

Implementing an error handler that traps error message 1222 allows an application to handle the time-out situation and take remedial action for example, automatically resubmitting the statement that was blocked, or rolling back the entire transaction.
To determine the current LOCK_TIMEOUT setting, execute the @@LOCK_TIMEOUT function, for example:

```
DECLARE @Timeout int
SELECT @Timeout = @@lock_timeout
SELECT @Timeout
GO
```

**@@LOCK_TIMEOUT**

Returns the current lock time-out setting, in milliseconds, for the current session.

**Syntax**

@@LOCK_TIMEOUT

**Return Types :** integer

**Remarks**

SET LOCK_TIMEOUT allows an application to set the maximum time that a statement waits on a blocked resource. When a statement has waited longer than the LOCK_TIMEOUT setting, the blocked statement is automatically canceled, and an error message is returned to the application.
At the beginning of a connection, @@LOCK_TIMEOUT returns a value of -1.

**Examples**

This example shows the result set when a LOCK_TIMEOUT value is not set.

```
SELECT @@LOCK_TIMEOUT
```

Here is the result set:

```
----------
-1
```

This example sets LOCK_TIMEOUT to 1800 milliseconds, and then calls @LOCK_TIMEOUT.

```
SET LOCK_TIMEOUT 1800
SELECT @@LOCK_TIMEOUT
```

Here is the result set:

```
------------------------------
1800
```

**SET LOCK_TIMEOUT**

Specifies the number of milliseconds a statement waits for a lock to be released.

**Syntax**

SET LOCK_TIMEOUT *timeout_period*

**Arguments**

*timeout_period*
Is the number of milliseconds that will pass before Microsoft® SQL Server™ returns a locking error. A value of -1 (default) indicates no time-out period (that is, wait forever). When a wait for a lock exceeds the time-out value, an error is returned. A value of 0 means not to wait at all and return a message as soon as a lock is encountered.

**Remarks**

At the beginning of a connection, this setting has a value of -1. After it is changed, the new setting stays in effect for the remainder of the connection.
The setting of SET LOCK_TIMEOUT is set at execute or run time and not at parse time.
The READPAST locking hint provides an alternative to this SET option.

**Permissions**

SET LOCK_TIMEOUT permissions default to all users.

**Examples**

This example sets the lock time-out period to 1,800 milliseconds.

```
SET LOCK_TIMEOUT 1800
GO
```

**Locking Hints**

A range of table-level locking hints can be specified using the SELECT, INSERT, UPDATE, and DELETE statements to direct Microsoft® SQL Server™ 2000 to the type of locks to be used. Table-level locking hints can be used when a finer control of the types of locks acquired on an object is required. These locking hints override the current transaction isolation level for the session.

> **Note** The SQL Server query optimizer automatically makes the correct determination. It is recommended that table-level locking hints be used to change the default locking behavior only when necessary. Disallowing a locking level can affect concurrency adversely.

| Locking hint | Description |
|---|---|
| HOLDLOCK | Hold a shared lock until completion of the transaction instead of releasing the lock as soon as the required table, row, or data page is no longer required. HOLDLOCK is equivalent to SERIALIZABLE. |
| NOLOCK | Do not issue shared locks and do not honor exclusive |

| | |
|---|---|
| | locks. When this option is in effect, it is possible to read an uncommitted transaction or a set of pages that are rolled back in the middle of a read. Dirty reads are possible. Only applies to the SELECT statement. |
| PAGLOCK | Use page locks where a single table lock would usually be taken. |
| READCOMMITTED | Perform a scan with the same locking semantics as a transaction running at the READ COMMITTED isolation level. By default, SQL Server 2000 operates at this isolation level. |
| READPAST | Skip locked rows. This option causes a transaction to skip rows locked by other transactions that would ordinarily appear in the result set, rather than block the transaction waiting for the other transactions to release their locks on these rows. The READPAST lock hint applies only to transactions operating at READ COMMITTED isolation and will read only past row-level locks. Applies only to the SELECT statement. |
| READUNCOMMITTED | Equivalent to NOLOCK. |
| REPEATABLEREAD | Perform a scan with the same locking semantics as a transaction running at the REPEATABLE READ isolation level. |
| ROWLOCK | Use row-level locks instead of the coarser-grained page- and table-level locks. |
| SERIALIZABLE | Perform a scan with the same locking semantics as a transaction running at the SERIALIZABLE isolation level. Equivalent to HOLDLOCK. |
| TABLOCK | Use a table lock instead of the finer-grained row- or page-level locks. SQL Server holds this lock until the end of the statement. However, if you also specify HOLDLOCK, the lock is held until the end of the transaction. |
| TABLOCKX | Use an exclusive lock on a table. This lock prevents others from reading or updating the table and is held until the end of the statement or transaction. |
| UPDLOCK | Use update locks instead of shared locks while reading a table, and hold locks until the end of the statement or transaction. UPDLOCK has the advantage of allowing you to read data (without blocking other readers) and update it later with the assurance that the data has not changed since you last read it. |

| | |
|---|---|
| XLOCK | Use an exclusive lock that will be held until the end of the transaction on all data processed by the statement. This lock can be specified with either PAGLOCK or TABLOCK, in which case the exclusive lock applies to the appropriate level of granularity. |

For example, if the transaction isolation level is set to SERIALIZABLE, and the table-level locking hint NOLOCK is used with the SELECT statement, key-range locks typically used to maintain serializable transactions are not taken.

```
USE pubs
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
BEGIN TRANSACTION
SELECT au_lname FROM authors WITH (NOLOCK)
GO
```

The locks generated are:

```
EXEC sp_lock
GO
```

| spid | dbid | ObjId | IndId | Type | Resource | Mode | Status |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | DB | | S | GRANT |
| 6 | 1 | 0 | 0 | DB | | S | GRANT |
| 7 | 1 | 0 | 0 | DB | | S | GRANT |
| 8 | 4 | 0 | 0 | DB | | S | GRANT |
| 8 | 4 | 0 | 0 | DB | | S | GRANT |
| 8 | 4 | 117575457 | 0 | TAB | | Sch-S | GRANT |
| 9 | 4 | 0 | 0 | DB | | S | GRANT |
| 9 | 1 | 21575115 | 0 | TAB | | IS | GRANT |

```
SELECT object_name(117575457)
GO
-----------------------------
authors
```

The only lock taken that references **authors** is a schema stability (Sch-S) lock. In this case, serializability is no longer guaranteed.