## Lab Experiment 11

### Demonstration & Implementation of SQL concepts related to DB backup/restoration and Transactions

### 

### Objectives

* Demonstrating the implementation of Database backup & Recovery Commands
* Demonstrating the implementation of Transactions, commit, rollback and save points

### Introduction

#### Backup a database

Any database can be backed up using either SSMS or SQL statements. See below:

1. Launch SQL Server Management Studio (SSMS) and connect to your SQL Server instance.
2. Expand the Databases node in Object Explorer.
3. Right-click the database, hover over Tasks, and select Back up...
4. Under Destination, confirm that the path for your backup is correct. If you need to change the path, select Remove to remove the existing path, and then Add to type in a new path. You can use the ellipses to navigate to a specific file.
5. Select OK to take a backup of your database.

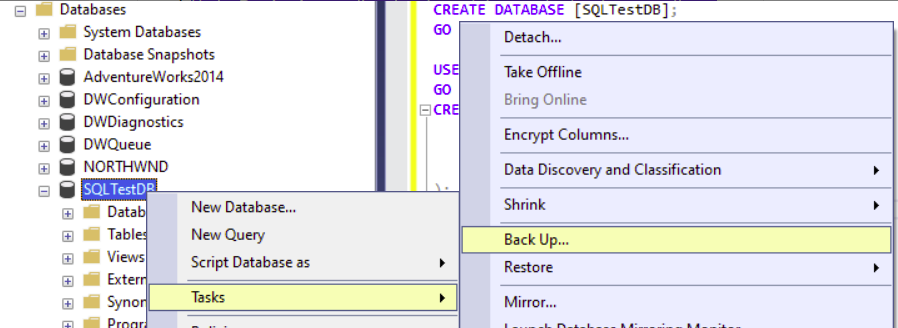


Fig. 1. Backing up a database using SSMS

Same can be done by using SQL statements:

*USE [master];*

*GO*

*BACKUP DATABASE [SQLTestDB]*

*TO DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\Backup\SQLTestDB.bak'*

*WITH NOFORMAT, NOINIT,*

*NAME = N'SQLTestDB-Full Database Backup', SKIP, NOREWIND, NOUNLOAD, STATS = 10;*

*GO*

More information about SQL’s back up command can be found [here](https://learn.microsoft.com/en-us/sql/t-sql/statements/backup-transact-sql?view=sql-server-ver16)

#### Restoring a database

Following steps if taken correctly, can restore a database.

1. Launch SQL Server Management Studio (SSMS) and connect to your SQL Server instance.
2. Right-click the Databases node in Object Explorer and select Restore Database....

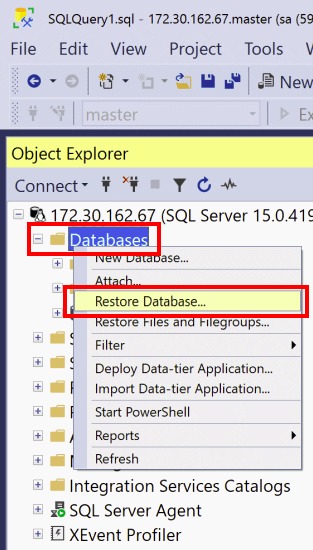


Fig. 2. Restoration of a database using SSMS

1. Select Device:, and then select the ellipses (...) to locate your backup file.
2. Select Add and navigate to where your .bak file is located. Select the .bak file and then select OK.
3. Select OK to close the Select backup devices dialog box.
4. Select OK to restore the backup of your database.

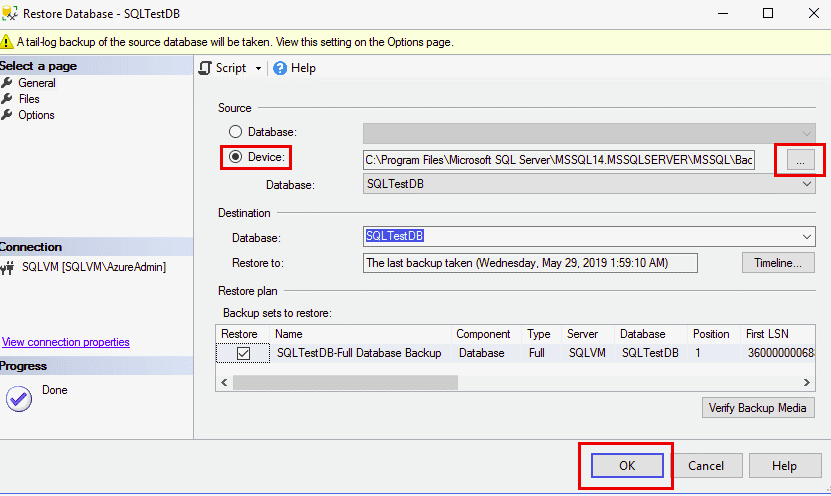


Fig. 3. Restoration of a database using SSMS - II

Following SQL commands restore a database as well:

*USE [master];*

*GO*

*RESTORE DATABASE [SQLTestDB]*

*FROM DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\Backup\SQLTestDB.bak' WITH FILE = 1, NOUNLOAD, STATS = 5;*

*GO*

To clean up the resources (database alongwith its back up history):

*EXEC msdb.dbo.sp\_delete\_database\_backuphistory @database\_name = N'SQLTestDB'*

*GO*

*USE [master];*

*GO*

*DROP DATABASE [SQLTestDB];*

*GO*

### Transactions

A transaction is the logical work unit that performs a single activity or multiple activities in a database. Transactions may consist of a single read, write, delete, or update operations or a combination of these. Suppose that, when we want to withdraw money from the ATM, the ATM application will achieve this operation in three steps. As a first step, the application will check the balance of the account, and then it will deduct the money from the source account. Along with these two processes, it will keep the log of this money withdrawing activity.

#### Modes of Transactions

SQL Server can operate 3 different transactions modes and these are:

1. **Autocommit Transaction** mode is the default transaction for the SQL Server. In this mode, each T-SQL statement is evaluated as a transaction and they are committed or rolled back according to their results. The successful statements are committed and the failed statements are rolled back immediately
2. **Implicit transaction** mode enables to SQL Server to start an implicit transaction for every DML statement but we need to use the commit or rolled back commands explicitly at the end of the statements
3. **Explicit transaction** mode provides to define a transaction exactly with the starting and ending points of the transaction

Defining an Implicit Transaction

To define an implicit transaction, we need to enable the IMPLICIT\_TRANSACTIONS option. The following query illustrates an example of an implicit transaction.

@@TRANCOUNT function returns the number of BEGIN TRANSACTION statements in the current session and we can use this function to count the open local transaction numbers in the examples

*SET IMPLICIT\_TRANSACTIONS ON*

*UPDATE*

*university*

*SET*

*country\_id = 74*

*WHERE*

*id = 1*

*SELECT*

*IIF(@@OPTIONS & 2 = 2,*

*'Implicit Transaction Mode ON',*

*'Implicit Transaction Mode OFF'*

*) AS 'Transaction Mode'*

*SELECT*

*@@TRANCOUNT AS OpenTransactions*

*COMMIT TRAN*

*SELECT*

*@@TRANCOUNT AS OpenTransactions*

Defining an Explicit Transaction

To define an explicit transaction, we start to use the BEGIN TRANSACTION command because this statement identifies the starting point of the explicit transaction. It has the following syntax:

*BEGIN TRANSACTION [ {transaction\_name | @tran\_name\_variable }*

*[WITH MARK ['description']]]*

**transaction\_name**option is used to assign a specific name to transactions

**@trans\_var**optionis a user-defined variable that is used to hold the transaction name

**WITH MARK**option enable to mark a particular transaction in the log file

After defining an explicit transaction through the BEGIN TRANSACTION command, the related resources acquired a lock depending on the isolation level of the transaction. For this reason as possible to use the shortest transaction will help to reduce lock issues. The following statement starts a transaction and then it will change the name of a particular row in the Person table.

*BEGIN TRAN*

*UPDATE Person*

*SET    Lastname = 'Lucky',*

*Firstname = 'Luke'*

*WHERE  PersonID = 1*

*SELECT @@TRANCOUNT AS OpenTransactions*

*COMMIT TRAN*

*SELECT @@TRANCOUNT AS OpenTransactions*

#### Rolling back transactions

the ROLLBACK TRANSACTION statement helps in undoing all data modifications that are applied by the transaction. In the following example, we will change a particular row but this data modification will not persist.

*BEGIN TRAN*

*UPDATE Person*

*SET Lastname = 'Donald',*

*Firstname = 'Duck' WHERE PersonID=2*

*SELECT \* FROM Person WHERE PersonID=2*

*ROLLBACK TRAN*

*SELECT \* FROM Person WHERE PersonID=2*

The following table illustrates the structure of the explicit transactions in SQL Server.

|  |  |
| --- | --- |
| BEGIN TRANSACTION | The starting point of the transaction |
| SQL commands | DML and SELECT statements |
| COMMIT TRANSACTION or ROLLBACK TRANSACTION | Apply data changing to the database or Erase data changing to the database |

#### Save Points in Transanctions

Savepoints can be used to rollback any particular part of the transaction rather than the entire transaction. So that we can only rollback any portion of the transaction where between after the save point and before the rollback command. To define a save point in a transaction we use the SAVE TRANSACTION syntax and then we add a name to the save point. Now, let’s illustrates an example of savepoint usage. When we execute the following query, only the insert statement will be committed and the delete statement will be rolled back.

*BEGIN TRANSACTION*

*INSERT INTO Person*

*VALUES('Mouse', 'Micky','500 South Buena Vista Street, Burbank','California',43)*

*SAVE TRANSACTION InsertStatement*

*DELETE Person WHERE PersonID=3*

*SELECT \* FROM Person*

*ROLLBACK TRANSACTION InsertStatement*

*COMMIT*

*SELECT \* FROM Person*

#### Auto Rollback Transactions

Generally, the transactions include more than one query. In this manner, if one of the SQL statements returns an error all modifications are erased, and the remaining statements are not executed. This process is called Auto Rollback Transaction in SQL. Now let’s explain this principle with a very simple example.

*BEGIN TRAN*

*INSERT INTO Person*

*VALUES('Bunny', 'Bugs','742 Evergreen Terrace','Springfield',54)*

*UPDATE Person SET Age='MiddleAge' WHERE PersonID=7*

*SELECT \* FROM Person*

*COMMIT TRAN*

In above transaction, the will be an error that would occur in the update statement due to the data type conversion issue. In this case, the inserted data is erased and the select statement did not execute.

#### Marked Transactions

SQL Server allows us to mark and add a description to a specific transaction in the log files. In this way, we can generate a recovery point that is independent of the time. Such as, when an accidental data modification occurs in the database and we don’t know the exact time of the data modification, the data recovery effort can be taken a long time. For this reason, marked transactions can be a useful solution to find out the exact time of the data modifications. In order to create a marked transaction, we need to give a name to the transaction and we also need to add WITH MARK syntax. In the following query, we will delete some rows and we will also mark the modifications in the log file.

*BEGIN TRAN DeletePerson WITH MARK 'MarkedTransactionDescription'*

*DELETE Person WHERE PersonID BETWEEN 3 AND 4*

*COMMIT TRAN DeletePerson*

The logmarkhistory table stores details about each marked transactions that have been committed and it is placed in the msdb database.

*SELECT \* FROM msdb.dbo.logmarkhistory*

#### 

As we can see in the above image the logmarkhistory gives all details about the marked transaction. The following two options help to use marked transactions as a recovery point.

1. STOPATMARK rolls forward to the mark and includes the marked transaction in the roll forward
2. STOPBEFOREMARK rolls forward to the mark and excludes the marked transaction from the roll forward

#### Task

Use superhero database. Create various implicit & explicit transactions that are a mix and match of select/update/delete/insert and other DML statements, use save points and work around committing and rolling back the changes. Also see how marked transactions can be helpful in your project.

**Rubric for Lab Assessment**

|  |  |  |  |
| --- | --- | --- | --- |
| **The student performance for the assigned task during the lab session was:** | | | |
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 |  |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 |  |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 |  |
| Worst | The student did not complete assigned tasks. | 1 |  |

**Instructor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**