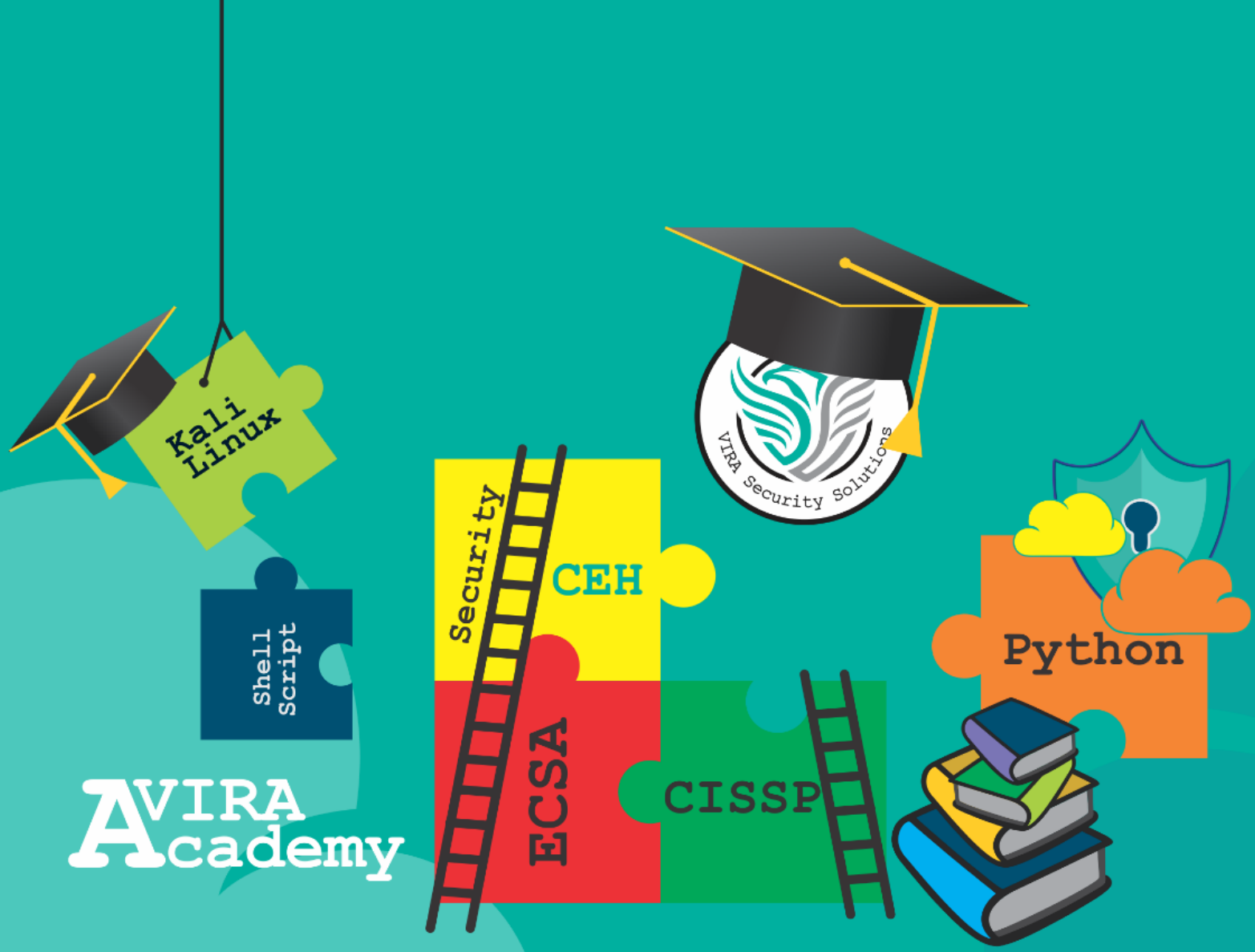

VIRA
Security
Solutions


راهکارهای امن ویرا
شماره ثبت: ۵۴۰۱۳۹
VIRA Security Solutions


Kali Linux
Shell Script
VIRA Academy
Security
CEH
ECSA
CISSP
VIRA Security Solutions
Python

# DBA

## SQL Server Administration

Part7


Mohammad Reza Gerami
mrgerami@aut.ac.ir
gerami@virasec.ir




# Index
# Cluster
# Non-Cluster

## Database Backup Strategies


**Two leading questions direct the restore strategy:**

What is the maximum amount of data can be tolerated?
Involves taking multiple transaction logs backup.

What is the accepted time frame for the database to be restored in case of a disaster?
Involves the use of high availability solutions.

To answer these questions you must ask the **why, when, where an who** for the strategy.

## Database Backup Strategies




Why Back-up?

Bakups are taken in order to be able to restore the database to its original state just before the disaster and protect against the following reasons:

- Hardware failure
- Malicious damage
- User-error
- Database corruption

## Database Backup Strategies




**What Back-up?**

Simply put, it's an image of the database at the time of the full backup.

## Database Backup Strategies


A **full database backup** backs up the whole database. It includes some part of the transactional log so that you could restore your database to the point when the full backup was finished. Usually, files with full database backup have '.bak' extension. It is recommended to periodically create full backups, but since it contains transaction log along with whole database data it takes significant space. To create full backup use the following command:

```
BACKUP DATABASE AdventureWorks TO DISK = 'C:\AdventureWorks.BAK' GO
```

## Database Backup Strategies


If your database is big enough, it may become quite space-consuming to create full backups each time. Here a differential backup comes to the rescue. This kind of database backup is related to the last full backup and contains all changes that have been made since the last full backup. Each next differential backup contains the same data that was stored in the previous one, but none of them contain transactional logs. You need a previous full database backup to restore a differential backup. The file extension of a differential backup is usually '.dif'.

```
BACKUP DATABASE Adventureworks TO DISK = 'adventureworks.dif' WITH DIFFERENTIAL
```

For relatively small and rarely changing databases a full backup is often sufficient. Otherwise, you need to look at other backup types.

# Database Backup Strategies


If it is crucial to restore your database close to the point of a failure or at any other point in time then you need to consider transactional log backups. This SQL Server backup type is possible only with full or bulk-logged recovery models. A transaction log backup contains all log records that have not been included in the last transaction log backup (or the last full backup). To create a transaction log backup to your database use the following command:

```
BACKUP LOG Adventureworks TO DISK = 'adventureworks.trn'
```

## Database Backup Strategies


Use COPY_ONLY option if you need to make an additional full or transaction log backups which will occur beyond the regular sequence of SQL Server backups. To perform copy-only backup simply add "COPY_ONLY" clause:

```
BACKUP DATABASE Adventureworks TO DISK = 'full.bak' WITH COPY_ONLY
```

# Database Backup Strategies


## File and Filegroup Backups



These backup types allow you to backup one or more database files or filegroups. To execute file backup use the following command:

```
BACKUP DATABASE Adventureworks
FILE = 'File'
TO DISK = 'File.bck‘
```

Use this command to perform filegroup backup:

```
BACKUP DATABASE Adventureworks
FILEGROUP = 'Group'
TO DISK = 'Group.bck'
```

# Database Backup Strategies

## Partial Database Backup

Typically partial backups are used in simple recovery model to make backups of very large databases that have one or more read-only filegroups. However, SQL Server also allows making partial backups with full or bulk-logged recovery models. Use the following T-SQL command to create a partial backup:

```
BACKUP DATABASE Adventureworks READ_WRITE_FILEGROUPS TO
DISK = 'partial_backup.bak'
```

# Database Backup Strategies


## Overwrite or append backup sets

Another option is the WITH INIT and WITH NOINIT options. By default, the NOINIT option is enabled. It means that the backup will append to other backups in the file. For example, if you already have a full and a differential backup in full.bak, they will all remain after executing:

```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak' WITH NOINIT
```

On the other hand, if you want to overwrite existing backups, the WITH INIT option will erase the previous set of backups:

```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak' WITH INIT
```

# Database Backup Strategies


## Set expiration dates for backups

If you want your backup to expire, you can use the WITH EXPIREDATE option. The following example shows how to back up with an expiration date on March 28, 2018:

```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak' WITH EXPIREDATE =
N'03/28/2018 00:00:00'T
```

Another option is the option to expire after a specified number of days. The following example, shows how to retain a backup for 3 days:

```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak' WITH
RETAINDAYS = 3
```

# Database Backup Strategies


## Encrypt a database backup

Another interesting option is the backup with encryption, this option will allow encrypting our backup. To do that, we will need to create a master key:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '2455678KL95234nl0zBe'
```

Next, we will need to create a certificate:

```
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'my certificate'
```

```
-- export the backup certificate to a file
BACKUP CERTIFICATE States2 TO FILE = 'Q:\States2Cert.cert'
WITH PRIVATE KEY (
FILE = 'Q:\States2Cert.key',
ENCRYPTION BY PASSWORD = 'Api1401@2015!!')
```

## Database Backup Strategies


Finally, we can backup our database:

```
BACKUP DATABASE [testdb] TO DISK = 'c:\backup\full.bak'
WITH FORMAT,
ENCRYPTION
(
 ALGORITHM = AES_256,
 SERVER CERTIFICATE = MyServerCert ),
 STATS = 10
GO
```


## Backup compression

In addition, it is possible to compress your backup using the option WITH COMPRESSION. This option will compress your backup:

```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak' WITH FORMAT, COMPRESSION
```

Note that the backup compression is only supported on SQL Server Enterprise and Standard editions (and the Business Intelligence Edition in SQL Server 2012).

# Database Backup Strategies


## SQL Server Backup Tips



If your database is big, you will need to combine full, differential and transaction log backups. If your database is big and it does not change too much, a differential backup will take less space than a full backup and you will save a lot of space.

Do not store your backup on the same drive than the database. If possible try to store your backup on another Server or even better in another physical place. If your hard drive fails and you have the database and backups, you may not be able to recover your data.

Test your backups to make sure that they are working fine.  There are some nice options that can be useful to verify that the backup is OK, like the verify only option. The following example will create a full backup and then test if it works using the RESTORE WITH VERIFYONLY option to just verify if the backup is OK:

## SQL Server Backup Tips




```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak'
GO

declare @backupSetId as int

select @backupSetId = position from msdb..backupset
where database_name=N'adventureworks' and backup_set_id=(select max(backup_set_id)
from msdb..backupset
where database_name=N'adventureworks' )

if @backupSetId is null
begin raiserror (N'Verify failed. Backup information for database "adventureworks" not found.' , 16, 1)
end

RESTORE VERIFYONLY FROM DISK = 'c:\backup\full.bak' WITH FILE = @backupsetid
```

## SQL Server Backup Tips




```
BACKUP DATABASE Adventureworks TO DISK = 'c:\backup\full.bak'
GO

declare @backupSetId as int

select @backupSetId = position from msdb..backupset
where database_name=N'adventureworks' and backup_set_id=(select max(backup_set_id)
from msdb..backupset
where database_name=N'adventureworks' )

if @backupSetId is null
begin raiserror (N'Verify failed. Backup information for database "adventureworks" not found.' , 16, 1)
end

RESTORE VERIFYONLY FROM DISK = 'c:\backup\full.bak' WITH FILE = @backupsetid
```




# Restore Database Backup

# Restore Database Backup




This command enables you to perform the following restore scenarios:

- ✓ Restore an entire database from a full database backup (a complete restore).
- ✓ Restore part of a database (a partial restore).
- ✓ Restore specific files or filegroups to a database (a file restore).
- ✓ Restore specific pages to a database (a page restore).
- ✓ Restore a transaction log onto a database (a transaction log restore).
- ✓ Revert a database to the point in time captured by a database snapshot.

# Restore Database Backup


## About Restore Scenarios



SQL Server supports a variety of restore scenarios:

Complete database restore

Restores the entire database, beginning with a full database backup, which may be followed by restoring a differential database backup (and log backups).

File restore

Restores a file or filegroup in a multi-filegroup database. Note that under the simple recovery model, the file must belong to a read-only filegroup. After a full file restore, a differential file backup can be restored.

Page restore

Restores individual pages. Page restore is available only under the full and bulk-logged recovery models

Piecemeal restore

Restores the database in stages, beginning with the primary filegroup and one or more secondary filegroups. A piecemeal restore begins with a RESTORE DATABASE using the PARTIAL option and specifying one or more secondary filegroups to be restored.

# Restore Database Backup


## About Restore Scenarios



SQL Server supports a variety of restore scenarios:

Recovery only

Recovers data that is already consistent with the database and needs only to be made available.

Transaction log restore.

Under the full or bulk-logged recovery model, restoring log backups is required to reach the desired recovery point.

Online Restore

**Note**

Online restore is allowed only in Enterprise edition of SQL Server.

# Restore Database Backup

## About Restore Scenarios

### Discontinued RESTORE Keywords

The following keywords were discontinued in SQL Server 2008:

| DISCONTINUED RESTORE KEYWORDS | | |
|---|---|---|
| Discontinued keyword | Replaced by... | Example of replacement keyword |
| LOAD | RESTORE | RESTORE DATABASE |
| TRANSACTION | LOG | RESTORE LOG |
| DBO_ONLY | RESTRICTED_USER | RESTORE DATABASE ... WITH RESTRICTED_USER |

# Restore Database Backup

## About Restore Scenarios

### Comparison of RECOVERY and NORECOVERY

Rollback is controlled by the RESTORE statement through the [ RECOVERY | NORECOVERY ] options:

- NORECOVERY specifies that rollback doesn't occur. This allows rollforward to continue with the next statement in the sequence.

  In this case, the restore sequence can restore other backups and roll them forward.

- RECOVERY (the default) indicates that rollback should be performed after rollforward is completed for the current backup.

Recovering the database requires that the entire set of data being restored (the *rollforward set*) is consistent with the database. If the rollforward set has not been rolled forward far enough to be consistent with the database and RECOVERY is specified, the Database Engine issues an error.

# Restore Database Backup


### Restoring a full database

The following example restores a full database backup from the AdventureWorksBackups logical backup device.

## SQL

```
RESTORE DATABASE AdventureWorks FROM AdventureWorksBackups;
```

### Note

For a database using the full or bulk-logged recovery model, SQL Server requires in most cases that you back up the tail of the log before restoring the database.

# Restore Database Backup


## Restoring full and differential database backups

The following example restores a full database backup followed by a differential backup from the d:\db\AdventureWorks.bak backup device, which contains both backups. The full database backup to be restored is the sixth backup set on the device (FILE = 6), and the differential database backup is the ninth backup set on the device (FILE = 9). As soon as the differential backup is recovered, the database is recovered.

## SQL

```
RESTORE DATABASE AdventureWorks FROM DISK = ‘d:\db\AdventureWorks.bak' WITH FILE = 6
NORECOVERY; RESTORE DATABASE AdventureWorks FROM DISK = ‘d:\db\AdventureWorks.bak' WITH
FILE = 9 RECOVERY;
```

# Restore Database Backup


## Restoring a database using RESTART syntax

The following example uses the RESTART option to restart a RESTORE operation interrupted by a server power failure.

**SQL**

```
-- This database RESTORE halted prematurely due to power failure. RESTORE DATABASE
AdventureWorks FROM AdventureWorksBackups; -- Here is the RESTORE RESTART operation.
RESTORE DATABASE AdventureWorks FROM AdventureWorksBackups WITH RESTART;
```

# Restore Database Backup


## Restoring a database and move files

The following example restores a full database and transaction log and moves the restored database into the D:\DB\ directory.

### SQL

```
USE [master]
BACKUP LOG [bkdb] TO  DISK = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\Backup\bkdb_LogBackup_1399-04-17_13-05-35.bak' WITH NOFORMAT, NOINIT,  NAME
= N'bkdb_LogBackup_1399-04-17_13-05-35', NOSKIP, NOREWIND, NOUNLOAD,  NORECOVERY ,  STATS = 5
RESTORE DATABASE [bkdb20002000] FROM  DISK = N'D:\DB\dbbackup.bak' WITH  FILE = 1,  MOVE N'bkdb' TO
N'D:\DB\bkdb20002000.mdf',  MOVE N'bkdb_log' TO N'D:\DB\bkdb20002000_log.ldf',  NORECOVERY,  NOUNLOAD,
REPLACE,  STATS = 5
RESTORE DATABASE [bkdb20002000] FROM  DISK = N'D:\DB\dbbackup.bak' WITH  FILE = 7,  NORECOVERY,  NOUNLOAD,
STATS = 5
RESTORE LOG [bkdb20002000] FROM  DISK = N'D:\DB\dbbackup.bak' WITH  FILE = 8,  NOUNLOAD,  STATS = 5

GO
```

# Restore Database Backup


## Copying a database using BACKUP and RESTORE

The following example uses both the BACKUP and RESTORE statements to make a copy of the **AdventureWorks** database. The MOVE statement causes the data and log file to be restored to the specified locations. The RESTORE FILELISTONLY statement is used to determine the number and names of the files in the database being restored. The new copy of the database is named TestDB. For more information, see RESTORE FILELISTONLY.

SQL

```
BACKUP DATABASE AdventureWorks TO AdventureWorksBackups ; RESTORE FILELISTONLY FROM AdventureWorksBackups ; RESTORE DATABASE TestDB FROM AdventureWorksBackups WITH MOVE 'AdventureWorks_Data' TO 'C:\MySQLServer\testdb.mdf', MOVE 'AdventureWorks_Log' TO 'C:\MySQLServer\testdb.ldf'; GO
```

# Restore Database Backup


## Restoring to a point-in-time using STOPAT

The following example restores a database to its state as of 12:00 AM on April 15, 2020 and shows a restore operation that involves multiple log backups. On the backup device, AdventureWorksBackups, the full database backup to be restored is the third backup set on the device (FILE = 3), the first log backup is the fourth backup set (FILE = 4), and the second log backup is the fifth backup set (FILE = 5).

```
RESTORE DATABASE AdventureWorks FROM AdventureWorksBackups WITH FILE=3, NORECOVERY;

RESTORE LOG AdventureWorks FROM AdventureWorksBackups WITH FILE=4, NORECOVERY, STOPAT
= 'Apr 15, 2020 12:00 AM';

RESTORE LOG AdventureWorks FROM AdventureWorksBackups WITH FILE=5, NORECOVERY, STOPAT
= 'Apr 15, 2020 12:00 AM'; RESTORE DATABASE AdventureWorks WITH RECOVERY;
```

# Restore Database Backup

## Restoring using TAPE syntax


The following example restores a full database backup from a TAPE backup device

```
RESTORE DATABASE AdventureWorks FROM TAPE = '\\.\tape0';
```

# Restore Database Backup


## Restoring using FILE and FILEGROUP syntax

The following example restores a database named MyDatabase that has two files, one secondary filegroup, and one transaction log. The database uses the full recovery model.

The database backup is the ninth backup set in the media set on a logical backup device named MyDatabaseBackups. Next, three log backups, which are in the next three backup sets (10, 11, and 12) on the MyDatabaseBackups device, are restored by using WITH NORECOVERY. After restoring the last log backup, the database is recovered.

# Restore Database Backup

## Restoring using FILE and FILEGROUP syntax


In the RESTORE DATABASE, notice that there are two types of FILE options. The FILE options preceding the backup device name specify the logical file names of the database files that are to be restored from the backup set; for example, FILE = 'MyDatabase_data_1'. This backup set is not the first database backup in the media set; therefore, its position in the media set is indicated by using the FILE option in the WITH clause, FILE=9

# Restore Database Backup

## Restoring using FILE and FILEGROUP syntax

```
RESTORE DATABASE MyDatabase FILE = 'MyDatabase_data_1', FILE = 'MyDatabase_data_2',
FILEGROUP = 'new_customers' FROM MyDatabaseBackups WITH FILE = 9, NORECOVERY;
GO

-- Restore the log backups
 RESTORE LOG MyDatabase FROM MyDatabaseBackups WITH FILE = 10, NORECOVERY;
GO

RESTORE LOG MyDatabase FROM MyDatabaseBackups WITH FILE = 11, NORECOVERY; GO RESTORE
LOG MyDatabase FROM MyDatabaseBackups WITH FILE = 12, NORECOVERY;
GO

 --Recover the database
 RESTORE DATABASE MyDatabase WITH RECOVERY; GO
```