In this lab class we will approach the following topics:

1.1. Transaction Logs and Recovery in SQL Server

Every SQL Server database has a **transaction log** that records all transactions and the database modifications made by each transaction. The transaction log is a critical component of the database. If there is a system failure, the transaction log might be required to bring your database back to a consistent state. The transaction log supports the following operations:

- Recovery of individual transactions in the case of a transaction rollback (**undoing**)
- Recovery of all incomplete transactions when SQL Server is started (**restart recovery**)
- Rolling a restored database, file, filegroup, or page forward to the point of failure (**restore recovery**)
- Supporting transactional replication in multiple copies of the same database.

If a computer that is running SQL Server fails, the databases may be left in a state where some modifications were never written from the buffer cache to the data files, and there may be some modifications from incomplete transactions in the data files. When an instance of SQL Server is started, it runs a recovery process of each database. Every modification recorded in the log, which may not have been written to the data files, is rolled forward. Every incomplete transaction found in the transaction log is then rolled back to make sure the integrity of the database is preserved.

The contents of the transaction log for an SQL Server database can be examined through the following command:

```
DBCC log(AdventureWorks2017, 0);
```

The first parameter refers to the identifier or name of the database in question. The second parameter refers to the type of output that is being requested:

0 – minimum information (operation, context, transaction id)
1 – more information (plus flags, tags, row length)

2 – detailed information (plus object name, index name, page id, slot id)
3 – full information about each operation
4 – full information about each operation plus hexadecimal dump of the current transaction log's row.

Alternatively to the DBCC log command, one can use the following SELECT instruction, where the parameters to the *fn_dblog* function refer to the start and end LSNs.

```
SELECT TOP 10 * FROM fn_dblog(NULL, NULL);
```

The SQL Server Transaction Log architecture is described in detail in the online documentation.[1]

1.2. WAL and the ARIES-like logging and recovery scheme

SQL Server, like most database systems, uses an ARIES-like logging and recovery scheme to recover from failures and guarantee transactional consistency. ARIES relies on the *Write-Ahead Logging (WAL) protocol*, which requires that log records be durably written prior to the corresponding data changes. Database systems use the write-through capability of the storage media to enforce write-ahead logging. The write-through capability enables write commands to pass through the storage media cache.

To clarify the WAL protocol, consider the following specific example (assume that there is no index and the page affected is page 150).

```
BEGIN TRANSACTION;
INSERT INTO tblTest VALUES (1);
COMMIT TRANSACTION;
```

We can break the activity down into simplistic logging steps:

Statement	Actions Performed
BEGIN TRANSACTION	A BEGIN_XACT log record is written to the log cache area, but there is no need to flush to stable storage because the SQL Server has not made any physical changes.
INSERT INTO tblTest	1. Data page 150 is retrieved from disk into SQL Server data cache, if not already available. 2. The page is marked dirty, and appropriate locks are obtained. 3. An *Insert* log record is built and added to the log cache. 4. A new row is added to the data page.

[1] https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide

	5. The log records associated with the transaction or page do not need to be flushed at this point because all changes remain in volatile storage.
COMMIT TRANSACTION	1. A *commit* log record is formed, and the log records associated with the transaction must be written to stable storage. The transaction is not considered committed until the log records are correctly assigned to stable storage. 2. Data page 150 remains in SQL Server data cache and is not immediately flushed to stable storage. When the log records are properly stored, recovery can redo the operation if necessary. 3. Locks are released.

The cache can hold multiple log and data write requests at the same time. The WAL protocol and the SQL Server implementation of the WAL protocol require flushing of the log writes to stable storage before the page write can be issued. However, the use of the cache offered by the disk hardware might return success from a log write request without the data being written to the actual drive (that is, written to stable storage). This may lead to SQL Server issuing the data page write request. SQL Server makes every effort to ensure the WAL, but testing has shown that many disk drive configurations may contain write caching without proper battery backup.

1.3. SQL Server Restore Recovery Models

Backing up/restoring databases and transaction logs are mechanisms that SQL Server supports to protect from data loss. The backup operation creates a copy of a database. This copy can be used to restore the database if the media failure occurs or if the database is somehow damaged (for example, from the user errors).

SQL Server supports the following kinds of backup:

- **Full database backup** (it is a full copy of the database).
- **Transaction log backup** (it contains the copy of the transaction log only).
- **Differential backup** (it contains the copy of only the database pages modified after the last full database backup).

The following T-SQL instructions exemplify database backups and restores. For instructions on how to perform database backups and restores using the SQL Server Management Studio, please refer to the following tutorials:

- Back up a database[2]
- Restore a database backup[3]

[2] https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/create-a-full-database-backup-sql-server?view=sql-server-2017

[3] https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-a-database-backup-using-ssms?view=sql-server-2017

```
--initial backup
BACKUP DATABASE db_name TO db_name_back_db WITH INIT;

--differential backup
BACKUP DATABASE db_name TO db_name _db WITH DIFFERENTIAL;

-- transactional log backup
BACKUP LOG db_name TO db_name _back_log WITH INIT;

-- restore database
RESTORE DATABASE db_name FROM db_name _db WITH NORECOVERY;
```

The behavior of Restore Recovery is based on a property of the Database that is the Recovery Model. **SQL Server offers three recovery models for each database:** simple, full, and bulk-logged recovery. These models determine how the data is backed up and what the exposure to data loss is. When creating a database, the default value of this option is **Full**.

- **Simple** - The simple recovery model allows to **recover data only to the most recent full database or differential backup**. With this recovery model, it is not possible to restore the database to the point of failure or to a specific point in time. Changes since the last backup will be lost. Transaction log backups are not available because the contents of the transaction log are truncated each time a checkpoint is issued for the database.

- **Full** - The full recovery model **uses database backups and transaction log backups to provide complete protection against failure**. Along with being able to restore a full or differential backup, it is possible to recover the database to the point of failure or to a specific point in time. All operations, including bulk operations such as SELECT INTO, CREATE INDEX and bulk-loading data, are fully logged and recoverable.

- **Bulk-Logged** - The bulk-logged recovery model provides **protection against failure combined with the best performance**. In order to get better performance, the following operations are minimally logged and not fully recoverable: SELECT INTO, bulk-load operations, CREATE INDEX as well as text and image operations. Under the bulk-logged recovery model, a damaged data file can result in having to redo work manually based on the operations that are not fully logged. In addition, the bulk-logged recovery model only allows the database to be recovered to the end of a transaction log backup when the log backup contains bulk changes.

Using SQL Server Management Studio, recovery models can be changed by right clicking on the database name, selecting Properties, selecting the Options tab and selecting recovery model from the drop-down list. Selecting OK will change the recovery model immediately.

Alternatively, one can use the following T-SQL command:

```
ALTER DATABASE dbName SET RECOVERY recoveryMode
```

If the log records are not deleted with a proper frequency, the transaction log file will grow in high volume. Therefore, inactive transaction records should be deleted from time to time.

Deleting all inactive transaction records from a transaction log file is called as **Log Truncation**. LOG truncates occurs automatically upon the following conditions:

- For a **simple recovery** model database, truncates occur **after a checkpoint.**
- For the **FULL and Bulk-logged** model, truncates occur **after Transaction Log backups**, **or if a checkpoint occurs after the previous backup**.

An overview on the restore and recovery in mechanisms in SQL Server is given in the online documentation.[4]

1.4. Disaster Recovery Plans in SQL Server

Concerning disaster recovery (DR), the following strategies are available in SQL Server:

- Mirrored backup media sets
- Backup and restore
- Log shipping
- Database mirroring
- Replication

Mirrored tape backups are very useful when you have **large databases with point in time recovery requirements, have a high bandwidth link to your disaster recovery site and identical tape devices in all locations**. You can simultaneously write your backup to up to four tape devices. This means you could backup your database locally to your multiplex tape unit, while at the same time write your backup to your DR site. The slowest link will throttle your overall throughput of your backup solution, so the WAN link to your DR site may slow down the speed of your local backup, if you are using mirrored tape backups. All of your backup devices must use identical hardware. Mirrored backup media sets can provide complete point in time recovery and can be used with very large databases.

If your **disaster recovery goals are not time critical** (you don't need point in time recovery), a disaster recovery plan using a **backup and restore of a full, differential, or partial database backup can be the optimal choice**. For smaller databases, a full or differential backup may satisfy all of your DR goals. However, if your databases get larger you may need to incorporate a partial backup strategy. This would involve backing up your read-only filegroups on a different frequency than your read-write file groups. You will need the last backup of the read-only file group, the last backup of the read-write file group(s), and the log backup chain since the last backup of the read-write file group. SQL Server also has the ability to continue to backup or restore a database transaction log or file group after a failure of the database media or the backup media. This means that if your database files get damaged you may still be able to backup the tail of the log, or if

[4] https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-and-recovery-overview-sql-server?view=sql-server-2017

your database backup is damaged you will be able to continue the restore after the error has been detected.

If your DR plan requires **point in time recovery**, **log shipping** may be your best option. This is generally used for small numbers of databases or small databases, and when you do not have real time synchronization requirements and can tolerate some exposure to data loss. Log shipping is a continuous backup and restore. However, log shipping does have significant exposure to data loss. At best, this exposure is the shortest length of time required to complete your transaction log dump and copy it over to the standby server. Use log shipping when you have point in time recovery requirements and you have a small number of databases you need to log ship. Log shipping is not scalable to large databases or a large number of databases because the bandwidth requirements are large and log shipping can become complex when you are log shipping large numbers of databases. To use the **log shipping wizard in SQL Server**, right click on your database and select Tasks and Ship Transaction Logs. A tutorial on log shipping can be found online.[5]

Replication will replicate all or selected transactions which occur on your source server to your destination server (DR site). Replication can offer **lower latencies** and can provide **lower exposure to data loss than log shipping or a backup and restore DR solution**. It can replicate a subset of your data and can modify it to have different values or a different schema on the DR site. It can also modify the data as it is replicated to the DR site. SQL Server will replicate most DDL (Data Definition Language) transparently, and the peer-to-peer replication model allows for a configuration of a DR node in a write-anywhere topology. However, it is not really resilient to conflicts or schema changes. Microsoft recommends that all schema changes and DML occur on one node in the topology. Replication generally offers very low latencies, however, under some database loads the latency can exceed that of log shipping.

Database mirroring is continuous log shipping and can have very low latencies. It has two modes high safety (essentially a split write) and high performance, which is an asynchronous commit operation on the DR site. High safety does have larger latencies than high performance, but your exposure to data loss is extremely low, if at all. To make database mirroring in high safety mode work optimally, Microsoft recommends you have a very good network in place, have long running transactions and have many connections to the database. Database mirroring can also be configured for automatic failover. Database mirroring does consume host-based resources and is not scalable to large numbers of databases on a single server. It can only be used in full-recovery model.

[5] http://www.sql-server-performance.com/2011/sql-server-log-shipping-for-large-databases/

2. Experiments and Exercises

2.1. Some Examples on Database Backups

We will now see some examples on how the management of transaction logs can be performed through T-SQL commands. The examples use the following code to create an EmployeeDB database on a local instance of SQL Server:

```
USE master;
DROP DATABASE IF EXISTS EmployeeDB;
CREATE DATABASE EmployeeDB
ON (NAME=EmployeeDB_dat, FILENAME='C:\Temp\EmployeeDb.mdf')
LOG ON (NAME=EmployeeDB_log, FILENAME='C:\Temp\EmployeeDb.ldf');
```

The following code can be used to retrieve data from the AdventureWorks database and create an Employees table:

```
USE EmployeeDB;
DROP TABLE IF EXISTS dbo.Employees;
SELECT * INTO dbo.Employees
FROM AdventureWorks2017.HumanResources.Employee;
```

It is now possible to switch the recovery model on a database by running an ALTER DATABASE statement and specifying the SET RECOVERY clause, as shown in the following example:

```
USE master;
ALTER DATABASE EmployeeDB SET RECOVERY FULL;
```

One can use the *sys.database_files* catalog view to retrieve data about the log file associated with the EmployeeDB database:

```
USE EmployeeDB;
SELECT name, size, max_size, growth, is_percent_growth
FROM sys.database_files
WHERE type_desc = 'LOG';
```

Alternatively, one can use the DBCC SQLPERF statement to return information about the transaction logs for each database in a SQL Server instance. To retrieve log data, you must specify the LOGSPACE keyword in parentheses, as shown in the following example:

```
DBCC SQLPERF(LOGSPACE);
```

The exact contents of the transaction log file can be inspected with the DBCC log command:

```
DBCC log(EmployeeDB, 0);
```

Before a log file can be backed up, a full database backup must be performed:

```
BACKUP DATABASE EmployeeDB TO DISK = 'C:\Temp\EmployeeDB_dat.bak';
```

After the database backup, one can make the following data modification statements so the log would contain transactions not included in the backup:

```
USE EmployeeDB;
UPDATE Employees SET JobTitle = 'To be determined';
DELETE Employees WHERE BusinessEntityID > 5;
```

The database can be restored to the state it was previously to the data modifications (you may need to close all the other query windows):

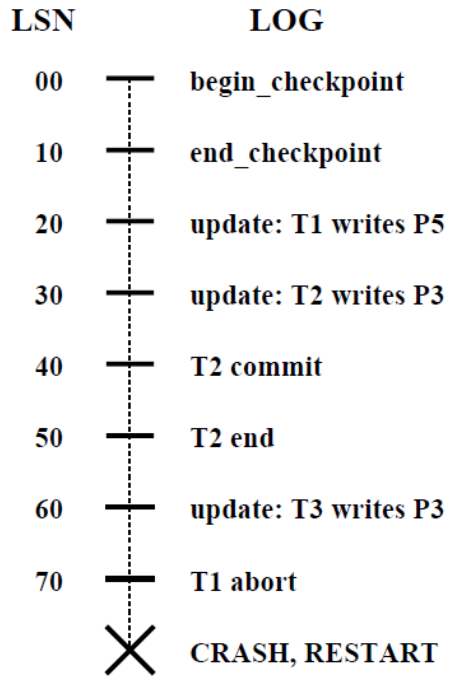
```
USE master;
ALTER DATABASE EmployeeDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
RESTORE DATABASE EmployeeDB
FROM DISK = 'C:\Temp\EmployeeDB_dat.bak'
WITH NORECOVERY, REPLACE;
RESTORE LOG EmployeeDB;
```

After the database has been backed up, you can also back up the transaction log:

```
BACKUP LOG EmployeeDB TO DISK = 'C:\Temp\EmployeeDB_log.bak';
```

2.2. Exercises

2.1.1 - Consider the following recovery situation:

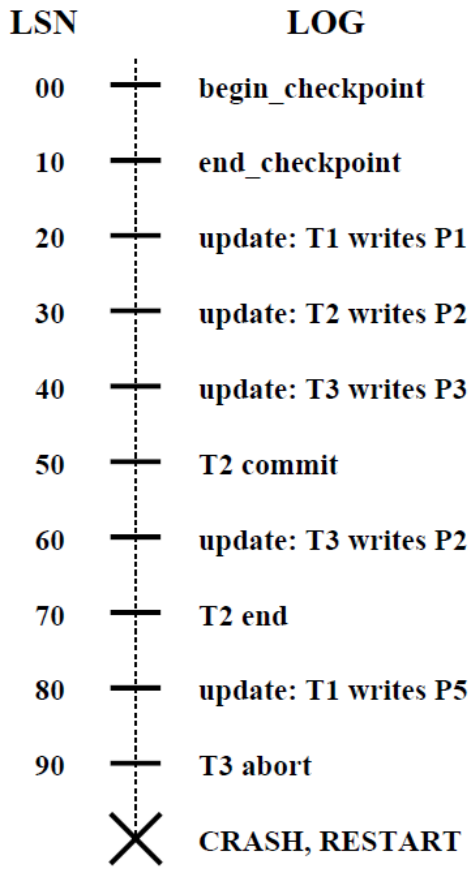


a) What is done during the Analysis phase? (Be precise about the points at which Analysis begins and ends and describe the contents of any tables constructed in this phase.)

b) What is done during the Redo phase? (Be precise about the points at which Redo begins and ends.)

c) What is done during the Undo phase? (Be precise about the points at which Undo begins and ends.)

2.2.2 - Consider the following recovery situation:



a) What is done during Analysis?

b) What is done during Redo?

c) What is done during Undo?

d) Show the events that are written to the log during the recovery process itself.