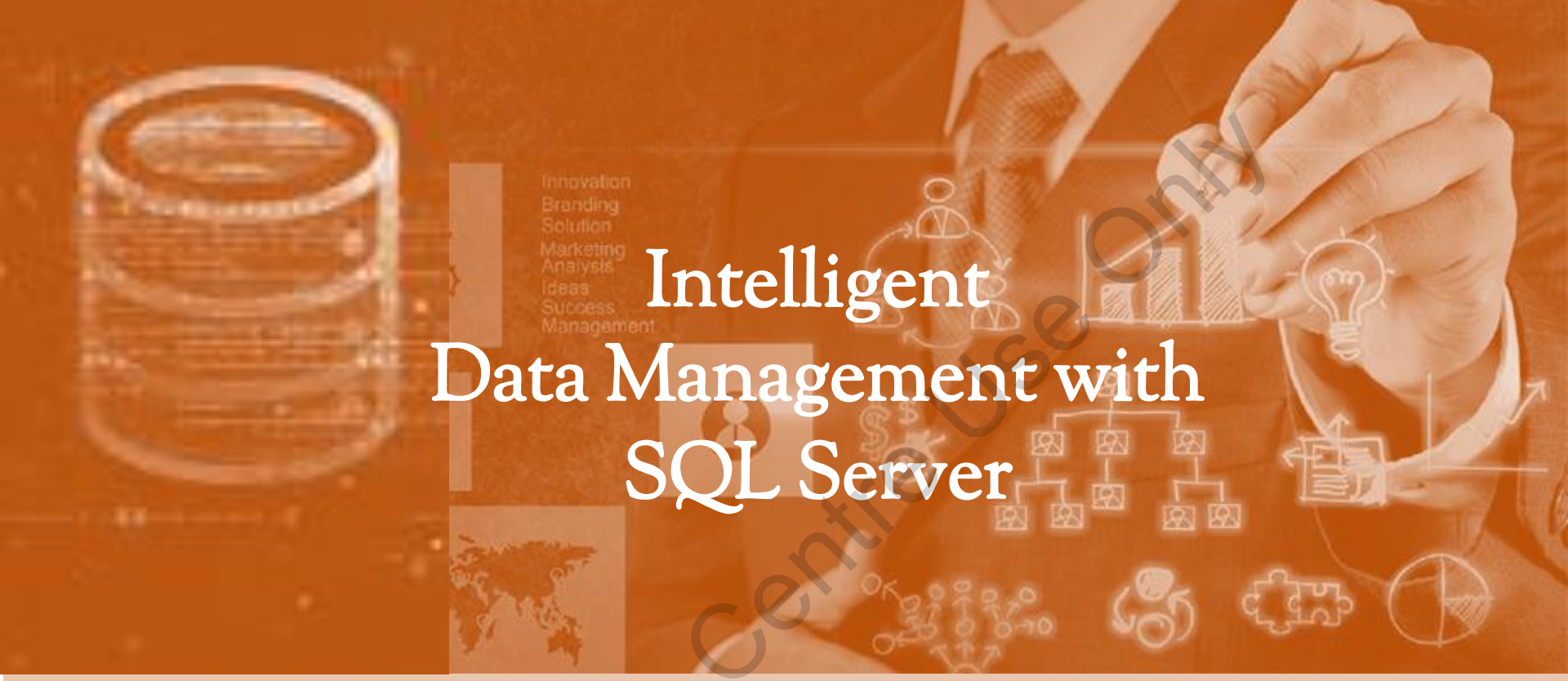


Intelligent Data Management with SQL Server

Session: 14

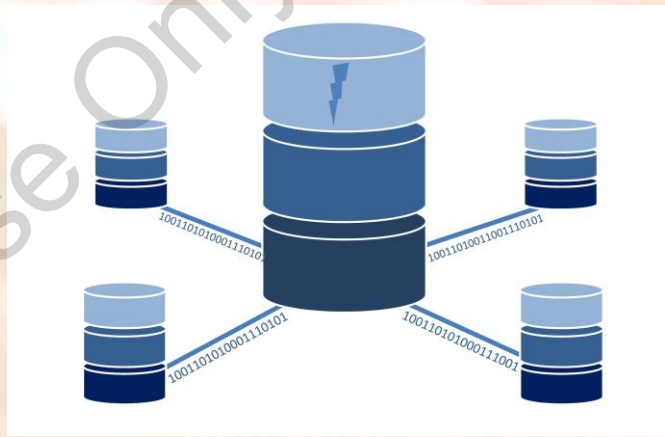
Transactions

Objectives

- Define and describe transactions
- Explain the procedure to implement transactions
- Explain the steps to mark a transaction
- Distinguish between implicit and explicit transactions
- Explain isolation levels
- Explain the scope and different types of locks
- Explain transaction management

Introduction

- A transaction is a single unit of work.
- A transaction is successful only when all data modifications that are made in a transaction are committed and are saved in the database permanently.
- If the transaction is rolled back or cancelled, then it means that the transaction has encountered errors and there are no changes made to the contents of the database.
- Hence, a transaction can be either committed or rolled back.



Need for Transactions 1-4

There are many circumstances where users are required to make many changes to the data in more than one database tables.

In many cases, the data will be inconsistent that executes the individual commands.

Suppose if the first statement executes correctly, but other statements fail then the data remains in an incorrect state.

For example,

A good scenario will be the funds transfer activity in a banking system. The transfer of funds will need an INSERT and two UPDATE statements.

Need for Transactions 2-4

Defining Transactions

- A logical unit of work must exhibit four properties, called the Atomicity, Consistency, Isolation, and Durability (ACID) properties, to qualify as a transaction.

- Atomicity: If the transaction has many operations then, all should be committed. If any of the operation in the group fails then, it should be rolled back.
- Consistency: The sequence of operations must be consistent.
- Isolation: Operations that are performed must be isolated from other operations on the same server or on the same database.
- Durability: The operations that are performed on the database must be saved and stored in the database permanently.

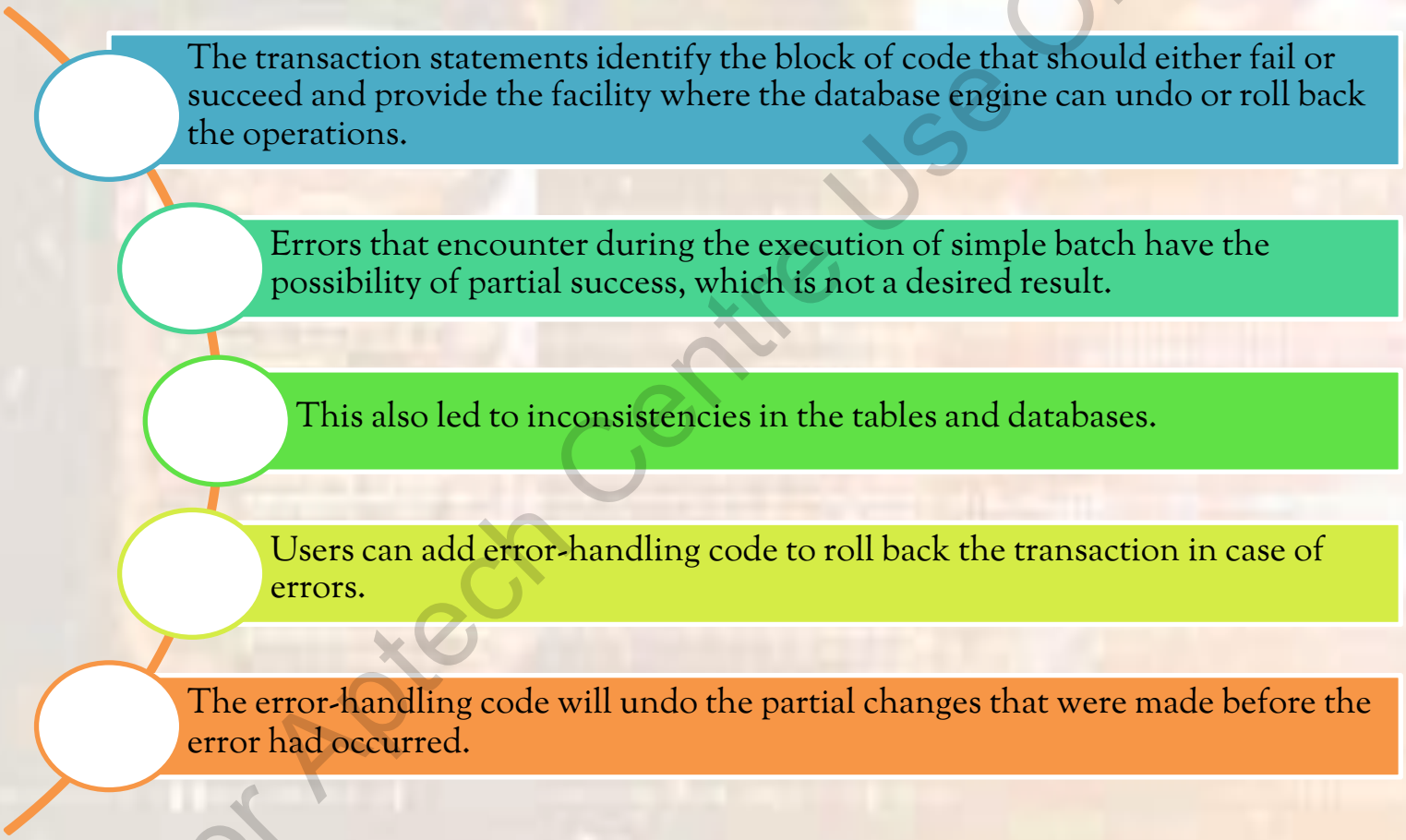
Need for Transactions 3-4

Implementing Transactions

- Autocommit Transactions: In this mode, one does not need to write any specific statements to start and end the transactions. It is the default mode for SQL Server Database Engine.
- Explicit Transactions: Each transaction explicitly starts with the BEGIN TRANSACTION statement and ends with a ROLLBACK or COMMIT transaction.
- Implicit Transactions: A new transaction is automatically started when the earlier transaction completes and every transaction is explicitly completed.
- Batch-scoped Transactions: These transactions are related to Multiple Active Result Sets (MARS).
- Distributed Transactions: It span two or more servers known as resource managers.

Need for Transactions 4-4

Transactions Extending Batches



Controlling Transactions

- Transactions can be controlled through applications by specifying the beginning and ending of a transaction.

- Transactions are managed at the connection level, by default.

- When a transaction is started on a connection, all Transact-SQL statements are executed on the same connection and are a part of the connection until the transaction ends.

Starting and Ending Transactions Using Transact-SQL 1-3

BEGIN TRANSACTION

- The BEGIN TRANSACTION statement marks the beginning point of an explicit or local transaction.

COMMIT TRANSACTION

- The COMMIT TRANSACTION statement marks an end of a successful implicit or explicit transaction.

COMMIT WORK

- The COMMIT WORK statement marks the end of a transaction.

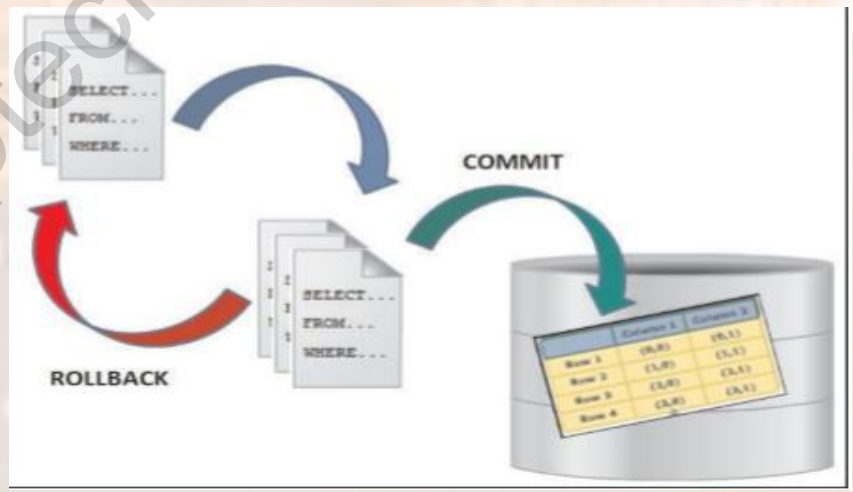
Starting and Ending Transactions Using Transact-SQL 2-3

ROLLBACK TRANSACTION

This transaction rolls back or cancels an implicit or explicit transaction to the starting point of the transaction or to a savepoint in a transaction.

It is used to delete all data modifications made from the beginning of the transaction or to a savepoint. It also releases the resources held by the transaction.

ROLLBACK WORK

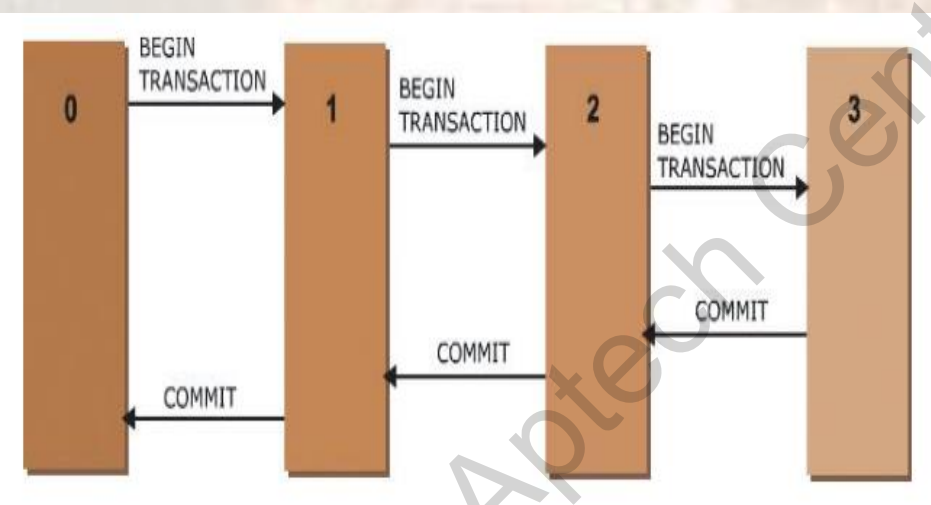


SAVE TRANSACTION

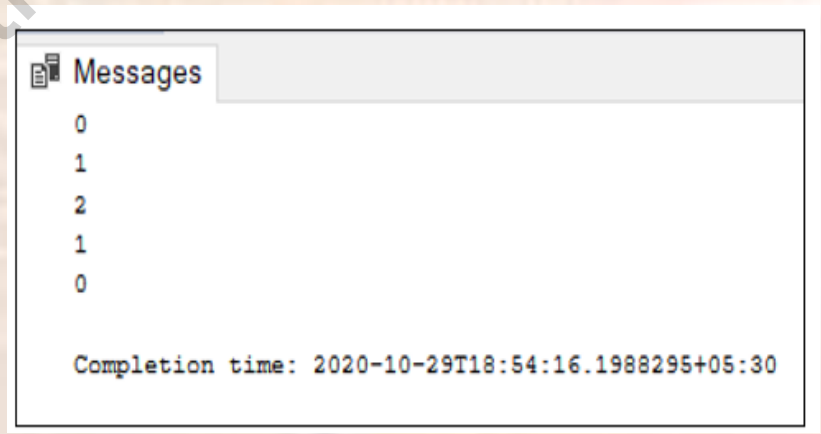
- The SAVE TRANSACTION statement sets a savepoint within a transaction.

The @@TRANCOUNT

- The @@TRANCOUNT system function returns a number of BEGIN TRANSACTION statements that occur in the current connection.



@@TRANCOUNT



Output

Marking a Transaction

Marking a transaction is useful only when the user is willing to lose recently committed transactions or is testing related databases.

Marking related transactions on a routine basis in every single related database creates a sequence of common recovery points in a database.

Concerns for Using Marked Transactions

- As the transaction mark consume log space, use them only for transactions that play an important role in the database recovery strategy.
- When the marked transaction is committed, then a row is inserted in the logmarkhistory table in msdb.
- If a marked transaction spans over multiple databases on different servers or on the same database server, the marks must be logged in the records of all affected databases.

Create Marked Transactions

For creating a marked transaction, users can use the BEGIN TRANSACTION statement and the WITH MARK [description] clause.

The transaction log records the mark description, name, user, database, datetime information, and the Log Sequence Number (LSN).

Steps to create a marked transaction in a set of databases:

- Name the transaction in the BEGIN TRAN statement and use the WITH MARK clause.
- Execute an update against all of the databases in the set.

Differences Between Implicit and Explicit Transactions

Implicit	Explicit
These transactions are maintained by SQL Server for each and every DML and DDL statements.	These transactions are defined by programmers.
These DML and DDL statements execute under the implicit transactions.	DML statements are included to execute as a unit.
SQL Server will roll back the entire statement.	`SELECT` Statements are not included as they do not modify data.

Implicit Vs. Explicit Transactions

Isolation Levels 1-2

- Transactions identify the isolation levels that define the degree to which one transaction must be isolated from the data modifications or resource that are made by other transactions.
- Isolation levels are defined in terms of which the concurrency side effects such as dirty reads are allowed.

Transaction isolation levels control the following:

- When data is read, are there any locks taken and what types of locks are requested?
- How much amount of time the read locks are held?
- If a read operation that is referencing a row modified by some other transaction is:
  - Blocking until the exclusive lock on the row is free.
  - Retrieving the committed version of the row that exists at the time when the transaction or statement started.
  - Reading the uncommitted data modification.

A transaction acquires an exclusive lock every time on each data that it modifies. Then, it holds that lock until the transaction is completed, irrespective of the isolation level that is set for that transaction.

Isolation Levels 2-2

Isolation Level	Dirty Read	NonRepeatable Read
Read committed	No	Yes
Read uncommitted	Yes	No
Snapshot	No	No
Repeatable Read	No	No
Serializable	No	No

Scope and Different Types of Locks 1-5

The SQL Server Database Engine locks the resources that use different lock modes, which determine the resources that are accessible to concurrent transactions.

Lock Mode	Description
Update	Is used on resources that are to be updated.
Shared	Is used for read operations that do not change data such as `SELECT` statement.
Intent	Is used to establish a hierarchy of locks.
Exclusive	Is used for `INSERT`, `UPDATE`, or `DELETE` data-modification operations.
`BULK UPDATE`	Is used while copying bulk data into the table.
Schema	Is used when the operation is dependent on the table schema.

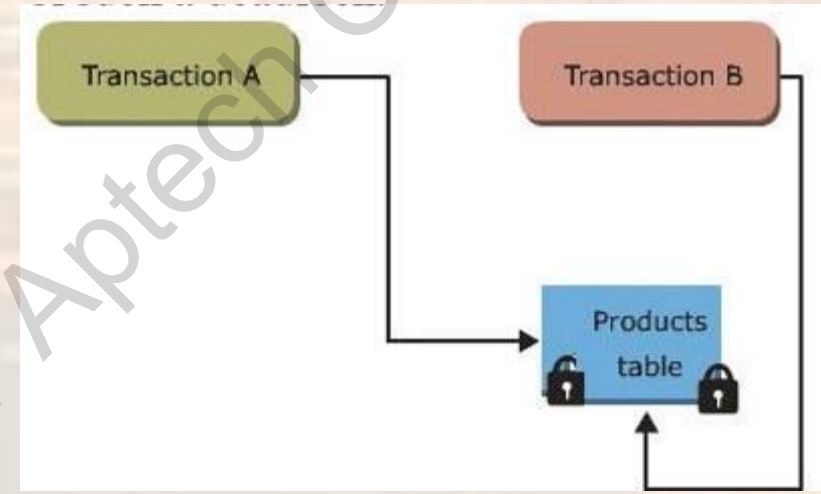
Lock Modes

Scope and Different Types of Locks 2-5

Different types of locks are as follows:

Update Locks

These locks avoid common forms of deadlock. In a serializable transaction, the transaction will read data, acquire a shared lock on the row or a page, and modify the data that requires lock conversion to an exclusive lock.



Deadlock

Scope and Different Types of Locks 3-5

Shared Locks

- These locks allow parallel transactions to read a resource under pessimistic concurrency control.
- Shared locks are released on a resource once the read operation is completed, except the isolation level is set to repeatable read or higher.

Exclusive Locks

- These locks prevent access to resources by concurrent transactions.
- By using an exclusive lock, no other transaction can change data and read operations take place only through the read uncommitted isolation level or NOLOCK hint.
- DML statements such as INSERT, DELETE, and UPDATE combine modification and read operations.

Scope and Different Types of Locks 4-5

Intent Locks

- To prevent other transactions from changing the higher-level resource in a way that will invalidate the lock at the lower-level.
- To improve the efficiency of the Database Engine for identifying the lock conflicts those are at the higher level of granularity.

Lock Mode	Description
Intent shared (IS)	Protects the requested shared lock on some resources that are lower in the hierarchy.
Intent exclusive (IX)	Protects the requested exclusive lock on some resources lower in the hierarchy. IX is a superset of IS, that protects requesting shared locks on lower level resources.
Shared with Intent Exclusive (SIX)	Protects the requested shared lock on all resources lower in the hierarchy and intent exclusive locks on some of the lower level resources. Concurrent IS locks are allowed at the top-level resource.
Intent Update (IU)	Protects the requested update locks on all resources lower in the hierarchy. IU locks are used only on page resources. IU locks are converted to IX locks if an update operation takes place.
Shared intent update (SIU)	Provides combination of S and IU locks, as a result of acquiring these locks separately and simultaneously holding both locks.
Update intent exclusive (UIX)	Provides combination of U and IX locks, as a result of acquiring these locks separately and simultaneously holding both locks.

Scope and Different Types of Locks 5-5

Bulk Update locks	Bulk update locks are used by the database engine. These locks are used when a large amount of data is copied into a table. These locks allow multiple threads to load bulk data continuously in the same table.
Schema Locks	Schema modification locks are used by Database Engine while performing a table DDL operation such as dropping a table or a column. Schema stability locks are used by the database engine while compiling and executing the queries.
Key-Range Locks	These types of locks protect a collection of rows that are implicitly present in a recordset. Key-range locks prevent phantom reads.

Transaction Management

- SQL Server implements several transaction isolation levels that ensure the ACID properties of these transactions.
- In reality, it means that it uses locks to facilitate transactional access to shared database resources and also, prevent the interference between the transactions.

Transaction Log 1-2

- Transaction log is a critical component of the database and if a system failure occurs, the transaction log will be required to bring the database to a consistent data.
- The transaction log should not be moved or deleted until users understand the consequences of doing it.

Operations Supported By Transaction Log

Individual transactions recovery	Incomplete transactions recovery when SQL Server starts	Transactional replication support	Disaster recovery solutions and high availability support	Roll back a file, restored database, filegroup, or page forward to the point of failure

Transaction Log 2-2

Log_reuse_wait	Log_reuse_wait desc	Description
0	NOTHING	Specifies that at present, there are more than one reusable virtual log file.
1	CHECKPOINT	Specifies that there is no checkpoint occurred since the last log truncation or the head of the log has not moved beyond a virtual log file.
2	LOG_BACKUP	Specifies a log backup that is required before the transaction log truncates.
3	ACTIVE_BACKUP_OR_RESTORE	Specifies that the data backup or a restore is in progress.
4	ACTIVE_TRANSACTION	Specifies that a transaction is active.
5	DATABASE_MIRRORING	Specifies that the database mirroring is paused or under high-performance mode, the mirror database is significantly behind the principal database.

Catalog View Columns

Summary

- A transaction is a sequence of operations that works as a single unit.
- BEGIN TRANSACTION marks the beginning point of an explicit or local transaction.
- COMMIT TRANSACTION marks an end of a successful implicit or explicit transaction.
- ROLLBACK with an optional keyword WORK rolls back a user-specified transaction to the beginning of the transaction.
- @@TRANCOUNT is a system function that returns a number of BEGIN TRANSACTION statements that occur in the current connection.
- Isolation levels are provided by the transaction to describe the extent to which a single transaction must be isolated from changes made by other transactions.
- The SQL Server Database Engine locks the resources using different lock modes, which determine the resources that are accessible to concurrent transactions.