20.1 Transactions with SQL

Isolation levels

Transaction statements are standardized in SQL and implemented in most relational databases. However, transaction statements interact with concurrency and recovery systems, which vary significantly. Consequently, most databases offer many nonstandard extensions.

This section describes transaction statements for MySQL configured with the InnoDB storage engine. The statements are similar for other storage engines and databases. MySQL storage engines are described elsewhere in this material.

The ***SET TRANSACTION*** statement sets the isolation level for subsequent transactions:

Figure 20.1.1: SET TRANSACTION syntax.

```
SET [ GLOBAL | SESSION ] TRANSACTION
ISOLATION LEVEL [ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ
UNCOMMITTED ];
```

The keyword following ISOLATION LEVEL balances isolation of concurrent transactions against performance. SERIALIZABLE provides complete isolation with longer transaction duration. READ UNCOMMITTED allows conflicts between concurrent transactions with shorter duration. The four isolation levels are described elsewhere in this material.

The GLOBAL and SESSION keywords determine the scope of the isolation level:

- ***SESSION*** sets the isolation level for all transactions in the current session. A ***session*** is a series of SQL statements submitted to a MySQL server, beginning when a user or program connects to the server and ending when the user or program disconnects.
- ***GLOBAL*** sets the isolation level for all transactions submitted to the MySQL server for all subsequent sessions. Existing sessions are not affected. The GLOBAL setting can be used only by a database administrator with appropriate MySQL privileges.

When neither keyword is specified, the isolation level applies only to the next transaction in the session.

PARTICIPATION ACTIVITY 20.1.1: SET TRANSACTION statement.

Animation captions:

PARTICIPATION ACTIVITY | 20.1.2: SET TRANSACTION statement.

Refer to the following sequence:

session begins

```
SET GLOBAL TRANSACTION
ISOLATION LEVEL READ UNCOMMITTED;
```

session ends

session begins

```
SET TRANSACTION
ISOLATION LEVEL SERIALIZABLE;
```

transaction 1

transaction 2

```
SET SESSION TRANSACTION
ISOLATION LEVEL REPEATABLE READ;
```

transaction 3

transaction 4

```
SET TRANSACTION
ISOLATION LEVEL READ COMMITTED;
```

transaction 5

session ends

Match the isolation level to the transaction.

If unable to drag and drop, refresh the page.

READ UNCOMMITTED SERIALIZABLE READ COMMITTED

REPEATABLE READ

	Transaction 1
	Transaction 2
	Transaction 4
	Transaction 5

Reset

Transaction boundaries

A ***transaction boundary*** is the first or last statement of a transaction. A transaction boundary is one of three statements:

- The ***START TRANSACTION*** statement starts a new transaction.
- The ***COMMIT*** statement commits the current transaction.
- The ***ROLLBACK*** statement rolls back the current transaction.

All statements between START TRANSACTION and the following COMMIT or ROLLBACK comprise a single transaction.

Figure 20.1.2: Transaction statements.

```
START
TRANSACTION;

COMMIT;

ROLLBACK;
```

In MySQL with InnoDB, the behavior of statements outside a transaction boundary depends on the ***autocommit*** system variable. When autocommit is ON, individual statements are separate transactions and immediately commit or roll back. When autocommit is OFF, individual statements do not commit immediately. Instead, a new transaction begins automatically when a program starts and after each COMMIT and ROLLBACK. The default setting is ON.

System variables like autocommit are assigned a value using a ***SET*** statement, as shown in the figure below.

Figure 20.1.3: SET autocommit.

```
SET autocommit = [ OFF | ON
];
```

COMMIT and ROLLBACK have optional keywords:

- ***AND CHAIN*** overrides the autocommit setting and starts a new transaction, as if a START TRANSACTION were executed. The isolation level of the new transaction is the same as the prior transaction.
- ***RELEASE*** ends the current session and disconnects from the server.

Figure 20.1.4: COMMIT and ROLLBACK statements.

```
COMMIT [ AND CHAIN ] [ RELEASE ];

ROLLBACK [ AND CHAIN ] [ RELEASE
];
```

Data definition statements such as CREATE TABLE and DROP INDEX, and administrative statements such as ANALYZE TABLE, always commit immediately. These statements behave as if a COMMIT were executed before and after the statement.

BEGIN statement

*The **BEGIN** statement is identical to START TRANSACTION. However, BEGIN is easily confused with the BEGIN - END statement pair, which has different meaning in many programming languages. For this reason, the BEGIN statement is not recommended.*

PARTICIPATION ACTIVITY | 20.1.3: Transaction boundary statements.

Animation captions:

PARTICIPATION ACTIVITY | 20.1.4: Transaction boundaries.

Refer to the following sequence:

session begins

```
SET autocommit = OFF;
```

session ends

session begins
statement 1
statement 2
statement 3

```
START TRANSACTION;
```

statement 4

```
COMMIT AND CHAIN;
```

statement 5
statement 6
statement 7

```
ROLLBACK;
```

statement 8
session ends

Match the transaction boundary to the statement.

If unable to drag and drop, refresh the page.

both start and end | end | neither start nor end | start

	Statement 1
	Statement 3
	Statement 6
	Statement 8

Reset

Savepoints

A ***savepoint*** is a point within a transaction where partial transaction results are saved temporarily. Savepoints prevent redoing work in lengthy transactions. If an error occurs within a transaction after a savepoint, the transaction can restart at the savepoint rather than at the beginning.

Savepoints are managed with three statements using an identifier, the savepoint name:

- The ***SAVEPOINT*** statement saves internal transaction data and associates the data with the identifier.
- The ***RELEASE SAVEPOINT*** statement discards the identifier and saved data.
- The ***ROLLBACK TO*** statement resets transaction data to the savepoint values, restarts processing at the savepoint, and releases all subsequent savepoints.

Savepoints are temporary and internal to a transaction. When the transaction commits, all savepoints are released.

Figure 20.1.5: Savepoint statements.

```
SAVEPOINT identifier;

ROLLBACK TO identifier;

RELEASE SAVEPOINT
identifier;
```

PARTICIPATION ACTIVITY 20.1.5: SAVEPOINT statement.

Animation captions:

PARTICIPATION ACTIVITY | 20.1.6: Savepoints.

1) A ________ statement temporarily saves data read and written by a transaction.

Check Show answer

2) A ________ statement erases saved data for zero, one, or many savepoints.

Check Show answer

3) A ________ statement erases saved data for exactly one savepoint.

Check Show answer

4) A ________ statement reverses all changes made by a transaction.

Check Show answer

Checkpoints

A ***dirty block*** is a database block that has been updated in main memory but not yet saved on storage media. A ***checkpoint*** saves dirty blocks and log records, as follows:

1. Suspend database processing.
2. Write all unsaved log records to the log file.
3. Write all dirty blocks to storage media.
4. Write a checkpoint record in the log.
5. Resume database processing.

Checkpoints enable rapid recovery from system failure, as described elsewhere in this material.

A ***fuzzy checkpoint*** resumes processing while saving dirty blocks. A fuzzy checkpoint improves database availability but complicates recovery if the system fails while saving dirty blocks.

Checkpoint syntax and procedures are dependent on database internals and vary greatly across databases. Most relational databases have a default process for automatic checkpoints. Database administrators with appropriate privileges can modify the default process or force a checkpoint manually.

MySQL with InnoDB executes a fuzzy checkpoint every second. A database administrator can modify the default behavior by setting system variables. Ex:

- `SET flush_time = 2` changes the default interval to 2 seconds.
- `SET flush = ON` executes a fuzzy checkpoint after each SQL statement.

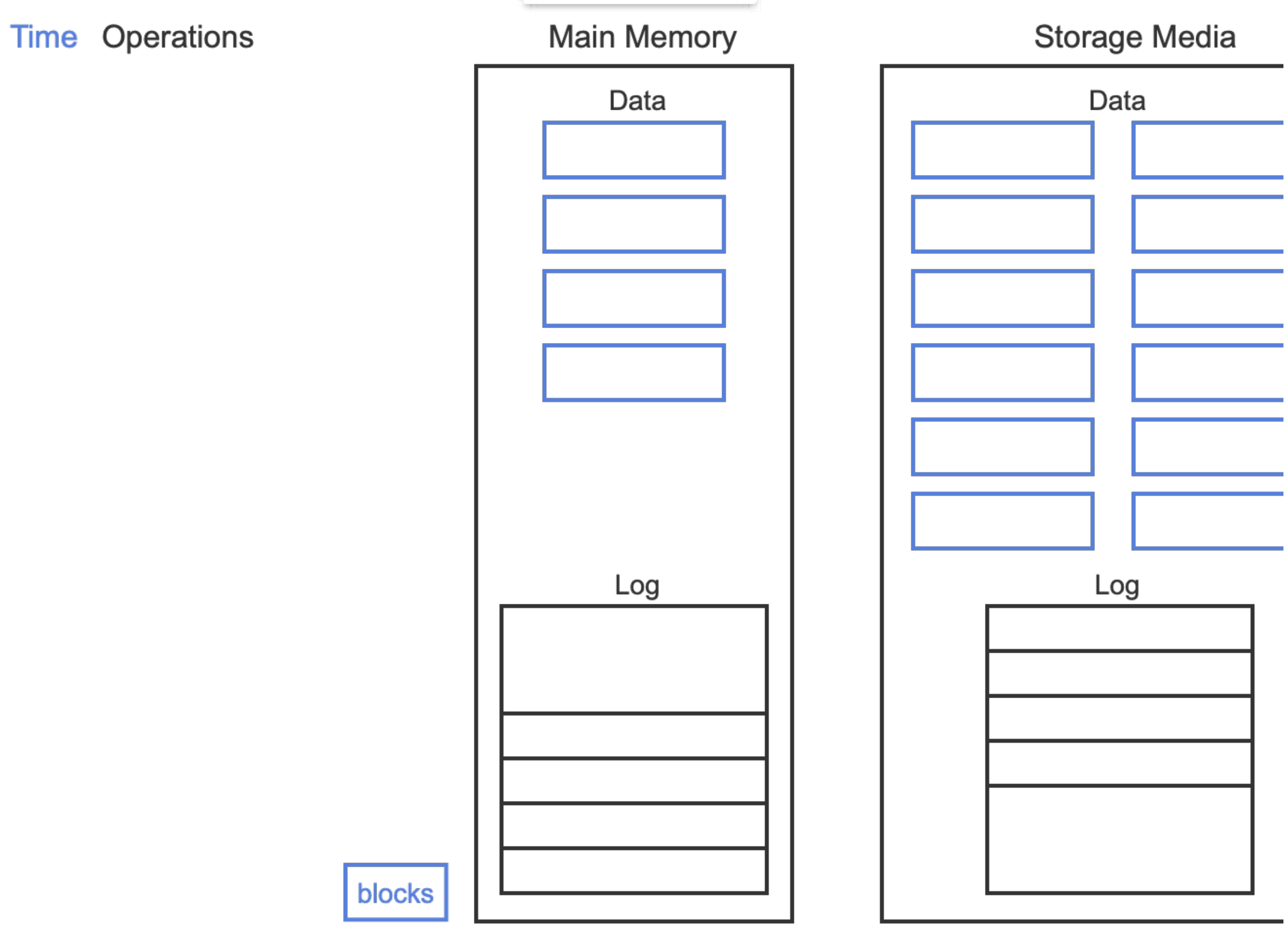
Log records and dirty blocks can be explicitly flushed with the ***FLUSH*** statement. Ex:

- `FLUSH TABLES` writes all dirty blocks to storage media.
- `FLUSH LOGS` saves all log records to storage media.

Some databases support manual checkpoints with the ***CHECKPOINT*** keyword. Ex: In SQL Server, `CHECKPOINT` forces a checkpoint. In Oracle Database, `ALTER SYSTEM CHECKPOINT` forces a checkpoint.

The above statements are examples. Commercial databases provide numerous checkpoint mechanisms with differing syntax and capabilities.

PARTICIPATION ACTIVITY | 20.1.7: Checkpoints.



Animation captions:

PARTICIPATION ACTIVITY | 20.1.8: Checkpoints.

1) CHECKPOINT statement syntax is specified in the SQL standard.

specified in the SQL standard.

- ○ True
- ○ False

2) Checkpoints can be initiated either manually by the database administrator or automatically by the database.

- ○ True
- ○ False

3) A dirty block is a block that has been corrupted and cannot be read.

- ○ True
- ○ False

4) MySQL with InnoDB suspends processing during a checkpoint and restarts after a checkpoint record is written to the log.

- ○ True
- ○ False

CHALLENGE ACTIVITY | 20.1.1: Transactions with SQL.

544874.3500394.qx3zqy7

Start

```
session begins
SET GLOBAL TRANSACTION
ISOLATION LEVEL SERIALIZABLE;
session ends

session begins
SET TRANSACTION
ISOLATION LEVEL REPEATABLE READ;
transaction 1
transaction 2
SET SESSION TRANSACTION
```

Select each transaction's isolation level.

`transaction 1`	Pick
`transaction 2`	Pick
`transaction 3`	Pick

```
ISOLATION LEVEL READ UNCOMMITTED;
transaction 3
transaction 4
SET TRANSACTION
ISOLATION LEVEL READ COMMITTED;
transaction 5
session ends
```

transaction 4	Pick
transaction 5	Pick

1 2

Check Next

Exploring further:

- MySQL transaction statements
- MySQL with InnoDB locking
- MySQL with InnoDB checkpoints

20.2 Recovery

Failure scenarios

The recovery system supports atomic transactions by ensuring partial transaction results are not saved in a database. The recovery system supports durable transactions by ensuring committed transactions are not lost due to hardware or software failures.

The recovery system must manage three failure scenarios:

1. A ***transaction failure*** results in a rollback. The application program may initiate a rollback due to logical errors. The database system may initiate rollback due to deadlock or insufficient disk space. The operating system may initiate rollback if a hardware or software component fails. Regardless of the cause, the recovery system restores all data changed by the transaction to the original values.
2. A ***system failure*** includes a variety of events resulting in the loss of main memory. Databases initially write to main memory blocks, which are lost when an application, the operating

initially write to main memory blocks, which are lost when an application, the operating system, or the database system fails. Blocks are subsequently saved on storage media, which normally survive a system failure. If main memory is lost before blocks are written to storage media, data written by committed transactions might be lost. In this event, the recovery system:

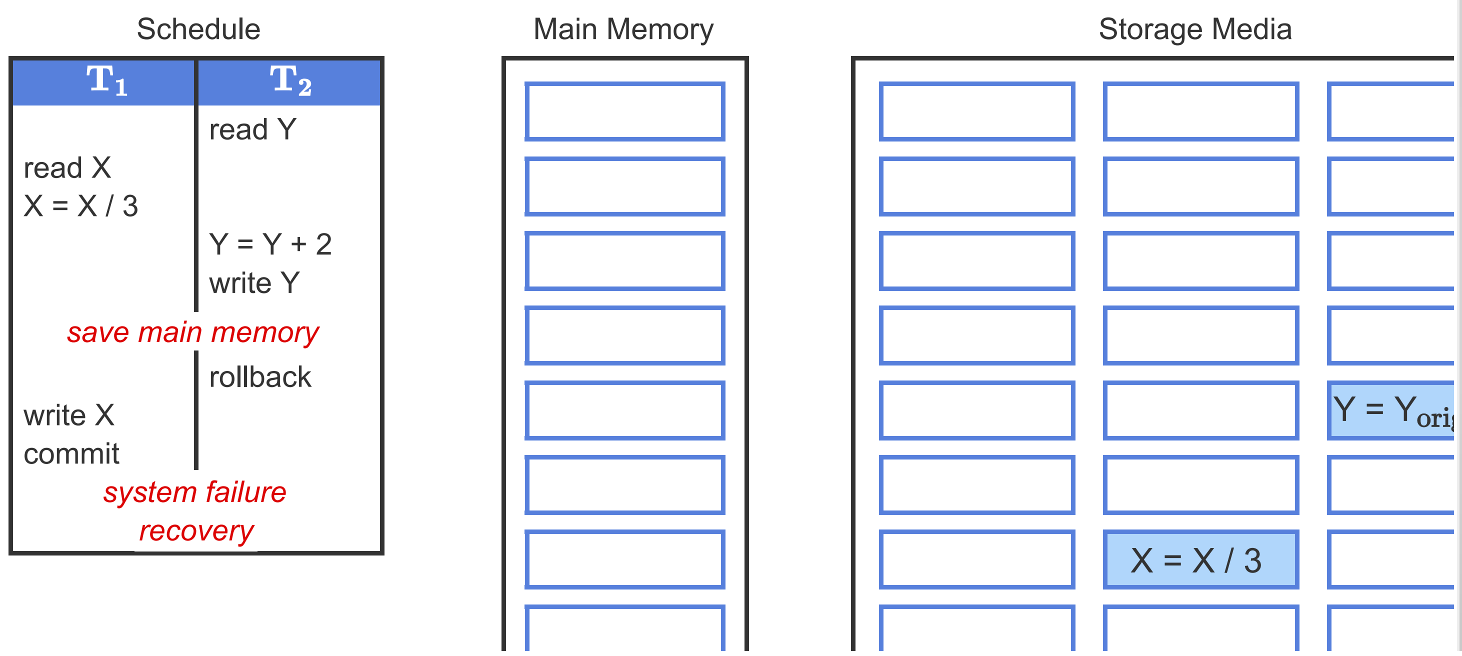
- Recovers data written to main memory, but not storage media, by committed transactions.
- Rolls back data written to storage media by uncommitted transactions.

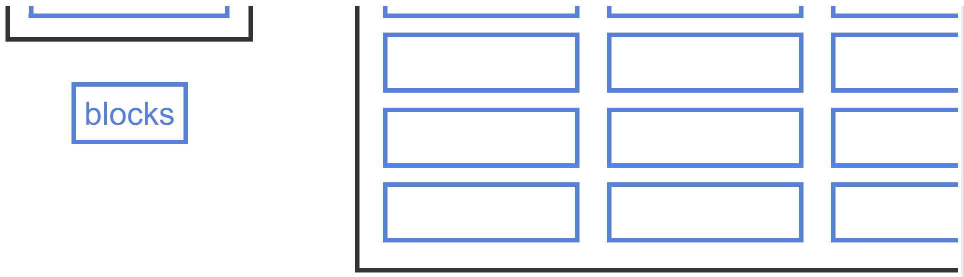
3. A ***storage media failure*** occurs when the database is corrupted or the database connection is lost.

   Many storage systems automatically make redundant copies of data. If one copy of data is corrupted, the storage system automatically switches to a backup copy without intervention by the database or operating system. Nevertheless, storage media do fail occasionally. Alternatively, the connection between the processor and database server might fail. Either way, the database is unavailable to the application.

In principle, recovery from storage media failure is similar to recovery from system failure. In both cases, the recovery system must restore committed transactions and roll back uncommitted transactions. However, storage media failure may cause massive data loss with lengthy recovery times. Consequently, most commercial databases implement special techniques for rapid recovery from storage media failure.

PARTICIPATION ACTIVITY 20.2.1: System failure.





Animation content:

Static figure:
A schedule has actions in the following sequence. The schedule has internal captions, shown in parentheses:
T2: read Y
T1: read X
T1: X = X / 3
T2: Y = Y + 2
T2: write Y
(save main memory)
T2: rollback
T1: write X
T1: commit
(system failure)
(recovery)

A main memory diagram contains 8 blocks.

A storage media diagram contains 33 blocks. One block contains X = X / 3. Another block contains Y = Y original.

Step 1: The database processes two transactions. The schedule appears without internal captions. Main memory and storage media diagrams appear. All blocks are empty.

Step 2: The database writes a new Y value to a block in main memory. Action T2: write Y is highlighted. Y = Y + 2 appears in a main memory block.

Step 3: The database saves main memory to storage media. All saved blocks in main memory are released for reuse. The save main memory caption appears in the schedule. Y = Y + 2 moves from the main memory block to a storage media block.

Step 4: The database rolls back T2, restoring Y's original value in main memory. Action T2: rollback is highlighted. Y = Y original appears in a main memory block.

rollback is highlighted. Y = Y original appears in a main memory block.

Step 5: The database writes a new X value in main memory and commits T1. Action T1: write X is highlighted. X = X / 3 appears in a main memory block.

Step 6: Before main memory is saved to storage media, the operating system fails, and main memory is lost. The system failure caption appears in the schedule. All text in main memory blocks disappears.

Step 7: The recovery system must restore 'X = X / 3' and roll back 'Y = Y + 2' in storage media. The recovery caption appears in the schedule. Y = Y original appears in a storage media block. X = X / 3 appears in another storage media block.

Animation captions:

1. The database processes two transactions.
2. The database writes a new Y value to a block in main memory.
3. The database saves main memory to storage media. All saved blocks in main memory are released for reuse.
4. The database rolls back T_2, restoring Y's original value in main memory.
5. The database writes a new X value in main memory and commits T_1.
6. Before main memory is saved to storage media, the operating system fails, and main memory is lost.
7. The recovery system must restore 'X = X / 3' and roll back 'Y = Y + 2' in storage media.

PARTICIPATION ACTIVITY | 20.2.2: Failure scenarios.

Select the failure scenario that best describes the example.

1) Application programs run on a client machine. The database runs on a separate server machine. The network between client and server fails, and the database does not respond to any application requests.

- ○ Transaction failure
- ○ System failure
- ○ Storage media failure

2) The database detects two deadlocked

2) The database detects two deadlocked transactions. To break the deadlock, the database rolls back one of the transactions.

- ○ Transaction failure
- ○ System failure
- ○ Storage media failure

3) Due to a security breach, computer memory is corrupted. After the security breach is resolved, the database administrator restarts the database.

- ○ Transaction failure
- ○ System failure
- ○ Storage media failure

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

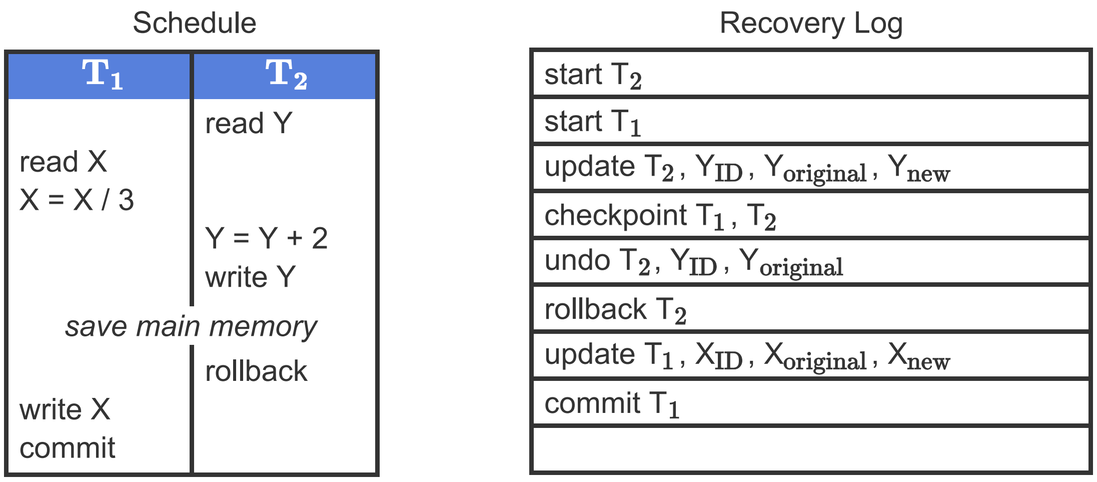
1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records also track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.

indicator and the transaction identifier.

4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.2.3: Recovery log records.



Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	*save main memory*
	rollback
write X	
commit	

Recovery Log

start T_2
start T_1
update T_2, Y_{ID}, $Y_{original}$, Y_{new}
checkpoint T_1, T_2
undo T_2, Y_{ID}, $Y_{original}$
rollback T_2
update T_1, X_{ID}, $X_{original}$, X_{new}
commit T_1

Animation content:

Static figure:
A schedule has actions in the following sequence. The schedule has internal captions, shown in parentheses:
T2: read Y
T1: read X
T1: X = X / 3
T2: Y = Y + 2
T2: write Y
(save main memory)
T2: rollback
T1: write X
T1: commit

T1: commit

A recovery log has the following records:
start T2
start T1
update T2, Y ID, Y original, Y new
checkpoint T1, T2
undo T2, Y ID, Y original
rollback T2
update T1, X ID, X original, X new
commit T1

Step 1: When the transactions execute, start records are written in the recovery log. The schedule appears. An empty recovery log appears. The two read actions are highlighted. The two start records appear in the recovery log.

Step 2: When T2 writes Y, an update record is written in the log. Action write Y is highlighted. The first update record appears in the recovery log.

Step 3: When the system saves all updates from main memory to storage media, a checkpoint record is written in the log. The save main memory caption is highlighted. The checkpoint record appears in the recovery log.

Step 4: When T2 rolls back, compensation and rollback records are written in the log. Action rollback is highlighted. The undo and rollback records appear in the recovery log.

Step 5: When T1 writes X, an update record is written in the log. Action write X is highlighted. The second update record appears in the recovery log.

Step 6: When T1 commits, a commit record is written in the log. Action commit is highlighted. The commit record appears in the recovery log.

Animation captions:

1. When the transactions execute, start records are written in the recovery log.
2. When T_2 writes Y, an update record is written in the log.
3. When the system saves all updates from main memory to storage media, a checkpoint record is written in the log.
4. When T_2 rolls back, compensation and rollback records are written in the log.
5. When T_1 writes X, an update record is written in the log.
6. When T_1 commits, a commit record is written in the log.

PARTICIPATION ACTIVITY | 20.2.4: Recovery log.

1) After all updates have been reversed in a rollback, a(n) _________ record is written in the log.

Check Show answer

2) When a transaction deletes a table row, a(n) _________ record is written in the log.

Check Show answer

3) A(n) _________ record indicates all data is saved from main memory to storage media.

Check Show answer

4) A(n) _________ record always appears in the log at the beginning of a transaction.

Check Show answer

5) A(n) _________ record is written in the log whenever an update is reversed during a rollback.

Check Show answer

6) A list of active transactions appears in a(n) ________ record.

Check Show answer

Recovery from transaction failure

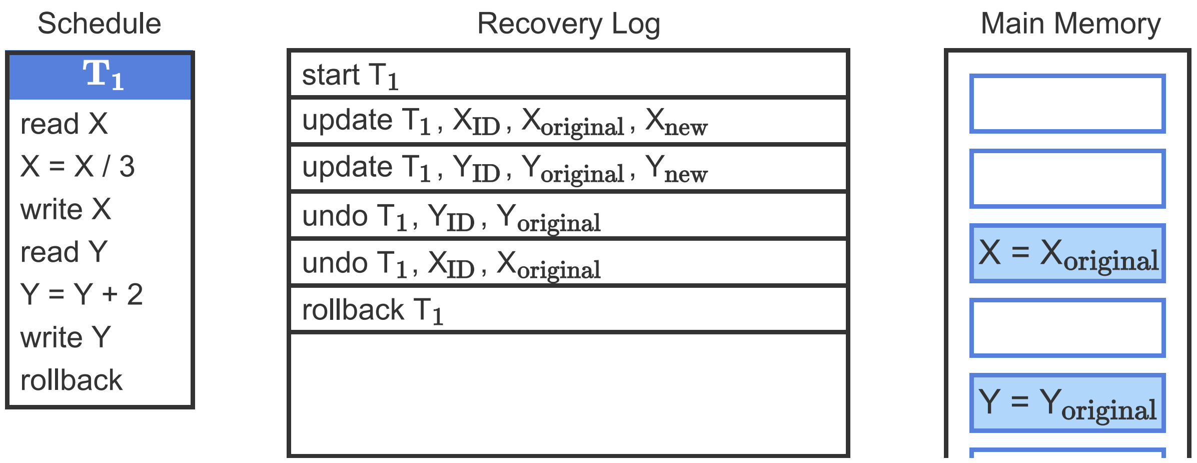
Recovery from a transaction failure is relatively simple. A database component, such as the concurrency system, instructs the recovery system to roll back transaction T. The recovery system reads the recovery log *backwards*, searching for update records for T. For each update record, the recovery system:

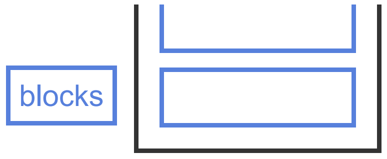
- Restores data D to original value V in the update record.
- Writes a compensation record for T, D, and V to the end of the log.

Eventually, the recovery system reads the 'start T' transaction record and writes a 'rollback T' record to the end of the log. At this point, the rollback is complete.

The recovery system reads the log backwards because data must be restored in reverse order of transaction operations.

PARTICIPATION ACTIVITY 20.2.5: Recovery from transaction failure.


Schedule
T_1
read X
X = X / 3
write X
read Y
Y = Y + 2
write Y
rollback
Recovery Log
start T_1
update T_1, X_{ID}, $X_{original}$, X_{new}
update T_1, Y_{ID}, $Y_{original}$, Y_{new}
undo T_1, Y_{ID}, $Y_{original}$
undo T_1, X_{ID}, $X_{original}$
rollback T_1
Main Memory
X = $X_{original}$
Y = $Y_{original}$



Animation content:

Static figure:
A schedule has one transaction with actions in the following sequence:
T1: read X
T1: X = X / 3
T1: write X
T1: read Y
T1: Y = Y + 2
T1: write Y
T1: rollback

A recovery log has these records:
start T1
update T1, X ID, X original, X new
update T1, Y ID, Y original, Y new
undo T1, Y ID, Y original
undo T1, X ID, X original
rollback T1

A main memory diagram has seven blocks. One block contains X = X original. Another block contains Y = Y original.

Step 1: As the transaction executes, a start record is written to the recovery log. The schedule, recovery log, and main memory appear. The recovery log and main memory are empty. The start record appears in the recovery log.

Step 2: Write statements generate update records in the log and save data in main memory. Actions write X and write Y are highlighted. The two update records appear. X = X / 3 and Y = Y + 2 appear in main memory blocks..

Step 3: When the system or programmer initiates rollback, the recovery system reads the log in reverse. Action rollback is highlighted. The second update record is highlighted.

Step 4: Y is restored to original value in main memory, and a compensation record is written to the log. The first undo record appears. In main memory, Y = Y original replaces Y = Y + 2.

Step 5: X is restored to original value in main memory, and a compensation record is written to the log. The first update record is highlighted. The second undo record appears. In main memory, X = X original replaces X = X / 3.

Step 6: The rollback record is written to the log. The rollback is complete. The rollback record appears.

Animation captions:

1. As the transaction executes, a start record is written to the recovery log.
2. Write statements generate update records in the log and save data in main memory.
3. When the system or programmer initiates rollback, the recovery system reads the log in reverse.
4. Y is restored to original value in main memory, and a compensation record is written to the log.
5. X is restored to original value in main memory, and a compensation record is written to the log.
6. The rollback record is written to the log. The rollback is complete.

PARTICIPATION ACTIVITY 20.2.6: Recovery from transaction failure.

Refer to the schedule below :

T_1 reads X
T_2 reads Y
T_2 writes Z
T_1 writes Y
T_2 writes X
deadlock

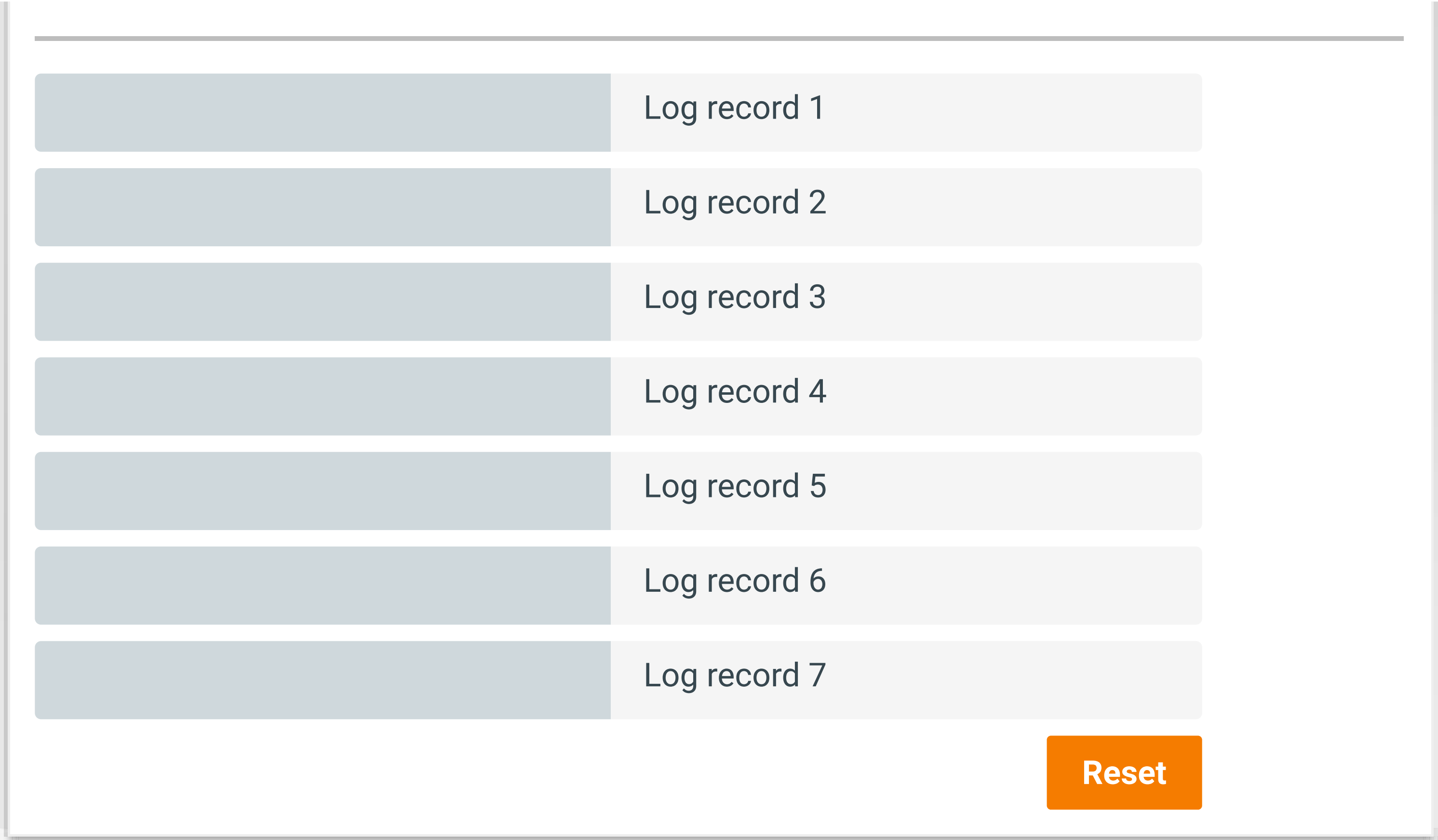
Deadlock occurs because T_1 is waiting for an exclusive lock on Y, and T_2 is waiting for an exclusive lock on X. When deadlock occurs, the database rolls back T_2 and completes T_1.

Order the log records to match the schedule.

If unable to drag and drop, refresh the page.

commit T_1 **start T_1** **update T_1, Y . . .** **rollback T_2** **start T_2**

undo T_2, Z . . . **update T_2, Z . . .**



Recovery from system failure

Recovery from a system failure has two phases. The ***redo phase*** restores all transactions that were committed or rolled back since the last checkpoint. The ***undo phase*** rolls back transactions that were neither committed nor rolled back.

In the *redo phase*, the recovery system reads the recovery log *forward* from the latest checkpoint record. While reading the log, the recovery system maintains a list of active transactions. The list is initialized with active transactions in the checkpoint record. As each log record is read:

- Transactions in a 'start' transaction record are added to the list.
- Transactions in a 'commit' or 'rollback' transaction record are removed from the list.
- Data in an update record is set to the new value.
- Data in a compensation record is restored to the original value.

The above redo process performs unnecessary work on rolled-back transactions. The process changes data in update records and later restores the same data in compensation records. In principle, the recovery system could avoid unnecessary work by ignoring all rolled-back transactions. Processing all log records is simpler than ignoring rolled-back transactions, however, and is thus commonly implemented.

At the end of the redo phase, all transactions remaining on the list have no 'commit' or 'rollback' records. These transactions are rolled back in the undo phase.

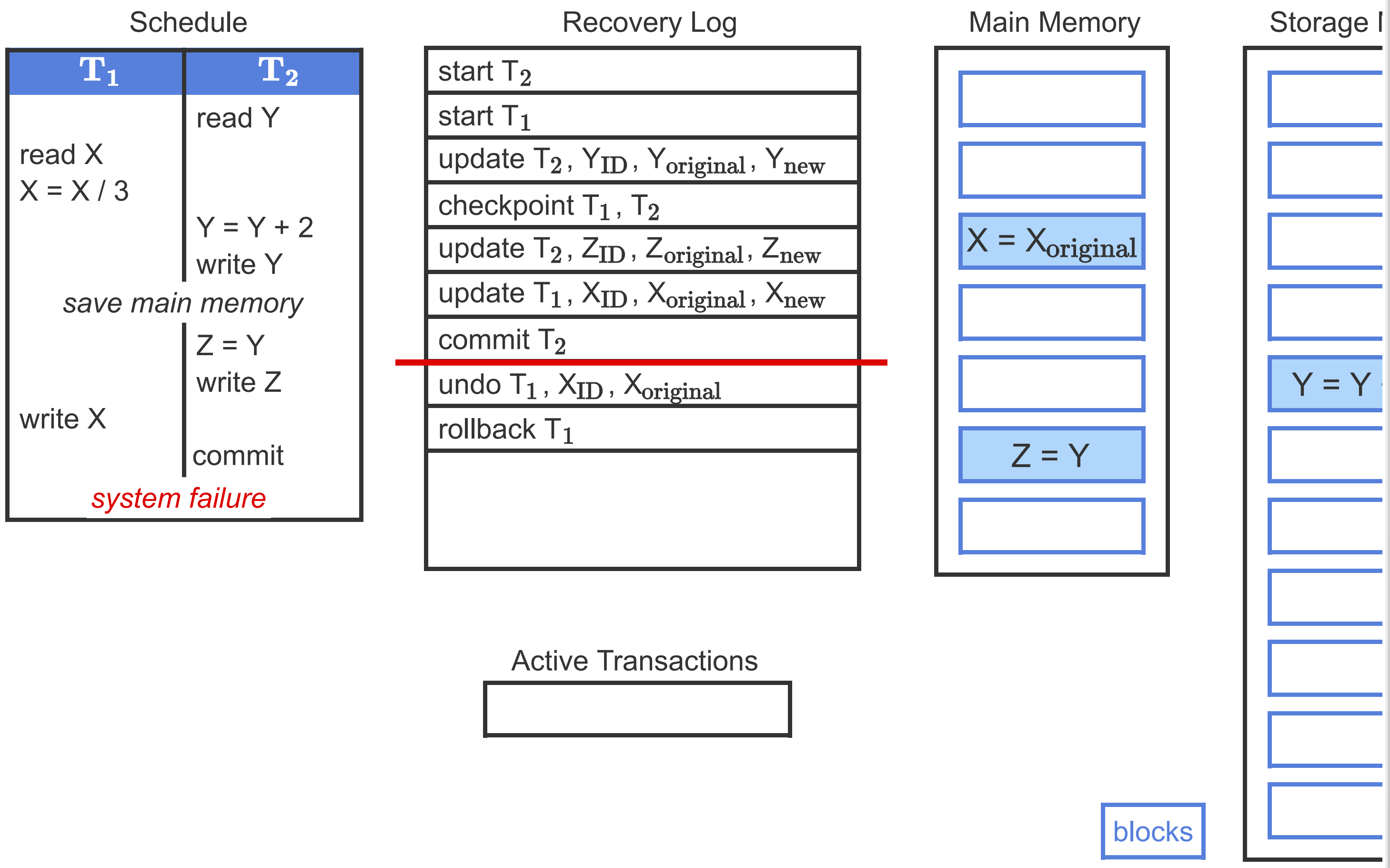
During the *undo phase*, the recovery system reads the recovery log *backwards* from the last record. While reading the log, the recovery system maintains a list of incomplete transactions. The list is initialized with active transactions remaining from the redo phase. Each transaction on the list is rolled back:

- When an update record is read, data is restored to the original value, and a compensation record is written at the end of the log.
- When the 'start' transaction record is read, the transaction is removed from the list, and a 'rollback' transaction record is written at the end of the log.

The above process is the same as the rollback process for transaction failures, described above.

When the transaction list is empty, the undo phase ends, and recovery is complete.

PARTICIPATION ACTIVITY | 20.2.7: Recovery from system failure.



Animation content:

Static figure:
A schedule has actions in the following sequence. The schedule has internal captions, shown in parentheses:
T2: read Y
T1: read X
T1: X = X / 3
T2: Y = Y + 2
T2: write Y
(save main memory)
T2: Z = Y
T2: write Z
T1: write X
T2: commit
(system failure)

A recovery log has the following records:
start T2
start T1
update T2, Y ID, Y original, Y new
checkpoint T1, T2
update T2, Z ID, Z original, Z new
update T1, X ID, X original, X new
commit T2
undo T1, X ID, X original
rollback T1

A red line appears below the commit record. An empty list of active transactions appears under the recovery log.

A main memory diagram contains 7 blocks. One block contains X = X original. Another block contains Z = Y

A storage media diagram has 11 blocks. One block contains Y = Y + 2.

Step 1: The schedule executes normally. The checkpoint record indicates transactions T1 and T2 are active. The schedule, recovery log, main memory, and storage media appear. The schedule executes through the commit action. All records through commit appear in the recovery log. X = X / 3 and Z = Y appear in main memory blocks. Y = Y + 2 appears in a storage media block.

Step 2: The system fails, and memory is lost. The caption system failure appears in the schedule. A red line appears under the commit record. Main memory blocks are erased.

Step 3: The recovery system initializes the list of active transactions from the last checkpoint. The checkpoint record is highlighted. T1 and T2 appear in the active transactions list.

Step 4: During the redo phase, the system reads the log forward from the checkpoint and restores data from update records. The two update records are highlighted. X = X / 3 and Z = Y reappear in main memory blocks.

Step 5: The commit record in the log removes T2 from the transaction list. The commit T2 record is highlighted. T2 disappears from the active transactions record.

Step 6: During the undo phase, the system reads the log backwards and rolls back changes made by transactions in the list. The undo record appears. The update T1 record is highlighted. X = X original replaces X = X / 3 in main memory.

Step 7: The system reads the 'start T1' record in the log, writes a rollback record and deletes T1 from the list. Recovery is complete. The start T1 record is highlighted. The rollback T1 record appears. T1 disappears from the active transactions record.

Animation captions:

1. The schedule executes normally. The checkpoint record indicates transactions T$_1$ and T$_2$ are active.
2. The system fails, and memory is lost.
3. The recovery system initializes the list of active transactions from the last checkpoint.
4. During the redo phase, the system reads the log forward from the checkpoint and restores data from update records.
5. The commit record in the log removes T$_2$ from the transaction list.
6. During the undo phase, the system reads the log backwards and rolls back changes made by transactions in the list.
7. The system reads the 'start T$_1$' record in the log, writes a rollback record and deletes T$_1$ from the list. Recovery is complete.

PARTICIPATION ACTIVITY | 20.2.8: Recovery from system failure.

1) During the *redo* phase, what log records remove a transaction from the active transaction list?

○ Commit records only

- Rollback records only
- Both commit and rollback records

2) During the *redo* phase, what log records generate a database write?

- Update records only
- Compensation (undo) records only
- Both update and compensation (undo) records

3) During the *undo* phase, the recovery system reads the log in reverse and stops at:

- The most recent checkpoint record
- The start record for the last transaction in the active transaction list
- The first start record after the most recent checkpoint

4) During the *undo* phase, the recovery system writes compensation records for:

- Transactions that do not commit or roll back following the most recent checkpoint
- All transactions listed in the most recent checkpoint
- All transactions that roll back following the most recent checkpoint

CHALLENGE ACTIVITY | 20.2.1: Recovery.

544874.3500394.qx3zqy7

Start

Given the schedule below, complete the recovery log to match the schedule.

Schedule

T_1	T_2
	read Y
read X	
X = X - 8	
	Y = Y * 4
write X	

save main memory

rollback	
	write Y
	commit

Recovery Log

(A)
start T_1
update T_1, X_{ID}, $X_{original}$, X_{new}
checkpoint T_1, T_2
undo T_1, X_{ID}, $X_{original}$
(B)
update T_2, Y_{ID}, $Y_{original}$, Y_{new}
(C)

(A) Pick

(B) Pick

(C) Pick

1 2 3 4

Check Next

Recovery from storage media failure

Availability is the percentage of time a system is working from the perspective of the system user. Many databases require high availability, in excess of 99% of the time. Ex: A database that manages stock exchange trades must have availability approaching 100%.

Availability is a primary concern in recovery from storage media failures.

In principle, recovery from storage media failure might be the same as recovery from system failure, starting at the beginning of the recovery log. The log contains all database operations, so the database may be fully restored from scratch. In practice, however, this approach is not feasible the database may be fully restored from scratch. In practice, however, this approach is not feasible. The log of all database history is exceedingly large, so recovery time is exceedingly long and availability exceedingly low.

Two recovery techniques are commonly used: cold backup and hot backup.

The ***cold backup*** technique periodically creates checkpoints and, while transaction processing is paused, copies the database to backup media. The backup media might be tapes or disk drives, normally stored in a separate location from the primary database. Typically, checkpoints and backups are taken hourly or daily when database activity is low.

When storage media fails, the recovery system:

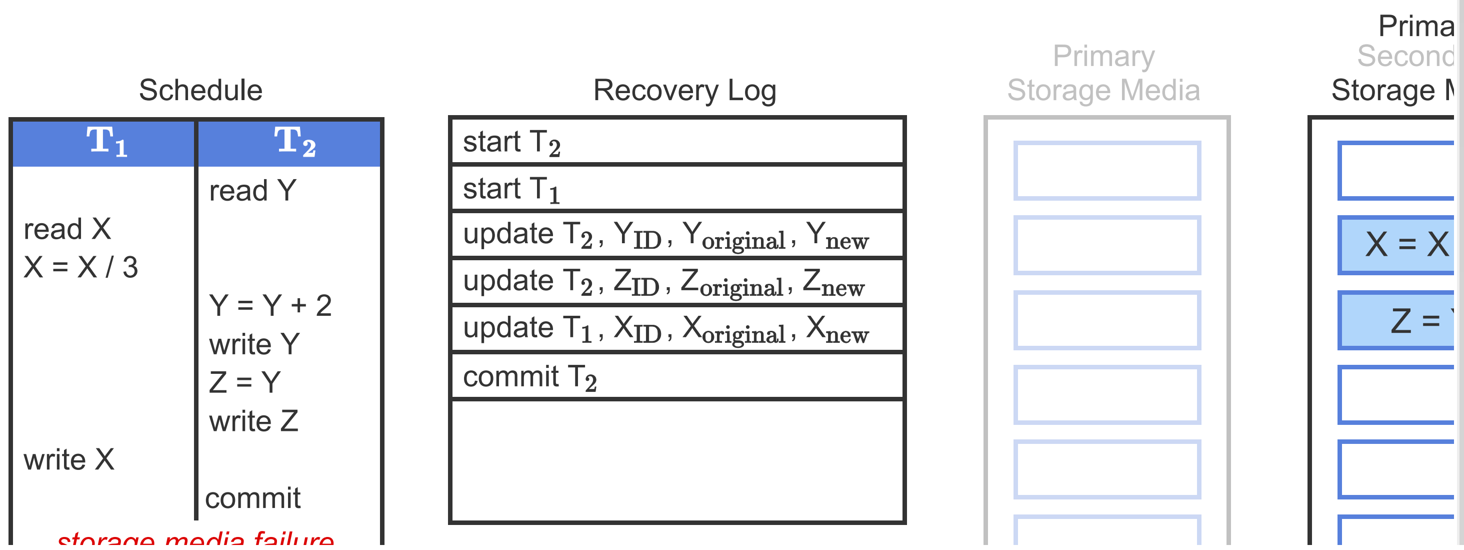
1. Copies the latest backup to the database.
2. Executes the system failure recovery process beginning at the latest checkpoint.

Depending on the backup size and period length, recovery might require minutes or hours. The database is unavailable during recovery, so the cold backup technique is used when low availability is acceptable.

The ***hot backup*** technique maintains a secondary database that is nearly synchronized with the primary database. As the primary database processes transactions, log records are sent to and processed by the secondary database. If the databases share a high-speed connection, the secondary database is only moments behind the primary database.

When storage media for the primary database fails, the secondary database becomes primary. This can be accomplished, for example, by swapping the internet addresses of primary and secondary servers. The new primary database is available in a matter of seconds, as soon as outstanding log records are processed.

PARTICIPATION ACTIVITY 20.2.9: Hot backup technique.



T_1	T_2
	read Y
read X X = X / 3	

	Y = Y + 2
	write Y
	Z = Y
	write Z

write X	
	commit

start T_2
start T_1
update T_2, Y_{ID}, $Y_{original}$, Y_{new}
update T_2, Z_{ID}, $Z_{original}$, Z_{new}
update T_1, X_{ID}, $X_{original}$, X_{new}
commit T_2

Animation content:

Static figure:
A schedule has actions in the following sequence. The schedule has internal captions, shown in parentheses:
T2: read Y
T1: read X
T1: X = X / 3
T2: Y = Y + 2
T2: write Y
T2: Z = Y
T2: write Z
T1: write X
T2: commit
(storage media failure)

A recovery log has the following records:
start T2
start T1
update T2, Y ID, Y original, Y new
update T2, Z ID, Z original, Z new
update T1, X ID, X original, X new
commit T2

A primary storage media diagram has empty blocks and is faded out.

A secondary storage media diagram appears. The following actions appear in three blocks:
X = X / 3
Z = Y
Y = Y + 2

Step 1: As transactions are executed, log records are sent to a secondary database. The schedule appears. Action T2: write Y is highlighted. The recovery log appears with only the start records and the first update record. Y = Y + 2 appears in blocks of both primary and secondary storage media.

Step 2: The secondary database may be in a remote location. Updates to the secondary database may be slightly delayed. Actions write Z, write X, and commit are highlighted. The remaining records appear in the recovery log. X = X / 3 and Z = Y appear in primary storage media blocks.

Step 3: The primary database fails. Caption storage media failure is highlighted in the schedule. Primary storage media is erased.

Step 4: After a momentary delay, the secondary database receives the remaining log records. X = X / 3 and Z = Y appear in secondary storage media blocks.

Step 5: The secondary database is now synchronized and becomes the primary database. The caption of secondary storage media changes to primary storage media.

Animation captions:

1. As transactions are executed, log records are sent to a secondary database.
2. The secondary database may be in a remote location. Updates to the secondary database may be slightly delayed.
3. The primary database fails.
4. After a momentary delay, the secondary database receives the remaining log records.
5. The secondary database is now synchronized and becomes the primary database.

PARTICIPATION ACTIVITY 20.2.10: Recovery from storage media failure.

1) Database availability is:

- ○ The percentage of time that a database is working properly.
- ○ The number of seconds per day that a database is responsive to application programs.
- ○ The percentage of time that a database is responsive to application programs.

2) With a cold backup, recovery from storage media failure reads the log:

- ○ From the beginning.

- ○ From the latest checkpoint.
- ○ From the latest commit record.

3) With a hot backup, as a transaction executes against the primary database:
   - ○ Log records are sent to the secondary database.
   - ○ The secondary database is updated synchronously, when the transaction commits or rolls back.
   - ○ The operating system synchronizes the secondary database.

20.3 LAB - Rollback and savepoint (Sakila)

Refer to the `actor` table of the Sakila database. The table in this lab has the same columns and data types but fewer rows.

Start a transaction and:

1. Insert a new actor with values 999, 'NICOLE', 'STREEP', '2021-06-01 12:00:00'
2. Set a SAVEPOINT.
3. Delete the actor with first name 'CUBA'.
4. Select all actors.
5. Roll back to the savepoint.
6. Select all actors a second time.

The actor with first name 'CUBA' should appear in the second SELECT but not the first.

In submit-mode tests that generate multiple result tables, the results are merged. Although the tests run correctly, the results appear in one table.

NOTE: Prior to running this lab in the Sakila database of MySQL Workbench:

1) Delete all `film_actor` rows that refer to actors with first name 'CUBA':

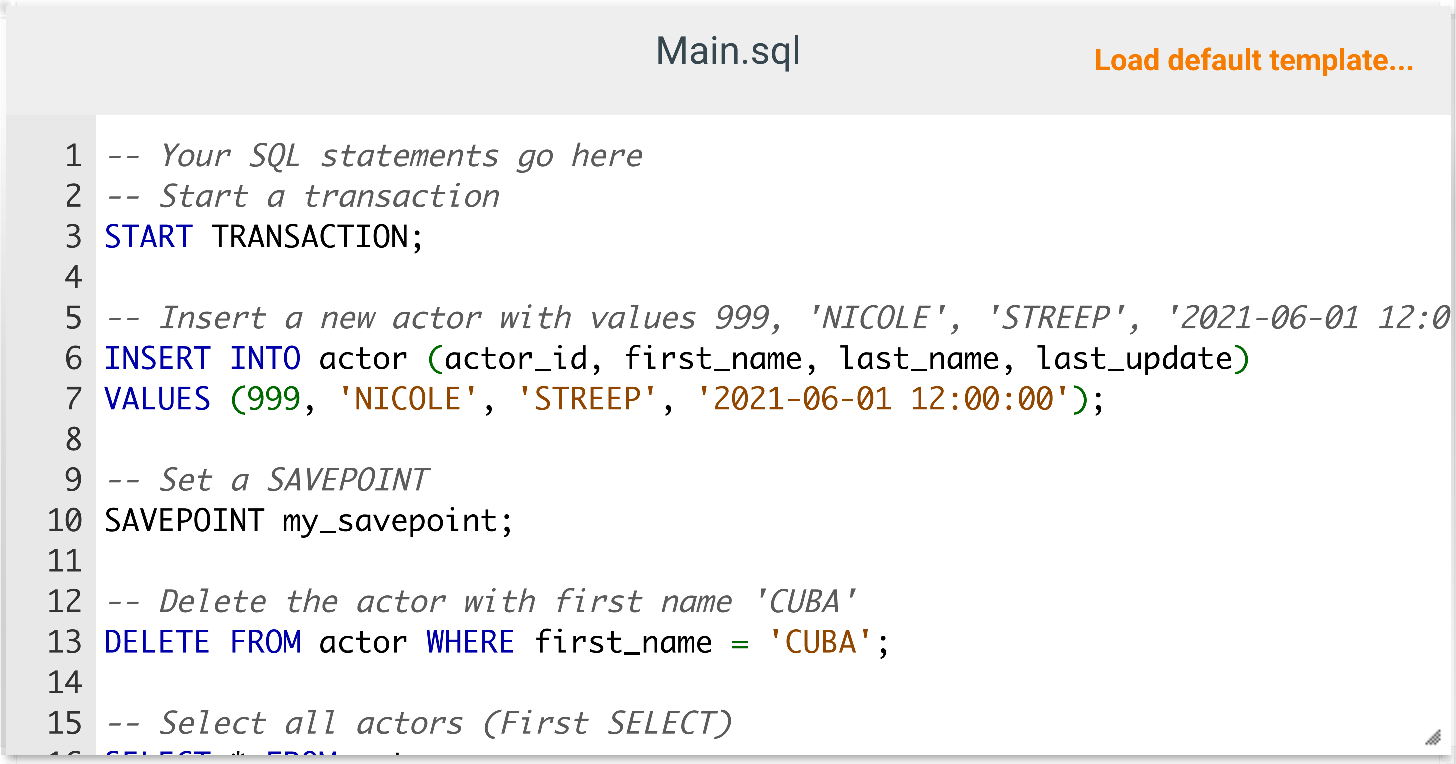
```
DELETE  film_actor
FROM film actor
```

```
INNER JOIN actor ON actor.actor_id = film_actor.actor_id
WHERE actor.first_name = 'CUBA';
```

2) MySQL *safe mode* requires that UPDATE and DELETE statements contain a WHERE clause with an indexed column. Turn safe mode off:

```
SET SQL_SAFE_UPDATES = 0;
```

LAB ACTIVITY 20.3.1: LAB - Rollback and savepoint (Sakila) 10 / 10



```
-- Your SQL statements go here
-- Start a transaction
START TRANSACTION;

-- Insert a new actor with values 999, 'NICOLE', 'STREEP', '2021-06-01 12:0
INSERT INTO actor (actor_id, first_name, last_name, last_update)
VALUES (999, 'NICOLE', 'STREEP', '2021-06-01 12:00:00');

-- Set a SAVEPOINT
SAVEPOINT my_savepoint;

-- Delete the actor with first name 'CUBA'
DELETE FROM actor WHERE first_name = 'CUBA';

-- Select all actors (First SELECT)
```

Develop mode **Submit mode**

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work   What is this?

```
4/24 W10 min:1
```