RDBMS transactions

Agenda

- Transaction Basics
- InnoDB Database State Changes
- InnoDB Isolation Levels
- Transactions and Stored Procedures

Transaction Basics

Transaction Basics

- A database transaction is a unit of work that must be completed by the database - follows the "all-or-nothing" principle

- A database transaction must be ACID (atomic, consistent, isolated, durable)

Transaction Basics

- If all steps in a transaction are successful then the transaction can be successfully committed.

- If a step breaks in a transaction - the transaction can be rolled back

Transaction Basics

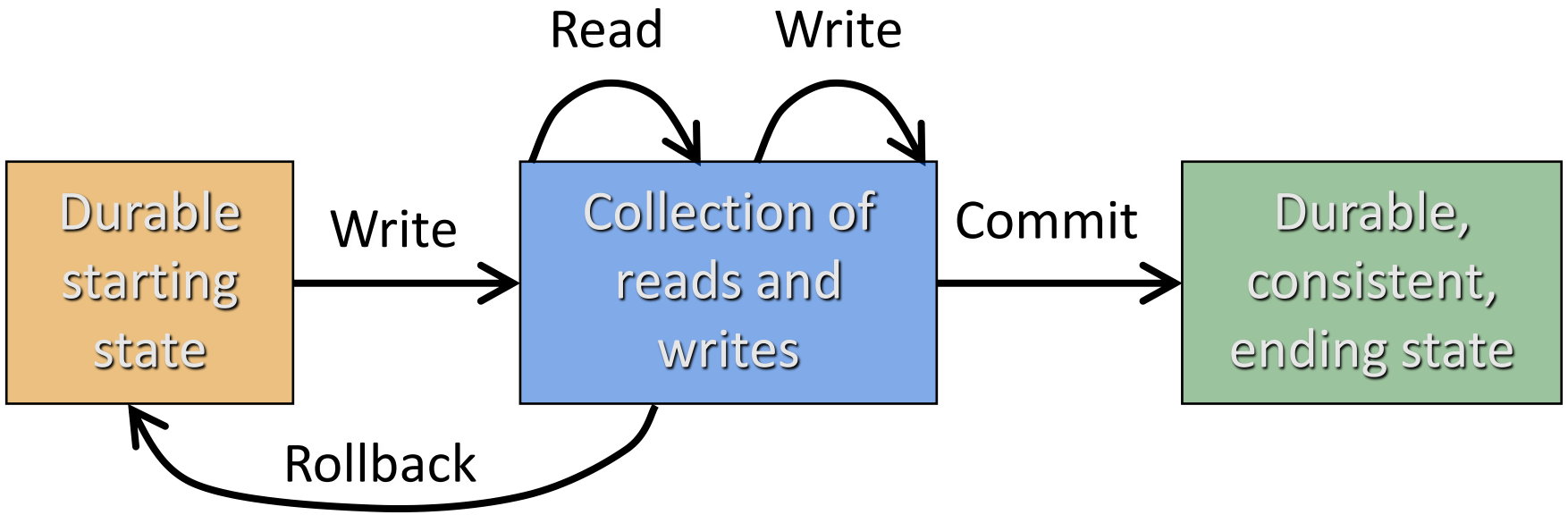
- Transactions are a sequence of actions (database operations) which are executed atomically:
  - either all of them succeed
  - or all of them fail
- Example:

  A bank transfer from one account into another (withdrawal + deposit) - if either the withdrawal or the deposit fails the whole operation is fail

Transaction Basics

- Transactions are typically used one business operations have multiple steps and access one or more tables
- If any of the steps in a business operation fails - the whole operation fails
- Transactions are one of the features of an RDBMS that provide difference between a database and a file system

Transaction Basics



Transaction Basics

- Transactions guarantee the consistency and the integrity of the database:
  - All changes in a transaction are temporary
  - Changes become final when `COMMIT` is executed
  - At any time all changes can be canceled by `ROLLBACK`

Transaction Basics

- A `COMMIT` command commits the current transaction
- A `ROLLBACK` command rollbacks the current transaction
- A `SAVEPOINT` command specifies point in the transaction at which to rollback subsequently

Transaction Basics

- Sample transactions:

Withdraw $100

1. Read current balance
2. New balance = current - 100
3. Write new balance
4. Dispense cash

Transfer $100

1. Read savings
2. New savings = current - 100
3. Read checking
4. New checking = current + 100
5. Write savings
6. Write checking

Transaction Basics

- Transactions are characterized by four major properties:
  - Atomicity - everything completes or fails at once
  - Consistency - the database remains consistent with logically correct data
  - Isolation - different transactions are isolated from each other depending on the selected isolation level
  - Durability - if a transaction is committed changes are durable

Transaction Basics

- Transactions in MySQL begin when:
  - the first executable SQL statement is executed
  - previous transaction commits
- Also known as **implicit transactions**
- A transaction can be started explicitly with the `START TRANSACTION` command

Transaction Basics

- Transactions in MySQL end with one of the following events:
  - `COMMIT` or `ROLLBACK` is issued
  - DDL or DCL statement completes (automatic commit)
  - User exits (automatic commit)
  - System crashes (automatic rollback)

Transaction Basics

- The current user can see the changes before a `COMMIT` or `ROLLBACK` is issued
  - `SELECT` statements return the changed data
- Other users *cannot* view the results of the DML statements made by the current user before a COMMIT or ROLLBACK is issued
  - `SELECT` statements return the old data
- The affected data can be locked - other users cannot change it until the transaction that modifies is committed or rollbacked

Transaction Basics

- Transaction start when a DML command is issued:

```
UPDATE employees SET salary = salary * 1.1
```

- If more DML commands are issued, they are joined to the currently active transaction

```
INSERT INTO emlpoyees(last_name) VALUES('test')
```

- Finally the transaction is committed and all changes are saved

```
COMMIT
```

Transaction Basics

- Data changes are made permanent in the database
- All users can see the results
- Locks on the affected data are released - it is made available for other users to manipulate

Transaction Basics

- Transactions can be canceled by issuing the command

```
ROLLBACK
```

- State of the data after `ROLLBACK`:
  - Data changes are undone
  - Previous state of the data is restored
  - Locks on the affected rows are released

Transaction Basics

- If a single DML statement fails during execution, only that statement is rolled back - all other changes are retained
- The user should terminate transactions explicitly (by executing a `COMMIT` or `ROLLBACK` statement)

Transaction Basics

- Table locks:
  - Prevent destructive interaction between concurrent transactions
  - Require no user action
  - Automatically use the lowest level of restrictiveness
  - Are held for the duration of the transaction
- When user needs a locked data it waits until it is unlocked

Transaction Basics

- When a user locks a record, all other users should wait until it is unlocked
  1. The first user locks the salaries

```
UPDATE employees SET salary = salary * 1.2
```

  2. The second user tries to update the same records and begins to wait (due to locking)

```
UPDATE employees SET salary = salary / 1.2
```

  3. The first user releases the locks

```
ROLLBACK
```

  4. The second user finishes waiting

Transaction Basics

- Most RDBMS support the so-called AUTOCOMMIT mode that runs each query in a separate transaction - this is also the default behaviour in MySQL (so far we didn't have to issue COMMIT after each INSERT/UPDATE or DELETE query)

- In order to disable AUTOCOMMIT mode in MySQL you have to issue:

```
SET AUTOCOMMIT = 0;
```

InnoDB Database State Changes

InnoDB Database State Changes

- State of the database data depends on the MySQL storage engine and the state of the transaction:
  - before `COMMIT`/`ROLLBACK`
  - after `COMMIT`
  - after `ROLLBACK`
  - after `ROLLBACK TO SAVEPOINT`

InnoDB Database State Changes

- Before `COMMIT/ROLLBACK:`
  - Row level locks are performed by InnoDB
  - Query is executed by InnoDB - modifications are stored in redo logs

InnoDB Database State Changes

- After `COMMIT`:
  - A hidden column in each modified table is updated with the corresponding transaction number
  - Data is written to data files
  - MySQL releases locks
  - MySQL marks the transaction complete

InnoDB Database State Changes

- After `ROLLBACK`:
  - Changes made to the transaction are rolled-back using the undo log files
  - Transaction locks for locked data are released
  - The transaction ends

InnoDB Database State Changes

- After `ROLLBACK TO SAVEPOINT:`
  - Only changes made after the savepoint are rolled back
  - The savepoint itself is preserved but all savepoints established after the specified one are lost
  - MySQL releases all table and row locks acquired since that savepoint

InnoDB Isolation Levels

InnoDB Isolation Levels

- Isolation levels define the degree at which changes made from one transaction are visible to other transactions on the same data
- Each RDBMS provides a mechanism for specifying an isolation level for a transaction
- Real-world considerations usually require a compromise between perfect transaction isolation and performance

InnoDB Isolation Levels

- The SQL92 standard defines three phenomena that can occur during concurrently executing transactions:
  - Dirty reads - a transaction reads not-yet-committed data from another transaction
  - Non-repeatable (fuzzy) - a transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data
  - Phantom reads - a transaction re-runs a query and finds that another committed transaction has added additional rows

InnoDB Isolation Levels

- Transactions can define different isolation levels

Level of isolation	Dirty reads	Repeatable reads	Phantom reads
Read uncommitted	yes	yes	yes
Read committed	no	yes	yes
Repeatable read	no	no	yes
Serializable	no	no	no

  - The stronger isolation ensures better consistency but works slower and the data is locked for a longer period of time

InnoDB Isolation Levels

- Isolation levels in InnoDB storage engine:
  - Repeatable read (default) - SELECT statements within a transaction see the same state as in the beginning of the transaction
  - Read uncommited - SELECT statements may read not-yet-commited data from another transaction
  - Read commited - Each SELECT statement reads the changes from the current transaction
  - Strongest isolation level - Each SELECT may block if another transaction has modified the data

InnoDB Isolation Levels

- Types of locks:
  - table
  - row
  - column

InnoDB Isolation Levels

- An isolation level can be set for all transactions in the current or all sessions with the `SET TRANSATION` statement - must be the first statement in a transaction

- Example:

```
SET LOCAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

InnoDB Isolation Levels

- Tables can be explicitly locked with the `LOCK TABLES` command

```
LOCK TABLES <table> <lock_type (e.g. READ or
WRITE)>, [… <table> <lock_type>]
```

- Rows can be explicitly locked with the `SELECT FOR UPDATE` statement

```
SELECT * FROM EMPLOYEES FOR UPDATE
```

- Table level locks are useful for preventing excessive locking on rows or the occurrence of a deadlock

Transactions and Stored Procedures

Transactions and Stored Procedures

- Transactions are not bound to a BEGIN … END block:
  - one block may have multiple transactions
  - one transaction can have multiple blocks
- Some operations are non-transactional (such as writing to a file) - these operations are not reversed when the transaction fails to commit

Transactions and Stored Procedures

- Transactional operations performed in a trigger are atomic in regard to the statement that invoked the trigger:
  - if the statement fails all transactional operations in the trigger are reversed
  - if it commits - all transactional operations in the trigger commit

Questions ?