MySQL Transactions

Sang Shin
www.javapassion.com
"Learning is fun!"

Topics

- What is a transaction?
- ACID properties
- Transaction support in MySQL
- Savepoints
- Transaction isolation levels
- Table locks

What is a Transaction?

What is a Transaction?

- A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit.
- Example: Transfer $100 from Savings account to Checking account is made of two update operations - these two operations need to be performed as a single unit
  - Update Savings table
  - Update Checking table

ACID Properties

ACID Properties

- Atomicity
- Consistency
- Isolation
- Durability

ACID Properties

- Atomicity
  - Ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state
- Consistency
  - Ensures that the database properly changes states upon a successfully committed transactions

ACID Properties

- Isolation
  - Ensures transactions to operate independently of and transparent to each other
- Durability
  - Ensures that the result or effect of a committed transaction persists in case of a system failure

Transaction Support in MySQL

MySQL Support of Transactions

- Only InnoDB storage engine supports transactions
  - Other storage engines ignore transaction statements
- Starting a transaction
  - START TRANSACTION;
- Ending a transaction
  - COMMIT; for committing or
  - ROLLBACK; for rolling back

Transaction COMMIT Example

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO person (person_id, first_name, last_name, age)
    -> VALUES
    -> (11, 'emma', 'kim', 11),
    -> (12, 'jisung', 'park', 22);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.05 sec)
```

Transaction ROLLBACK Example

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO person (person_id, first_name, last_name, age)
    -> VALUES
    -> (11, 'emma', 'kim', 11),
    -> (12, 'jisung', 'park', 22);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> ROLLBACK;
Query OK, 0 rows affected (0.05 sec)
```

Savepoints

Savepoints

- User-defined points within a transaction that can be used for partial roll back
  - Reverse all changes made after the savepoint
- Syntax
  - ROLLBACK TO SAVEPOINT <savepoint-name>

Savepoint Example

```
// Start a new transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

// Insert a new record 31 within the transaction
mysql> INSERT INTO person (person_id, first_name, last_name, age)
    -> VALUES (31, 'hannah', 'song', 15);
Query OK, 1 row affected (0.00 sec)

// Perform SAVEPOINT and name it as "person31"
mysql> SAVEPOINT person31;
Query OK, 0 rows affected (0.00 sec)

// Insert a new record 32
mysql> INSERT INTO person (person_id, first_name, last_name, age)
    -> VALUES (32, 'dadu', 'kim', 25);
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT person31;
Query OK, 0 rows affected (0.00 sec)
```

Transaction Isolation Levels

4 Isolation Levels

- READ UNCOMMITTED
  - Provides lowest level of isolation among transactions but best performing
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE
  - Provides the highest level of isolation among transactions but least performing

READ UNCOMMITTED Isolation level

- Causes 'dirty reads' symptom
  - Uncommitted changes in one transaction (client #1 below) is visible in other transactions (client #2 below)
- Example

Client #1: Start Transaction----INSERT a record A----------ROLLBACK---------->

Client #2: Start Transaction----------(See Record A:Dirty read)----------(Does not see Record A)-->

READ COMMITTED Isolation level

- Committed updates in one transaction (client # 1 below) are visible within another transaction (client #2 below). This means identical queries within a transaction (in client #2 below) can return differing results
- Example

Client #1: Start Transaction----INSERT a record A----------COMMIT---------->

Client #2: Start Transaction----------(Not see Record A)---------(See Record A)------COMMIT------>

REPEATABLE READ Isolation level

- Committed changes in one transaction (client #1 below) is visible in another transaction (client #2 below) only after its own Commit.
  - Within a transaction, all reads are consistent.
- The default isolation level for InnoDB tables.
- Example

Client #1: Start Transaction----INSERT a record A----------COMMIT---------->

Client #2: Start Transaction----------(Not see Record A)----------COMMIT--(See Record A) ---->

SERIALIZABLE Isolation level

- Transactions are serialized
- Until the previous transaction ends via either COMMIT or ROLLBACK, the database operations in other transactions are blocked
- Highest isolation level but not practical in real-life environment
- Example

Client #1: Start Transaction----INSERT a record A----------COMMIT----------->

Client #2: Start Transaction----------(SELECT Blocked)---------- (SELECT returns)--COMMIT---->

Table Locks

When to Use Table Locks?

- For non-InnoDB storage engines, every change is immediately saved to disk - not suitable for multi-user environment where transactional behavior is essential
- Table locks can be used to simulate the transactional properties in non-InnoDB storage engines
- Table locks are not as fined grained as “row lock” or “page (a set of rows) lock”
- Two types of table locks
  - READ, WRITE

READ Table lock

- Anybody can read data from the table
- Nobody can make a change to the table until the lock is released
- Syntax
  - LOCK TABLE person READ;
  - UNLOCK TABLES;

WRITE Table lock

- The client who issued the WRITE table lock can read and make change to the table
- Others cannot either read or write until the lock is released
- Syntax
  - LOCK TABLE person WRITE;
  - UNLOCK TABLES;

Thank you!

Sang Shin
www.javapassion.com
“Learning is fun!”