Lecture 12

Transactions

John Connor

November 28, 2018

Definition

A transaction is a sequence of SQL statements that either all succeed or all fail.

Syntax

To begin a transaction: `BEGIN TRANSACTION;`

Syntax

To begin a transaction: `BEGIN TRANSACTION;`
To end a transaction by making the changes permanent: `COMMIT;`

Syntax

To begin a transaction: `BEGIN TRANSACTION;`
To end a transaction by making the changes permanent: `COMMIT;`
To abort, or undo the changes: `ROLLBACK;`

Motivation: Hardware Failure

In a banking database, if we want to move funds from account 123 to account 456, we might execute the following statements:

```
UPDATE Account
   SET Balance = Balance + 100
 WHERE ID = 456;

UPDATE Account
   SET Balance = Balance - 100
 WHERE ID = 123;
```

Motivation: Hardware Failure

In a banking database, if we want to move funds from account 123 to account 456, we might execute the following statements:

```
UPDATE Account
   SET Balance = Balance + 100
 WHERE ID = 456;

UPDATE Account
   SET Balance = Balance - 100
 WHERE ID = 123;
```

but what happens if there is a power failure after the first statement has executed but before the second statement has executed?

Motivation: Isolation Levels

Say that two users are connected to a database, and each has a single active transaction.

Should a user be able to see the changes the other user is making before they commit their transaction? What if a user makes decisions on the modified data but then the other users aborts the transaction and all of their changes get undone?

Motivation: Isolation Levels

Say that two users are connected to a database, and each has a single active transaction.

Should a user be able to see the changes the other user is making before they commit their transaction? What if a user makes decisions on the modified data but then the other users aborts the transaction and all of their changes get undone?

(Reading uncommitted data from another transaction is called a dirty read.)

Isolation Levels

1. Read / Write (The default isolation level)

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed
5. Read Committed

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed
5. Read Committed
6. Read Only Repeatable Read

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed
5. Read Committed
6. Read Only Repeatable Read
7. Repeatable Read

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed
5. Read Committed
6. Read Only Repeatable Read
7. Repeatable Read
8. Read Only Serializable

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed
5. Read Committed
6. Read Only Repeatable Read
7. Repeatable Read
8. Read Only Serializable
9. Serializable

Isolation Levels

1. Read / Write (The default isolation level)
2. Read Only Read Uncommitted
3. Read Uncommitted
4. Read Only Read Committed
5. Read Committed
6. Read Only Repeatable Read
7. Repeatable Read
8. Read Only Serializable
9. Serializable

A transactions isolation level affects only what the transaction can see, it does not affect what *other* transactions can see.

Isolation Level: Read / Write

When is read / write appropriate?

Isolation Level: Read / Write

When is read / write appropriate?

Is it appropriate for moving funds in a banking system? (With the intent that `Balance` should never be below 0?)

```
CREATE PROCEDURE Withdraw (IN id INT, IN amount INT)
DECLARE balance INT;
BEGIN
  SET balance = SELECT Balance
                FROM Account
                WHERE Account.ID = id;
  IF (amount <= balance) THEN
     UPDATE Account SET Balance = balance - amount
      WHERE Account.ID = id;
  END IF
END;
```

Isolation Level: Read / Write

When is read / write appropriate?

What about a situation in which summary information is inserted every hour?

```
CREATE PROCEDURE Summarize(IN id INT)
DECLARE total INT;
BEGIN
   SET total = SELECT COUNT(*) FROM Account;
   INSERT INTO NumberOfAccount(LAST_QUERY_TIME(), total)
END;
```

Isolation Level: Read Uncommitted

Let T_1, T_2 be transactions with Read Uncommitted isolation levels. If T_1 executes the query `SELECT * FROM Foo;` and sees

A	B	C
7	1	1
1	7	1

and then T_2 executes the queries

```
DELETE FROM Foo WHERE A = 1;
INSERT INTO Foo VALUES (1, 1, 7);
```

what will T_1 see after executing `SELECT * FROM Foo;` again?

Isolation Level: Read Uncommitted

A	B	C
7	1	1
1	1	7

Isolation Level: Read Uncommitted

If T_1 executes

```
INSERT INTO Foo VALUES
(SELECT MAX(A), MAX(B), MAX(C) FROM Foo);
```

and then T_2 executes `ROLLBACK`, what will T_1 see when executing `SELECT * FROM Foo;`?

Isolation Level: Read Uncommitted

If T_1 executes

```
INSERT INTO Foo VALUES
(SELECT MAX(A), MAX(B), MAX(C) FROM Foo);
```

and then T_2 executes `ROLLBACK`, what will T_1 see when executing `SELECT * FROM Foo;`?

A	B	C
7	1	1
1	7	1
7	1	7

Isolation Level: Read Committed

Let T_1, T_2 be transactions with Read Committed isolation levels. If T_1 executes the query `SELECT * FROM Foo;` and sees

A	B	C
7	1	1
1	7	1

and then T_2 executes the queries

```
DELETE FROM Foo WHERE A = 1;
INSERT INTO Foo VALUES (1, 1, 7);
```

what will T_1 see after executing `SELECT * FROM Foo;` again?

Isolation Level: Read Committed

A	B	C
7	1	1
1	7	1

Isolation Level: Read Committed

If T_2 executes `COMMIT`, then what will T_1 see when executing `SELECT * FROM Foo;` again?

Isolation Level: Read Committed

If T_2 executes `COMMIT`, then what will T_1 see when executing `SELECT * FROM Foo;` again?

A	B	C
7	1	1
1	1	7

Isolation Level: Repeatable Read

Let T_1, T_2 be transactions with Repeatable Read isolation levels. If T_1 executes the query `SELECT * FROM Foo;` and sees

A	B	C
1	2	3
4	5	6

and then T_2 executes the queries

```
DELETE FROM Foo WHERE A = 1;
INSERT INTO Foo VALUES (7, 8, 9);
COMMIT;
```

what will T_1 see after executing `SELECT * FROM Foo;` again?

Isolation Level: Repeatable Read

A	B	C
1	2	3
4	5	6
7	8	9

Isolation Level: Repeatable Read

A	B	C
1	2	3
4	5	6
7	8	9

The tuple $(7, 8, 9)$ is called a *phantom tuple*.

Lost Updates

Let T_1, T_2 be "Read Committed" transactions. If the relation Foo is the following

A	B	C
1	2	3
4	5	6

and T_1 and T_2 both execute the query

UPDATE Foo **SET** A = A + 1 **WHERE** A = 1;

followed by `COMMIT`, what will the result be?

Lost Updates

Let T_1, T_2 be "Read Committed" transactions. If the relation Foo is the following

A	B	C
1	2	3
4	5	6

and T_1 and T_2 both execute the query

UPDATE Foo **SET** A = A + 1 **WHERE** A = 1;

followed by `COMMIT`, what will the result be?

A	B	C
2	2	3
4	5	6

Lost Updates

Let T_1, T_2 be "Read Committed" transactions. If the relation Foo is the following

A	B	C
1	2	3
4	5	6

and T_1 and T_2 both execute the query

```
UPDATE Foo SET A = A + 1 WHERE A < 4;
```

followed by `COMMIT`, what will the result be?

Lost Updates

Let T_1, T_2 be "Read Committed" transactions. If the relation Foo is the following

A	B	C
1	2	3
4	5	6

and T_1 and T_2 both execute the query

```
UPDATE Foo SET A = A + 1 WHERE A < 4;
```

followed by `COMMIT`, what will the result be?

Lost Updates

There are at least two possibilites!

Lost Updates

There are at least two possibilites!

A	B	C
2	2	3
4	5	6

A	B	C
3	2	3
4	5	6

Lost Updates

There are at least two possibilites!

A	B	C
2	2	3
4	5	6

A	B	C
3	2	3
4	5	6

but which is the “correct” value?

Lost Updates

In the situation where we end up with the table

A	B	C
2	2	3
4	5	6

we say that we "lost an update". This is not always bad, but the situation *can* be avoided.

Isolation Level: Serializable

Let T_1, T_2 be serializable transactions. If T_1 executes the query `SELECT * FROM Foo;` and sees

A	B	C
1	2	3
4	5	6

and then T_2 executes the queries

```
DELETE FROM Foo WHERE A = 1;
INSERT INTO Foo VALUES (7, 8, 9);
```

what will T_1 see after executing `SELECT * FROM Foo;` again?

Isolation Level: Serializable

A	B	C
1	2	3
4	5	6

Isolation Level: Serializable

A	B	C
1	2	3
4	5	6

If T_2 executes `COMMIT`, what will T_1 see when executing `SELECT * FROM Foo;`?

Isolation Level: Serializable

A	B	C
1	2	3
4	5	6

If T_2 executes `COMMIT`, what will T_1 see when executing `SELECT * FROM Foo;`?

A	B	C
1	2	3
4	5	6

ACID

1. Atomic. All of the statements in a transaction should either succeed, or all of the statements in a transaction should fail.
2. Isolated. No other transaction should be allowed to access or see the intermediate states a transaction creates.
3. Consistent. The data is consistent before the transaction begins, and is in a consistent state after the transaction ends.
4. Durable. A transaction on completion must persist. It should withstand system failures and should not be undone.

Summary

Isolation Level	Dirty Reads	Nonrepeatable Reads	Phantoms
Read Uncommitted	✓	✓	✓
Read Committed	X	✓	✓
Repeatable Read	X	X	✓
Serializable	X	X	X