Concurrency Control

Code examples in MySQL

Database schema & initial population

```
USE db2_schema;
DROP TABLE IF EXISTS TableA;
DROP TABLE IF EXISTS TableB;
CREATE TABLE TableA (
   ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   Val CHAR(1),
   IntVal INT
);
INSERT INTO TableA (Val, IntVal) VALUES ('A', 100), ('B', 300);
CREATE TABLE TableB(
      ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      Val CHAR(1),
      IntVal INT
);
INSERT INTO TableB (Val, IntVal) VALUES ('C', 200), ('D', 400);
```

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

TableB

ID	VAL	INTVAL
1	C	200
2	D	400

Database schema & initial population

```
CREATE TABLE TableC1(
        IntVal1 INT
);

CREATE TABLE TableC2(
        IntVal2 INT
);
```

TableC1

IntVal1
50

TableC2

IntVal2
50

```
INSERT INTO TableC1 (IntVal1)  VALUES ('50');
INSERT INTO TableC2 (IntVal2)  VALUES ('50');
```

Example of Lost Update

Transaction 1

```
USE db2_schema;
SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;

START TRANSACTION;

SELECT @startVal := IntVal from TableA WHERE ID = 1;
SELECT SLEEP(3);
UPDATE TableA SET IntVal = @startVal+1 WHERE ID = 1;
COMMIT;

SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Transaction 2

```
USE db2_schema;
SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;

START TRANSACTION;
UPDATE TableA SET IntVal = IntVal+33 WHERE ID = 1;
COMMIT;

SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Lost Update Execution

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Transaction 1 read TableA → @startVal = 100
Transaction 2 updates TableA → IntVal = 133
Transaction 1 waits
Transaction 1 updates TableA → IntVal = 101

Final State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Example of Dirty Read

Transaction 1

```
USE db2_schema;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;


START TRANSACTION;

UPDATE TableA SET IntVal = IntVal+1000 WHERE ID = 1;

SELECT SLEEP(5);

ROLLBACK;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Transaction 2

```
USE db2_schema;


SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;


START TRANSACTION;

SELECT @Dirtyval:= IntVal, NOW() AS CompletionTime FROM TableA
WHERE ID=1;


UPDATE TableA SET IntVal = @Dirtyval+33 WHERE ID = 1;


COMMIT;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Dirty Read Execution

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Transaction 1 updates TableA → IntVal = 1100
Transaction 2 reads TableA → @DirtyVal = 1100
Transaction 1 does a rollback
Transaction 2 updates TableA → IntVal = 1133

Final State

TableA

ID	VAL	INTVAL
1	A	1133
2	B	300

Example of Dirty Read - fixed

Transaction 1

```
USE db2_schema;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;


START TRANSACTION;

UPDATE TableA SET IntVal = IntVal+1000 WHERE ID = 1;

SELECT SLEEP(5);

ROLLBACK;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Transaction 2

```
USE db2_schema;


SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;


START TRANSACTION;

SELECT @Dirtyval:= IntVal, NOW() AS CompletionTime FROM TableA
WHERE ID=1;


UPDATE TableA SET IntVal = @Dirtyval+33 WHERE ID = 1;


COMMIT;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Dirty Read Execution - fixed

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Transaction 1 updates TableA → IntVal = 1100
Transaction 2 waits on TableA
Transaction 1 does a rollback
Transaction 2 reads TableA → @DirtyVal = 100
Transaction 2 updates TableA → IntVal = 133

Final State

TableA

ID	VAL	INTVAL
1	A	133
2	B	300

Example of Non Repeatable Read

Application 1

```
USE db2_schema;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;


SELECT SLEEP(5);


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Transaction 2

```
USE db2_schema;


START TRANSACTION;

UPDATE TableA SET IntVal = IntVal+33 WHERE ID = 1;

COMMIT;


SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Note: “application” 1 uses no transactions, every SELECT statement executes in its own transaction

Non Repeatable Read Execution

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Transaction 1 reads TableA → IntVal = 100
Transaction 2 updates TableA → IntVal = 133
Transaction 1 reads TableA → IntVal = 133

Final State

TableA

ID	VAL	INTVAL
1	A	133
2	B	300

Example of Non Repeatable Read - fixed

Transaction 1

```
USE db2_schema;
## Default isolation level is REPEATABLE READ
START TRANSACTION;
SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT SLEEP(5);
SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;

COMMIT;
```

Transaction 2

```
USE db2_schema;

START TRANSACTION;
UPDATE TableA SET IntVal = IntVal+33 WHERE ID = 1;
COMMIT;

SELECT IntVal, NOW() AS CompletionTime FROM TableA WHERE ID=1;
```

Note: now application 1 uses a transaction and thus repeatable read isolation (default) applies

Non Repeatable Read Execution - fixed

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Final State

TableA

ID	VAL	INTVAL
1	A	133
2	B	300

Transaction 1 reads TableA → IntVal = 100
Transaction 2 updates TableA → IntVal = 133
Transaction 1 reads TableA <u>from the snapshot</u> → IntVal = 100

Example of Phantom Update

Transaction 1

```
USE db2_schema;


SELECT IF((IntVal1+IntVal2=100), 'YES', 'NO')  as Test, NOW()
AS CompletionTime FROM TableC1,TableC2;


SELECT  @IV1:=IntVal1 FROM TableC1;

SELECT SLEEP(5);

SELECT  @IV2:=IntVal2 FROM TableC2;


SELECT IF((@IV1+@IV2=100), 'YES', 'NO')  as Test, NOW() AS
CompletionTime;
```

Transaction 2

```
USE db2_schema;


START TRANSACTION;


UPDATE TableC1 SET IntVal1 = IntVal1+10;

UPDATE TableC2 SET IntVal2 = IntVal2-10;

COMMIT;


SELECT IF((IntVal1+IntVal2=100), 'YES', 'NO')  as Test, NOW()
AS CompletionTime FROM TableC1, TableC2;
```

Phantom Update Execution

Initial State

TableC1

INTVAL1
50

TableC2

INTVAL2
50

Transaction 1 reads TableC1 → @IV1 = 50
Transaction 2 updates TableC1 → IntVal1 = 60
Transaction 2 updates TableC2 → IntVal2 = 40
Transaction 1 reads TableAC2→ IV2 = 40 (IV1+IV2=90)

Final State

TableC1

INTVAL1
60

TableC2

INTVAL2
40

Example of Phantom Update - fixed

Transaction 1

```
USE db2_schema;

## Default isolation level is REPEATABLE READ

START TRANSACTION;


SELECT IF((IntVal1+IntVal2=100), 'YES', 'NO')  as Test, NOW() AS
CompletionTime FROM TableC1,TableC2;


SELECT  @IV1:=IntVal1 FROM TableC1;

SELECT SLEEP(5);

SELECT  @IV2:=IntVal2 FROM TableC2;


SELECT IF((@IV1+@IV2=100), 'YES', 'NO')  as Test, NOW() AS
CompletionTime;



COMMIT;
```

Transaction 2

```
USE db2_schema;


START TRANSACTION;


UPDATE TableC1 SET IntVal1 = IntVal1+10;

UPDATE TableC2 SET IntVal2 = IntVal2-10;

COMMIT;


SELECT IF((IntVal1+IntVal2=100), 'YES', 'NO')  as Test, NOW()
AS CompletionTime FROM TableC1, TableC2;
```

Phantom Update Execution - fixed

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Transaction 1 reads TableC1 → @IV1 = 50
Transaction 2 updates TableC1 → IntVal1 = 60
Transaction 2 updates TableC2 → IntVal2 = 40
Transaction 1 reads TableAC2→ IV2 = 50 from snapshot
(IV1+IV2=100)

Final State

TableA

ID	VAL	INTVAL
1	A	133
2	B	300

Example of Phantom Insert

Transaction 1

```
USE db2_schema;


SELECT count(*)  as CountA, NOW() AS CompletionTime FROM
TableA;


SELECT SLEEP(5);


SELECT count(*)  as CountA, NOW() AS CompletionTime FROM
TableA;
```

Transaction 2

```
USE db2_schema;


START TRANSACTION;


INSERT INTO TableA (Val, IntVal) VALUES ('X', 500), ('Y', 700);


COMMIT;
```

Note: “application” 1 uses no transactions, every SELECT statement executes in its own transaction

Phantom Insert Execution

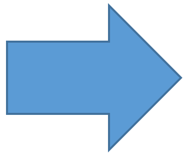
Initial State

TableC1

INTVAL1
50

TableC2

INTVAL2
50



Transaction 1 reads countA = 2
Transaction 2 changes TableA   (inserts 2 tuples)
Transaction 1 reads countA = 4

Final State

TableC1

INTVAL1
60

TableC2

INTVAL2
40

Example of Phantom Insert - fixed

Transaction 1

```
USE db2_schema;

## Default isolation level is REPEATABLE READ

START TRANSACTION;


SELECT count(*)  as CountA, NOW() AS CompletionTime FROM
TableA;


SELECT SLEEP(5);


SELECT count(*)  as CountA, NOW() AS CompletionTime FROM
TableA;



COMMIT;
```

Transaction 2

```
USE db2_schema;


START TRANSACTION;


INSERT INTO TableA (Val, IntVal) VALUES ('X', 500), ('Y', 700);


COMMIT;
```

Now application 1 uses a single transaction
Note that phantoms are avoided also with REPEATABLE READ

Phantom Insert Execution - fixed

Initial State

TableA

ID	VAL	INTVAL
1	A	100
2	B	300

Transaction 1 reads countA = 2
Transaction 2 changes TableA (inserts 2 tuples)
Transaction 1 reads countA = 2 from snapshot

Final State

TableA

ID	VAL	INTVAL
1	A	133
2	B	300

Example of Deadlock

Transaction 1

```
USE db2_schema;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;

START TRANSACTION;
## Keep Xlock for 7 seconds
UPDATE TableA SET Val = 'E' WHERE ID = 1;
SELECT SLEEP(7);

UPDATE TableB SET Val= 'G' WHERE ID = 1;
COMMIT;

SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;
```

Transaction 2

```
USE db2_schema;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;

START TRANSACTION;
## Keep Xlock for 7 seconds
UPDATE TableB SET Val = 'F' WHERE ID = 1;
SELECT SLEEP(7);

UPDATE TableA SET Val = 'H' WHERE ID = 1;
COMMIT;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;
```

Execution

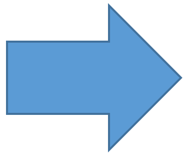
Initial State

TableA

ID	VAL
1	A
2	B

TableB

ID	VAL
1	C
2	D



Transaction 1 updates TableA (A → E)
Transaction 2 updates TableB (C → F)
Transaction 1 waits on TableB
Transaction 2 waits on TableA
Deadlock
Transaction2 is killed
Transaction 1 updates TableB(C → G)

Final State

TableA

ID	VAL
1	E
2	B

TableB

ID	VAL
1	G
2	D

How to see the deadlock

```
SELECT ENGINE_TRANSACTION_ID, THREAD_ID, EVENT_ID,
OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME, LOCK_TYPE,
LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM
performance_schema.data_locks;
```

```
SELECT REQUESTING_ENGINE_TRANSACTION_ID,
REQUESTING_THREAD_ID, REQUESTING_EVENT_ID,
BLOCKING_ENGINE_TRANSACTION_ID, BLOCKING_THREAD_ID,
BLOCKING_EVENT_ID FROM
performance_schema.data_lock_waits;
```

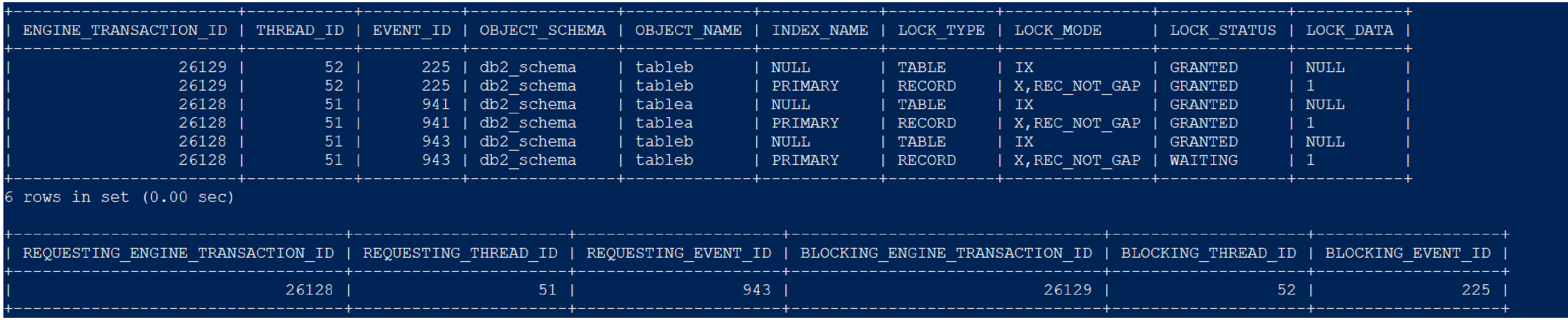
Before the second update of transaction 2

```
UPDATE TableB SET Val = 'F' WHERE ID = 1;

SELECT SLEEP(5);
```

```
SELECT ENGINE_TRANSACTION_ID, THREAD_ID, EVENT_ID, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME,
LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM performance_schema.data_locks;

SELECT REQUESTING_ENGINE_TRANSACTION_ID, REQUESTING_THREAD_ID, REQUESTING_EVENT_ID,
BLOCKING_ENGINE_TRANSACTION_ID, BLOCKING_THREAD_ID, BLOCKING_EVENT_ID FROM
performance_schema.data_lock_waits;
```

```
UPDATE TableA SET Val = 'H' WHERE ID = 1;
```

Herarchical locks and wait-for relation



```
+-----------------------+-----------+----------+---------------+-------------+------------+-----------+---------------+-------------+-----------+
| ENGINE_TRANSACTION_ID | THREAD_ID | EVENT_ID | OBJECT_SCHEMA | OBJECT_NAME | INDEX_NAME | LOCK_TYPE | LOCK_MODE     | LOCK_STATUS | LOCK_DATA |
+-----------------------+-----------+----------+---------------+-------------+------------+-----------+---------------+-------------+-----------+
|                 26129 |        52 |      225 | db2_schema    | tableb      | NULL       | TABLE     | IX            | GRANTED     | NULL      |
|                 26129 |        52 |      225 | db2_schema    | tableb      | PRIMARY    | RECORD    | X,REC_NOT_GAP | GRANTED     | 1         |
|                 26128 |        51 |      941 | db2_schema    | tablea      | NULL       | TABLE     | IX            | GRANTED     | NULL      |
|                 26128 |        51 |      941 | db2_schema    | tablea      | PRIMARY    | RECORD    | X,REC_NOT_GAP | GRANTED     | 1         |
|                 26128 |        51 |      943 | db2_schema    | tableb      | NULL       | TABLE     | IX            | GRANTED     | NULL      |
|                 26128 |        51 |      943 | db2_schema    | tableb      | PRIMARY    | RECORD    | X,REC_NOT_GAP | WAITING     | 1         |
+-----------------------+-----------+----------+---------------+-------------+------------+-----------+---------------+-------------+-----------+
6 rows in set (0.00 sec)

+----------------------------------+----------------------+---------------------+--------------------------------+--------------------+-------------------+
| REQUESTING_ENGINE_TRANSACTION_ID | REQUESTING_THREAD_ID | REQUESTING_EVENT_ID | BLOCKING_ENGINE_TRANSACTION_ID | BLOCKING_THREAD_ID | BLOCKING_EVENT_ID |
+----------------------------------+----------------------+---------------------+--------------------------------+--------------------+-------------------+
|                            26128 |                   51 |                 943 |                          26129 |                 52 |               225 |
+----------------------------------+----------------------+---------------------+--------------------------------+--------------------+-------------------+
```

Example of Update Lock

Transaction 1

```
USE db2_schema;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;

START TRANSACTION;
SELECT ID FROM TableB WHERE ID=1 FOR UPDATE;
UPDATE TableA SET Val = 'E' WHERE ID = 1;
SELECT SLEEP(7);
UPDATE TableB SET Val= 'G' WHERE ID = 1;
COMMIT;

SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;
```

Transaction 2

```
USE db2_schema;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;

START TRANSACTION;
UPDATE TableB SET Val = 'F' WHERE ID = 1;
SELECT SLEEP(7);
UPDATE TableA SET Val = 'H' WHERE ID = 1;
COMMIT;

SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;
```

Execution

Initial State

TableA

ID	VAL
1	A
2	B

TableB

ID	VAL
1	C
2	D

Transaction 1 locks TableB for udpate;
Transaction 1 updates TableA (A → E)
Transaction 2 waits on TableB
Transaction 1 updates TableB(C → G)
Transaction 1 commits
Transaction 2 updates TableB (C → F)
Transaction 2 updates TableA (C → H)

Final State

TableA

ID	VAL
1	F
2	B

TableB

ID	VAL
1	H
2	D

Example of Deadlock –with SERIALIZABLE

Transaction 1

```
USE db2_schema;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;

START TRANSACTION;
## Keep Xlock for 7 seconds
UPDATE TableA SET Val = 'E' WHERE ID = 1;
SELECT SLEEP(7);

UPDATE TableB SET Val= 'G' WHERE ID = 1;
COMMIT;

SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;
```

Transaction 2

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
USE db2_schema;
SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;

START TRANSACTION;
## Keep Xlock for 7 seconds
UPDATE TableB SET Val = 'F' WHERE ID = 1;
SELECT SLEEP(7);

UPDATE TableA SET Val = 'H' WHERE ID = 1;
COMMIT;

SELECT Val, NOW() AS CompletionTime FROM TableA WHERE ID=1;
SELECT Val, NOW() AS CompletionTime FROM TableB WHERE ID=1;
```

Execution

Initial State

TableA

ID	VAL
1	A
2	B

TableB

ID	VAL
1	C
2	D

Transaction 1 updates TableA (A → E)
Transaction 2 updates TableB (C → F)
Transaction 1 waits on TableB
Transaction 2 waits on TableA
Deadlock
Transaction2 is killed
Transaction 1 updates TableB(C → G)

Final State

TableA

ID	VAL
1	E
2	B

TableB

ID	VAL
1	G
2	D

Status inspection queries

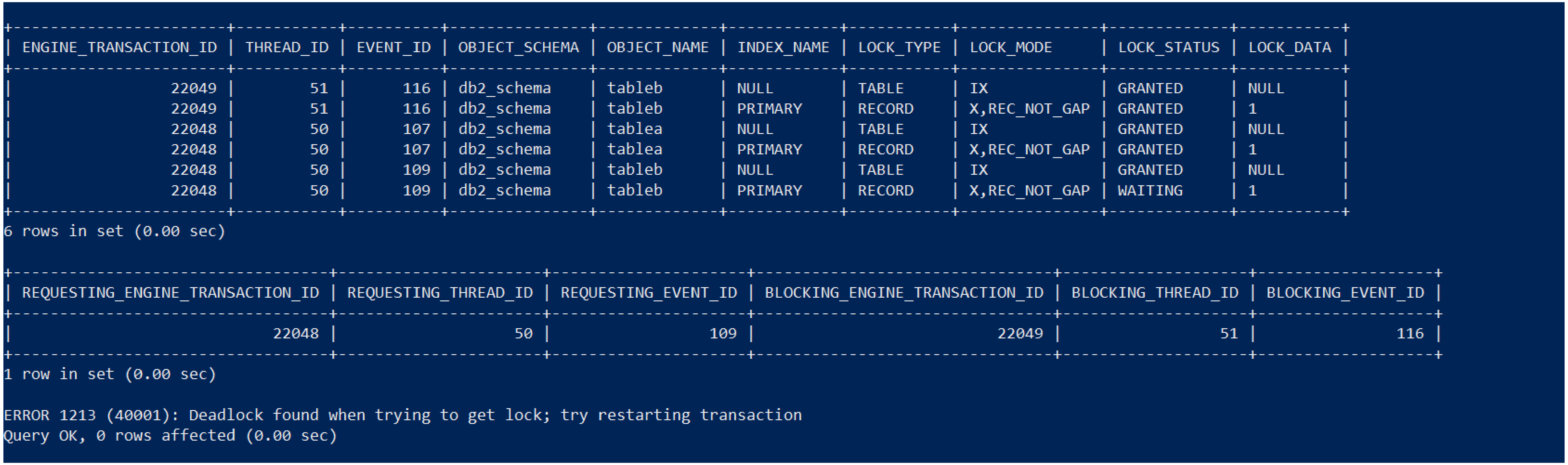
- Lock inspection

```
## Show active locks

SELECT ENGINE_TRANSACTION_ID, THREAD_ID, EVENT_ID, OBJECT_SCHEMA,
OBJECT_NAME, INDEX_NAME, LOCK_TYPE, LOCK_MODE, LOCK_STATUS, LOCK_DATA FROM
performance_schema.data_locks;
```

- Wait-for relationship inspection

```
SELECT REQUESTING_ENGINE_LOCK_ID, REQUESTING_ENGINE_TRANSACTION_ID,
REQUESTING_THREAD_ID, REQUESTING_EVENT_ID, BLOCKING_ENGINE_LOCK_ID,
BLOCKING_ENGINE_TRANSACTION_ID, BLOCKING_THREAD_ID, BLOCKING_EVENT_ID FROM
performance_schema.data_lock_waits;
```

Check locks by yourself on all the examples..



```
+-----------------------+-----------+----------+---------------+-------------+------------+-----------+---------------+-------------+-----------+
| ENGINE_TRANSACTION_ID | THREAD_ID | EVENT_ID | OBJECT_SCHEMA | OBJECT_NAME | INDEX_NAME | LOCK_TYPE | LOCK_MODE     | LOCK_STATUS | LOCK_DATA |
+-----------------------+-----------+----------+---------------+-------------+------------+-----------+---------------+-------------+-----------+
|                 22049 |        51 |      116 | db2_schema    | tableb      | NULL       | TABLE     | IX            | GRANTED     | NULL      |
|                 22049 |        51 |      116 | db2_schema    | tableb      | PRIMARY    | RECORD    | X,REC_NOT_GAP | GRANTED     | 1         |
|                 22048 |        50 |      107 | db2_schema    | tablea      | NULL       | TABLE     | IX            | GRANTED     | NULL      |
|                 22048 |        50 |      107 | db2_schema    | tablea      | PRIMARY    | RECORD    | X,REC_NOT_GAP | GRANTED     | 1         |
|                 22048 |        50 |      109 | db2_schema    | tableb      | NULL       | TABLE     | IX            | GRANTED     | NULL      |
|                 22048 |        50 |      109 | db2_schema    | tableb      | PRIMARY    | RECORD    | X,REC_NOT_GAP | WAITING     | 1         |
+-----------------------+-----------+----------+---------------+-------------+------------+-----------+---------------+-------------+-----------+
6 rows in set (0.00 sec)

+----------------------------------+----------------------+---------------------+--------------------------------+--------------------+-------------------+
| REQUESTING_ENGINE_TRANSACTION_ID | REQUESTING_THREAD_ID | REQUESTING_EVENT_ID | BLOCKING_ENGINE_TRANSACTION_ID | BLOCKING_THREAD_ID | BLOCKING_EVENT_ID |
+----------------------------------+----------------------+---------------------+--------------------------------+--------------------+-------------------+
|                            22048 |                   50 |                 109 |                          22049 |                 51 |               116 |
+----------------------------------+----------------------+---------------------+--------------------------------+--------------------+-------------------+
1 row in set (0.00 sec)

ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
Query OK, 0 rows affected (0.00 sec)
```

Commands

```
INITIALIZATION
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\code\initialize_db.sql
LOST UPDATE
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\Code\lostupdate\LU_transaction1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\Code\lostupdate\LU_transaction2.sql
DIRTY READ
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\Code\dirtyread\DR_transaction1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\Code\dirtyread\DR_transaction2.sql
NON REPEATABLE READ
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_concurrencycontrol\code\nonrepeatableread\nrr_transaction1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_concurrencycontrol\code\nonrepeatableread\nrr_transaction2.sql
PHANTOM UPDATE
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_concurrencycontrol\code\phantomupdate\PU_transaction1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_concurrencycontrol\code\phantomupdate\PU_transaction2.sql
PHANTOM INSERT
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_concurrencycontrol\code\phantominsert\PI_transaction1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_concurrencycontrol\code\phantominsert\PI_transaction2.sql
DEADLOCK
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\code\deadlock\transaction1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\code\deadlock\transaction2.sql
UPDATELOCK
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\Code\updatelock\transaction_uplock_1.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\Code\updatelock\transaction_uplock_2.sql
SERIALIZABLE
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\code\deadlock\transaction1-ser.sql
source C:\Users\Piero\Dropbox\DB2\Lucidi\02_ConcurrencyControl\code\deadlock\transaction2-ser.sql
```