Troubleshooting Locking Issues

Sveta Smirnova
Principal Technical Services Engineer
May, 12, 2016

Table of Contents

Introduction

How locking issues affect your application?

- You run two or more parallel queries
- Each of them executes fast
- But while running in parallel they start performing slowly

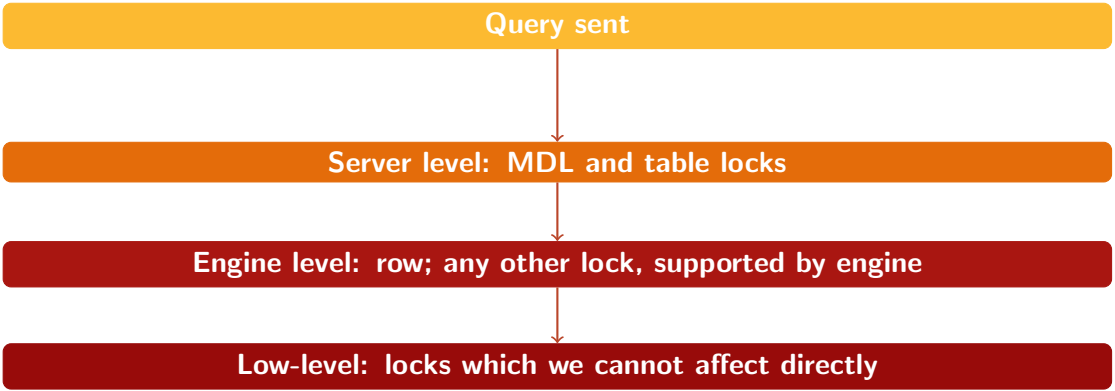
Why fast query runs slow in parallel?

- Locks, set to protect data
  - We can control them directly
- Locks, set to protect access to hardware
  - We can control them only indirectly
  - Will be discussed in the next two webinars

Which locks exist in MySQL?

- Metadata locks (MDL)
- Table locks
- Engine-level locks
  - Row
  - Can be any or not at all

Query: when locks acquired



What do they lock?

- Read locks
  - Block writes
- Write locks
  - Block both reads and writes
- Locks compatibility
  -

	Read	Write
Read	Compatible	Conflict
Write	Conflict	Conflict

InnoDB locks

- Read locks
  - Shared (S)
  - Intention Shared (IS)
- Write locks
  - Exclusive (X)
  - Intention exclusive (IX)
- Locks compatibility

InnoDB locks

- Read locks
- Write locks
- Locks compatibility
  -

	X	IX	S	IS
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

  - See also user manual

Locks and transactions

- Server-level
  - MDL Locks: transactional
  - Table Locks: not transactional
- Engine-level
  - Table Locks
  - Row Locks
  - Fancy Locks

How to diagnose MDL locks

Which queries are affected?

- When acquired?
  - By any access of a table, schema, routine, etc.
    - SELECT/INSERT/UPDATE/DELETE
    - ALTER
    - Any operation!
- When freed?
  - When transaction finishes
    - AUTOCOMMIT=1: when query ends
    - AUTOCOMMIT=0: when COMMIT or ROLLBACK issued
- Locks compatibility
  -

SELECT	DML	DDL
Compatible	Compatible	Conflict

Which thread is waiting for MDL?

- SHOW PROCESSLIST - 5.5+
- I_S.PROCESSLIST - 5.5+
- P_S.THREADS - 5.6+
- P_S.METADATA_LOCKS - 5.7+

Which thread is waiting for MDL?

- SHOW PROCESSLIST - 5.5+

```
mysql> show processlist;
+----+-----+-----------------------------------+--------------------------+
| Id | ... | State                             | Info                     |
+----+-----+-----------------------------------+--------------------------+
|  2 | ... |                                   | NULL                     |
|  3 | ... | Waiting for table metadata lock   | alter table titles ad... |
|  4 | ... | starting                          | show processlist         |
+----+-----+-----------------------------------+--------------------------+
3 rows in set (0,00 sec)
```

Which thread is waiting for MDL?

- I_S.PROCESSLIST - 5.5+

```
mysql> select id, State, Info from information_schema.processlist
    -> where State like '%metadata%'\G
*************************** 1. row ***************************
   id: 3
State: Waiting for table metadata lock
 Info: alter table titles add key(title)
1 row in set (0,00 sec)
```

Which thread is waiting for MDL?

- P_S.THREADS - 5.6+

```
mysql> select thread_id, processlist_id, processlist_state,
    -> processlist_info from performance_schema.threads
    -> where PROCESSLIST_STATE like '%metadata%'\G
*************************** 1. row ***************************
        thread_id: 25
   processlist_id: 3
processlist_state: Waiting for table metadata lock
 processlist_info: alter table titles add key(title)
1 row in set (0,00 sec)
```

Which thread is waiting for MDL?

- P_S.METADATA_LOCKS - 5.7+

```
mysql> select processlist_id, object_type, lock_type, lock_status,
    -> source, processlist_info from metadata_locks
    -> join threads on (owner_thread_id = thread_id) where
    -> object_schema='employees' and object_name='titles' and
    -> lock_status='pending'\G
*************************** 1. row ***************************
  processlist_id: 3
     object_type: TABLE
       lock_type: EXCLUSIVE
     lock_status: PENDING
          source: mdl.cc:3889
processlist_info: alter table titles add key(title)
```

Which thread holds MDL?

```
mysql> select processlist_id pid, object_type, lock_type, lock_status, source
    -> from metadata_locks join threads on (owner_thread_id = thread_id)
    -> where object_schema='employees' and object_name='titles';
+-----+-------------+-------------------+-------------+-------------------+
| pid | object_type | lock_type         | lock_status | source            |
+-----+-------------+-------------------+-------------+-------------------+
|   2 | TABLE       | SHARED_READ       | GRANTED     | sql_parse.cc:5937 |
|   3 | TABLE       | SHARED_UPGRADABLE | GRANTED     | sql_parse.cc:5937 |
|   3 | TABLE       | EXCLUSIVE         | PENDING     | mdl.cc:3889       |
+-----+-------------+-------------------+-------------+-------------------+
3 rows in set (0,00 sec)
```

Possible fixes and best practices

MDL: best practices

- Schedule DDL in less busy time
- Be prepared to rollback transaction
- Keep transactions as small as possible
- Set lock_wait_timeout less than default (1 year) if you cannot afford
  - ALTER to wait forever
  - DML queries to wait when ALTER finishes
  - If set globally this will affect all connections!

How to diagnose Table Locks

When Table Locks are set?

- LOCK TABLE ... READ | WRITE
- Access to table of the engine which does not implement own locks
  - MyISAM
  - Memory
  - Others, which do not have own locking model
- While full table scan is performing on the InnoDB table
  - After query finishes table locks are freed
  - Accessed rows remain locked
  - Mind the difference between table and row locks!

Which thread is waiting for table lock?

- SHOW PROCESSLIST

```
mysql> show processlist;
+----+-----+-----------------------------+---------------------------+
| Id | ... | State                       | Info                      |
+----+-----+-----------------------------+---------------------------+
|  3 | ... | Waiting for table level lock | update emp_myisam set ... |
|  4 | ... | starting                    | show processlist          |
|  6 | ... | Sending data                | SELECT * FROM emp_myisam  |
+----+-----+-----------------------------+---------------------------+
3 rows in set (0,00 sec)
```

- I_S.PROCESSLIST - 5.1+
- P_S.THREADS - 5.6 +
- P_S.TABLE_HANDLES - 5.7+

Which thread is waiting for table lock?

- SHOW PROCESSLIST
- I_S.PROCESSLIST - 5.1+

```
mysql> select id, State, Info from information_schema.processlist
    -> where State like '%table level lock%'\G
*************************** 1. row ***************************
   id: 3
State: Waiting for table level lock
 Info: update emp_myisam set first_name='Steve' where first_name='Sveta'
1 row in set (0,00 sec)
```

- P_S.THREADS - 5.6 +
- P_S.TABLE_HANDLES - 5.7+

Which thread is waiting for table lock?

- SHOW PROCESSLIST
- I_S.PROCESSLIST - 5.1+
- P_S.THREADS - 5.6 +

```
mysql> select thread_id, processlist_id, processlist_state,
    -> processlist_info from performance_schema.threads where
    -> PROCESSLIST_STATE like '%table level lock%'\G
*************************** 1. row ***************************
        thread_id: 25
   processlist_id: 3
processlist_state: Waiting for table level lock
 processlist_info: update emp_myisam set first_name='Sveta' where ...
1 row in set (0,00 sec)
```

- P_S.TABLE_HANDLES - 5.7+

Which thread is waiting for table lock?

- SHOW PROCESSLIST
- I_S.PROCESSLIST - 5.1+
- P_S.THREADS - 5.6 +
- P_S.TABLE_HANDLES - 5.7+

```
mysql> select OBJECT_SCHEMA, OBJECT_NAME, OWNER_THREAD_ID oid,
    -> INTERNAL_LOCK, EXTERNAL_LOCK from table_handles;
+---------------+-------------+-----+---------------+----------------+
| OBJECT_SCHEMA | OBJECT_NAME | OID | INTERNAL_LOCK | EXTERNAL_LOCK  |
+---------------+-------------+-----+---------------+----------------+
| employees     | emp_myisam  |  28 | READ          | READ EXTERNAL  |
| employees     | emp_myisam  |  25 | WRITE         | WRITE EXTERNAL |
+---------------+-------------+-----+---------------+----------------+
```

Which thread holds the lock?

- P_S.TABLE_HANDLES
  - LOCK TABLE ... READ

```
mysql> select * from table_handles\G
*************************** 1. row ***************************
           OBJECT_TYPE: TABLE
         OBJECT_SCHEMA: employees
           OBJECT_NAME: titles
OBJECT_INSTANCE_BEGIN: 140663937105248
      OWNER_THREAD_ID: 28
       OWNER_EVENT_ID: 951
        INTERNAL_LOCK: NULL
        EXTERNAL_LOCK: READ EXTERNAL
```

Which thread holds the lock?

- P_S.TABLE_HANDLES
  - Write to MyISAM table

```
mysql> select * from table_handles\G
*************************** 1. row ***************************
           OBJECT_TYPE: TABLE
         OBJECT_SCHEMA: employees
           OBJECT_NAME: emp
OBJECT_INSTANCE_BEGIN: 140663879605856
      OWNER_THREAD_ID: 26
       OWNER_EVENT_ID: 10419193
        INTERNAL_LOCK: WRITE
        EXTERNAL_LOCK: WRITE EXTERNAL
```

Which thread holds the lock?

- P_S.TABLE_HANDLES
  - Table scan of InnoDB table

```
mysql> select * from table_handles\G
*************************** 1. row ***************************
           OBJECT_TYPE: TABLE
         OBJECT_SCHEMA: employees
           OBJECT_NAME: employees
OBJECT_INSTANCE_BEGIN: 139929095050288
      OWNER_THREAD_ID: 28
       OWNER_EVENT_ID: 10020056
        INTERNAL_LOCK: NULL
        EXTERNAL_LOCK: WRITE EXTERNAL
1 row in set (0,00 sec)
```

Which thread holds the lock?

- P_S.TABLE_HANDLES
- Special case: table scan of InnoDB table
  - During scan

```
mysql> show engine innodb status\G
...
---TRANSACTION 4875, ACTIVE 81 sec fetching rows
mysql tables in use 1, locked 1
887 lock struct(s), heap size 123352, 300910 row lock(s)
MySQL thread id 6, OS thread handle 139930801792768, query id 158
localhost 127.0.0.1 root Sending data
select * from employees for update
...
```

Which thread holds the lock?

- P_S.TABLE_HANDLES
- Special case: table scan of InnoDB table
  - During scan
  - Scan finished, but transaction remains open

```
mysql> show engine innodb status\G
...
---TRANSACTION 4875, ACTIVE 1702 sec
887 lock struct(s), heap size 123352, 300910 row lock(s)
MySQL thread id 6, OS thread handle 139930801792768, query id 158
localhost 127.0.0.1 root cleaning up
...
```

Possible fixes and best practices

Table Locks best practices

- There is no timeout for server table-level locks!
- Use storage engine which supports row-level locking (InnoDB, TokuDB)
- Mix LOCK TABLE and transactions with great care
  - See also user manual
- Tune your queries

How to diagnose InnoDB row locks

InnoDB row locks: when set?

- Shared (S, read) locks
  - SELECT
- Exclusive (X, write) locks
  - DML
- Affected by transaction isolation level!

Which thread waits InnoDB row lock?

- INNODB_LOCK_WAITS

```
mysql> select * from information_schema.innodb_lock_waits\G
*************************** 1. row ***************************
requesting_trx_id: 4903
requested_lock_id: 4903:26:4:2
  blocking_trx_id: 4901
 blocking_lock_id: 4901:26:4:2
1 row in set (0,00 sec)
```

- InnoDB Monitor
- InnoDB Lock Monitor

Which thread waits InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor

```
---TRANSACTION 4903, ACTIVE 7 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1160, 1 row lock(s)
MySQL thread id 9, OS thread handle 139930801792768, query id 261
localhost 127.0.0.1 root updating
update employees set first_name='Sveta' where first_name='Steve'
------- TRX HAS BEEN WAITING 7 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 26 page no 4 n bits 408 index PRIMARY of
table `employees`.`employees` trx id 4903 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 8;
compact format; info bits 0
```

Which thread waits InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor

```
0: len 4; hex 80002711; asc    ' ;;
1: len 6; hex 00000000053c; asc       <;;
2: len 7; hex b2000001260110; asc     &  ;;
3: len 3; hex 8f4322; asc  C";;
4: len 6; hex 47656f726769; asc Georgi;;
5: len 7; hex 466163656c6c6f; asc Facello;;
6: len 1; hex 01; asc  ;;
7: len 3; hex 8f84da; asc     ;;
```

- InnoDB Lock Monitor

Which thread waits InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor
- InnoDB Lock Monitor

```
---TRANSACTION 4908, ACTIVE 1 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1160, 1 row lock(s)
MySQL thread id 9, OS thread handle 139930801792768, query id 290
 localhost 127.0.0.1 root updating
update employees set first_name='Sveta' where emp_no=10001
------- TRX HAS BEEN WAITING 1 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 26 page no 4 n bits 408 index PRIMARY of
table `employees`.`employees` trx id 4908 lock_mode X locks rec
but not gap waiting
```

Which thread waits InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor
- InnoDB Lock Monitor

```
0: len 4; hex 80002711; asc    ' ;;
1: len 6; hex 00000000053c; asc       <;;
2: len 7; hex b2000001260110; asc     &  ;;
3: len 3; hex 8f4322; asc  C";;
4: len 6; hex 47656f726769; asc Georgi;;
5: len 7; hex 466163656c6c6f; asc Facello;;
6: len 1; hex 01; asc  ;;
7: len 3; hex 8f84da; asc    ;;

------------------
```

Which thread waits InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor
- InnoDB Lock Monitor

```
TABLE LOCK table ‘employees‘.‘employees‘ trx id 4908 lock mode IX
RECORD LOCKS space id 26 page no 4 n bits 408 index PRIMARY of
table ‘employees‘.‘employees‘ trx id 4908 lock_mode X locks rec
but not gap waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 8; compact
format; info bits 0
 0: len 4; hex 80002711; asc    ’ ;;
 1: len 6; hex 00000000053c; asc      <;;
 2: len 7; hex b2000001260110; asc     &  ;;
 ...
```

Which thread holds InnoDB row lock?

- INNODB_LOCK_WAITS

```
mysql> select * from information_schema.innodb_lock_waits\G
*************************** 1. row ***************************
requesting_trx_id: 4903
requested_lock_id: 4903:26:4:2
  blocking_trx_id: 4901
 blocking_lock_id: 4901:26:4:2
1 row in set (0,00 sec)
```

- InnoDB Monitor
- InnoDB Lock Monitor

Which thread holds InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor

```
---TRANSACTION 4901, ACTIVE 48 sec
887 lock struct(s), heap size 123352, 300910 row lock(s)
MySQL thread id 8, OS thread handle 139930802591488, query id 255 localhos
```

- InnoDB Lock Monitor

Which thread holds InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor
- InnoDB Lock Monitor

```
---TRANSACTION 4907, ACTIVE 33 sec
2 lock struct(s), heap size 1160, 1 row lock(s)
MySQL thread id 8, OS thread handle 139930802591488, query id 288
localhost 127.0.0.1 root cleaning up
TABLE LOCK table `employees`.`employees` trx id 4907 lock mode IX
RECORD LOCKS space id 26 page no 4 n bits 408 index PRIMARY of
table `employees`.`employees` trx id 4907 lock_mode X locks rec
but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 8;
compact format; info bits 0
```

Which thread holds InnoDB row lock?

- INNODB_LOCK_WAITS
- InnoDB Monitor
- InnoDB Lock Monitor

```
0: len 4; hex 80002711; asc    ' ;;
1: len 6; hex 00000000053c; asc       <;;
2: len 7; hex b2000001260110; asc     &  ;;
3: len 3; hex 8f4322; asc  C";;
4: len 6; hex 47656f726769; asc Georgi;;
5: len 7; hex 466163656c6c6f; asc Facello;;
6: len 1; hex 01; asc  ;;
7: len 3; hex 8f84da; asc    ;;
```

Quick overview of InnoDB locks

- I_S.INNODB_LOCKS
  - Our waiting transaction

```
mysql> select * from information_schema.innodb_locks\G
*************************** 1. row ***************************
    lock_id: 4903:26:4:2
lock_trx_id: 4903
  lock_mode: X
  lock_type: RECORD
 lock_table: `employees`.`employees`
 lock_index: PRIMARY
 lock_space: 26              lock_rec: 2
  lock_page: 4             lock_data: 10001
```

Quick overview of InnoDB locks

- I_S.INNODB_LOCKS
  - Our blocking transaction

```
*************************** 2. row ***************************
    lock_id: 4901:26:4:2
lock_trx_id: 4901
  lock_mode: X
  lock_type: RECORD
 lock_table: `employees`.`employees`
 lock_index: PRIMARY
 lock_space: 26
  lock_page: 4
   lock_rec: 2
  lock_data: 10001
```

Quick overview of InnoDB locks

- I_S.INNODB_LOCKS

- No informaiton about who waits and who holds the lock!

Which transaction isolation level?

- INFORMATION_SCHEMA.INNODB_TRX

```
mysql> select trx_id, trx_state, trx_requested_lock_id,
    -> trx_mysql_thread_id, trx_query, trx_rows_locked,
    -> trx_isolation_level from information_schema.innodb_trx\G
*************************** 1. row ***************************
              trx_id: 5387
           trx_state: RUNNING
trx_requested_lock_id: NULL
  trx_mysql_thread_id: 5
           trx_query: NULL
       trx_rows_locked: 2
  trx_isolation_level: REPEATABLE READ
```

Which transaction isolation level?

- INFORMATION_SCHEMA.INNODB_TRX

```
*************************** 2. row ***************************
                    trx_id: 421220235425984
                 trx_state: RUNNING
trx_requested_lock_id: NULL
  trx_mysql_thread_id: 2
                 trx_query: NULL
       trx_rows_locked: 1
  trx_isolation_level: SERIALIZABLE
2 rows in set (0,00 sec)
```

Which transaction isolation level?

- INFORMATION_SCHEMA.INNODB_TRX
- P_S.variables
  - Not reliable
  - Does not reflect changes, set by SET TRANSACTION ISOLATION LEVEL ...

Which transaction isolation level?

- INFORMATION_SCHEMA.INNODB_TRX
- P_S.variables
  - Not reliable
  - Reflects changes, set by SET tx_isolation:

```
mysql> select * from performance_schema.variables_by_thread
    -> where variable_name='tx_isolation';
+-----------+---------------+-----------------+
| THREAD_ID | VARIABLE_NAME | VARIABLE_VALUE  |
+-----------+---------------+-----------------+
|        24 | tx_isolation  | SERIALIZABLE    |
|        26 | tx_isolation  | REPEATABLE-READ |
|        27 | tx_isolation  | REPEATABLE-READ |
+-----------+---------------+-----------------+
```

Deadlocks can occur...

- When transaction lock multiple rows in table(s), but in opposite order
- When such statements lock ranges of index records or gaps
- Not 100% avoidable with row locking!
- InnoDB can detect most of them
  - In such case it rolls back transaction which modified less data

Deadlocks diagnostics

- InnoDB Monitors
- ```
  ------------------------
  LATEST DETECTED DEADLOCK
  ------------------------
  2016-05-09 03:04:12 0x7f18d9d85700
  *** (1) TRANSACTION:
  TRANSACTION 5411, ACTIVE 13 sec starting index read
  mysql tables in use 1, locked 1
  LOCK WAIT 3 lock struct(s), heap size 1160, 2 row lock(s)
  MySQL thread id 7, OS thread handle 139744709977856, query id 217
  localhost 127.0.0.1 root statistics
  SELECT b FROM t WHERE a=2 FOR UPDATE
  ```
  ```

Deadlocks diagnostics

- InnoDB Monitors
- *** (1) WAITING FOR THIS LOCK TO BE GRANTED:
  RECORD LOCKS space id 0 page no 504 n bits 72 index PRIMARY of
  table ‘test‘.‘t‘ trx id 5411 lock_mode X locks rec but not gap waiting
  Record lock, heap no 2 PHYSICAL RECORD: n_fields 4;
  compact format; info bits 0
   0: len 4; hex 80000002; asc   ;;
   1: len 6; hex 00000000151b; asc   ;;
   2: len 7; hex b90000011f0110; asc   ;;
   3: len 4; hex 80000001; asc   ;;

Deadlocks diagnostics

- InnoDB Monitors
- *** (2) TRANSACTION:
  TRANSACTION 5412, ACTIVE 8 sec starting index read
  mysql tables in use 1, locked 1
  3 lock struct(s), heap size 1160, 2 row lock(s)
  MySQL thread id 8, OS thread handle 139744710776576, query id 218
  localhost 127.0.0.1 root statistics
  SELECT b FROM t WHERE a=1 FOR UPDATE
  *** (2) HOLDS THE LOCK(S):
  RECORD LOCKS space id 0 page no 504 n bits 72 index PRIMARY of
  table ‘test‘.‘t‘ trx id 5412 lock_mode X locks rec but not gap
  Record lock, heap no 2 PHYSICAL RECORD: n_fields 4;
  compact format; info bits 0

Deadlocks diagnostics

- InnoDB Monitors

- ```
  0: len 4; hex 80000002; asc     ;;
  1: len 6; hex 00000000151b; asc       ;;
  2: len 7; hex b90000011f0110; asc        ;;
  3: len 4; hex 80000001; asc     ;;
  ```
  ```

Deadlocks diagnostics

- InnoDB Monitors
- *** (2) WAITING FOR THIS LOCK TO BE GRANTED:
  RECORD LOCKS space id 0 page no 504 n bits 72 index PRIMARY of
  table ‘test‘.‘t‘ trx id 5412 lock_mode X locks rec but not gap waiting
  Record lock, heap no 3 PHYSICAL RECORD: n_fields 4;
  compact format; info bits 0
   0: len 4; hex 80000001; asc   ;;
   1: len 6; hex 00000000151c; asc   ;;
   2: len 7; hex ba000001200110; asc   ;;
   3: len 4; hex 80000002; asc   ;;

  *** WE ROLL BACK TRANSACTION (2)

Deadlocks diagnostics

- InnoDB Monitors
- innodb_print_all_deadlocks - into error log

```
Version: '5.7.11-debug-log'  socket: '/tmp/mysqld.1.sock' port: 13000
2016-05-09T00:06:15.025428Z 8 [Note] InnoDB: Transactions
deadlock detected, dumping detailed information.
2016-05-09T00:06:15.025500Z 8 [Note] InnoDB:
*** (1) TRANSACTION:
TRANSACTION 5415, ACTIVE 21 sec starting index read
mysql tables in use 1, locked 1
...
2016-05-09T00:06:15.031636Z 8 [Note] InnoDB:
*** WE ROLL BACK TRANSACTION (2)
```

Possible fixes and best practices

InnoDB row locks best practices

- Keep transactions small
- Use read-only transactions if you only need to select rows
  - START TRANSACTION READ ONLY
  - AUTOCOMMIT=1
  - Not listed in SHOW ENGINE INNODB STATUS
- Have small innodb_lock_wait_timeout and be prepared to restart transaction

InnoDB deadlocks best practices

- Deadlocks are not avoidable by InnoDB design
- But chance to have them can be minimized by application design
- In any case prepare to restart transaction, failed due to deadlock

Summary

Locks interactions

- MySQL cannot detect conflicts between locks, set on different levels
  - MDL vs row-level
  - Table vs row-level
- ”Deadlock” can occur and will hang forever
  - Only solution is to kill one of transactions

Summary

- Processlist provides quick overview of locked connection threads
  - But not about engine-level locks!
- Detailed information can be found in
  - Performance Schema
  - Information Schema
  - SHOW ENGINE INNODB STATUS
  - Other engine-specific tools

More information

- InnoDB Locking Explained With Stick Figures
- Locks Set by Different SQL Statements in InnoDB
- InnoDB Lock Modes

Place for your questions

???

Thank you!

http://www.slideshare.net/SvetaSmirnova

https://twitter.com/svetsmirnova