Innodb status variables

The Pythian Group
www.pythian.com

Overview

- SHOW ENGINE INNODB STATUS
- Internal Innodb status variables
- Need to know how InnoDB works

Header

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
090904  9:36:09 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 12 seconds
```

Semaphores

From wikipedia: “a semaphore is a protected variable or abstract data type which constitutes the classic **method for restricting access to shared resources** such as shared memory in a parallel programming environment.

http://en.wikipedia.org/wiki/Semaphore_(programming)

Basically: locking.

Semaphores

```
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 32171153, signal count 31011552
Mutex spin waits 0, rounds 20027565614, OS waits 21875962
RW-shared spins 15207459, OS waits 1053752; RW-excl spins 95741267, OS
   waits 2485654
```

- innodb_sync_spin_loops
- innodb_thread_concurrency

Semaphore Examples

```
--Thread 1157658944 has waited at btr/btr0cur.c line 384 for 242.00 seconds the semaphore:
S-lock on RW-latch at 0x2aaab11afd38 created in file dict/dict0dict.c line 1356
a writer (thread id 1105209664) has reserved it in mode exclusive
number of readers 0, waiters flag 1

--Thread 8113 has waited at ibuf0ibuf.c line 366 for 1.00 seconds the semaphore:
S-lock on RW-latch at 67e58d0 created in file dict0dict.c line 3706
number of readers 0, waiters flag 0
Last time read locked in file ibuf0ibuf.c line 366
Last time write locked in file ibuf0ibuf.c line 359

--Thread 8128 has waited at srv0srv.c line 1491 for 0.00 seconds the semaphore:
Mutex at 50cba68 created file srv0srv.c line 872, lock var 1
```

btr0pcur.c	**B-tree / persistent cursor**	**16,720**	**index tree persistent cursor**
ibuf0ibuf.c	**Insert Buffer /**	**91,397**	**Insert buffer**
srv0srv.c	**Server / Server**	**75,633**	**Server main program**

http://forge.mysql.com/wiki/MySQL_Internals_Files_In_InnoDB_Sources

lock0lock.c   trx0trx.c   row0vers.c

LATEST FOREIGN KEY ERROR

```
------------------------
LATEST FOREIGN KEY ERROR
------------------------
090818 10:07:52 Transaction:
TRANSACTION 800FC62A, ACTIVE 0 sec, process no 26213, OS thread id
   1077606720 updating or deleting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1216, 2 row lock(s), undo log entries 1
MySQL thread id 1032146, query id 334809087 web02 localhost test updating
DELETE FROM `db`.`table1` WHERE `id` = "124083"
Foreign key constraint fails for table `db`.`table1`:
CONSTRAINT `ibfk_1` FOREIGN KEY (`id`) REFERENCES `table2` (`t2_id`)
Trying to delete or update in parent table, in index `PRIMARY` tuple:
DATA TUPLE: 12 fields;
 But in child table `db`.`table2` in index `t2_id`, there is a record:
PHYSICAL RECORD: n_fields 4; compact format; info bits 0
```

LATEST DETECTED DEADLOCK

```
*** (1) TRANSACTION:
TRANSACTION 0 2133803100, ACTIVE 0 sec, process no 2246, OS thread id
   46921598638416 starting index read
mysql tables in use 1, locked 1 LOCK WAIT 4 lock struct(s), heap size
   1216, 2 row lock(s)
MySQL thread id 3581917, query id 3269458032 localhost test statistics
SELECT * FROM `tbl1` WHERE (`tbl1`.`id` = 432323)  FOR UPDATE
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 2123005 n bits 240 index `PRIMARY` of
   table `test`.`tbl1` trx id 0 2133803100 lock_mode X locks rec but not
   gap waiting
*** (2) TRANSACTION:
TRANSACTION 0 2133803101, ACTIVE 0 sec, process no 2246, OS thread id
   46921794419024 starting index read, thread declared inside InnoDB 500
UPDATE `tbl2` SET `foo` = 'bar', WHERE `id` = 2891733
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 2123005 n bits 240 index `PRIMARY` of
   table `test`.`tbl1` trx id 0 2133803101 lock_mode X locks rec but not
   gap
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 1073304 n bits 152 index `PRIMARY` of
   table `test`.`tbl2` trx id 0 2133803101 lock_mode X locks rec but not
   gap waiting
*** WE ROLL BACK TRANSACTION (2)
```

TRANSACTIONS

```
------------
TRANSACTIONS
------------
Trx id counter 0 243137723
Purge done for trx's n:o < 0 243137700 undo n:o < 0 0
History list length 25
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 243137707, not started, process no 359, OS thread id
47049732798800
MySQL thread id 31020, query id 27084360 localhost dev_user
SHOW ENGINE INNODB STATUS
---TRANSACTION 0 243085791, not started, process no 359, OS thread id
47049451391312
MySQL thread id 29646, query id 26084726 localhost 127.0.0.1 root
---TRANSACTION 0 243137721, not started, process no 359, OS thread id
47049446906192
MySQL thread id 1962, query id 27084359 Has read all relay log; waiting
for the slave I/O thread to update it
```

The PYTHIAN GROUP™ *your database maestros*

TRANSACTIONS

---TRANSACTION 1 9440456, ACTIVE 292 sec, process no 19041, OS thread id 1160870208 **starting index read, thread declared inside InnoDB 15**

mysql tables in use 2, locked 0

, holds adaptive hash latch

MySQL thread id 745660, query id 1616649038 **db01.company.com 192.168.1.10 dev_user Copying to tmp table**

SELECT tbl1.id FROM tbl1 left join tbl2 on tbl2.id = tbl1.join_fk order by tbl2.field2 DESC, tbl1.field3 DESC

Trx read view will not see trx with id >= 1 9440457, sees < 1 9437436

- innodb_thread_concurrency
- innodb_thread_sleep_delay
- innodb_commit_concurrency
- innodb_concurrency_tickets

FILE I/O

```
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer
  thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
163285 OS file reads, 1701867 OS file writes, 834324 OS fsyncs
0.37 reads/s, 16384 avg bytes/read, 10.84 writes/s, 2.28 fsyncs/s
```

INSERT BUFFER AND ADAPTIVE HASH INDEX

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 5, seg size 7,
36923 inserts, 36924 merged recs, 27673 merges
Hash table size 3187567, node heap has 3949 buffer(s)
2.00 hash searches/s, 17.08 non-hash searches/s
```

An extreme example

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 4073, seg size 4075,
102097 inserts, 102097 merged recs, 72909 merges
Hash table size 25499819, used cells 22802357, node heap has 84495 buffer(s)
8457.31 hash searches/s, 1731.17 non-hash searches/s
```

Innodb_buffer_pool_pages_misc was 11% of the InnoDB buffer pool, which was 12 Gb – meaning there were over 1 Gb of space for the misc pages!

innodb_adaptive_hash_index status variable turns the adaptive hash index off.

LOG

```
---
LOG
---
Log sequence number 334 1812523962
Log flushed up to   334 1812523129
Last checkpoint at  334 1812506173
1 pending log writes, 0 pending chkp writes
742679 log i/o's done, 1.50 log i/o's/second
```

LOG

```
---
LOG
---
Log sequence number 120 4209985949
Log flushed up to 120 4209985949
Last checkpoint at 120 4191865426
0 pending log writes, 0 pending chkp writes
325299 log i/o's done, 1.30 log i/o's/second
```

BUFFER POOL AND MEMORY

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 1808574926; in additional pool allocated 2096896
Dictionary memory allocated 1991080
Buffer pool size   98304
Free buffers       0
Database pages     94074
Modified db pages  565
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 1063693, created 336507, written 2087885
1.00 reads/s, 0.67 creates/s, 0.00 writes/s
Buffer pool hit rate 962 / 1000
```

ROW OPERATIONS

```
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread process no. 359, id 47049446496592, state: flushing log
Number of rows inserted 31473405, updated 36004623, deleted 116956,
  read 455968337
0.33 inserts/s, 0.00 updates/s, 0.33 deletes/s, 0.33 reads/s
```

Feedback

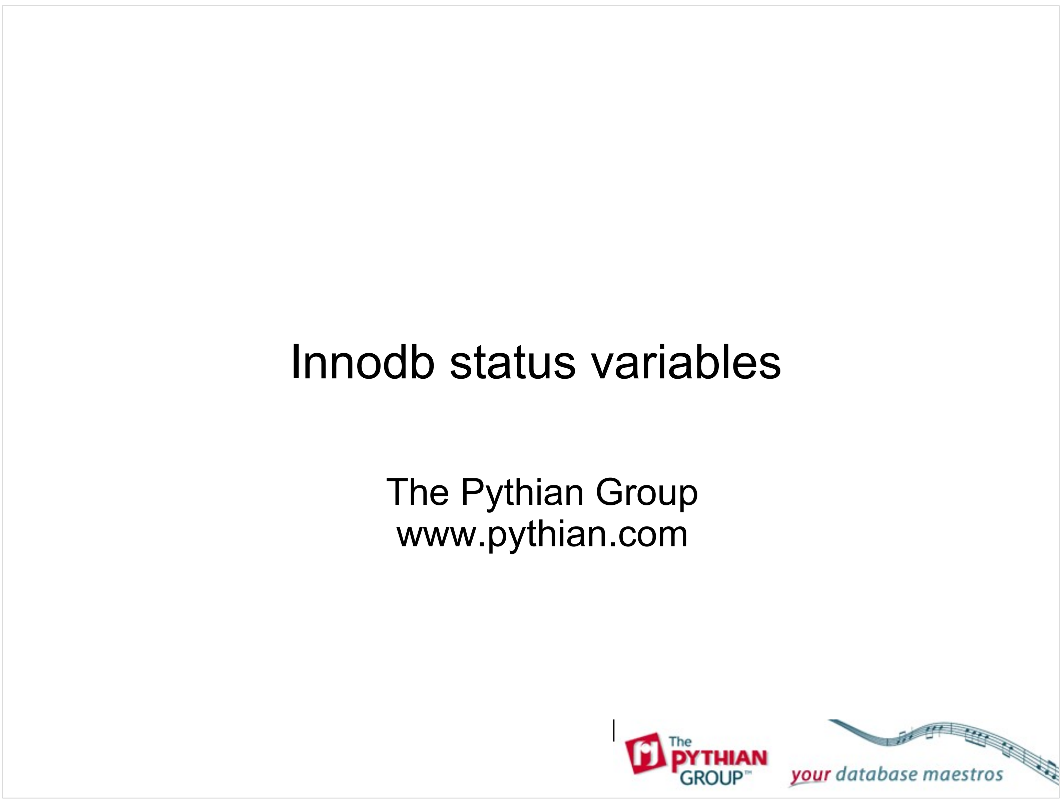
Questions?

Comments?

Suggestions?

Covered the information from SHOW ENGINE INNODB STATUS, and how to tune:

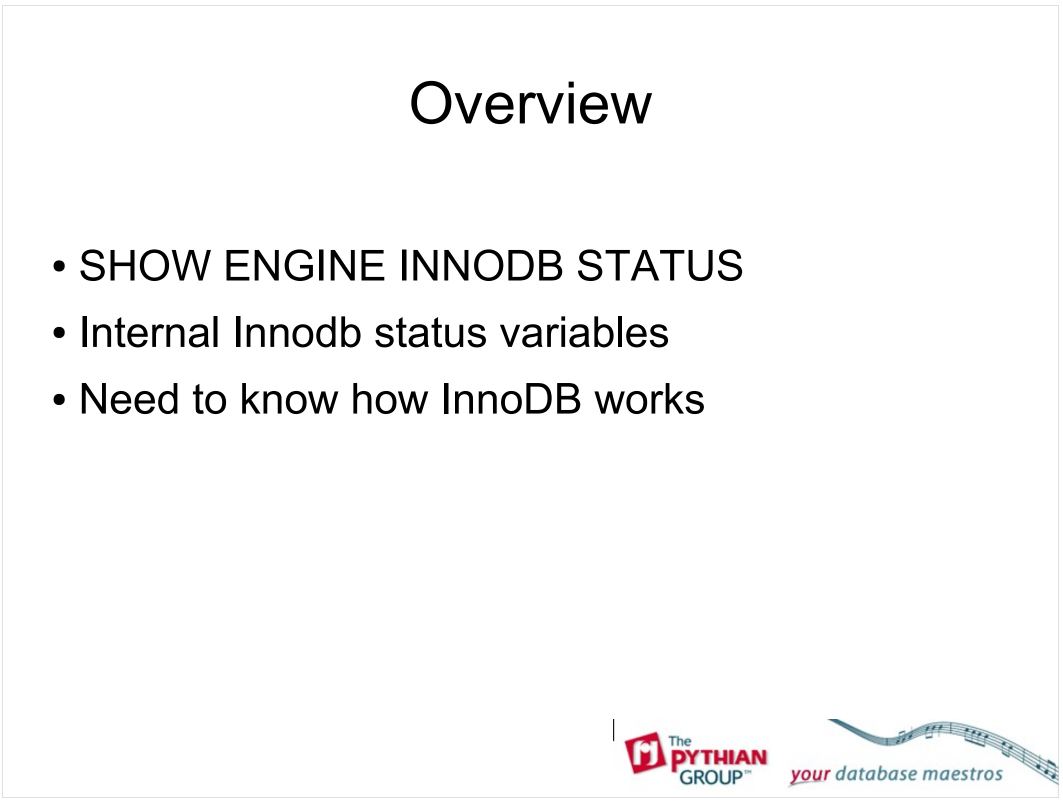
- innodb_thread_concurrency
- innodb_adaptive_hash_index
- innodb_sync_spin_loops
- innodb_thread_sleep_delay
- innodb_commit_concurrency
- innodb_concurrency_tickets



Innodb status variables

The Pythian Group
www.pythian.com

This presentation should be focused on how to prepare application and database server for production and avoid performance issues.

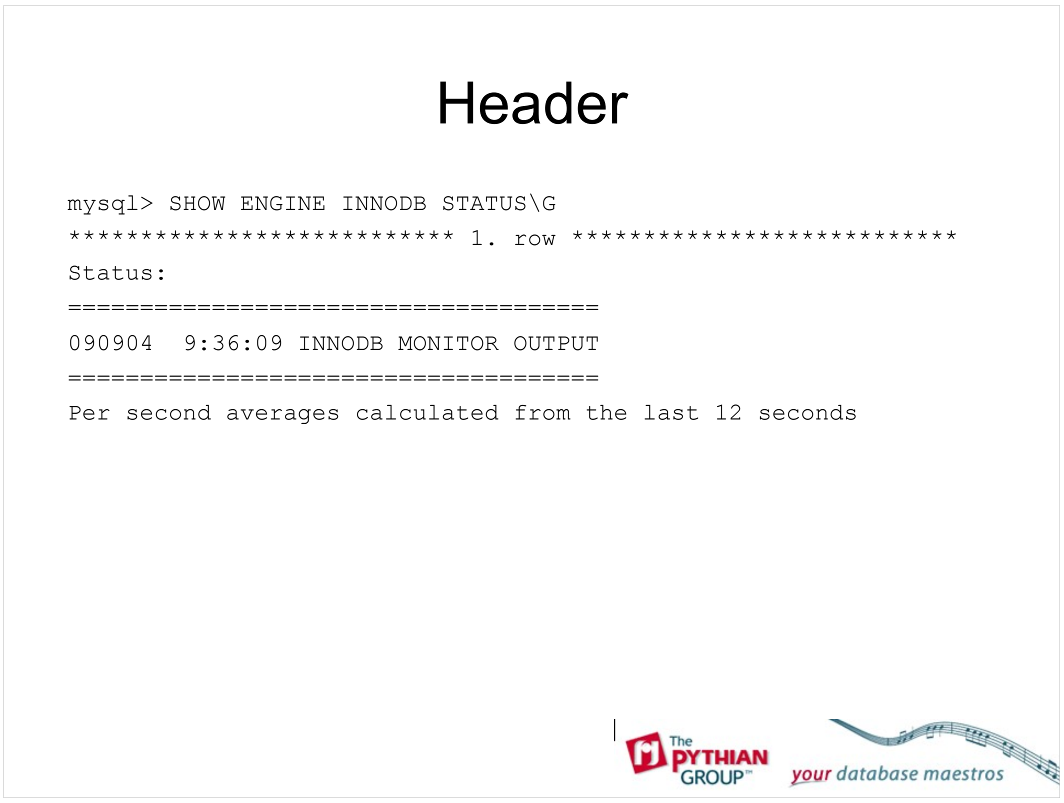


Overview

- SHOW ENGINE INNODB STATUS
- Internal Innodb status variables
- Need to know how InnoDB works

SHOW INNODB STATUS syntax will be deprecated in 5.2

This is for the official release, there's other information in the Percona release – more than 4 writers & readers in File I/O.



Header

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
090904  9:36:09 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 12 seconds
```

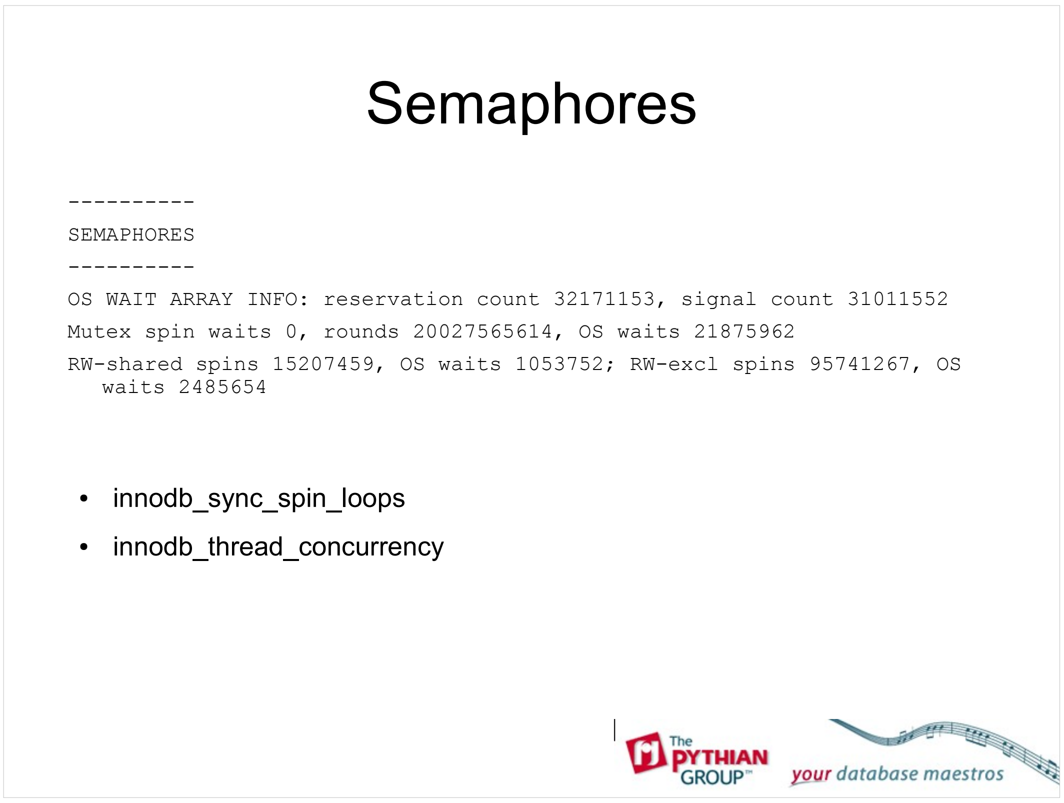
Header is simple and provides current date and time. Also it shows the period during which averages were calculated. User can't control it and usually it is not longer than 1 minute

Semaphores

From wikipedia: “a semaphore is a protected variable or abstract data type which constitutes the classic **method for restricting access to shared resources** such as shared memory in a parallel programming environment.

http://en.wikipedia.org/wiki/Semaphore_(programming)

Basically: locking.



Semaphores

```
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 32171153, signal count 31011552
Mutex spin waits 0, rounds 20027565614, OS waits 21875962
RW-shared spins 15207459, OS waits 1053752; RW-excl spins 95741267, OS
   waits 2485654
```

- innodb_sync_spin_loops
- innodb_thread_concurrency

A large number of threads waiting for semaphores may be lots of disk I/O, or contention problems in InnoDB. Contention can be due to heavy query parallelism or problems in OS thread scheduling. If lots of contention, reduce innodb_thread_concurrency (global, dynamic, 8 by default, 0-1,000). 0=unlimited

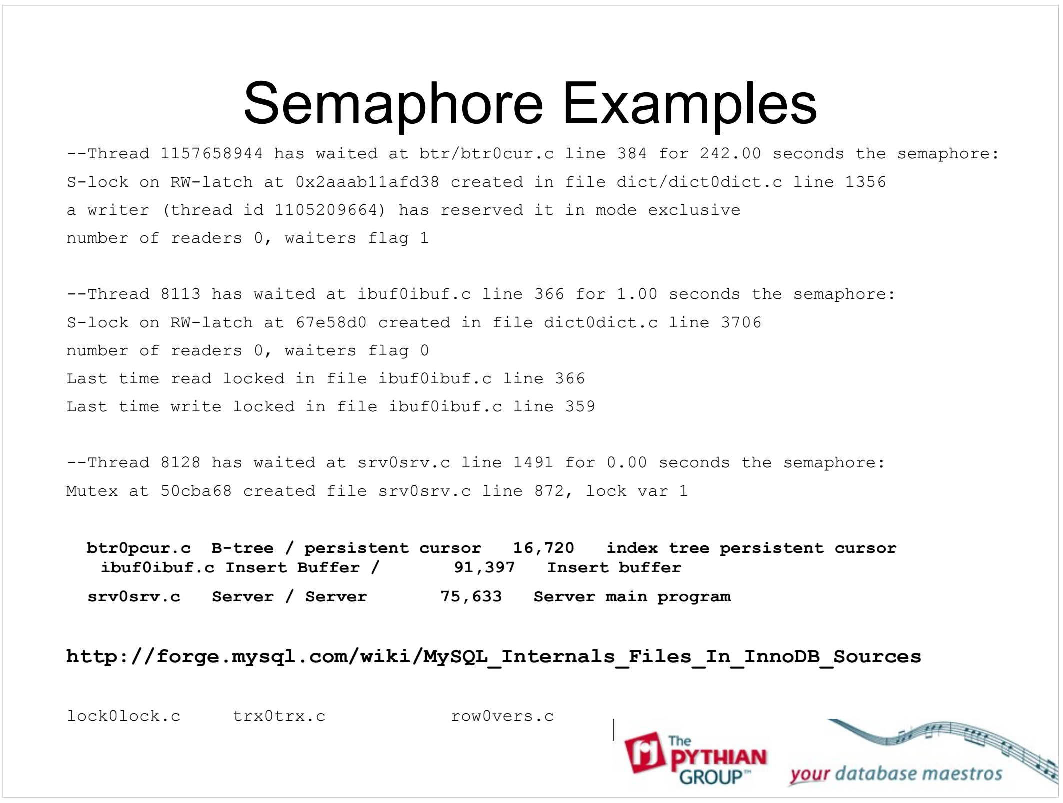
(think of a buffet line, if everyone goes at once it will take a long time).

Innodb has multiphase wait policy. First it tries to use a spin wait – just burns CPU cycles by checking if threads can proceed. Escalates to OS wait, which performs a context switch to let another thread run.

spin waits = more cpu, no context switch vs. OS waits.

innodb_sync_spin_loops “The number of times a thread waits for an InnoDB mutex to be freed before the thread is suspended. The default value is 20.” Global, dynamic, range is 0-4,294,967,295.

OS WAITS ARRAY shows reservations for OS waits vs. how many signals were actually sent to threads.



Semaphore Examples

```
--Thread 1157658944 has waited at btr/btr0cur.c line 384 for 242.00 seconds the semaphore:
S-lock on RW-latch at 0x2aaab11afd38 created in file dict/dict0dict.c line 1356
a writer (thread id 1105209664) has reserved it in mode exclusive
number of readers 0, waiters flag 1

--Thread 8113 has waited at ibuf0ibuf.c line 366 for 1.00 seconds the semaphore:
S-lock on RW-latch at 67e58d0 created in file dict0dict.c line 3706
number of readers 0, waiters flag 0
Last time read locked in file ibuf0ibuf.c line 366
Last time write locked in file ibuf0ibuf.c line 359

--Thread 8128 has waited at srv0srv.c line 1491 for 0.00 seconds the semaphore:
Mutex at 50cba68 created file srv0srv.c line 872, lock var 1
```

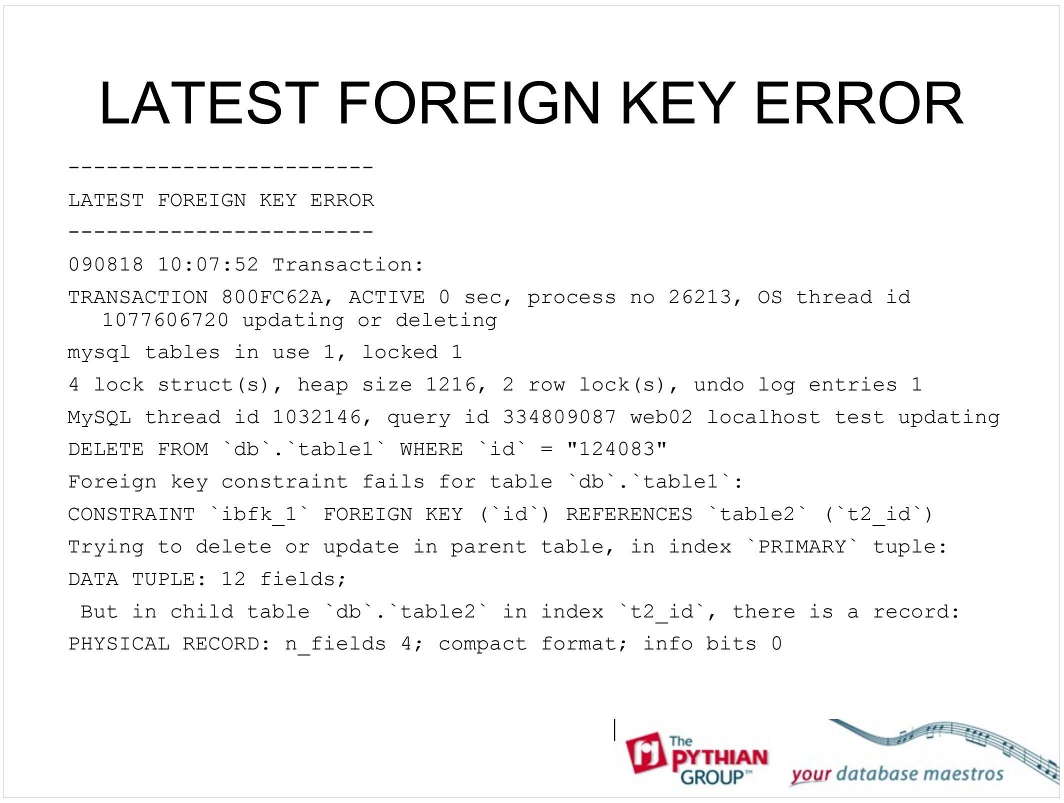
```
  btr0pcur.c  B-tree / persistent cursor   16,720   index tree persistent cursor
   ibuf0ibuf.c Insert Buffer /       91,397   Insert buffer
  srv0srv.c   Server / Server       75,633   Server main program
```

http://forge.mysql.com/wiki/MySQL_Internals_Files_In_InnoDB_Sources

```
lock0lock.c     trx0trx.c           row0vers.c
```

You can figure out what the thread was waiting for based on where the locking is. Go to the source code if you have to, although getting a general idea is usually enough – is it due to waiting for a buffer? waiting for an index write? etc.

In this example, the srv0srv.c was caused by waits seen during long SELECT FOR UPDATE query



LATEST FOREIGN KEY ERROR

```
------------------------
LATEST FOREIGN KEY ERROR
------------------------
090818 10:07:52 Transaction:
TRANSACTION 800FC62A, ACTIVE 0 sec, process no 26213, OS thread id
   1077606720 updating or deleting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1216, 2 row lock(s), undo log entries 1
MySQL thread id 1032146, query id 334809087 web02 localhost test updating
DELETE FROM `db`.`table1` WHERE `id` = "124083"
Foreign key constraint fails for table `db`.`table1`:
CONSTRAINT `ibfk_1` FOREIGN KEY (`id`) REFERENCES `table2` (`t2_id`)
Trying to delete or update in parent table, in index `PRIMARY` tuple:
DATA TUPLE: 12 fields;
 But in child table `db`.`table2` in index `t2_id`, there is a record:
PHYSICAL RECORD: n_fields 4; compact format; info bits 0
```

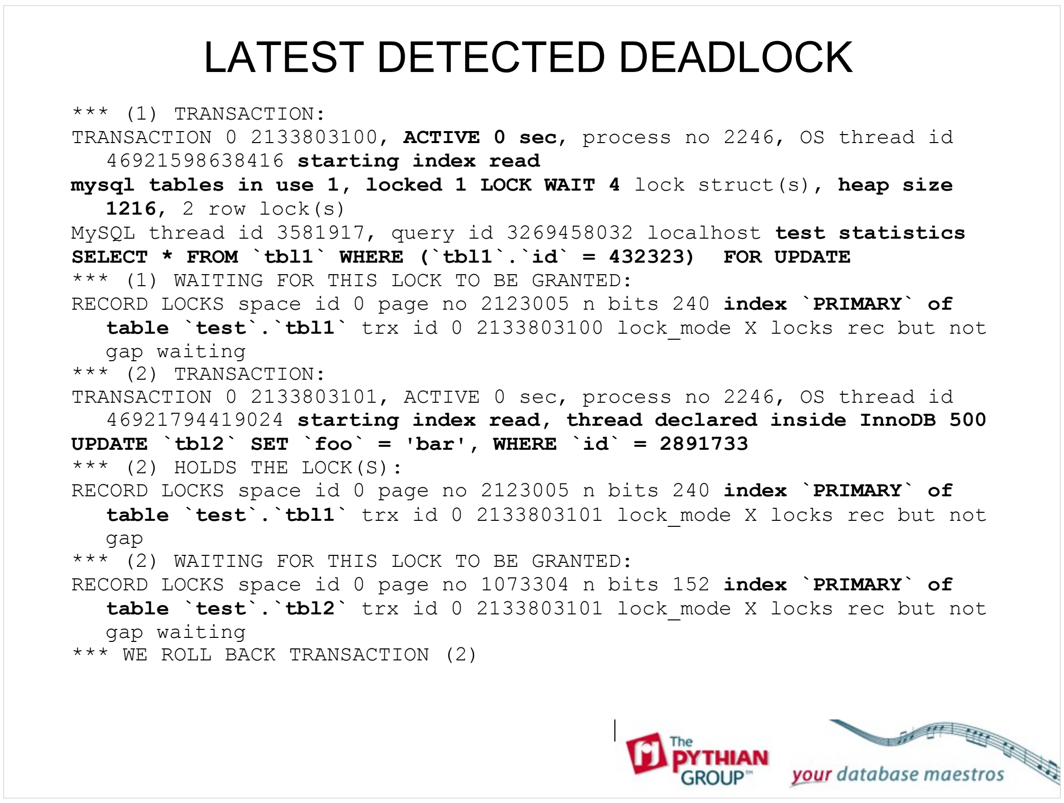
The PYTHIAN GROUP™ your database maestros

This section appears only if there were recent foreign key errors. The output is a bit simplified here and doesn't show actual index data to make it shorter.

There are two cases when foreign key error can happen.

1. When inserting, deleting, updating rows having dependencies in other tables
2. When adding/deleting/changing columns that refer to other tables

Check this section when:
error 150 – foreign key constraint failed
or something like error 1025 – error on rename table



LATEST DETECTED DEADLOCK

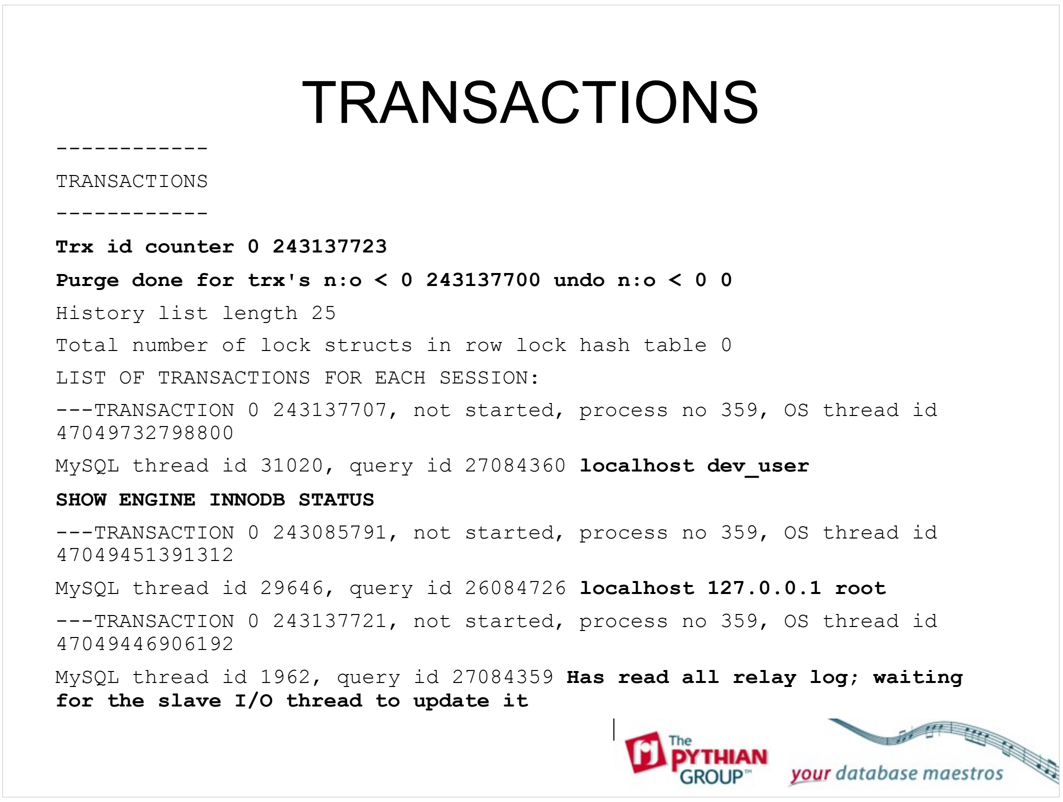
```
*** (1) TRANSACTION:
TRANSACTION 0 2133803100, ACTIVE 0 sec, process no 2246, OS thread id
   46921598638416 starting index read
mysql tables in use 1, locked 1 LOCK WAIT 4 lock struct(s), heap size
   1216, 2 row lock(s)
MySQL thread id 3581917, query id 3269458032 localhost test statistics
SELECT * FROM `tbl1` WHERE (`tbl1`.`id` = 432323)  FOR UPDATE
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 2123005 n bits 240 index `PRIMARY` of
   table `test`.`tbl1` trx id 0 2133803100 lock_mode X locks rec but not
   gap waiting
*** (2) TRANSACTION:
TRANSACTION 0 2133803101, ACTIVE 0 sec, process no 2246, OS thread id
   46921794419024 starting index read, thread declared inside InnoDB 500
UPDATE `tbl2` SET `foo` = 'bar', WHERE `id` = 2891733
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 2123005 n bits 240 index `PRIMARY` of
   table `test`.`tbl1` trx id 0 2133803101 lock_mode X locks rec but not
   gap
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 1073304 n bits 152 index `PRIMARY` of
   table `test`.`tbl2` trx id 0 2133803101 lock_mode X locks rec but not
   gap waiting
*** WE ROLL BACK TRANSACTION (2)
```

The PYTHIAN GROUP™ your database maestros

Shows only two last transactions involved and last two statements in each transaction
Output may be too large to display, because it shows row data as well.

Heap size is amount of memory used to hold row locks

InnoDB rolled back the transaction which had done fewest updates.



TRANSCRIPTIONS

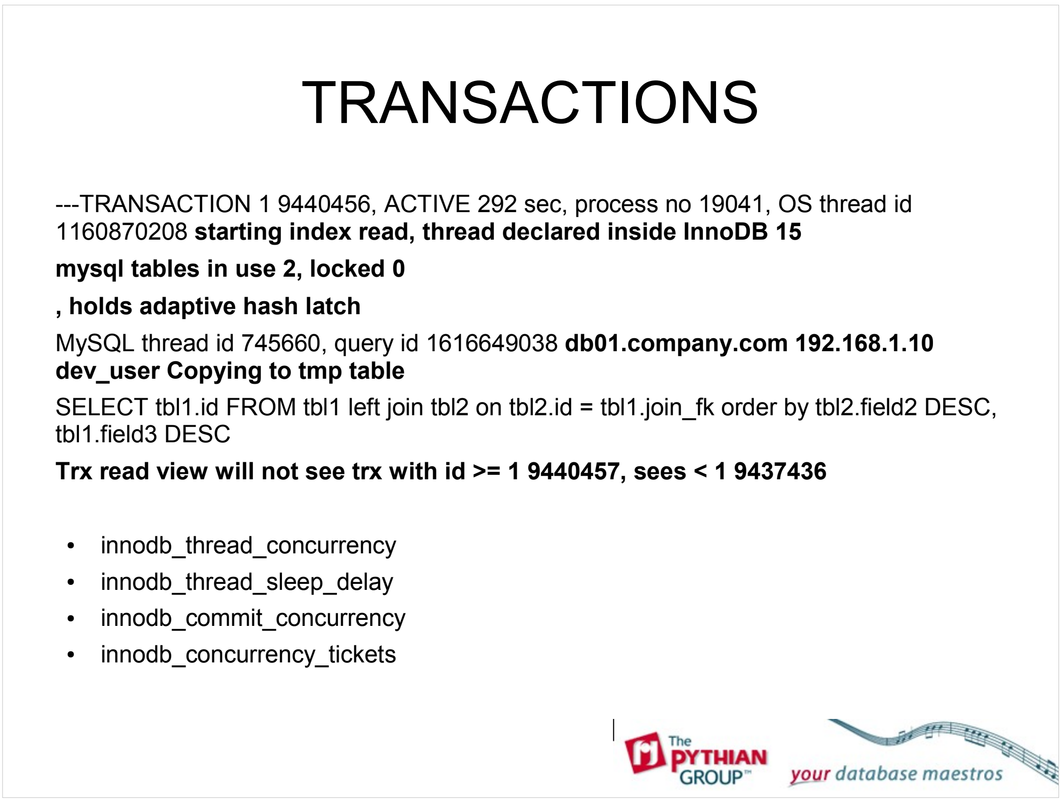
```
------------
TRANSACTIONS
------------
Trx id counter 0 243137723
Purge done for trx's n:o < 0 243137700 undo n:o < 0 0
History list length 25
Total number of lock structs in row lock hash table 0
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 243137707, not started, process no 359, OS thread id
47049732798800
MySQL thread id 31020, query id 27084360 localhost dev_user
SHOW ENGINE INNODB STATUS
---TRANSACTION 0 243085791, not started, process no 359, OS thread id
47049451391312
MySQL thread id 29646, query id 26084726 localhost 127.0.0.1 root
---TRANSACTION 0 243137721, not started, process no 359, OS thread id
47049446906192
MySQL thread id 1962, query id 27084359 Has read all relay log; waiting
for the slave I/O thread to update it
```

MVCC in play with purging – purging of versions from undo area happens when no current transaction is using that version (because if they are, they might need the info inside it). Make sure to commit once you're done, otherwise you are needlessly allowing the data to remain, and purge does not happen. Subtract current txn # and last purged txn # to get difference.

innodb_max_purge_lag – default is 0, no delay. Increase to put some lag in DML statements, so purge has time to run.

"undo" shows the transaction whose latest version is being purged, or 0 if there is no active purge.

- History = the # of unpurged transactions in undo area
- OS thread id can be used when debugging with gdb or lsof or other tools.
- lock struct = possibly many rows per lock struct.



TRANSACTIONS

---TRANSACTION 1 9440456, ACTIVE 292 sec, process no 19041, OS thread id 1160870208 **starting index read, thread declared inside InnoDB 15**

mysql tables in use 2, locked 0

, holds adaptive hash latch

MySQL thread id 745660, query id 1616649038 **db01.company.com 192.168.1.10 dev_user Copying to tmp table**

SELECT tbl1.id FROM tbl1 left join tbl2 on tbl2.id = tbl1.join_fk order by tbl2.field2 DESC, tbl1.field3 DESC

Trx read view will not see trx with id >= 1 9440457, sees < 1 9437436

- innodb_thread_concurrency
- innodb_thread_sleep_delay
- innodb_commit_concurrency
- innodb_concurrency_tickets

The PYTHIAN GROUP™ your database maestros

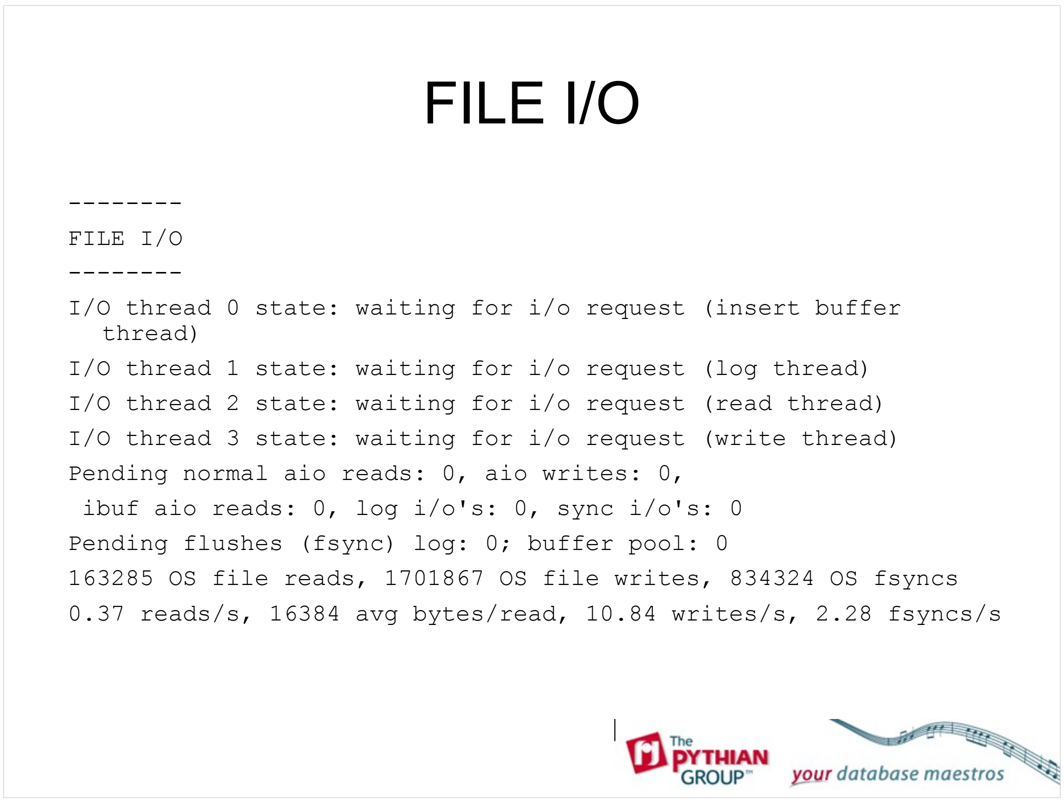
The thread is in the InnoDB kernel and has 15 tickets left to use.
innodb_concurrency_tickets – # times a thread can enter InnoDB without having to be queued if the threads exceed the value of innodb_thread_concurrency, default 500

MVCC – Repeatable read!
States for transactions: not started, fetching rows, updating, etc.
active (even if thread is sleeping, b/c it could be part of a multi-query transaction).

innodb_thread_concurrency limits how many threads in Innodb kernel, statuses for threads not in kernel:
waiting in InnoDB queue. sleeping before joining InnoDB queue

If no free slot for a new thread, InnoDB makes a thread sleep for a small amount of time before entering the kernel. This is the status variable innodb_thread_sleep_delay (value is in microsends, 10,000 by default, or 0.1 seconds).
# tables locked = table level locking, usually 0

innodb_commit_concurrency – 0 is for “unlimited”, which is also the default. basically how many simultaneous commits. dynamic, global, but can't change from 0 to nonzero and vice versa at runtime, 5.1.36 and later.



FILE I/O

```
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer
  thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
163285 OS file reads, 1701867 OS file writes, 834324 OS fsyncs
0.37 reads/s, 16384 avg bytes/read, 10.84 writes/s, 2.28 fsyncs/s
```

The PYTHIAN GROUP™ your database maestros

If any of these aren't waiting for an I/O request, it means they is contention for some resource – usually it is I/O. (or if there are things pending)

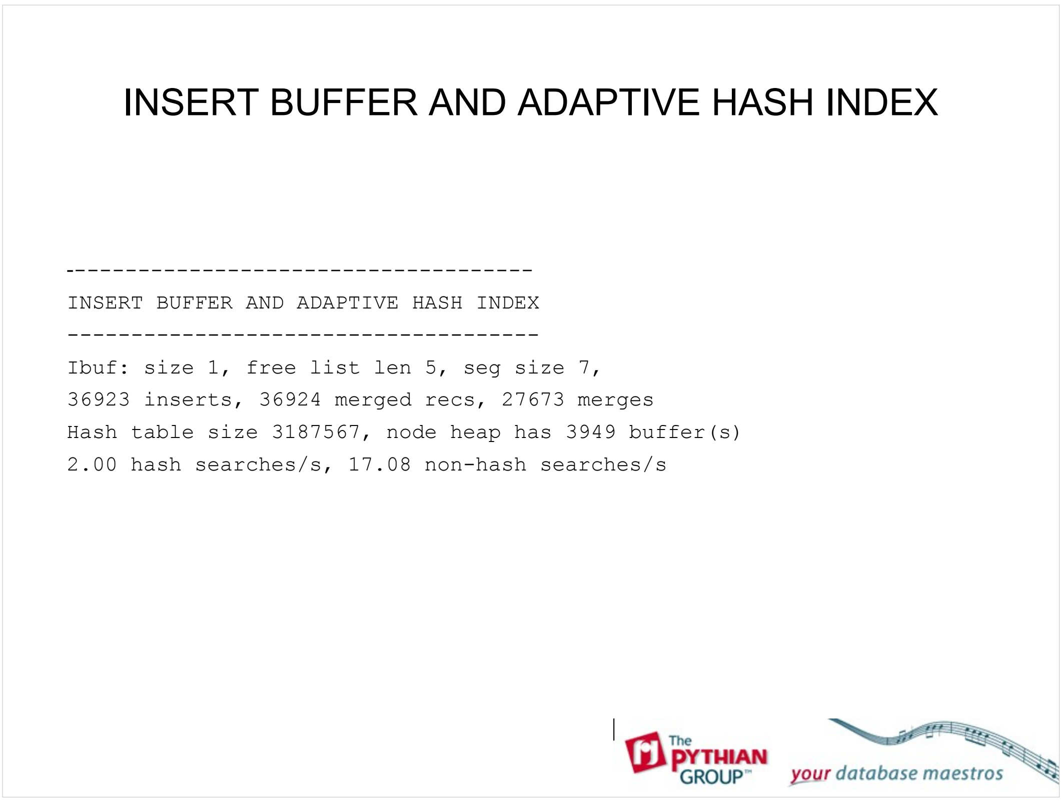
Insert buffer thread – does insert buffer merges (records being merged from buffer into the tablespace)

Log thread – does asynchronous log flushes

Read thread – does read-ahead to prefetch data InnoDB thinks it will need soon

Write thread – flushes dirty buffers

“On Unix, the number of threads is always 4. On Windows, the number depends on the setting of the innodb_file_io_threads system variable.”



INSERT BUFFER AND ADAPTIVE HASH INDEX

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 5, seg size 7,
36923 inserts, 36924 merged recs, 27673 merges
Hash table size 3187567, node heap has 3949 buffer(s)
2.00 hash searches/s, 17.08 non-hash searches/s
```

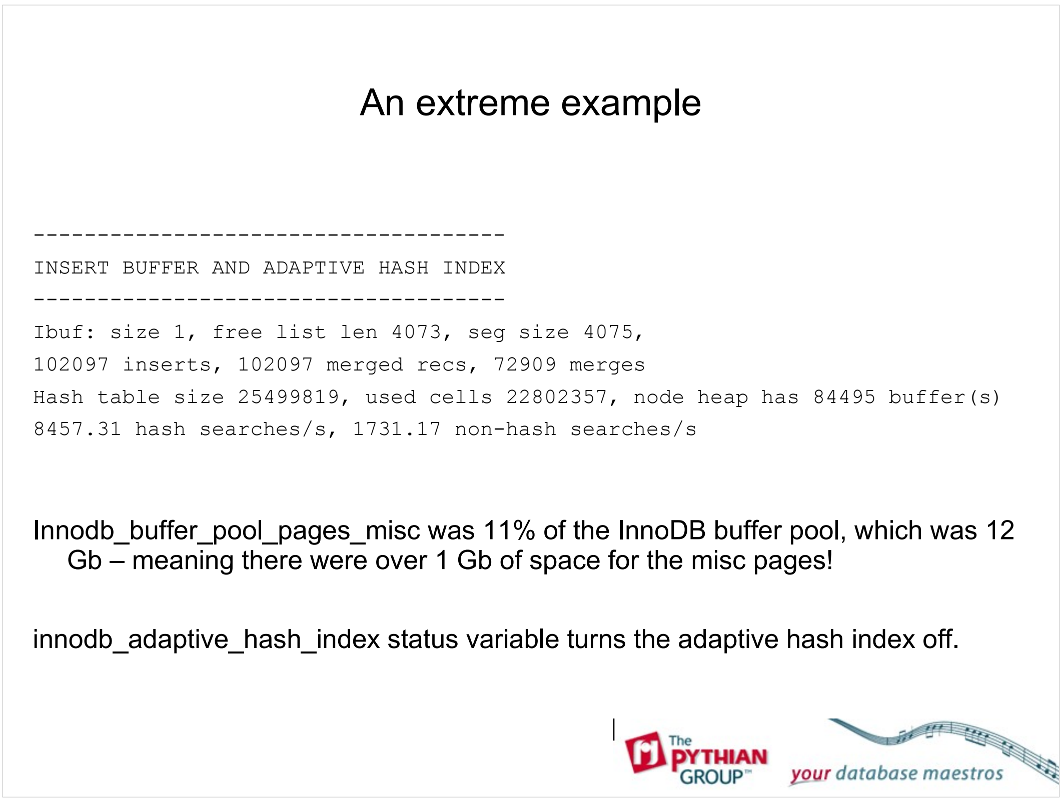
When inserting into non-unique secondary indexes InnoDB may put records into insert buffer and flush them to disk after transaction commits

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes. InnoDB has a mechanism that monitors index searches made to the indexes defined for a table. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.

InnoDB builds hash indexes on demand for those pages of the index that are often accessed.

The ration of merges to inserts indicates how efficient insert buffer is.

The ratio of hash lookups to non-hash looks shows how efficient adaptive hash index is



An extreme example

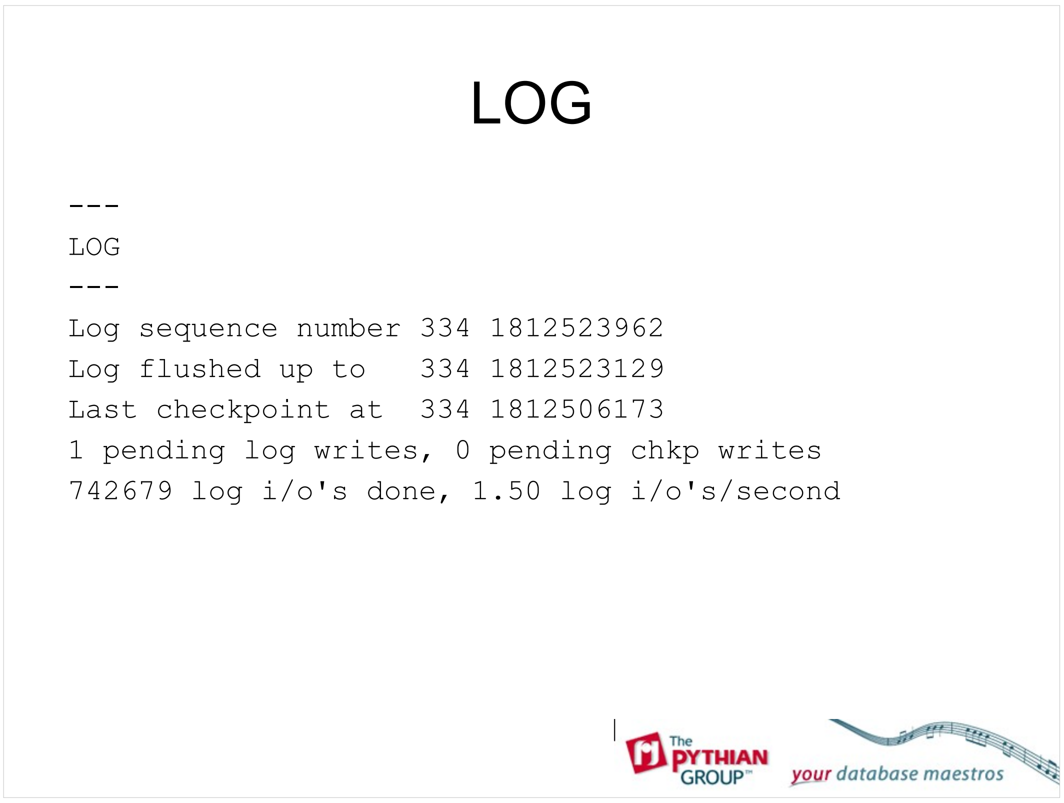
```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 4073, seg size 4075,
102097 inserts, 102097 merged recs, 72909 merges
Hash table size 25499819, used cells 22802357, node heap has 84495 buffer(s)
8457.31 hash searches/s, 1731.17 non-hash searches/s
```

Innodb_buffer_pool_pages_misc was 11% of the InnoDB buffer pool, which was 12 Gb – meaning there were over 1 Gb of space for the misc pages!

innodb_adaptive_hash_index status variable turns the adaptive hash index off.

innodb_adaptive_hash_index is global, static (not dynamic). Bug before 5.0.52, can turn it off as of 5.0.52, bug fixed in 5.0.54.

careful if you're running 5.0.45!



LOG

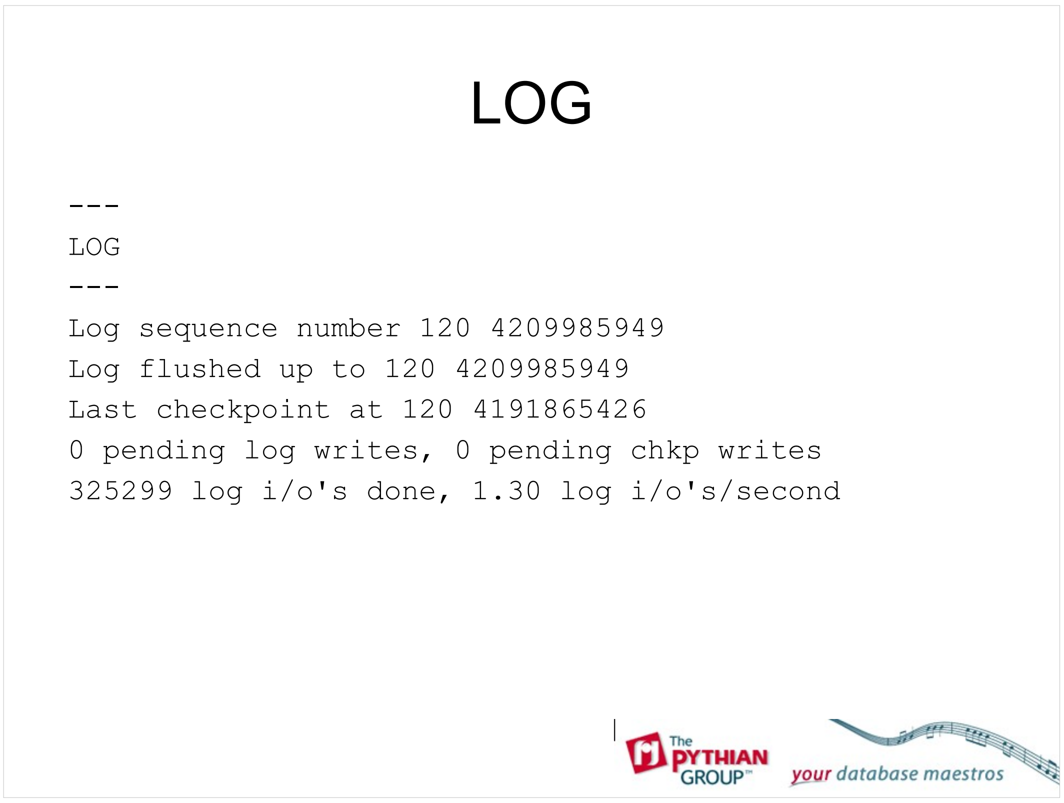
```
---
LOG
---
Log sequence number 334 1812523962
Log flushed up to   334 1812523129
Last checkpoint at  334 1812506173
1 pending log writes, 0 pending chkp writes
742679 log i/o's done, 1.50 log i/o's/second
```

The values are numbers in bytes, representing the log sequence number. You can see how much unflushed data there is and how much data there is since the last checkpoint.

Checkpoint -- marks a state when data and log file were in known state and can be used for recovery

unflushed data = (1812523962-1812523129)/1024 = 0.8135 Kb
last checkpoint was (1812506173-1812523129)/1024 16.5586 Kb ago.

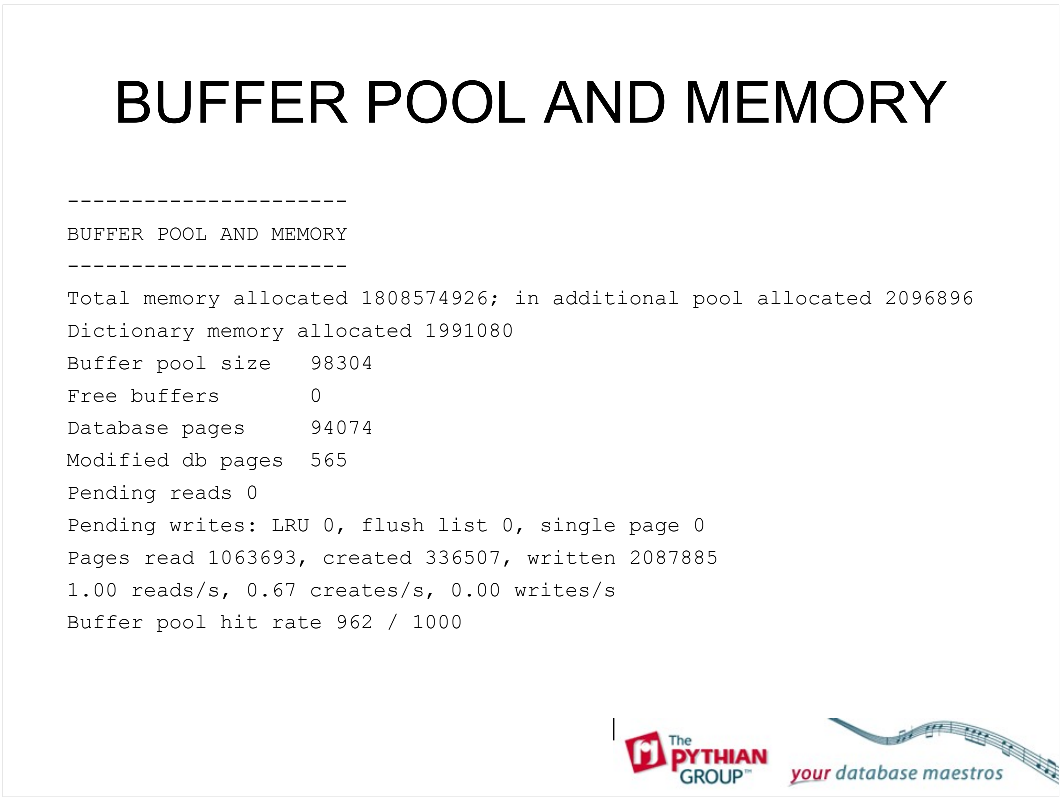
If a significant amount, like 30%, of innodb_log_buffer_size is unflushed, you may want to increase the amount of innodb_log_buffer_size.



LOG

```
---
LOG
---
Log sequence number 120 4209985949
Log flushed up to 120 4209985949
Last checkpoint at 120 4191865426
0 pending log writes, 0 pending chkp writes
325299 log i/o's done, 1.30 log i/o's/second
```

Here, the data is flushed, last checkpoint is over 17 Mb ago. This is on a server with innodb_flush_logs_at_trx_commit=0, so I always expect the logs to be flushed!



BUFFER POOL AND MEMORY

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 1808574926; in additional pool allocated 2096896
Dictionary memory allocated 1991080
Buffer pool size   98304
Free buffers       0
Database pages     94074
Modified db pages  565
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 1063693, created 336507, written 2087885
1.00 reads/s, 0.67 creates/s, 0.00 writes/s
Buffer pool hit rate 962 / 1000
```

Total memory allocated and in additional pool are in bytes

dictionary memory was added in 5.1

buffer pool size is in pages

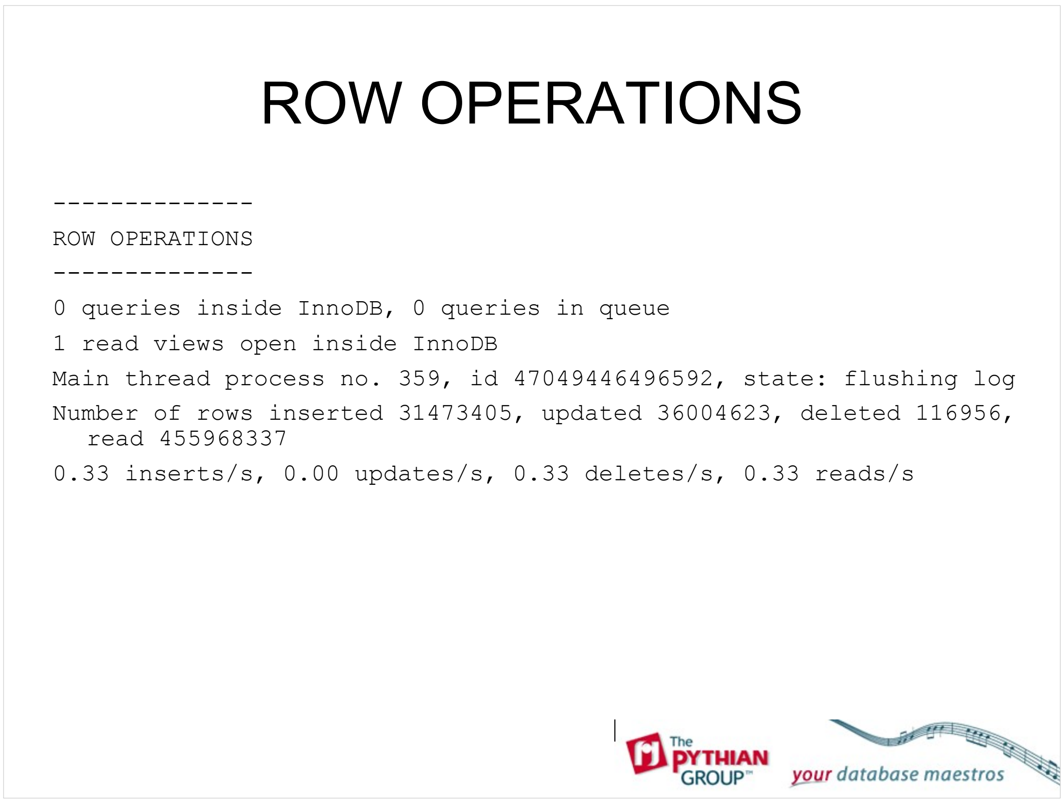
modified db pages are pages not flushed to disk yet (>0 even when flush_logs_at_trx_commit=0).

Pages read 1063693, created 336507, written 2087885 (pages that were read/written to disk)

LRU – least recently used

flush list hold the pages that have to be flushed by checkpoint process

single page writes – writes that will not be batched with other writes when written to disk



ROW OPERATIONS

```
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread process no. 359, id 47049446496592, state: flushing log
Number of rows inserted 31473405, updated 36004623, deleted 116956,
   read 455968337
0.33 inserts/s, 0.00 updates/s, 0.33 deletes/s, 0.33 reads/s
```

Read view is a consistent MVCC snapshot of the db as of point transaction started

Feedback

Questions?

Comments?

Suggestions?

Covered the information from SHOW ENGINE INNODB STATUS, and how to tune:

- innodb_thread_concurrency
- innodb_adaptive_hash_index
- innodb_sync_spin_loops
- innodb_thread_sleep_delay
- innodb_commit_concurrency
- innodb_concurrency_tickets

The PYTHIAN GROUP™ *your database maestros*