

Identifying and Killing Rogue Queries

ClustrixDB provides several mechanisms to identify queries that consume a disproportionate amount of system resources. Such queries are typically a result of poor indexing or bugs in the application.

ClustrixDB supports the following syntax for killing queries:

```
KILL [QUERY | CONNECTION]
session_id
```

Identifying Long Running Queries

The following statement will output the longest running query in the system. It's often the first step that a system administrator will take to identify possible problems on a misbehaving cluster. The sessions virtual relation provides a great deal of detail about each session's executing state. In addition to current statements, the connection information and transaction state will also be displayed.

```
sql> select * from system.sessions where statement_state = 'executing' order by time_in_state_s desc limit 1\G
***** 1. row *****
      nodeid: 2
      session_id: 99938306
      source_ip: 10.2.2.243
      source_port: 40758
      local_ip: 10.2.14.15
      local_port: 3306
      user: 4099
      database: system
      trx_state: open
      statement_state: executing
          xid: 5832691561615822852
          cpu: 4
      isolation: REPEATABLE-READ
      last_statement: select * from sessions where statement_state = 'executing' order by time_in_state_s desc
limit 1
      time_in_state_s: 0
      created: 2016-01-12 22:01:40
      heap_id: 288230379201751147
      trx_age_s: 0
      trx_mode: autocommit
      trx_counter_select: 1
      trx_counter_insert: 0
      trx_counter_update: 0
      trx_counter_delete: 0
      trx_is_writer: 0
1 row in set (0.00 sec)
```

Identifying Long Running Writer Transactions

In a fully relational SQL database such as ClustrixDB, long running write transactions may cause a problem. Frequently, misbehaving applications erroneously leave the AUTOCOMMIT option OFF, leaving every session to run in a single, very long transaction. When such cases occur, these transactions will accrue a large collection of write locks, preventing other transactions that attempt to modify the same data from running. To identify such cases, ClustrixDB includes several columns in the sessions relation that track the age of the transaction, the number and types of statements executed in the current transaction, and whether the transaction has issued any writes (boolean value 0, 1).

For example, to find the oldest write transaction in the system, issue the following:

```
sql> select * from system.sessions where trx_is_writer order by trx_age desc limit 1\G
***** 1. row *****
      nodeid: 2
      session_id: 99938306
      source_ip: 10.2.2.243
      source_port: 40758
      local_ip: 10.2.14.15
      local_port: 3306
      user: 4099
      database: sergei
      trx_state: open
      statement_state: executing
          xid: 5832694275126951940
          cpu: 4
      isolation: REPEATABLE-READ
      last_statement: select * from system.sessions where trx_is_writer order by trx_age desc limit 1
      time_in_state_s: 0
          created: 2016-01-12 22:01:40
          heap_id: 288230379201751394
      trx_age_s: 31
      trx_mode: explicit
      trx_counter_select: 2
      trx_counter_insert: 5
      trx_counter_update: 1
      trx_counter_delete: 3
          trx_is_writer: 1
1 row in set (0.00 sec)
```