CassandraCli08

Cassandra ships with a very basic interactive command line interface, or shell. Using the CLI you can connect to remote nodes in the cluster, create or update your schema, set and retrieve records and columns, or query node and cluster meta-data (i.e. cluster name, keyspace listings and disposition, etc). This page is intended for those using Cassandra 0.8.x. For CLI docs on 0.7.x see this page, 0.6.x, see this page.

You can start the CLI using the `bin/cassandra-cli` startup script in your Cassandra installation.

```
evans@achilles:~/cassandra$ bin/cassandra-cli
Welcome to cassandra CLI.

Type 'help;' or '?' for help. Type 'quit;' or 'exit;' to quit.
[default@unknown] connect localhost/9160;
Connected to: "Test Cluster" on localhost/9160
[default@unknown] create keyspace Twissandra;
d105c4f1-3c93-11e0-9fb5-e700f669bcfc
[default@unknown] use Twissandra;
Authenticated to keyspace: Twissandra
[default@Twissandra] create column family User with comparator = UTF8Type;
00389812-3c94-11e0-9fb5-e700f669bcfc
[default@Twissandra] quit;
evans@achilles:~/cassandra$
```

In the above example we started the cli with no options. You can specify things like `-host`, `-port`, `-keyspace`, `-username`, `-password`, etc. Use `bin/cassandra-cli -?` for a full set of options.

We went on to connect to our local Cassandra node. We created keyspace `Twissandra` and column family `User`. Note that with the column family, we used a UTF8Type comparator. That means that the columns will be sorted based on UTF8Type sorting. It also means that when the column names are displayed on the command-line, they will be displayed as UTF8Type (readable) text. For more information and options for creating column families type `help create column family;` on the command line. Finally, we exited from our cli shell.

Let's get back into the shell with some options specified and create some data. You should be aware that using the right assumption for your column family keys is 'essential' for the CLI to work correctly. None of the data retrieval/manipulation commands will work as expected if the key assumption is wrong. If you are just exploring cassandra from the CLI, you can leave the assumptions at their defaults, though.

```
tblose@quasar:~/dev/workspaces/cassandra$ bin/cassandra-cli -host localhost -port 9160
Connected to localhost/9160
Welcome to cassandra CLI.

Type 'help;' or '?' for help. Type 'quit;' or 'exit;' to quit.
[default@unknown] use Twissandra;
Authenticated to keyspace: Twissandra
[default@Twissandra] set User['jsmith']['first'] = 'John';
Value inserted.
[default@Twissandra] set User['jsmith']['last'] = 'Smith';
Value inserted.
[default@Twissandra] set User['jsmith']['age'] = '39';
Value inserted.
```

Note that before we can start adding data, we have to `use Twissandra;` to set our context. We created a record in the `User` column family using the key `jsmith`. This record has three columns, `first`, `last`, and `age`. Each of these commands is the equivalent to an `insert()` using the Thrift API.

Now let's read back the `jsmith` row to see what it contains:

```
[default@Twissandra] get User['jsmith'];
=> (column=age, value=3339, timestamp=1298504259386000)
=> (column=first, value=4a6f686e, timestamp=1298504239938000)
=> (column=last, value=536d697468, timestamp=1298504248570000)
Returned 3 results.
```

Note: Using the `get` command in this form is the equivalent to a `get_slice()` using the Thrift API.

Why are the values all hex? It's because the default validation class is BytesType, which displays in hex in the output. Let's update the column metadata of the column family to not only make them output in a readable format, but also add a secondary index on age. We'll also add another record so that we can see the secondary index work.

```
[default@Twissandra] update column family User
with
...        column_metadata =
...        [
...        {column_name: first, validation_class: UTF8Type},
...        {column_name: last, validation_class: UTF8Type},
...        {column_name: age, validation_class: UTF8Type, index_type: KEYS}
...        ];
fd98427f-3fa6-11e0-8f42-e700f669bcfc
[default@Twissandra] set User['zaphod']['first'] = 'Zaphod';
Value inserted.
[default@Twissandra] set User['zaphod']['last'] = 'Beeblebrox';
Value inserted.
[default@Twissandra] set User['zaphod']['age'] = '42';
Value inserted.
[default@Twissandra] get User where age = '42';
-------------------
RowKey: zaphod
=> (column=age, value=42, timestamp=1298504874382000)
=> (column=first, value=Zaphod, timestamp=1298504803709000)
=> (column=last, value=Beeblebrox, timestamp=1298504848982000)

1 Row Returned.
```

In the above example, you can see that we can span commands over multiple lines. We add column metadata that validates the column data as well as display value unencoded in the cli output. We also add an index on age. The KEYS index type means that we can only perform equality operations over it. We add one more row with an age of '42' and finally query the column family for rows with an age of 42.

One final thing that is very handy about the cassandra-cli, you can script your schema creation in a file and run it through the cli. You just create a text file with any number of creation commands and run the cli with the `-f` option:

```
tblose@quasar:~/dev/workspaces/cassandra$ bin/cassandra-cli -host localhost -port 9160 -f ~/cassandra-schema.txt
Connected to: "Test Cluster" on localhost/9160
1eafa8f4-3faf-11e0-a627-e700f669bcfc
Authenticated to keyspace: Twissandra
1f09fdf5-3faf-11e0-a627-e700f669bcfc
```

with `cassandra-schema.txt`:

```
create keyspace Twissandra;
use Twissandra;

create column family User with
  comparator = UTF8Type and
  column_metadata =
  [
    {column_name: first, validation_class: UTF8Type},
    {column_name: last, validation_class: UTF8Type},
    {column_name: age, validation_class: UTF8Type, index_type: KEYS}
  ];
```

This has just been a brief introduction with a couple of examples. For more information on how things work, type `help;` on the cli or see the help below.

Commands

assume

```
assume <cf> comparator as <type>;
assume <cf> sub_comparator as <type>;
assume <cf> validator as <type>;
assume <cf> keys as <type>;
```

Assume one of the attributes (comparator, sub_comparator, validator or keys) of the given column family match specified type. The specified type will be used when displaying data returned from the column family.

This statement does not change the column family definition stored in Cassandra. It only affects the cli and how it will transform values to be sent to and interprets results from Cassandra.

If results from Cassandra do not validate according to the assumptions an error is displayed in the cli.

Required Parameters

- `cf:` Name of the column family to make the assumption about.
- `type:` Validator type to use when processing values.
  Supported values are:
  - AsciiType
  - BytesType
  - CounterColumnType (distributed counter column)
  - IntegerType (a generic variable-length integer type)
  - LexicalUUIDType
  - LongType
  - UTF8Type
    It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

Examples

```
assume Standard1 comparator as lexicaluuid;
assume Standard1 keys as ascii;
```

connect

```
connect <hostname>/<port> (<username> '<password>')?;
```

Connect to the a Cassandra node on the specified port.

If a username and password are supplied the login will occur when the 'use' statement is executed. If the server does not support authentication it will silently ignore credentials.

For information on configuring authentication and authorisation see the conf/cassandra.yaml file or the project documentation.

Required Parameters

- `hostname:` Machine name or IP address of the node to connect to.
- `port:` rpc_port to connect to the node on, as defined inconf/Cassandra.yaml for the node. The default port is 9160.

Optional Parameters

- `password:` Password for the supplied username.
- `username:` Username to authenticate to the node as.

Examples

```
connect localhost/9160;
connect localhost/9160 user 'badpasswd';
connect 127.0.0.1/9160 user 'badpasswd';
```

consistencylevel

```
consistencylevel as <level>;
```

Sets the consistency level for the client to use. Defaults to One.

Required Parameters

- `level:` Consistency level the client should use. Value is case insensitive.
  Supported values are:
  - ONE
  - TWO
  - THREE
  - QUORUM
  - ALL
  - LOCAL_QUORUM
  - EACH_QUORUM
  - ANY
    Note: Consistency level ANY can only be used for write operations.

count

```
count <cf>['<key>'];
count <cf>['<key>']['<super>'];
```

Count the number of columns in the row with the specified key, or subcolumns in the specified super column.

Required Parameters

- `cf:` Name of the column family to read from..
- `key:` Key for the row to count.

Optional Parameters

- `super:` Name of the super column to count subcolumns in.

Examples

```
count Super1['testkey']['my super'];
count Standard1['testkey'];
```

create column family

```
create column family <name>;
create column family <name> with <att1>=<value1>;
create column family <name> with <att1>=<value1> and <att2>=<value2>...;
```

Create a column family in the current keyspace with the specified attributes.

Required Parameters

- `name:` Name of the new column family. Names may only contain letters, numbers and underscores.

column family Attributes (all are optional):

- `column_metadata:` Defines the validation and indexes for known columns in this column family.
  Columns not listed in the column_metadata section will use the default_validator to validate their values.
  Column Required parameters:
  - `name:` Binds a validator (and optionally an indexer) to columns with this name in any row of the enclosing column family.
  - `validator:` Validator to use for values for this column.
    Supported values are:
    - AsciiType
    - BytesType
    - CounterColumnType (distributed counter column)
    - IntegerType (a generic variable-length integer type)

        - LexicalUUIDType
        - LongType
        - UTF8Type
          It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.
          Column Optional parameters:
    - `index_name:` Name for the index. Both an index name and type must be specified.

    - `index_type:` The type of index to be created.
      Suported values are:
        - `0:` for a KEYS based index

- `column_type:` Type of columns this column family holds, valid values are Standard and Super. Default is Standard.

- `comment:` Human readable column family description.

- `comparator:` Validator to use to validate and compare column names in this column family. For Standard column families it applies to columns, for Super column families applied to super columns. Also see the subcomparator attribute. Default is BytesType, which is a straight forward lexical comparison of the bytes in each column.
  Supported values are:
    - AsciiType
    - BytesType
    - CounterColumnType (distributed counter column)
    - IntegerType (a generic variable-length integer type)
    - LexicalUUIDType
    - LongType
    - UTF8Type
      It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `default_validation_class:` Validator to use for values in columns which are not listed in the column_metadata. Default is BytesType which applies no validation.
  Supported values are:
    - AsciiType
    - BytesType
    - CounterColumnType (distributed counter column)
    - IntegerType (a generic variable-length integer type)
    - LexicalUUIDType
    - LongType
    - UTF8Type
      It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `key_validation_class:` Validator to use for keys. Default is BytesType which applies no validation.
  Supported values are:
    - AsciiType
    - BytesType
    - IntegerType (a generic variable-length integer type)
    - LexicalUUIDType
    - LongType
    - UTF8Type
      It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `gc_grace:` Time to wait in seconds before garbage collecting tombstone deletion markers. Default value is 864000 or 10 days.
  Set this to a large enough value that you are confident that the deletion markers will be propagated to all replicas by the time this many seconds has elapsed, even in the face of hardware failures.
  See http://wiki.apache.org/Cassandra/DistributedDeletes

- `keys_cached:` Maximum number of keys to cache in memory. Valid values are either a double between 0 and 1 (inclusive on both ends) denoting what fraction should be cached. Or an absolute number of rows to cache. Default value is 200000.
  Each key cache hit saves 1 seek and each row cache hit saves 2 seeks at the minimum, sometimes more. The key cache is fairly tiny for the amount of time it saves, so it's worthwhile to use it at large numbers all the way up to 1.0 (all keys cached). The row cache saves even more time, but must store the whole values of its rows, so it is extremely space-intensive. It's best to only use the row cache if you have hot rows or static rows.

- `keys_cache_save_period:` Duration in seconds after which Cassandra should safe the keys cache. Caches are saved to saved_caches_directory as specified in conf/Cassandra.yaml. Default is 14400 or 4 hours.
  Saved caches greatly improve cold-start speeds, and is relatively cheap in terms of I/O for the key cache. Row cache saving is much more expensive and has limited use.

- `memtable_flush_after:` Maximum number of minutes to leave a dirty memtable unflushed. This value needs to be large enough that it won't cause a flush storm of all your memtables flushing at once because none have hit the size or count thresholds yet. For production a larger value such as 1440 is recommended. Default is 60.
  NOTE: While any affected column families have unflushed data from a commit log segment, that segment cannot be deleted.

- `memtable_operations:` Number of operations in millions before the memtable is flushed. Default is memtable_throughput / 64 * 0.3

- `memtable_throughput:` Maximum size in MB to let a memtable get to before it is flushed. Default is to use 1/16 the JVM heap size.

- `read_repair_chance:` Probability (0.0-1.0) with which to perform read repairs for any read operation. Default is 1.0 to enable read repair.
  Note that disabling read repair entirely means that the dynamic snitch will not have any latency information from all the replicas to recognize when

one is performing worse than usual.

- `rows_cached:` Maximum number of rows whose entire contents we cache in memory. Valid values are either a double between 0 and 1 (inclusive on both ends) denoting what fraction should be cached. Or an absolute number of rows to cache. Default value is 0, to disable row caching.
  Each key cache hit saves 1 seek and each row cache hit saves 2 seeks at the minimum, sometimes more. The key cache is fairly tiny for the amount of time it saves, so it's worthwhile to use it at large numbers all the way up to 1.0 (all keys cached). The row cache saves even more time, but must store the whole values of its rows, so it is extremely space-intensive. It's best to only use the row cache if you have hot rows or static rows.

- `row_cache_save_period:` Duration in seconds after which Cassandra should safe the row cache. Caches are saved to saved_caches_directory as specified in conf/Cassandra.yaml. Default is 0 to disable saving the row cache.
  Saved caches greatly improve cold-start speeds, and is relatively cheap in terms of I/O for the key cache. Row cache saving is much more expensive and has limited use.

- `subcomparator:` Validator to use to validate and compare sub column names in this column family. Only applied to Super column families. Default is BytesType, which is a straight forward lexical comparison of the bytes in each column.
  Supported values are:
  - AsciiType
  - BytesType
  - CounterColumnType (distributed counter column)
  - IntegerType (a generic variable-length integer type)
  - LexicalUUIDType
  - LongType
  - UTF8Type
    It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `max_compaction_threshold:` The maximum number of SSTables allowed before aminor compaction is forced. Default is 32, setting to 0 disables minorcompactions.

Decreasing this will cause minor compactions to start more frequently andbe less intensive. The min_compaction_threshold and max_compaction_thresholdboundaries are the number of tables Cassandra attempts to merge together atonce.

- `min_compaction_threshold:` The minimum number of SSTables neededto start a minor compaction. Default is 4, setting to 0 disables minorcompactions.

Increasing this will cause minor compactions to start less frequently andbe more intensive. The min_compaction_threshold and max_compaction_thresholdboundaries are the number of tables Cassandra attempts to merge together atonce.

- `replicate_on_write:` Replicate every counter update from the leader to thefollower replicas. Accepts the values true and false.

- `row_cache_provider:` The provider for the row cache to use for thiscolumn family. Defaults to ConcurrentLinkedHashCacheProvider. .

Supported values are:

- ConcurrentLinkedHashCacheProvider
- SerializingCacheProvider

It is also valid to specify the fully-qualified class name to a classthat implements org.apache.cassandra.cache.IRowCacheProvider.

ConcurrentLinkedHashCacheProvider provides the same features as the versionsprior to Cassandra v0.8. Row data is cached using the Java JVM heap.

SerializingCacheProvider serialises the contents of the row andstores the data off the JVM Heap. This may reduce the GC pressure.NOTE: Thsi provider requires JNA.jar to be in the class path toenable native methods.

Examples

```
create column family Super4
    with column_type = 'Super'
    and comparator = 'AsciiType'
    and rows_cached = 10000;
create column family Standard3
    with comparator = 'LongType'
    and rows_cached = 10000;
create column family Standard4
    with comparator = AsciiType
    and column_metadata =
    [{
        column_name : Test,
        validation_class : IntegerType,
        index_type : 0,
        index_name : IdxName},
    {
        column_name : 'other name',
        validation_class : LongType
    }];
```

create keyspace

```
create keyspace <keyspace>;
create keyspace <keyspace> with <att1>=<value1>;
create keyspace <keyspace> with <att1>=<value1> and <att2>=<value2> ...;
```

Create a keyspace with the specified attributes.

Required Parameters

- `keyspace:` Name of the new keyspace, "system" is reserved for Cassandra internals. Names may only contain letters, numbers and underscores.

Keyspace Attributes (all are optional):

- `placement_strategy:` Class used to determine how replicas are distributed among nodes. Defaults to [NetworkTopologyStrategy] with one datacenter defined with a replication factor of 1 ("[datacenter1:1]").
  Supported values are:
    - org.apache.Cassandra.locator.SimpleStrategy
    - org.apache.Cassandra.locator.NetworkTopologyStrategy
    - org.apache.Cassandra.locator.OldNetworkTopologyStrategy
      [SimpleStrategy] merely places the first replica at the node whose token is closest to the key (as determined by the Partitioner), and additional replicas on subsequent nodes along the ring in increasing Token order.
      Supports a single strategy option 'replication_factor' that specifies the replication factor for the cluster.
      With [NetworkTopologyStrategy], for each datacenter, you can specify how many replicas you want on a per-keyspace basis. Replicas are placed on different racks within each DC, if possible.
      Supports strategy options which specify the replication factor for each datacenter. The replication factor for the entire cluster is the sum of all per datacenter values. Note that the datacenter names must match those used in conf/cassandra-topology.properties.
      [OldNetworkToplogyStrategy] [formerly RackAwareStrategy] places one replica in each of two datacenters, and the third on a different rack in in the first. Additional datacenters are not guaranteed to get a replica. Additional replicas after three are placed in ring order after the third without regard to rack or datacenter.
      Supports a single strategy option 'replication_factor' that specifies the replication factor for the cluster.

- `strategy_options:` Optional additional options for placement_strategy. Options have the form [{key:value}], see the information on each strategy and the examples.

Examples

```
create keyspace Keyspace2
    with placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy'
    and strategy_options = [{replication_factor:4}];
create keyspace Keyspace3
    with placement_strategy = 'org.apache.cassandra.locator.NetworkTopologyStrategy'
    and strategy_options=[{DC1:2, DC2:2}];
create keyspace Keyspace4
    with placement_strategy = 'org.apache.cassandra.locator.OldNetworkTopologyStrategy'
    and strategy_options = [{replication_factor:1}];
```

decr

```
decr <cf>['<key>']['<col>'] [by <value>];
decr <cf>['<key>']['<super>']['<col>'] [by <value>];
```

Decrement the specified column by the supplied value.

Note: Counter columns must be defined using a 'create column family' or 'update column family' statement in the column_metadata as using the ColumnCounterType validator.

Required Parameters

- `cf:` Name of the column family to decrement the column in.
- `col:` Name of the counter column to increment.
- `key:` Key for the row to decrement the counter in.

Optional Parameters

- `super:` Name of the super column that contains the counter column.
- `value:` Signed integer value to decrement the column by. If not supplied1 is used.

Examples

```
decr Counter1[ascii('testkey')][ascii('test col')];
decr SuperCounter1[ascii('testkey')][ascii('my super')][ascii('test col')] by 42;
decr Counter1[ascii('testkey')][ascii('test col')] by 10;
```

del

```
del <cf>['<key>'];
del <cf>['<key>']['<col>'];
del <cf>['<key>']['<super>'];
del <cf>['<key>']['<super>']['<col>'];
del <cf>[<function>(<key>)][<function>(<super>)][<function>(<col>)];
```

Deletes a row, a column, or a subcolumn.

Required Parameters

- `cf:` Name of the column family to delete from.
- `key:` Key for the row delete from.

Optional Parameters

- `col:` Name of the column to delete.
- `function:` Name of a function to call to parse the supplied argument to the specified type. Some functions will generate values without needing an argument.
  Supported values are:
  - ascii
  - `bytes:` if used without arguments generates a zero length byte array
  - integer
  - `lexicaluuid:` if used without arguments generates a new random uuid
  - long
  - `timeuuid:` if used without arguments generates a new time uuid
  - utf8
- `super:` Name of the super column to delete from. If col is not specifiedthe super column and all sub columns will be deleted.

Examples

```
del Super1[ascii('testkey')][ascii('my_super')][ascii('test_col')];
del Standard1['testkey'][ascii('test col')];
del Standard1['testkey'];
del Standard1[utf8('testkey')];
```

describe cluster

```
describe cluster;
```

Describes the snitch, partitioner and schema versions for the currently connected cluster.

NOTE: The cluster should only report one schema version. Multiple versions may indicate a failed schema modification, consult the project documentation.

Examples

```
describe cluster;
```

describe keyspace

```
describe keyspace (<keyspace>)?;
```

Describes the settings for the current or named keyspace, and the settings for all column families in the keyspace.

Optional Parameters

- `keyspace:` Name of the keyspace to describe.

Examples

```
describe keyspace;
describe keyspace system;
```

drop column family

```
drop column family <cf>;
```

Drops the specified column family.

A snapshot of the data is created in a sub directory of the Keyspace data directory. The files must be manually deleted using either "nodetool clearsnapshot" or the command line.

Required Parameters

- `cf:` Name of the column family to delete.

Example:drop column family Standard2;

drop keyspace

```
drop keyspace <keyspace>;
```

Drops the specified keyspace.

A snapshot of the data is created in a sub directory of the Keyspace data directory. The files must be manually deleted using either "nodetool clearsnapshot" or the command line.

Required Parameters

- `keyspace:` Name of the keyspace to delete.

Example:drop keyspace Keyspace1;

exit

```
exit;
quit;
```

Exit this utility.

Examples

```
exit;
quit;
```

get

```
get <cf>['<key>'];
get <cf>['<key>']['<col>'] (as <type>)*;
get <cf>['<key>']['<super>']['<col>'] (as <type>)*;
get <cf>['<key>']['<super>'];
get <cf>['<key>'][<function>];
get <cf>[function(<key>)][<function>(<super>)][<function>(<col>)];
get <cf> where <col> <operator> <value> [
    and <col> <operator> <value> and ...] [limit <limit>];
get <cf> where <col> <operator> <function>(<value>) [
    and <col> <operator> <function> and ...] [limit <limit>];
```

Gets columns or super columns for the specified column family and key. Or returns all columns from rows which meet the specified criteria when using the 'where' form.

Note: The implementation of secondary indexes in Cassandra 0.7 has some restrictions, see http://www.datastax.com/dev/blog/whats-new-Cassandra-07-secondary-indexes

Required Parameters

- `cf:` Name of the column family to read from.

Optional Parameters

- `col:` Name of the column to read. Or in the 'where' form name of the columnto test the value of.

- `function:` Name of a function to call to parse the supplied argument to the specified type. Some functions will generate values without needing an argument.
  Valid options are:
  - ascii
  - `bytes:` if used without arguments generates a zero length byte array
  - integer
  - `lexicaluuid:` if used without arguments generates a new random uuid
  - long
  - `timeuuid:` if used without arguments generates a new time uuid
  - utf8
- `key:` Key for the row to read columns from. This parameter is required in all cases except when the 'where' clause is used.
- `limit:` Number of rows to return. Default is 100.
- `operator:` Operator to test the column value with. Supported operators are =, >, >=, <, <= .
  In Cassandra 0.7 at least one = operator must be present.
- `super:` Name of the super column to read from. If super is supplied without col then all columns from the super column are returned.
- `type:` Data type to interpret the the columns value as for display.
  Valid options are:
  - AsciiType
  - BytesType
  - CounterColumnType (distributed counter column)
  - IntegerType (a generic variable-length integer type)
  - LexicalUUIDType
  - LongType
  - UTF8Type
- `value:` The value to test the column for, if a function is provided thevalue is parsed by the function. Otherwise the meta data for the targetcolumn is used to determine the correct type.

Examples

```
get Standard1[ascii('testkey')];
#tell cli to convert keys from ascii to bytes
assume Standard1 keys as ascii;
get Standard1[testkey][test_column] as IntegerType;
get Standard1[testkey][utf8(hello)];
get Indexed1 where birthdate=19750403;
```

help

```
help <command>;
```

Display the general help page with a list of available commands.;

incr

```
incr <cf>['<key>']['<col>'] [by <value>];
incr <cf>['<key>']['<super>']['<col>'] [by <value>];
```

Increment the specified counter column by the supplied value.

Note: Counter columns must be defined using a 'create column family' or 'update column family' statement in the column_metadata as using the ColumnCounterType validator.

Required Parameters

- `cf:` Name of the column family to increment the column in.
- `col:` Name of the counter column to increment.
- `key:` Key for the row to increment the counter in.

Optional Parameters

- `super:` Name of the super column that contains the counter column.
- `value:` Signed integer value to increment the column by. If not supplied1 is used.

Examples

```
incr Counter1[ascii('testkey')][ascii('test col')];
incr SuperCounter1[ascii('testkey')][ascii('my super')][ascii('test col')] by 42;
incr Counter1[ascii('testkey')][ascii('test col')] by -4;
```

list

```
list <cf>;
list <cf>[<startKey>:];
list <cf>[<startKey>:<endKey>];
list <cf>[<startKey>:<endKey>] limit <limit>;
```

List a range of rows, and all of their columns, in the specified column family.

The order of rows returned is dependant on the Partitioner in use.

Required Parameters

- `cf:` Name of the column family to list rows from.

Optional Parameters

- `endKey:` Key to end the range at. The end key will be includedin the result. Defaults to an empty byte array.
- `limit:` Number of rows to return. Default is 100.
- `startKey:` Key start the range from. The start key will beincluded in the result. Defaults to an empty byte array.

Examples

```
list Standard1;
list Super1[j:];
list Standard1[j:k] limit 40;
```

set

```
set <cf>['<key>']['<col>'] = <value>;
set <cf>['<key>']['<super>']['<col>'] = <value>;
set <cf>['<key>']['<col>'] = <function>(<argument>);
set <cf>['<key>']['<super>']['<col>'] = <function>(<argument>);
set <cf>[<key>][<function>(<col>)] = <value> || <function>;
set <cf>[<function>(<key>)][<function>(<col>) || <col>] =
    <value> || <function> with ttl = <secs>;
```

Sets the column value for the specified column family and key.

Required Parameters

- `cf:` Name of the column family to set columns in.
- `col:` Name of the column to set.

- `key:` Key for the row to set columns in.

Optional Parameters

- `function:` Name of a function to call to parse the supplied argument to the specified type. Some functions will generate values without needing an argument.
  Valid options are:
  - ascii
  - `bytes:` if used without arguments generates a zero length byte array
  - integer
  - `lexicaluuid:` if used without arguments generates a new random uuid
  - long
  - `timeuuid:` if used without arguments generates a new time uuid
  - utf8
- `secs:` Time To Live for the column in seconds. Defaults to no ttl.
- `super:` Name of the super column to contain the column.
- `value:` The value to set the column to.

Examples

```
set Super1[ascii('testkey')][ascii('my super')][ascii('test col')]='this is a test';
set Standard1['testkey']['test col']='this is also a test';
set Standard1[testkey][testcol] = utf8('this is utf8 string.');
set Standard1[testkey][timeuuid()] = utf8('hello world');
set Standard1[testkey][timeuuid()] = utf8('hello world') with ttl = 30;
```

show api version

```
show api version;
```

Displays the API version number.

This version number is used by high level clients and is not the same as the server release version.

Examples

```
show api version;
```

show cluster name

```
show cluster name;
```

Displays the name of the currently connected cluster.

Examples

```
show cluster name;
```

show keyspaces

```
show keyspaces;
```

Describes the settings and the column families for all keyspaces on the currently connected cluster.

Examples

```
show keyspaces;
```

truncate

```
truncate <cf>;
```

Truncate specified column family.

Note: All nodes in the cluster must be up to truncate command to execute.

A snapshot of the data is created, which is deleted asyncronously during a 'graveyard' compaction.

Required Parameters

- `cf:` Name of the column family to truncate.

Examples

```
truncate Standard1;
```

update column family

```
update column family <name>;
update column family <name> with <att1>=<value1>;
update column family <name> with <att1>=<value1> and <att2>=<value2>...;
```

Updates the settings for a column family in the current keyspace.

Required Parameters

- `name:` Name of the column family to update.

column family Attributes (all are optional):

- `column_metadata:` Defines the validation and indexes for known columns in this column family.
  Columns not listed in the column_metadata section will use the default_validator to validate their values.
  Column Required parameters:
  - `name:` Binds a validator (and optionally an indexer) to columns with this name in any row of the enclosing column family.
  - `validator:` Validator to use for values for this column.
    Supported values are:
    - AsciiType
    - BytesType
    - CounterColumnType (distributed counter column)
    - IntegerType (a generic variable-length integer type)
    - LexicalUUIDType
    - LongType
    - UTF8Type
      It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.
      Column Optional parameters:
  - `index_name:` Name for the index. Both an index name and type must be specified.
    NOTE: After the update has completed the column family will only contain the secondary indexes listed in the update statement. Existing indexes will be dropped if they are not present in the update.
  - `index_type:` The type of index to be created.
    Suported values are:
    - `0:` for a KEYS based index
- `column_type:` Type of columns this column family holds, valid values are Standard and Super. Default is Standard.
- `comment:` Column family description.

- `comparator:` Validator to use to validate and compare column names in this column family. For Standard column families it applies to columns, for Super column families applied to super columns. Also see the subcomparator attribute. Default is BytesType, which is a straight forward lexical comparison of the bytes in each column.
  Supported values are:
  - AsciiType
  - BytesType
  - CounterColumnType (distributed counter column)
  - IntegerType (a generic variable-length integer type)
  - LexicalUUIDType
  - LongType
  - UTF8Type
    It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `default_validation_class:` Validator to use for values in columns which are not listed in the column_metadata. Default is BytesType which applies no validation.
  Supported values are:
  - AsciiType
  - BytesType
  - CounterColumnType (distributed counter column)
  - IntegerType (a generic variable-length integer type)
  - LexicalUUIDType
  - LongType
  - UTF8Type
    It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `key_valiation_class:` Validator to use for keys. Default is BytesType which applies no validation.
  Supported values are:
  - AsciiType
  - BytesType
  - IntegerType (a generic variable-length integer type)
  - LexicalUUIDType
  - LongType
  - UTF8Type
    It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `gc_grace:` Time to wait in seconds before garbage collecting tombstone deletion markers. Default value is 864000 or 10 days.
  Set this to a large enough value that you are confident that the deletion markers will be propagated to all replicas by the time this many seconds has elapsed, even in the face of hardware failures.
  See http://wiki.apache.org/Cassandra/DistributedDeletes

- `keys_cached:` Maximum number of keys to cache in memory. Valid values are either a double between 0 and 1 (inclusive on both ends) denoting what fraction should be cached. Or an absolute number of rows to cache. Default value is 200000.
  Each key cache hit saves 1 seek and each row cache hit saves 2 seeks at the minimum, sometimes more. The key cache is fairly tiny for the amount of time it saves, so it's worthwhile to use it at large numbers all the way up to 1.0 (all keys cached). The row cache saves even more time, but must store the whole values of its rows, so it is extremely space-intensive. It's best to only use the row cache if you have hot rows or static rows.

- `keys_cache_save_period:` Duration in seconds after which Cassandra should safe the keys cache. Caches are saved to saved_caches_directory as specified in conf/Cassandra.yaml. Default is 14400 or 4 hours.
  Saved caches greatly improve cold-start speeds, and is relatively cheap in terms of I/O for the key cache. Row cache saving is much more expensive and has limited use.

- `memtable_flush_after:` Maximum number of minutes to leave a dirty memtable unflushed. This value needs to be large enough that it won't cause a flush storm of all your memtables flushing at once because none have hit the size or count thresholds yet. For production a larger value such as 1440 is recommended. Default is 60.
  NOTE: While any affected column families have unflushed data from a commit log segment, that segment cannot be deleted.

- `memtable_operations:` Number of operations in millions before the memtable is flushed. Default is memtable_throughput / 64 * 0.3

- `memtable_throughput:` Maximum size in MB to let a memtable get to before it is flushed. Default is to use 1/16 the JVM heap size.

- `read_repair_chance:` Probability (0.0-1.0) with which to perform read repairs for any read operation. Default is 1.0 to enable read repair.

- `rows_cached:` Maximum number of rows whose entire contents we cache in memory. Valid values are either a double between 0 and 1 (inclusive on both ends) denoting what fraction should be cached. Or an absolute number of rows to cache. Default value is 0, to disable row caching.
  Each key cache hit saves 1 seek and each row cache hit saves 2 seeks at the minimum, sometimes more. The key cache is fairly tiny for the amount of time it saves, so it's worthwhile to use it at large numbers all the way up to 1.0 (all keys cached). The row cache saves even more time, but must store the whole values of its rows, so it is extremely space-intensive. It's best to only use the row cache if you have hot rows or static rows.

- `row_cache_save_period:` Duration in seconds after which Cassandra should safe the row cache. Caches are saved to saved_caches_directory as specified in conf/Cassandra.yaml. Default is 0 to disable saving the row cache.
  Saved caches greatly improve cold-start speeds, and is relatively cheap in terms of I/O for the key cache. Row cache saving is much more expensive and has limited use.

- `subcomparator:` Validator to use to validate and compare sub column names in this column family. Only applied to Super column families. Default is BytesType, which is a straight forward lexical comparison of the bytes in each column.
  Supported values are:

    - AsciiType
    - BytesType
    - CounterColumnType (distributed counter column)
    - IntegerType (a generic variable-length integer type)
    - LexicalUUIDType
    - LongType
    - UTF8Type
      It is also valid to specify the fully-qualified class name to a class that extends org.apache.Cassandra.db.marshal.AbstractType.

- `max_compaction_threshold:` The maximum number of SSTables allowed before aminor compaction is forced. Default is 32, setting to 0 disables minorcompactions.

Decreasing this will cause minor compactions to start more frequently andbe less intensive. The min_compaction_threshold and max_compaction_thresholdboundaries are the number of tables Cassandra attempts to merge together atonce.

- `min_compaction_threshold:` The minimum number of SSTables neededto start a minor compaction. Default is 4, setting to 0 disables minorcompactions.

Increasing this will cause minor compactions to start less frequently andbe more intensive. The min_compaction_threshold and max_compaction_thresholdboundaries are the number of tables Cassandra attempts to merge together atonce.

- `replicate_on_write:` Replicate every counter update from the leader to thefollower replicas. Accepts the values true and false.

- `row_cache_provider:` The provider for the row cache to use for thiscolumn family. Defaults to ConcurrentLinkedHashCacheProvider. .

Supported values are:

- ConcurrentLinkedHashCacheProvider
- SerializingCacheProvider

It is also valid to specify the fully-qualified class name to a classthat implements org.apache.cassandra.cache.IRowCacheProvider.

ConcurrentLinkedHashCacheProvider provides the same features as the versionsprior to Cassandra v0.8. Row data is cached using the Java JVM heap.

SerializingCacheProvider serialises the contents of the row andstores the data off the JVM Heap. This may reduce the GC pressure.NOTE: Thsi provider requires JNA.jar to be in the class path toenable native methods.

Examples

```
update column family Super4
    with column_type = 'Super'
    and comparator = 'AsciiType'
    and rows_cached = 10000;
update column family Standard3
    with comparator = 'LongType'
    and rows_cached = 10000;
update column family Standard4
    with comparator = AsciiType
    and column_metadata =
    [{
        column_name : Test,
        validation_class : IntegerType,
        index_type : 0,
        index_name : IdxName},
    {
        column_name : 'other name',
        validation_class : LongType
    }];
```

update keyspace

```
update keyspace <keyspace>;
update keyspace <keyspace> with <att1>=<value1>;
update keyspace <keyspace> with <att1>=<value1> and <att2>=<value2> ...;
```

Update a keyspace with the specified attributes.

Note: updating some keyspace properties may require additional maintenance actions. Consult project documentation for more details.

Required Parameters

- `keyspace:` Name of the keyspace to update.

Keyspace Attributes (all are optional):

- `placement_strategy:` Class used to determine how replicas are distributed among nodes. Defaults to [NetworkTopologyStrategy] with one datacenter defined with a replication factor of 1 ("[datacenter1:1]").
  Supported values are:
    - org.apache.Cassandra.locator.SimpleStrategy
    - org.apache.Cassandra.locator.NetworkTopologyStrategy
    - org.apache.Cassandra.locator.OldNetworkTopologyStrategy
      [SimpleStrategy] merely places the first replica at the node whose token is closest to the key (as determined by the Partitioner), and additional replicas on subsequent nodes along the ring in increasing Token order.
      Supports a single strategy option 'replication_factor' that specifies the replication factor for the cluster.
      With [NetworkTopologyStrategy], for each datacenter, you can specify how many replicas you want on a per-keyspace basis. Replicas are placed on different racks within each DC, if possible.
      Supports strategy options which specify the replication factor for each datacenter. The replication factor for the entire cluster is the sum of all per datacenter values. Note that the datacenter names must match those used in conf/cassandra-topology.properties.
      [OldNetworkToplogyStrategy] [formerly RackAwareStrategy] places one replica in each of two datacenters, and the third on a different rack in in the first. Additional datacenters are not guaranteed to get a replica. Additional replicas after three are placed in ring order after the third without regard to rack or datacenter.
      Supports a single strategy option 'replication_factor' that specifies the replication factor for the cluster.

- `strategy_options:` Optional additional options for placement_strategy. Options have the form [{key:value}], see the information on each strategy and the examples.

Examples

```
update keyspace Keyspace2
    with placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy'
    and strategy_options = [{replication_factor:4}];
update keyspace Keyspace3
    with placement_strategy = 'org.apache.cassandra.locator.NetworkTopologyStrategy'
    and strategy_options=[{DC1:2, DC2:2}];
update keyspace Keyspace4
    with placement_strategy = 'org.apache.cassandra.locator.OldNetworkTopologyStrategy'
    and strategy_options = [{replication_factor:1}];
```

use

```
use <keyspace>;
use <keyspace> <username> '<password>';
```

Use the specified keyspace.

If a username and password are supplied they will be used to authorize against the keyspace. Otherwise the credentials supplied to the 'connect' statement will be used to authorize the user . If the server does not support authentication it will silently ignore credentials.

Required Parameters

- `keyspace:` Name of the keyspace to use. The keyspace must exist.

Optional Parameters

- `password:` Password for the supplied username.
- `username:` Username to login to the node as.

Examples

```
use Keyspace1;
use Keyspace1 user 'badpasswd';
```

https://c.statcounter.com/9397521/0/fe557aad/1/|stats