Going Native With Apache Cassandra™

NoSQL Matters, Cologne, 2014
www.datastax.com
@DataStaxEU
About Me

Johnny Miller
Solutions Architect

@CyanMiller
www.linkedin.com/in/johnnymiller

We are hiring
www.datastax.com/careers
@DataStaxCareers
DataStax

- Founded in April 2010
- We drive Apache Cassandra™
- 400+ customers (25 of the Fortune 100)
- 200+ employees
- Home to Apache Cassandra™ Chair & most committers
  - Contribute ~ 90% of code into Apache Cassandra™ code base
- Headquartered in San Francisco Bay area
- European headquarters established in London
- Offices in France and Germany
<table>
<thead>
<tr>
<th>Rank</th>
<th>Last Month</th>
<th>DBMS</th>
<th>Database Model</th>
<th>Score</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Oracle</td>
<td>Relational DBMS</td>
<td>1514.08</td>
<td>+22.28</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>MySQL</td>
<td>Relational DBMS</td>
<td>1292.67</td>
<td>+2.45</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Microsoft SQL Server</td>
<td>Relational DBMS</td>
<td>1210.43</td>
<td>+5.15</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>PostgreSQL</td>
<td>Relational DBMS</td>
<td>230.23</td>
<td>-4.82</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>MongoDB</td>
<td>Document store</td>
<td>214.34</td>
<td>+14.35</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>DB2</td>
<td>Relational DBMS</td>
<td>184.58</td>
<td>-2.74</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Microsoft Access</td>
<td>Relational DBMS</td>
<td>142.76</td>
<td>-3.72</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>SQLite</td>
<td>Relational DBMS</td>
<td>90.17</td>
<td>-2.80</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>Cassandra</td>
<td>Wide column store</td>
<td>78.72</td>
<td>+0.63</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>Sybase ASE</td>
<td>Relational DBMS</td>
<td>78.14</td>
<td>-3.42</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>Solr</td>
<td>Search engine</td>
<td>62.88</td>
<td>+1.74</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>Teradata</td>
<td>Relational DBMS</td>
<td>61.73</td>
<td>-0.90</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>Redis</td>
<td>Key-value store</td>
<td>58.46</td>
<td>+5.01</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>FileMaker</td>
<td>Relational DBMS</td>
<td>54.38</td>
<td>+1.47</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>Informix</td>
<td>Relational DBMS</td>
<td>36.70</td>
<td>-0.49</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>HBase</td>
<td>Wide column store</td>
<td>36.60</td>
<td>+1.46</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>Hive</td>
<td>Relational DBMS</td>
<td>31.02</td>
<td>+0.81</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>Memcached</td>
<td>Key-value store</td>
<td>30.99</td>
<td>-1.92</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>Elasticsearch</td>
<td>Search engine</td>
<td>29.80</td>
<td>+3.63</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>CouchDB</td>
<td>Document store</td>
<td>22.55</td>
<td>-0.31</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>Splunk</td>
<td>Search engine</td>
<td>22.51</td>
<td>+0.08</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>Neo4j</td>
<td>Graph DBMS</td>
<td>20.56</td>
<td>+1.79</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>SAP HANA</td>
<td>Relational DBMS</td>
<td>18.02</td>
<td>+2.10</td>
</tr>
<tr>
<td>24</td>
<td>23</td>
<td>Firebird</td>
<td>Relational DBMS</td>
<td>15.68</td>
<td>-0.98</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>Netezza</td>
<td>Relational DBMS</td>
<td>15.56</td>
<td>-0.29</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>Windows Azure SQL Database</td>
<td>Relational DBMS</td>
<td>13.69</td>
<td>+0.66</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>Couchbase</td>
<td>Document store</td>
<td>12.92</td>
<td>+1.10</td>
</tr>
</tbody>
</table>

What do I mean by going native?

• Traditionally, Cassandra clients (Hector, Astynax\(^1\) etc..) were developed using Thrift

• With Cassandra 1.2 (Jan 2013) and the introduction of CQL3 and the CQL native protocol a new easier way of using Cassandra was introduced.

• Why?
  • Easier to develop and model
  • Best practices for building modern distributed applications
  • Integrated tools and experience
  • Enable Cassandra to evolve easier and support new features

\(^1\)Astynax is being updated to include the native driver: https://github.com/Netflix/astyanax/wiki/Astyanax-over-Java-Driver
There is a proposal to freeze thrift starting with 2.1.0


- Will retain it for backwards compatibility, but no new features or changes to the Thrift API after 2.1.0

“CQL3 is almost two years old now and has proved to be the better API that Cassandra needed. CQL drivers have caught up with and passed the Thrift ones in terms of features, performance, and usability. CQL is easier to learn and more productive than Thrift.”

- Jonathan Ellis, Apache Chair, Cassandra
CQL

- **Cassandra Query Language**

- CQL is intended to provide a common, simpler and easier to use interface into Cassandra - and you probably already know it!

  e.g. SELECT * FROM users

- **Usual statements**
  - CREATE / DROP / ALTER TABLE / SELECT

Find out more:
- http://www.datastax.com/documentation/cql/3.1
CQLSH

```
johnny@JPM-MacBook-Pro:/apps/dse/4.0/dse-4.0.2/bin$ ./cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.6.28 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
cqlsh> HELP

Documented shell commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>CREATE_TABLE_OPTIONS</td>
<td>SELECT</td>
</tr>
<tr>
<td>ALTER_ADD</td>
<td>CREATE_TABLE_TYPES</td>
<td>SELECT_COLUMNFAMILY</td>
</tr>
<tr>
<td>ALTER_ALTER</td>
<td>CREATE_USER</td>
<td>SELECT_EXPR</td>
</tr>
<tr>
<td>ALTER_DROP</td>
<td>DELETE</td>
<td>SELECT_LIMIT</td>
</tr>
<tr>
<td>ALTER_RENAME</td>
<td>DELETE_COLUMNS</td>
<td>SELECT_TABLE</td>
</tr>
<tr>
<td>ALTER_USER</td>
<td>DELETE_USING</td>
<td>SELECT WHERE</td>
</tr>
<tr>
<td>ALTER_WITH</td>
<td>DELETE WHERE</td>
<td>TEXT_OUTPUT</td>
</tr>
<tr>
<td>APPLY</td>
<td>DROP</td>
<td>TIMESTAMP_INPUT</td>
</tr>
<tr>
<td>ASCII_OUTPUT</td>
<td>DROP_COLUMNFAMILY</td>
<td>TIMESTAMP_OUTPUT</td>
</tr>
<tr>
<td>BEGIN</td>
<td>DROP_INDEX</td>
<td>TRUNCATE</td>
</tr>
<tr>
<td>BLOB_INPUT</td>
<td>DROP_KEYSPACE</td>
<td>TYPES</td>
</tr>
<tr>
<td>BOOLEAN_INPUT</td>
<td>DROP_TABLE</td>
<td>UPDATE</td>
</tr>
<tr>
<td>COMPOUND_PRIMARY_KEYS</td>
<td>DROP_USER</td>
<td>UPDATE_COUNTERS</td>
</tr>
<tr>
<td>CREATE</td>
<td>GRANT</td>
<td>UPDATE_SET</td>
</tr>
<tr>
<td>CREATE_COLUMNFAMILY</td>
<td>INSERT</td>
<td>UPDATE_USING</td>
</tr>
<tr>
<td>CREATE_COLUMNFAMILY_OPTIONS</td>
<td>LIST</td>
<td>UPDATE WHERE</td>
</tr>
<tr>
<td>CREATE_COLUMNFAMILY_TYPES</td>
<td>LIST_PERMISSIONS</td>
<td>USE</td>
</tr>
<tr>
<td>CREATE_INDEX</td>
<td>LIST_USERS</td>
<td>UUID_INPUT</td>
</tr>
<tr>
<td>CREATE_KEYSPACE</td>
<td>PERMISSIONS</td>
<td></td>
</tr>
<tr>
<td>CREATE_TABLE</td>
<td>REVOKE</td>
<td></td>
</tr>
</tbody>
</table>
```

©2014 DataStax. Do not distribute without consent.
CREATE KEYSSPACE johnny WITH REPLICATION =
{‘class’:’NetworkTopologyStrategy’, ‘USA’:3, ‘EU’: 2};
CREATE TABLE sporty_league (  
  team_name varchar,  
  player_name varchar,  
  jersey int,  
  PRIMARY KEY (team_name, player_name)  
);  

SELECT * FROM sporty_league WHERE team_name = 'Mighty Mutts' and player_name = 'Lucky';  

INSERT INTO sporty_league (team_name, player_name, jersey) VALUES ('Mighty Mutts','Felix', 90);  

Predicates  
  • On the partition key: = and IN  
  • On the cluster columns: <, <=, =, >=, >, IN
CQL supports having columns that contain collections of data.

The collection types include:

- Set, List and Map.

Some performance considerations around collections.

- Sometimes more efficient to denormalise further rather than use collections if intending to store lots of data.
- Favour sets over list – more performant

Watch out for collection indexing in Cassandra 2.1!
Query Tracing

- You can turn tracing on or off for queries with the TRACING ON | OFF command via the CQLSH command line utility.
- This can help you understand what Cassandra is doing and identify any performance problems.

In-Memory Tables  
(DataStax Enterprise 4.0)

CREATE TABLE users (  
    uid text,  
    fname text,  
    lname text,  
    PRIMARY KEY (uid)  
) WITH compaction={"class": 'MemoryOnlyStrategy', 'size_limit_in_mb':100}  
AND memtable_flush_period_in_ms=3600000;

• We expect that in memory column families will be on average 20-50% faster with significantly less observed variance on read queries.
• Great use case is for workloads with a lot of overwrites
• Caution: more tables = more memory = gc death spiral

Find out more:  
• http://www.datastax.com/2014/02/why-we-added-in-memory-to-cassandra
Lightweight Transactions (LWT)

Why?

- Solve a class of race conditions in Cassandra that you would otherwise need to install an external locking manager to solve.

Syntax:

```
INSERT INTO customer_account (customerID, customer_email)
VALUES (‘Johnny’, ‘jmiller@datastax.com’)  
    IF NOT EXISTS;
```

```
UPDATE  customer_account
SET customer_email='jmiller@datastax.com'
    IF customer_email='jmiller@datastax.com';
```

Example Use Case:

- Registering a user

---

Not Will Ferrell @itsWillyFerrell · Apr 5

In about 20 years, the hardest thing our kids will have to do is find a username that isn't taken.
SELECT name
FROM users
WHERE username = 'johnny';
(0 rows)

INSERT INTO users
(username, name, email,
password, created_date)
VALUES ('johnny',
'Johnny Miller',
['jmiller@datastax.com'],
'ba27e03fd9...',
'2011-06-20 13:50:00');

SELECT name
FROM users
WHERE username = 'johnny';
(0 rows)

INSERT INTO users
(username, name, email,
password, created_date)
VALUES ('johnny',
'Johnny Miller',
['jmiller@datastax.com'],
'ea24e13ad9...',
'2011-06-20 13:50:01');

This one wins!
Lightweight Transactions

```
INSERT INTO users
    (username, name, email,
     password, created_date)
VALUES ('johnny',
    'Johnny Miller',
    ['jmiller@datastax.com'],
    'ba27e03fd9...',
    '2011-06-20 13:50:00')
IF NOT EXISTS;

[applied] True

[applied] | username | name               |
-----------+-----------+--------------------
False     | johnny   | 2011-06-20 ...    | Johnny Miller

INSERT INTO users
    (username, name, email,
     password, created_date)
VALUES ('johnny',
    'Johnny Miller',
    ['jmiller@datastax.com'],
    'ea24e13ad9...',
    '2011-06-20 13:50:01')
IF NOT EXISTS;
```
Lightweight Transactions

- **Uses Paxos algorithm**
  - All operations are quorum-based i.e. we can loose nodes and it's still going to work!

- **Consequences of Lightweight Transactions**
  - 4 round trips vs. 1 for normal updates
  - Operations are done on a per-partition basis
  - Will be going across data centres to obtain consensus (unless you use LOCAL_SERIAL consistency)
  - Cassandra user will need read and write access i.e. you get back the row!

Great for 1% your app, but eventual consistency is still your friend!

**Find out more:**
- Eventual Consistency != Hopeful Consistency
  http://www.youtube.com/watch?v=A6qzx_HE3EU
Plus much, much more...

- **Counters**
  
  ```
  UPDATE UserActions SET total = total + 2
  WHERE user = 123 AND action = 'xyz';
  ```

- **Time to live (TTL)**

  ```
  INSERT INTO users (id, first, last) VALUES ('abc123', 'abe', 'lincoln') USING TTL 3600;
  ```

- **Atomic Batch Statements**

  ```
  BEGIN BATCH
    INSERT INTO users (userID, password, name) VALUES ('user2', 'ch@ngem3b', 'second user')
    UPDATE users SET password = 'ps22dhds' WHERE userID = 'user2'
    INSERT INTO users (userID, password) VALUES ('user3', 'ch@ngem3c')
    DELETE name FROM users WHERE userID = 'user2'
  APPLY BATCH;
  ```

- **New CQL features coming in Cassandra 2.0.6**
  
CQL Native Protocol
Request Pipelining

Thrift:

Native:
Notifications

Notifications are for technical events only:
- Topology changes
- Node status changes
- Schema changes

Polling

Without Notifications

Pushing

With Notifications

©2014 DataStax. Do not distribute without consent.
Asynchronous Architecture

Client Thread

Driver

Client Thread

Client Thread

©2014 DataStax. Do not distribute without consent.
Native Drivers
Native Drivers

Get them here: http://www.datastax.com/download

- Java
- C#
- Python
- C++ (beta)
- ODBC (beta)
- Clojure
- Erlang
- Node.js
- Ruby
- Plus many, many more….
Cluster cluster = Cluster.builder()
    .addContactPoints("10.158.02.40", "10.158.02.44")
    .build();

Session session = cluster.connect("a keyspace");

session.execute(
    "INSERT INTO user (username, password) 
     VALUES('johnny', 'password1234')"
);

Note: Clusters and Sessions should be long-lived and re-used.
ResultSet rs = session.execute("SELECT * FROM user");

List<Row> rows = rs.all();

for (Row row : rows) {
    String userName = row.getString("username");
    String password = row.getString("password");
}
Asynchronous Read

```java
ResultSetFuture future = session.executeAsync("SELECT * FROM user");

for (Row row : future.get()) {
    String userName = row.getString("username");
    String password = row.getString("password");
}
```

**Note:** The future returned implements Guava's ListenableFuture interface. This means you can use all Guava's Futures\(^1\) methods!

\(^1\)http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/util/concurrent/Futures.html
final ResultSetFuture future =
    session.executeAsync("SELECT * FROM user");

future.addListener(new Runnable() {
    public void run() {
        for (Row row : future.get()) {
            String userName = row.getString("username");
            String password = row.getString("password");
        }
    }
}, executor);
Parallelize Calls

```java
int queryCount = 99;

List<ResultSetFuture> futures = new ArrayList<ResultSetFuture>();

for (int i=0; i<queryCount; i++) {
    futures.add(session.executeAsync("SELECT * FROM user "+"WHERE username = "+i+");
}

for(ResultSetFuture future : futures) {
    for (Row row : future.getUninterruptibly()) {
        //do something
    }
}
```
Tip

• If you need to do a lot of work, it’s often better to make many small queries concurrently than to make one big query.
  • `executeAsync` and Futures – makes this really easy!
  • Big queries can put a high load on one coordinator
  • Big queries can skew your 99th percentile latencies for other queries
  • If one small query fails you can easily retry, if a big query than you have to retry the whole thing
PreparedStatement statement = session.prepare("INSERT INTO user (username, password) VALUES (?, ?)");

BoundStatement bs = statement.bind();

bs.setString("username", "johnny");
bs.setString("password", "password1234");

session.execute(bs);
Query query = QueryBuilder
   .select()
   .all()
   .from("aKeyspace", "user")
   .where(eq("username", "johnny"));

query.setConsistencyLevel(ConsistencyLevel.ONE);

ResultSet rs = session.execute(query);
Multi Data Center Load Balancing

- Local nodes are queried first, if none are available the request will be sent to a remote data center

```java
Cluster cluster = Cluster.builder()
    .addContactPoints("10.158.02.40", "10.158.02.44")
    .withLoadBalancingPolicy(
        new DCAwareRoundRobinPolicy("DC1"))
    .build();
```

Name of the local DC
Token Aware Load Balancing

- Nodes that own a replica of the data being read or written by the query will be contacted first

```java
Cluster cluster = Cluster.builder()
    .addContactPoints("10.158.02.40", "10.158.02.44")
    .withLoadBalancingPolicy(
        new TokenAwarePolicy(
            new DCAwareRoundRobinPolicy("DC1")))
    .build();
```

http://www.datastax.com/drivers/java/2.0/com/datastax/driver/core/policies/TokenAwarePolicy.html
Retry Policies

• The behavior to adopt when a request returns a timeout or is unavailable.

```java
Cluster cluster = Cluster.builder()
    .addContactPoints("10.158.02.40", "10.158.02.44")
    .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)
    .withLoadBalancingPolicy(new TokenAwarePolicy(new DCAwareRoundRobinPolicy("DC1")))
    .build();
```

• DefaultRetryPolicy
• DowngradingConsistencyRetryPolicy
• FallthroughRetryPolicy
• LoggingRetryPolicy

http://www.datastax.com/drivers/java/2.0/com/datastax/driver/core/policies/RetryPolicy.html
Reconnection Policies

• Policy that decides how often the reconnection to a dead node is attempted.

```
Cluster cluster = Cluster.builder()
    .addContactPoints("10.158.02.40", "10.158.02.44")
    .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)
    .withReconnectionPolicy(new ConstantReconnectionPolicy(1000))
    .withLoadBalancingPolicy(new TokenAwarePolicy(new DCAwareRoundRobinPolicy("DC1")))
    .build();
```

- ConstantReconnectionPolicy
- ExponentialReconnectionPolicy
Automatic Paging

- This was new in Cassandra 2.0
- Previously – you would select data in batches
Query Tracing

- Tracing is enabled on a per-query basis.

```java
Query query = QueryBuilder
    .select()
    .all()
    .from("akeyspace", "user")
    .where(eq("username", "johnny"))
    .enableTracing();

ResultSet rs = session.execute(query);
ExecutionInfo executionInfo = rs.getExecutionInfo();
QueryTrace queryTrace = executionInfo.getQueryTrace();
```
Connected to cluster: xerxes
Simplex keyspace and schema created.
Host (queried): /127.0.0.1
Host (tried): /127.0.0.1
Trace id: 96ac9400-a3a5-11e2-96a9-4db56cdc5fe7

<table>
<thead>
<tr>
<th>activity</th>
<th>timestamp</th>
<th>source</th>
<th>source_elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing statement</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>28</td>
</tr>
<tr>
<td>Preparing statement</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>199</td>
</tr>
<tr>
<td>Determining replicas for mutation</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>348</td>
</tr>
<tr>
<td>Sending message to /127.0.0.3</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>788</td>
</tr>
<tr>
<td>Sending message to /127.0.0.2</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>805</td>
</tr>
<tr>
<td>Acquiring switchLock read lock</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>828</td>
</tr>
<tr>
<td>Appending to commitlog</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>848</td>
</tr>
<tr>
<td>Adding to songs memtable</td>
<td>12:17:16.736</td>
<td>/127.0.0.1</td>
<td>900</td>
</tr>
<tr>
<td>Message received from /127.0.0.1</td>
<td>12:17:16.737</td>
<td>/127.0.0.2</td>
<td>34</td>
</tr>
<tr>
<td>Message received from /127.0.0.3</td>
<td>12:17:16.737</td>
<td>/127.0.0.1</td>
<td>25</td>
</tr>
<tr>
<td>Acquiring switchLock read lock</td>
<td>12:17:16.737</td>
<td>/127.0.0.2</td>
<td>672</td>
</tr>
<tr>
<td>Acquiring switchLock read lock</td>
<td>12:17:16.737</td>
<td>/127.0.0.3</td>
<td>525</td>
</tr>
<tr>
<td>Appending to commitlog</td>
<td>12:17:16.737</td>
<td>/127.0.0.2</td>
<td>692</td>
</tr>
<tr>
<td>Appending to commitlog</td>
<td>12:17:16.737</td>
<td>/127.0.0.3</td>
<td>541</td>
</tr>
<tr>
<td>Adding to songs memtable</td>
<td>12:17:16.737</td>
<td>/127.0.0.2</td>
<td>741</td>
</tr>
<tr>
<td>Adding to songs memtable</td>
<td>12:17:16.737</td>
<td>/127.0.0.3</td>
<td>583</td>
</tr>
<tr>
<td>Enqueuing response to /127.0.0.1</td>
<td>12:17:16.737</td>
<td>/127.0.0.1</td>
<td>751</td>
</tr>
<tr>
<td>Enqueuing response to /127.0.0.1</td>
<td>12:17:16.738</td>
<td>/127.0.0.2</td>
<td>950</td>
</tr>
<tr>
<td>Message received from /127.0.0.3</td>
<td>12:17:16.738</td>
<td>/127.0.0.1</td>
<td>178</td>
</tr>
<tr>
<td>Sending message to /127.0.0.1</td>
<td>12:17:16.738</td>
<td>/127.0.0.2</td>
<td>1189</td>
</tr>
<tr>
<td>Message received from /127.0.0.2</td>
<td>12:17:16.738</td>
<td>/127.0.0.1</td>
<td>249</td>
</tr>
<tr>
<td>Processing response from /127.0.0.3</td>
<td>12:17:16.738</td>
<td>/127.0.0.1</td>
<td>345</td>
</tr>
<tr>
<td>Processing response from /127.0.0.2</td>
<td>12:17:16.738</td>
<td>/127.0.0.1</td>
<td>377</td>
</tr>
</tbody>
</table>
DevCenter

- Desktop app
- friendly, familiar, productive
- Free

http://www.datastax.com/devcenter
Cassandra 2.1 Preview
CREATE TYPE `address` (  
  street text,  
  city text,  
  zip_code int,  
  phones set<text>  
)

CREATE TABLE users (  
  id uuid PRIMARY KEY,  
  name text,  
  addresses map<text, `address`>  
)

SELECT id, name, `addresses`.city, `addresses`.phones FROM users;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th><code>addresses</code>.city</th>
<th><code>addresses</code>.phones</th>
</tr>
</thead>
<tbody>
<tr>
<td>63bf691f</td>
<td>johnny</td>
<td>London</td>
<td>{'0201234567', '0796622222'}</td>
</tr>
</tbody>
</table>
User Defined Types

Considerations

- you cannot update only parts of a UDT value, you have to overwrite the whole thing every time (limitation in current implementation, may change).
- Always read entirely under the hood (as of the current implementation at least)
- UDTs are not meant to store large and complex "documents" as of their current implementation, but rather to help make the denormalization of short amount of data more convenient and flexible.
- It is possible to use a UDT as type of any CQL column, including clustering ones.

Find out more:
- http://www.datastax.com/dev/blog/cql-in-2-1
SECONDARY INDEXES ON COLLECTIONS

CREATE TABLE songs (  
id uuid PRIMARY KEY,  
artist text,  
album text,  
title text,  
data blob,  
tags set<text>
);

CREATE INDEX song_tags_idx ON songs(tags);

SELECT * FROM songs WHERE tags CONTAINS 'blues';

<table>
<thead>
<tr>
<th>id</th>
<th>album</th>
<th>artist</th>
<th>tags</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5027b27e</td>
<td>Country Blues</td>
<td>Lightnin' Hopkins</td>
<td>{'acoustic', 'blues'}</td>
<td>Worrying My Mind</td>
</tr>
</tbody>
</table>
Secondary indexes on map keys

- If you prefer indexing the map keys, you can do so by creating a `KEYS` index and by using `CONTAINS KEY`.

```
CREATE TABLE products (  
id int PRIMARY KEY,  
description text,  
price int,  
categories set<text>,  
features map<text, text>
);

CREATE INDEX feat_key_index ON products(KEYS(features));

SELECT id, description  
FROM products  
WHERE features CONTAINS KEY 'refresh-rate';
```

<table>
<thead>
<tr>
<th>id</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34134</td>
<td>120-inch 1080p 3D plasma TV</td>
</tr>
</tbody>
</table>
Counters++

- simpler implementation, no more edge cases
- possible to properly repair now
- significantly less garbage and internode traffic generated
- better performance for 99% of uses
CREATE TABLE notifications (  
    target_user text,  
    notification_id timeuuid,  
    source_id uuid,  
    source_type text,  
    activity text,  
    PRIMARY KEY (target_user, notification_id)  
)  
WITH CLUSTERING ORDER BY (notification_id DESC)  
AND caching = 'rows_only'  
AND rows_per_partition_to_cache = '3';
2.1 Roadmap

- Beta1 - 20th Feb
- Beta2 - ?
- RC - ?
- Final release currently **mid-2014**
Come and talk to us!
Find Out More

DataStax:
• http://www.datastax.com

Getting Started:
• http://www.datastax.com/documentation/gettingstarted/index.html

Training:
• http://www.datastax.com/training

Downloads:
• http://www.datastax.com/download

Documentation:
• http://www.datastax.com/docs

Developer Blog:
• http://www.datastax.com/dev/blog

Community Site:
• http://planetcassandra.org

Webinars:
• http://planetcassandra.org/Learn/CassandraCommunityWebinars
THANK YOU