



Cassandra: Practical Experience

- **Objective:** Cassandra basics for **users**
 - **not** administration

- **Pre-installed** Apache Cassandra at our VM

stratus

- <http://stratus.fi.muni.cz> template "PA195 - Cassandra"
- # cd apache-cassandra-3.10/

Template ID 249

- or **Install** Cassandra

- **Straightforward** installation from a tar archive
- <http://cassandra.apache.org/download/>
 - store the tar.gz file to /var/tmp/
 - unpack the archive: \$ tar xvzf apache-cassandra.tar.gz



Cassandra Directory

- **Inspect** the installation **directory**
 - binaries
 - configuration (conf/cassandra.yaml)
 - logs
 - tools
 - ...
- **Start in Stratus VM**
 - # cd apache-cassandra-3.10/
 - # bin/cassandra

CQLSH: CQL Shell

- **Connect** to a running Cassandra node
 - run **CQL shell**: `cqlsh`
- **CQLSH basic** features & commands
 - CQL syntax: CREATE, INSERT, SELECT, DROP,...
 - `help`
 - use `tab` for command **completion**
 - use `up/down` arrows for command **history**
 - use `ctrl+R` for reverse **command search**

Query language docs: <http://cassandra.apache.org/doc/latest/cql/index.html>

Task 1: Keyspace and Table

- **Keyspace** is like “database” in RDBMS

```
DESCRIBE KEYSPACES;
```

```
CREATE KEYSPACE pa195 WITH REPLICATION = { 'class':  
'SimpleStrategy', 'replication_factor': 1 };
```

```
USE pa195;
```

```
DESCRIBE KEYSPACE;
```

```
CREATE TABLE students (  
    uco int PRIMARY KEY,  
    first_name text,  
    surname text,  
    active boolean );
```

```
DESCRIBE TABLE students;
```

Task 2: Data Manipulation and Query



```
INSERT INTO students (uco, first_name, surname, active)
    VALUES (4335, 'David', 'Novák', false);
INSERT INTO students (uco, first_name, surname, active)
    VALUES (123, 'Some name', 'Some surname', true);
SELECT * FROM students;
SELECT * FROM students WHERE first_name = 'David';
InvalidRequest: Error from server: code=2200 [Invalid query] ...
CREATE INDEX ON students (first_name);
SELECT * FROM students WHERE first_name = 'David' ;
```

Dynamic Columns with CQL 3

```
CREATE TABLE students2 (  
    uco int,  
    field text,  
    value text,  
    PRIMARY KEY (uco, field) );  
  
INSERT INTO students2 (uco, field, value)  
    VALUES (4335, 'first_name', 'David');  
INSERT INTO students2 (uco, field, value)  
    VALUES (4335, 'surname', 'Novák');  
INSERT INTO students2 (uco, field, value)  
    VALUES (4335, 'email', 'david@novak.name');  
  
SELECT * FROM students2;
```


Partition Key

```
CREATE TABLE students2 (  
    uco int,  
    field text,  
    value text,  
    PRIMARY KEY (uco, field) );
```

- Keys work like this:
 - **Primary key** is compulsory
 - **Unique** per table (row key)
 - **Partition key** - the key to **partition** table **to nodes**
 - Records with the same key are stored on the same node
 - The **first column** of the primary key (or a set of columns)
 - **Clustering columns**
 - Determine **per-partition clustering**, i.e., the order for physical storing rows

Data Sharding in Columnar Systems



System	Terminology
BigTable	tablets
HBase	regions
Cassandra	partitions

část 1
(partition 1)

část 2
(partition 2)

user_id (row key)	login	name
1	honza	Jan...
4	david	David...
...		
1000	karel	Karel...
1001	irena	Irena...
1003	jirka	Jiří...
...		
2000		
....		

Tables: Dynamic Columns

- **Values** can use “collection” types:
 - **set** – **unordered** unique values
 - **list** – **ordered** list of elements
 - **map** – name + value pairs
 - a way to **realize super-columns**
- **Realization** of the original idea of **free columns**
 - **Internally**, all **values** in collections as individual **columns**
 - Cassandra can well **handle** “unlimited” number of columns

Tables: Dynamic Columns (2)

```
CREATE TABLE users (  
    login text PRIMARY KEY,  
    name text,  
    emails set<text>, // column of type "set"  
    profile map<text, text> // column of type "map"  
);  
  
INSERT INTO users (login, name, emails, profile)  
VALUES ( 'honza', 'Jan Novák', { 'honza@novak.cz' },  
        { 'colorschema': 'green', 'design': 'simple' }  
);  
  
UPDATE users  
SET emails = emails + { 'jn@firma.cz' }  
WHERE login = 'honza';
```

Search with CQL

- The **syntax** of CQL selects is **similar** to SQL
 - But search just in **one table** (no joins)

```
SELECT <selectExpr>
FROM [<keyspace>.<table>]
[WHERE <clause>]
[ORDER BY <clustering_colname> [DESC]]
[LIMIT m];
```

```
SELECT column_name, column_value
FROM mytable
WHERE row_id=3
ORDER BY column_value;
```

CQL: Limitations on “Where” Part

- The **search condition** can be:

- on columns in the **partition key**

- And only using **operators** == and IN

```
... WHERE row_id IN (3, 4, 5)
```

- Therefore, the query hits only **one or several** physical **nodes** (not all)

- on columns from the **clustering key**

- Especially, if there is also condition on the **partitioning** key

```
... WHERE row_id=3 AND column_name='login'
```

- If it is not, the system must **filter all entries**

```
SELECT * FROM mytable
```

```
WHERE column_name IN ('login', 'name') ALLOW FILTERING;
```

```
CREATE TABLE mytable (  
    row_id int,  
    column_name text,  
    column_value text,  
    PRIMARY KEY  
        (row_id, column_name)  
);
```

CQL: Limitations on “Where” Part (1)



- Other **columns** can be **queried**
 - If there is an **index** built the column
- **Indexes** can be built also on **collection** columns (set, list, map)
 - And then **queried** by CONTAINS like this

```
CREATE INDEX ON users(emails);
CREATE INDEX ON users(KEYS (profile));
SELECT login FROM users
    WHERE emails CONTAINS 'jn@firma.cz';
SELECT * FROM users
    WHERE profile CONTAINS KEY 'colorschema';
```



Task 3: Table for people & import

- Get data from study materials
 - people-cassandra.json
- Check its structure & create a table “people” for it
- Import the json
 - Convert JSON array into a series of INSERT command
 - or use people-cassandra-insert.json
 - `# time bin/cqlsh -f people-cassandra-insert.json`

Task 4: Querying for people



1. Total number of rows in people
2. Select records with surname "Hanna"
3. Find people with salary greater than 3000
4. Distinct surnames
 - a. Count distinct surnames (it is 640) ???
5. Expire "email" in the record id 1190 in 60s
6. Delete "number" from address of record id 1190
7. Expire the record with id 1190 in 60s

Copy & paste the queries and their responses to the file `cassandra.txt` and upload to the IS's vault! *In case you fail to form any query, make a notice there!*



Task 5: Cassandra's Stress Test

- **Task:** use a cassandra tool to test **performance**
 - `cassandra$ tools/bin/cassandra-stress help`
- **Read** these docs & manuals
 - <https://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsCStress.html>
 - <http://www.datastax.com/dev/blog/>
- **Modify** and use **the** YAML configs
 - Either in `cassandra/tools/*.yaml`
 - or use examples from the manual
- **Report** the insert & query performance
 - Find out what the values in the result mean



Task 5: Cassandra's Stress Test

Show defaults

```
$ tools/bin/cassandra-stress help -schema
```

Insert (write) one million rows

```
$ tools/bin/cassandra-stress write n=1000000 -rate threads=50
```

Read two hundred thousand rows.

```
$ tools/bin/cassandra-stress read n=200000 -rate threads=50
```

Read rows for a duration of 3 minutes.

```
$ tools/bin/cassandra-stress read duration=3m -rate threads=50
```

Read 200,000 rows without a warmup of 50,000 rows first.

```
$ tools/bin/cassandra-stress read n=200000 no-warmup -rate threads=50
```

References

- I. Holubová, J. Kosek, K. Minařík, D. Novák. Big Data a NoSQL databáze. Praha: Grada Publishing, 2015. 288 p.
- RNDr. Irena Holubova, Ph.D. MMF UK course NDBI040: Big Data Management and NoSQL Databases
- <http://www.datastax.com/documentation/cassandra/1.2/>
- <http://www.datastax.com/documentation/cassandra/2.0/>
- <http://wiki.apache.org/cassandra/>