Modern Database Concepts

Practicals: Column-family stores

Doc. RNDr. Irena Holubova, Ph.D.
holubova@ksi.mff.cuni.cz

Apache Cassandra

- Developed at Facebook
- Initial release: 2008
- Stable release: 2013
  - Apache Licence
- Written in: Java
- OS: cross-platform
- Operations:
  - CQL (Cassandra Query Language)
  - MapReduce support
    - Can cooperate with Hadoop (data storage instead of HDFS)

http://cassandra.apache.org/

CQL Shell

`cqlsh`

- Run the CQL shell
- By default it connects to localhost on default port
  - The database should run on our testing server

`clear`

- Clear the shell terminal window

`exit/quit`

- Terminate the current connection

Cassandra
CQL

- Cassandra query language
- SQL-like commands
  - CREATE, ALTER, UPDATE, DROP, DELETE, TRUNCATE, INSERT, …
- Much simpler than SQL
  - Does <u>not allow</u> joins or subqueries
  - Where clauses are simple
  - …
- Different approach than column families (since CQL 3 called tables)
  - More general
  - Closer to key/value and document databases

Cassandra

Working with a Table – Collections

- Collection types:
  - **set** – a set of <u>unique</u> values
    - Returned in <u>alphabetical</u> order, when queried
  - **list** – ordered list of elements
    - Can store the same value multiple times
    - Returned sorted according to index value in the list
  - **map** – name + value pairs
  - Each element is internally stored as one Cassandra column

  => Each element can have an individual time-to-live

Cassandra

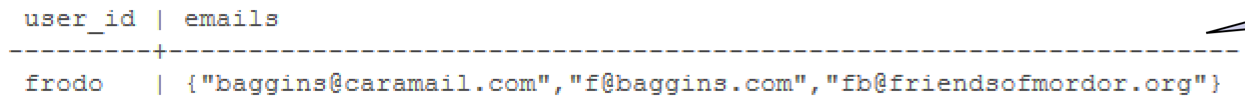
Working with a Table – Set

```
CREATE TABLE users (
  user_id text PRIMARY KEY,
  first_name text,
  last_name text,
  emails set<text> );

INSERT INTO users (user_id, first_name, last_name, emails)
VALUES('frodo', 'Frodo', 'Baggins', {'f@baggins.com', 'baggins@gmail.com'});

UPDATE users SET emails = emails + {'fb@friendsofmordor.org'}
WHERE user_id = 'frodo';

SELECT user_id, emails FROM users WHERE user_id = 'frodo';
```



```
 user_id | emails
---------+-------------------------------------------------------------------
 frodo   | {"baggins@caramail.com","f@baggins.com","fb@friendsofmordor.org"}
```

order

```
UPDATE users SET emails = emails - {'fb@friendsofmordor.org'}
WHERE user_id = 'frodo';

UPDATE users SET emails = {} WHERE user_id = 'frodo';
```

Cassandra

Working with a Table – List

```
ALTER TABLE users ADD top_places list<text>;

UPDATE users SET top_places = [ 'rivendell', 'rohan' ]
WHERE user_id = 'frodo';

UPDATE users SET top_places = [ 'the shire' ] + top_places
WHERE user_id = 'frodo';

UPDATE users SET top_places = top_places + [ 'mordor' ]
WHERE user_id = 'frodo';

UPDATE users SET top_places[2] = 'riddermark'
WHERE user_id = 'frodo';

DELETE top_places[3] FROM users WHERE user_id = 'frodo';

UPDATE users SET top_places = top_places - ['riddermark']
WHERE user_id = 'frodo';
```

Cassandra

Working with a Table – Map

```
ALTER TABLE users ADD todo map<timestamp, text>;

UPDATE users SET todo = { '2012-9-24' : 'enter mordor',
'2012-10-2 12:00' : 'throw ring into mount doom' }
WHERE user_id = 'frodo';

UPDATE users SET todo['2012-10-2 12:00'] =
'throw my precious into mount doom'
WHERE user_id = 'frodo';

INSERT INTO users (user_id, todo) VALUES ('frodo', {
'2013-9-22 12:01' : 'birthday wishes to Bilbo',
'2013-10-1 18:00' : 'Check into Inn of Prancing Pony' });

DELETE todo['2012-9-24'] FROM users
WHERE user_id = 'frodo';
```

Cassandra

Querying

optional

```
SELECT select_expression
FROM keyspace_name.table_name
WHERE relation AND relation ...
GROUP BY columns
ORDER BY ( clustering_key ( ASC | DESC )...)
LIMIT n
ALLOW FILTERING
```

- `select_expression`:
  - List of columns
  - `DISTINCT`
  - `COUNT`
  - Aliases (`AS`)
  - `TTL(column_name)`
  - `WRITETIME(column_name)`

Cassandra

Querying

- `relation`:
  - column_name (= | < | > | <= | >=) key_value
  - column_name IN ((key_value,...))
  - TOKEN (column_name, ...) (= | < | > | <= | >=) (term | TOKEN (term, ...))

hash

- `term`:
  - constant
  - set/list/map

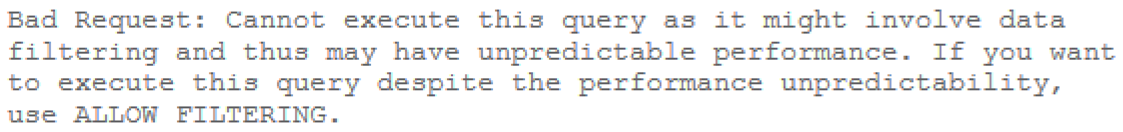
Cassandra

Querying – `GROUP BY`

- Groups rows of a table according to certain columns
- Only groupings induced by primary key columns are allowed!
- Aggregate functions
  - `COUNT, MIN, MAX, SUM, AVG`
  - User-defined
  - When a non-grouping column is selected without an aggregate function, the first value encounter is always returned

Cassandra
Querying – `ALLOW FILTERING`

- Non-filtering queries
  - Queries where we know that all records read will be returned (maybe partly) in the result set
  - Have predictable performance
- Attempt a potentially expensive (i.e., filtering) query
- `ALLOW FILTERING`
  - “We know what we are doing”
  - Usually together with `LIMIT n`



```
Bad Request: Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want
to execute this query despite the performance unpredictability,
use ALLOW FILTERING.
```

Cassandra

Querying – ALLOW FILTERING

```
CREATE TABLE users (
    username text PRIMARY KEY,
    firstname text,
    lastname text,
    birth_year int,
    country text
);
CREATE INDEX ON users(birth_year);
```

query performance proportional to the amount of data returned

```
SELECT * FROM users;

SELECT firstname, lastname FROM users
WHERE birth_year = 1981;
```

Cassandra

Querying – `ALLOW FILTERING`

```
SELECT firstname, lastname
FROM users
WHERE birth_year = 1981 AND country = 'FR';
```

No guarantee that Cassandra won't have to scan large amount of data even if the result is small

```
SELECT firstname, lastname
FROM users
WHERE birth_year = 1981 AND country = 'FR'
ALLOW FILTERING;
```

Assignment

- Chose your unique problem domain
  - E.g., the results of football matches of various teams
- For your selected problem domain think about an application that uses CQL (create tables, store data, create meaningful queries)
- Submit a script with respective commands for Cassandra + explanatory <u>comments</u>

References

- Cassandra Web http://cassandra.apache.org/
- CQL: http://cassandra.apache.org/doc/latest/cql/