DATASTAX

Chicago 2015

CQL: This is not the SQL you are looking for

Aaron Ploetz

DATASTAX

Wait... CQL is *not* SQL?

- CQL3 introduced in Cassandra 1.1.
- CQL is beneficial to new users who have a relational background (which is most of us).
- However similar, CQL is **NOT** a direct implementation of SQL.
- New users leave themselves open to issues and frustration when they use CQL with SQL-based expectations.

$ whoami

Aaron Ploetz

- @APloetz 
- Lead Database Engineer 
- Using Cassandra since version 0.8.
- Contributor to the Cassandra tag on 
- 2014/15 DataStax MVP for Apache Cassandra

1	SQL features/keywords not present in CQL
2	Differences between CQL and SQL keywords
3	Secondary Indexes
4	Anti-Patterns
5	Questions

DATASTAX

SQL features/keywords not present in CQL

- JOINs
- LIKE
- Subqueries
- Aggregation
- Arithmetic
  - Except for counters and collections.

DATASTAX

Differences between CQL and SQL keywords

- WHERE
- PRIMARY KEY
- ORDER BY
- IN
- DISTINCT
- COUNT
- LIMIT
- INSERT vs. UPDATE (“upsert”)

DATASTAX

WHERE

- Only supports AND, IN, =, >, >=, <, <=.
  - Some only function under certain conditions.
  - Also: CONTAINS, CONTAINS KEY for indexed collections.
  - Does not exist: OR, !=
- Conditions can only operate on PRIMARY KEY components, and in the defined order of the keys.

WHERE *(cont)*

```
CREATE TABLE shipcrewregistry
(shipname text, lastname text, firstname
text, citizenid uuid,
aliases set<text>, PRIMARY KEY
(shipname, lastname, firstname,
citizenid));
```


```
SELECT * FROM shipcrewregistry WHERE
shipname='Serenity';
```

- Start with partition key(s); cannot skip PRIMARY KEY components.

DATASTAX

ALLOW FILTERING

- Actually I lied, you can skip primary key components if you apply the ALLOW FILTERING clause.
- **SELECT * FROM shipcrewregistry WHERE lastname='Washburne';**

ALLOW FILTERING *(cont)*

- **SELECT * FROM shipcrewregistry WHERE lastname='Washburne' ALLOW FILTERING;**
- But I don't recommend that.
- ALLOW FILTERING pulls back all rows and then applies your WHERE conditions.
- The folks at DataStax have proposed some alternate names...
- Bottom line, if you are using ALLOW FILTERING, you are doing it wrong.

DATASTAX

PRIMARY KEY

- PRIMARY KEYs function differently between Cassandra and relational databases.
- Cassandra uses primary keys to determine data distribution and on-disk sort order.
  - Partition keys are the equivalent of “old school” row keys.
  - Clustering keys determine on-disk sort order within a partitioning key.

DATASTAX

ORDER BY

- One of the most misunderstood aspects of CQL.
- Can only order by clustering columns, in the key order of the clustering columns listed in the table definition (CLUSTERING ORDER).
- Which means, that you really don't need ORDER BY.
- So what does it do? It can reverse the sort direction (ASCending vs. DESCending) of the first clustering column.

PRIMARY KEY / ORDER BY Example: Table Definition

DATASTAX

```
CREATE TABLE postsByUserYear
(userid text, year bigint,          tag text, posttime
timestamp,      content text, postid UUID,
PRIMARY KEY ((userid, year),      posttime,
tag))                  WITH CLUSTERING ORDER BY
(posttime desc, tag asc);
```

PRIMARY KEY / ORDER BY Example: Queries

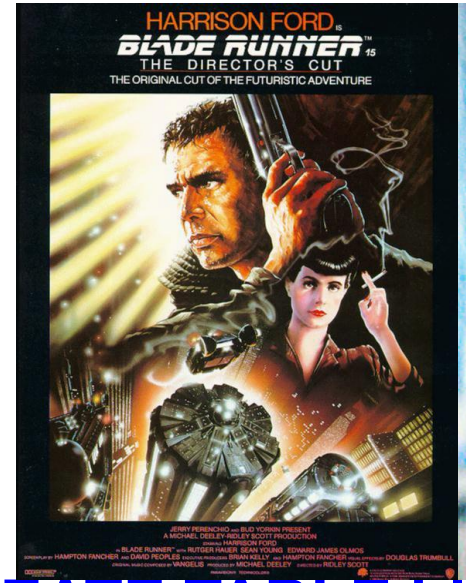
DATASTAX

- SELECT * FROM postsByUserYear WHERE userid='2';
- SELECT * FROM postsByUserYear ORDER BY posttime;
- SELECT * FROM postsByUserYear WHERE userid='2' AND year=2015 ORDER BY posttime DESC;
- SELECT * FROM postsByUserYear WHERE userid='2' AND year=2015 ORDER BY tag;

IN

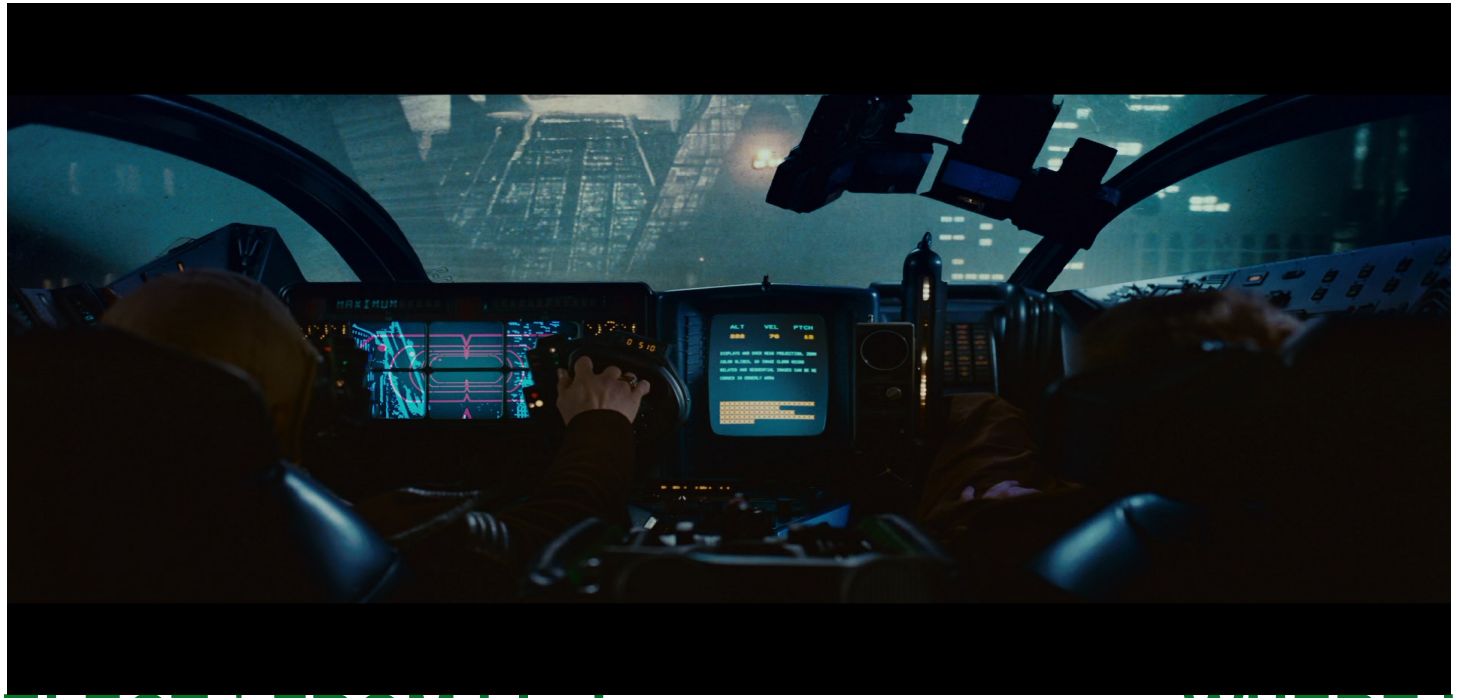
- Can only operate on the last partition key and/or the last clustering key.
  - *And only when the first partition/clustering keys are restricted by an equals relation.*
- Does not perform well...especially with large clusters.

Testing IN



```
CREATE TABLE bladerunners (id text, type    text, ts
timestamp, name text, data text,  PRIMARY KEY (id));
```

Testing IN *(cont)*



- SELECT * FROM bladerunners WHERE id IN ('B26354','B26354');

DATASTAX

DISTINCT

- Returns a list of the queried partition keys.
- Can only operate on partition key column(s).
- In Cassandra DISTINCT returns the partition (row) keys, so it is a fairly light operation (relative to the size of the cluster and/or data set).
- Whereas in the relational world, DISTINCT is a very resource intensive operation.

DATASTAX

COUNT

- Counts the number of rows returned, dependent on the WHERE clause.
- Does not aggregate.
- Similar to its RDBMs counterpart.
- Can be (inadvertently) restricted by LIMIT.
- Resource intensive command; especially because it has to scan each row in the table (which may be on different nodes), and apply the WHERE conditions.

DATASTAX

Limit

- Limits your query to N rows (where N is a positive integer).
- **SELECT * FROM bladerunners LIMIT 2;**
- Does not allow you to specify a start point.
  - You *cannot* use LIMIT to “page” through your result set.

DATASTAX

Cassandra “Upserts”

- Under the hood, INSERT and UPDATE are treated the same by Cassandra.
- Colloquially known as an “Upsert.”
- Both INSERT and UPDATE operations require the complete PRIMARY KEY.

DATASTAX

So why the different syntax?

- Flexibility. Some situations call for one or the other.
- Counter columns/tables can only be incremented with an UPDATE.
- INSERTs can save you some dev time in the application layer if your PRIMARY KEY changes.

“Upsert” example

- UPDATE bladerunners SET data='This guy is a one-man slaughterhouse.',name='Harry Bryant',ts='2015-03-30 14:47:00-0600',type='Captain' WHERE id='B16442';

- UPDATE bladerunners SET data = 'Drink some for me, huh pal?' WHERE id='B16442';

DATASTAX

“Upsert” example *(cont)*

- INSERT INTO bladerunners (id, type, ts, data, name) VALUES ('B29591','Blade Runner','2015-03-30 14:34:00-0600','Captain Bryant would like a word.','Eduardo Gaff');

- INSERT INTO bladerunners (id,data) VALUES ('B29591','It''s too bad she won't live. But then again, who does?');

DATASTAX

Secondary Indexes

- Cassandra provides secondary indexes to allow queries on non-partition key columns.
- In 2.1.x you can even create indexes on collections and user defined types.
- Designed for convenience, **not** for performance.
- Does not perform well on high-cardinality columns.
- Extremely low cardinality is also not a good idea.
- Low performance on a frequently updated column.
- In my opinion, try to avoid using them all together.

DATASTAX

Anti-Patterns

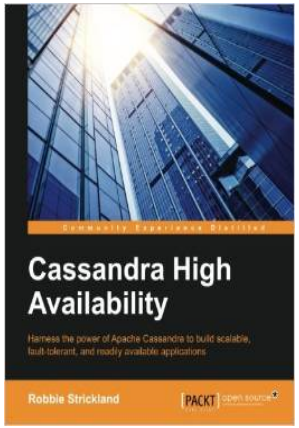
- Multi-Key queries: IN
- Secondary Index queries
- DELETEs or INSERTing null values

Summary

- While CQL is designed to make use of our previous experience using SQL, it is important to remember that the two do not behave the same.
- Even if you are at an expert level in SQL, read the CQL documentation before making any assumptions.

Additional Reading

- Getting Started with Time Series Data Modeling – Patrick McFadin
- SELECT – DataStax CQL 3.1 documentation
- Counting Keys in Cassandra – Richard Low
- Cassandra High Availability – Robbie Strickland



DATASTAX

Questions?