Module 4

NoSQL Databases

María del Mar Roldán – University of Málaga

Learning Objectives

- Understand the Cassandra data model
- Introduce cqlsh
- Understand and use the DDL subset of CQL
- Introduce DevCenter
- Understand and use the DML subset of CQL
- ~~Understand basics of data modeling~~ (Challenge)

What is the syntax of the INSERT statement?

INSERT INTO table_name (column1, column2 ...)

VALUES (value1, value2 ...)

- Inserts a row into a table
  - Must specify columns to insert values into
    - Primary key columns are mandatory (identify the row)
    - Other columns do not have to have values
      - Non-existent ‘values’ do not take up space
- Atomicity and isolation
  - Inserts are atomic
    - All values of a row are inserted or none
  - Inserts are isolated
    - Two inserts with the same values in primary key columns will not interfere-executed one after another

What is the syntax of the INSERT statement?

- To insert a row into a table

```
CREATE TABLE albums_by_performer (
performer VARCHAR,
year INT,
title VARCHAR,
genre VARCHAR,
PRIMARY KEY (performer, year, title)
) WITH CLUSTERING ORDER BY (year DESC, title ASC);

INSERT INTO albums_by_performer (performer,year,title,genre)
VALUES ('The Beatles', 1966, 'Revolver', 'Rock');
INSERT INTO albums_by_performer (performer, year, title)
VALUES ('The Beatles', 1995, 'Beatlemania');
```

What is the syntax of the INSERT statement?

performer	year	title	Genre
The Beatles	1995	Beatlemania	
The Beatles	1966	Revolver	Rock

Apache Cassandra - UPDATE

```
UPDATE <keyspace>.<table>
SET column_name1 = value, column_name2 = value,
WHERE primary_key_column = value
```

- Updates columns in an existing row
  - Row must be identified by values in primary key columns
  - Primary key columns cannot be updated
  - An existing value is replaced with a new value
  - A new value is added if a value for a column did not exist before
- Atomicity and isolation
  - Updates are atomic
    - All values of a row are updated or none
  - Updates are isolated
    - Two updates with the same values in primary key columns will not interfere – executed one after another

Apache Cassandra - UPDATE

- To update a row in a table

```
UPDATE albums_by_performer
SET genre = 'Rock'
WHERE performer = 'The Beatles' AND
year = 1995 AND
title = 'Beatlemania';
```

Apache Cassandra - UPDATE

- Before update

performer	year	title	Genre
The Beatles	1995	Beatlemania	
The Beatles	1966	Revolver	Rock

- After update

performer	year	title	Genre
The Beatles	1995	Beatlemania	Rock
The Beatles	1966	Revolver	Rock

Apache Cassandra - UPDATE

- UPdate + inSERT
  - Both UPDATE and INSERT are write operations
  - No reading before writing
- Term “upsert” denotes the following behavior
  - INSERT updates or overwrites an existing row
    - When inserting a row in a table that already has another row with the same values in primary key columns
  - UPDATE inserts a new row
    - When a to-be-updated row, identified by values in primary key columns, does not exist
  - Upserts are legal and do not result in error or warning messages

Apache Cassandra - UPDATE

- Introduces a new clause IF NOT EXISTS for inserts
  - Insert operation executes if a row with the same primary key does not exist
  - Uses a consensus algorithm called Paxos to ensure inserts are done serially
  - Multiple messages are passed between coordinator and replicas with a **large performance penalty**
  - [applied] column returns true if row does not exist and insert executes
  - [applied] column is false if row exists and the existing row will be returned

Apache Cassandra – IF NOT EXISTS

INSERT INTO albums_by_performer (performer,year,title)
VALUES ('The Beatles', 1966, 'Revolver') IF NOT EXISTS;

[applied]
true

INSERT INTO albums_by_performer (performer,year,title)
VALUES ('The Beatles', 1966, 'Revolver') IF NOT EXISTS;

[applied]	performer	year
false	The Beatles	1966

Apache Cassandra - IF

- Update uses IF to verify the value for column(s) before execution
  - [applied] column returns true if condition(s) matches and update written
  - [applied] column is false if condition(s) do not match and the current row will be returned

Apache Cassandra

UPDATE albums_by_performer SET year = 1968 WHERE performer = 'The Beatles' IF title = 'Revolver';

[applied]
true

UPDATE albums_by_performer SET year = 1968 WHERE performer = 'The Beatles' IF title = 'Revolver' AND year = 1967;

[applied]	performer	year
false	The Beatles	1966

Apache Cassandra - TTL

- Time-to-live (TTL) defines expiring columns
- INSERT and UPDATE can optionally assign data values a time-to-live
  - TTL is specified in seconds
  - Expired columns/values are eventually deleted
  - With no TTL specified, columns/values never expire
- TTL is useful for automatic deletion
  - When data gets outdated after some time
  - When only most recent data is needed
  - Older data may be archived elsewhere by a background process
  - Helps keep the size of a table and its partitions manageable
  - Restricts the data view to most recent data

Apache Cassandra – TTL

- To store a row for 86400 seconds (1 day)
  - Re-inserting the same row before it expires will overwrite TTL

```
INSERT INTO track_ratings_by_user
(user,activity,rating,album_title,album_year,track_title)
VALUES (uuid(),now(),5,'Revolver',1966,'Yellow
Submarine')
USING TTL 86400;
```

Apache Cassandra

- To store a column value for 30 seconds
  - Only column ‘rating’ for this row is affected by TTL

```
UPDATE track_ratings_by_user
USING TTL 30
SET rating = 0
WHERE user = 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328 AND
activity = dbf3fbfc-9fe4-11e3-8d05-425861b86ab6;
```

Apache Cassandra - DELETE

- Deletes a partition, a row or specified columns in a row
  - Row must be identified by values in primary key columns
  - Primary key columns cannot be deleted without deleting the whole row
- To delete a partition from a table
- To delete a row from a table
- To delete a column from a table row

Apache Cassandra - DELETE

- Deletes a partition, a row or specified columns in a row
  - Row must be identified by values in primary key columns
  - Primary key columns cannot be deleted without deleting the whole row
- **To delete a partition from a table**

```
DELETE FROM track_ratings_by_user
WHERE user = 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328;
```

Apache Cassandra - DELETE

- Deletes a partition, a row or specified columns in a row
  - Row must be identified by values in primary key columns
  - Primary key columns cannot be deleted without deleting the whole row
- To delete a partition from a table
- **To delete a row from a table**

```
DELETE FROM track_ratings_by_user
WHERE user = 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328 AND activity = dbf3fbfc-9fe4-11e3-8d05-425861b86ab6;
```

Apache Cassandra - DELETE

- Deletes a partition, a row or specified columns in a row
  - Row must be identified by values in primary key columns
  - Primary key columns cannot be deleted without deleting the whole row
- To delete a partition from a table
- To delete a row from a table
- **To delete a column from a table row**

DELETE rating FROM track_ratings_by_user

WHERE user = 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328 AND activity = dbf3fbfc-9fe4-11e3-8d05-425861b86ab6;

Apache Cassandra - TRUNCATE

- TRUNCATE removes all rows in a table
  - The table definition (schema) is not affected

```
TRUNCATE track_ratings_by_user;
```

Apache Cassandra - LIST

- CQL list – defining and inserting
  - Collection column cannot be part of a primary key

```
CREATE TABLE song (
id UUID PRIMARY KEY,
title VARCHAR,
songwriters LIST<VARCHAR>
);
INSERT INTO song (id, title, songwriters)
VALUES (uuid(),
'I Want to Hold Your Hand', ['John', 'Paul']);
```

Apache Cassandra - LIST

- CQL list – defining and inserting
  - Collection column cannot be part of a primary key

```
CREATE TABLE song (
id UUID PRIMARY KEY,
title VARCHAR,
songwriters LIST<VARCHAR>
);
INSERT INTO song (id, title, songwriters)
VALUES (uuid(),
'I Want to
```

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['John', 'Paul']	I Want to Hold Your Hand

Apache Cassandra - LIST

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['John', 'Paul']	I Want to Hold Your Hand

UPDATE song SET songwriters = songwriters +

['Paul', 'Jonathan']

WHERE id = 252608cb-0f56-4cf3-82ee-b7fe00f3920f;

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['John', 'Paul', 'Paul', 'Jonathan']	I Want to Hold Your Hand

Apache Cassandra - LIST

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['John', 'Paul', 'Paul', 'Jonathan']	I Want to Hold Your Hand

UPDATE song SET songwriters = ['Patrick'] + songwriters

WHERE id = 252608cb-0f56-4cf3-82ee-b7fe00f3920f;

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['Patrick', 'John', 'Paul', 'Paul', 'Jonathan']	I Want to Hold Your Hand

Apache Cassandra - LIST

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['Patrick', 'John', 'Paul', 'Paul', 'Jonathan']	I Want to Hold Your Hand

UPDATE song SET songwriters[3] = 'Ringo'

WHERE id = 252608cb-0f56-4cf3-82ee-b7fe00f3920f;

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['Patrick', 'John', 'Paul', 'Ringo', 'Jonathan']	I Want to Hold Your Hand

Apache Cassandra - LIST

id	songwriters	title
252608cb-0f56-4cf3-82ee-b7fe00f3920f	['Patrick', 'John', 'Paul', 'Ringo', 'Jonathan']	I Want to Hold Your Hand

UPDATE song SET songwriters = songwriters -

['Patrick', 'Jonathan', 'Ringo']

WHERE id = 252608cb-0f56-4cf3-82ee-b7fe00f3920f;

id	songwriters	title
252608cb-0f56- …	['John', 'Paul']	I Want to Hold Your Hand

Apache Cassandra - LIST

id	songwriters	title
252608cb-0f56- …	['John', 'Paul']	I Want to Hold Your Hand

DELETE songwriters[0], songwriters[1] FROM song

WHERE id = 252608cb-0f56-4cf3-82ee-b7fe00f3920f;

id	songwriters	title
252608cb-0f56- …		I Want to Hold Your Hand

Apache Cassandra - MAP

- CQL map – defining and inserting
  - Collection column cannot be part of a primary key

```
CREATE TABLE album (
title VARCHAR,
year INT,
tracks MAP<INT,VARCHAR>,
PRIMARY KEY ((title, year))
);
INSERT INTO album (title, year, tracks)
VALUES ('Revolver', 1966, {1: 'Taxman', 2: 'Eleanor Rigby'});
```

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby'}

Apache Cassandra - MAP

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby'}

```
UPDATE album SET tracks[14] = 'Yellow Submarine'
WHERE title = 'Revolver' AND year = 1966;
```

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby', 14: 'Yellow Submarine'}

Apache Cassandra - MAP

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby', 14: 'Yellow Submarine'}

UPDATE album SET tracks[14] = 'Tomorrow Never Knows'
WHERE title = 'Revolver' AND year = 1966;

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby', 14: 'Tomorrow Never Knows'}

Apache Cassandra - MAP

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby', 14: 'Tomorrow Never Knows'}

DELETE tracks[14] FROM album
WHERE title = 'Revolver' AND year = 1966;

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby'}

Apache Cassandra - MAP

title	year	tracks
Revolver	1966	{1: 'Taxman', 2: 'Eleanor Rigby'}

DELETE tracks FROM album
WHERE title = 'Revolver' AND year = 1966;

title	year	tracks
Revolver	1966	

Apache Cassandra - TUPLE

```
CREATE TABLE user (
id UUID PRIMARY KEY,
email text,
name text,
preferences set<text>,
equalizer frozen<tuple<float,float,float,float,float,
float,float,float,float,float>>
);
INSERT INTO user (id, equalizer)
VALUES (uuid(),(3.0, 6.0, 9.0, 7.0, 6.0, 5.0, 7.0, 9.0, 11.0, 8.0));
```

Apache Cassandra - TUPLE

```
CREATE TABLE user (
id UUID PRIMARY KEY,
email text,
name text,
preferences set<text>,
equalizer frozen<tuple<float,float,float,float,float,
float,float,float,float,float>>
);
INSERT IN
VALUES (u
```

id	equalizer
62d4f220-5361-…	(3.0, 6.0, 9.0, 7.0, 6.0, 5.0, 7.0, 9.0, 11.0, 8.0)

Apache Cassandra - TUPLE

id	equalizer
62d4f220-5361-…	(3.0, 6.0, 9.0, 7.0, 6.0, 5.0, 7.0, 9.0, 11.0, 8.0)

UPDATE user SET equalizer =

(4.0, 1.6, -1.8, -5.6, -0.7, 0.9, 2.9, 4.3, 4.3, 4.3)

WHERE id = 6ed4f220-5361-11e4-8d89-c971d060d947;

id	equalizer
62d4f220-5361-…	(4.0, 1.6, -1.8, -5.6, -0.7, 0.9, 2.9, 4.3, 4.3, 4.3)

Apache Cassandra - TUPLE

id	equalizer
62d4f220-5361-...	(4.0, 1.6, -1.8, -5.6, -0.7, 0.9, 2.9, 4.3, 4.3, 4.3)

DELETE equalizer from user

WHERE id = 6ed4f220-5361-11e4-8d89-c971d060d947

id	equalizer
62d4f220-5361-...	

Apache Cassandra - BATCH

- BATCH statement combines multiple INSERT, UPDATE, and DELETE statements into a single logical operation
  - Saves on client-server and coordinator-replica communication
  - Atomic operation
    - If any statement in the batch succeeds, all will
  - No batch isolation
    - Other “transactions” can read and write data being affected by partially executed batch

Apache Cassandra - BATCH

```
BEGIN BATCH
    DELETE FROM albums_by_performer
    WHERE performer = 'The Beatles' AND year = 1966 AND title = 'Revolver';
    INSERT INTO albums_by_performer (performer, year, title, genre)
    VALUES ('The Beatles', 1966, 'Revolver', 'Rock');
APPLY BATCH;
```

Apache Cassandra - LIGHTWEIGHT

- Lightweight transactions in batch
  - Batch will execute only if conditions for all lightweight transactions are met
  - All operations in batch will execute serially with the increased performance overhead

Apache Cassandra - LIGHTWEIGHT

```
BEGIN BATCH
    UPDATE user SET lock = true IF lock = false;
    WHERE performer = 'The Beatles' AND year = 1966 AND title = 'Revolver';
    INSERT INTO albums_by_performer (performer, year, title, genre)
    VALUES ('The Beatles', 1966, 'Revolver', 'Rock');
    UPDATE user SET lock = false;
APPLY BATCH;
```

Apache Cassandra - SELECT

- Retrieves rows from a table that satisfy an optional condition
  - SELECT – Which columns to retrieve?
  - FROM – Which table to retrieve from?
  - WHERE – What condition must rows satisfy?
  - ORDER BY – How to sort a result set?
  - LIMIT – How many rows to return?
  - ALLOW FILTERING – Is scanning over all partitions allowed?

Apache Cassandra - SELECT

SELECT select_expression

FROM keyspace_name.table_name

WHERE relation AND relation ...

ORDER BY (clustering_column (ASC | DESC)...)

LIMIT n

ALLOW FILTERING

Apache Cassandra - SELECT

- To retrieve all rows
  - SELECT * FROM album;
- To retrieve specific columns of all rows
  - SELECT performer, title, year FROM album;
- To retrieve a specific field from a user-defined type column
  - SELECT performer.lastname FROM album;
- To compute the number of rows in a table
  - SELECT COUNT(*) FROM album;

Apache Cassandra - SELECT

- Equality search – one partition
  - To retrieve one partition, values for all partition key columns must be specified
  - In a single-row partition, row = partition

```
CREATE TABLE tracks_by_album ( …
PRIMARY KEY ((album_title, year), number));

SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966;
```

Apache Cassandra - SELECT

- Equality search – one partition
  - To retrieve one partition, values for all partition key columns must be specified
  - In a si

album_title	year	number	track_title
Revolver	1966	1	Taxman
Revolver	1966	2	Eleanor Rigby
…	…	…	…
Revolver	1966	14	Tomorrow Never Knows

CREATE

PRIMAR

SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE **album_title = 'Revolver' AND year = 1966**;

Apache Cassandra - SELECT

- Equality search – one row
  - To retrieve one row, values for all primary key columns must be specified
  - In a single-row partition, primary key = partition key

```
CREATE TABLE tracks_by_album ( …
PRIMARY KEY ((album_title, year), number));

SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966 AND
number = 6;
```

Apache Cassandra - SELECT

- Equality search – one row
  - To retrieve one row, values for all primary key columns must be specified
  - In a single-row partition, primary key = partition key

CREATE TABLE tracks_by_album (…
PRIMAR
SELECT

album_title	year	number	track_title
Revolver	1966	6	Yellow Submarine

FROM tracks_by_album
WHERE **album_title = 'Revolver' AND year = 1966 AND number = 6**;

Apache Cassandra - SELECT

- Equality search – subset of rows
  - To retrieve a subset of rows in a partition, values for all partition key columns and all clustering columns must be specified with the last clustering column value being a set
  - IN is only allowed on the **last clustering column** of a primary key

```
CREATE TABLE tracks_by_album ( …
PRIMARY KEY ((album_title, year), number));

SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966 AND
number IN (2,6,7,14);
```

Apache Cassandra - SELECT

- Equality search – subset of rows
  - To retrieve a subset of rows in a partition, values for all partition key columns and all cluste… g a set
  - IN is c…

album_title	year	number	track_title
Revolver	1966	2	Eleanor Rigby
Revolver	1966	6	Yellow Submarine
Revolver	1966	7	She Said She Said
Revolver	1966	14	Tomorrow Never Knows

CREATE…

PRIMAR…

```
SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966 AND
number IN (2,6,7,14);
```

Apache Cassandra - SELECT

- Equality search – subset of rows
  - To retrieve a subset of rows in a partition, values for all partition key columns and one or more but not all clustering columns must be specified
  - Clustering columns in a predicate must constitute a **prefix of clustering columns** specified in the primary key definition

```
CREATE TABLE albums_by_performer ( …
PRIMARY KEY (performer, year, title));

SELECT title, year
FROM albums_by_performer
WHERE performer = 'The Beatles' AND year = 1970;
```

Apache Cassandra - SELECT

- Equality search – subset of rows
  - To retrieve a subset of rows in a partition, values for all partition key columns and one or more but not all clustering columns must be specified
  - Clustering columns in a predicate must constitute a **prefix of clustering columns** specified in the

title	year
At The Hollywood Bowl	1970
Let It Be	1970
The Beatles Christmas Album	1970

CREATE

PRIMAR

SELECT title, year

FROM albums_by_performer

WHERE **performer = 'The Beatles' AND year = 1970**;

Apache Cassandra - SELECT

- Equality search – multiple partitions
  - To retrieve multiple partitions, a set of values for a partition key must be specified using IN
  - IN is only allowed on the last column of a partition key

```
CREATE TABLE albums_by_performer ( …
PRIMARY KEY (performer, year, title));
SELECT performer, title, year
FROM albums_by_performer
WHERE performer IN ('The Beatles', 'Deep Purple');
```

Apache Cassandra - SELECT

- Equality search – multiple partitions
  - To ret ed using IN
  - IN is c

performer	title	year
The Beatles	Let It Be...Naked	2003
…	…	…
The Beatles	With The Beatles	1963
Deep Purple	Abandon	1998
…	…	…

CREATE
PRIMAR
SELECT
FROM albums_by_performer
WHERE **performer IN ('The Beatles', 'Deep Purple')**;

Apache Cassandra - SELECT

- Range search
  - >, >=, <, <=
  - Can only a range search on a partition key using the token() function
  - Results are not meaningful for RandomPartitioner and Murmur3Partitioner
  - Allowed on only one clustering column in a predicate
  - This column should be defined later in the PRIMARY KEY clause than any other clustering column used in a predicate

```
WHERE token(key) >= token(?) AND token(key) < token(?)
```

Apache Cassandra - SELECT

CREATE TABLE tracks_by_album (…

PRIMARY KEY ((album_title, year), **number**));

SELECT album_title, year, number, track_title

FROM tracks_by_album

WHERE album_title = 'Revolver' AND year = 1966 AND **number >= 6 AND number < 8**;

Apache Cassandra - SELECT

```
CREATE TABLE tracks_by_album ( …
PRIMARY KEY ((album_title, year), number));


SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966 AND number >= 6 AND number < 8;
```

album_title	year	number	track_title
Revolver	1966	6	Yellow Submarine
Revolver	1966	7	She Said She Said

Apache Cassandra – ALLOW FILTERING

- Allows scanning over all partitions
  - Predicate does not specify values for partition key columns
  - Relaxes the requirement that a partition key must be specified
  - Potentially expensive queries that may return large results
  - Use with caution
  - LIMIT clause is recommended
  - Predicate can have equality or inequality relations on clustering columns
- Return 7th tracks for the first 10 albums in the table
- Return the number of albums with 30 or more tracks

Apache Cassandra

- Allows scanning over all partitions
  - Predicate does not specify values for partition key columns
  - Relaxes the requirement that a partition key must be specified
  - Potentially expensive queries that may return large results
  - Use with caution
  - LIMIT clause is recommended
  - Predicate can have equality or inequality relations on clustering columns
- **Return 7th tracks for the first 10 albums in the table**

  SELECT * FROM tracks_by_album
  WHERE number = 7 LIMIT 10 ALLOW FILTERING;
- Return the number of albums with 30 or more tracks

Apache Cassandra

- Allows scanning over all partitions
  - Predicate does not specify values for partition key columns
  - Relaxes the requirement that a partition key must be specified
  - Potentially expensive queries that may return large results
  - Use with caution
  - LIMIT clause is recommended
  - Predicate can have equality or inequality relations on clustering columns
- Return 7th tracks for the first 10 albums in the table
- **Return the number of albums with 30 or more tracks**

```
SELECT COUNT(*) FROM tracks_by_album
WHERE number = 30 LIMIT 100000 ALLOW FILTERING;
```

Apache Cassandra – Indexes in Queries

- A predicate may involve only an indexed column
- A predicate may involve primary key and indexed columns
  - Useful to narrow a search in a large multi-row partition
- A predicate may involve multiple indexed columns
  - ALLOW FILTERING must be used

Apache Cassandra – Indexes in Queries

- **A predicate may involve only an indexed column**

  CREATE INDEX performer_country_key ON performer (country);

  SELECT name FROM performer WHERE country = 'Iceland';

- A predicate may involve primary key and indexed columns
  - Useful to narrow a search in a large multi-row partition
- A predicate may involve multiple indexed columns
  - ALLOW FILTERING must be used

Apache Cassandra – Indexes in Queries

- A predicate may involve only an indexed column
- A predicate may involve primary key and indexed columns
  - Useful to narrow a search in a large multi-row partition
- **A predicate may involve multiple indexed columns**
  - ALLOW FILTERING must be used

```
CREATE INDEX performer_country_key ON performer (country);
CREATE INDEX performer_style_key ON performer (style);
SELECT name FROM performer
WHERE country = 'Iceland' AND style = 'Rock' ALLOW FILTERING;
```

Apache Cassandra – SELECT in Collections

- Searches on indexed collections uses the CONTAINS keyword
- Set, List, Map – Search for a value
- Map – Search for a key

Apache Cassandra

- Searches on indexed collections uses the CONTAINS keyword
- **Set, List, Map – Search for a value**

```
CREATE INDEX ON user (preferences);

SELECT id FROM user
WHERE preferences CONTAINS 'Rock';
```

- Map – Search for a key

Apache Cassandra – SELECT in Collections

- Searches on indexed collections uses the CONTAINS keyword
- Set, List, Map – Search for a value
- **Map – Search for a key**

```
CREATE INDEX ON album (tracks);

SELECT title, tracks FROM album
WHERE tracks CONTAINS KEY 20;
```

Apache Cassandra – SELECT in Collections

- The column is treated as a blob and must search on all fields
  - User-defined type – Search all fields
  - Tuple – Search all fields

Apache Cassandra

- The column is treated as a blob and must search on all fields
  - **User-defined type – Search all fields**

```
CREATE INDEX ON track_ratings_by_user (song);

SELECT * FROM track_ratings_by_user
WHERE song = {album_title: 'Beatles For Sale',
album_year: 1964,
track_title: 'Cant Buy Me Love'};
```

  - Tuple – Search all fields

Apache Cassandra – SELECT in Collections

- The column is treated as a blob and must search on all fields
  - User-defined type – Search all fields
  - **Tuple – Search all fields**

```
CREATE INDEX ON user (equalizer);

SELECT * FROM user
WHERE equalizer = (1.0, 2.0, 3.0, 4.0, 5.0,
6.0, 7.0, 8.0, 9.0, 10.0);
```

Apache Cassandra – ORDER BY

- ORDER BY specifies how query results must be sorted
  - Allowed only on clustering columns
  - Default order is ASC or as defined by WITH CLUSTERING ORDER
  - Default order can be reversed for all clustering columns at once

Apache Cassandra – ORDER BY

```
CREATE TABLE tracks_by_album ( …
PRIMARY KEY ((album_title, year), number));

SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966
ORDER BY number DESC;
```

Apache C

CREATE TA

PRIMARY

album_title	year	number	track_title
Revolver	1966	14	Tomorrow Never Knows
Revolver	1966	13	Got to Get You Into My Life
…	…	…	…
Revolver	1966	1	Taxman

```
SELECT album_title, year, number, track_title
FROM tracks_by_album
WHERE album_title = 'Revolver' AND year = 1966
ORDER BY number DESC;
```

Apache Cassandra - Functions

- TIMEUUID functions
  - dateOf() – extracts the timestamp as a date of a timeuuid column
  - now() – generates a new unique timeuuid
  - minTimeuuid() and maxTimeuuid() – return a UUID-like result given a conditional time component as an argument
  - unixTimestampOf() – extracts the “raw” timestamp of a timeuuid column as a 64-bit integer

Apache Cassandra - Functions

- TIMEUUID functions
- **dateOf() – extracts the timestamp as a date of a timeuuid column**
  - SELECT dateOf(timeuuid_column), … FROM …;
- now() – generates a new unique timeuuid
- minTimeuuid() and maxTimeuuid() – return a UUID-like result given a conditional time component as an argument
- unixTimestampOf() – extracts the “raw” timestamp of a timeuuid column as a 64-bit integer

Apache Cassandra - Functions

- TIMEUUID functions
- dateOf() – extracts the timestamp as a date of a timeuuid column
- **now() – generates a new unique timeuuid**
  - INSERT INTO … (timeuuid_column, …) VALUES (now(), …);
- minTimeuuid() and maxTimeuuid() – return a UUID-like result given a conditional time component as an argument
- unixTimestampOf() – extracts the “raw” timestamp of a timeuuid column as a 64-bit integer

Apache Cassandra - Functions

- TIMEUUID functions
- dateOf() – extracts the timestamp as a date of a timeuuid column
- now() – generates a new unique timeuuid
- **minTimeuuid() and maxTimeuuid() – return a UUID-like result given a conditional time component as an argument**
  - SELECT * FROM … WHERE … AND timeuuid_column > maxTimeuuid('2014-01-01 00:00+0000') AND timeuuid_column < minTimeuuid('2014-03-01 00:00+0000');
- unixTimestampOf() – extracts the “raw” timestamp of a timeuuid column as a 64-bit integer

Apache Cassandra - Functions

- TIMEUUID functions
- dateOf() – extracts the timestamp as a date of a timeuuid column
- now() – generates a new unique timeuuid
- minTimeuuid() and maxTimeuuid() – return a UUID-like result given a conditional time component as an argument
- **unixTimestampOf() – extracts the "raw" timestamp of a timeuuid column as a 64-bit integer**
  - SELECT unixTimestampOf(timeuuid_column), … FROM …;

Apache Cassandra - BLOB

- Blob conversion functions
  - Series of typeAsBlob() and blobAsType() functions
  - The Cassandra blob data type represents a constant hexadecimal number defined as 0[xX](hex)+ where hex is a hexadecimal character, such as [0-9a-fA-F].
  - For example, 0xcafe.
  - The maximum theoretical size for a blob is 2 GB. The practical limit on blob size, however, is less than 1 MB. A blob type is suitable for storing a small image or short string.
- Token access function
  - token() function

Apache Cassandra - BLOB

- Blob conversion functions
  - **Series of typeAsBlob() and blobAsType() functions**

```
SELECT varcharAsBlob(varchar_column), … FROM …;
SELECT blobAsBigint(blob_column), … FROM …;

CREATE TABLE bios ( user_name varchar PRIMARY KEY,
   bio blob
 );
INSERT INTO bios (user_name, bio) VALUES ('fred', bigintAsBlob(3));
```

- Token access function
  - token() function

Apache Cassandra - BLOB

- Blob conversion functions
  - Series of typeAsBlob() and blobAsType() functions
- Token access function
  - **token() function**

SELECT * FROM … WHERE token(partition_key) > token(2014);