Music Service

The example of a music service shows the how to use compound keys, clustering columns, and collections to model Cassandra data.

```
CREATE KEYSPACE IF NOT EXISTS admatic WITH replication = {'class': 'NetworkTopologyS
trategy', 'dc1': '2', 'dc2': '2'};
USE admatic ;

CREATE TABLE songs (
  id uuid PRIMARY KEY,
  title text,
  album text,
  artist text,
  data blob
 );
```

In a relational database, you would create a playlists table with a foreign key to the songs, but in Apache Cassandra™, you denormalize the data because joins are not performant in a distributed system. Later, this document covers how to use a collection to accomplish the same goal as joining the tables to tag songs.

```
CREATE TABLE playlists (
  id uuid,
  song_order int,
  song_id uuid,
  title text,
  album text,
  artist text,
  PRIMARY KEY  (id, song_order ) );

INSERT INTO playlists (id, song_order, song_id, title, artist, album)
  VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 1,
  a3e64f8f-bd44-4f28-b8d9-6938726e34d4, 'La Grange', 'ZZ Top', 'Tres Hombres');

INSERT INTO playlists (id, song_order, song_id, title, artist, album)
  VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 2,
  8a172618-b121-4136-bb10-f665cfc469eb, 'Moving in Stereo', 'Fu Manchu', 'We Must Ob
ey');

INSERT INTO playlists (id, song_order, song_id, title, artist, album)
  VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 3,
  2b09185b-fb5a-4734-9b56-49077de9edbf, 'Outside Woman Blues', 'Back Door Slam', 'Ro
ll Away');

SELECT * FROM playlists;

   id                                   | song_order | album          | artist
```

```
 | song_id                              | title
  --------------------------------------+------------+-------------+---------------
-+--------------------------------------+---------------------
   62c36092-82a1-3a00-93d1-46196ee77204 |          1 | Tres Hombres |         ZZ Top
 | a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |          La Grange
   62c36092-82a1-3a00-93d1-46196ee77204 |          2 | We Must Obey |      Fu Manchu
 | 8a172618-b121-4136-bb10-f665cfc469eb |     Moving in Stereo
   62c36092-82a1-3a00-93d1-46196ee77204 |          3 |    Roll Away | Back Door Slam
 | 2b09185b-fb5a-4734-9b56-49077de9edbf | Outside Woman Blues

  (3 rows)
```

```
SELECT album, title FROM playlists WHERE artist = 'Fu Manchu';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"
```

Cassandra will reject this query because the query requires a sequential scan across the entire playlists dataset, because artist is not a partition key or clustering column. By creating an index on artist, Cassandra can now pull out the records.

```
CREATE INDEX ON playlists( artist );
```

Now, you can query the playlists for songs by Fu Manchu. The output looks like this:

```
SELECT album, title FROM playlists WHERE artist = 'Fu Manchu';

 album        | title
--------------+------------------
 We Must Obey | Moving in Stereo

(1 rows)
```

Compound keys and clustering

A compound primary key consists of the partition key and one or more additional columns that determine clustering. The partition key determines which node stores the data. It is responsible for data distribution across the nodes. The additional columns determine per-partition clustering. Clustering is a storage engine process that sorts data within the partition.

```
CREATE TABLE playlists (
  id uuid,
  song_order int,
  song_id uuid,
  title text,
  album text,
  artist text,
  PRIMARY KEY  (id, song_order ) );
```

On a physical node, when rows for a partition key are stored in order based on the clustering columns, retrieval of rows is very efficient. For example, because the id in the playlists table is the partition key, all the songs for a playlist are clustered in the order of the remaining song_order column. The others columns are displayed in alphabetical order by Cassandra.

Insertion, update, and deletion operations on rows sharing the same partition key for a table are performed atomically and in isolation.

You can query a single sequential set of data on disk to get the songs for a playlist.

```
SELECT * FROM playlists WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204
  ORDER BY song_order DESC LIMIT 50;

 id                                   | song_order | album         | artist          |
 song_id                              | title
--------------------------------------+------------+---------------+-----------------+
--------------------------------------+---------------------
 62c36092-82a1-3a00-93d1-46196ee77204 |          3 |     Roll Away | Back Door Slam |
 2b09185b-fb5a-4734-9b56-49077de9edbf | Outside Woman Blues
 62c36092-82a1-3a00-93d1-46196ee77204 |          2 | We Must Obey |      Fu Manchu |
 8a172618-b121-4136-bb10-f665cfc469eb |    Moving in Stereo
 62c36092-82a1-3a00-93d1-46196ee77204 |          1 | Tres Hombres |          ZZ Top |
 a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |           La Grange

(3 rows)
```

Cassandra stores an entire row of data on a node by partition key. If you have too much data in a partition and want to spread the data over multiple nodes, use a composite partition key.

Collection columns

CQL contains these collection types:

- set
- list
- map

In a relational database, to allow users to have multiple email addresses, you create an email_addresses table having a many-to-one (joined) relationship to a users table. CQL handles the classic multiple email addresses use case, and other use cases, by defining columns as collections. Using the set collection type to solve the multiple email addresses problem is convenient and intuitive.

Adding a collection to a table

The music service example includes the capability to tag the songs. From a relational standpoint, you can think of storage engine rows as partitions, within which (object) rows are clustered. To tag songs, use a collection set. Declare the collection set using the CREATE TABLE or ALTER TABLE statements. Because the playlists table already exists from the earlier example, just alter that table to add a collection set, tags:

```
ALTER TABLE playlists ADD tags set<text>;
```

Updating a collection

```
SELECT * FROM playlists;

 id                                   | song_order | album          | artist          |
 song_id                              | tags | title
--------------------------------------+------------+--------------+----------------+
--------------------------------------+------+--------------------
 62c36092-82a1-3a00-93d1-46196ee77204 |          1 | Tres Hombres |          ZZ Top |
 a3e64f8f-bd44-4f28-b8d9-6938726e34d4 | null |           La Grange
 62c36092-82a1-3a00-93d1-46196ee77204 |          2 | We Must Obey |       Fu Manchu |
 8a172618-b121-4136-bb10-f665cfc469eb | null |     Moving in Stereo
 62c36092-82a1-3a00-93d1-46196ee77204 |          3 |     Roll Away | Back Door Slam |
 2b09185b-fb5a-4734-9b56-49077de9edbf | null | Outside Woman Blues

(3 rows)

UPDATE playlists SET tags = tags + {'2007'}
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_order = 2;
UPDATE playlists SET tags = tags + {'covers'}
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_order = 2;
UPDATE playlists SET tags = tags + {'1973'}
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_order = 1;
UPDATE playlists SET tags = tags + {'blues'}
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_order = 1;
UPDATE playlists SET tags = tags + {'rock'}
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_order = 4;

SELECT * FROM playlists;

 id                                   | song_order | album          | artist          |
 song_id                              | tags                | title
--------------------------------------+------------+--------------+----------------+
--------------------------------------+--------------------+----------------------
 62c36092-82a1-3a00-93d1-46196ee77204 |          1 | Tres Hombres |          ZZ Top |
 a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |  {'1973', 'blues'} |           La Grange
 62c36092-82a1-3a00-93d1-46196ee77204 |          2 | We Must Obey |       Fu Manchu |
 8a172618-b121-4136-bb10-f665cfc469eb | {'2007', 'covers'} |     Moving in Stereo
 62c36092-82a1-3a00-93d1-46196ee77204 |          3 |     Roll Away | Back Door Slam |
 2b09185b-fb5a-4734-9b56-49077de9edbf |               null | Outside Woman Blues
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |         null |            null |
                                  null |             {'rock'} |               null

(4 rows)
```

A music reviews list and a schedule (map collection) of live appearances can be added to the table:

```
ALTER TABLE playlists ADD reviews list<text>;
ALTER TABLE playlists ADD venue map<timestamp, text>;

SELECT * FROM playlists;

 id                                   | song_order | album         | artist          |
 reviews | song_id                              | tags                 | title
       | venue
--------------------------------------+------------+---------------+----------------+
---------+--------------------------------------+---------------------+--------------
-------+-------
 62c36092-82a1-3a00-93d1-46196ee77204 |          1 | Tres Hombres |          ZZ Top |
    null | a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |  {'1973', 'blues'} |            La
Grange |  null
 62c36092-82a1-3a00-93d1-46196ee77204 |          2 | We Must Obey |      Fu Manchu |
    null | 8a172618-b121-4136-bb10-f665cfc469eb | {'2007', 'covers'} |     Moving in
Stereo |  null
 62c36092-82a1-3a00-93d1-46196ee77204 |          3 |    Roll Away | Back Door Slam |
    null | 2b09185b-fb5a-4734-9b56-49077de9edbf |                null | Outside Woman
 Blues |  null
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |         null |            null |
    null |                                 null |           {'rock'} |
  null |  null

(4 rows)
```

Each element of a set, list, or map is internally stored as one Cassandra column. To update a set, use the UPDATE command and the addition (+) operator to add an element or the subtraction (-) operator to remove an element. For example, to update a set:

```
SELECT * FROM playlists WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_ord
er = 4;

 id                                   | song_order | album | artist | reviews | song
_id | tags       | title | venue
--------------------------------------+------------+-------+---------+----------+-----
----+----------+-------+-------
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |  null |    null |     null |    n
ull | {'rock'} |  null |  null

(1 rows)

UPDATE playlists
  SET tags = tags + {'punk rock'}
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_order = 4;
```

```
SELECT * FROM playlists WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 AND song_ord
er = 4;

 id                                   | song_order | album | artist | reviews | song
_id | tags                   | title | venue
--------------------------------------+------------+-------+--------+---------+-----
----+------------------------+-------+-------
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |  null |   null |    null |    n
ull | {'punk rock', 'rock'} |  null |  null

(1 rows)
```

To update a list, a similar syntax using square brackets instead of curly brackets is used.

```
SELECT * FROM playlists WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 and song_ord
er = 4;

 id                                   | song_order | album | artist | reviews | song
_id | tags                   | title | venue
--------------------------------------+------------+-------+--------+---------+-----
----+------------------------+-------+-------
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |  null |   null |    null |    n
ull | {'punk rock', 'rock'} |  null |  null

(1 rows)

UPDATE playlists
  SET reviews = reviews + [ 'best lyrics' ]
  WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 and song_order = 4;

SELECT * FROM playlists WHERE id = 62c36092-82a1-3a00-93d1-46196ee77204 and song_ord
er = 4;

 id                                   | song_order | album | artist | reviews
  | song_id | tags                   | title | venue
--------------------------------------+------------+-------+--------+---------------
--+---------+------------------------+-------+-------
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |  null |   null | ['best lyrics'
] |    null | {'punk rock', 'rock'} |  null |  null

(1 rows)
```

To update a map, use INSERT to specify the data in a map collection.

```
SELECT * FROM playlists;

 id                                   | song_order | album          | artist          |
 reviews          | song_id                              | tags                    | ti
tle                 | venue
--------------------------------------+------------+----------------+-----------------+
------------------+---------------------------------------+-----------------------+---
-------------------+--------
 62c36092-82a1-3a00-93d1-46196ee77204 |           1 | Tres Hombres |         ZZ Top |
           null | a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |     {'1973', 'blues'} |
        La Grange |  null
 62c36092-82a1-3a00-93d1-46196ee77204 |           2 | We Must Obey |      Fu Manchu |
           null | 8a172618-b121-4136-bb10-f665cfc469eb |    {'2007', 'covers'} |
 Moving in Stereo |  null
 62c36092-82a1-3a00-93d1-46196ee77204 |           3 |    Roll Away | Back Door Slam |
           null | 2b09185b-fb5a-4734-9b56-49077de9edbf |                  null | Ou
tside Woman Blues |  null
 62c36092-82a1-3a00-93d1-46196ee77204 |           4 |         null |           null |
 ['best lyrics'] |                                  null | {'punk rock', 'rock'} |
             null |  null

(4 rows)

INSERT INTO playlists (id, song_order, venue)
  VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 4,
  { '2013-9-22 22:00'  : 'The Fillmore',
  '2013-10-1 21:00' : 'The Apple Barrel'});

INSERT INTO playlists (id, song_order, venue)
  VALUES (62c36092-82a1-3a00-93d1-46196ee77204, 3,
  { '2014-1-22 22:00'  : 'Cactus Cafe',
  '2014-01-12 20:00' : 'Mohawk'});

SELECT * FROM playlists;

 id                                   | song_order | album        | artist         |
 reviews         | song_id                              | tags                  | ti
tle                | venue
--------------------------------------+------------+--------------+----------------+
------------------+---------------------------------------+-----------------------+---
-------------------+--------------------------------------------------------------------
-------------------------------------------
 62c36092-82a1-3a00-93d1-46196ee77204 |           1 | Tres Hombres |         ZZ Top |
           null | a3e64f8f-bd44-4f28-b8d9-6938726e34d4 |     {'1973', 'blues'} |
        La Grange |
                                      null
 62c36092-82a1-3a00-93d1-46196ee77204 |           2 | We Must Obey |      Fu Manchu |
           null | 8a172618-b121-4136-bb10-f665cfc469eb |    {'2007', 'covers'} |
 Moving in Stereo |
                                      null
 62c36092-82a1-3a00-93d1-46196ee77204 |           3 |    Roll Away | Back Door Slam |
           null | 2b09185b-fb5a-4734-9b56-49077de9edbf |                  null | Ou
tside Woman Blues |            {'2014-01-12 20:00:00.000000+0000': 'Mohawk', '2014-0
1-22 22:00:00.000000+0000': 'Cactus Cafe'}
 62c36092-82a1-3a00-93d1-46196ee77204 |           4 |         null |           null |
 ['best lyrics'] |                                  null | {'punk rock', 'rock'} |
             null | {'2013-09-22 22:00:00.000000+0000': 'The Fillmore', '2013-10-01
21:00:00.000000+0000': 'The Apple Barrel'}

(4 rows)
```

Inserting data into the map replaces the entire map.

Indexing a collection

In Apache Cassandra™ 2.1 and later, you can index collections and query the database to find a collection containing a particular value.

Continuing with the music service example, suppose you want to find songs tagged blues and that debuted at the Fillmore. Index the tags set and venue map. Query for values in the tags set and the venue map, as shown in the next section.

```
SELECT * FROM playlists WHERE tags CONTAINS 'blues' AND venue CONTAINS 'The Fillmore
';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"

CREATE INDEX ON playlists (tags);
CREATE INDEX mymapvalues ON playlists (venue);

SELECT * FROM playlists WHERE tags CONTAINS 'blues' AND venue CONTAINS 'The Fillmore
';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"

SELECT * FROM playlists WHERE tags CONTAINS 'blues';

 id                                   | song_order | album        | artist | reviews
 | song_id                              | tags               | title      | venue
--------------------------------------+------------+--------------+--------+--------
-+--------------------------------------+--------------------+------------+-------
 62c36092-82a1-3a00-93d1-46196ee77204 |          1 | Tres Hombres | ZZ Top |    null
 | a3e64f8f-bd44-4f28-b8d9-6938726e34d4 | {'1973', 'blues'} | La Grange |  null

(1 rows)

SELECT * FROM playlists WHERE venue CONTAINS 'The Fillmore';

 id                                   | song_order | album | artist | reviews
  | song_id | tags                   | title | venue
--------------------------------------+------------+-------+--------+----------------
--+---------+------------------------+-------+-------------------------------------
---------------------------------------------------------------------
 62c36092-82a1-3a00-93d1-46196ee77204 |          4 |  null |   null | ['best lyrics'
] |    null | {'punk rock', 'rock'} |  null | {'2013-09-22 22:00:00.000000+0000': 'T
he Fillmore', '2013-10-01 21:00:00.000000+0000': 'The Apple Barrel'}
```

```
(1 rows)
```

When to use a collection

Use collections when you want to store or denormalize a small amount of data. Values of items in collections are limited to 64K. Other limitations also apply. Collections work well for storing data such as the phone numbers of a user and labels applied to an email. If the data you need to store has unbounded growth potential, such as all the messages sent by a user or events registered by a sensor, do not use collections. Instead, use a table having a compound primary key and store data in the clustering columns.