Complex Data Types

Paul O’Fallon

@paulofallon

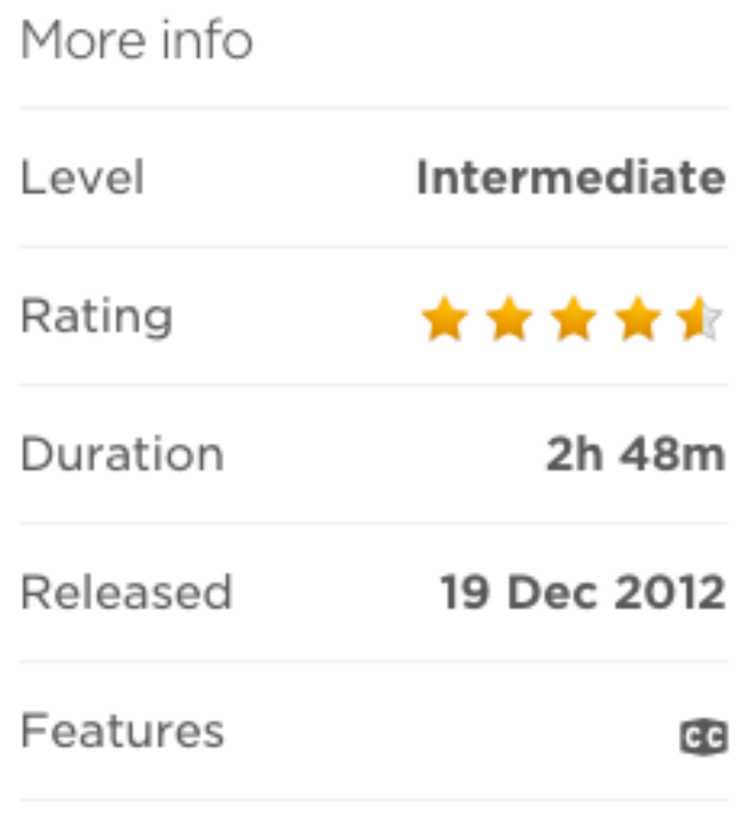
Complex Data Types

Collections

Tuples

User Defined Types

Collections: Set



More info

Level	**Intermediate**
Rating	★★★★½
Duration	**2h 48m**
Released	**19 Dec 2012**
Features	CC

```
CREATE TABLE courses (
    id varchar,
    name static,
    // ...
    features set<varchar> static,
    module_id int,
    // ...
    PRIMARY KEY (id, module_id)
);
```

Collections: Set

Inserting with a set

```
INSERT INTO courses (id, name, features)
VALUES ('node-intro','Introduction to Node.js',{'cc'});
```

Adding to a set

```
UPDATE courses SET features = features + {'cc'}
WHERE course_id = 'node-intro';
```

Collections: Set

Removing from a set

```
UPDATE courses SET features = features - {'cc'}
WHERE course_id = 'node-intro';
```

Emptying the entire set

```
UPDATE courses SET features = {}
WHERE course_id = 'node-intro';
```

Collections: List

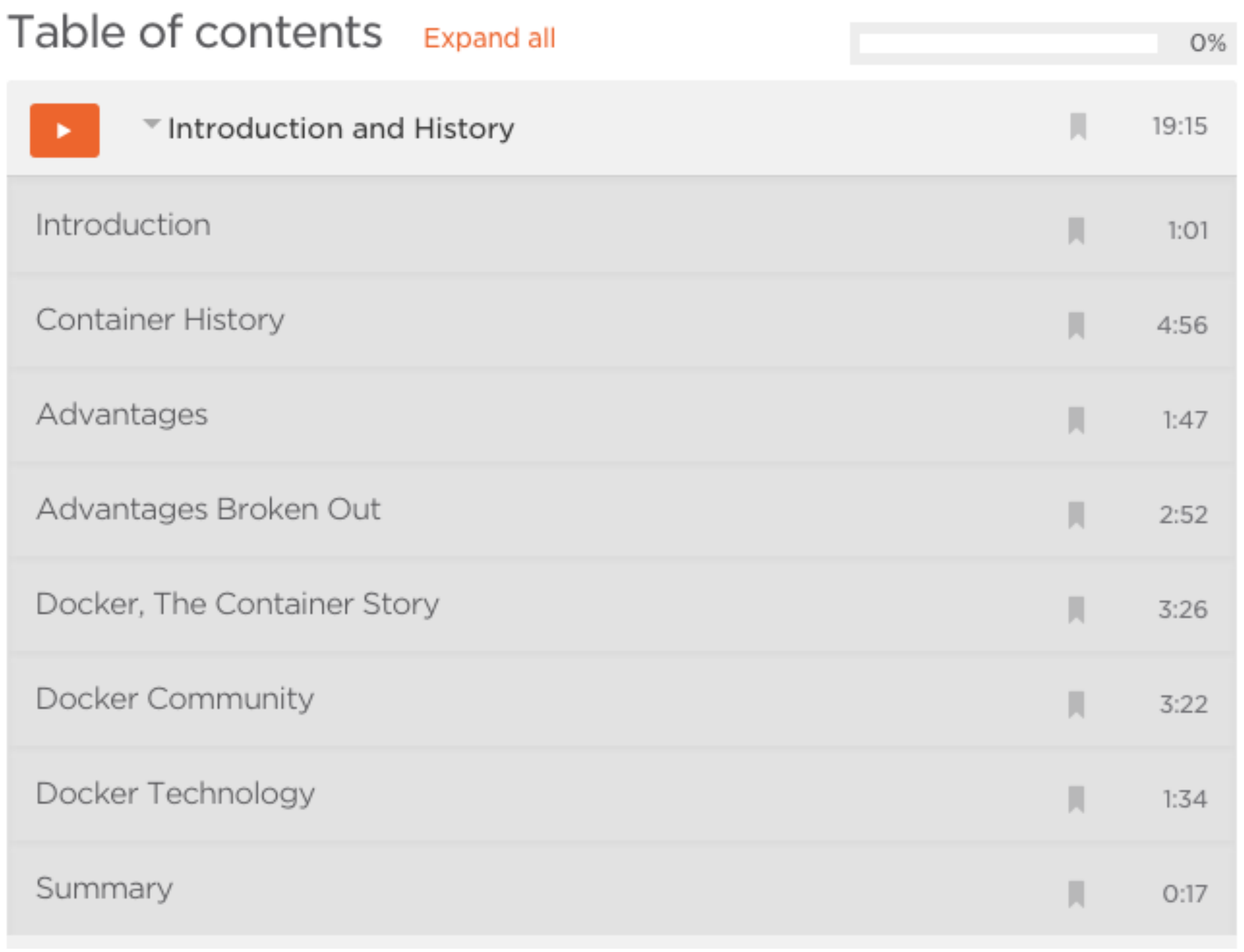


Table of contents

Introduction and History	19:15
Introduction	1:01
Container History	4:56
Advantages	1:47
Advantages Broken Out	2:52
Docker, The Container Story	3:26
Docker Community	3:22
Docker Technology	1:34
Summary	0:17

Collections: List

```
CREATE TABLE courses (
    id varchar,
    name static,
    // ...
    module_id int,
    clips list<varchar>,
    // ...
    PRIMARY KEY (id, module_id)
);
```

Collections: List

Inserting with a list

```
INSERT INTO courses (id, module_id, clips)
VALUES ('docker-fundamentals',1,['Container History']);
```

Adding to a list

```
UPDATE courses SET clips = ['Introduction'] + clips
WHERE course_id = 'docker-fundamentals' AND module_id = 1;

UPDATE courses SET clips = clips + ['Advantages']
WHERE course_id = 'docker-fundamentals' AND module_id = 1;
```

Collections: List

Removing from a list

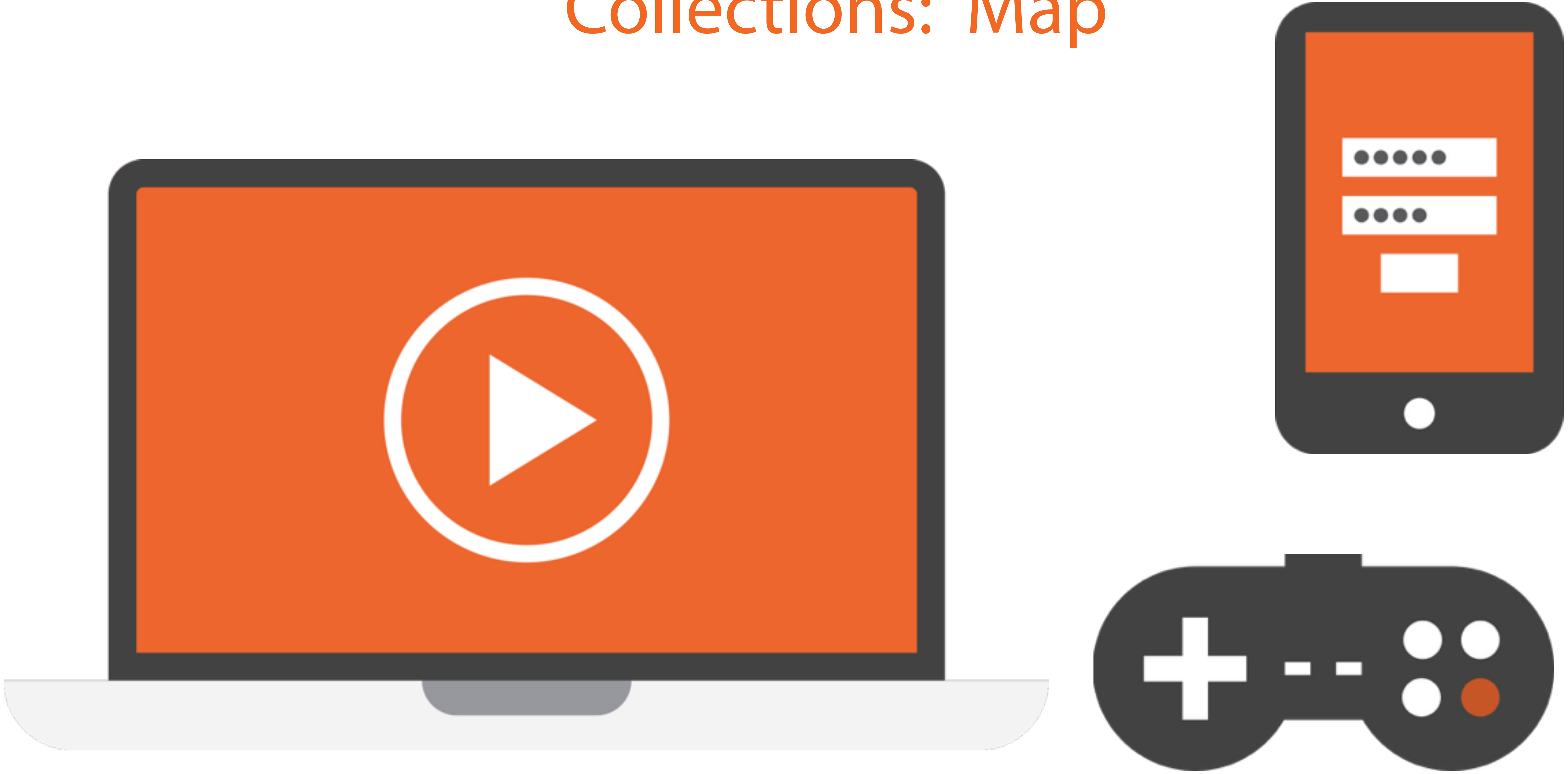
```
UPDATE courses SET clips = clips - ['Introduction']
WHERE course_id = 'docker-fundamentals' and module_id = 1;
```

Manipulating a list by element id

```
UPDATE courses SET clips[3] = 'Advantages Broken Out'
WHERE course_id = 'docker-fundamentals' AND module_id = 1;

DELETE clips[3] FROM courses
WHERE course_id = 'docker-fundamentals' AND module_id = 1;
```

Collections: Map



Collections: Map

```
CREATE TABLE users (
    id varchar,
    first_name varchar,
    last_name varchar,
    password varchar,
    reset_token varchar,
    last_login map<varchar,timestamp>,
    PRIMARY KEY (id)
);
```

Collections: Map

Inserting with a map

```
INSERT INTO users (id, first_name, last_name, last_login)
VALUES ('john-doe', 'John', 'Doe',
    {'383cc0867cd2': '2015-06-30 09:02:24'});
```

Updating / adding to a map

```
UPDATE users SET last_login['383cc0867cd2']
  = '2015-07-01 11:17:42' WHERE user_id = 'john-doe';

UPDATE users SET last_login = last_login + {'7eb0a8997f39':
'2015-07-02 07:32:17'} WHERE user_id = 'john-doe';
```

Collections: Map

Removing from a map

```
DELETE last_login['383cc0867cd2'] FROM users
WHERE id = 'john-doe';

UPDATE users SET last_login = last_login - {'7eb0a8997f39'}
WHERE id = 'john-doe';
```

Emptying the entire map

```
UPDATE users SET last_login = {}
WHERE id = 'john-doe';
```

Collections and TTL

```
UPDATE users USING TTL 31536000
SET last_login['383cc0867cd2'] = '2015-07-01 11:17:42'
WHERE user_id = 'john-doe';
```

Tuples

(varchar, int, int, varchar, timestamp)

Tuples

$$\begin{tikzcd}
\texttt{383cc0867cd2} \arrow[r] & \texttt{2015-07-01 11:17:42} \arrow[d, Rightarrow] \\
\texttt{383cc0867cd2} \arrow[r] & \begin{matrix} \texttt{2015-07-01 11:17:42} \\ + \\ \texttt{98.203.153.64} \end{matrix}
\end{tikzcd}$$

Tuples

```
CREATE TABLE users (
    id varchar,
    first_name varchar,
    last_name varchar,
    password varchar,
    reset_token varchar,
    last_login map<varchar,
      frozen<tuple<timestamp,inet>>>,
    PRIMARY KEY (id)
);
```

“Frozen”

- Nested types are serialized as a single (blob) value
- True for nested collections as well (`list<set<varchar>>`)
- Nested values must be set or read as a whole
- `frozen<>` makes this distinction obvious
- Leaves open the possibility of "non-frozen" support in the future

User Defined Types

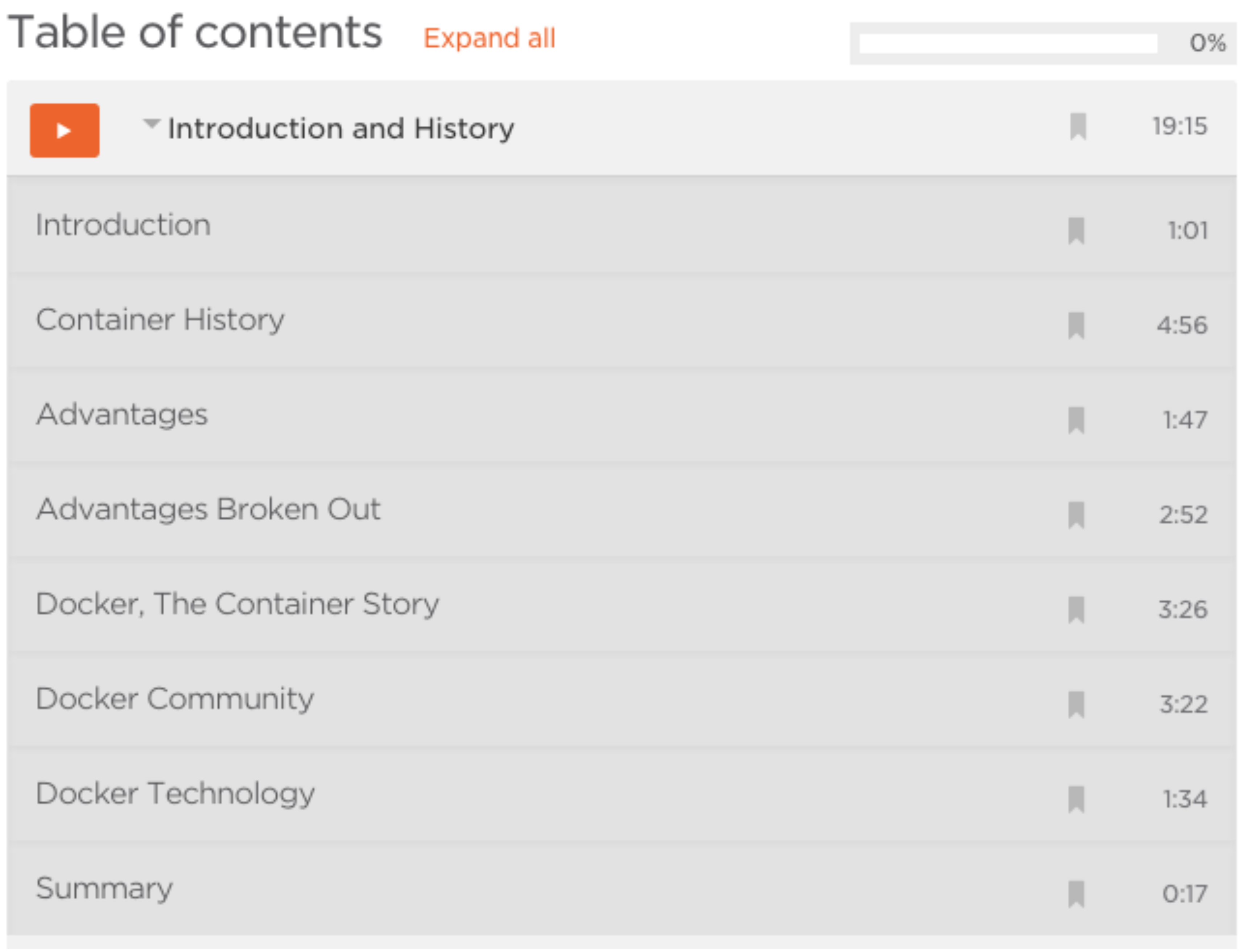


Table of contents

User Defined Types

```
CREATE TYPE clip (name varchar, duration int);

CREATE TABLE courses (
    id varchar,
    author varchar static,
    // ...
    clips list<frozen<clip>>,
    module_id int,
    // ...
    PRIMARY KEY (id, module_id)
);
```

Why Not Just Use a Tuple?

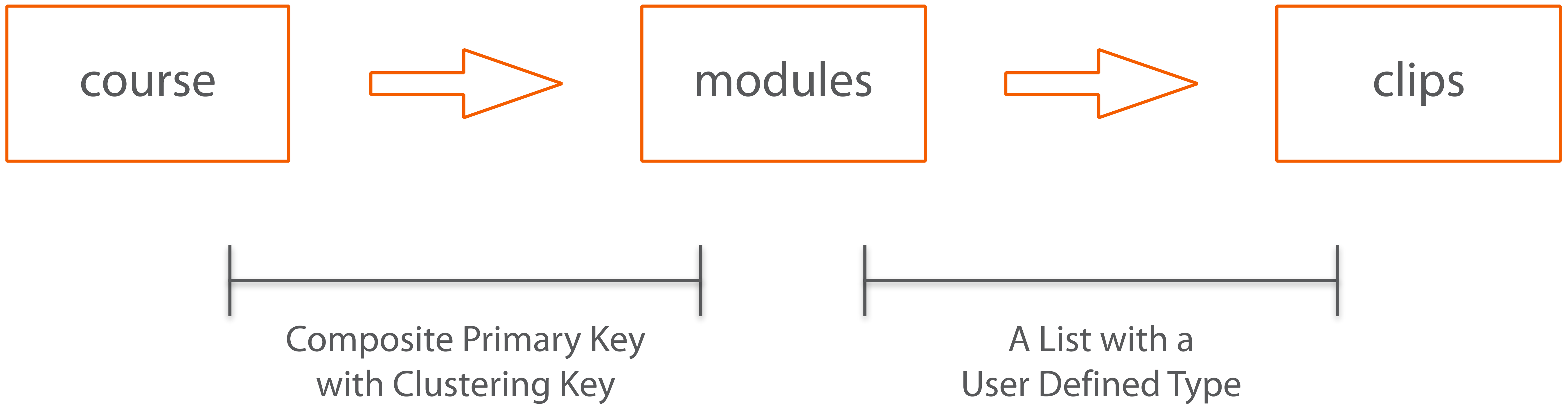
- Can identify individual components by name (not just order)
- Even more helpful with multiple components of the same type
- Helpful to start with a Tuple when modeling a User Defined Type
- Opportunity to reuse User Defined Type across tables

User Defined Types

```
CREATE TYPE person (name varchar, id varchar);

CREATE TABLE courses (
    id varchar,
    author frozen<person> static,
    // ...
    clips list<frozen<clip>>,
    module_id int,
    // ...
    PRIMARY KEY (id, module_id)
);
```

Courses, Modules and Clips


course
modules
clips
Composite Primary Key with Clustering Key
A List with a User Defined Type

All in a single partition!

Conclusion

- Sets, Lists and Maps
- Collections and TTLs
- Tuples
- “Frozen” Nested Complex Types
- User Defined Types