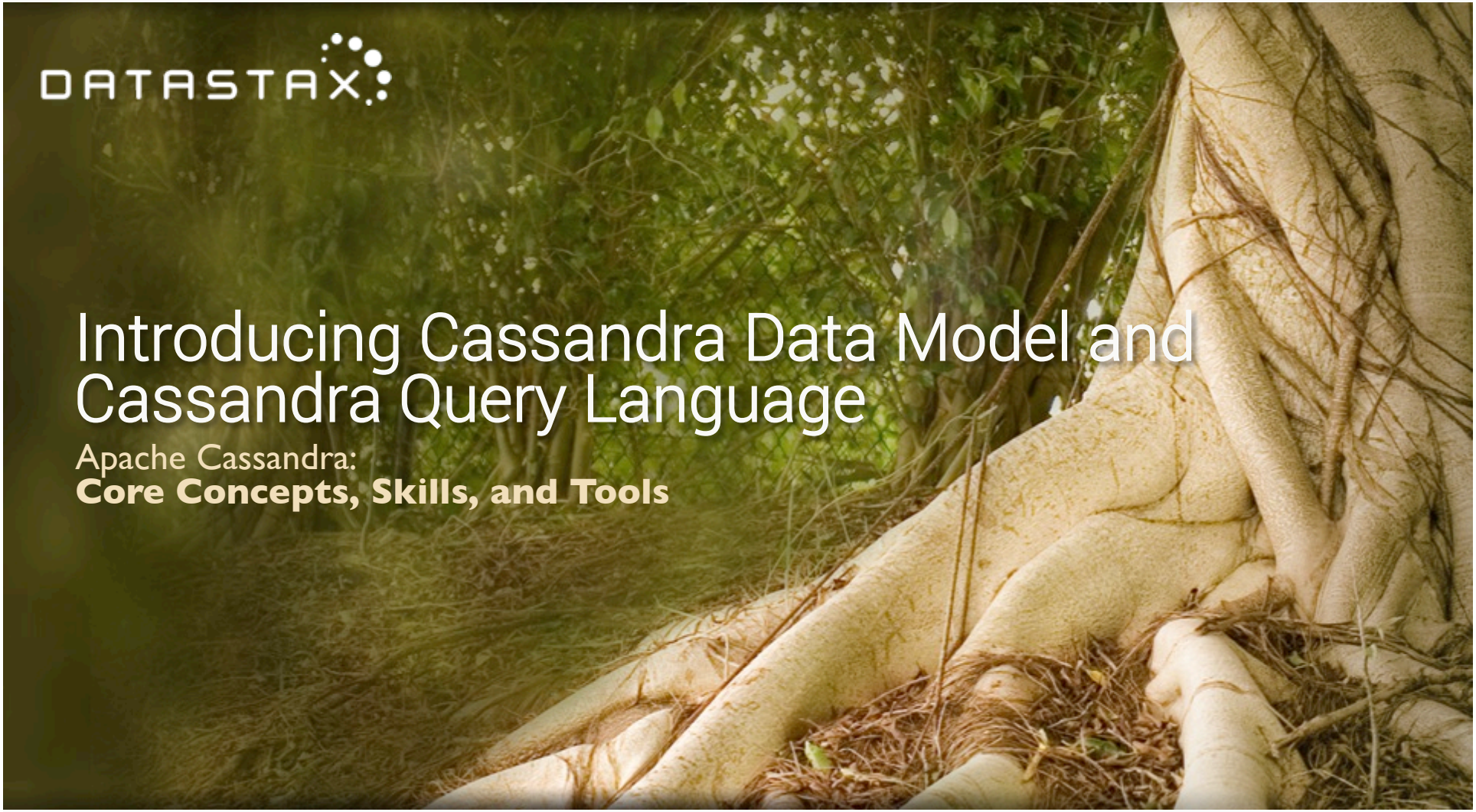


Introducing Cassandra Data Model and Cassandra Query Language

Apache Cassandra:
Core Concepts, Skills, and Tools

Artem Chebotko, Leo Schuman

Exercise 2: Represent column families as tables

Learning Objectives

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

What is *cqlsh* and how do you launch it?

DATASTAX

- Cassandra client with the command-line interface
  - Supports Cassandra Query Language statements
  - Supports *cqlsh* shell commands

- Launching on Linux

```
$ ./cqlsh [options] [host [port]]
```

- Launching on Windows

```
python cqlsh [options] [host [port]]
```

- Examples

```
$ ./cqlsh
```

```
$ ./cqlsh -u student -p cassandra 127.0.0.1 9160
```

What shell commands does *cqlsh* support?

DATASTAX

Command	Description
`CAPTURE`	Captures command output and appends it to a file
`CONSISTENCY`	Shows the current consistency level, or given a level, sets it
`COPY`	Imports and exports CSV (comma-separated values) data
`DESCRIBE`	Provides information about a Cassandra cluster or data objects
`EXPAND`	Formats the output of a query vertically
`EXIT` or `QUIT`	Terminates cqlsh
`SHOW`	Shows the Cassandra version, host, or data type assumptions
`SOURCE`	Executes a file containing CQL statements
`TRACING`	Enables or disables request tracing

What shell commands does *cqlsh* support?

DATASTAX

- CQL commands must be terminated with semi-colon
- SOURCE

```
SOURCE 'file'
```

```
SOURCE './myscript.cql';
```

- COPY

```
COPY table_name ( column, ...)
FROM ( 'file_name' | STDIN )
WITH option = 'value' AND ... ;
```

```
COPY table_name ( column , ... )
TO ( 'file_name' | STDOUT )
WITH option = 'value' AND ... ;
```

```
COPY performers_by_style (style, name)
FROM ‘./performers_by_style.csv'
WITH HEADER = 'true';
```

What shell commands does *cqlsh* support?

- DESCRIBE

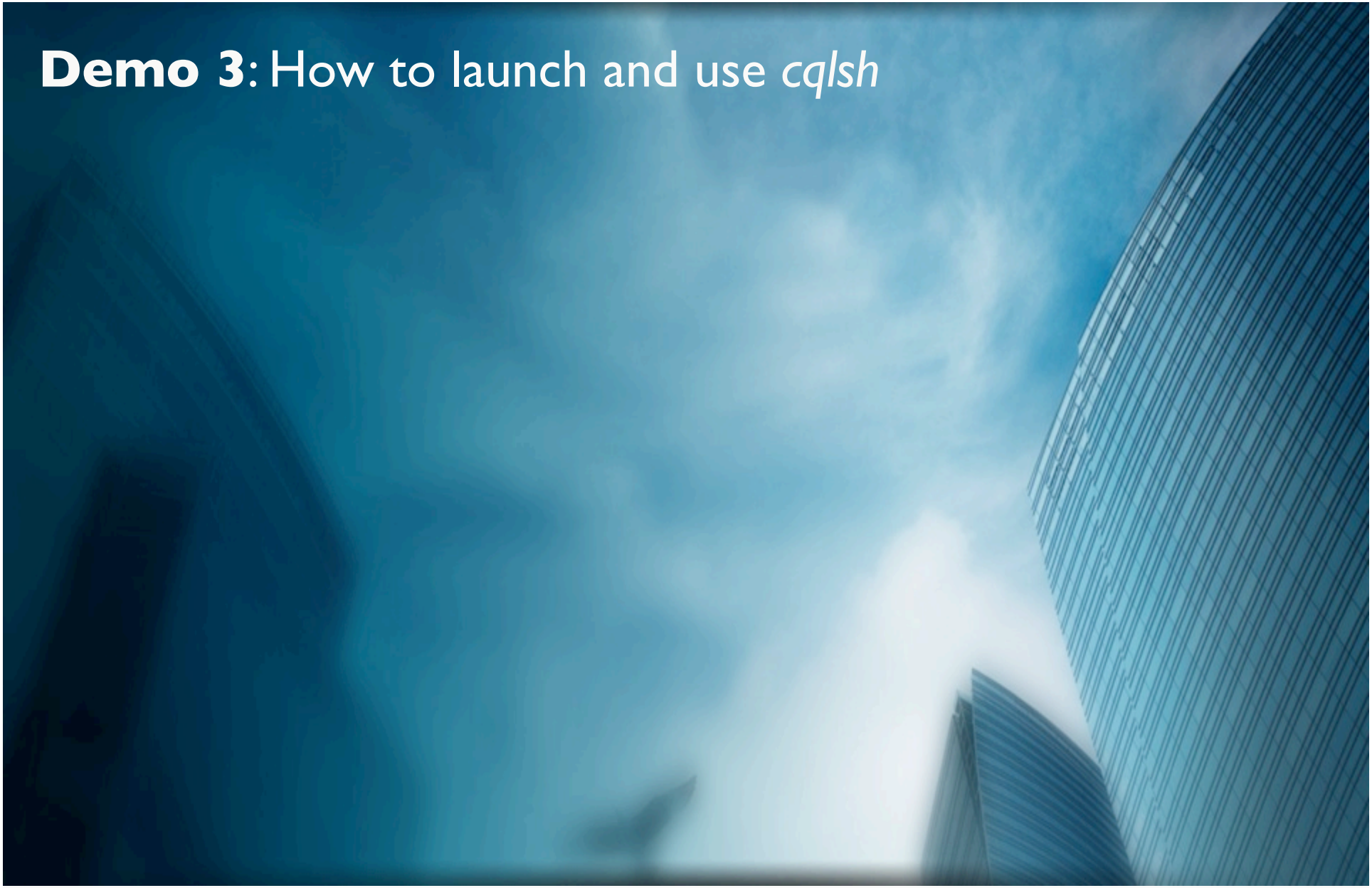
```
DESCRIBE CLUSTER | SCHEMA | KEYSPACES |
KEYSPACE keyspace_name | TABLES | TABLE table_name
```

```
DESCRIBE TABLE album;
```

- EXIT

```
EXIT | QUIT;
```

Demo 3: How to launch and use *cqlsh*



Learning Objectives

DATASTAX

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

What is a keyspace or schema?

DATASTAX

- Keyspace – a top-level namespace for a CQL table schema
  - Defines the replication strategy for a set of tables
    - Keyspace per application is a good idea
  - Data objects (e.g., tables) belong to a single keyspace

- Replication strategy – the number and pattern by which partitions are copied among nodes in a cluster
  - Two strategies available
    - Simple Strategy (used for prototyping)
    - Network Topology Strategy (production)

How to create, use and drop keyspaces/schemas?

DATASTAX

- To create a keyspace

```
CREATE KEYSPACE musicdb
WITH replication = {
'class': 'SimpleStrategy',
'replication_factor' : 3
};
```

- To assign the working default keyspace for a *cqlsh* session

```
USE musicdb;
```

- To delete a keyspace and all internal data objects

```
DROP KEYSPACE musicdb;
```

What is the syntax of the CREATE TABLE statement?

DATASTAX

- The CQL below creates a table in the current keyspace

Primary key declared inline

```
CREATE TABLE performer (
  name VARCHAR PRIMARY KEY,
  type VARCHAR,
  country VARCHAR,
  style VARCHAR,
  founded INT,
  born INT,
  died INT
);
```

Primary key declared in separate clause

```
CREATE TABLE performer (
  name VARCHAR,
  type VARCHAR,
  country VARCHAR,
  style VARCHAR,
  founded INT,
  born INT,
  died INT,
  PRIMARY KEY (name)
);
```

How are primary key, partition key, and clustering columns defined?

DATASTAX

- Simple partition key, no clustering columns

```
PRIMARY KEY ( partition_key_column )
```

- Composite partition key, no clustering columns

```
PRIMARY KEY ( ( partition_key_col1, …, partition_key_colN ) )
```

- Simple partition key and clustering columns

```
PRIMARY KEY ( partition_key_column,
              clustering_column1, …, clustering_columnM )
```

- Composite partition key and clustering columns

```
PRIMARY KEY ( ( partition_key_col1, …, partition_key_colN ),
              clustering_column1, …, clustering_columnM )
```

How are primary key, partition key, static and clustering columns defined?

DATASTAX

- Example

Can find all performers and albums for a given track title

```
CREATE TABLE albums_by_track (
  track_title VARCHAR,
  performer VARCHAR,
  year INT,
  album_title VARCHAR,
  PRIMARY KEY
  (track_title, performer,
   year, album_title)
);
```

Can find a performer, genre, and all track numbers and titles for a given album title and year

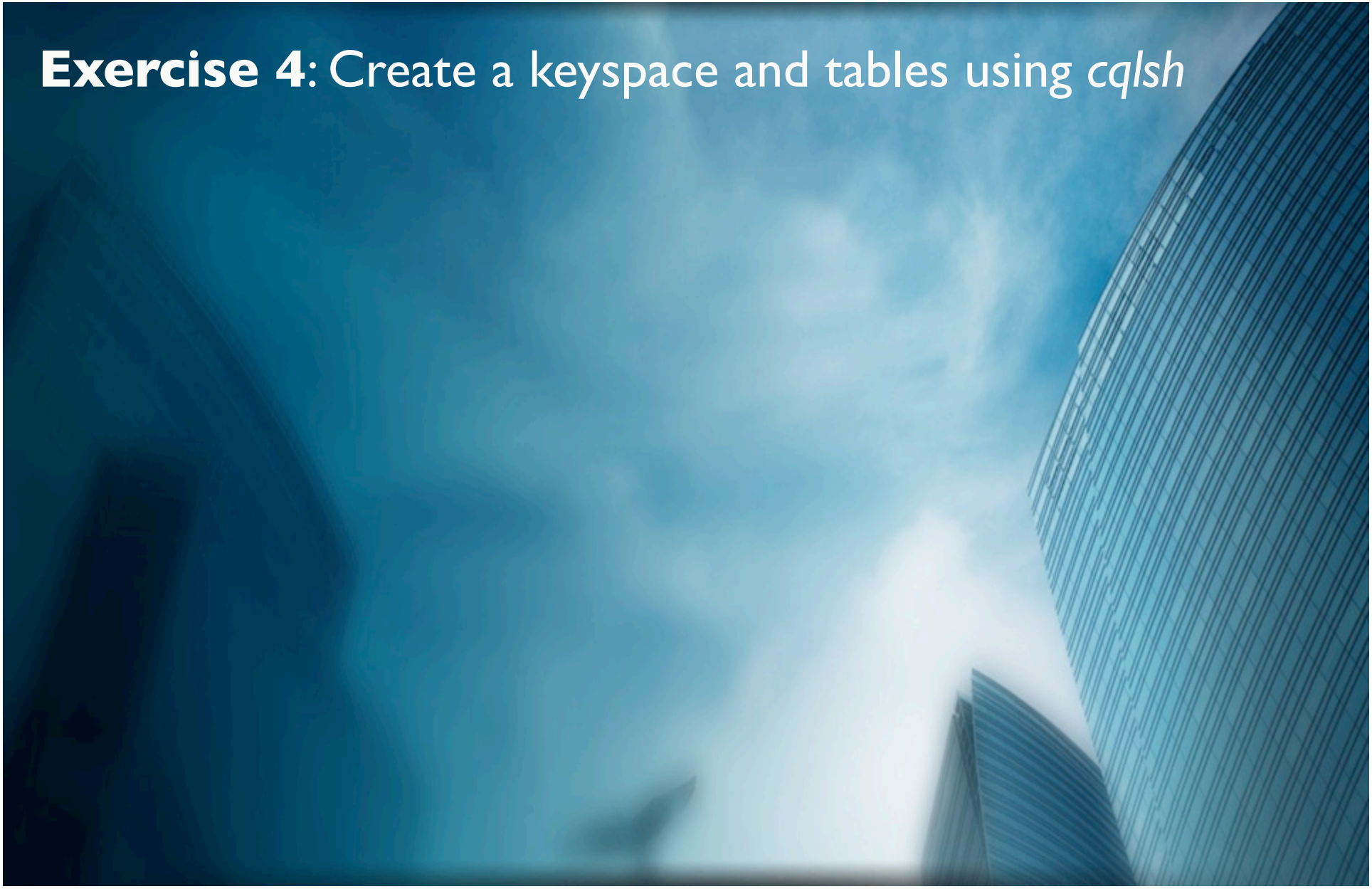
```
CREATE TABLE tracks_by_album (
  album_title VARCHAR,
  year INT,
  performer VARCHAR STATIC,
  genre VARCHAR STATIC,
  number INT,
  track_title VARCHAR,
  PRIMARY KEY
  ((album_title, year),
    number)
);
```

What CQL data types are available?

DATASTAX

CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Exercise 4: Create a keyspace and tables using *cqlsh*



What are UUID and TIMEUUID for?

DATASTAX

- UUID and TIMEUUID are universally unique identifiers
  - Generated programmatically
  - Format

    hex{8}-hex{4}-hex{4}-hex{4}-hex{12}

    52b11d6d-16e2-4ee2-b2a9-5ef1e9589328
  - Used to assign conflict-free (unique) identifiers to data objects
  - Numeric range so vast that duplication is statistically all but impossible
- UUID data type supports Version 4 UUIDs
  - Randomly generated sequence of 32 hex digits separated by dashes
  - 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328

What are UUID and TIMEUUID for?

DATASTAX

- TIMEUUID data type supports Version 1 UUIDs
  - Embeds a time value within a UUID
  - Generated using time (60 bits), a clock sequence number (14 bits), and MAC address (48 bits)

    `1be43390-9fe4-11e3-8d05-425861b86ab6`

    - CQL function `now()` generates a new TIMEUUID
  - Time can be extracted from TIMEUUID
    - CQL function `dateOf()` extracts the embedded timestamp as a date
  - TIMEUUID values in clustering columns or in column names are ordered based on time
    - DESC order on TIMEUUID lists most recent data first

What are UUID and TIMEUUID for?

DATASTAX

- Example
  - Users are identified by UUID
  - User activities (i.e., rating a track) are identified by TIMEUUID
    - A user may rate the same track multiple times
    - Activities are ordered by the time component of TIMEUUID

```
CREATE TABLE track_ratings_by_user (
  user UUID,
  activity TIMEUUID,
  rating INT,
  album_title VARCHAR,
  album_year INT,
  track_title VARCHAR,
  PRIMARY KEY (user, activity)
) WITH CLUSTERING ORDER BY (activity DESC);
```

What is TIMESTAMP for?

DATASTAX

- TIMESTAMP holds date and time
  - 64-bit integer representing a number of milliseconds since January 1 1970 at 00:00:00 GMT
  - Entered as
    - 64-bit integer
    - String literal in the ISO 8601 format
      - 1979-12-18 08:12:51-0400
      - 2014-02-27
      - Other variations are allowed
  - Displayed in *cqlsh* as
    - yyyy-mm-dd HH:mm:ssZ

What are special properties of the COUNTER data type?

DATASTAX

- Cassandra supports distributed counters
  - Useful for tracking a count
  - Counter column stores a number that can only be updated
    - Incremented or decremented
    - Cannot assign an initial value to a counter (initial value is 0)
  - Counter column cannot be part of a primary key
  - If a table has a counter column, all non-counter columns must be part of a primary key

```
CREATE TABLE ratings_by_track (
  album_title VARCHAR,  album_year INT,  track_title VARCHAR,
  num_ratings COUNTER,
  sum_ratings COUNTER,
  PRIMARY KEY (album_title, album_year, track_title)
);
```

What is the purpose of the CLUSTERING ORDER BY clause?

DATASTAX

- CLUSTERING ORDER BY defines how data values in clustering columns are ordered (ASC or DESC) in a table
  - ASC is the default order for all clustering columns
  - When retrieving data, the default order or the order specified by a CLUSTERING ORDER BY clause is used
    - The order can be reversed in a query using the ORDER BY clause

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

Exercise 5: Create tables using UUID, TIMEUUID, and COUNTER columns

DATASTAX

What is the syntax of the ALTER TABLE statement?

DATASTAX

- ALTER TABLE manipulates the table metadata
  - Adding a column

```
ALTER TABLE album ADD cover_image VARCHAR;
```

  - Changing a column data type

```
ALTER TABLE album ALTER cover_image TYPE BLOB;
```

    - Types must be compatible
    - Clustering and indexed columns are not supported

  - Dropping a column

```
ALTER TABLE album DROP cover_image;
```

    - PRIMARY KEY columns are not supported

What is the syntax of the DROP TABLE statement?

DATASTAX

- DROP TABLE removes a table (all data in the table is lost)

```
DROP TABLE album;
```

What are collection columns for?

DATASTAX

- Collection columns are multi-valued columns
  - Designed to store discrete sets of data (e.g., tags for a blog post)
    - A collection is retrieved in its entirety
  - 64,000 - maximum number of elements in a collection
    - In practice – dozens or hundreds
  - 64 KB - maximum size of each collection element
    - In practice – much smaller
  - Collection columns
    - cannot be part of a primary key
    - cannot be part of a partition key
    - cannot be used as a clustering column
    - cannot nest inside of another collection

How are collection columns defined?

- Set – typed collection of unique values

```
keywords SET<VARCHAR>
```

  - Ordered by values
  - No duplicates

- List – typed collection of non-unique values

```
songwriters LIST<VARCHAR>
```

  - Ordered by position
  - Duplicates are allowed

- Map – typed collection of key-value pairs

```
tracks MAP<INT,VARCHAR>
```

  - Ordered by keys
  - Unique keys but not values

What is a user-defined type?

DATASTAX

- User-defined types group related fields of information
  - Represents related data in a single table, instead of multiple, separate tables
  - Uses any data type, including collections and other user-defined types
  - Reserved words cannot be used as a name for a user-defined type
    - byte
    - smallint
    - complex
    - enum
    - date
    - interval
    - macaddr
    - bitstring

```
CREATE TYPE track (
  album_title VARCHAR,
  album_year INT,
  track_title VARCHAR,
);
```

What is a user-defined type?

DATASTAX

- Table columns can be user-defined types
  - Requires the use of the *frozen* keyword in C* 2.1
  - A user-defined type can be used as a data type for a collection

```
CREATE TABLE musicdb.track_ratings_by_user (
  user UUID,
  activity TIMEUUID,
  rating INT,
  song frozen <track>,
PRIMARY KEY (user, activity)
) WITH CLUSTERING ORDER BY (activity DESC);
```

What is the syntax of the ALTER TYPE statement?

DATASTAX

- ALTER TYPE can change a user-defined type
  - Change the type of a field
    - Types must be compatible

```
ALTER TYPE track ALTER album_title TYPE BLOB;
```

  - Add a field to a type

```
ALTER TYPE track ADD track_number INT;
```

  - Rename a field of a type

```
ALTER TYPE track RENAME album_year TO year;
```

  - Rename a user-defined type

```
ALTER TYPE track RENAME TO song;
```

What is the syntax of the DROP TYPE statement?

DATASTAX

- DROP TYPE removes a user-defined type
  - Cannot drop a user-defined type that is in use by a table or another type

```
DROP TYPE track;
```

What is a tuple?

- Tuples hold fixed-length sets of typed positional fields
  - Convenient alternative to creating a user-defined type
  - Accommodates up to 32768 fields, but generally only use a few
  - Useful when prototyping
  - Must use the frozen keyword in C* 2.1
  - Tuples can be nested in other tuples

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

What is a secondary index?

DATASTAX

- Tables are indexed on columns in a primary key
  - Search on a partition key is very efficient
  - Search on a partition key and clustering columns is very efficient
  - Search on other columns is not supported
- Secondary indexes
  - Can index additional columns to enable searching by those columns
    - one column per index
  - Cannot be created for
    - counter columns
    - static columns

How do you create and drop secondary indexes?

DATASTAX

- To create a secondary index

```
CREATE TABLE performer (
  name VARCHAR,
  type VARCHAR,
  country VARCHAR,
  style VARCHAR,
  founded INT,
  born INT,
  died INT,
  PRIMARY KEY (name)
);

CREATE INDEX performer_style_key ON performer (style);
```

- To drop a secondary index

```
DROP INDEX performer_style_key;
```

When do you want to use a secondary index?

DATASTAX

- Secondary indexes are for searching convenience
  - Use with low-cardinality columns
    - Columns that may contain a relatively small set of distinct values
      - For example, there are many artists but only a few dozen music styles
      - Allows searching for all artists for a specified style (a potentially expensive query because it may return a large result set)
  - Use with smaller datasets or when prototyping
- Do not use
  - On high-cardinality columns
  - On counter column tables
  - On a frequently updated or deleted columns
  - To look for a row in a large partition unless narrowly queried
    - e.g., search on both a partition key and an indexed column

Exercise 6: Add user-defined type, alter tables, add collection column, and add secondary indexes

Learning Objectives

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

What is *DevCenter* and how do you launch it?

DATASTAX

- Cassandra client with the GUI interface
  - IDE for developers and administrators
  - Supports Cassandra Query Language statements
  - Does not support *cqlsh* commands
    - SOURCE, COPY, DESCRIBE, etc.
- Launching on Linux

```
$ ./DevCenter
```

- Launching on Windows

```
DevCenter.exe
```

- Launching on Mac OS

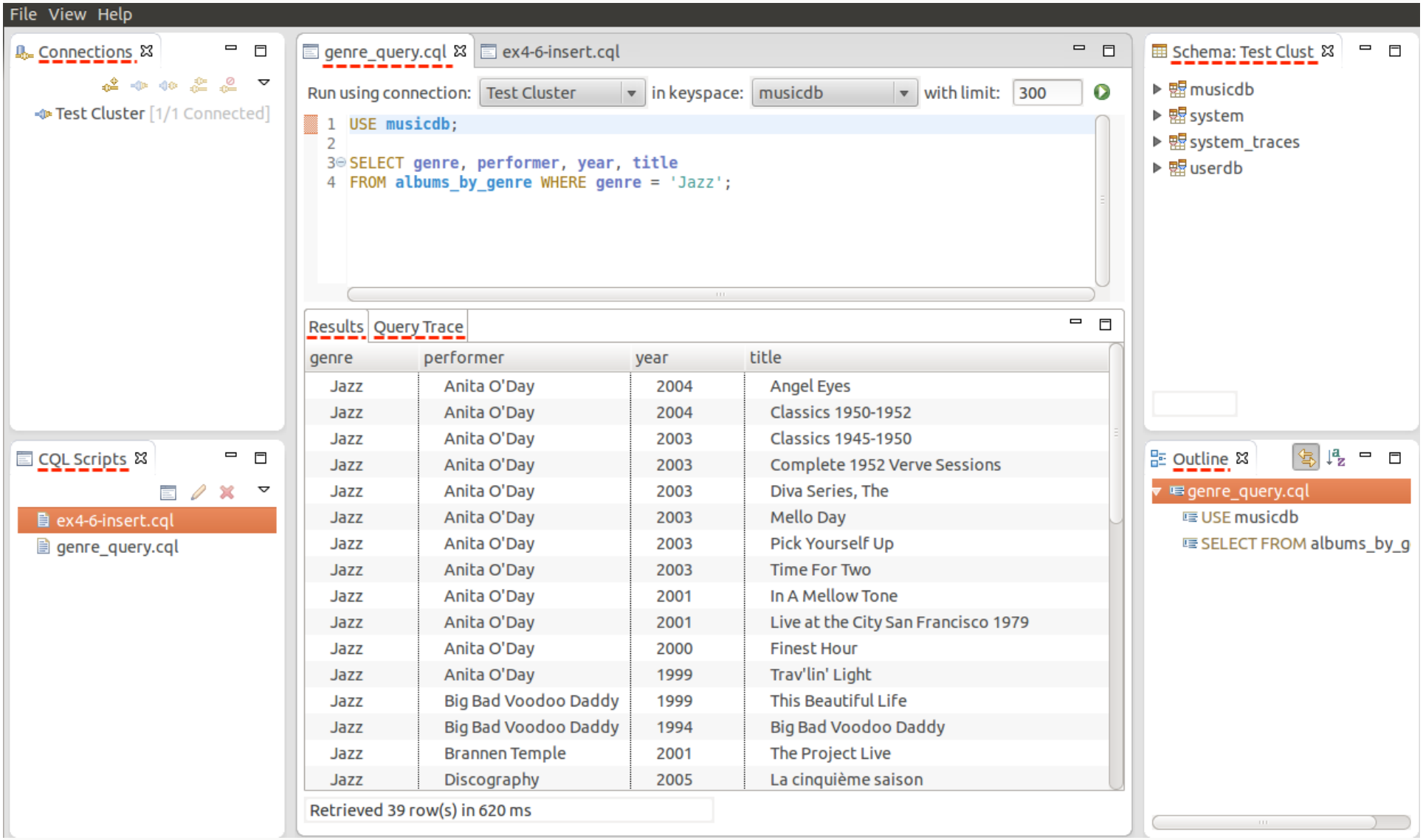
```
DevCenter.app
```

What are the main features of *DevCenter*?

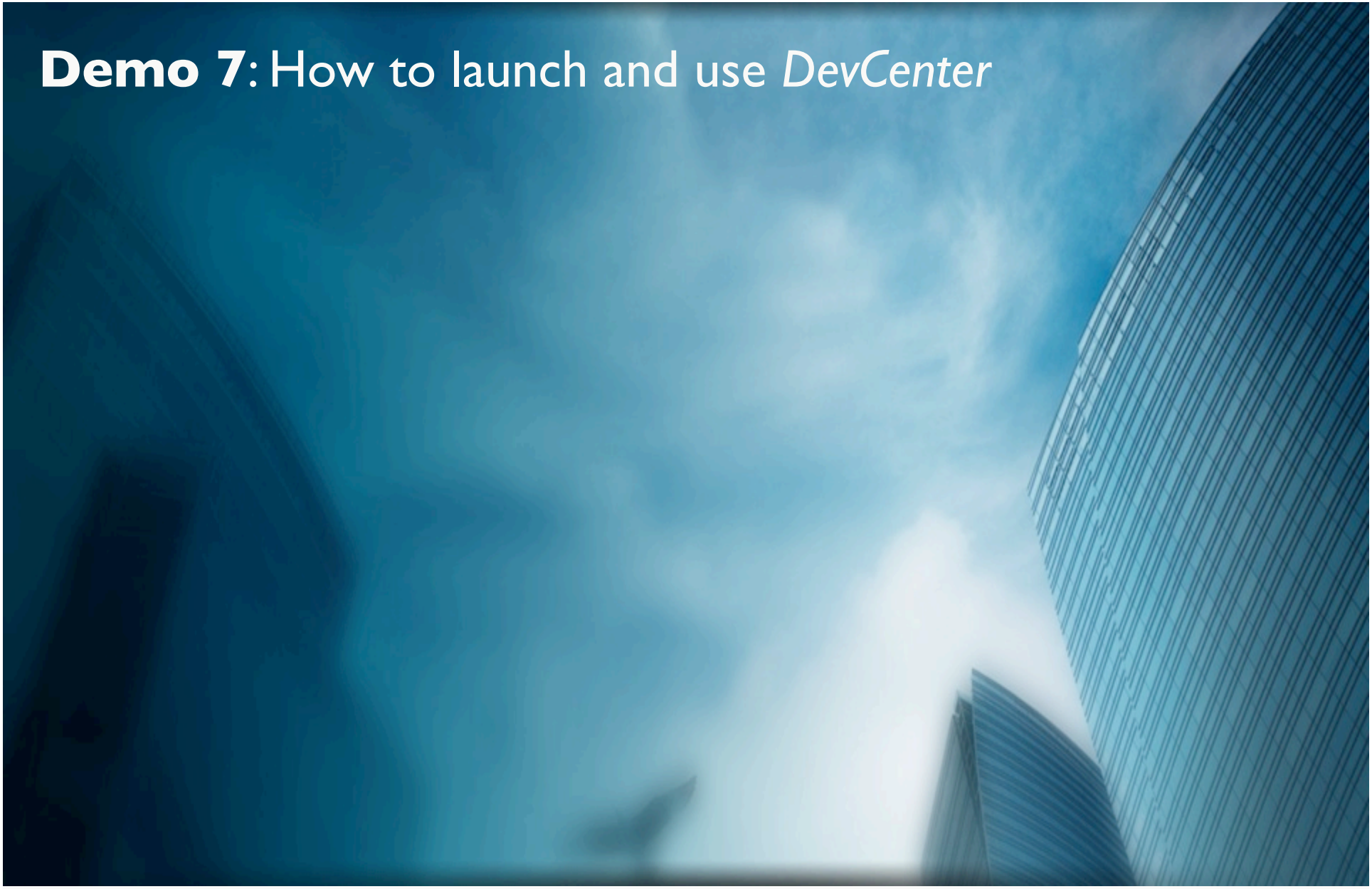
DATASTAX

- Main features
  - Create and manage Cassandra connections
  - Create, edit, and execute CQL scripts
    - syntax highlighting
    - code auto-completion
    - real-time script validation against the current connection
  - Explore database objects via the Schema explorer
  - Navigate long CQL scripts via the Outline view
  - Execute CQL queries and view results and query trace

What are the main features of *DevCenter*?



Demo 7: How to launch and use *DevCenter*



Learning Objectives

DATASTAX

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

What is the syntax of the INSERT statement?

DATASTAX

```
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?

DATASTAX

- 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');
```

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

What is the syntax of the INSERT statement?

- To insert a row into a table with UUID and TIMEUUID columns

```
CREATE TABLE track_ratings_by_user (
  user UUID,
  activity TIMEUUID,
  rating INT,
  album_title VARCHAR,
  album_year INT,
  track_title VARCHAR,
  PRIMARY KEY (user, activity)
) WITH CLUSTERING ORDER BY (activity DESC);
```

```
INSERT INTO track_ratings_by_user
(user,activity,rating,album_title,album_year,track_title)
VALUES (52b11d6d-16e2-4ee2b2a9-5ef1e9589328,
dbf3fbfc-9fe4-11e3-8d05-425861b86ab6, 5,'Revolver',1966,'Yellow
Submarine');
```

user	activity	album_title	album_year	rating	track_title
52b11d6d-16e2- …	dbf3fbfc-9fe4-…	Revolver	1966	5	Yellow Submarine

What is the syntax of the UPDATE statement?

DATASTAX

```
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

What is the syntax of the UPDATE statement?

DATASTAX

- To update a row in a table

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

- 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

What is an "upsert"?

DATASTAX

- 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

What are lightweight transactions or 'compare and set'?

DATASTAX

- 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

```
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', 1995, 'Beatlemania') IF NOT EXISTS;
```

[applied]	performer	year
false	The Beatles	1966

What are lightweight transactions or Compare and Set?

- 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

```
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

What is the purpose of the TTL option?

DATASTAX

- 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

What is the purpose of the TTL option?

- To store a row for 86400 seconds (1 day)

```
INSERT INTO track_ratings_by_user
(user,activity,rating,album_title,album_year,track_title)
VALUES (52b11d6d-16e2-4ee2-b2a9-5ef1e9589328,
dbf3fbfc-9fe4-11e3-8d05-25861b86ab6,5,'Revolver',1966,'Yellow
Submarine')
USING TTL 86400;
```

  - Re-inserting the same row before it expires will overwrite TTL

- To store a column value for 30 seconds

```
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;
```

  - Only column 'rating' for this row is affected by TTL

What is the syntax of the DELETE statement?

DATASTAX

- 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;
```

- 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;
```

- 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;
```

What is the syntax of the TRUNCATE statement?

DATASTAX

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

```
TRUNCATE track_ratings_by_user;
```

Exercise 8: Inserting and updating values using DevCenter

How do you manipulate counters?

DATASTAX

- COUNTER – defining and updating
  - INSERT is not allowed
  - Initial counter value is 0

```
CREATE TABLE stats (
  performer VARCHAR,
  albums COUNTER,
  concerts COUNTER,
  PRIMARY KEY (performer)
);
```

```
UPDATE stats
SET albums = albums + 1, concerts = concerts + 10
WHERE performer = 'The Beatles';
```

performer	albums	concerts
The Beatles	1	10

How do you manipulate collections?

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

```
CREATE TABLE band (
  name VARCHAR PRIMARY KEY,
  members SET<VARCHAR>
);
```

```
INSERT INTO band (name, members)
VALUES ('The Beatles', {'Paul', 'John', 'George', 'Ringo'});
```

name	members
The Beatles	{'George', 'John', 'Paul', 'Ringo'}

How do you manipulate collections?

- CQL set – performing union, difference and deletion

```
UPDATE band SET members = members +
{'Pete', 'Stuart', 'Paul', 'Jonathan'}
WHERE name = 'The Beatles';
```

name	members
The Beatles	{'George', 'John', 'Jonathan', 'Paul', 'Pete', 'Ringo', 'Stuart'}

```
UPDATE band SET members = members - {'Jonathan'}
WHERE name = 'The Beatles';
```

name	members
The Beatles	{'George', 'John', 'Paul', 'Pete', 'Ringo', 'Stuart'}

```
DELETE members FROM band WHERE name = 'The Beatles';
```

name	members
The Beatles	

How do you manipulate collections?

- 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 (252608cb-0f56-4cf3-82ee-b7fe00f3920f,
        'I Want to Hold Your Hand', ['John', 'Paul']);
```

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

How do you manipulate collections?

DATASTAX

- CQL list – appending and prepending

```
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

```
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

How do you manipulate collections?

- CQL list – updating, subtracting and deleting

```
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

```
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

```
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

How do you manipulate collections?

DATASTAX

- 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'}

How do you manipulate collections?

- CQL map – updating

```
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'}

```
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'}

How do you manipulate collections?

- CQL map – deleting

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

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	

How do you manipulate user-defined types and tuples?

- User-defined type - Defining and inserting

```
CREATE TYPE track (
  album_title text,
  album_year int,
  track_title text
);

CREATE TABLE track_ratings_by_user (
  user UUID,
  activity TIMEUUID,
  rating INT,
  song frozen <track>,
  PRIMARY KEY (user, activity)
) WITH CLUSTERING ORDER BY (activity desc));
```

```
INSERT INTO track_ratings_by_user (user, activity, rating,
song ) VALUES (6ed4f220-5361-11e4-8d89-c971d060d947,
779a96e0-6eea-11e4-9803-0900200c9a66, 10,
{album_title: 'Let It Be', album_year: 1970,
 track_title: 'Let It Be'});
```

user	activity	rating	song
62d4f220-5361-…	779a96e0-6eea-…	10	{album_title: 'Let It Be', album_year: 1970, track_title: 'Let It Be'}

How do you manipulate user-defined types and tuples?

- User-defined type - Updating

```
UPDATE track_ratings_by_user
SET song = {album_title: 'Let It Be', album_year: 1970,
            track_title: 'Two of Us'}
WHERE user = 6ed4f220-5361-11e4-8d89-c971d060d947 AND
activity = 779a96e0-6eea-11e4-9803-0900200c9a66;
```

user	activity	rating	song
62d4f220-5361-…	779a96e0-6eea-…	10	{album_title: 'Let It Be', album_year: 1970, track_title: 'Two of Us'}

- User-defined type - Deleting

```
DELETE song from track_ratings_by_user WHERE user =
6ed4f220-5361-11e4-8d89-c971d060d947 AND activity =
779a96e0-6eea-11e4-9803-0900200c9a66;
```

user	activity	rating	song
62d4f220-5361-…	779a96e0-6eea-…	10	

How do you manipulate user-defined types and tuples?

- Tuple - Defining and inserting

```
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 (6ed4f220-5361-11e4-8d89-c971d060d947,
(3.0, 6.0, 9.0, 7.0, 6.0, 5.0, 7.0, 9.0, 11.0, 8.0));
```

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

How do you manipulate user-defined types and tuples?

- Tuple - Updating

```
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;
```

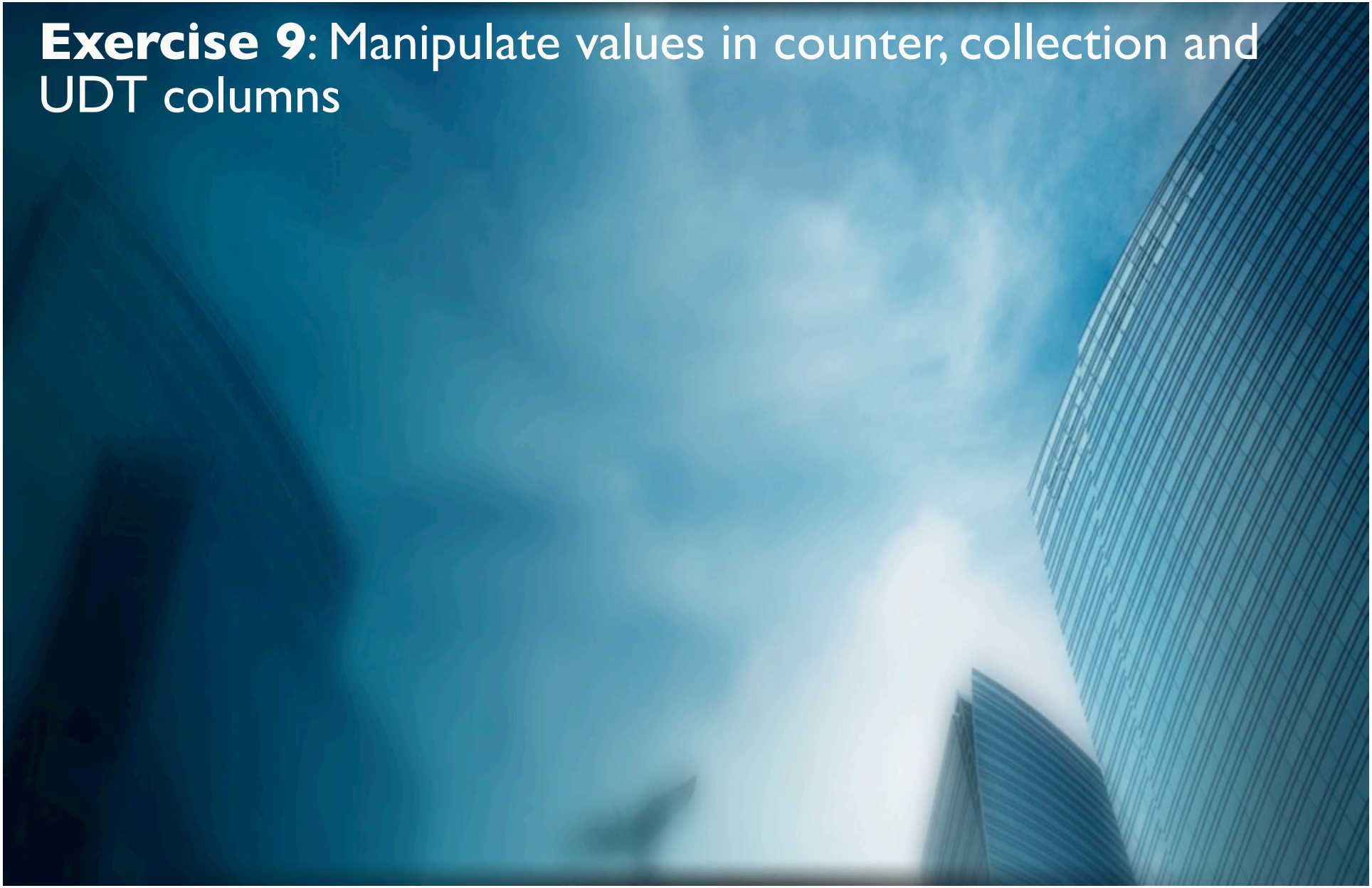
id	equalizer
62d4f220-5361-...	(4.0, 1.6, -1.8, -5.6, -0.7, 0.9, 2.9, 4.3, 4.3, 4.3)

- Tuple - Deleting

```
DELETE equalizer from user
WHERE id = 6ed4f220-5361-11e4-8d89-c971d060d947
```

id	equalizer
62d4f220-5361-...	

Exercise 9: Manipulate values in counter, collection and UDT columns



What is the purpose of the BATCH statement?

- 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 a partially executed batch

What is the purpose of the BATCH statement?

DATASTAX

- Example

```
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;
```

- BEGIN UNLOGGED BATCH
  - Does not write to the batchlog
  - Saves time but no longer atomic
  - Allows operations on counter columns

What is the purpose of the BATCH statement?

DATASTAX

- 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

```
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;
```

What is the syntax of the SELECT statement?

DATASTAX

- 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?

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

What is the syntax of the SELECT statement?

DATASTAX

- 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;
```

What predicates are allowed in the WHERE clause?

DATASTAX

- 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;
```

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

What predicates are allowed in the WHERE clause?

DATASTAX

- 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;
```

album_title	year	number	track_title
Revolver	1966	6	Yellow Submarine

What predicates are allowed in the WHERE clause?

DATASTAX

- 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);
```

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

What predicates are allowed in the WHERE clause?

DATASTAX

- 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;
```

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

What predicates are allowed in the WHERE clause?

DATASTAX

- 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');
```

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

What predicates are allowed in the WHERE clause?

DATASTAX

- Range search
  - >, >=, <, <=
  - Can only a range search on a partition key using the `token()` function

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

    - 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

What predicates are allowed in the WHERE clause?

DATASTAX

- Range search – subset of rows

```
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

What predicates are allowed in the WHERE clause?

DATASTAX

- Range search – slice of a partition

```
CREATE TABLE track_by_duration ( …
PRIMARY KEY (track_title, minutes, seconds));
```

```
SELECT album_title, year, number, track_title
FROM tracks_by_duration
WHERE album_title = 'Revolver' AND year = 1966 AND
                (minutes, seconds) >= (2, 30) AND
                (minutes, seconds) < (6, 0);
```

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

What is the purpose of the LIMIT clause?

DATASTAX

- LIMIT restricts the number of returned rows
  - Default value is 10,000 (cqlsh)

- To retrieve less rows

```
SELECT * FROM performer LIMIT 10;
```

- To retrieve more rows

```
SELECT * FROM performer LIMIT 100000;
```

What is the purpose of the ALLOW FILTERING clause?

DATASTAX

- 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

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

How are indexed columns used in a query?

DATASTAX

- 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

```
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;
```

How are indexed collection columns queried?

DATASTAX

- 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

```
CREATE INDEX ON album (tracks);
```

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

How are indexed UDT and tuple columns queried?

DATASTAX

- 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);
```

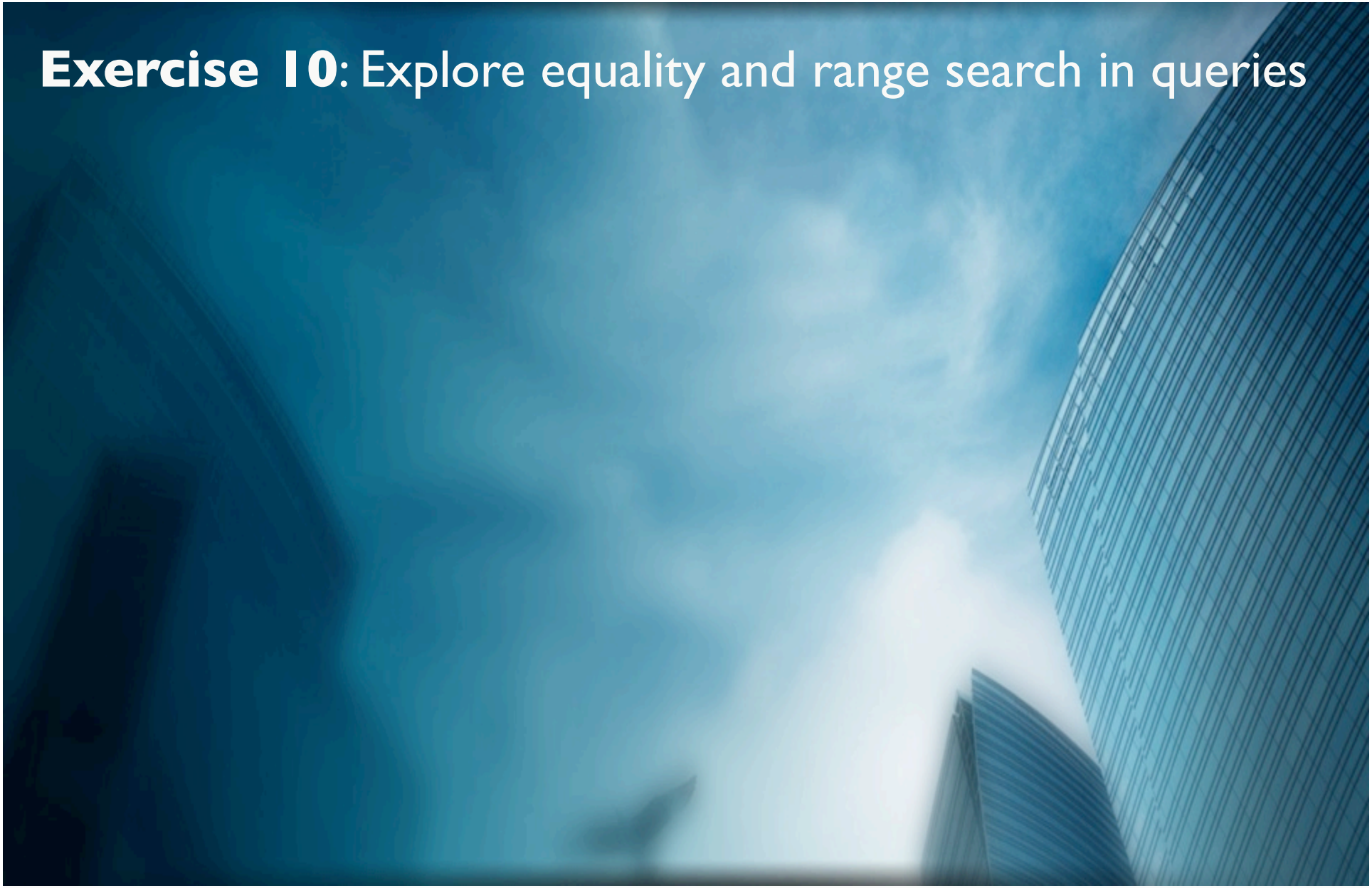
```
SELECT * FROM track_ratings_by_user
WHERE song = {album_title: 'Beatles For Sale',
              album_year: 1964,
              track_title: 'Cant Buy Me Love'};
```

- Tuple – Search all fields

```
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);
```

Exercise 10: Explore equality and range search in queries



What is the purpose of the ORDER BY clause?

DATASTAX

- 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

```
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;
```

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

What functions are available in CQL?

DATASTAX

- TIMEUUID functions
  - `dateOf()` – extracts the timestamp as a date of a timeuuid column

```
SELECT dateOf(timeuuid_column), … FROM …;
```

  - `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

```
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

```
SELECT unixTimestampOf(timeuuid_column), … FROM …;
```

What functions are available in CQL?

DATASTAX

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

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

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

- Token access function
  - `token()` function

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

Demo 11: Explore queries with various predicates (optional)

Learning Objectives

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

What is data modeling?

DATASTAX

- Data modeling is a process that involves
  - Collection and analysis of data requirements in an information system
  - Identification of participating entities and relationships among them
  - Identification of data access patterns
  - A particular way of organizing and structuring data
  - Design and specification of a database schema
  - Schema optimization and data indexing techniques

- Data modeling = Science + Art

What are the key steps of data modeling?

DATASTAX

- Key steps of data modeling for Cassandra

1. Understand data and application queries
   - Data may or may not exist in some format (RDBMS, XML, CSV, …)
   - Queries can be organized into a query graph
2. Design column families
   - Design is based on access patterns or queries over data
3. Implement the design using CQL
   - Optimizations concerning data types, keys, partition sizes, ordering

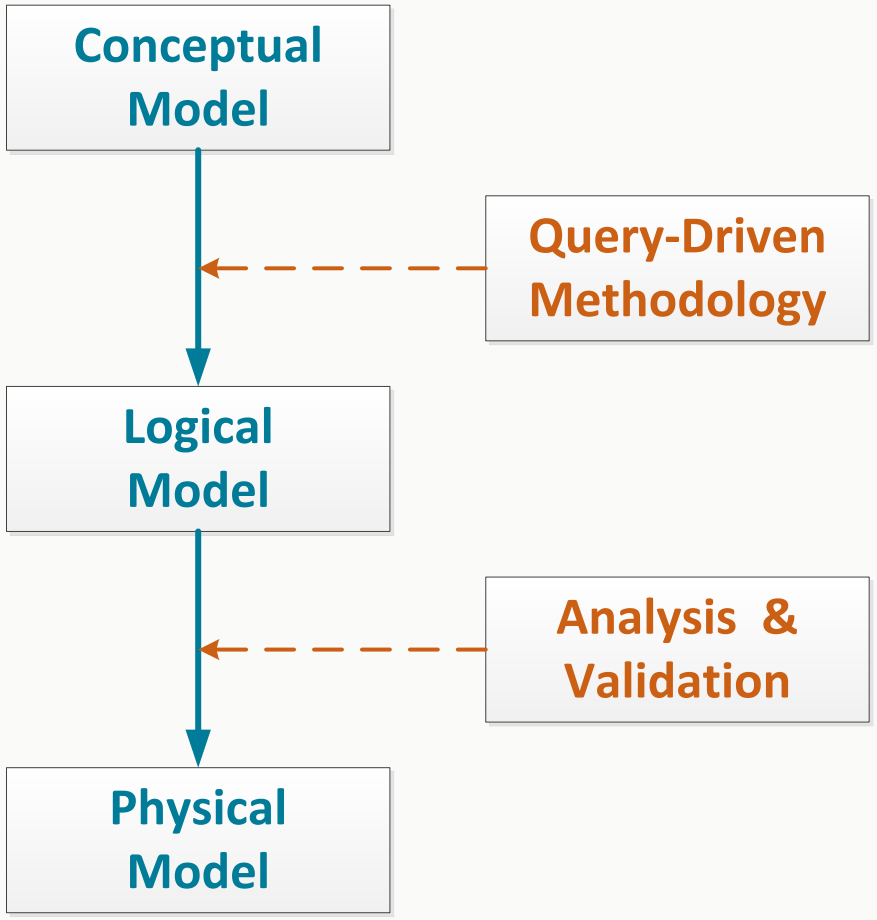
What are the key steps of data modeling?

DATASTAX

- The products of the data modeling steps are documented as
  - Conceptual data model
    - Technology-independent, unified view of data
    - Entity-relationship model, dimensional model, etc.
  - Logical data model
    - Unique for Cassandra
    - Column family diagrams
  - Physical data model
    - Unique for Cassandra
    - CQL definitions

What is a data modeling framework?

- Defines transitions between models
  - Query-driven methodology
  - Formal analysis and validation
- Defines a scientific approach to data modeling
  - Modeling rules
  - Mapping patterns
  - Schema optimization techniques



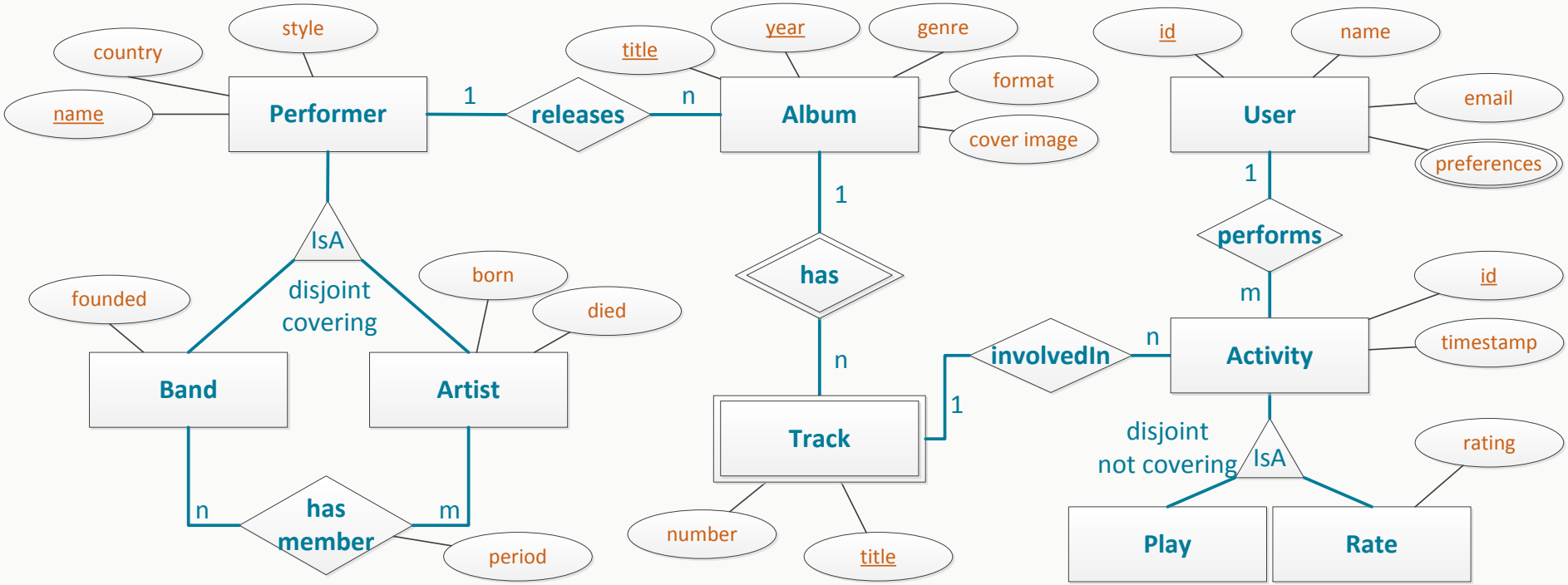
What is a conceptual data model?

DATASTAX

- Unified view of data
  - Captures understanding of data entities and relationships
- Technology-independent
  - Has nothing to do with existing database models
- Graphical representations
  - Entity-relationship diagrams
    - Chen notation recommended
  - Dimensional modeling diagrams
  - UML diagrams

What is a conceptual data model?

- Conceptual data model for music data
  - ER diagram (Chen notation)
  - Describes entities, relationships, roles, keys, cardinalities
    - What is possible and what is not in existing or future data



What is the Cassandra data modeling methodology?

DATASTAX

- Defines how a conceptual DM maps to a logical DM
  - Modeling rules
    - Ensure that a query is efficiently supported by a column family
  - Mapping patterns
    - Pattern input: one or more components of a conceptual DM
    - Pattern input: a query
    - Pattern output: a column family or several alternative solutions
- Enables an algorithmic approach to Cassandra data modeling
  - For each query
    - Identify a subset of the conceptual DM that describes query data
    - Apply a suitable mapping pattern on the subset and the query

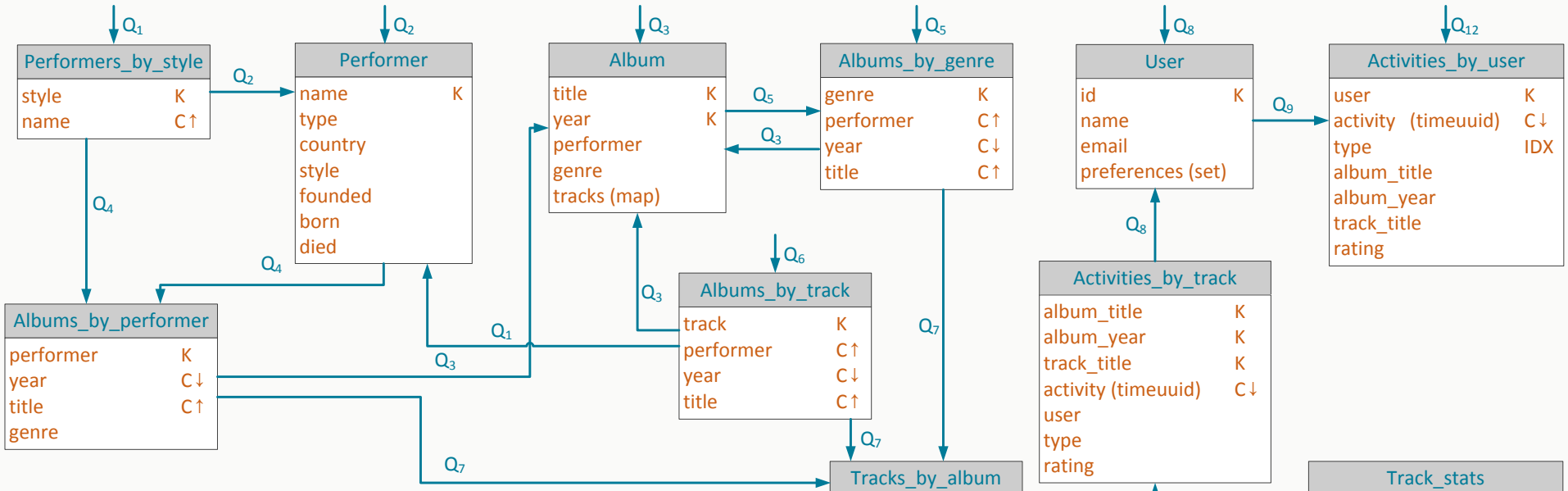
What is a logical data model?

DATASTAX

- Data is viewed and organized into column families or tables
  - Both column families and tables can be used at the logical level
    - Table is a two-dimensional view of a multi-dimensional column family

- Chebotko Diagram
  - Graphical representation of a logical data model
  - A column family is represented by a rectangle
    - Column family name
    - Columns that may optionally be designated as *K* (partition key), *C* (clustering column), *S* (static column), and *IDX* (indexed column)
  - Access patterns are represented by links between column families
    - Labeled with queries

What is a logical data model?



ACCESS PATTERNS

Q_1: Find performers for a specified style; order by performer (ASC).
Q_2: Find information for a specified performer (artist or band).
Q_3: Find information for a specified album (title and year).
Q_4: Find albums for a specified performer; order by album release year (DESC) and title (ASC).
Q_5: Find albums for a specified genre; order by performer (ASC), year (DESC), and title (ASC).
Q_6: Find albums and performers for a specified track title; order by performer (ASC), year (DESC), and title (ASC).
Q_7: Find tracks for a specified album (title and year); order by track number (ASC).
Q_8: Find information for a specified user.
Q_9: Find activities for a specified user; order by activity time (DESC).
Q_{10}: Find statistics for a specified track.
Q_{11}: Find user activities for a specified track; order by activity time (DESC).
Q_{12}: Find user activities for a specified activity type.
...

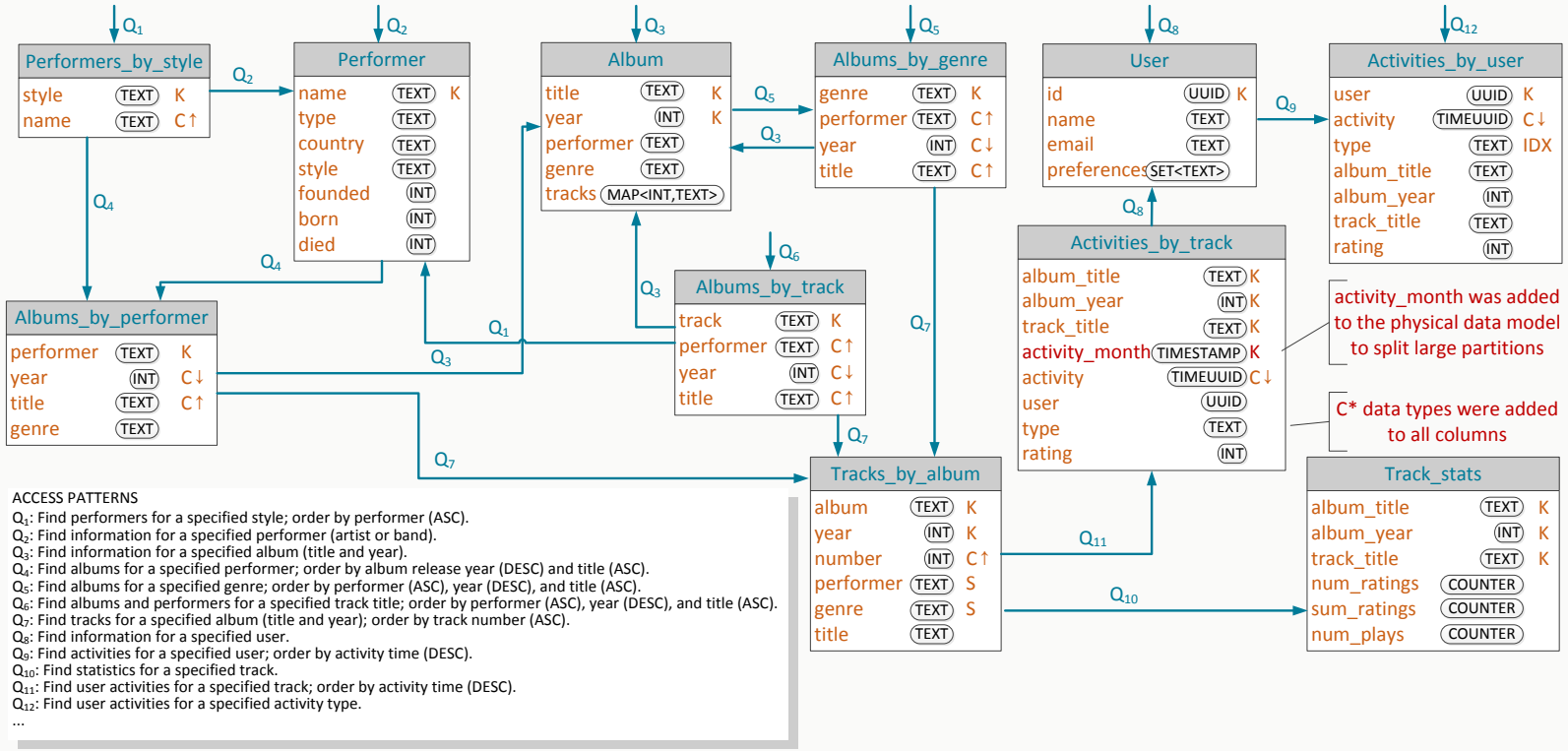
How do you analyse and validate a logical design?

DATASTAX

- Important considerations
  - Natural or surrogate keys?
  - Are write conflicts (overwrites) possible?
  - What data types to use?
  - How large are partitions?
  - How much data duplication is required?
  - Are client-side joins required and at what cost?
  - Are data consistency anomalies possible?
  - How to enable transactions and data aggregation?
  - …
- Various optimization techniques are defined and applied
  - Result in a physical data model

What is a physical data model?

- Final blueprint of database schema design
  - CQL script that instantiates a database schema in Cassandra
  - Chetbotko Diagrams can be used at the physical level to visualize the design
    - When there are significant differences from the logical design
    - A physical-level Chetbotko Diagram should show column data types



Is relational database design similar to Cassandra database design?

DATASTAX

No!

Cassandra

- Multi-dimensional column family
  - Equally good for simple and complex data
- All data required to answer a query must be nested in a column family
  - Referential integrity is a non-issue
- Data modeling methodology is driven by queries and data
  - Data duplication is considered normal (side effect of data nesting)

Relational

- Two-dimensional relation
  - Suited for simple data
  - Complex data requires many relations and "star" schemas
- Data from many relations is combined to answer a query
  - Referential integrity is important
- Data modeling is driven by data only
  - Data duplication is considered a problem (normalization theory)

How do you migrate from a relational database to Cassandra?

DATASTAX

- The common ground
  - The conceptual data model is the same (technology-independent)
  - Application queries executed over data are the same
    - SQL and CQL are not
- General idea
  - Extract (reverse engineer) a conceptual data model from a relational database schema
  - Analyze queries
  - Perform logical and physical design for Cassandra as usual
  - Execute SQL queries and import their results into respective column families in Cassandra
  - Rewrite queries in CQL
- There can be many nuances

Where do you learn more about data modeling?

- A course specifically dedicated to data modeling
  - Apache Cassandra: Data Modeling
- datastax.com
- planetcassandra.org
- cassandra.apache.org

Demo 12: Explore CQL and data modeling resources

Summary

DATASTAX

- Data in Cassandra is stored in column families or tables
- Column family is a set of rows with unique row keys
- Table is a set of partitions with unique partition keys
- Table is a two-dimensional view of a multi-dimensional column family
- Table partitions and partition keys correspond to column family rows and row keys
- Table rows are different from column family rows
- Table partitions can be single-row or multi-row depending on the absence or presence of clustering columns, respectively
- Table primary key uniquely identifies a row and is formed by a partition key and clustering columns

Summary

- CQL keyspace-related statements: CREATE KEYSPACE, USE, DROP KEYSPACE
- CQL table-related statements: CREATE TABLE, ALTER TABLE, DROP TABLE
- CQL index-related statements: CREATE INDEX, DROP INDEX
- CQL data types: VARCHAR, TEXT, INT, UUID, TIMEUUID, TIMESTAMP, COUNTER, SET, LIST, MAP, etc.
- CQL data manipulation statements: INSERT, UPDATE, DELETE, TRUNCATE, BATCH, SELECT (INSERT and UPDATE have a TTL option)
- CQL query clauses: SELECT, FROM, WHERE, ORDER BY, LIMIT, ALLOW FILTERING

Summary

- Data modeling steps require to understand data and queries, design column families, optimize, and implement tables in CQL
- Conceptual data model is technology-independent
- Logical data model is captured using column family diagrams
- Physical data model is captured in CQL schema definitions
- Data modeling framework defines transitions between conceptual, logical and physical data models
- Data modeling methodology is query-driven

Review Questions

- What is the relationship between a column family and a CQL table?
- How are wide rows implemented in CQL?
- How are clustering columns ordered?
- What is the difference between UUID and TIMEUUID?
- When should secondary indexes be used?
- Are CQL counters 100% accurate?
- How does an upsert work?
- What predicates are allowed in a CQL query?
- When should the ALLOW FILTERING clause be used?
- How can data from two tables be combined in a CQL query?
- What are components of the data modeling framework?
- What is the purpose of Chetboko Diagrams?