DATASTAX ACADEMY

DS220

Practical Application Data Modeling with Apache Cassandra™

DATASTAX ACADEMY

Course Introduction

DATASTAX ACADEMY

Student Introduction

Getting to know you

- Name
- Where you are from
- Where you work
- Goals for using Cassandra
- Prior experience
- Personal trivia

DATASTAX ACADEMY

Learning Objectives

What will you know by end of course

- Discuss basics of data modeling
- Explore how Apache Cassandra™ differs from relational world
- Explore KillrVideo domain
- Understand the building blocks of a good data model
- Improve our data model using a query driven methodology
  - Conceptual
  - Logical
  - Physical
- Optimize our data model
- Discuss a few data modeling use cases

DATASTAX ACADEMY

Data Modeling Overview

DATASTAX ACADEMY

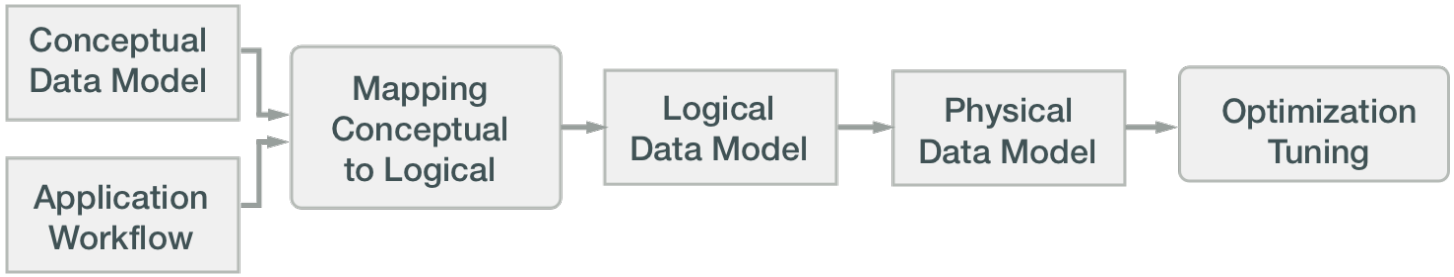
Data Modeling

What is Data Modeling?

- Analyze requirements of the domain
- Identify entities and relationships—Conceptual Data Model
- Identify queries—Workflow and Access Patterns
- Specify the schema—Logical Data Model
- Get something working with CQL—Physical Data Model
- Optimize and tune

Data Modeling Methodology

Learn it, live it, love it!



Data Modeling is…

a science!

- Apply tested methodologies
- Make improvements
- Reproducible

Data Modeling is…

also an art!

- Think outside of the box
- Non-standard solution—requires creativity
- Be careful—different data models have different costs

DATASTAX ACADEMY

Model for Queries

Queries are king in Apache Cassandra™

- If possible:
  - Understand your application's workflow
  - Understand the types of queries your application will support
  - Build tables around your queries (more on this later!)

DATASTAX ACADEMY

Relational vs. Apache Cassandra

Relational vs. Apache Cassandra™

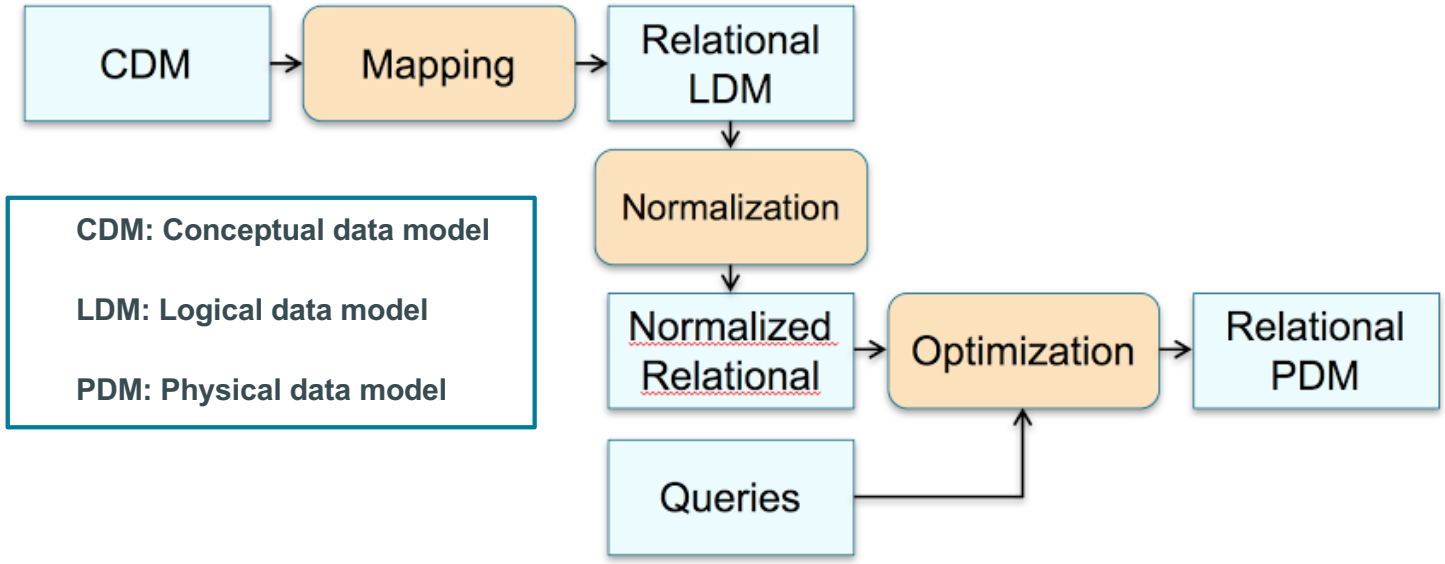
Simple differentiation overview

Relational Database	Apache Cassandra™
Sample relational model methodology	Cassandra modeling methodology
Data—Model—Application	Application—Model—Data
Entities are king	Queries are king
Primary key for uniqueness	Primary keys are much more
Often have single point of failure	Distributed architecture
ACID compliant	CAP theorem
Joins and indexes	Denormalization
Referential Integrity enforced	RI not enforced

Stay tuned! More details on upcoming slides…

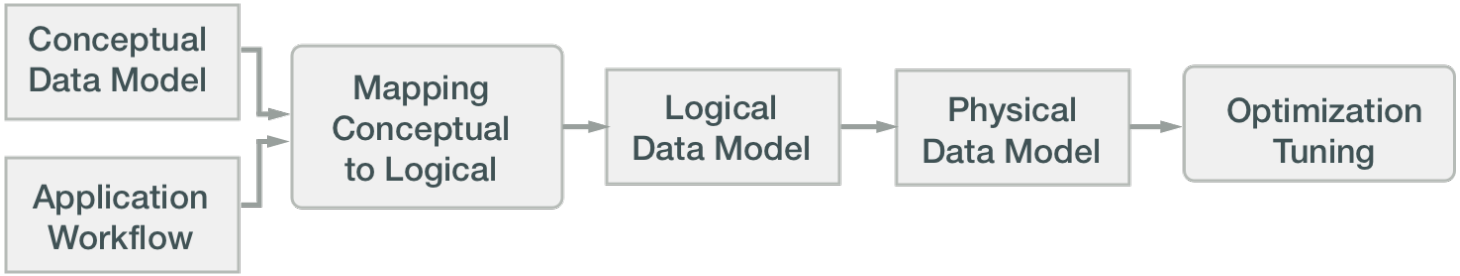
Relational Data Modeling Methodology

Sample Methodology—One of many!



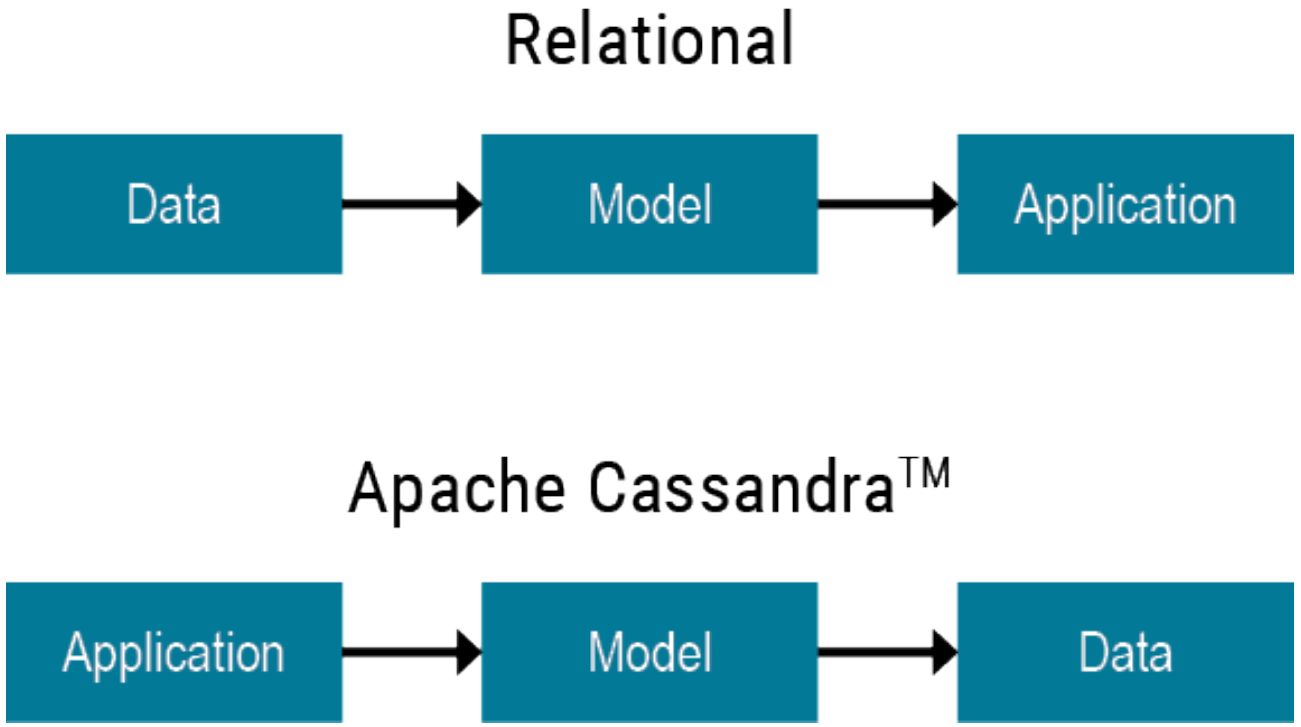
Cassandra Data Modeling Methodology

Our data modeling methodology—We will explore fully later!



Data Modeling Methodologies, Cont.

Relational vs. Apache Cassandra™



Transactions and ACID Compliance

Relational databases are ACID compliant

Term	Definition
Atomicity	All statements in a transaction succeed (commit), or none of them do (rollback)
Consistency	Transactions cannot leave the database in an inconsistent state. The new database state must satisfy integrity constraints
Isolation	Transactions do not interfere with each other
Durability	Completed transactions persist in the event of subsequent failure

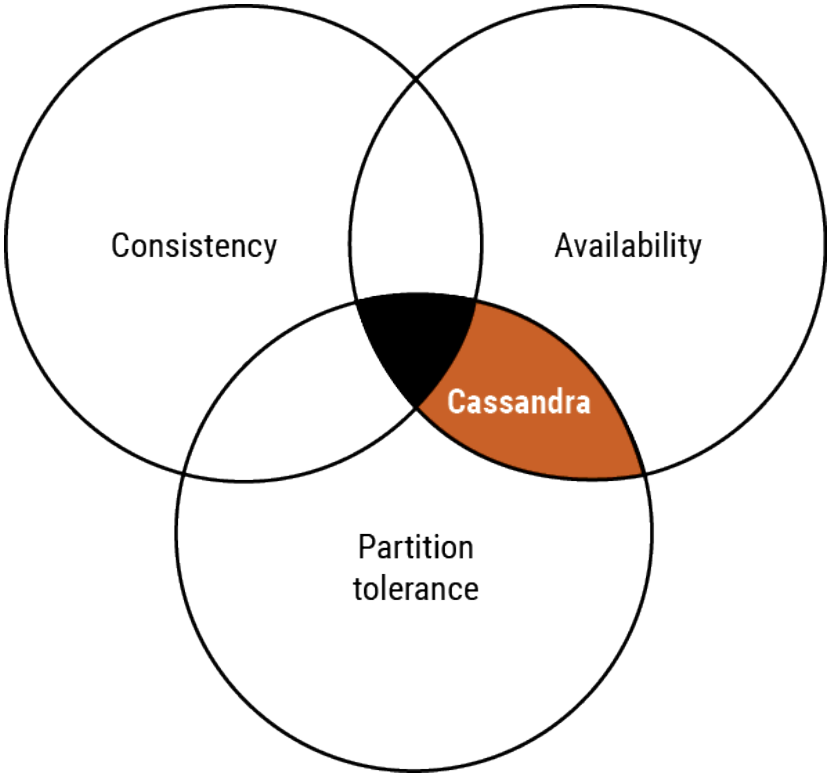
Transactions in Apache Cassandra™

Cassandra does not support ACID transactions

- ACID causes a significant performance penalty
- Not required for many use cases
- However, a single Cassandra write operation demonstrates ACID properties
  - INSERTs, UPDATEs, and DELETEs are atomic, isolated, and durable
  - Tunable consistency for data replicated to nodes, but does not handle application integrity constraints

Apache Cassandra™ and CAP Theorem

Consistency, Availability, and Partition Tolerance

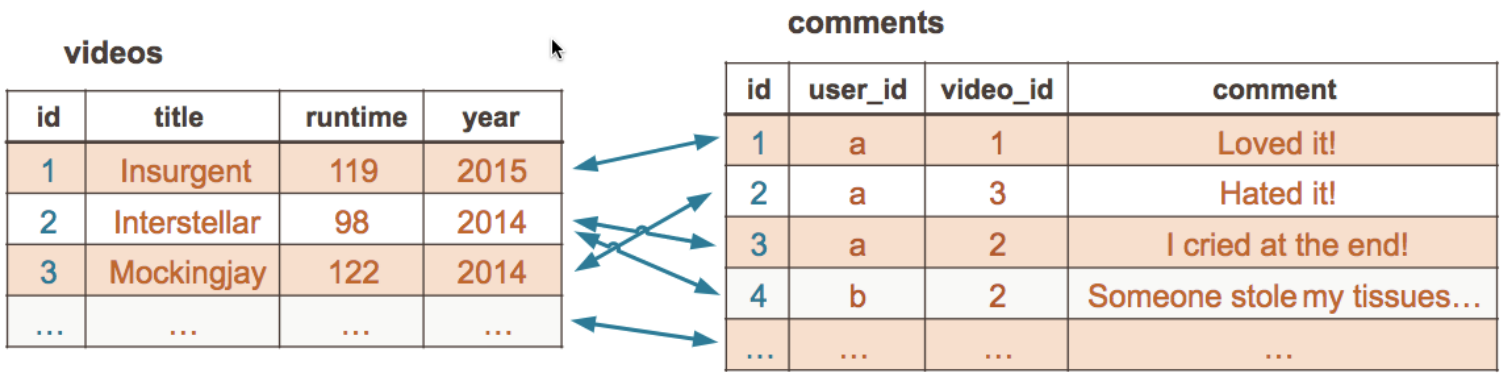


- By default, Cassandra is an AP database.
- However, this is tunable with consistency level.
- By tuning consistency level, you can make it more CP than AP.
- However! Cassandra isn't designed to be CA because you can't sacrifice partition tolerance.

Relational Joins

Joining users to comments tables

```
SELECT comment
FROM videos JOIN comments
  ON videos.id = comments.video_id
WHERE title = 'Interstellar'
```



videos

id	title	runtime	year
1	Insurgent	119	2015
2	Interstellar	98	2014
3	Mockingjay	122	2014
…	…	…	…

comments

id	user_id	video_id	comment
1	a	1	Loved it!
2	a	3	Hated it!
3	a	2	I cried at the end!
4	b	2	Someone stole my tissues…
…	…	…	…

Relational Joins, Cont.

Joining users to comments tables

```
SELECT comment
FROM videos JOIN comments
     ON videos.id =
comments.video_id
WHERE title = 'Interstellar'
```

videos JOIN comments

id	title	runtime	year	id	user_id	video_id	comment
1	Insurgent	119	2015	1	a	1	Loved it!
3	Mockingjay	122	2014	2	a	3	Hated it!
2	Interstellar	98	2014	3	a	2	I cried at the end!
2	Interstellar	98	2014	4	b	2	Someone stole my tissues…
…	…	…	…	…	…	…	…

Apache Cassandra™ and Denormalization

Cassandra doesn't support joins

```
CREATE TABLE comments_by_video (
      video_title text,
      comment_id timeuuid,
      user_id text,
      video_id timeuuid,
      comment text,
      PRIMARY KEY ((video_title),
comment_id)
);
```

comments_by_video

video_title	id	user_id	video_id	comment
Insurgent	1	a	1	Loved it!
Mockingjay	2	a	3	Hated it!
Interstellar	3	a	2	I cried at the end!
Interstellar	4	b	2	Someone stole my tissues…
…	…	…	…	…

```
CREATE TABLE comments_by_user (
      user_login text,
      comment_id timeuuid,
      user_id text,
      video_id timeuuid,
      comment text,
      PRIMARY KEY ((user_login),
comment_id)
);
```

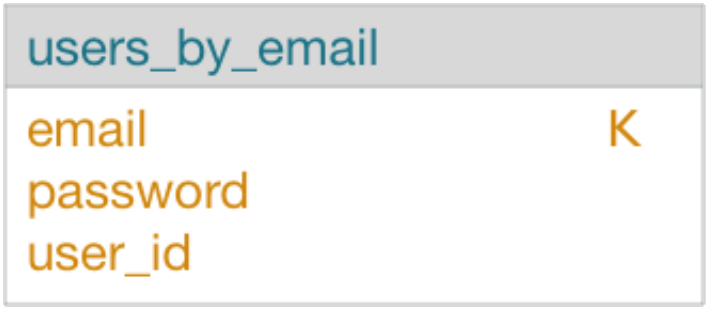
comments_by_user

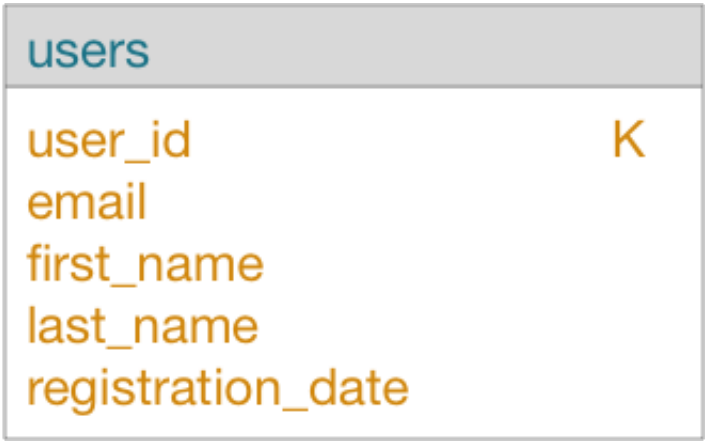
user_login	id	user_id	video_id	comment
emotions	1	a	1	Loved it!
emotions	2	a	3	Hated it!
emotions	3	a	2	I cried at the end!
clueless	4	b	2	Someone stole my tissues…
…	…	…	…	…

Referential Integrity—Relational

Joins rely on referential integrity constraints to combine data

- A value in one table requires the same value to exist in another table
- If there is a user in table `users_by_email`, then the user must also exist in table `users`





DATASTAX ACADEMY

Referential Integrity—Apache Cassandra™

Apache Cassandra™ does **NOT** enforce referential integrity

- Due to performance reasons—would require a read before a write
- Not an issue that has to be fixed on the Apache Cassandra™ side
- Referential integrity can be enforced in an application design—more work for developers

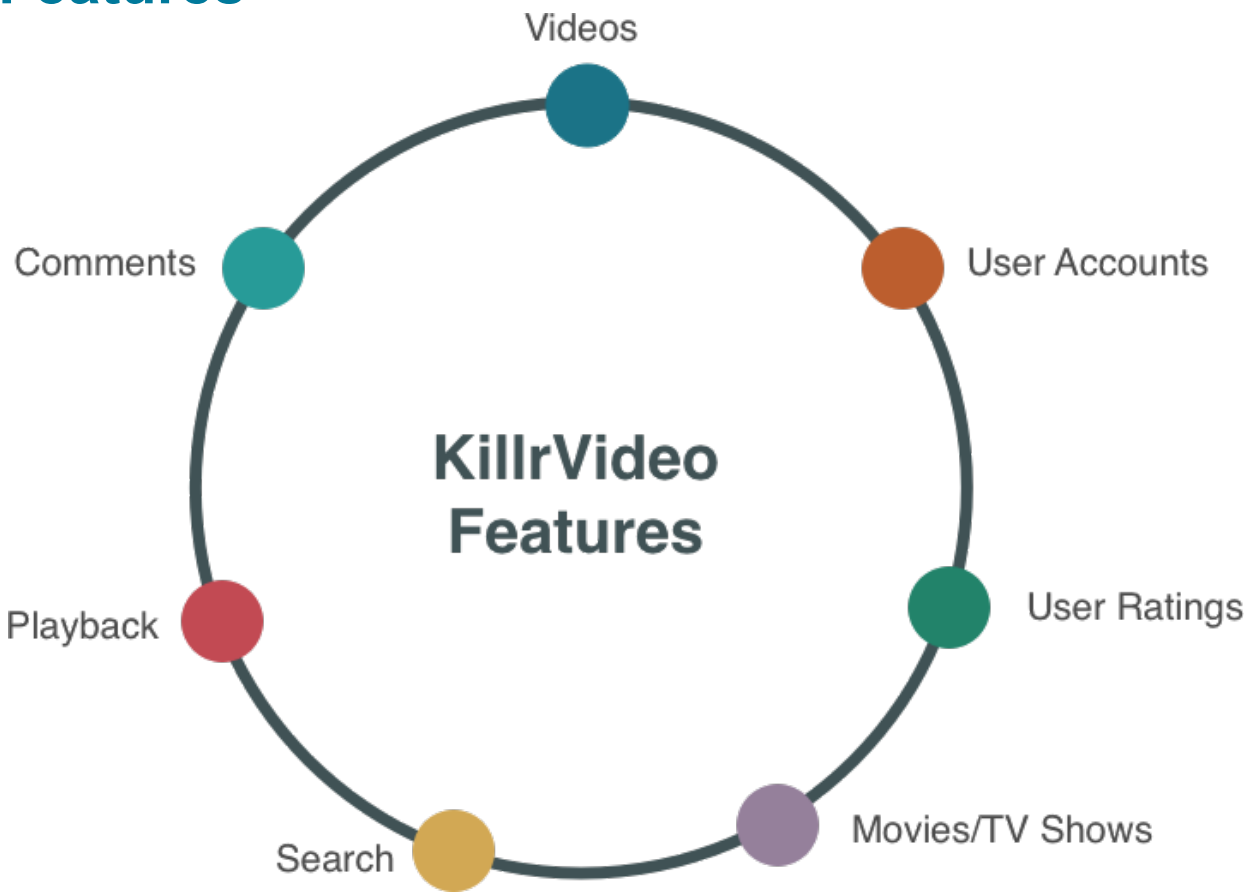
<OR>

- Run DSE Analytics with Apache SparkTM to validate that duplicate data is consistent

DATASTAX ACADEMY

Working with KillrVideo

KillrVideo Features



Problems KillrVideo Faces

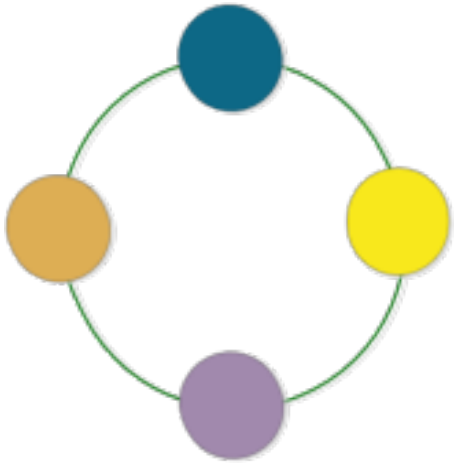
- **Scalability**—Must be able to support constant addition of new users and videos
- **Reliability**—Must always be available
- **Ease of use**—Must be easy to manage and maintain

Solutions Attempted—Relational

- Single points of failure
- Scaling complexity
- Reliability issues
- Difficult to serve users worldwide

KillrVideo and DataStax Enterprise

- Peers instead of master/slave
- Linear scale performance
- Always on reliability
- Data can be stored geographically close to clients



DATASTAX ACADEMY

Exercise 1.1: Setting up lab environment

DATASTAX ACADEMY

Enough CQL to get you started

DATASTAX ACADEMY

Basic CQL Concepts

The following slides will cover these concepts

- Keyspaces
- Tables
- Basic data types
- Importing CSV files
- SELECT
- TRUNCATE
- ALTER TABLE
- SOURCE

DATASTAX ACADEMY

Working with Keyspaces

Creating your container

- Top-level namespace/container
- Similar to a relational database schema

```
CREATE KEYSPACE killrvideo
  WITH REPLICATION = {
  'class': 'SimpleStrategy',
  'replication_factor' : 1
};
```

- Replication parameters required

USE

Accessing your keyspace

- USE switches between keyspaces

```
USE killrvideo;
```

DATASTAX ACADEMY

Tables

More on table functionality later—these are the basics

- Keyspaces contain tables and tables contain data

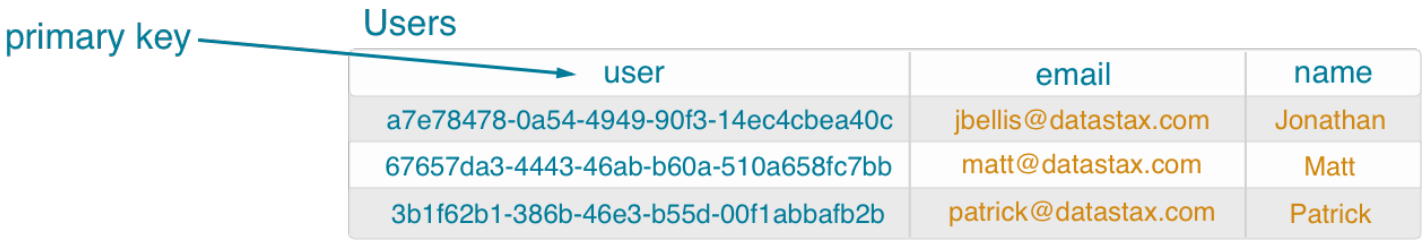
```
CREATE TABLE table1 (
   column1 TEXT,
   column2 TEXT,
   column3 INT,
   PRIMARY KEY (column1)
);

CREATE TABLE users (
   user_id UUID,
   first_name TEXT,
   last_name TEXT,
   PRIMARY KEY (user_id)
);
```

Primary Keys

Basic refresher! More details in next section

```
CREATE TABLE users (
  user uuid,
  email text,
  name text,
  PRIMARY KEY (user)
);
```

Uniquely identify rows



DATASTAX ACADEMY

Basic Data Types

Just to get you started! More on data types later.

- Text
  - UTF8 encoded string
  - varchar is same as text
- Int
  - Signed
  - 32 bits

UUID and TIMEUUID

- Universally Unique Identifier
  - Ex: 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328
  - Generate via uuid()
- TIMEUUID embeds a TIMESTAMP value
  - Ex: 1be43390-9fe4-11e3-8d05-425861b86ab6
  - Sortable
  - Generate via now()

TIMESTAMP

- Stores date and time
- 64-bit integer
- Milliseconds since January 1 1970 at 00:00:00 GMT
- Displayed in cqlsh as yyyy-mm-dd HH:mm:ssZ
- As literal in cqlsh is '1979-07-24 08:30:15'

DATASTAX ACADEMY

COPY

Works with CSV files

- imports/exports CSV (comma-separated values)

```
COPY table1 (column1, column2, column3) FROM 'table1data.csv';
```

- Header parameter skips the first line in the file

```
COPY table1 (column1, column2, column3) FROM'table1data.csv’
WITH HEADER=true;
```

DATASTAX ACADEMY

SELECT

Pulls data from a table

```
SELECT *
FROM table1;

SELECT column1, column2, column3
FROM table1;

SELECT COUNT(*)
FROM table1;

SELECT *
FROM table1
LIMIT 10;
```

TRUNCATE

Removing table data

- Deletes all data from a table immediately and irreversibly.
- Truncate sends a JMX command to all nodes to delete SSTables that hold the data
- If any of these nodes are down the command will fail

```
TRUNCATE table1;
```

ALTER TABLE

What can be altered?

- Can change datatype of column, add columns, drop columns, rename columns and change table properties
- BUT! You CANNOT alter the PRIMARY KEY columns

```
ALTER TABLE table1 ADD another_column
text;
ALTER TABLE table1 DROP another_column;
```

SOURCE

Executes CQL scripts

- Executes a file containing CQL statements
- Enclose file name in single quotes
- Output for each statement appears in turn
  - Example:

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

DATASTAX ACADEMY

Exercise 1.2: Basic CQL Fundamentals

