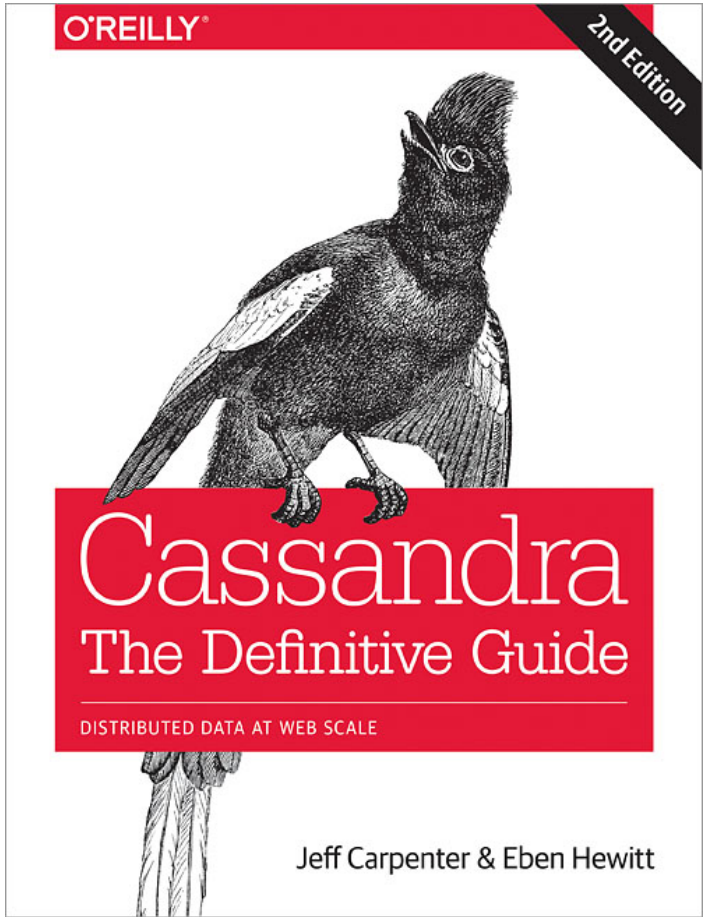
Building Applications with Apache Cassandra

A quickstart guide to Cassandra for Java developers and architects

Hoping You Came Prepared…

- Download Virtual Machine for Activities
  - If not, please take a minute now and start the download
  - See Slack/chat for URL
- Review Cassandra, The Definitive Guide, 2nd Ed
  - Chapter 1 – Beyond Relational Databases
  - Chapter 2 – Introducing Cassandra
  - Chapter 4 – The Cassandra Query Language (through “Cassandra’s Data Model”)



Goals

By the end of this live, online course, you'll understand:

- Cassandra's architecture and how it works
- How Cassandra is different from traditional databases, and how those differences affect data modeling and application development
- How to identify use cases where Cassandra is a good fit
- Cassandra's features, especially as available through the client drivers
- Commonly used configuration options

Goals

By the end of this live, online course, you'll be able to:

- Design Cassandra data models that will perform / scale effectively
- Create application programs using the DataStax Java Driver
- Use tools including *cqlsh*, *nodetool*, and CCM

Non-Goals

- Advanced configuration options (see Chapter 7)
- Monitoring beyond *nodetool* basics (see Chapter 10)
- Maintaining Cassandra clusters (see Chapter 11)
- Performance tuning (see chapter 12)
- Security beyond basic client authentication (see Chapter 13)
- Integrating Cassandra with other technologies (see chapter 14 and datastax.com)

Who am I?

- Developer
- Archiatect
- Author
- Developer Advocate

- Defense
- Hospitality
- R&D

- Distributed Systems
- Large Scale
- Cassandra

Contact Info

jeff.carpenter@datastax.com

@jscarp

jeffreyscarpenter

Blog: medium.com/@jscarp

Unit 1: Introduction, Install and *cqlsh*

~50 minutes

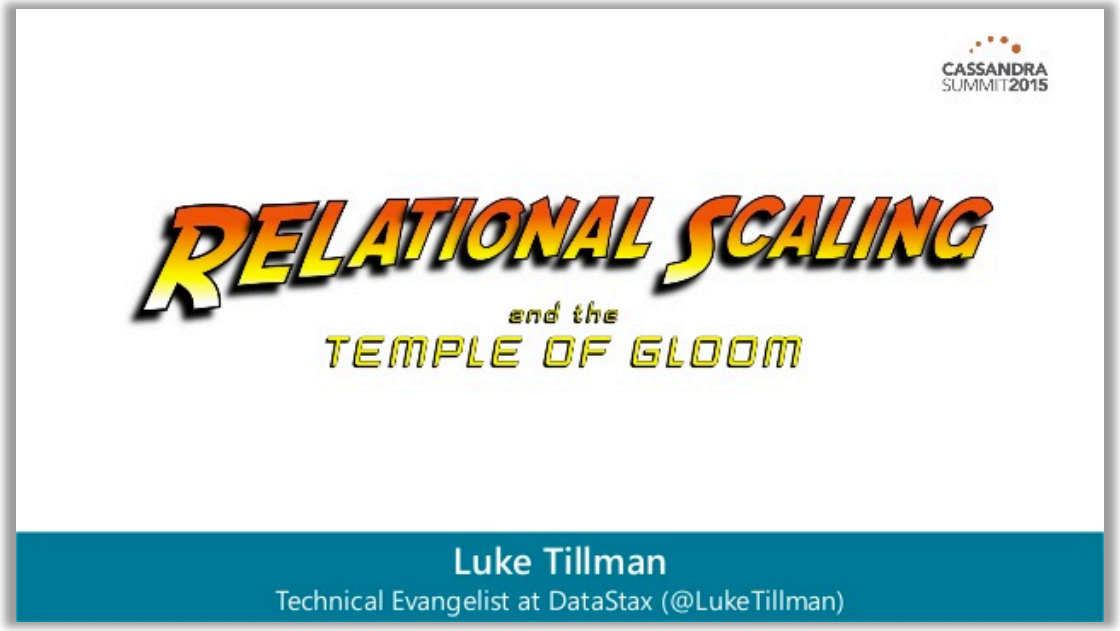
Unit 1 Goals

- Understand the challenges that come with scaling traditional RDBMS
- Articulate how Cassandra is different and the use cases for which it is a good fit
- Get a good ”hands on” experience with installing Cassandra and running basic commands in *cqlsh*

What's wrong with RDBMS?

- Pros
  - Relational data modeling is well understood
  - SQL is easy to use and ubiquitous
  - ACID transactions - data integrity
- Cons
  - Scaling is hard, sharding and replication have side-effects (performance, reliability, cost)
  - Trend toward denormalization

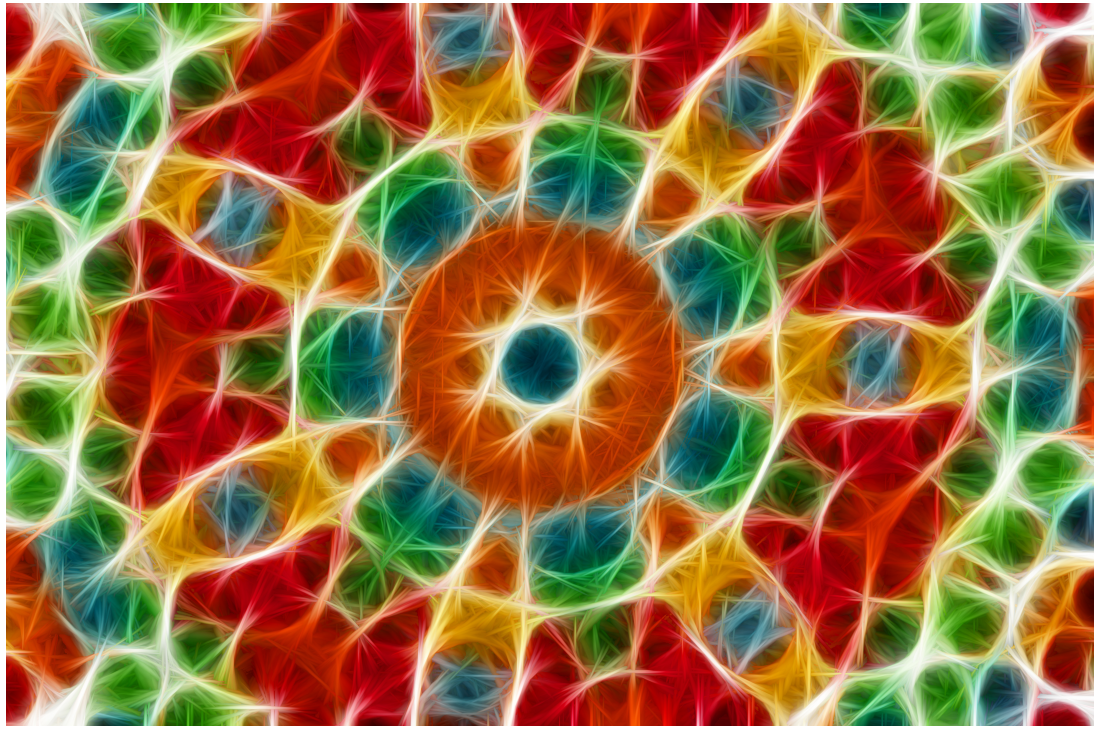
For fun: Relational Scaling and the Temple of Gloom



See also: Cassandra, The Definitive Guide, 2nd Edition, Chapter 1: Beyond Relational Databases

The NoSQL Revolution (~2009)

- Key-value − Dynamo, Riak, Voldemort, Redis, Memcached
- Column-oriented − BigTable, Hbase, Cassandra*
- Document − Mongo DB, Document DB, CouchDB
- Graph − Neo4J, DSE Graph
- Multi-model − DataStax Enterprise, CosmosDB
- Relational renaissance − AWS Aurora, Google Spanner



See also: How Many Databases?

Apache Cassandra at a Glance

- First developed by Facebook
- Became a top-level Apache Foundation project in 2010
- Distributed, decentralized
- Elastic scalability / high performance
- High availability / fault tolerant
- Tuneable consistency
- Partitioned row store

Apache Cassandra ® Apache Software Foundation

See also: Cassandra, The Definitive Guide, 2nd Edition, Chapter 2: Introducing Cassandra

Problems Cassandra is Especially Good At

- Large scale storage
  - >10s of TB
- Lots of writes
  - Time-series data, IoT
- Statistics and analytics
  - For example, as a Spark data source
- Geographic distribution
  - Multiple data centers

Discussion: Is Cassandra a Good fit for my project?

- What region are you from?
- What industry are you in?
- What are your use cases?

- Feel free to share in the chat (briefly!)

Exercise 1a - Installing and Starting Cassandra

- Goal:
  - Gain familiarity with the contents of the Apache Cassandra distribution
- Steps:
  - Start VM
  - Cassandra archive is located in the Downloads folder
  - Open a terminal and expand the archive in your home directory:

```
$ tar xvf Downloads/apache-cassandra-3.11.3-bin.tar.gz
```

  - Examine the directory

```
$ cd apache-cassandra-3.11.3
```

Exercise 1a - Installing and Starting Cassandra

- Examining the Installation:
  - Note the contents of the following directories, and then we'll discuss:
    - /bin
    - /conf
    - /doc
    - /lib
    - /tools
  - See if you can find the answers to these questions:
    - How do I start Cassandra?
    - Where can I find a list of what is fixed in this release?
    - Which config files are required to be present?
    - How would I learn about the Cassandra Query Language (CQL) syntax?
    - Where should I go to learn how to troubleshoot my installation?

Examining the Installation - Highlights

- /bin
  - Scripts to start/stop Cassandra, nodetool, utilities for maintaining SSTables
- /conf
  - Configuration files: cassandra.yaml,
- /doc
  - Quick reference documentation for Cassandra and CQL
- /lib
  - Executables for Cassandra and libraries on which it depends
- /tools
  - Tools for stress testing and analyzing SSTables

What's missing?

Exercise 1a - Installing and Starting Cassandra

- Questions and Answers
  - How do I start Cassandra?
    - /bin/cassandra
  - Where can I find a list of what is fixed in this release?
    - CHANGES.txt
  - Which config files are required to be present?
    - /conf/cassandra.yaml, logback.xml
  - How would I learn about the Cassandra Query Language (CQL) syntax?
    - /doc/cql3/CQL.html
  - Where should I go to learn how to troubleshoot my installation?
    - /doc/html/troubleshooting/index.html

Exercise 1a - Installing and Starting Cassandra

- Starting Cassandra:
  - In a terminal window, start Cassandra

```
$ bin/cassandra -f
```

  - This will cause Cassandra to run in the foreground so you can see its logs
  - Not a normal production setting, but useful for demonstration

Getting Cassandra

- Distributions
  - Apache Cassandra
  - DataStax Enterprise
- Local Installation
  - Manual
  - Build from source (Chapter 3)
  - Apt-get (Ubuntu)
  - Homebrew (Mac)
- Deployment Options
  - Docker images
  - Virtual Machines
  - Mesosphere DC/OS (Open source and DSE)
  - Cloud marketplaces (AWS, Azure, Google Cloud)
  - Managed services (DataStax Managed Cloud, Instaclustr)

Exercise 1b – Running *cqlsh*

- Goal:
  - Learn how to use *cqlsh* including commonly used commands
- Steps
  - In a terminal window, start *cqlsh*

```
$ bin/cqlsh
```

  - The shell automatically connects to localhost at default port (9042)
  - Use ”HELP” command will list available commands
  - For a list of sample commands to try out, see this file

```
~/cassandra-guide/resources/cqlsh_intro.cql
```

See also: https://github.com/jeffreyscarpenter/cassandra-guide

CQL Commands

- Data Definition (DDL)
  - CREATE KEYSPACE
  - ALTER KEYSPACE
  - DROP KEYSPACE
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
  - TRUNCATE
  - USE
- Data Manipulation (DML)
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - BATCH

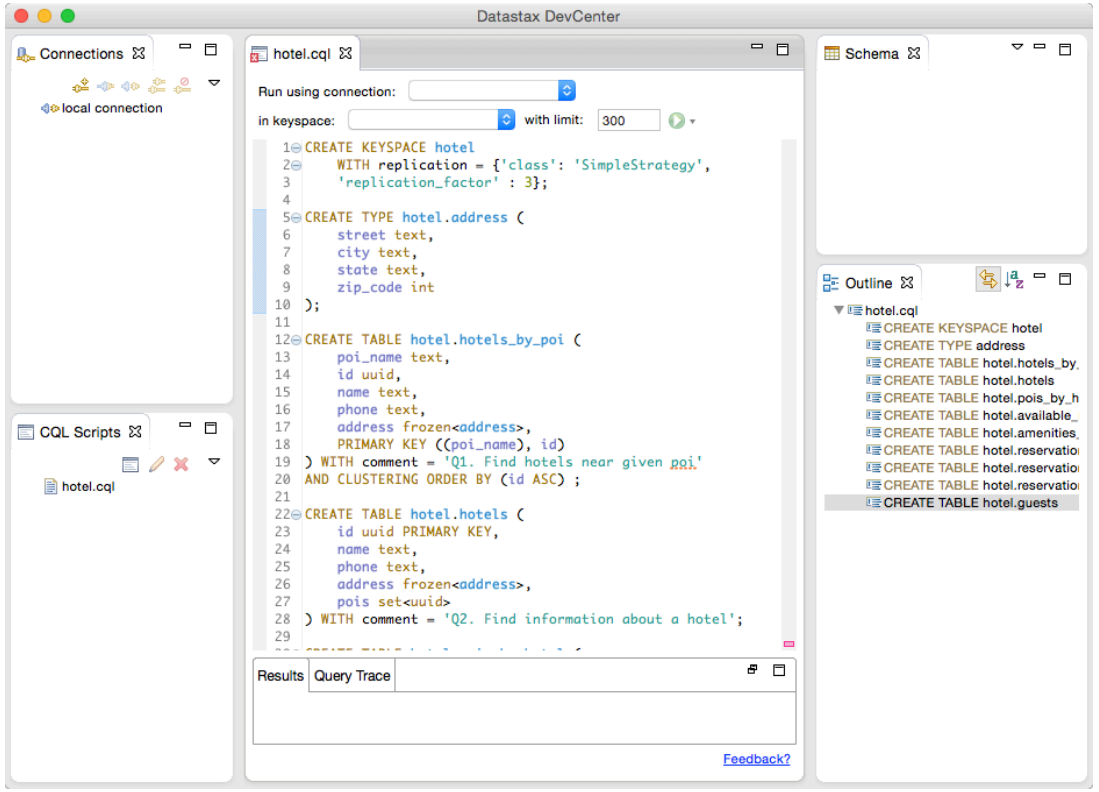
CQL Commands (continued)

- Indexes and Materialized Views
  - CREATE INDEX
  - DROP INDEX
  - CREATE MATERIALIZED VIEW
  - ALTER MATERIALIZED VIEW
  - DROP MATERIALIZED VIEW
- Security
  - CREATE USER
  - ALTER USER
  - DROP USER
  - CREATE ROLE
  - ALTER ROLE
  - DROP ROLE
  - GRANT
  - REVOKE
  - LIST PERMISSIONS
  - LIST ROLES
  - LIST USERS

Cqlsh-Only Commands

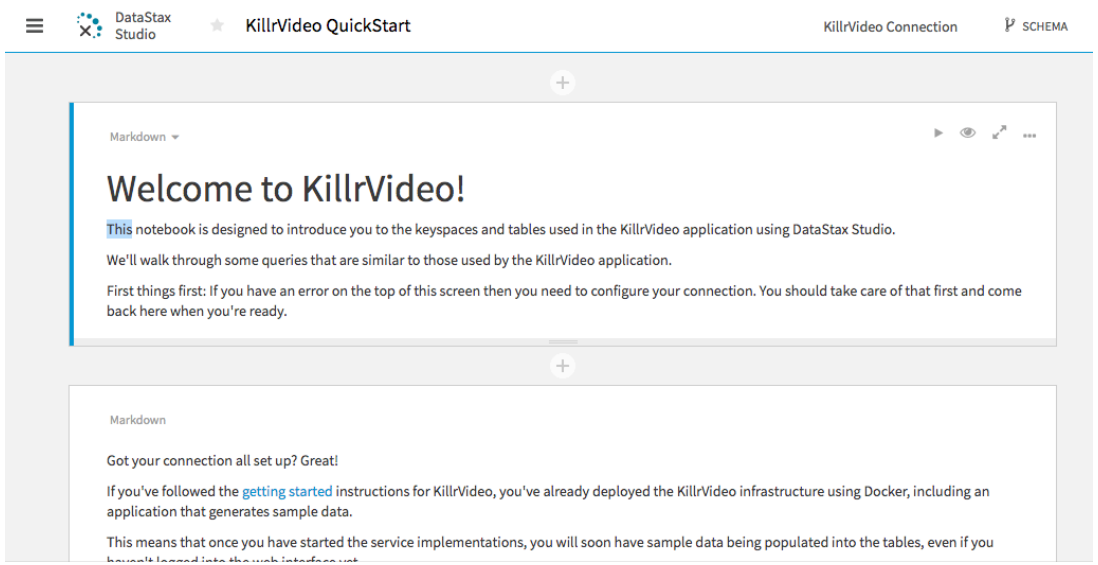
- Default settings
  - CONSISTENCY
  - SERIAL CONSISTENCY
  - PAGING
  - TRACING
  - EXPAND
- Metadata
  - DESCRIBE KEYSPACE
  - DESCRIBE TABLE
- File manipulation
  - COPY
  - SOURCE
  - CAPTURE
- Other commands
  - CLEAR
  - HELP
  - EXIT

DataStax DevCenter



- Graphical, Eclipse-style tool for editing schema and executing CQL
- Works with Apache Cassandra and DataStax Enterprise
- No longer supported, superseded by DataStax Studio

DataStax Studio



- Web application, notebook-style tool for editing schema and executing CQL
- Also supports Graph (Gremlin) schema and queries
- Works with DataStax Enterprise only

Break

5 minutes

Unit 2: Data Modeling

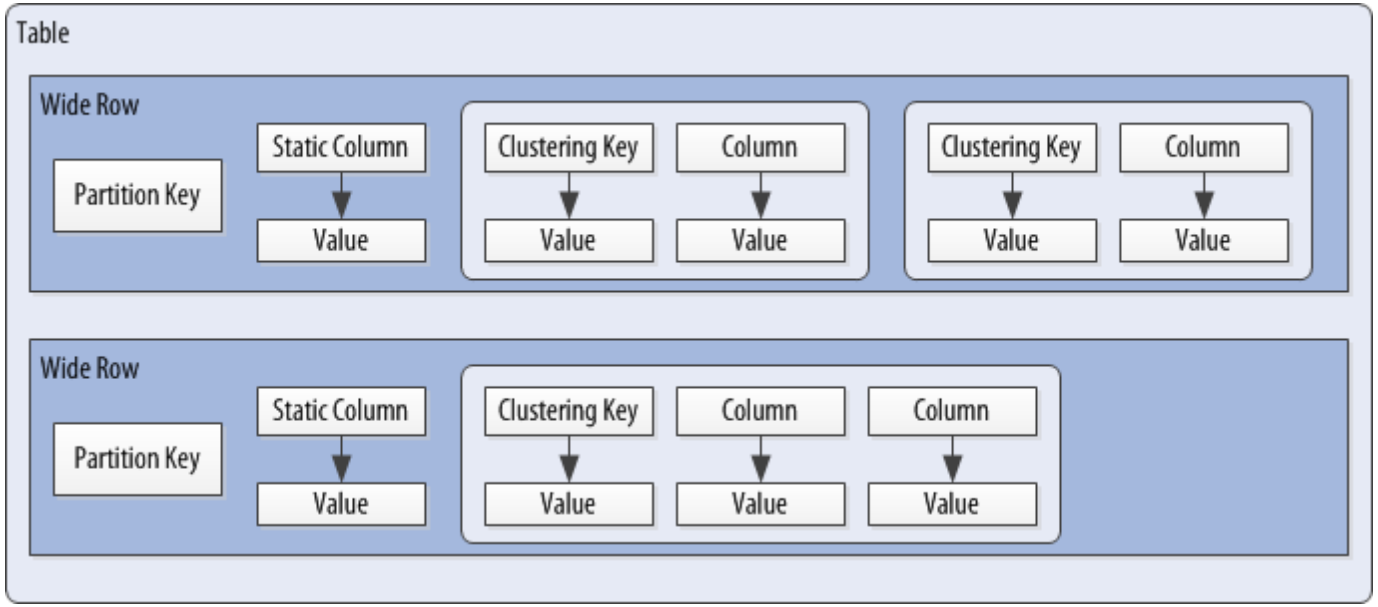
~55 minutes

Unit 2 Goals

- Familiarity with Cassandra's data model and terminology
- Learn what data types are built into Cassandra, how they map to Java types, and how you can extend Cassandra with your own data types
- Understand the basic principles of Cassandra data modeling and how it is different from traditional modeling
- Get practical experience designing data models for real applications using Cassandra

Terminology - Cassandra's Data Model

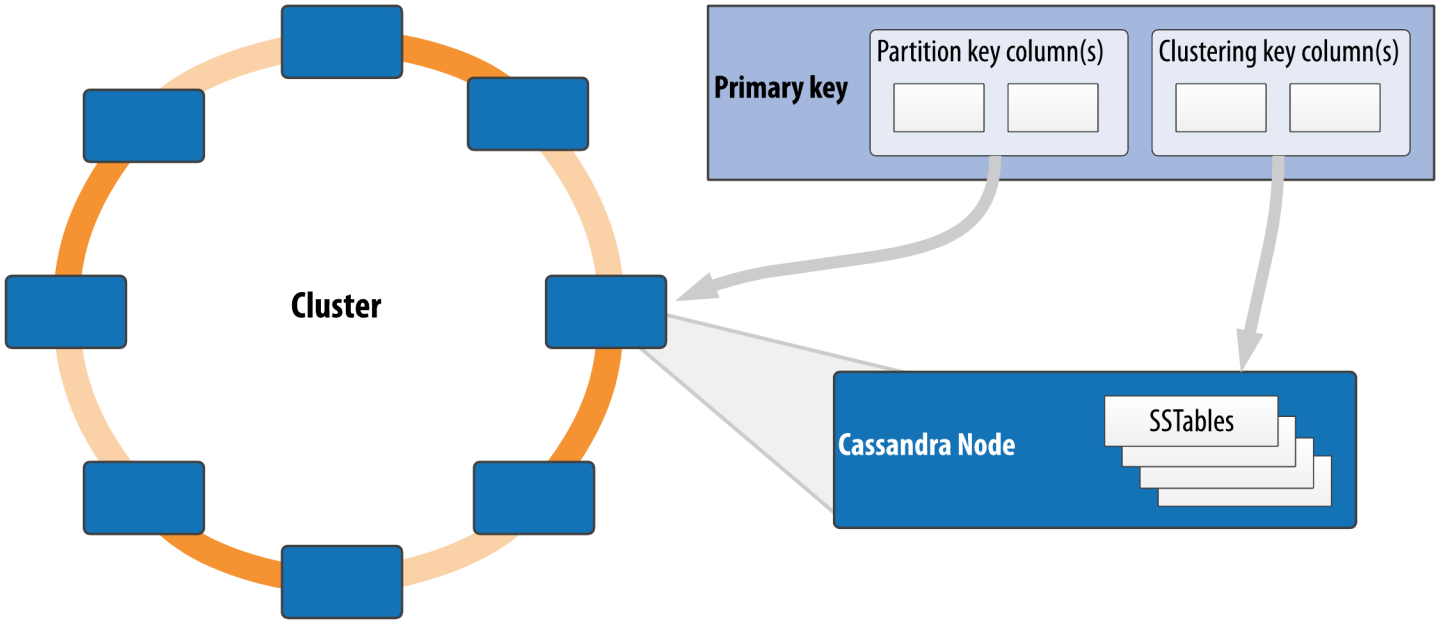
A Cassandra table



See also: Cassandra, The Definitive Guide, 2nd Edition, Chapter 4: The Cassandra Query Language

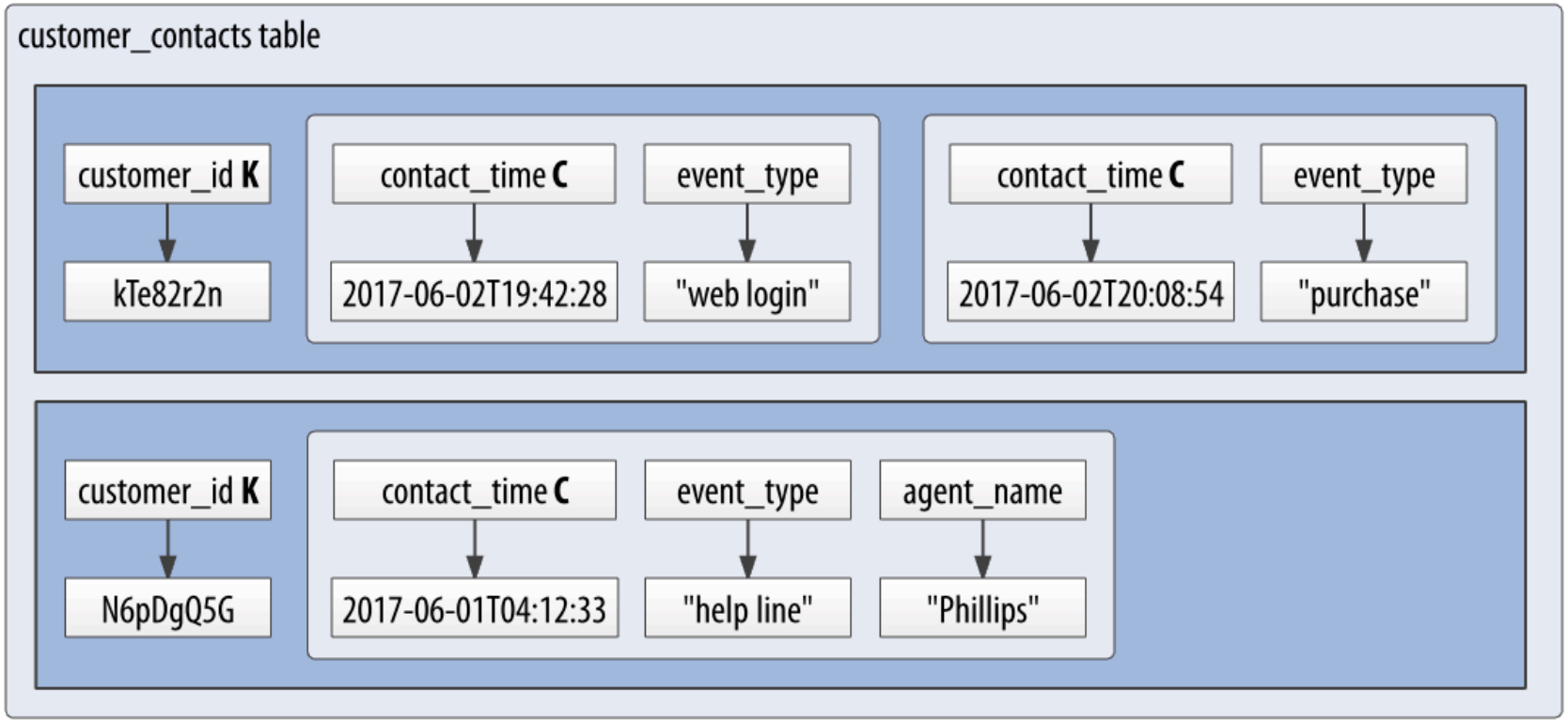
Terminology – Partition and Clustering Keys

PRIMARY KEY ((partition_key, pk_2), clustering_column1, cc2)



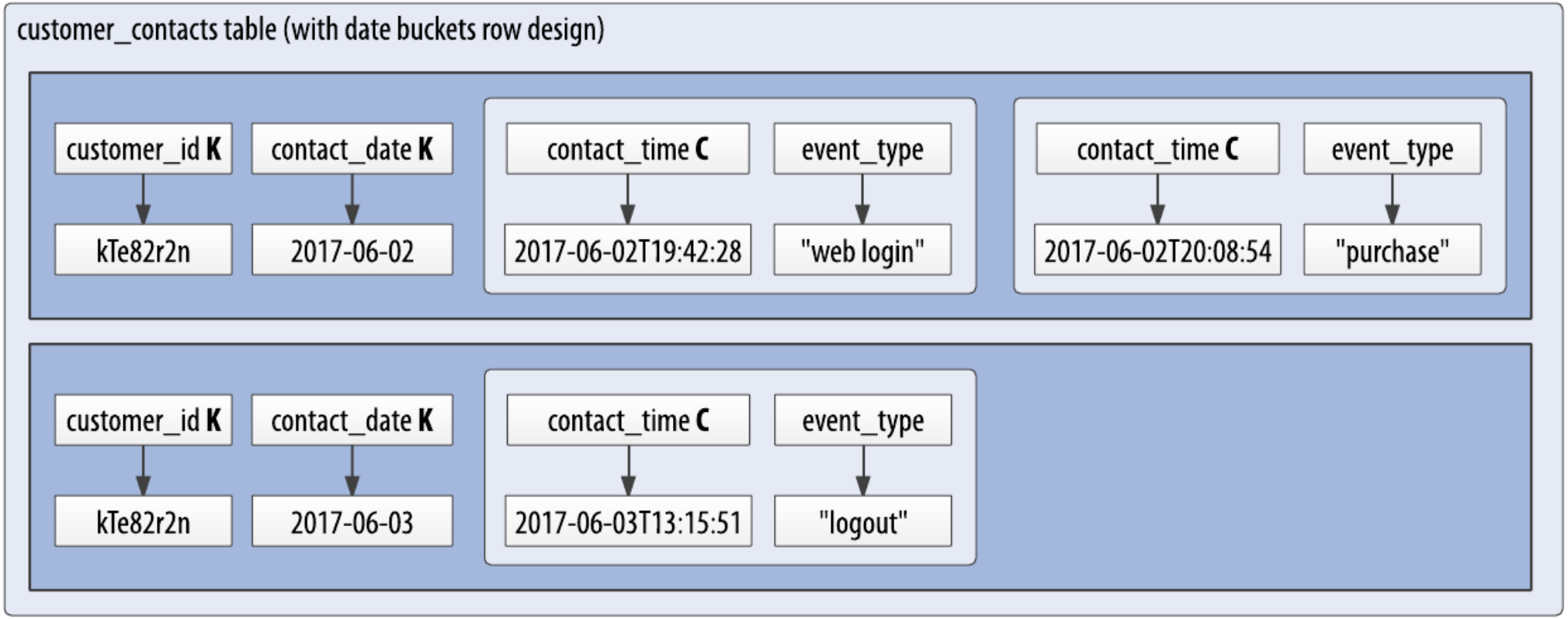
Partition and Clustering Key Example

Labeling:
K – partition key
C – clustering key



PRIMARY KEY ((customer_id), contact_time)

Buckets – a way to avoid overly wide rows



PRIMARY KEY ((customer_id, **contact_date**), contact_time)

Exercise 2a – Cassandra Data Types

- Goal:
  - Create and manipulate tables using *cqlsh*
  - Learn how to use Cassandra's built-in types and collections
- Steps
  - Make sure *cqlsh* is running
  - Work through the commands in the file

```
~/cassandra-guide/resources/learning_cql_data_types.cql
```

Cassandra Data Types

Numeric Data Types

CQL Type	Description	Java Type
int	32-bit signed integer	int
bigint	64-bit signed integer	long
smallint	16-bit signed integer	short
tinyint	8-bit signed integer	tinyint
varint	Variable-precision signed integer	java.math. BigInteger
float	32-bit floating point	float
double	64-bit floating point	double
decimal	Variable-precision decimal	java.math. BigDecimal

Text Data Types

CQL Type	Description	Java Type
text, varchar	UTF-8 character string	java.lang. String
ascii	ASCII character string	java.lang. String

See also:
http://docs.datastax.com/en/developer/java-driver/3.6/manual/

Cassandra Data Types

Time / Identity types

CQL Type	Description	Java Type
timestamp	ISO 8601 date/time, encoded as 64-bit int	java.util.Date
date	ISO 8601 date only	java.util. LocalDate
time	ISO 8601 time only	long
uuid	Type 4 Universal Unique Identifier	java.util.UUID
timeuuid	Type 1 UUID	java.util.UUID

Miscellaneous types

CQL Type	Description	Java Type
boolean	True/false value	boolean
blob	Arbitrary array of bytes	java.nio. ByteBuffer
inet	IP v4/v6 address	java.net. InetAddress
counter	64-bit integer which can only be incremented or decremented	long

Cassandra Data Types

Collections

CQL Type	Description	Java Type
set	Unordered collection of elements	java.util.Set

```
// set creation
ALTER TABLE user ADD emails set<text>
```

```
// populate set
UPDATE user SET emails = {
  'mary@example.com' } WHERE …

// add element
UPDATE user SET emails = emails + {
  'mary@example.com' } WHERE …

// Remove element
UPDATE user SET emails = emails - {
  'mary@example.com' } WHERE …
```

See also: http://docs.datastax.com/en/cql/3.3/cql/cql_reference/collection_type_r.html

Cassandra Data Types

Collections

CQL Type	Description	Java Type
list	Ordered collection of elements	java.util.List

```
// list creation
ALTER TABLE user ADD phone_numbers
list<text>
```

```
// populate list
UPDATE user SET phone_numbers = [
  '1-800-999-999' ] WHERE …

// append element
UPDATE user SET emails = phone_numbers + {
  '480-441-6957' } WHERE …

// set item
UPDATE user SET emails[0] = '480-111-1111'
  WHERE …
```

See also: http://docs.datastax.com/en/cql/3.3/cql/cql_reference/collection_type_r.html

Cassandra Data Types

Collections

CQL Type	Description	Java Type
map	Collection of key/value pairs	java.util.Map

```
// map creation
ALTER TABLE user ADD login_sessions
map<timeuuid, int>
```

```
// populate map
UPDATE user SET login_sessions = {
  6061b850…, 1 } WHERE …

// add element
UPDATE user SET login_sessions =
  login_sessions + { a9fac1d0…, 2} WHERE …

// Remove element
UPDATE user SET login_sessions =
  login_sessions - { a9fac1d0… } WHERE …
```

See also: http://docs.datastax.com/en/cql/3.3/cql/cql_reference/collection_type_r.html

Cassandra Data Types

Tuple

CQL Type	Description	Java Type
tuple	Fixed structure of other data types	com.datastax.driver.core.TupleType

```
// tuple creation
ALTER TABLE user ADD address
<tuple<text, text, text, text, text>>;
```

```
// populate tuple
UPDATE user SET address = (
  '7712 E Broadway', // street address
  'Tucson',          // city
  'AZ',              // state or province
  '85715',           // postal code
  'USA'              // country
) WHERE …
```

See also: http://docs.datastax.com/en/cql/3.3/cql/cql_reference/tupleType.html

User Defined Types

- Scoped to keyspace
- Can be nested
- Similar syntax to defining table

```
CREATE TYPE address (
    street text,
    city text,
    state_or_province text,
    postal_code text,
    country text
);
```

```
ALTER TABLE user ADD address address;

UPDATE user SET address =
  street: '7712 E Broadway',
  city: 'Tucson',
  state_or_providence: 'AZ',
  postal_code: '85715',
  country: 'USA'
) WHERE …
```

Exercise 2b - Query Options

- Goal:
  - Learn about the various clauses available in the CQL SELECT statement (WHERE, IN, SORT)
- Steps
  - Make sure *cqlsh* is running
  - Work through the commands in the file

```
~/cassandra-guide/resources/learning_cql_query_clauses.cql
```

JSON Input / Output

- Convenient way to insert/ retrieve data
- Can save processing JSON input in your application
- Not ”schemaless”!

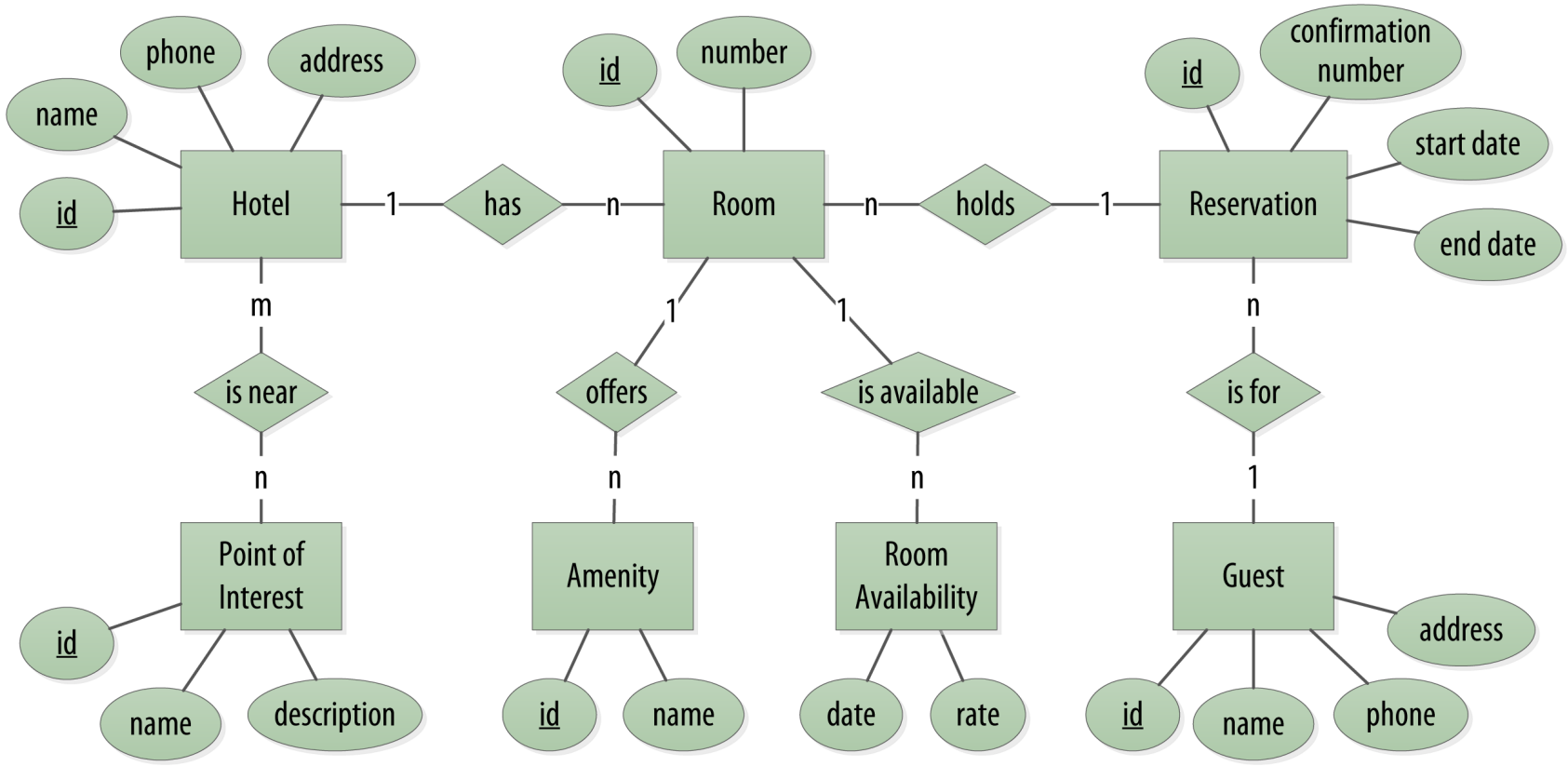
```
INSERT INTO user JSON '{"first_name": "Bill",
"last_name": "Nguyen",: "title": "Mr."}';


SELECT JSON * FROM user;
 [json]
----------------------------------------------
 {"first_name": "Mary", "last_name":
"Rodriguez",: "title": null}
 {"first_name": "Bill", "last_name":
"Nguyen",: "title": "Mr."}
```

Cassandra Data Modeling

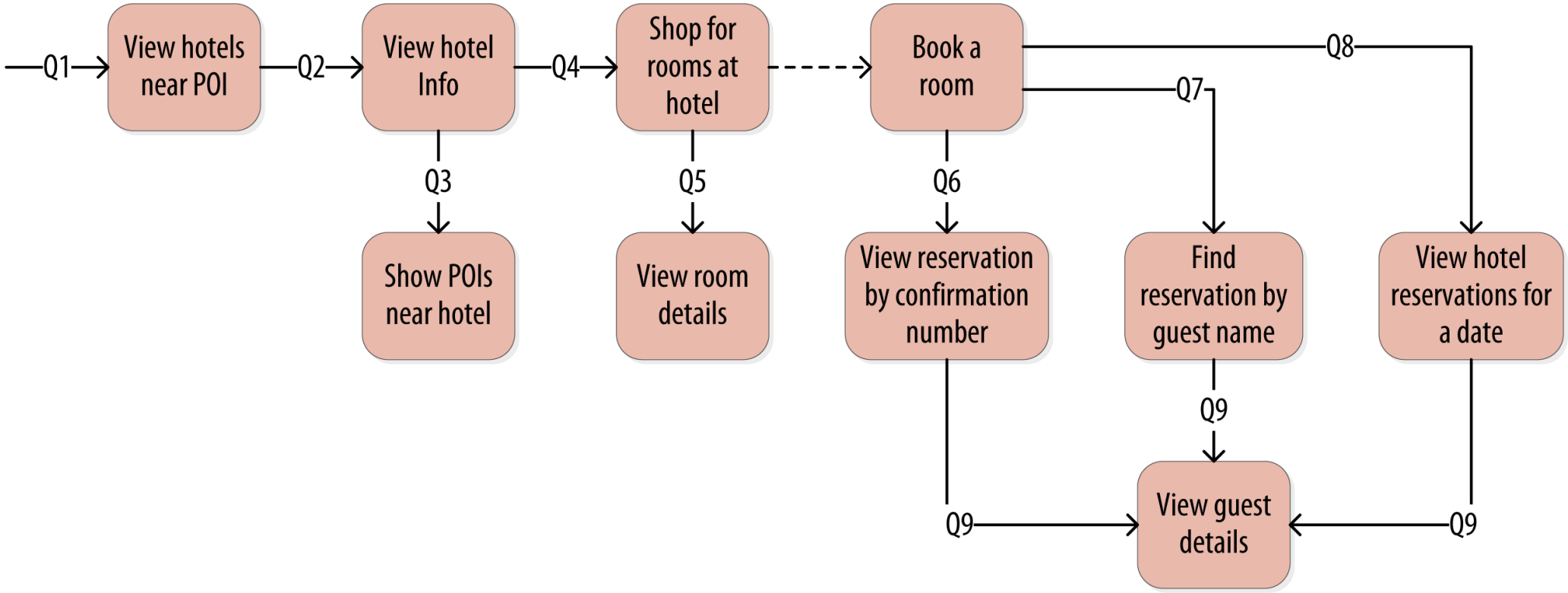
- Classic data modeling approach
  - Conceptual
  - Logical
  - Physical
- Additional Cassandra data modeling steps
  - Identifying access patterns
  - Evaluating data models

See also: Cassandra, The Definitive Guide, 2nd Edition, Chapter 5: Data Modeling

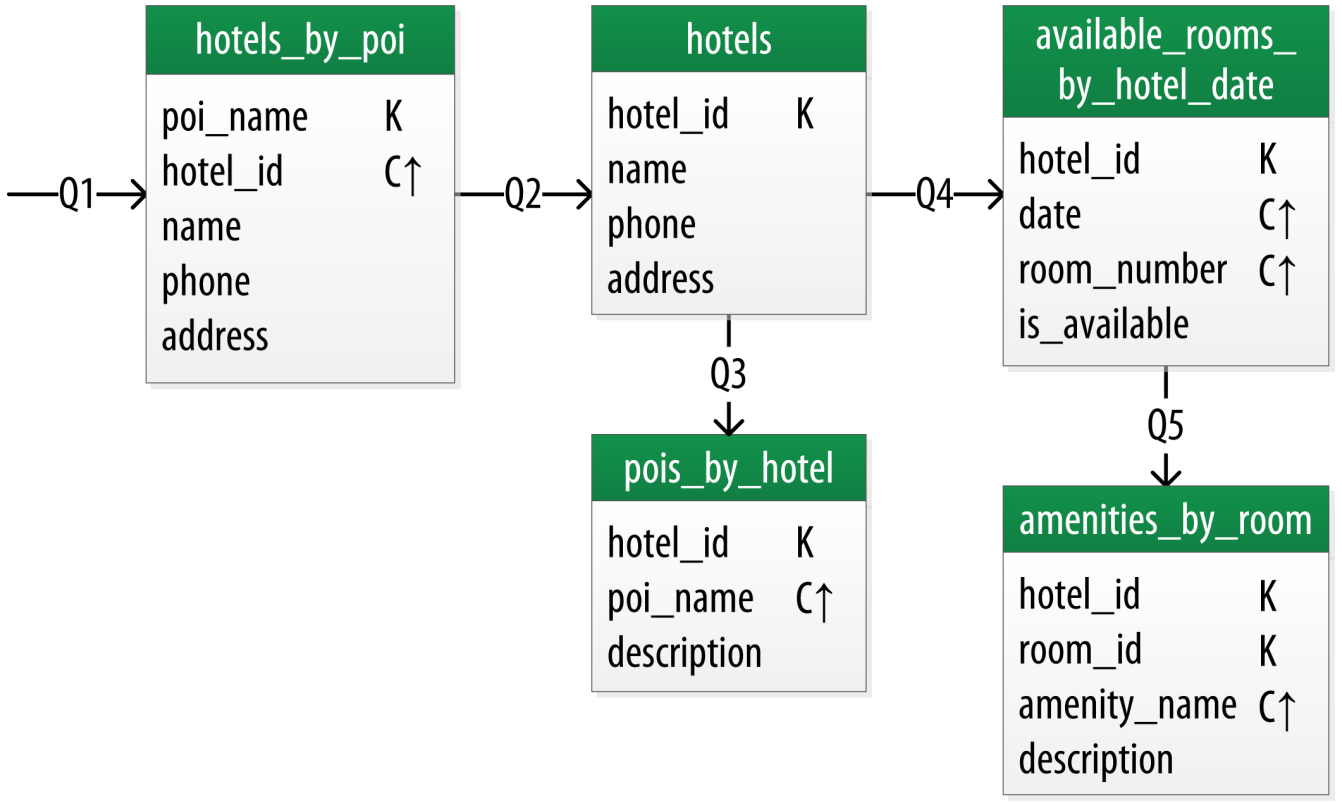
Conceptual Data Model – Hotel Domain



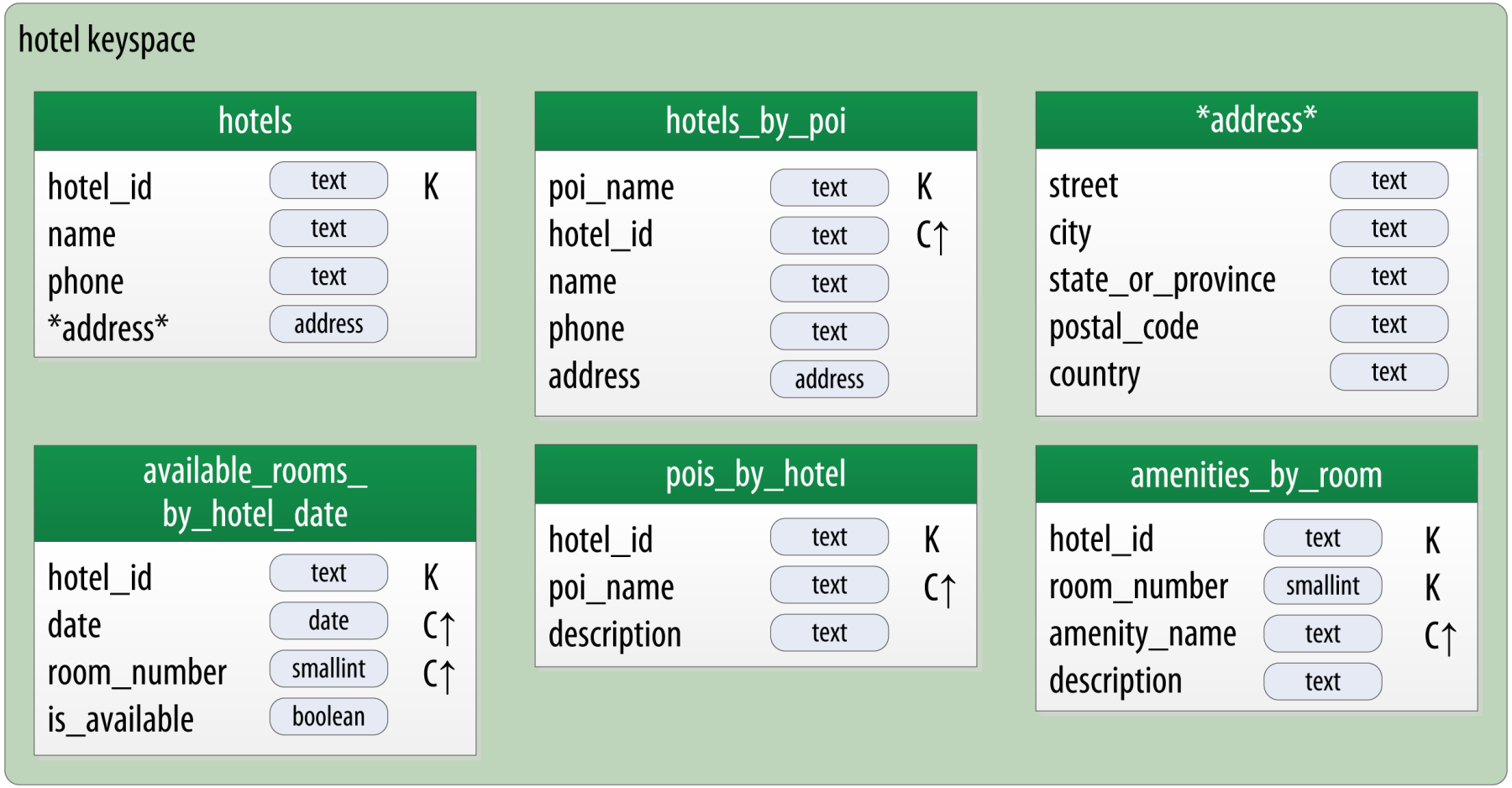
Access Patterns – Hotel Domain



Logical Data Modeling – Chebotko Diagram



Physical Data Model – Chebotko Diagram



CQL Schemas

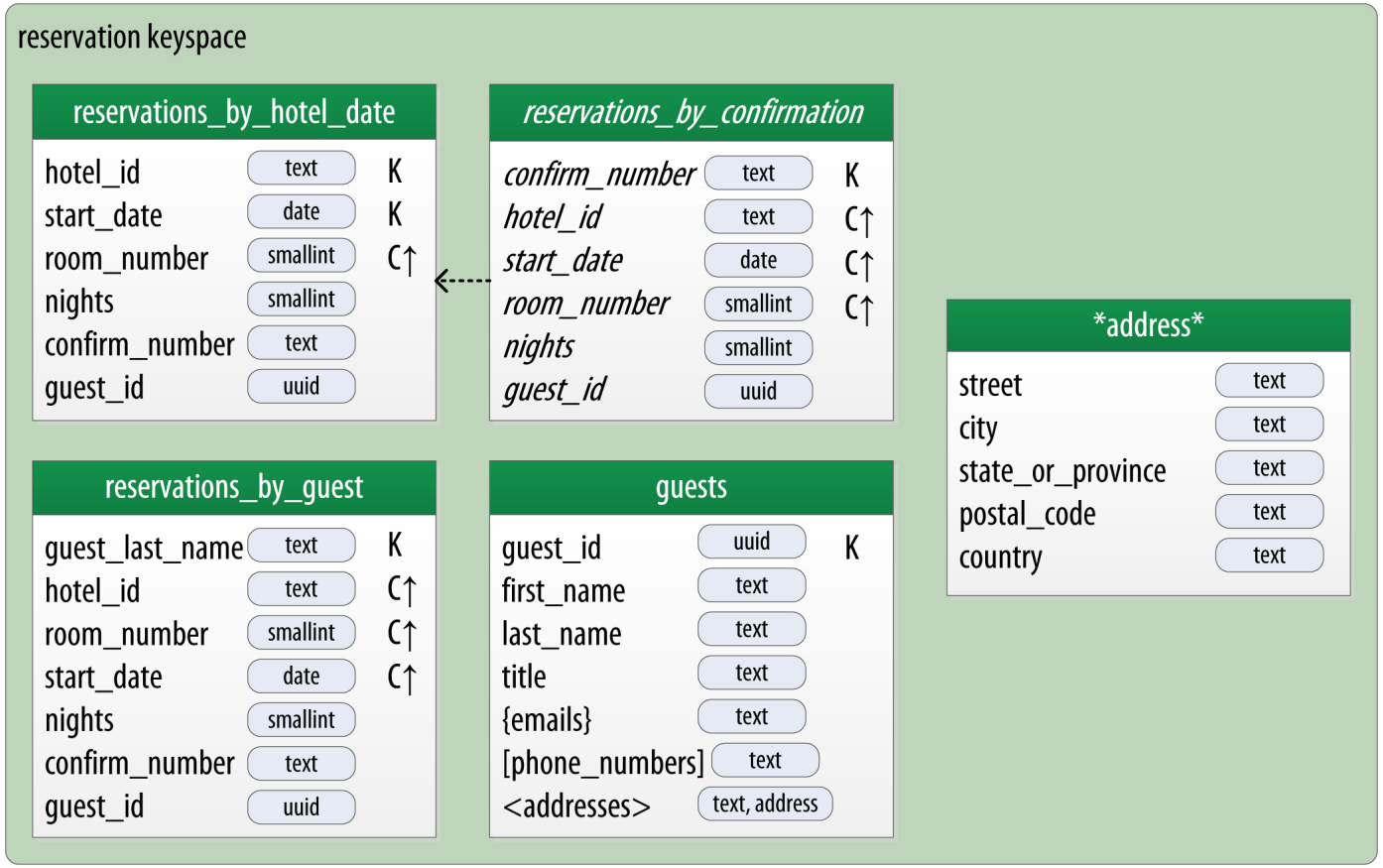
```
CREATE KEYSPACE hotel
  WITH replication = {'class':
  'SimpleStrategy',
  'replication_factor' : 3};

CREATE TYPE hotel.address (
  street text,
  city text,
  state_or_province text,
  postal_code text,
  country text);
```

```
CREATE TABLE hotel.hotels_by_poi (
  poi_name text,
  hotel_id text,
  name text,
  phone text,
  address frozen<address>,
  PRIMARY KEY ((poi_name),
    hotel_id)
)
WITH CLUSTERING ORDER BY (
  hotel_id ASC) ;
```

Exercise 2c - Data Modeling

- Goal:
  - Create a schema for hotel reservations based on the design provided
- Steps
  - Create a keyspace for reservations using the SimpleReplication strategy and a replication factor of 1
  - Define a UDT for the Address data type
  - Define tables for reservations_by_hotel_date and reservations_by_guest
  - Create reservations_by_confirmation as a materialized view on reservations_by_hotel_guest

Exercise 2c - Data Modeling



Data Modeling Challenges

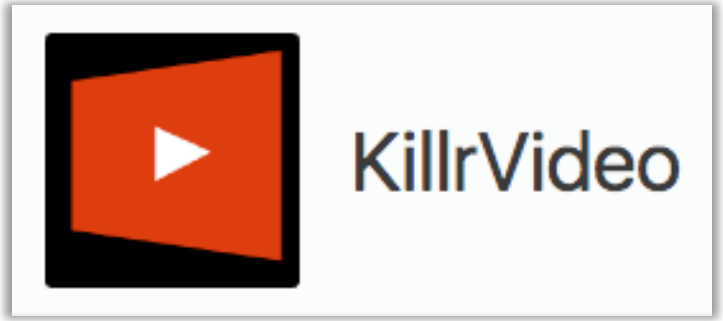
- What were some of the struggles you encountered with creating the data model (not tool related)?

- If you did have any tool challenges, record them in the chat and I will follow up later on.

More Cassandra Data Modeling Resources

Training

Free online training at DataStax Academy



Reference Application

See a real data model in this working reference application

Reading

Check out my blog on advanced data modeling: “Data Model Meets World”

See also: Cassandra, The Definitive Guide, 2nd Edition, Chapter 5: Data Modeling

Break

5 minutes

Unit 3: Client Development

~60 minutes

Unit 3 Goals

- Learn about the various language drivers available for writing Cassandra client applications and the common features they share
- Get “hands on” experience writing, configuring, and running a Java application using the DataStax Java Driver
- Understand the various methods of executing statements supported by the driver and how to use the mapper

Client Drivers

- Historic Drivers Developed by various community projects, mostly based on Thrift and now deprecated
  - Hector (Java), Astyanax (Java, Netflix), Pycassa (Python), Perlcassa (Perl), Helenus (node.js), Cassandra-Sharp (C#/.NET)
- DataStax Drivers
  - Based on CQL
  - Open Source - https://github.com/datastax

See also: Cassandra, The Definitive Guide, 2nd Edition, Chapter 8: Clients

Client Drivers

Apache Cassandra Drivers (Open Source)	DataStax Drivers (DataStax Enterprise)
DataStax Java Driver	DataStax Enterprise Java Driver
DataStax Python Driver	DataStax Enterprise Python Driver
DataStax Node.js Driver	DataStax Enterprise Node.js Driver
DataStax Ruby Driver	DataStax Enterprise Ruby Driver
DataStax C# Driver	DataStax Enterprise C# Driver
DataStax C/C++ Driver	DataStax Enterprise C/C++ Driver
DataStax PHP Driver	DataStax Enterprise PHP Driver

https://academy.datastax.com/downloads/download-drivers

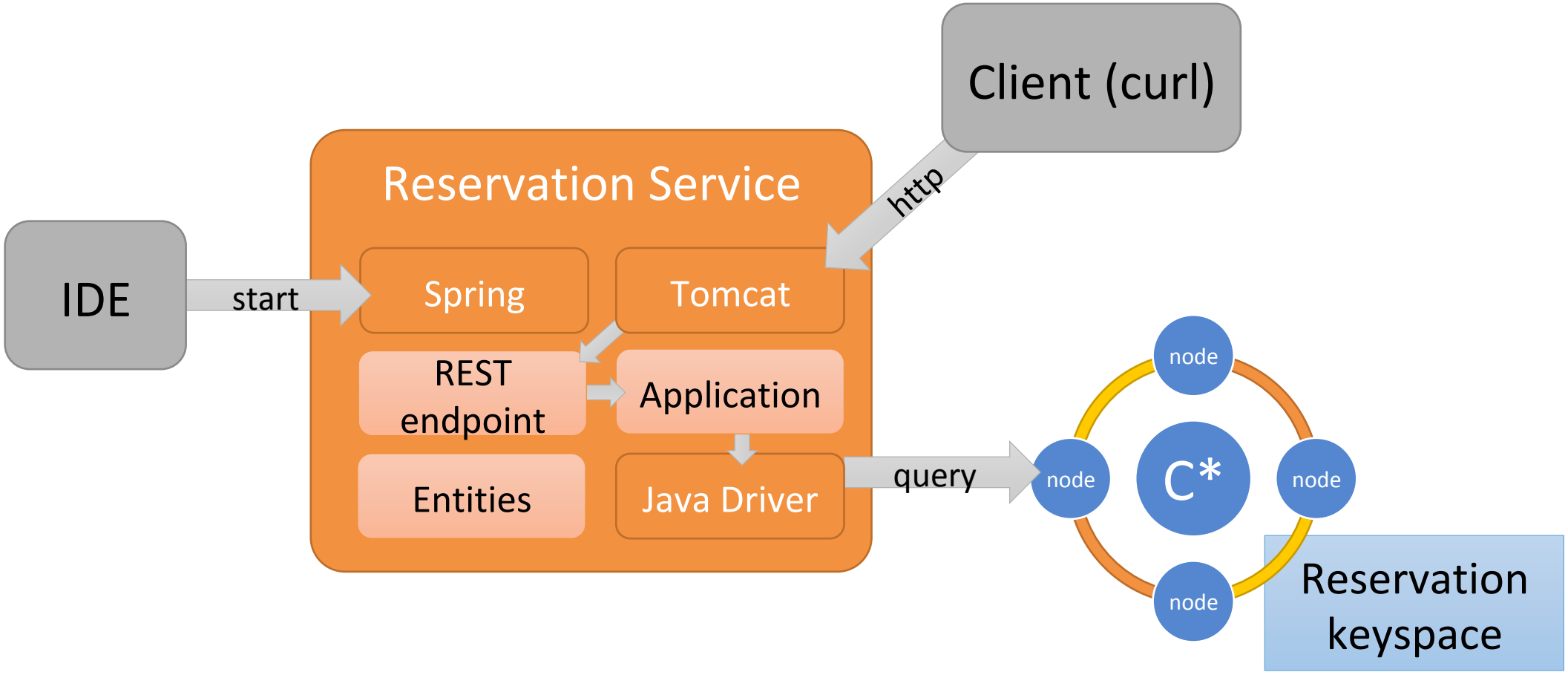
DataStax Cassandra Drivers

- Common features:
  - Connection management
  - Creating and executing statements, and accessing the results
  - Synchronous and asynchronous execution
  - Object mapping
  - Logging and metrics
  - Policy management
  - Threading, networking and resource management
  - Schema access / management

Our Project: Reservation Service

- Goals
  - Create a microservice implementation that persists data using Cassandra
  - Leverage best practices
  - Complete application except for the Cassandra bits
- Choices
  - Java
  - IntelliJ IDEA Community Edition
  - Spring Boot
  - RESTful API

Reservation Service Architecture



Exercise 3a – Reservation Service Setup

- Goal:
  - Set up the reservation service in our dev environment
- Steps
  - The sample application code has already been cloned from GitHub

```
$ git clone git://github.com/jeffreyscarpenter/reservation-service.git
-or-
$ cd reservation-service; git pull
```

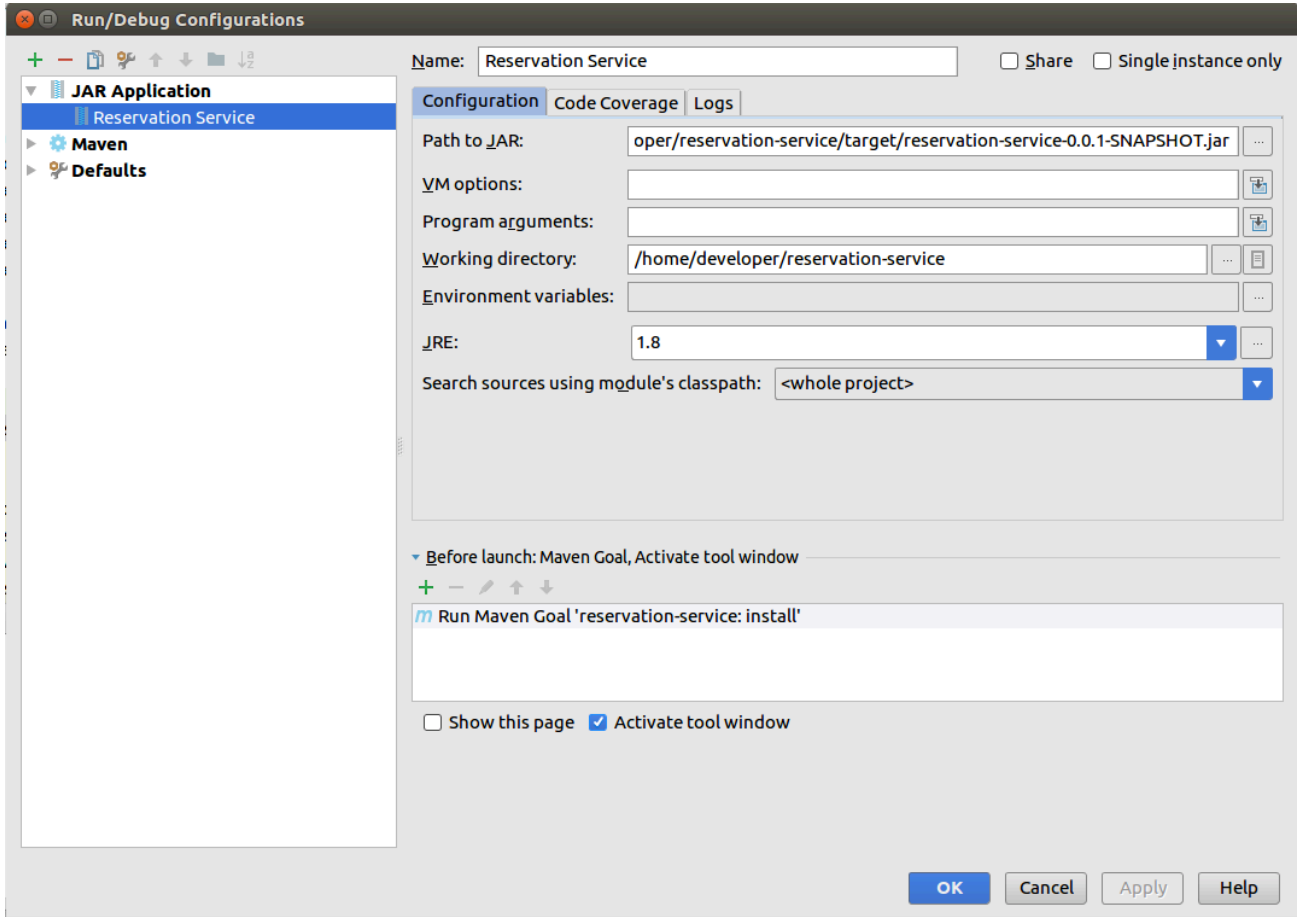
  - Run *cqlsh* and create schema

```
cqlsh> SOURCE '~/reservation-service/src/main/resources/reservation.cql'
```

Exercise 3a – Reservation Service Setup

- Instructions
  - Open IntelliJ, and select “Reservation Service” from recent projects
    - For reference: the project was created by “Select to create a new project from existing source” and configuring the project as a Maven project on import
  - Run the application
    - IntelliJ Ultimate: Run -> Run ReservationServiceApplication
    - IntelliJ Community Edition: Run -> Run Reservation Service
      - custom run config – see next slide
  - Eclipse is available on the VM but I recommend using IntelliJ
    - That’s how I set up the instructions
    - Better syntax highlighting
    - Better Spring Boot integration

Exercise 3a – Reservation Service Setup

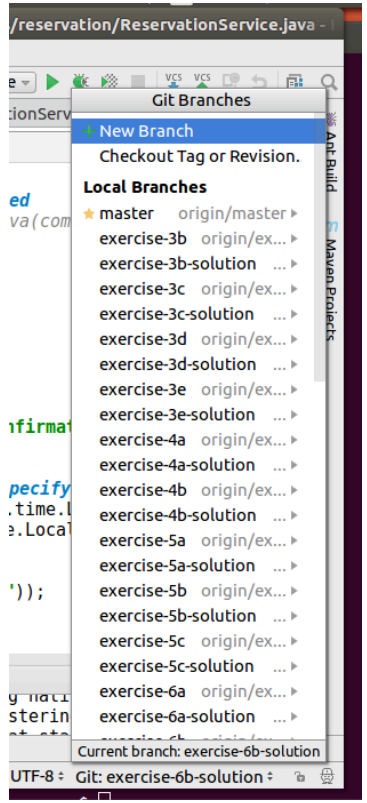


(This is just FYI, I've created this run configuration for you since we're using IntelliJ Community Edition, you don't have to do this in Ultimate)

Exercise 3a – Reservation Service Setup

- Examining the code:
  - Note the contents of the following files and directories, and then we'll discuss:
    - README.md
    - pom.mvn
    - /src
    - /resources
  - See if you can find the answers to these questions:
    - How do you access code samples for the various exercises?
    - What CQL type do we use to represent the start date and time of the reservation?
    - What Java type do the start date and time of the reservation get mapped to?

Exercise 3a – Reservation Service Setup



- Questions and Answers
  - How do you access code samples for the various exercises?
    - The start and end point for each exercise are tagged in Git
      - git checkout exercise-3b
      - git checkout exercise-3b-solution
    - You can also change branch you are looking at in IntelliJ IDEA (see picture)
  - What CQL type do we use to represent the start date and time of the reservation?
    - date (not timestamp)
  - What Java type do the start date and time of the reservation get mapped to?
    - java.time.LocalDate – I've chosen to use this instead of java.util.Date
    - This will be easier in some ways, harder in others, will use codecs later to simplify

Reservation Service Contents

- ReservationServiceApplication.java
  - Spring boot auto-generated main function
- ReservationServiceRestController.java
  - Simple RESTful endpoint that turns HTTP requests into calls on ReservationService
- ReservationService.java
  - Contains the data storage logic of the service − main focus of our exercises
- Reservation.java
  - Entity definition
- ConfirmationNumberGenerator.java
  - Simple function to generate 6-digit alphanumeric confirmation numbers
  - The service is responsible for generating the confirmation numbers, not clients

Reservation Service Caveats

- Not production-ready
  - Minimal data validation
  - Minimal fault handling
  - Schema makes use of Strings as identifiers instead of UUIDs in some cases
  - All code in single package (com.cassandraguide.services.reservation)
- These allow us to focus on the Cassandra bits
  - Checkout KillrVideo for a more robust reference application

DataStax Drivers - Key Concepts

Class	Description
Cluster & ClusterBuilder	Representation of a cluster and default configuration options
Session	Manages connection to a cluster and associated resources
Statement	Represents a CQL query that can be executed: SimpleStatement, PreparedStatement -> BoundStatement, BuiltStatement
ResultSet	Result of executing a Statement, including metadata and rows matching select criteria
QueryBuilder	Fluent API for creating queries (BuiltStatements)
Mapper (Java, C#, Python)	Template for mapping Java classes to tables
*Policy (Various classes)	Multiple classes for configuring performance of the driver
*Codec (Various classes)	Manage conversions between Java and CQL types

* Topics covered in Part 2 of this course

DataStax Java Driver – Maven Configuration

- Update Maven pom.xml file for your project to include the Java Driver:

```
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-core</artifactId>
  <version>3.6.0</version>
</dependency>
```

- (IntelliJ) Right click on project, select “Maven->Reimport”
  - You may also see an option to “auto import”

http://http://docs.datastax.com/en/developer/java-driver/3.6/index.html(“Getting the Driver”)

DataStax Java Driver – Cluster and ClusterBuilder

- ClusterBuilder is a fluent interface for creating a cluster including:
  - Contact points (InetAddress)
  - Policies (we'll cover later)

```
Cluster cluster = Cluster.builder()
  .addContactPoints("127.0.0.1")
  .withLoadBalancingPolicy( ... )
  .withRetryPolicy( ... )
  .withReconnectionPolicy( ... )
  .withPoolingOptions( ... )
  .withQueryOptions( ... )
  .build();
```

DataStax Java Driver - Session

- Session represents a connection to the cluster with allocated resources
  - Thread pools
  - TCP connections
- Relatively heavyweight object
  - Create and reuse across multiple calls
- Can optionally bind to a keyspace

```
// Session bound to a keyspace
Session session =
        cluster.connect(“reservation”);

// Session not bound to a keyspace
Session anotherSession =
        cluster.connect();
```

DataStax Java Driver – SimpleStatement

- Use Session to execute statements
- Can execute raw CQL as a string, or create a SimpleStatement and pass to Session.execute()
- Tip: instead of concatenating strings, use parameterization

```
session.execute(“SELECT * FROM reservation”); // just a raw string

SimpleStatement statement = new SimpleStatement(“SELECT * FROM reservation”);
session.executeAsync(statement);

statement = new SimpleStatement(“SELECT * FROM reservation WHERE confirmation_number = ?”,
confirmationNumber); // parameterization
```

Exercise 3b – DataStax Java Driver Setup and Simple Statement

- Goals:
  - Configure project dependencies and connect to node on local machine
  - Implement TODOs in ReservationService methods using SimpleStatement
- Steps

```
reservation-service$ git checkout exercise-3b
```

  - Configure maven dependencies for our project in pom.xml
  - Configure the ReservationService class to create a Cluster to connect to our local Cassandra node
    - Specify localhost as the connection point (“127.0.0.1”)
  - Use the Cluster to create a Session bound to the ”reservation” keyspace

Exercise 3b - DataStax Java Driver Setup and Simple Statement

- Methods to update:
  - createReservation()
  - retrieveReservation()
  - updateReservation()
  - getAllReservations()
  - deleteReservation()
- Notes
  - Don’t worry about searchReservationsByHotelDate() in any of today’s exercises
  - Try to get at least the first two done, we can do the rest together

Exercise 3b - DataStax Java Driver Setup and Simple Statement

- Test:
  - Use the *curl* command to send simple HTTP requests to the service
  - This file contains sample *curl* commands for simple testing:

```
$ more ~/reservation-service/src/test/simple_test_script.sh
```

  - Take ID returned from create (POST) and substitute that in retrieve (GET)

Testing with *curl*

```
# createReservation() (The “-i” option prints out headers, including location for GET)
curl -i -X POST --data '{"hotelId": "NY456", "startDate": "2017-06-08", "endDate":
"2017-06-10", "roomNumber": "111", "guestId": "1b4d86f4-ccff-4256-a63d-45c905df2677"}' -H
"Content-Type: application/json" http://localhost:8080/reservations/
# retrieveReservation()
curl http://localhost:8080/reservations/RS2G0Z
# updateReservation()
curl -X PUT --data '{"confirmationNumber": "RS2G0Z","hotelId": "AZ123", "startDate":
"2017-06-08", "endDate": "2017-06-14", "roomNumber": "999", "guestId": "1b4d86f4-ccff-4256-
a63d-45c905df2677"}' -H "Content-Type: application/json" http://localhost:8080/reservations/
RS2G0Z
# getAllReservations()
curl http://localhost:8080/reservations/
# deleteReservation()
curl -X DELETE http://localhost:8080/reservations/RS2G0Z
```

Exercise 3b - DataStax Java Driver Setup and Simple Statement

- Finishing up:
  - To save your work (locally):

```
reservation-service$ git commit –a –m “my work”
```

  - To see the solution

```
reservation-service$ git checkout exercise-3b-solution
```

  - Discard your changes / revert (works on all exercises)

```
reservation-service$ git checkout -- .
```

DataStax Java Driver – Prepared Statement

- Created on client, driver registers with each node in the cluster
- Keep a copy for use on incoming requests
- When executed on client, only the prepared statement ID and values are sent
- Pros: efficiency, security

```
import com.datastax.driver.core.PreparedStatement;

…
PreparedStatement hotelInsertPrepared = session.prepare(
      "INSERT INTO hotels (id, name, phone) VALUES (?, ?, ?)");
```

DataStax Java Driver – Bound Statement

- To execute a query based on a PreparedStatement, we provide values to create a BoundStatement
- The BoundStatement can then be executed using the Session

```
// PreparedStatement listed attributes: id, name, phone
BoundStatement hotelInsertBound = hotelInsertPrepared.bind(
      id, "Super Hotel at WestWorld", "1-888-999-9999");

ResultSet hotelInsertResult = session.execute(hotelInsertBound);
```

Exercise 3c – Prepared Statement

- Goal:
  - Re-Implement the Reservation Service operations using PreparedStatements
- Steps:

```
reservation-service$ git checkout exercise-3c
```

  - Add import statements
  - Create PreparedStatements in constructor
  - Create/Execute BoundStatements in each operation
    - createReservation()
    - retrieveReservation()
    - updateReservation()
    - getAllReservations()
    - deleteReservation()

Exercise 3c – Prepared Statement

- Finishing up:
  - To save your work (locally):

```
reservation-service$ git commit –a –m “my work”
```

  - To see the solution

```
reservation-service$ git checkout exercise-3c-solution
```

- Questions?

DataStax Java Driver – Query Builder

- Fluent API for building up statements a portion at a time
- More programmatic, less string manipulation

```
import com.datastax.driver.core.querybuilder.BuiltStatement;
import com.datastax.driver.core.querybuilder.QueryBuilder;
import static com.datastax.driver.core.querybuilder.QueryBuilder.eq;


BuiltStatement hotelInsertBuilt = QueryBuilder.insertInto("hotels")
      .value("id", id)
      .value("name", "Super Hotel at WestWorld")
      .value("phone", "1-888-999-9999");
BuiltStatement hotelSelectBuilt = QueryBuilder.select().all().
      from("hotels").where(eq("id", id));
```

See more examples at: https://gist.github.com/yangzhe1991/10349122

Exercise 3d – Query Builder

- Goal:
  - Re-implement the Reservation Service operations using QueryBuilder
- Steps:

```
reservation-service$ git checkout exercise-3d
```

  - Add import statements
  - Create a QueryBuilder we can reuse in each operation
  - Create/execute BuiltStatements in each operation
    - createReservation()
    - retrieveReservation()
    - updateReservation()
    - getAllReservations()
    - deleteReservation()

Exercise 3d – Query Builder

- Finishing up:
  - To save your work (locally):

```
reservation-service$ git commit –a –m “my work”
```

  - To see the solution

```
reservation-service$ git checkout exercise-3d-solution
```

- Questions?

DataStax Java Driver – Mapper

- Abstracts details of mapping Java attributes to/from CQL types and UDTs
- Works via annotations
- Packaged separately from the driver – pom.xml update required

```
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-core</artifactId>
  <version>3.3.0</version>
</dependency>
```

DataStax Java Driver – Mapper Annotations

- Data Type Class

```
import com.datastax.driver.mapping.annotations.Column;
import com.datastax.driver.mapping.annotations.PartitionKey;
import com.datastax.driver.mapping.annotations.Table;

@Table(keyspace = ”hotel", name = ”hotels") public class Hotel {
    @PartitionKey @Column(name = ”id") private String id;
    @Column(name = ”name") private String name;`
    // Other attributes, Get/set/equals/hashCode methods
}
```

DataStax Java Driver – Using the Mapper

- Executing queries

```
import com.datastax.driver.mapping.MappingManager;
import com.datastax.driver.mapping.Mapper;

MappingManager mappingManager = new MappingManager(session);
Mapper<Hotel> hotelMapper = mappingManager.mapper(Hotel.class);

hotelMapper.save(hotel);  // pass a Hotel object to insert/update
Hotel hotel = hotelMapper.get(hotelId); // pass primary key to load
hotelMapper.delete(hotelId) // pass primary key to delete
```

Exercise 3e – Mapper

- Goal:
  - Re-implement the Reservation Service operations using Mapper
- Steps:

```
reservation-service$ git checkout exercise-3e
```

  - Update pom.xml to include mapper
  - Complete the annotations in ReservationByConfirmation.java
  - Imports and implement statements in ReservationService.java
    - createReservation()
    - retrieveReservation()
    - updateReservation()
    - getAllReservations()
    - deleteReservation()

Exercise 3e - Mapper

- Finishing up:
  - To save your work (locally):

```
reservation-service$ git commit –a –m “my work”
```

  - To see the solution

```
reservation-service$ git checkout exercise-3e-solution
```

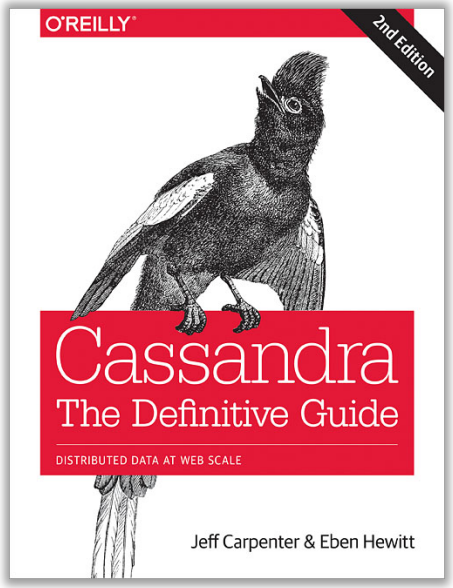
- Questions?

DataStax Java Driver – Statements

- So which is the best? It depends on your situation and preferences
  - PreparedStatement – Efficiency and safety for structured queries
  - SimpleStatement – Ad-hoc queries with variable structure
    - Always use parameterization for security
  - QueryBuilder – clean looking code
  - ObjectMapper – leveraging simple domain models, abstracting driver

Preparing for Part 2

- Finish up any remaining work from activities
- Reinforce today's learning with readings in Cassandra, The Definitive Guide, 2nd Ed:
  - Chapter 3 – Installing Cassandra
  - Chapter 4 – The Cassandra Query Language
    - Especially "CQL Types"
- Read before the next session:
  - Chapter 6 – The Cassandra Architecture
  - Chapter 7 – Configuring Cassandra
- Make sure you keep your VM around…



End of Part 1

See you in two days!