APAN PS5400: Managing Data

Week 8: NoSQL Databases, Cassandra

Lecturer: Francois Scharffe

Columbia University
School of Professional Studies

Recap of last week

Interacting with data through

- Web forms
- Programs (Python)
- APIs
- Chat bots
- JSON objects

This week

- NoSQL data bases
- CAP theorem
- Cassandra database system
- Downloading Cassandra

What are NoSQL databases?

- NoSQL stands for Not Only SQL
- Poorly named—confuses a data model (relational) for a query language (SQL)
- Should have been called Not Only Relational
- NoSQL family contains a variety of models which are included under this umbrella (see next slide)

NoSQL DB Types

Column oriented: Cassandra, Hbase

Document oriented: MongoDB, Couch DB

Graphical: Neo4j, AWS Neptune

Key-value: redis, riak

COLUMBIA UNIVERSITY
School of Professional Studies

What makes it NoSQL?

NoSQL *models* differ from the relational *model* in two main respects

- No fixed schema
  - *In relational DB each table has a fixed number of columns (fields), with each column having a fixed data type.*
  - *Each row of a table has the same number of fields (columns), with every value in a column being of the same type.*
  - *Not so in NoSQL databases*
- Typically, no relational tables—tables which reflect relations between entity sets (or between tables that reflect entity sets)
  - E.g., a NoSQL database may have a 'table' for Students and another for Courses but will not have a 'table' for Enrolled to reflect the relation about which students are enrolled in which courses
  - They provide alternate mechanisms for storing and accessing this information

Another characteristic: Performance over Perfection

Sharding

- A single ‘table’ is split into several shards (pieces)
- This implies to access a part of the ‘table’ the DBMS does not have to access the entire ‘table’
- Each shard may be stored in separate nodes

Replication

- Each shard is replicated several times
- Each replica is stored in a separate node.

Why do we need NoSQL databases

- Variety of data that cannot easily be accommodated in the relational model
  - Unstructured data (documents, etc.)
  - Graphical data
- Volume of data that requires ‘easy’ expansion of storage capacity
  - Thus, massively distributed infrastructure
- Applications that require faster response over massive datasets than can be handled by transaction processing based on joins and ACID properties.

Disadvantages of NoSQL databases

- No standardized query language
  - Need to learn the query language for each NoSQL DB system
  - Cassandra has a SQL-like query language
- Weaker control over consistency, availability, and durability
- Integrity constraints have to be implemented at the program level rather than guaranteed by the DBMS

RDBMS or NoSQL

The decision depends on the following factors (among others):

- The volume of data
- What kind of data is to be stored
- The degree to which concurrent transactions will be executed
- The volume of transactions in a unit of time
- The nature of the most frequent operations
- The degree of scalability of the DB
- The degree of consistency and integrity needed

Architectural Implications

A database system that can handle vast amounts of data that comes in rapidly requires the following features

- New data must be stored by adding new nodes to the system rather than by getting larger and larger storage space
  - Scaling horizontally rather than scaling vertically
- These must be tied together in a network
- But since any part of a network can fail (and often does fail), the system must be designed so that the DB can perform transactions on data that may be stored in temporarily inaccessible nodes
- This feature implies that data must be replicated in several different nodes some of which, hopefully, will be accessible.
- This feature is called Partition Tolerance

Consistency Issues for NoSQL DBs

How do we make sure that each replica contains the same data at all times?

- Strange if a couple with a joint bank account were to get a different balance if they checked it at different ATMs at the same time.

In RDBMS we don't commit a write to any replica until we commit the same write to all replicas.

- This ensures that the data in all replicas is the same.

What happens if one or more nodes are down or the link is broken?

- Can we guarantee consistency???

Availability Issues for NoSQL DBs

Availability means all data is available for read or write by all (authorized) users at all times.

If consistency is required in the face of node or network failure can availability be guaranteed?

- All (replicas) or none for a write means sometimes None
- That implies sometimes not available
- No read until all have same data means sometimes reads are blocked.

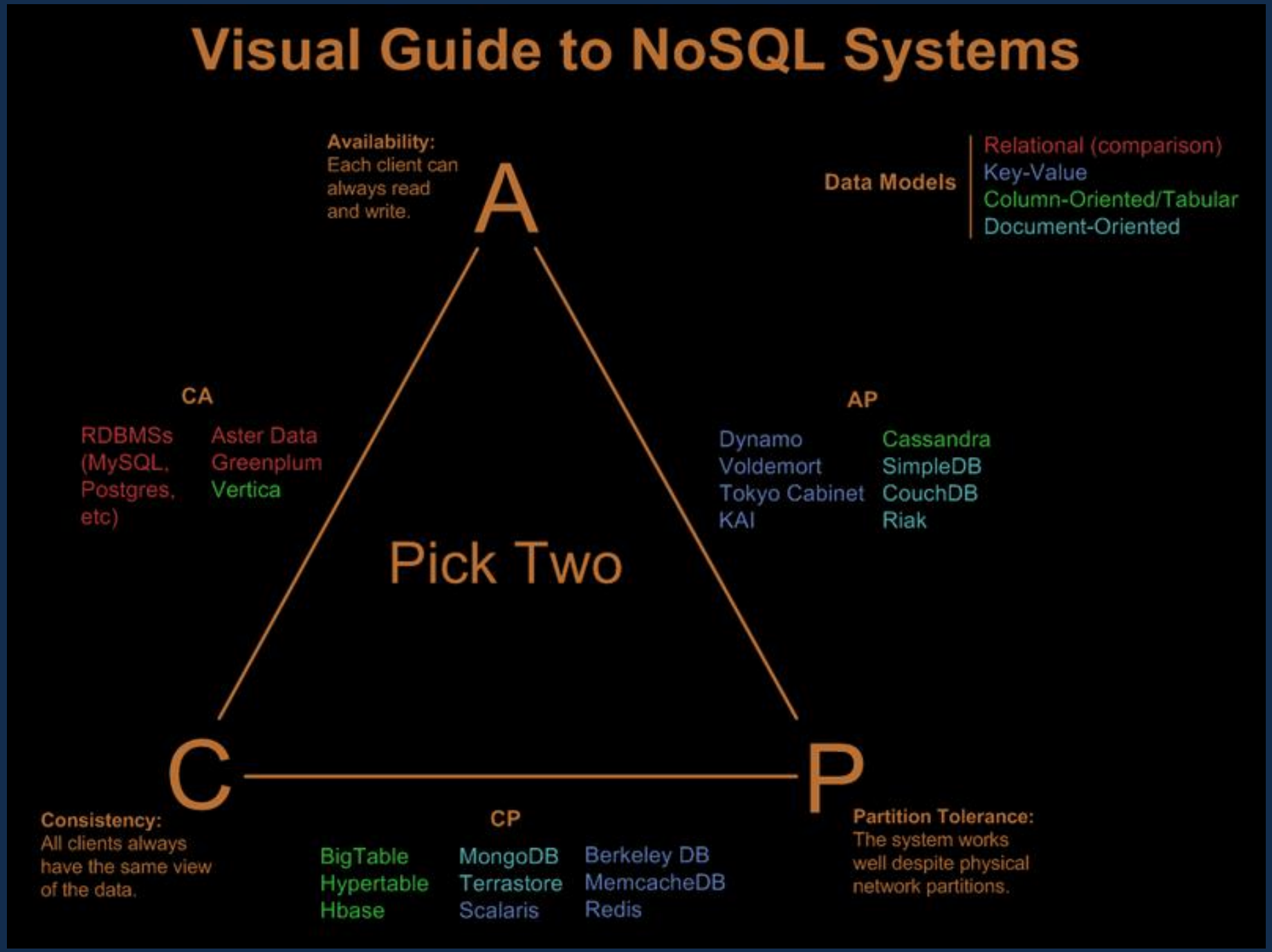
CAP Theorem

"If you cannot limit the number of faults and requests can be directed to any server and you insist on serving every request you receive then you cannot possibly be consistent" Eric Brewer (2001)

You must always give something up: consistency, availability or tolerance to node failure and network reconfiguration

In "Big Data" scenarios you need sharding and replication

- Cannot give up partitioning (P)
- That is, tolerance to node failure and network reconfiguration
- So you must give up Consistency(C) or Availability (A)



http://blog.nahurst.com/visual-guide-to-nosql-systems

CA system: In this system, you drop partition-tolerance for consistency and availability. This happens when you put everything related to a transaction on one machine or a system that fails like an atomic unit, for example, a rack. This system will have serious problems in scaling.

CP system: In a CP system, availability is sacrificed for consistency and partition-tolerance. If the system is available to serve the requests, data will be consistent. In the event of a node failure, some data will not be available. A sharded database is an example of such a system.

AP system: An available and partition-tolerance system is like an always-on system on risk of producing conflicting results in the event of network partition. This is good for user experience, your application stays available, and inconsistency in rare events may be alright for some use cases.

From Neeraj, N.. (2013). *Mastering Apache Cassandra: Get Comfortable with the Fastest NoSQL Database, Its Architecture, Key Programming Patterns, Infrastructure Management, and More!*

Cassandra as an AP System

Not good if system is chronically inconsistent

- Users will stop trusting it

Need a mechanism for detecting and repairing the inconsistency

- Even as it occasionally gives inconsistent information

This is called Eventual Consistency

BaSE Properties

NoSQL databases like Cassandra try to guarantee BaSE properties instead of ACID properties

- Ba: Basically available
  - The application basically works all the time
  - If all the nodes which store a required shard for an operation are down, then it is not available for that operation
- S: Soft state (meaning, the information will expire unless refreshed)
  - As opposed to durability in ACID.
- E: Eventually consistent

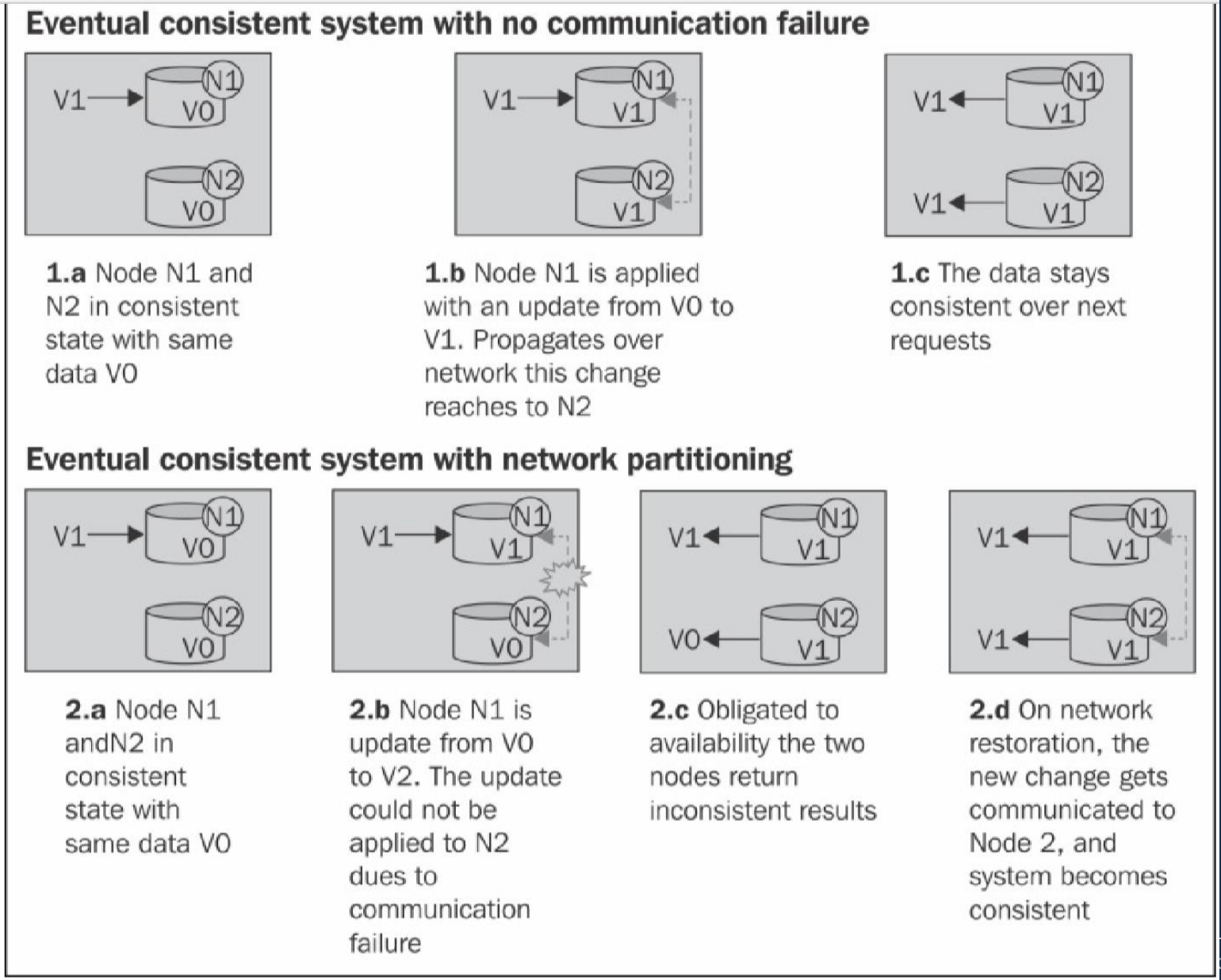


Figure 2.2: Life of an eventual consistent system

Cassandra

Users

- Facebook
- Twitter
- Many others

Based on

- Big Table
- Dynamo

Cassandra Data Model

Three data containers, one within the other

Outermost container is **Keyspace**

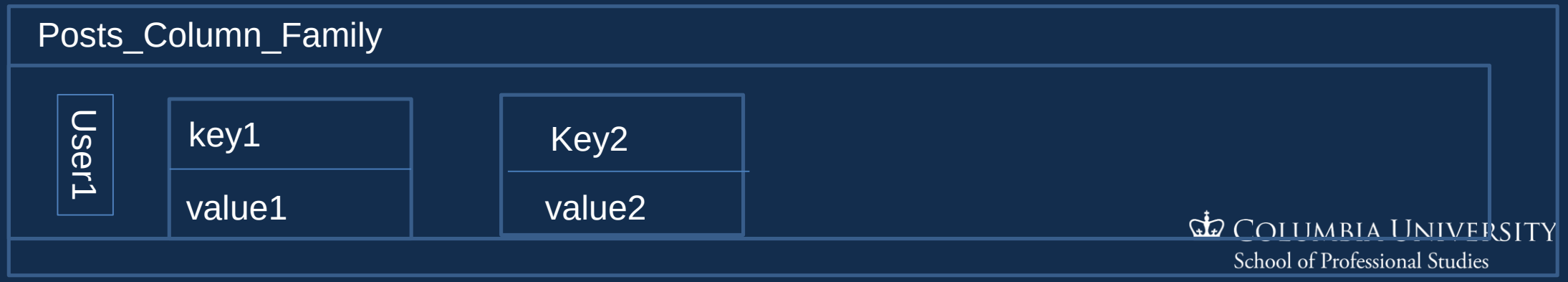
- Analogous to a database in RDBMS

Next inner container is **Column Family (Now called ‘tables’)**

- Analogous to a table in RDBMS

Within a Column Family are **Columns**

- A row consists of a RowKey and Columns (which consists of a key and a value)



RDBMS Tables vs Cassandra Column Families

Column families, unlike tables, can be schema free (schema optional). This means you can have different column names for different rows within the same column family. There may be a row that has user_name, age, phone_office, and phone_home, while another row can have user_name, age, phone_office, office_address, and email.

From Neeraj, N.. (2013). *Mastering Apache Cassandra: Get Comfortable with the Fastest NoSQL Database, Its Architecture, Key Programming Patterns, Infrastructure Management, and More!*

Another Difference

Cassandra has no relational (connecting) tables

- If you have Students and Classes tables in RDBMS, you will need a relational table EnrolledIn, which shows which student is enrolled in which class
- Foreign key reference from StudentId in EnrolledIn to StudentId in Students
- Similarly for CourseId

In Cassandra, you could achieve some of the same effect by having a column in Student row that shows a ClassId for a student enrolled in a class

So avoids joins when querying, which can be an expensive operation.

Partitioning Rows

Each row has a row key

A set of rows (from a Column family) are partitioned into subsets (shards) and these shards are in different servers.

Each shard must contain a contiguous set of keys

Writes in Action

Client needs to connect to *any* of the nodes and send a Write request

This node invokes the StorageProxy service, which gets all the replicas where this Write needs to happen

It send the Write message to these replicas and waits for confirmation

- Sends it to one, and that sends to another, and so on

Important: Write is considered successful if a certain number of replicas (not necessarily all) send Success message

- How many is enough
- That depends on a pre-specified Consistency Level

If not enough respond with Success, the Write fails

Read in Action

Client sends a Read request to any node

It gets all the replicas based on the key

It sorts these replicas in terms of which is closest to it

It sends the Read request to that Replica, which passes on the request to the other Replicas

- Other replicas don't send the actual data but a "digest" of the data (a hash of the data), that is, the data in a highly compressed form

Depending on the Read Consistency Level specified, the Read is sent to the Client if enough replicas send the same result.

Instructions for installing Cassandra 3.11.3

Link to detailed instructions for installing Cassandra are provided under the Resources for Module 8.

Unfortunately, they are more complex than for installing MySQL. Please follow them closely.

```
cqlsh> CREATE KEYSPACE mySpace with replication =  {‘class’: ‘SimpleStrategy’, ‘replication_factor’ : 3}; #end all commands with a semi-column

cqlsh> use mySpace;

cqlsh:mySpace> CREATE TABLE students (id int Primary Key, name TEXT, gpa double);
# See how the prompt changed from cqlsh to cqlsh:mySpace

cqlsh:mySpace> insert into students (id, name, gpa) values (111, 'Liu', 3.7);

cqlsh:mySpace > select * from students;
id    | name | gpa
111| Liu      | 3.7
```

If you get an error message on any of your queries suggesting you use ALLOW FILTERING, then re-write the query as: select * from students ALLOW FILTERING;

Reasons for this have to do with the underlying storage model.

As you can see, the commands are a lot like regular SQL for Relational Databases

Assignment for this week

- Need to represent information about
  - Two classes of entities: books and users, and
  - Events of the type of book check outs
- In relational model, the check outs would be in a table checkouts(book_id, user_id, date) with FK references to the books and users table.
- Consider the query: which users (by name) checked out which books (by name)
  - In RDB we would do joins to answer this query
- Cassandra does not allow joins. So to answer that query all the information to answer this query would have to be in one table. Please keep this in mind in doing the assignment.
  - This does not mean the assignment requires you to create only one table.

This week

- NoSQL databases
- CAP theorem
- Cassandra database model

Next week

- MongoDB data model
- Installing MongoDB
- Interacting with MongoDB