Apache Cassandra

Jean-Claude Charr

MCF IUT NFC

Overview

- Apache Cassandra is an open source, distributed, NoSQL database. It presents a partitioned wide column storage model.
- It was designed at Facebook and it implements a combination of Amazon's Dynamo and Google's Bigtable.
- It was developed to deliver always available low-latency reads and writes which the relational database systems struggle to meet.
- It should be able to handle largescale applications in terms of data footprint, query volume and storage requirements.

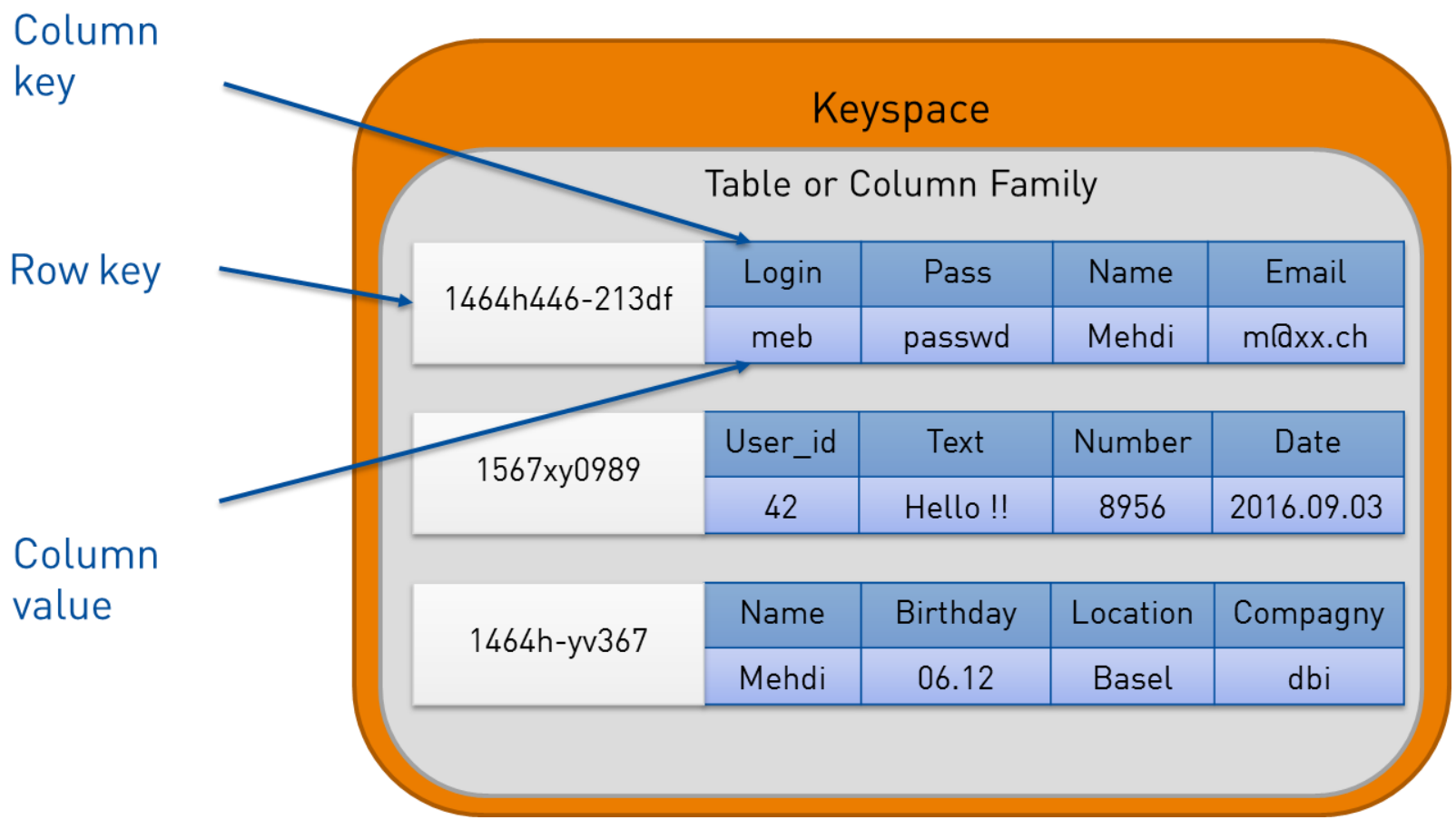
Design objectives of Cassandra

- Full multi-master database replication
- Global availability at low latency
- Scaling out on commodity hardware
- Linear throughput increase with each additional processor
- Online load balancing and cluster growth
- Partitioned key-oriented queries
- Flexible schema

Data organization in Cassandra

- Tables are located in keyspaces. A keyspace defines options that apply to all the keyspace’s tables. The replication strategy and factor are important keyspace options.
- A Table defines the typed schema for a collection of partitions. Tables contain partitions, which contain rows, which contain columns. Cassandra tables can flexibly add new columns to tables with zero downtime.
- A Partition defines the mandatory part of the primary key all rows in Cassandra must have to identify the node in a cluster where the row is stored. Queries always supply the partition key.
- A Row contains a collection of columns and is identified by a unique primary key made up of the partition key and optionally additional clustering keys.
- A Column is a single datum with a type which belongs to a row.

Inside a Keyspace

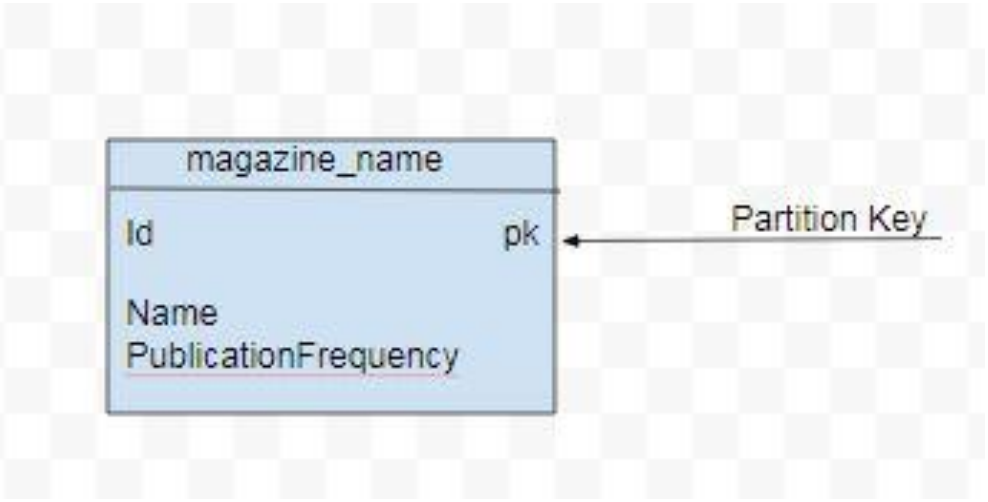


Data modeling

- It is the process of identifying entities and their relationships.
- In relational databases, queries are driven by the structure of the tables and related data are queried as table joins.
- In Cassandra, data modeling is query-driven. The data access patterns determine the structure and organization of data in the database.
- Queries are best designed to access a single table, which implies that all entities involved in a query must be in the same table to make data access (reads) very fast. Joins are not supported in Cassandra.
- A single entity may be included in multiple tables, denormalization.

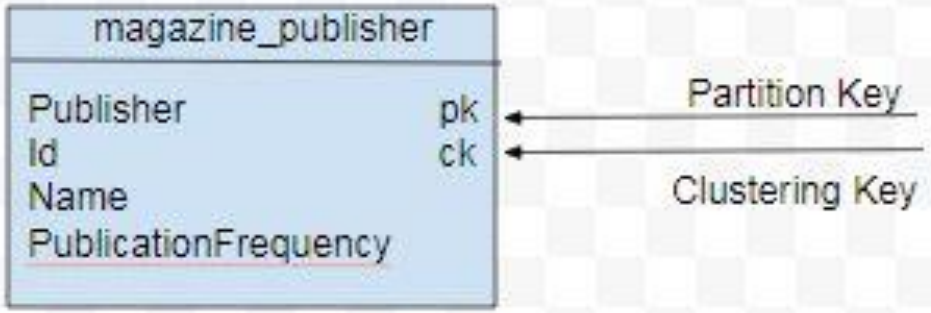
Example of data modeling

- Consider a dataset for magazines with attributes such as magazine id, magazine name, publication frequency, publication date, and publisher.
- The application will list all the magazine names including their publication frequency:



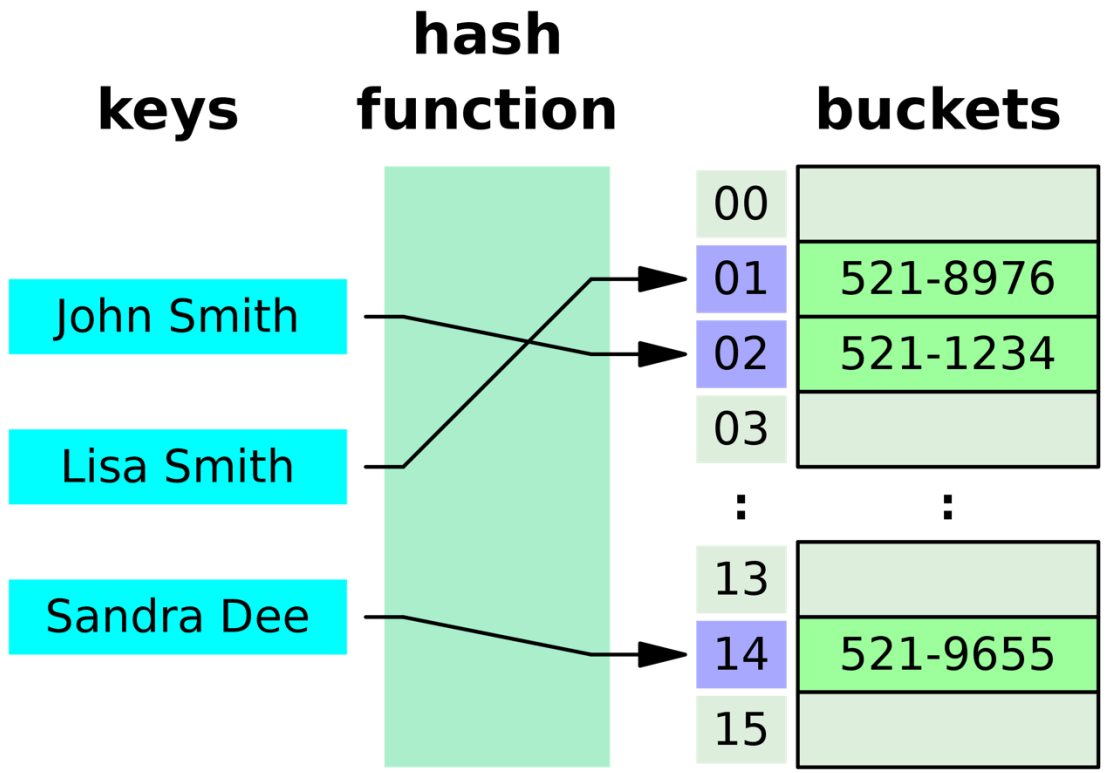
Example of data modeling

- The application will also list all the magazine names by publisher:



Hashing

- Hashing is a technique used to map data with which given a key, a hash function generates a hash value (or simply a hash) that is stored in a hash table.
- A hash table maps keys to values using a hash function. During lookup, the key is hashed and the resulting hash indicates where the corresponding value is stored.



Distributed hash table (DHT)

- It is maintained by all the participating nodes and any of them can efficiently retrieve the value associated with a given key. The main advantage of a DHT is that nodes can be added or removed with a minimal amount of disruption.

- how does the distribution take place?
- Hash modulo the number of nodes

KEY	HASH	HASH mod 3
"john"	1633428562	2
"bill"	7594634739	0
"jane"	5000799124	1
"steve"	9787173343	0
"kate"	3421657995	2

The Rehashing Problem

- What happens if one of the servers crashes or a new node is added?
- Keys need to be redistributed to account for the missing server or the added one.
- With the modulo distribution, when the number of servers changes, most hashes modulo N will change, so most keys will need to be moved to a different server.

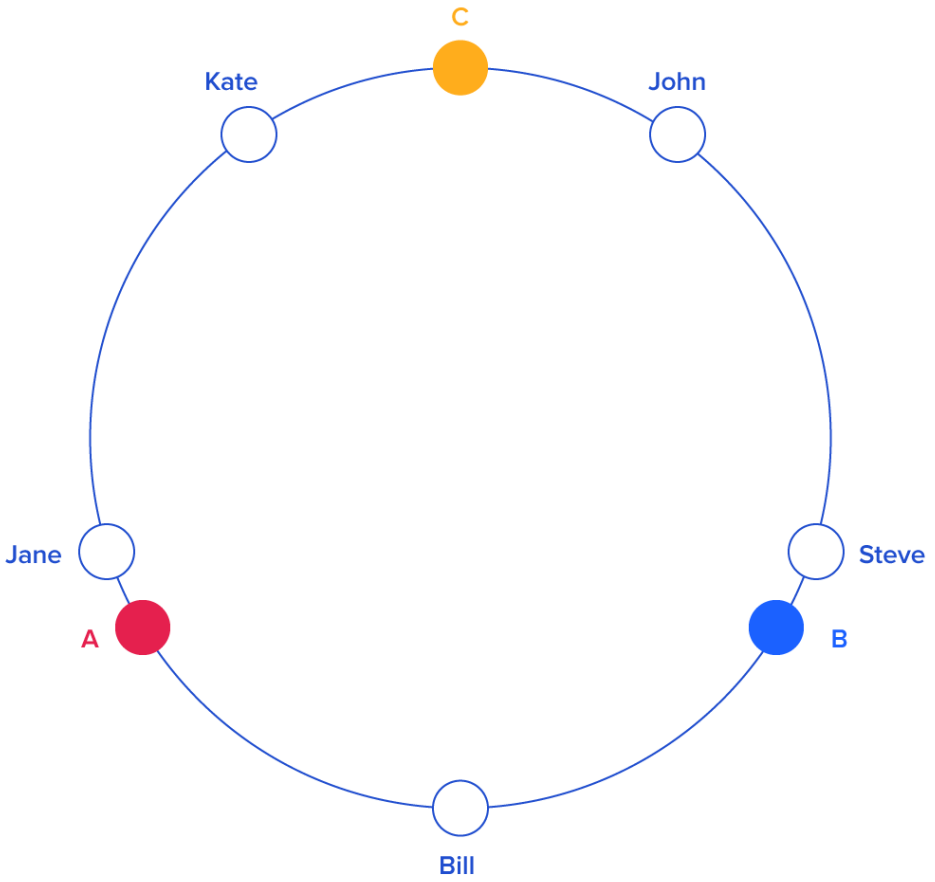
KEY	HASH	HASH mod 2
"john"	1633428562	0
"bill"	7594634739	1
"jane"	5000799124	0
"steve"	9787173343	1
"kate"	3421657995	1

Consistent Hashing

- Consistent Hashing is a distributed hashing scheme that operates independently of the number of servers or objects in a distributed hash table by assigning them a position on an abstract circle, or hash ring. This allows servers and objects to scale without affecting the overall system.
- The hash output range from 0 to 360 and is mapped onto the edge of a circle.

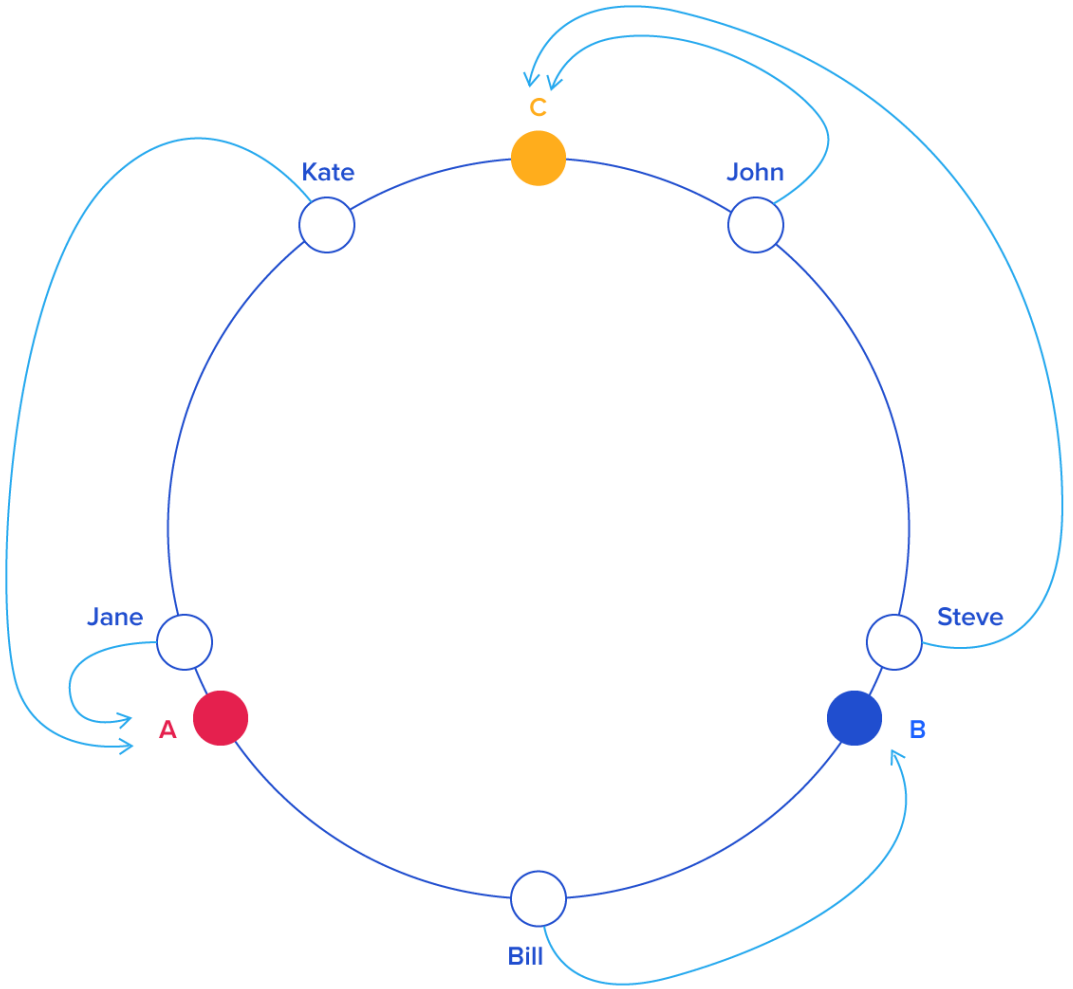
Consistent Hashing: mapping

KEY	HASH	ANGLE (DEG)
"john"	1633428562	58.7
"C"	2269549488	81.7
"kate"	3421657995	123.1
"jane"	5000799124	180
"A"	5572014557	200.5
"bill"	7594634739	273.4
"B"	8077113361	290.7
"steve"	787173343	352.3



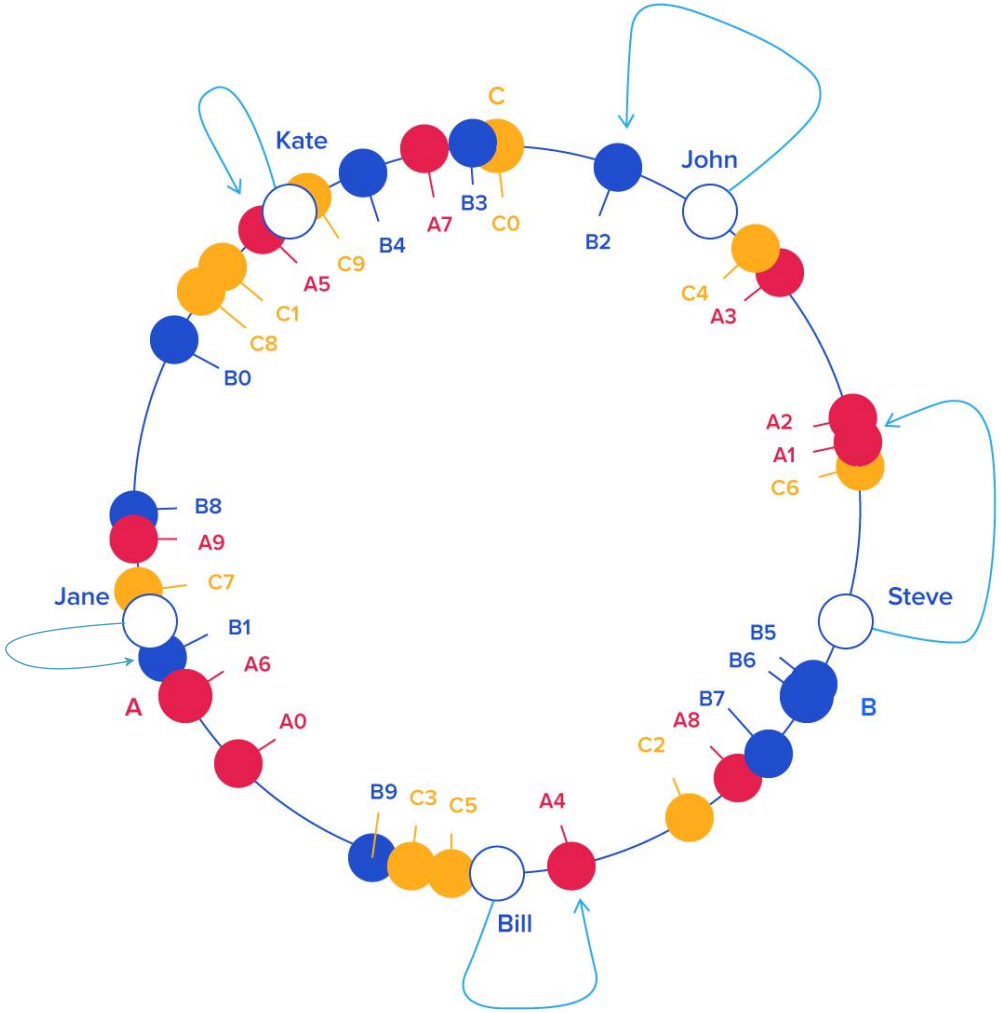
Consistent Hashing: distribution

- A simple rule to associate the data with the nodes: Each object key will belong in the server whose key is the closest, in a counterclockwise direction (or clockwise, depending on the conventions used).

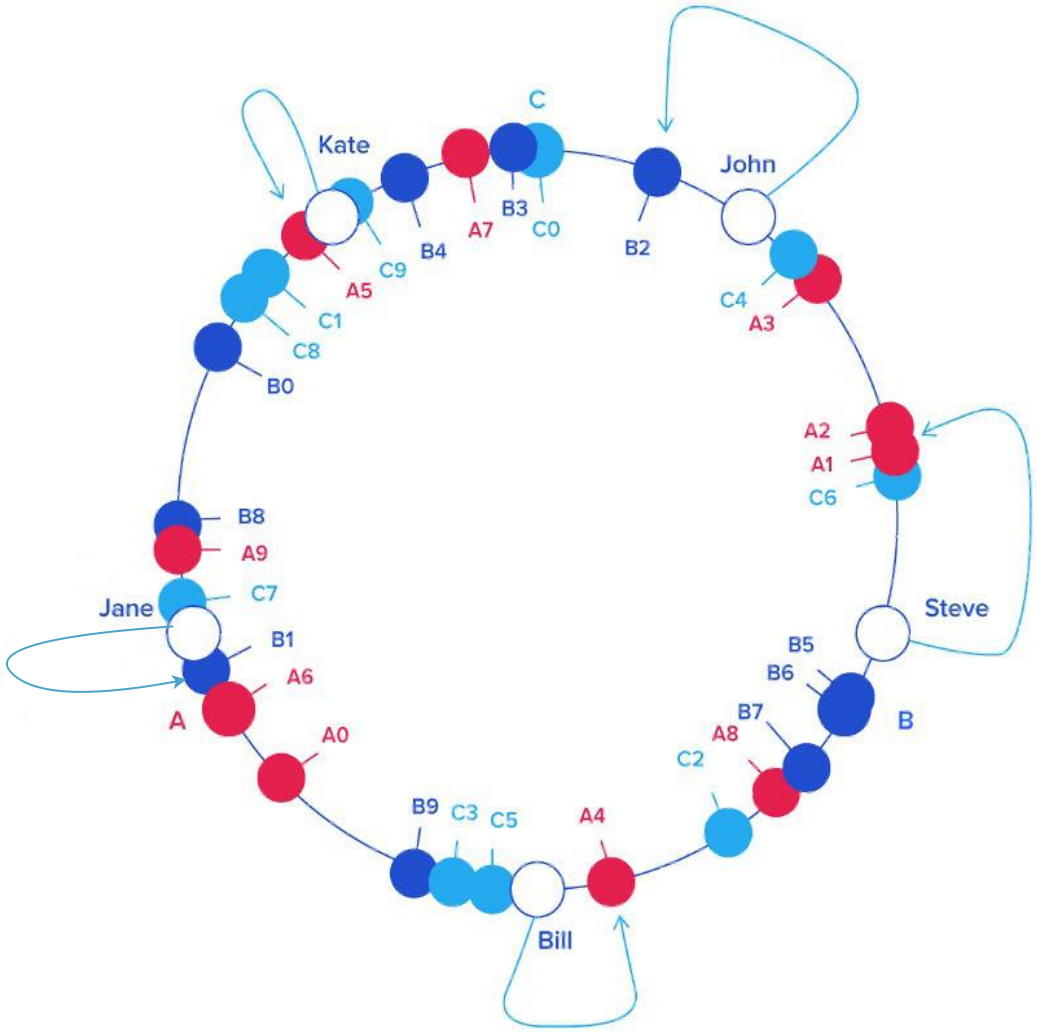


Consistent Hashing: even distribution

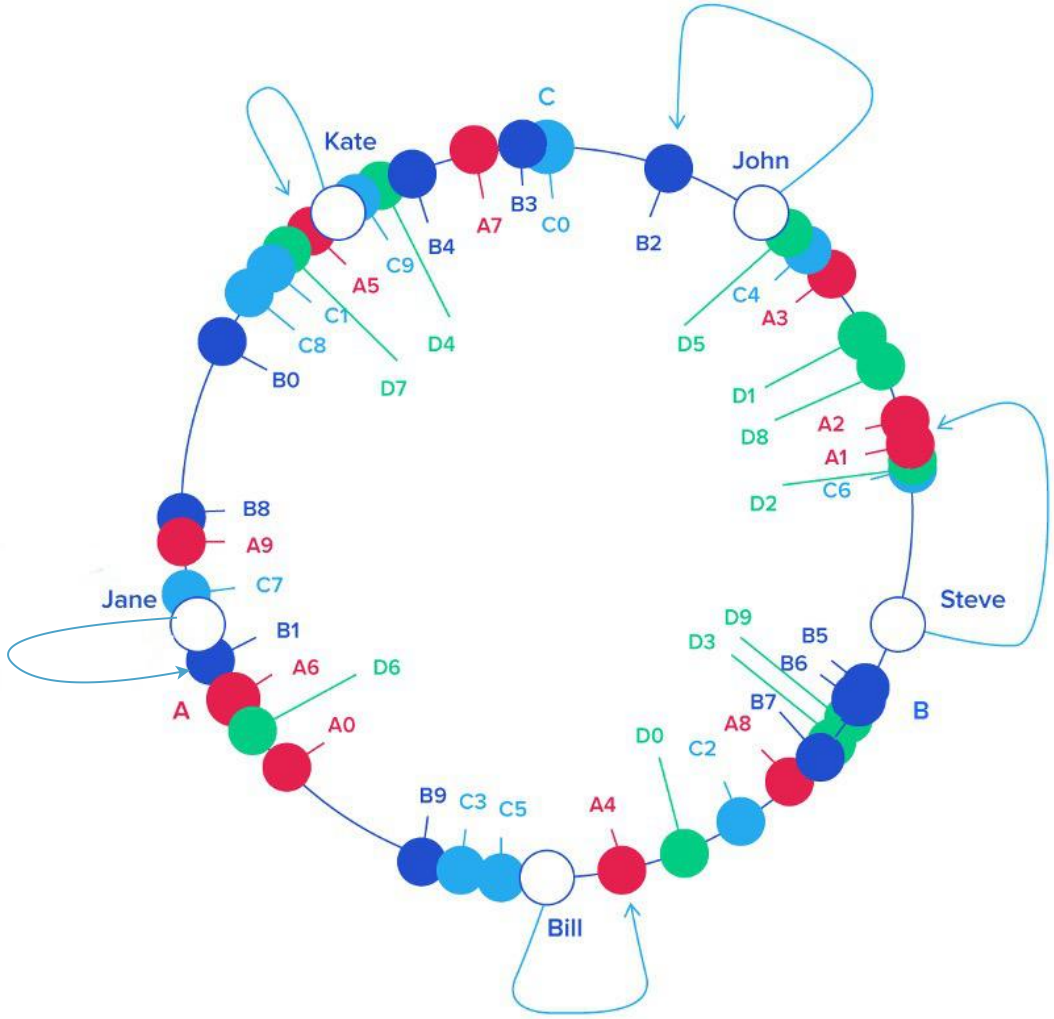
- To ensure keys are evenly distributed among servers, many labels (angles) could be assigned to each server. So instead of having labels A, B and C, we could have A0 .. A9, B0 .. B9 and C0 .. C9, all interspersed along the circle.
- The factor by which to increase the number of labels (server keys), known as weight, depends on the situation and may even be different for each server.



Consistent Hashing: removing a node



Consistent Hashing: adding a new node



Partitions

- Cassandra stores data across a cluster of nodes.
- A partition key is used to partition data among the nodes using a variant of the consistent hashing technique.
- The choice of the partition key is important to distribute data evenly across the nodes of the cluster.
- An efficient query must minimize the number of partitions read because different partitions could be located on different nodes.
- The partition key is generated from the first field of a primary key.
- As a general guideline, the values per partition must stay below 100,000 and disk space per partition below 100MB.

Clustering Key

- The clustering key is used for sorting within a partition which makes retrieval of adjacent data more efficient.

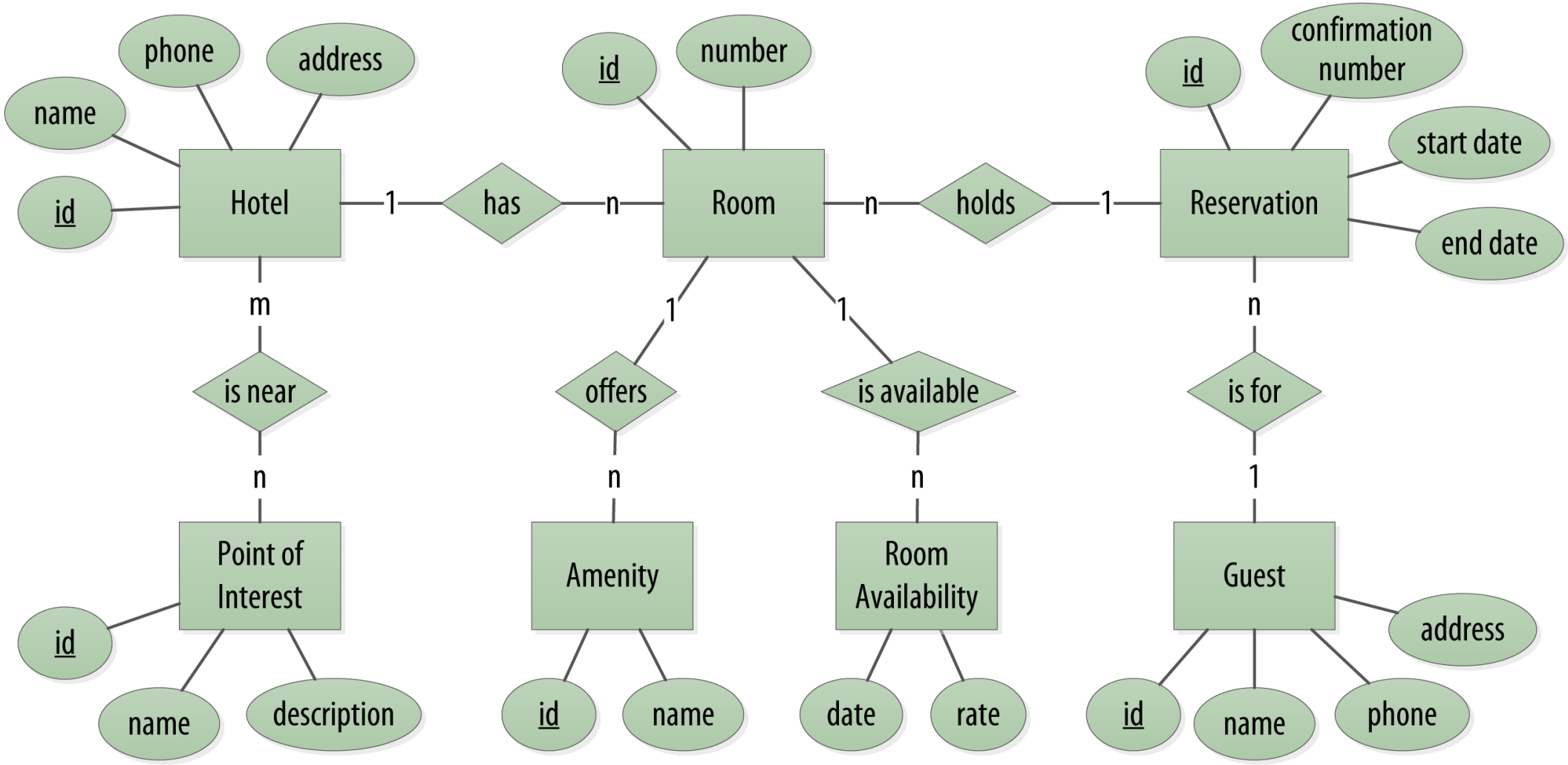
```
CREATE TABLE t (
   id int,
   c text,
   k int,
   v text,
   PRIMARY KEY (id,c)
);
```

```
CREATE TABLE t (
   id1 int,
   id2 int,
   c1 text,
   c2 text
   k int,
   v text,
   PRIMARY KEY ((id1,id2),c1,c2)
);
```

Summary of Design Differences Between RDBMS and Cassandra

- No joins
- No referential integrity
- Denormalization
- Query-first design
- Designing for optimal storage
- Sorting is a design decision

Data modeling in relational database



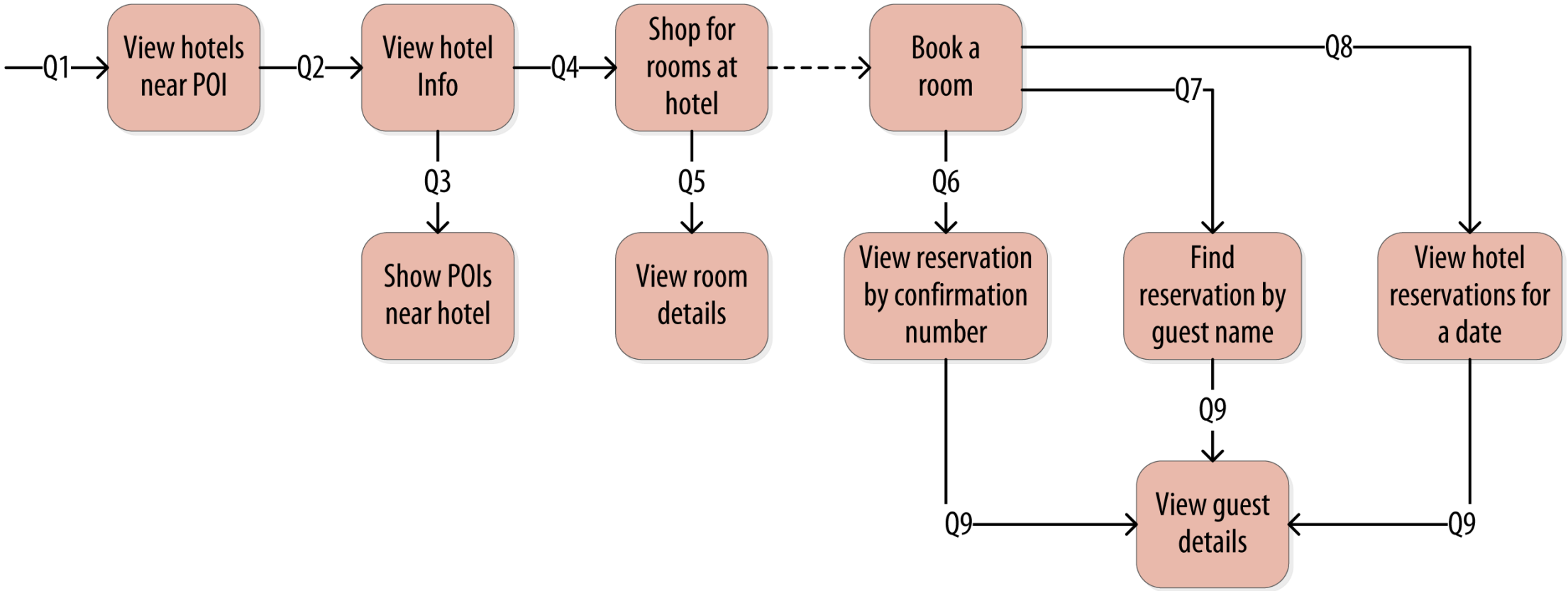
Cassandra’s query first design

- Q1. Find hotels near a given point of interest.
- Q2. Find information about a given hotel, such as its name and location.
- Q3. Find points of interest near a given hotel.
- Q4. Find an available room in a given date range.
- Q5. Find the rate and amenities for a room.

Cassandra’s query first design

- Q6. Lookup a reservation by confirmation number.
- Q7. Lookup a reservation by hotel, date, and guest name.
- Q8. Lookup all reservations by guest name.
- Q9. View guest details.

Workflow of the application



Logical Data Modeling

- Create a logical model containing a table for each query, capturing entities and relationships from the conceptual model.
- A table's name starts with the name of the primary entity that you are querying followed by "by" and the name of other related entities that you are querying by their attributes. For example, hotels_by_poi.
- The primary key for a table, consists of the partition key columns based on the required query attributes, and clustering columns to guarantee uniqueness and support desired sort ordering.
- Additional attributes identified by the query are added to the table. If any of these additional attributes are the same for every instance of the partition key, mark the column as static.

Cassandra’s data models in diagrammatic form

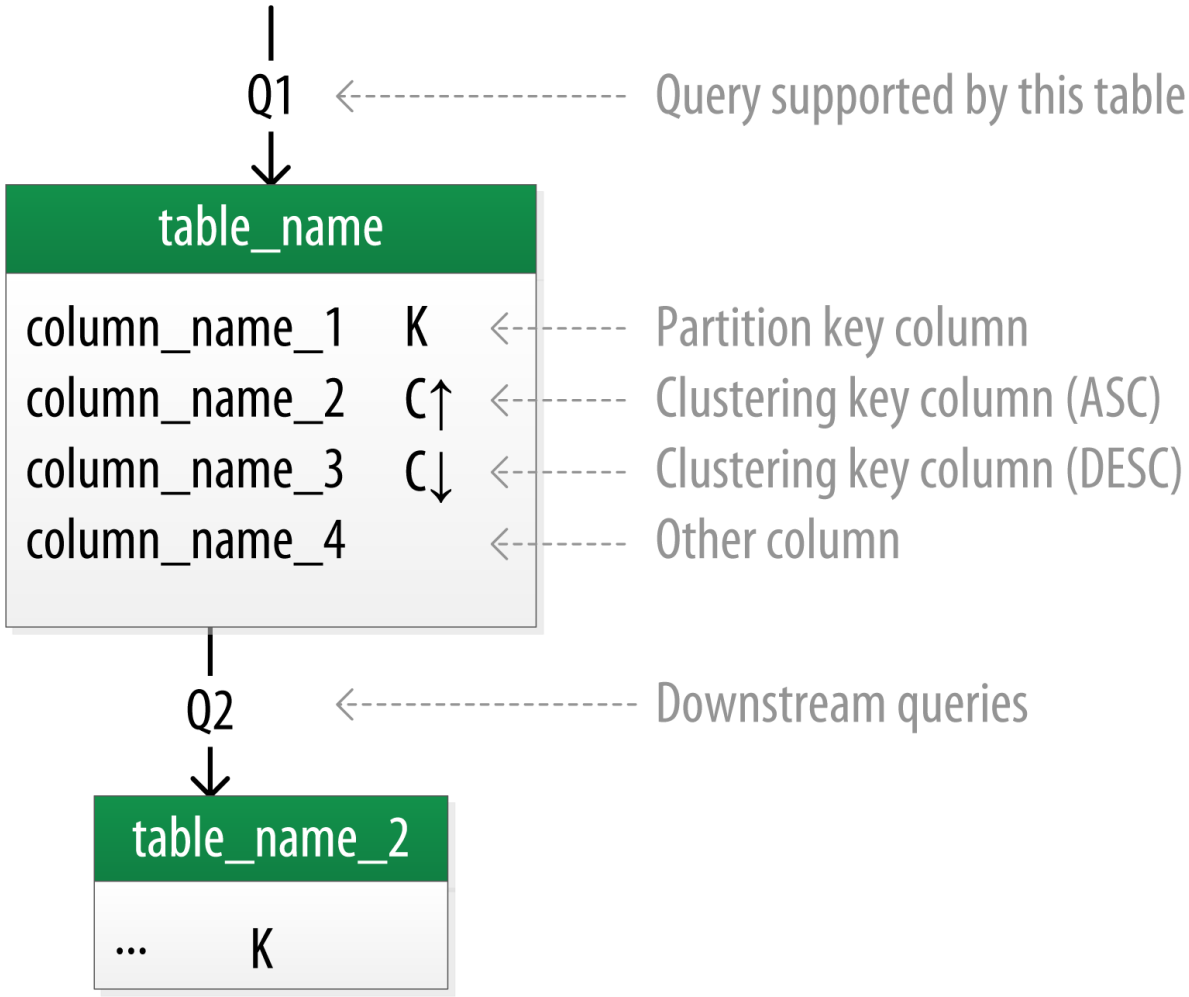
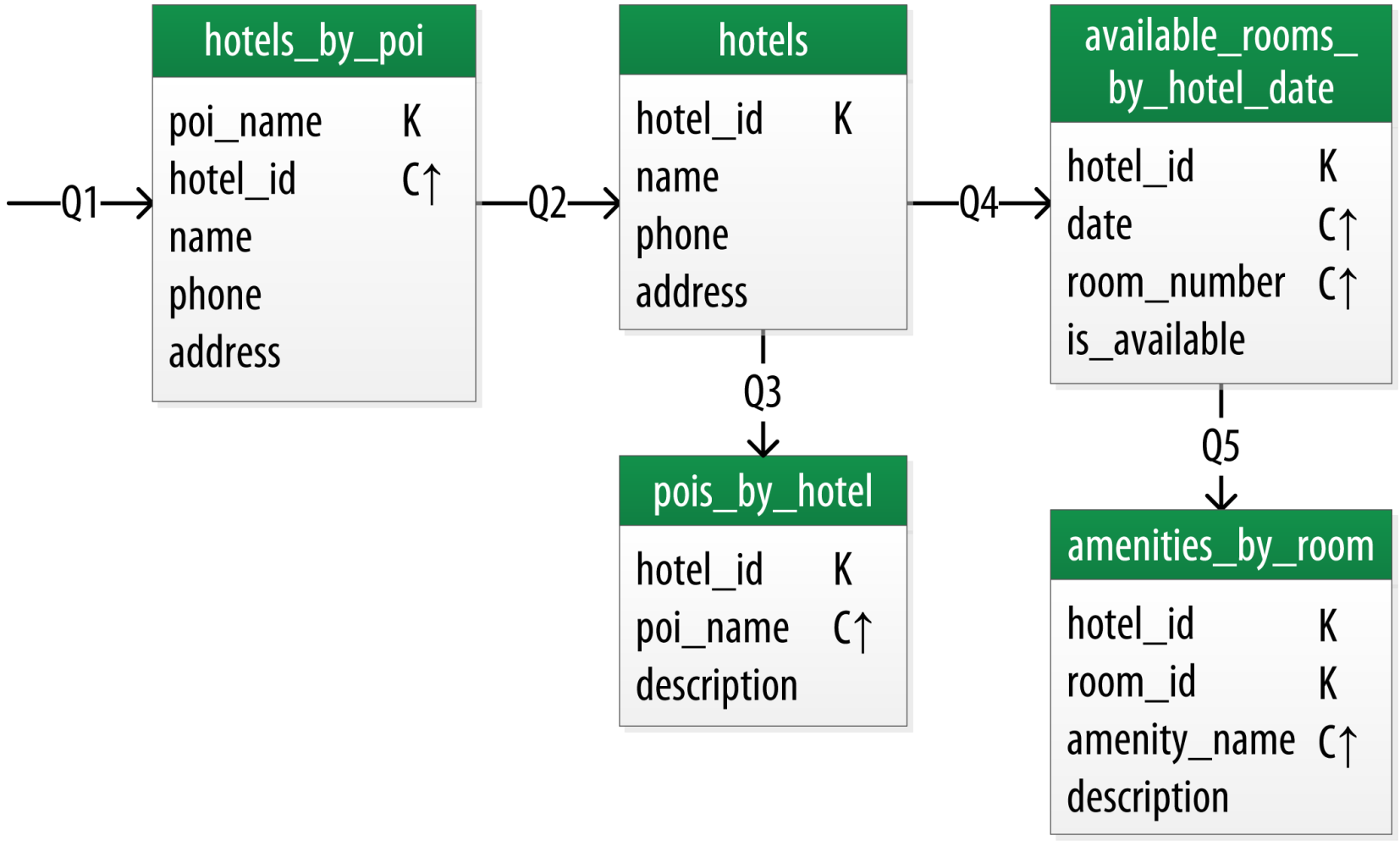
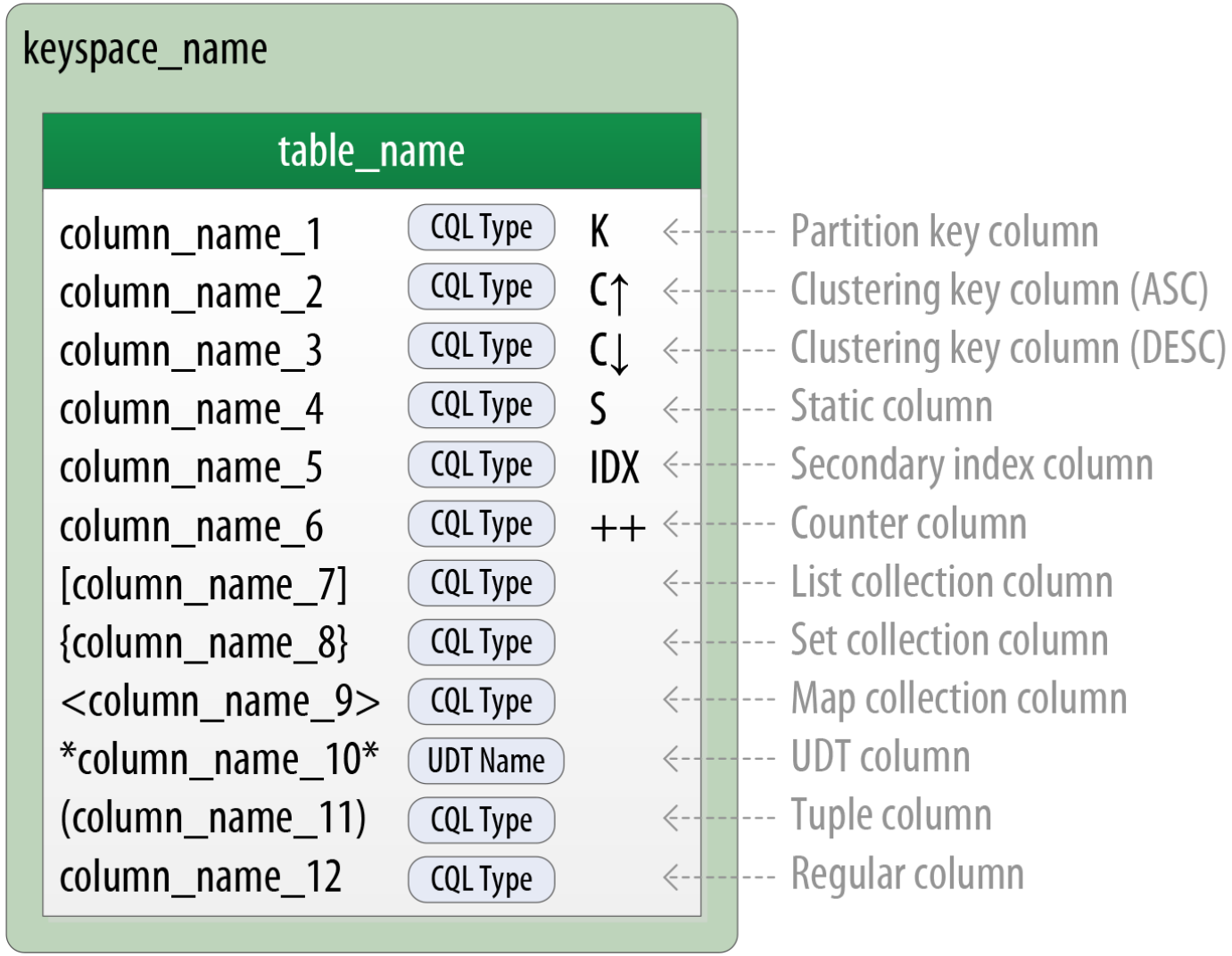


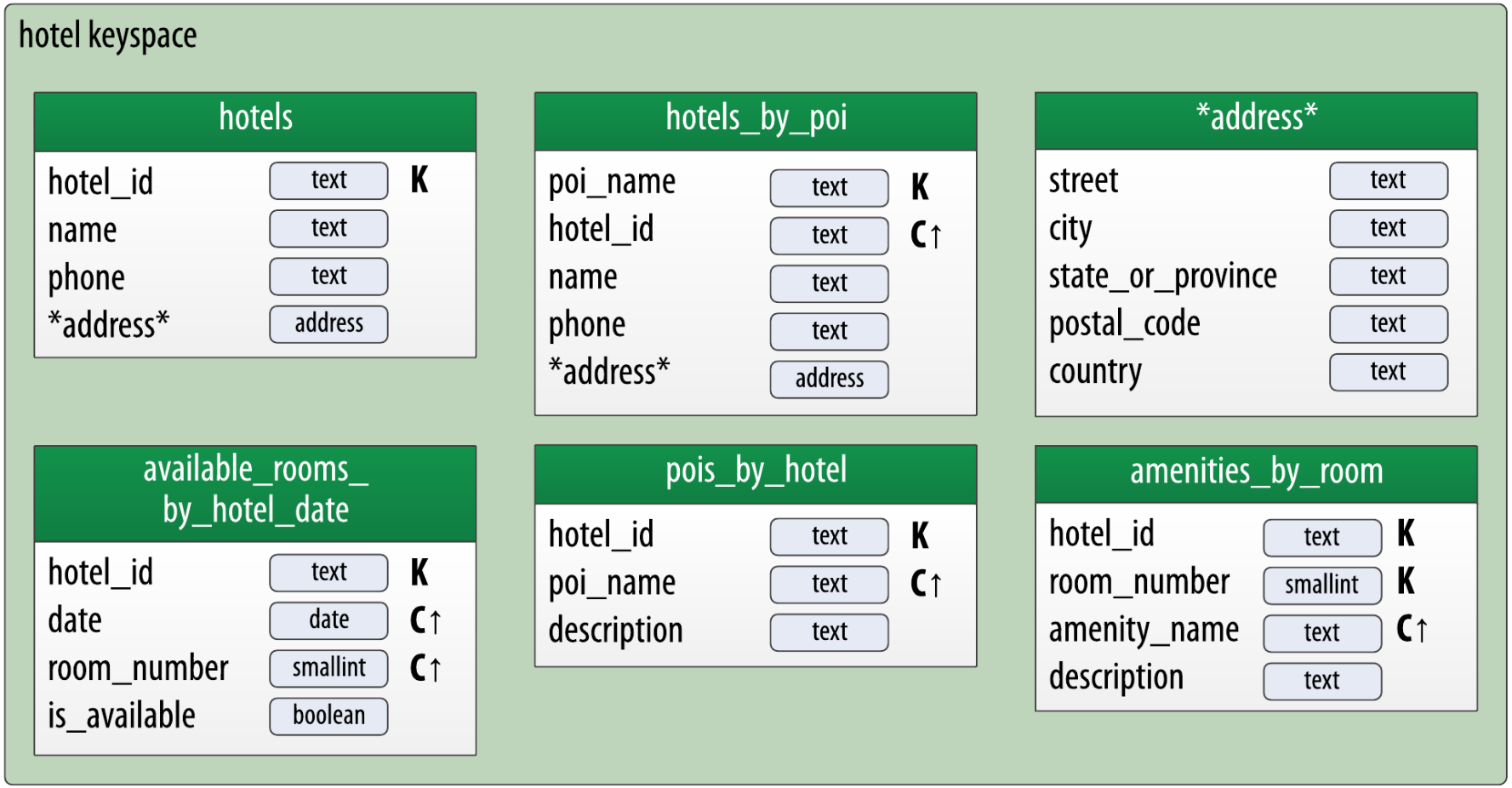
Diagram for the hotel reservation example



Physical Data Modeling diagram



Physical Data Modeling Hotel Example



Calculating Partition Size

- Partition size is measured by the number of cells (values) that are stored in the partition.
- Cassandra's hard limit is 2 billion cells per partition.
- To compute the number of values : $N_v = N_r \times (N_c - N_{pk} - N_s) + N_s$

  Where: N_r is the numer of rows

  N_c is the number of columns

  N_{pk} is the number of primary key columns

  N_s is the number of static columns

Calculating Size on Disk

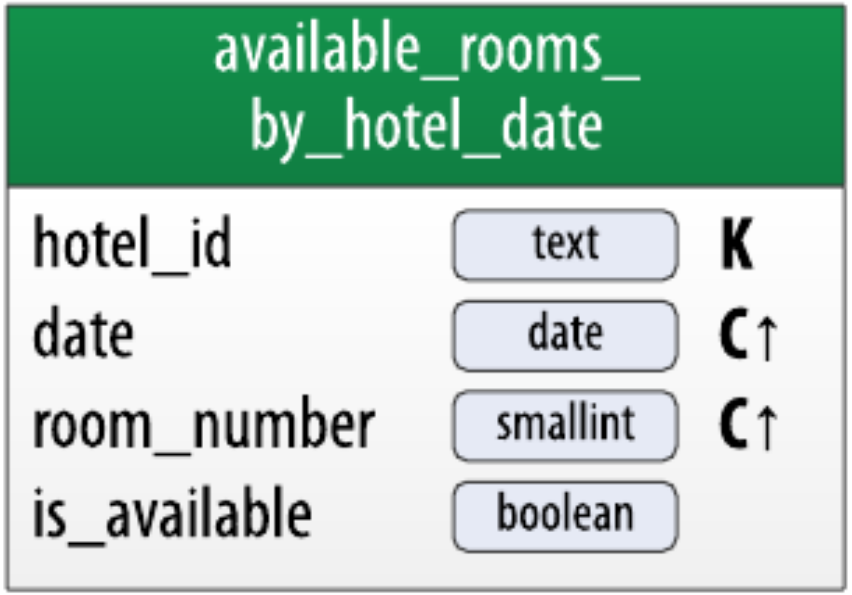
$$S_t = \sum_i sizeOf\left(c_{k_i}\right) + \sum_j sizeOf\left(c_{s_j}\right) + N_r \times \left(\sum_k sizeOf\left(c_{r_k}\right) + \sum_l sizeOf\left(c_{c_l}\right) \right) +$$

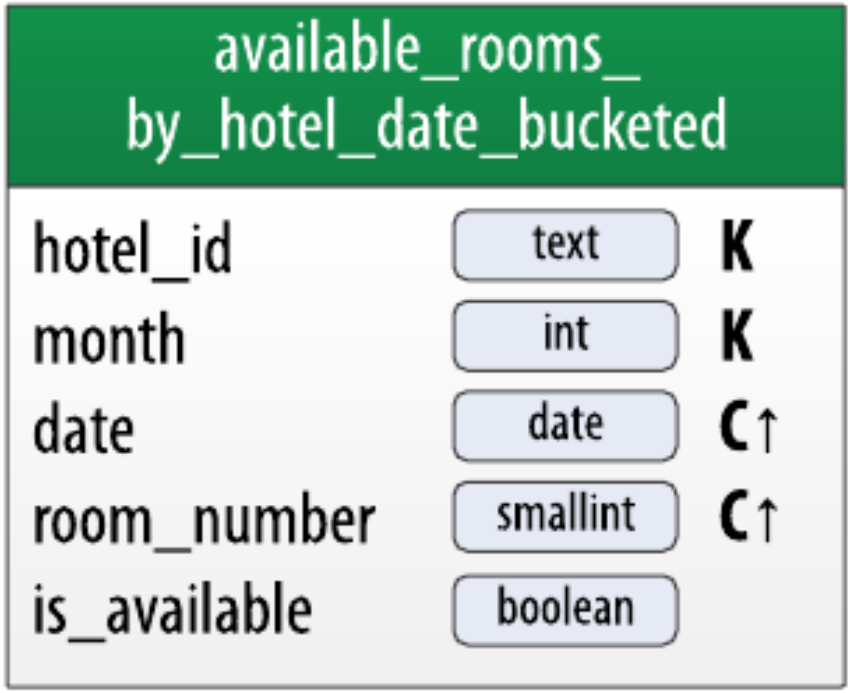
$$N_v \times sizeOf\left(t_{avg}\right)$$

- c_k refers to partition key columns, c_s to static columns, c_r to regular columns, and c_c to clustering columns.
- The term t_{avg} refers to the average number of bytes of metadata stored per cell, such as timestamps. It is typical to use an estimate of 8 bytes for this value.

Breaking Up Large Partitions

- Add an additional column to the partition key





CQL : Cassandra Query Language

- It is an SQL-like language, to create and update database schema and access data.
- CQL supports numerous advanced features such as:
  - Single partition lightweight transactions with atomic compare and set semantics.
  - User-defined types, functions and aggregates
  - Collection types including sets, maps, and lists.
- Cassandra explicitly chooses not to implement operations that require cross partition coordination as they are typically slow and hard to provide highly available global semantics. For example, Cassandra does not support:
  - Cross partition transactions
  - Distributed joins
  - Foreign keys or referential integrity.

Data Types

- Native types : int, float, text, time, timestamp, counter, uuid, etc.
- Collections : maps, sets and lists. To store a small amount of data.
  - A map is a (sorted) set of key-value pairs, where keys are unique and the map is sorted by its keys.
  - A set is a (sorted) collection of unique values.
  - A list is a (sorted) collection of non-unique values where elements are ordered by their position in the list.
- User-Defined Types (UDTs) can be created, modified and removed.
- Tuples can contain elements of different types and are frozen (immutable).

Data definition

- Schema defines the layout of the data in the table.
- Tables are located in keyspaces.
- A keyspace defines options that apply to all the keyspace’s tables, such as the replication strategy and factor.
- CREATE KEYSPACE keySpaceName WITH replication = {'class’: ['SimpleStrategy’ or 'NetworkTopologyStrategy’], 'DC1' : 1, 'DC2' : 3};
- Use keySpaceName;

Create Table

- A CQL table has a name and is composed of a set of rows. Creating a table amounts to defining the columns, which of those columns comprise the primary key, as well as defined options for the table.
- A column has a name and a type, restricting the values that are accepted for that column. Additionally, a column can be STATIC or PRIMARY KEY
- A column that is static will be “shared” by all the rows belonging to the same partition (having the same partition key).

Primary key

- Within a table, a row is uniquely identified by its PRIMARY KEY, and hence all tables must define a single PRIMARY KEY.
- A PRIMARY KEY is composed of one or more of the defined columns in the table.
- A CQL primary key is composed of two parts:
  - The partition key is the first component of the primary key definition. It can be a single column or, using an additional set of parenthesis, can be multiple columns. A table must have at least one partition key.
  - Clustering columns are the columns that follow the partition key in the primary key definition. The order of those columns defines the clustering order.

Table options

- A CQL table has a number of options that can be set at creation. These options are specified after the WITH keyword.
- CLUSTERING ORDER BY table option uses a comma-separated list of the clustering columns, each set for either ASC or DESC. Select queries cannot apply other row sorting than the reverse order.
- Compression options.
- Caching options.

Defining Database Schema

- 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 hotels_by_poi table

```
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 comment = ‘Q1. Find hotels near given poi’
                                    AND CLUSTERING ORDER BY (hotel_id ASC) ;
```

Select query

- The SELECT statement returns one or more columns for one or more rows in a table matching the request. Additionally, functions including aggregations can be applied to the result.
- A SELECT statement contains the name of the columns to be selected and the name of the table on which the selection is executed.
- CQL does not execute joins or sub-queries and a select statement only apply to a single table.
- A select can also have a where clause that can further narrow the query results. Additional clauses can order or limit the results

Where clause

- The WHERE clause specifies which rows are queried. It specifies a relationship for PRIMARY KEY columns or a column that has a secondary index defined, along with a set value.
- Not all relationships are allowed in a query. For instance, only an equality is allowed on a partition key.
- A partition key must be specified before clustering columns in the WHERE clause.
- The relationship for clustering columns must specify a contiguous set of rows to order.

Group by

- The GROUP BY option can condense all selected rows that share the same values for a set of columns into a single row.
- Using the GROUP BY option, rows can be grouped at the partition key or clustering column level. Consequently, the GROUP BY option only accepts primary key columns in defined order as arguments.
- Aggregate functions will produce a separate value for each group. If no GROUP BY clause is specified, aggregates functions will produce a single value for all the rows.

Allowing filtering

- CQL only allows select queries that don’t involve a full scan of all partitions. The ALLOW FILTERING option explicitly executes a full scan.
- SELECT * FROM users WHERE birth_year = 1981 AND country = 'FR' ALLOW FILTERING;
- To add constraints to queries, not related to the partition key, a secondary index must be created on the columns related to the constraint.

Insert and update

- INSERT writes one or more columns for a given row in a table. Unlike in SQL, CQL's INSERT updates the row if it existed.
- INSERT INTO tableName (columnNames) VALUES ('values');
- UPDATE writes one or more Non-primary key columns for a given row in a table. The WHERE clause is used to select the row to update and must include all columns of the PRIMARY KEY. Unlike in SQL, CQL's update insert the row if it does not exist.
- UPDATE tableName USING SET columnName = 'value' WHERE condition.

Delete and Batch

- DELETE removes columns and rows. If column names are provided directly after the DELETE keyword, only those columns are deleted from the row indicated by the WHERE clause. Otherwise, whole rows are removed. WHERE specifies which rows are to be deleted.
- DELETE FROM WHERE condition;
- BATCH group multiple modification statements (insertions /updates and deletions) into a single statement. All updates in a BATCH belonging to a given partition key are performed in isolation.

References

- https://www.toptal.com/
- Wikipedia
- https://cassandra.apache.org/