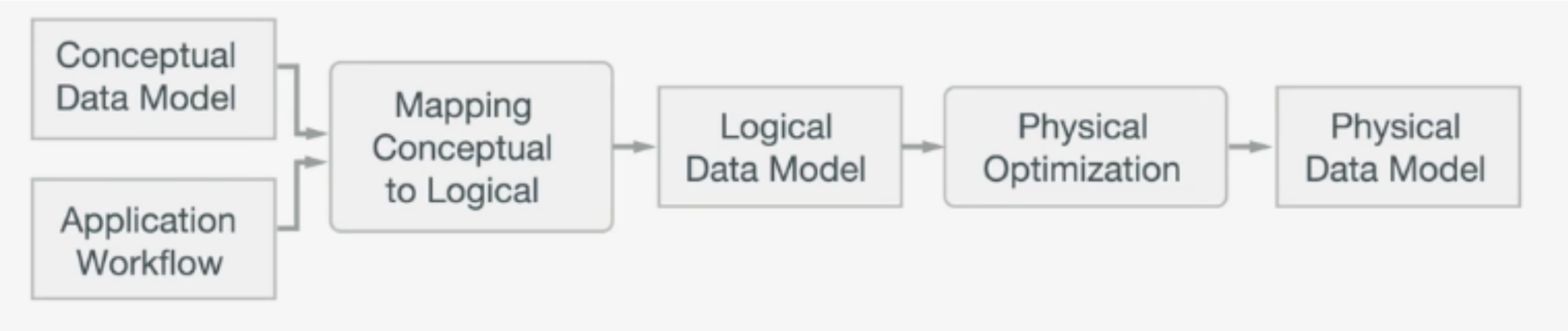
Cassandra Data Model

Objectives

- To design data models suitable for Cassandra’s database design patterns, including a data modeling process and notation
- To optimize the logical and physical data model

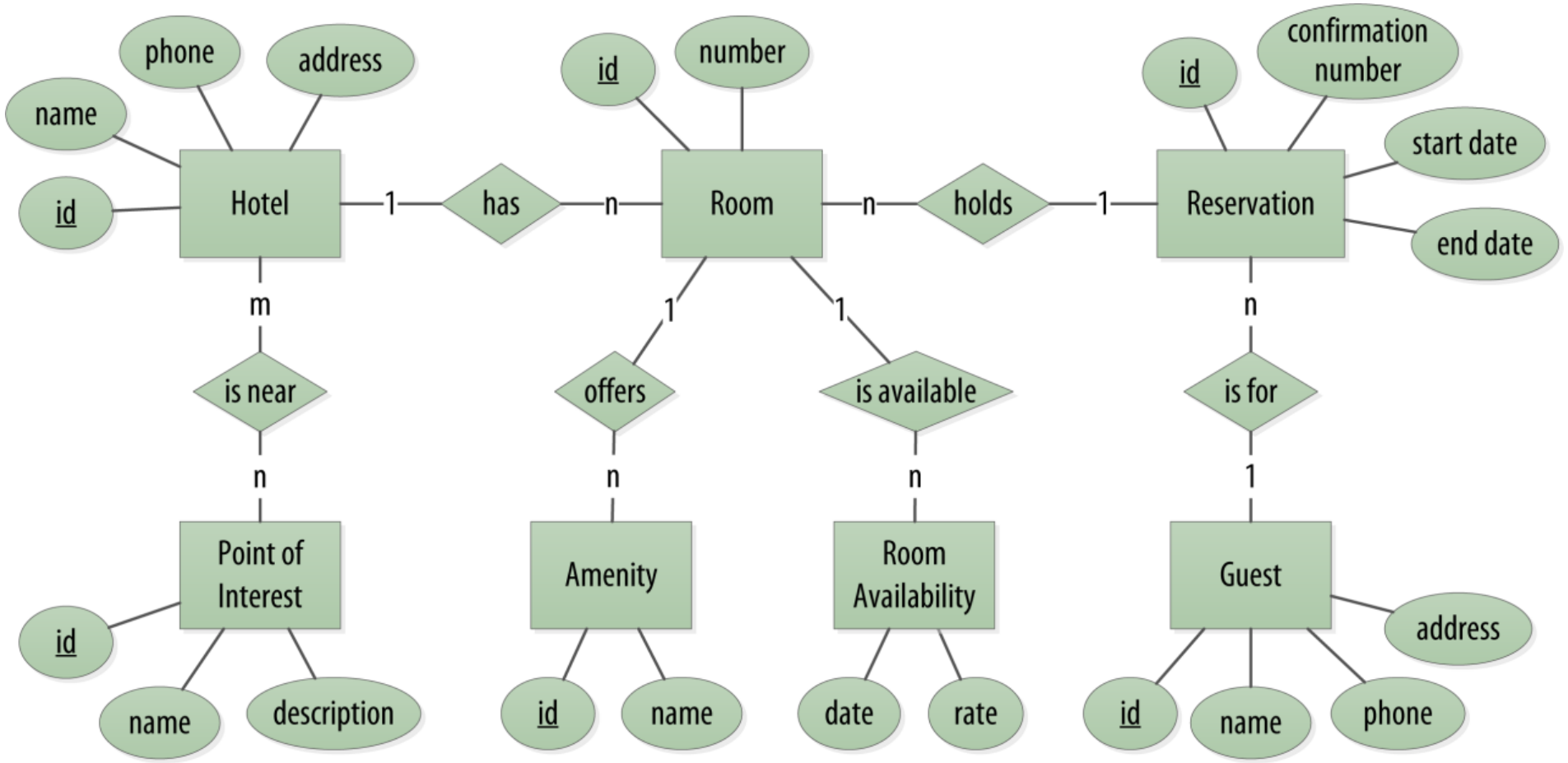
Approach to Cassandra Data Modeling



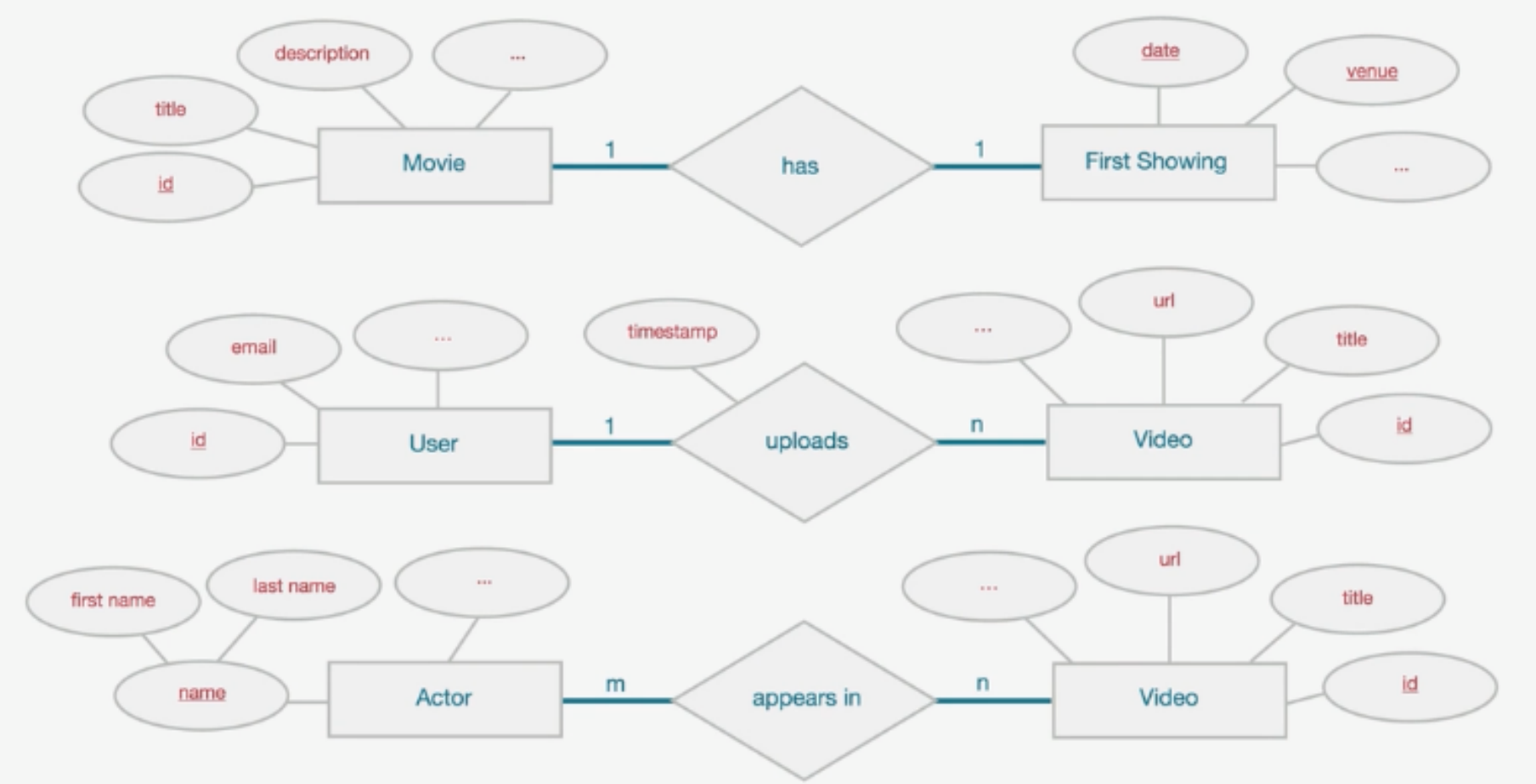
Study Design

- Our conceptual domain includes hotels, guests that stay in the hotels, a collection of rooms for each hotel, the rates and availability of those rooms, and a record of reservations booked for guests. Hotels typically also maintain a collection of “points of interest,” which are parks, museums, shopping galleries, monuments, or other places near the hotel that guests might want to visit during their stay. Both hotels and points of interest need to maintain geolocation data so that they can be found on maps for mashups, and to calculate distances

Hotel ERD

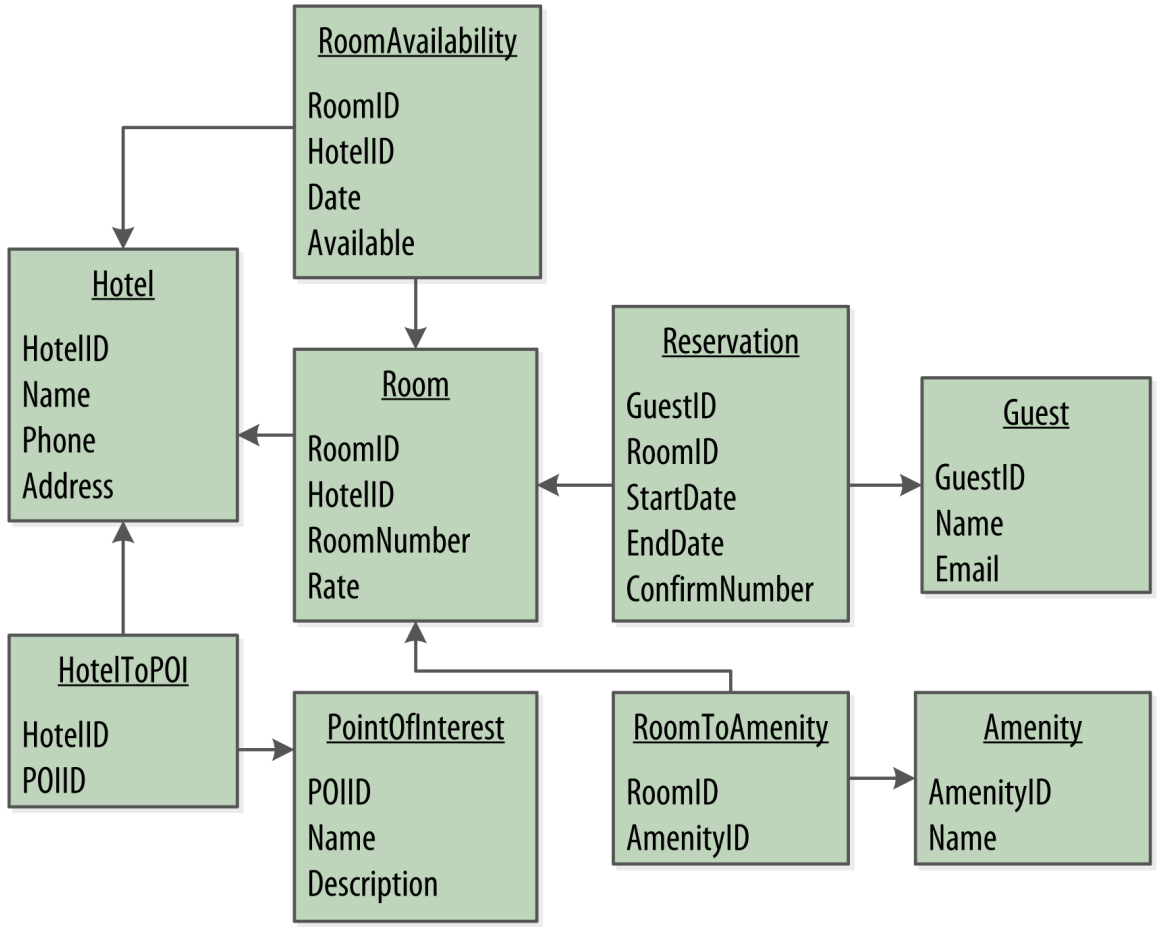

phone
address
name
id
Hotel
1
has
n
id
number
Room
n
holds
1
id
confirmation number
start date
end date
Reservation
m
is near
n
Point of Interest
id
name
description
1
offers
n
Amenity
id
name
1
is available
n
Room Availability
date
rate
n
is for
1
Guest
address
id
name
phone

Chen Notation



Note: it is based on toy example - Killrvideo

RDBMS Design



Design Differences between RDBMS and Cassandra

	RDBMS	Cassandra
Joins	Yes	No - either do the work on the client side or create a de-normalized second table that represents the join results
Referential integrity	Yes - Specify foreign keys in a table to reference the primary key of a record in another table	No - Supports features such as lightweight transactions and batches, no concept of referential integrity across tables
De-normalization	No, but companies end up de-normalizing data for two common reasons - A. One is performance by avoiding join B. capture the snapshot of data, for example a when an order record it created in CRM, in the order detail customer details and product details are also captured.	Yes Perfectly normal occurrence

Features of Cassandra as improvement from RDMS

- Server-side de-normalization with materialized views
- Query-first design
- Designing for optimal storage
- Sorting is a design decision

Server-side de-normalization with materialized views

- Cassandra provides a feature known as materialized views which allows to create multiple de-normalized views of data based on a base table design
- Cassandra manages materialized views on the server, including the work of keeping the views in sync with the table

Query-first design

- Model the queries and let the data be organized around them.
- Think of the most common query paths your application will use, and then create the tables that you need to support them

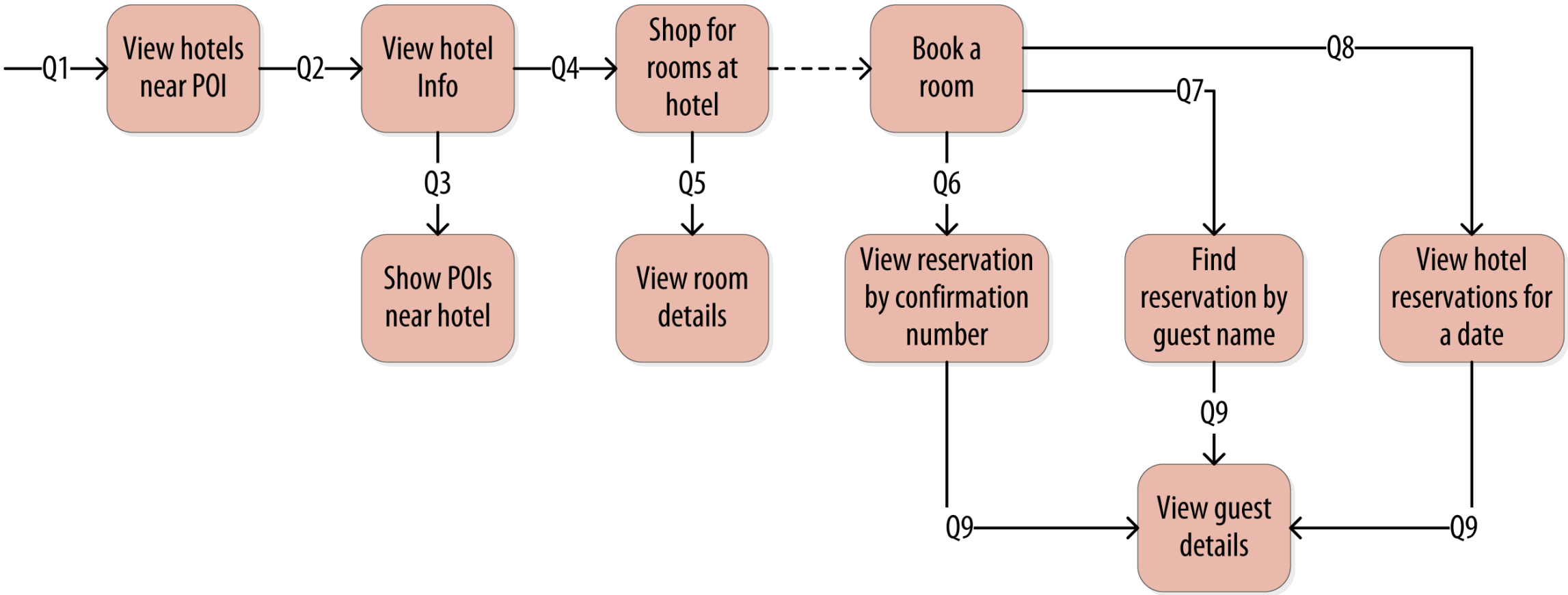
Designing for optimal storage

- In a relational database, it is hardly transparent to the user how tables are stored on disk, and it is rare to hear of recommendations about data modeling based on how the RDBMS might store tables on disk
- Cassandra tables are each stored in separate files on disk, it’s important to keep related columns defined together in the same table

Sorting is a design decision

- In an RDBMS, you can easily change the order in which records are returned to you by using order by in your query. The default sort order is not configurable; by default, records are returned in the order in which they are written
- In Cassandra, it is a design decision. The sort order available on queries is fixed, and is determined entirely by the selection of clustering columns you supply in the create table command. The cql select statement does support order by semantics, but only in the order specified by the clustering columns.

Defining Application Queries



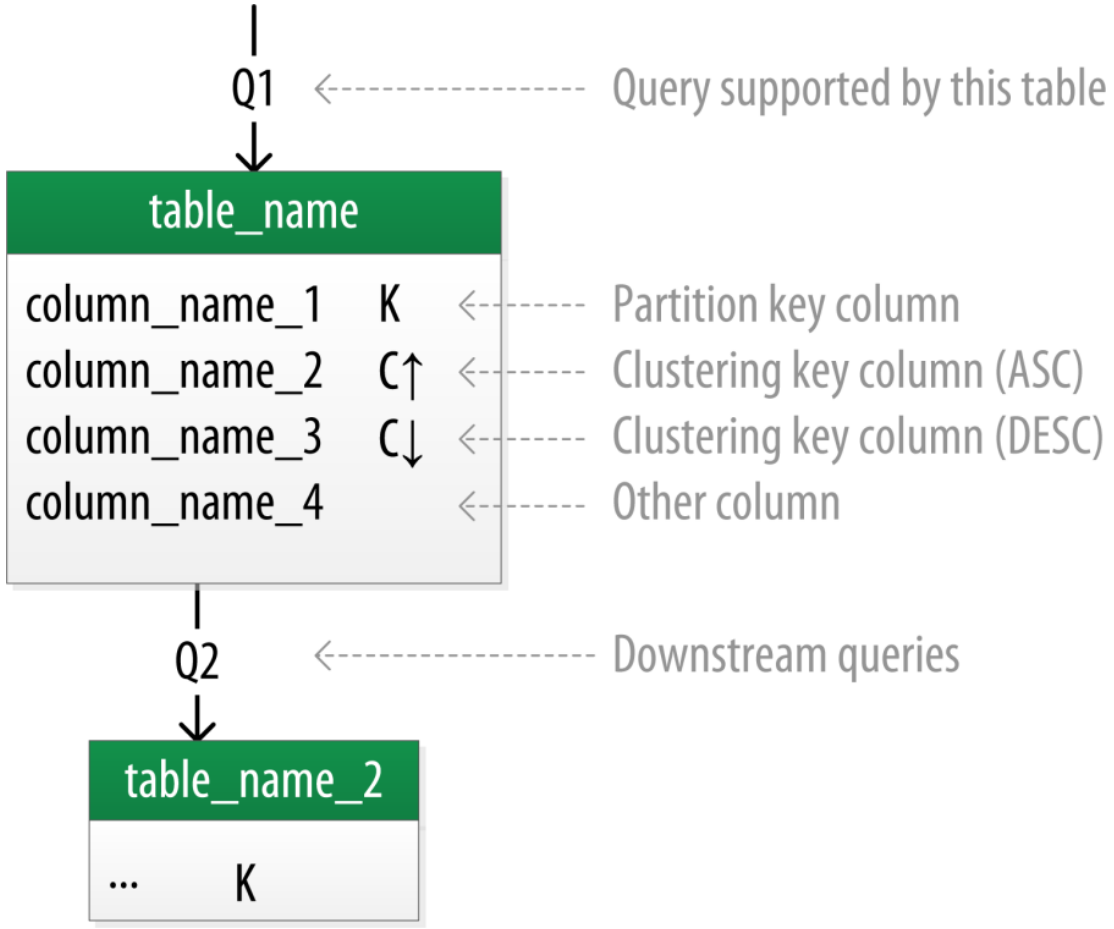
Numbering the queries

- 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.
- 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.

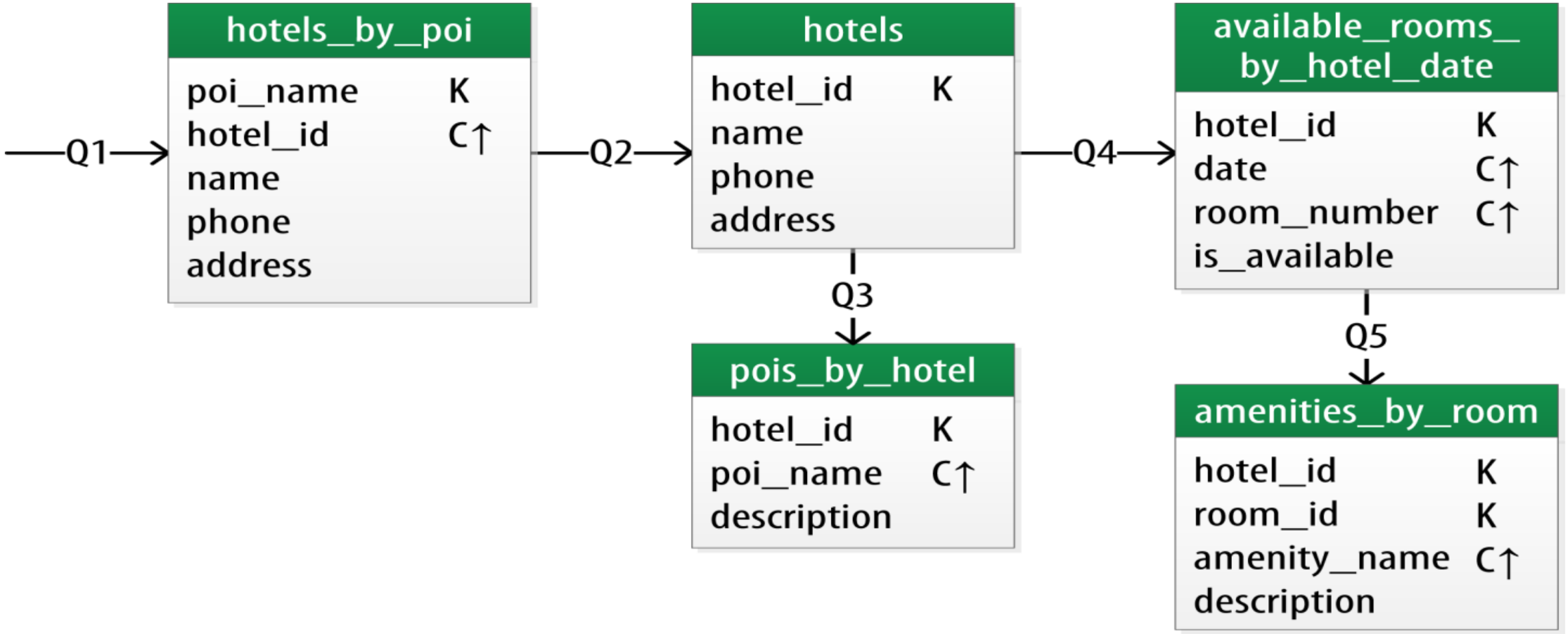
Logical Data Modeling

- Create a logical model containing a table for each query, capturing entities and relationships from the conceptual model
- Identify the primary entity type for which we are querying and use that to start the entity name. If we are querying by attributes of other related entities, we append those to the table name, separated with _by_. For example, hotels_by_poi
- Identify the primary key for the table, adding partition key columns based on the required query attributes, and clustering columns in order to guarantee uniqueness and support desired sort ordering
- Complete each table by adding any additional attributes identified by the query. If any of these additional attributes are the same for every instance of the partition key, we mark the column as static

Chebotko diagrams



Logical Data Model



Make primary keys unique

- Making sure that it defines a unique data element otherwise run the risk of accidentally overwriting data.
- Now for second query (Q2), we'll need a table to get information about a specific hotel. One approach would have been to put all of the attributes of a hotel in the hotels_by_poi table, but we chose to add only those attributes that were required by our application workflow
- We already have the hotel_id from Q1, we use that as our reference to the hotel we're looking for. Therefore our second table is just called hotels

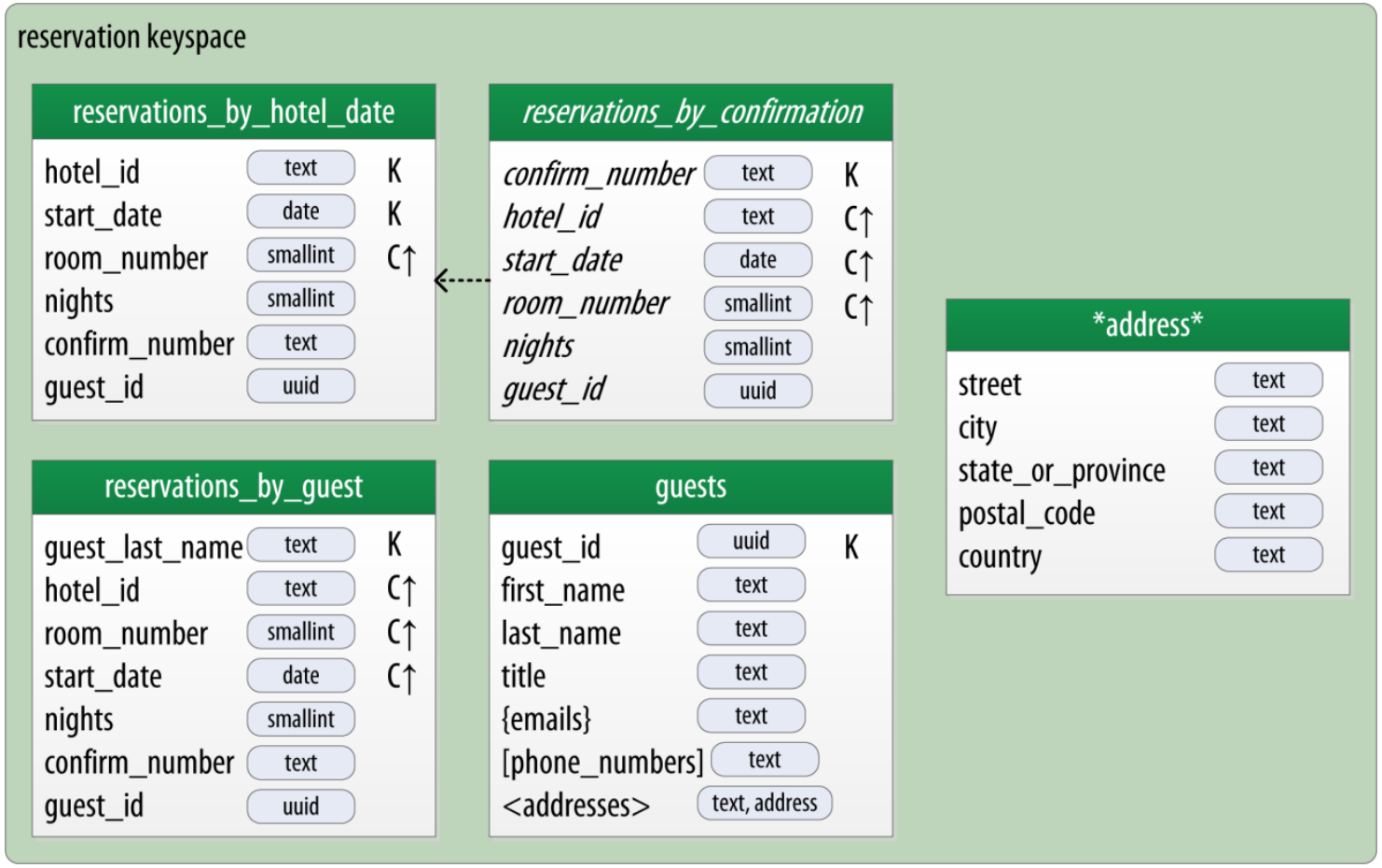
Using unique identifiers as references

- Use unique IDs to uniquely reference elements, and to use these uuids as references in tables representing other entities. Helps to minimize coupling between different entity types
- This may prove especially helpful if you are using a microservice architectural style for your application, in which there are separate services responsible for each entity type.
- Q3 is just a reverse of Q1—looking for points of interest near a hotel, rather than hotels near a point of interest. We add the point of interest name as a clustering key to guarantee uniqueness.
- To support query Q4 to help our user find available rooms at a selected hotel for the nights they are interested in staying. Involves both a start date and an end date. Because we're querying over a range instead of a single date, we know that we'll need to use the date as a clustering key. Use the hotel_id as a primary key to group room data for each hotel on a single date

Searching over a range

- Use clustering columns to store attributes that you need to access in a range query
- In order to round out the shopping portion of our data model, we add the amenities_by_room table to support Q5
- Allows user to view the amenities of one of the rooms that is available for the desired stay dates.

Hotel Reservation Data Model



Evaluating and refining

- Calculating partition size … no of cell values
- Calculating Size on Disk … byte size of the partition

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
- Recommended number of values hundreds of thousands

$$N_v = \ N_r \times (N_c - N_{pk} - N_s) + N_s$$

- N_v: the number of cell values
- N_r: the number of of records for the partition key.
- N_s: the number of static columns
- N_c: the number of columns
- N_{pk}: the number of primary key columns

Calculating Size on Disk

Size of the value

Size of the key

$$\sum_i sizeOf(c_{k_i}) + \sum_j sizeOf(c_{s_j}) + N_r \times \sum_k \left(sizeOf(c_{r_k}) + \sum_l sizeOf(c_{c_l}) \right) + 8 \times N_v$$

Recommendation for Partition Size: 100's MB

Cc – clustering keys
Cr – non primary keys except static column
Cs – static column
Ck – partition key
Nv – number of cell values calculated in the slide before

Schema

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

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

Schema

```
CREATE TABLE reservation.reservations_by_hotel_date (
    hotel_id text,
    start_date date,
    end_date date,
    room_number smallint,
    confirm_number text,
    guest_id uuid,
    PRIMARY KEY ((hotel_id, start_date), room_number)
) WITH comment = 'Q7. Find reservations by hotel and date';

CREATE MATERIALIZED VIEW reservation.reservations_by_confirmation AS
    SELECT * FROM reservation.reservations_by_hotel_date
    WHERE confirm_number IS NOT NULL and hotel_id IS NOT NULL and
        start_date IS NOT NULL and room_number IS NOT NULL
    PRIMARY KEY (confirm_number, hotel_id, start_date, room_number);
```

Schema

```
CREATE TABLE reservation.reservations_by_guest (
    guest_last_name text,
    hotel_id text,
    start_date date,
    end_date date,
    room_number smallint,
    confirm_number text,
    guest_id uuid,
    PRIMARY KEY ((guest_last_name), hotel_id)
) WITH comment = 'Q8. Find reservations by guest name';
```

Schema

```
CREATE TABLE reservation.guests (
    guest_id uuid PRIMARY KEY,
    first_name text,
    last_name text,
    title text,
    emails set<text>,
    phone_numbers list<text>,
    addresses map<text, frozen<address>>,
    confirm_number text
) WITH comment = 'Q9. Find guest by ID';
```

Exercise

Physical Optimization

Two Designs - which is better?

Model 1

videos_by_user	
user_id	K
uploaded_timestamp	C↓
video_id	C↑
title	
type	
{tags}	
<preview_thumbnails>	

Model 2

videos_by_user	
user_id	K
uploaded_timestamp	C↓
type	C↑
title	C↑
video_id	C↑
{tags}	
<preview_thumbnails>	

Background

- The average user uploads about 15 videos
- Very active users—especially if they run a video channel—would have approximately 500 videos
- An artificial limit on the number of videos a user can upload is set to 40,000

The data size for each of the columns

Column Name	Data Size
userid	16 bytes
uploaded_timestamp	8 bytes
videoid	16 bytes
title	55 bytes (average)
type	12 bytes (average)
tags	30 bytes (average)
preview_thumbnails	2,340 bytes (average)

Number of Cell Values

Model 1

Total number of columns = 7
Number of primary key columns = 3
Number of static columns = 0

- **Average User** = 15 CQL rows
  15 * (7 - 3 - 0) + 0 = 60 values per partition
- **Active User** = 500 CQL rows
  500 * (7 - 3 - 0) + 0 = 2,000 values per partition
- **Worst Case** = 40,000 CQL rows
  40000 * (7 - 3 - 0) + 0 = 160,000 values per partition

Model 2

Total number of columns = 7
Number of primary key columns = 5
Number of static columns = 0

- **Average User** = 15 CQL rows
  15 * (7 - 5 - 0) + 0 = 30 values per partition
- **Active User** = 500 CQL rows
  500 * (7 - 5 - 0) + 0 = 1,000 values per partition
- **Worst Case** = 40,000 CQL rows
  40000 * (7 - 5 - 0) + 0 = 80,000 values per partition

Partition Size

Model 1

- **Co-worker #1 Table**
  Total size of partition columns = 16 bytes
  Total size of static columns = 0 bytes
  Total size of clustering columns = 8 + 16 = 24 bytes
  Total size of regular columns = (55 + 24) + (12 + 24) + (30 + 24) + (2340 + 24) = 2,533 bytes
- **Average User** = 15 CQL rows, 60 values per partition
  16 + 0 + (15 * 2533) + (8 * 60) = 38,491 bytes
- **Active User** = 500 CQL rows, 2,000 per partition
  16 + 0 + (500 * 2533) + (8 * 2000) = 1,282,516 bytes or ~1.22 MB
- **Worst Case** = 40,000 CQL rows, 160,000 values per partition
  16 + 0 + (40000 * 2533) + (8 * 160000) = 102,600,016 bytes or ~97.85 MB

Model 2:

- Total size of partition columns = 16 bytes
  Total size of static columns = 0 bytes
  Total size of clustering columns = 8 + 16 + 12 + 55 = 91 bytes
  Total size of regular columns = (30 + 91) + (2340 + 91) = 2,552 bytes
- **Average User** = 15 CQL rows, 30 values per partition
  16 + 0 + (15 * 2552) + (8 * 30) = 38,536 bytes
- **Active User** = 500 CQL rows, 1,000 per partition
  16 + 0 + (500 * 2552) + (8 * 1000) = 1,284,016 bytes or ~1.22 MB
- **Worst Case** = 40,000 CQL rows, 80,000 values per partition
  16 + 0 + (40000 * 2552) + (8 * 80000) = 102,720,016 bytes or ~97.96 MB

Results

	No of values		Size of partitions	
	Model 1	Model 2	Model 1	Model 2
Average User (15)	60	30	38,491 bytes	38,536 bytes
Active User (500)	2000	1000	1,282,516 bytes or ~1.22 MB	1,284,016 bytes or ~1.22 MB
Most Active User (40,000)	160,000	80,000	102,600,016 bytes or ~97.85 MB	102,720,016 bytes or ~97.96 MB

Conclusion: Both model 1 and model 2 are well with recommended limits on number of cell values and size of partition, while model 2 is slightly bigger in terms of partition size, but offers some extra capability of additional filters on the clustering columns. Thus model has slight advantage in terms of functionality.