# DataStax Developer Day

# Cassandra Data Modeling

DATASTAX®

Cassandra Data Modeling

DATASTAX®

# SQL (relational) vs. CQL (Cassandra)

# Structuring Your Database

- Normalization: To reduce data redundancy and increase data integrity.

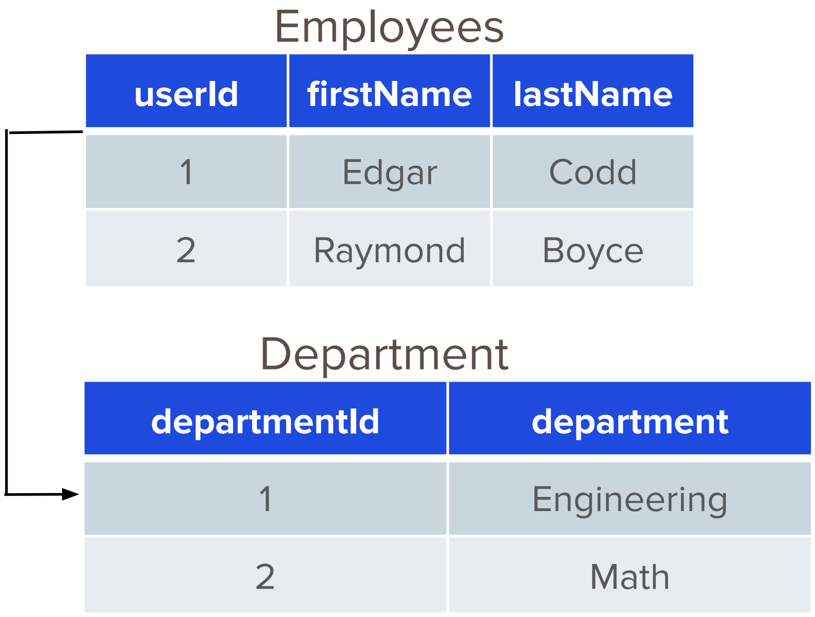
- Denormalization: Must be done in read heavy workloads to increase performance

# Normalization

- Structuring a relational database
- Normal forms (3NF max)
- Why?
  - Reduce data redundancy
  - Increase data integrity.

# Relational Data Models

- Multiple normal forms
  - most do not go beyond 3NF
- Foreign Keys
- Joins



Employees

| userId | firstName | lastName |
|---|---|---|
| 1 | Edgar | Codd |
| 2 | Raymond | Boyce |

Department

| departmentId | department |
|---|---|
| 1 | Engineering |
| 2 | Math |

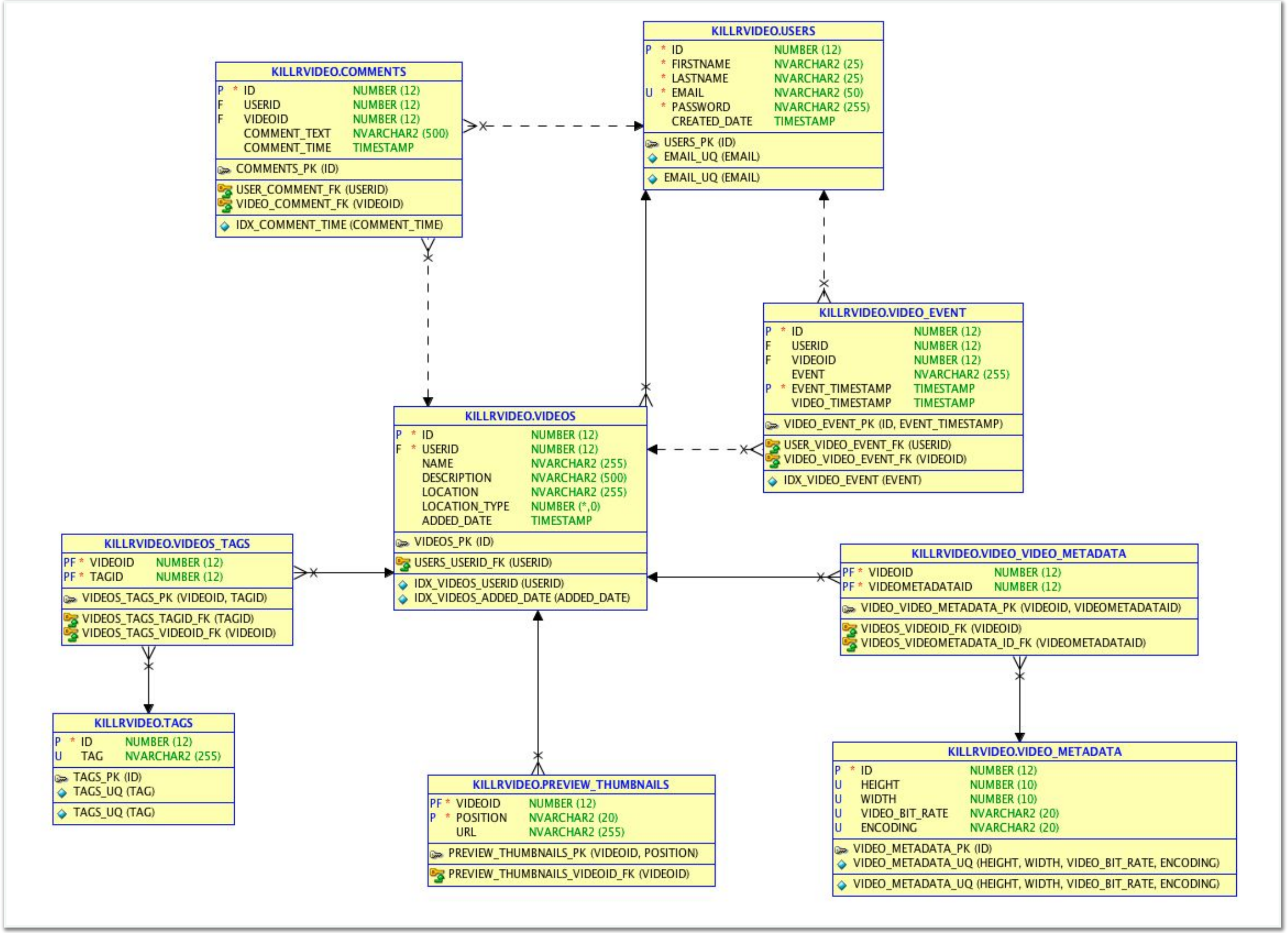
# Relational Modeling

- Create entity table
- Add constraints
- Index fields
- Foreign Key relationships

```
CREATE TABLE users (
  id          number(12) NOT NULL ,
  firstname    nvarchar2(25) NOT NULL ,
  lastname     nvarchar2(25) NOT NULL,
  email       nvarchar2(50) NOT NULL,
  password     nvarchar2(255) NOT NULL,
  created_date timestamp(6),
  PRIMARY KEY (id),
  CONSTRAINT email_uq UNIQUE (email)
);


-- Users by email address index
CREATE INDEX idx_users_email ON users (email);
```

```
CREATE TABLE videos (
  id number(12),
  userid number(12) NOT NULL,
  name nvarchar2(255),
  description nvarchar2(500),
  location nvarchar2(255),
  location_type int,
  added_date timestamp,
  CONSTRAINT users_userid_fk
    FOREIGN KEY (userid)
    REFERENCES users (Id) ON DELETE CASCADE,
  PRIMARY KEY (id)
);
```



**KILLRVIDEO.COMMENTS**

| | | |
|---|---|---|
| P * | ID | NUMBER (12) |
| F | USERID | NUMBER (12) |
| F | VIDEOID | NUMBER (12) |
| | COMMENT_TEXT | NVARCHAR2 (500) |
| | COMMENT_TIME | TIMESTAMP |

COMMENTS_PK (ID)

USER_COMMENT_FK (USERID)
VIDEO_COMMENT_FK (VIDEOID)

IDX_COMMENT_TIME (COMMENT_TIME)

**KILLRVIDEO.USERS**

| | | |
|---|---|---|
| P * | ID | NUMBER (12) |
| * | FIRSTNAME | NVARCHAR2 (25) |
| * | LASTNAME | NVARCHAR2 (25) |
| U * | EMAIL | NVARCHAR2 (50) |
| * | PASSWORD | NVARCHAR2 (255) |
| | CREATED_DATE | TIMESTAMP |

USERS_PK (ID)
EMAIL_UQ (EMAIL)

EMAIL_UQ (EMAIL)

**KILLRVIDEO.VIDEO_EVENT**

| | | |
|---|---|---|
| P * | ID | NUMBER (12) |
| F | USERID | NUMBER (12) |
| F | VIDEOID | NUMBER (12) |
| | EVENT | NVARCHAR2 (255) |
| P * | EVENT_TIMESTAMP | TIMESTAMP |
| | VIDEO_TIMESTAMP | TIMESTAMP |

VIDEO_EVENT_PK (ID, EVENT_TIMESTAMP)

USER_VIDEO_EVENT_FK (USERID)
VIDEO_VIDEO_EVENT_FK (VIDEOID)

IDX_VIDEO_EVENT (EVENT)

**KILLRVIDEO.VIDEOS**

| | | |
|---|---|---|
| P * | ID | NUMBER (12) |
| F * | USERID | NUMBER (12) |
| | NAME | NVARCHAR2 (255) |
| | DESCRIPTION | NVARCHAR2 (500) |
| | LOCATION | NVARCHAR2 (255) |
| | LOCATION_TYPE | NUMBER (*,0) |
| | ADDED_DATE | TIMESTAMP |

VIDEOS_PK (ID)

USERS_USERID_FK (USERID)

IDX_VIDEOS_USERID (USERID)
IDX_VIDEOS_ADDED_DATE (ADDED_DATE)

**KILLRVIDEO.VIDEOS_TAGS**

| | | |
|---|---|---|
| PF * | VIDEOID | NUMBER (12) |
| PF * | TAGID | NUMBER (12) |

VIDEOS_TAGS_PK (VIDEOID, TAGID)

VIDEOS_TAGS_TAGID_FK (TAGID)
VIDEOS_TAGS_VIDEOID_FK (VIDEOID)

**KILLRVIDEO.VIDEO_VIDEO_METADATA**

| | | |
|---|---|---|
| PF * | VIDEOID | NUMBER (12) |
| PF * | VIDEOMETADATAID | NUMBER (12) |

VIDEO_VIDEO_METADATA_PK (VIDEOID, VIDEOMETADATAID)

VIDEOS_VIDEOID_FK (VIDEOID)
VIDEOS_VIDEOMETADATA_ID_FK (VIDEOMETADATAID)

**KILLRVIDEO.TAGS**

| | | |
|---|---|---|
| P * | ID | NUMBER (12) |
| U | TAG | NVARCHAR2 (255) |

TAGS_PK (ID)
TAGS_UQ (TAG)

TAGS_UQ (TAG)

**KILLRVIDEO.PREVIEW_THUMBNAILS**

| | | |
|---|---|---|
| PF * | VIDEOID | NUMBER (12) |
| P * | POSITION | NVARCHAR2 (20) |
| | URL | NVARCHAR2 (255) |

PREVIEW_THUMBNAILS_PK (VIDEOID, POSITION)

PREVIEW_THUMBNAILS_VIDEOID_FK (VIDEOID)

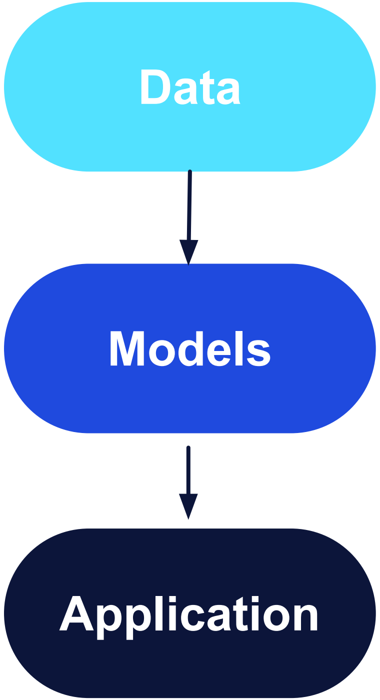
**KILLRVIDEO.VIDEO_METADATA**

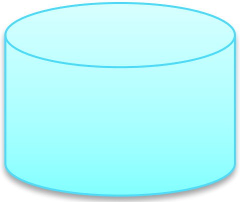
| | | |
|---|---|---|
| P * | ID | NUMBER (12) |
| U | HEIGHT | NUMBER (10) |
| U | WIDTH | NUMBER (10) |
| U | VIDEO_BIT_RATE | NVARCHAR2 (20) |
| U | ENCODING | NVARCHAR2 (20) |

VIDEO_METADATA_PK (ID)
VIDEO_METADATA_UQ (HEIGHT, WIDTH, VIDEO_BIT_RATE, ENCODING)

VIDEO_METADATA_UQ (HEIGHT, WIDTH, VIDEO_BIT_RATE, ENCODING)

# Relational Modeling





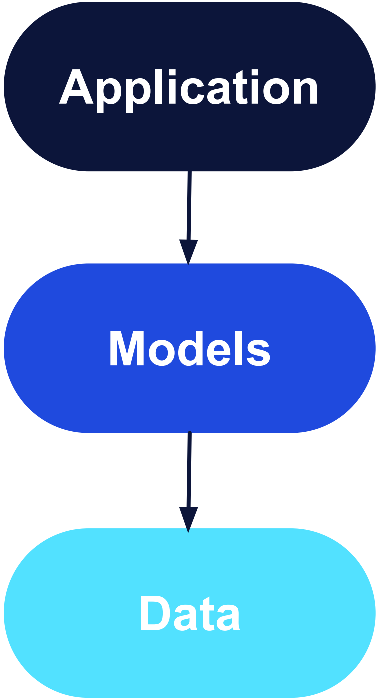
Employees

| userId | firstName | lastName |
|---|---|---|
| 1 | Edgar | Codd |
| 2 | Raymond | Boyce |

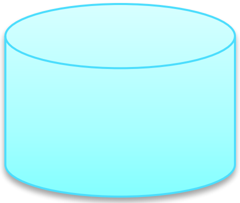
Department

| departmentId | department |
|---|---|
| 1 | Engineering |
| 2 | Math |

# Cassandra Modeling



| id | firstName | lastName | department |
|---|---|---|---|
| 1 | Edgar | Codd | Engineering |
| 2 | Raymond | Boyce | Math |



# Denormalization - Why?

- Improve read performance of a database
- Reduce write performance
  - Adding redundant copies of data

# CQL vs SQL

- No joins
- Limited aggregations

```
SELECT e.First, e.Last, d.Dept
FROM Department d, Employees e
WHERE ‘Codd’ = e.Last
AND e.deptId = d.id
```

Employees

| userId | firstName | lastName |
|---|---|---|
| 1 | Edgar | Codd |
| 2 | Raymond | Boyce |

Department

| departmentId | department |
|---|---|
| 1 | Engineering |
| 2 | Math |

# Applying Denormalization

- Combine table columns into a single view
- Eliminate the need for joins
- Queries are concise and easy to understand

Employees

| id | firstName | lastName | department |
|---|---|---|---|
| 1 | Edgar | Codd | Engineering |
| 2 | Raymond | Boyce | Math |

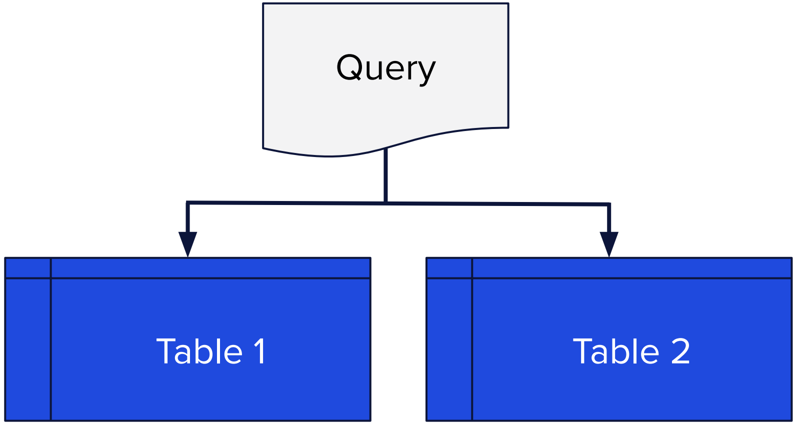
```
SELECT First, Last, Dept
FROM employees
WHERE id = ‘1’
```

# Denormalization in Apache Cassandra

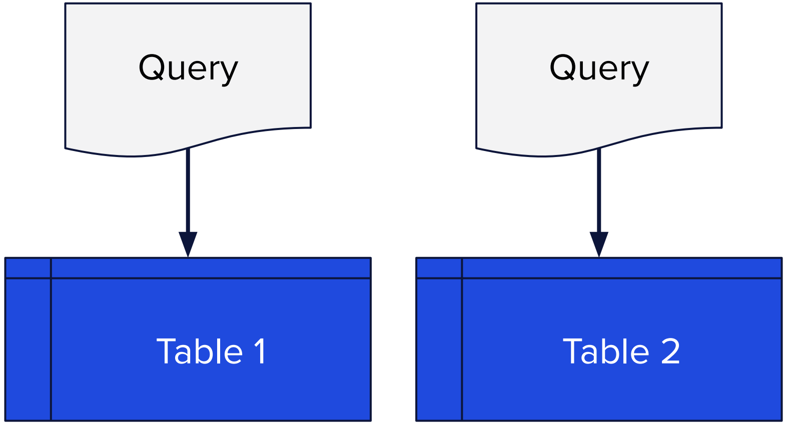
- Denormalization of tables in Apache Cassandra is absolutely critical.
- The biggest take away is to think about your queries first.
- There are no JOINS in Apache Cassandra.

# Queries in Relational vs NoSQL Databases

- In a relational database, one query can access and join data from multiple tables



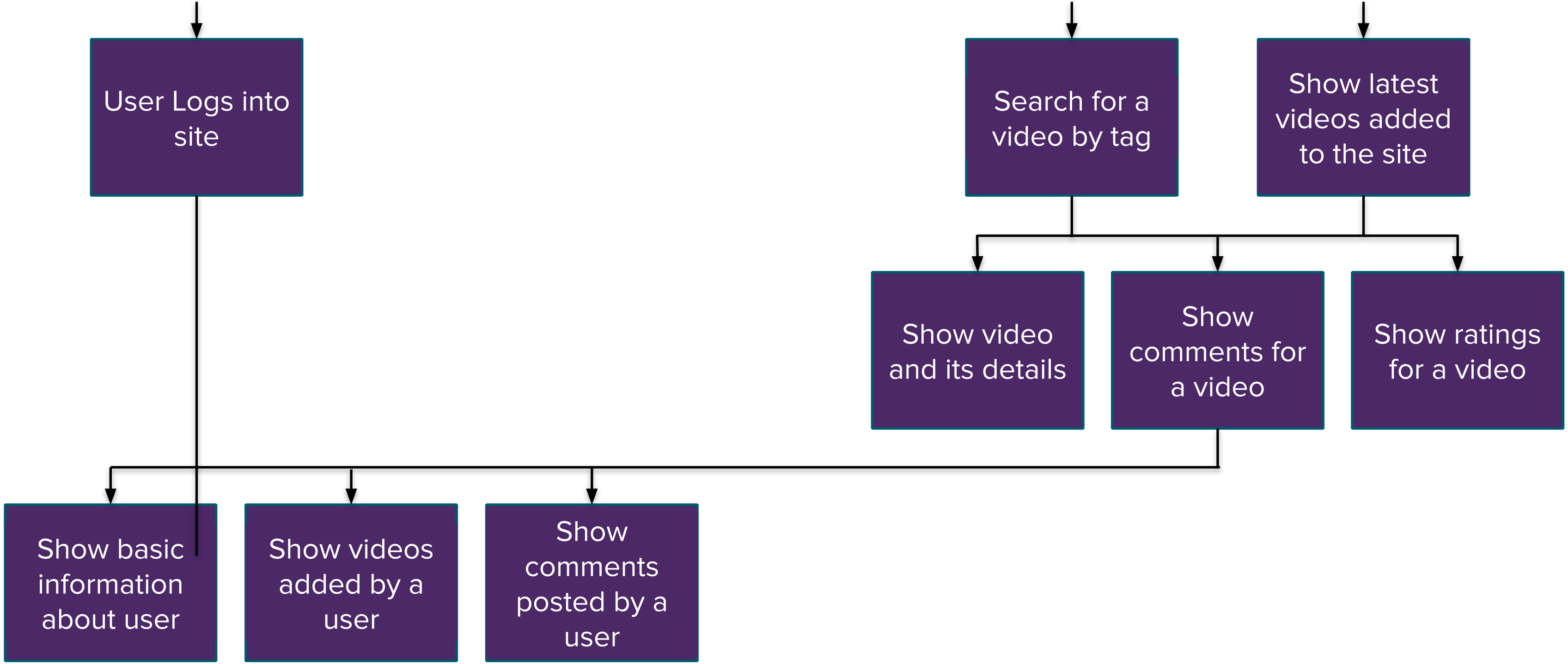
- In Apache Cassandra, you cannot join data, queries can only access data from one table



# Modeling Queries

- What are your application’s workflows?
- Knowing your queries in advance is CRITICAL
- Different from RDBMS because I can’t just JOIN or create a new indexes to support new queries
- One table per one query

# Some Application Workflows in KillrVideo



User Logs into site

Search for a video by tag

Show latest videos added to the site

Show video and its details

Show comments for a video

Show ratings for a video

Show basic information about user

Show videos added by a user

Show comments posted by a user

# Some Queries in KillrVideo to Support Workflows

## Users

| | |
|---|---|
| User Logs into site | Find user by email address |
| Show basic information about user | Find user by id |

## Comments

| | |
|---|---|
| Show comments for a video | Find comments by video (latest first) |
| Show comments posted by a user | Find comments by user (latest first) |

## Ratings

| | |
|---|---|
| Show ratings for a video | Find ratings by video |

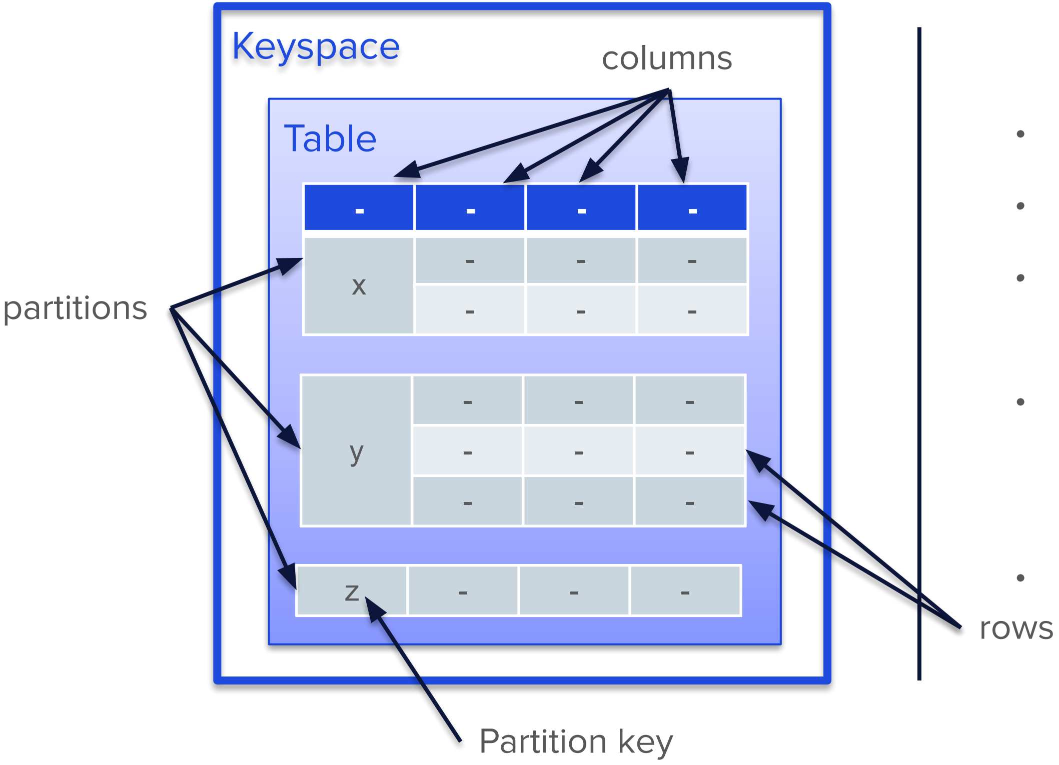
## Cassandra Data Modeling

# Denormalization Mind Shift

# Cassandra Data Modeling Principles

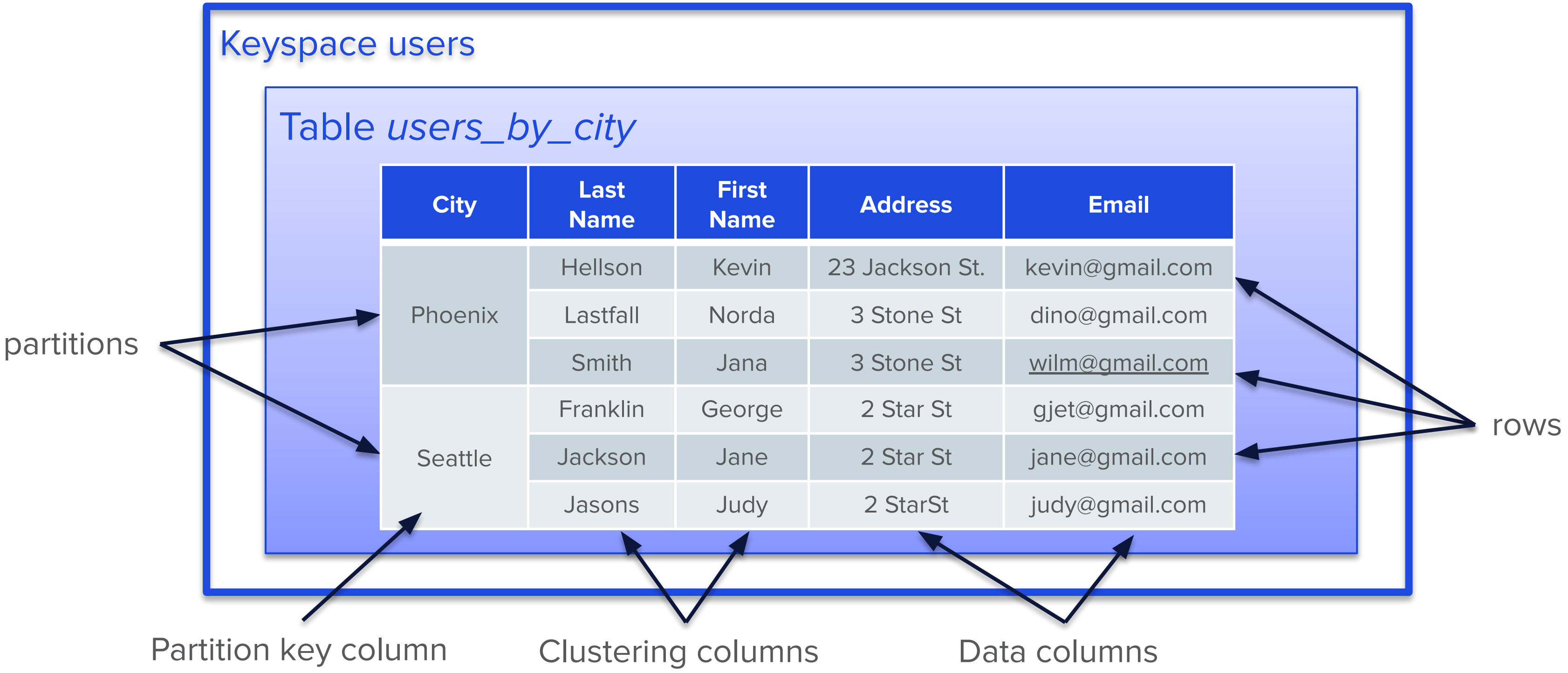
- Design tables around queries
- Use partition key column(s) to group data you would like to be able to get in a single query
- Use clustering columns to guarantee unique rows and control sort order
- Use additional columns to provide the details you need
  - Denormalization - including data that might have been joined from elsewhere in a relational model

# Cassandra Structure - Partition



- Tabular data model, with one twist
- *Keyspaces* contain *tables*
- *Tables* are organized in *rows* and *columns*
- Groups of related rows called *partitions* are stored together on the same node (or nodes)
- Each row contains a *partition key*
  - One or more columns that are hashed to determine which node(s) store that data

# Example Data – Users organized by city



Keyspace users

Table *users_by_city*

| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Phoenix | Hellson | Kevin | 23 Jackson St. | kevin@gmail.com |
| | Lastfall | Norda | 3 Stone St | dino@gmail.com |
| | Smith | Jana | 3 Stone St | wilm@gmail.com |
| Seattle | Franklin | George | 2 Star St | gjet@gmail.com |
| | Jackson | Jane | 2 Star St | jane@gmail.com |
| | Jasons | Judy | 2 StarSt | judy@gmail.com |

partitions

rows

Partition key column

Clustering columns

Data columns

# Tables Hold Many Partitions

| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Phoenix | Hellson | Kevin | 23 Jackson St. | kevin@gmail.com |
| | Lastfall | Norda | 3 Stone St | dino@gmail.com |
| | Smith | Jana | 3 Stone St | wilm@gmail.com |

Table *users_by_city*

# Tables Hold Many Partitions

| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Seattle | Franklin | George | 2 Star St | gjet@gmail.com |
| | Jackson | Jane | 2 Star St | jane@gmail.com |
| | Jasons | Judy | 2 StarSt | judy@gmail.com |

Table *users_by_city*

| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Phoenix | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |

# Tables Hold Many Partitions

| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Charlotte | Azrael | Chris | 5 Blue St | chris@gmail.com |
| | Stilson | Brainy | 7 Azure ln | brain@gmail.com |
| | Smith | Cristina | 4 Teal Cir | clu@gmail.com |
| | Sage | Grant | 9 Royal St | grant@gmail.com |
| | Seterson | Peter | 2 Navy Ct | peter@gmail.com |

## Table *users_by_city*

| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Phoenix | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| Seattle | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |

# Tables Hold Many Partitions

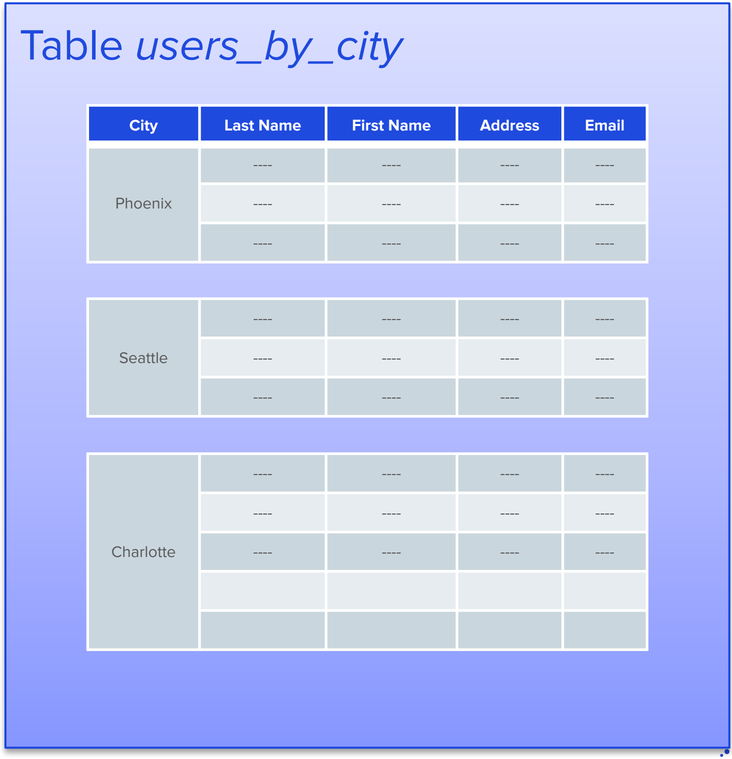
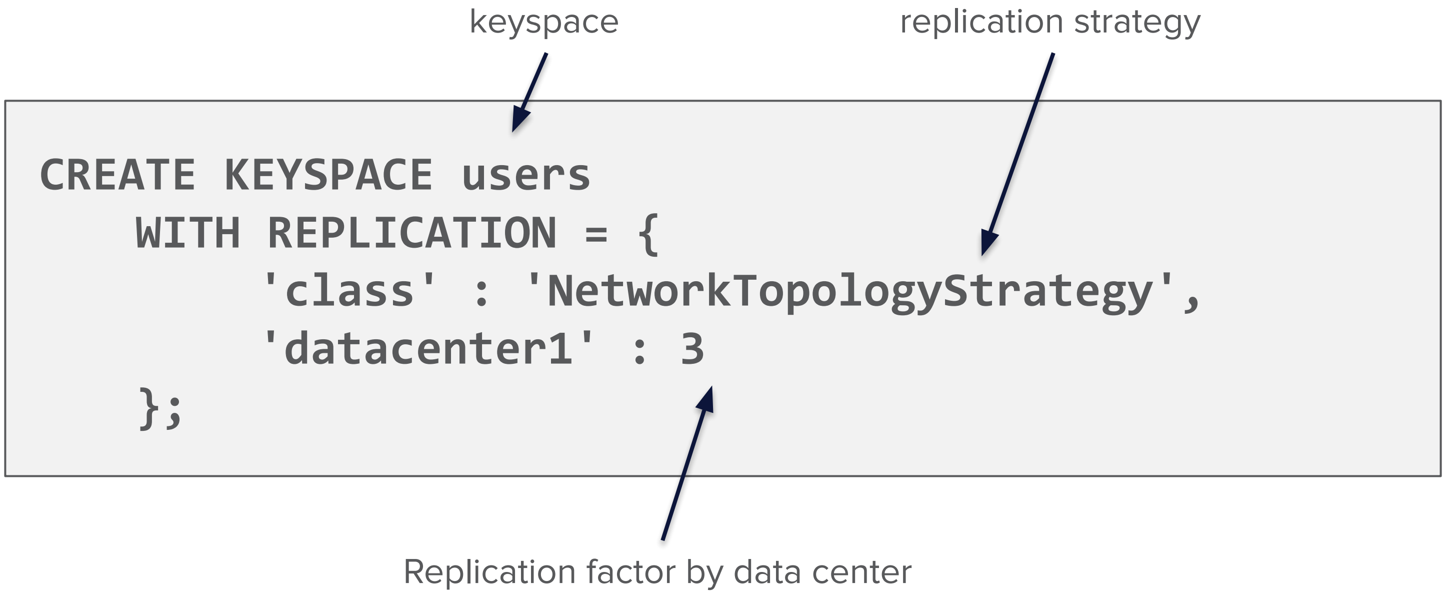


Table *users_by_city*

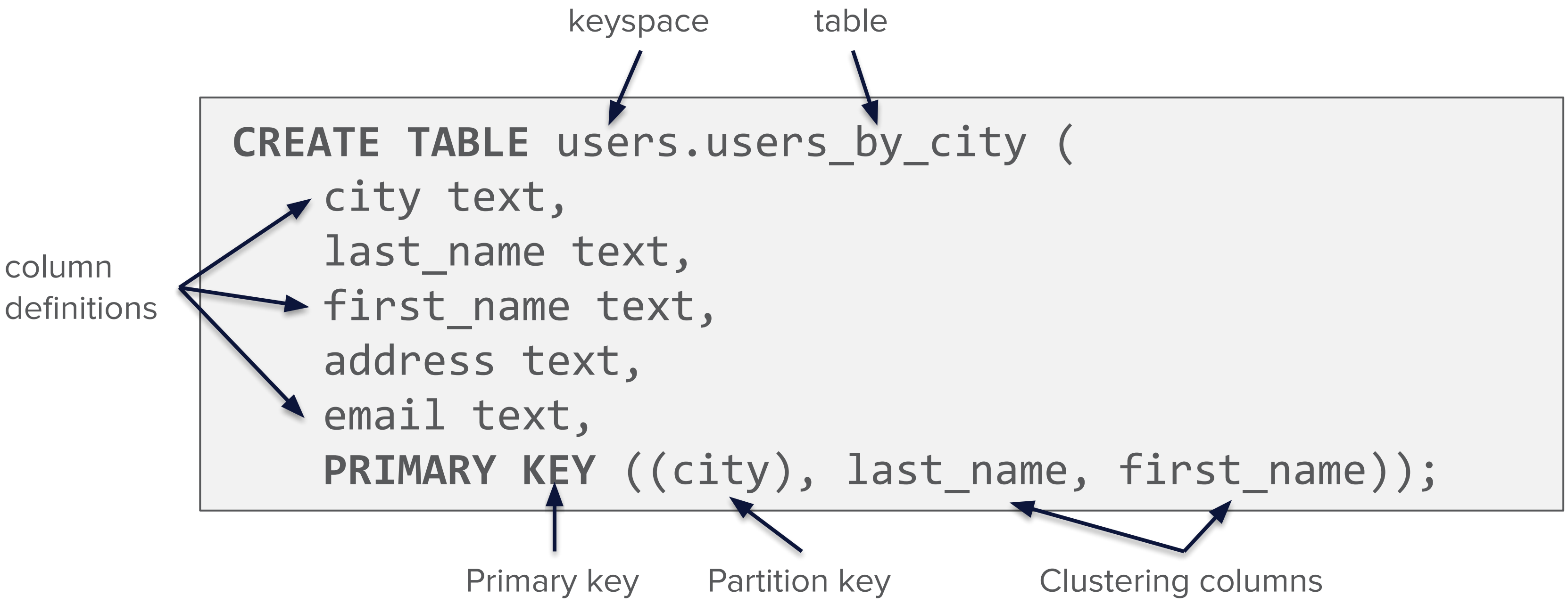
| City | Last Name | First Name | Address | Email |
|---|---|---|---|---|
| Phoenix | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| Seattle | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| Charlotte | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | ---- | ---- | ---- | ---- |
| | | | | |
| | | | | |

# Creating a Keyspace in CQL



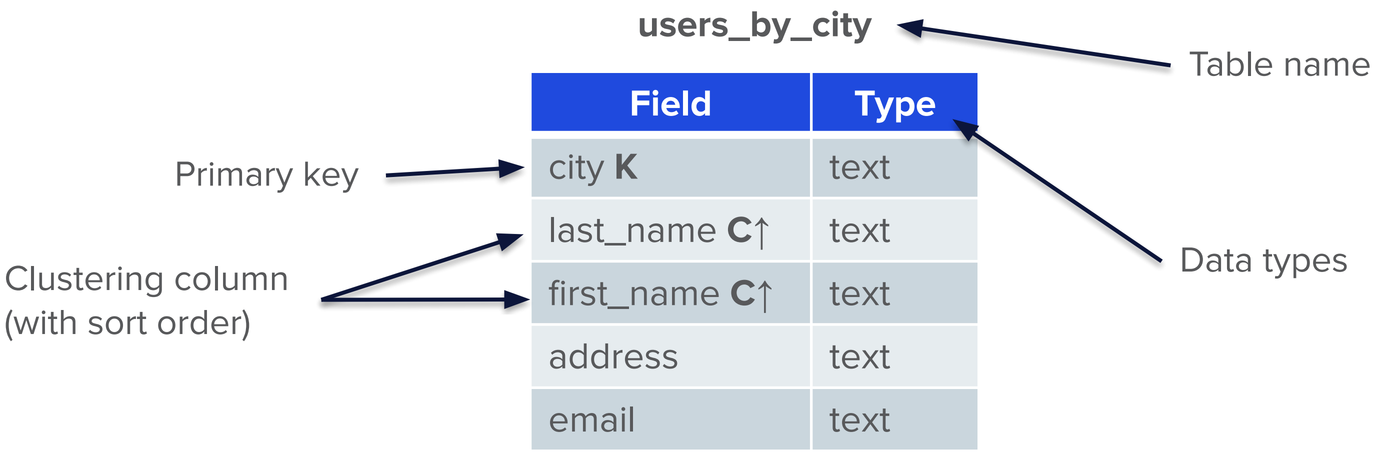
```
CREATE KEYSPACE users
    WITH REPLICATION = {
         'class' : 'NetworkTopologyStrategy',
         'datacenter1' : 3
    };
```

# Creating a Table in CQL



```
CREATE TABLE users.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name));
```

# Cassandra Table Design Notation - Chebotko Diagram



**users_by_city**

| Field | Type |
|---|---|
| city **K** | text |
| last_name **C↑** | text |
| first_name **C↑** | text |
| address | text |
| email | text |

Table name

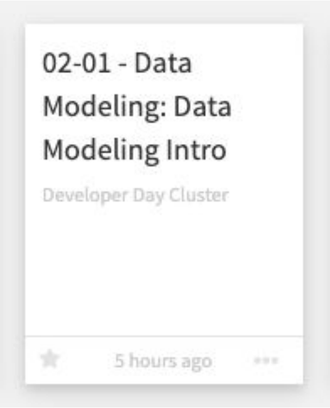
Primary key

Data types

Clustering column
(with sort order)

# Time for an exercise!

## “Data Modeling Intro” Notebook



02-01 - Data Modeling: Data Modeling Intro

Developer Day Cluster

5 hours ago

# Data Modeling – Key Concepts

- Keyspace – contains tables
- Table – contains partitions
- Row – has a primary key and data columns
- Partition – basic unit of storage/retrieval
  - Identified by partition key embedded within primary key
  - Contains one or more rows
- Primary key – intra-table row identifier
  - Consists of partition key and clustering columns
  - Partition key – partition identifier, hashes to partition token
  - Clustering column – intra-partition key for sorting rows within partition

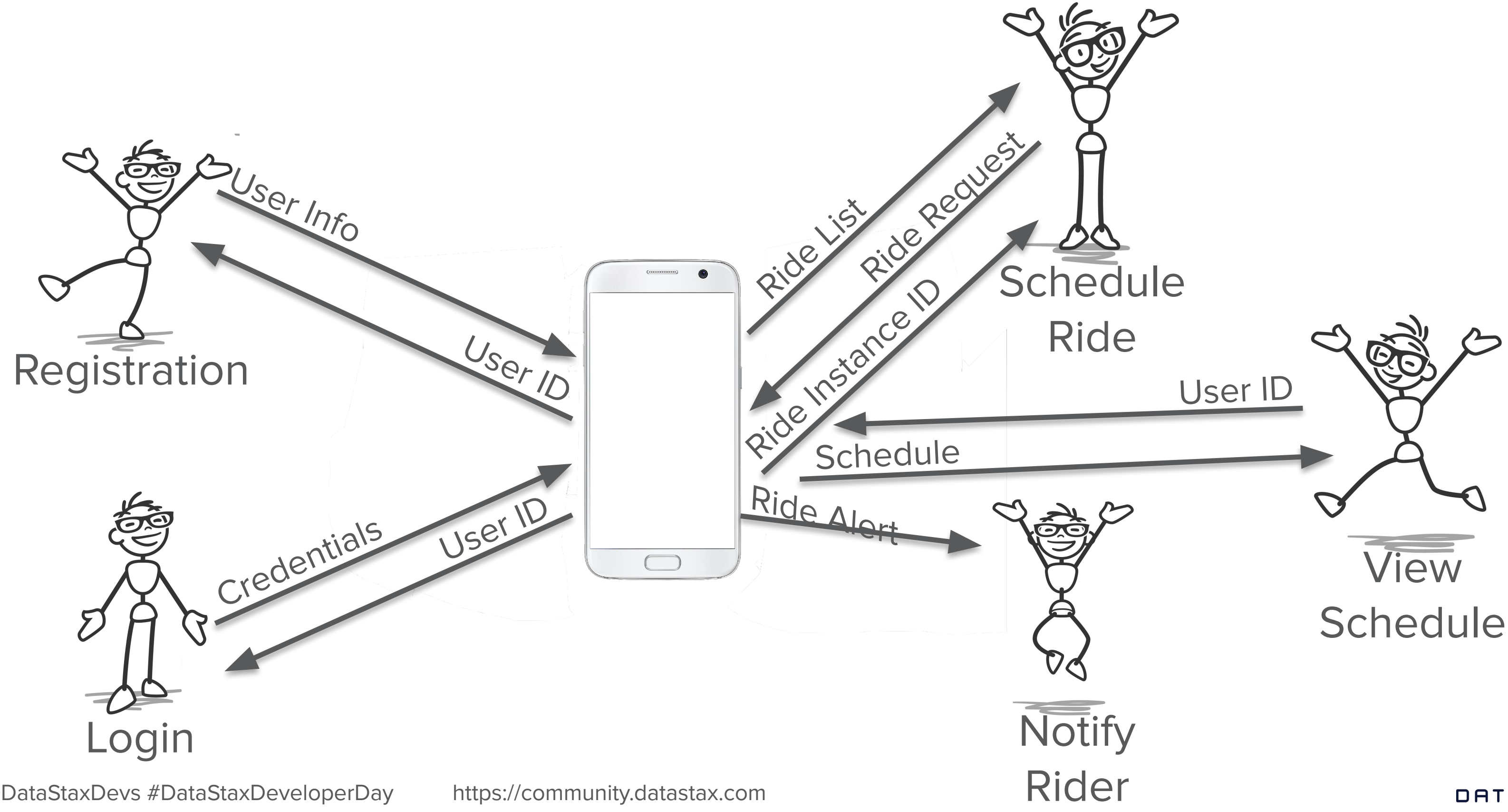
# Welcome to Cassandra-Land

The Theme Park Where You Can Find…

- Distributed & Fault-Tolerant Rides
- Amazing Throughput
- And Fast Response Times

But We Need an App!

# Cassandra-Land Use Cases



# Cassandra-Land Use Cases

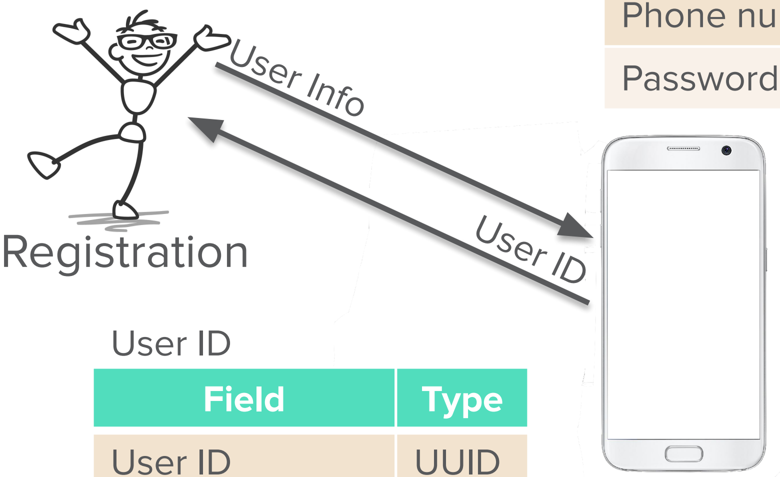
- Creating a Keyspace

```
CREATE KEYSPACE <keyspace name> WITH REPLICATION = {
     'class' : <replication strategy>,
    <datacenter name> : <replication factor>, ... };
```

- For example

```
CREATE KEYSPACE park WITH REPLICATION = {
     'class' : NetworkTopologyStrategy,
    'USWestDC': 3, 'USEastDC’: 3 };
```

# Cassandra-Land Registration Use Case



Registration

User Info

| Field | Type |
|---|---|
| Phone number | text |
| Password | text |

User ID

| Field | Type |
|---|---|
| User ID | UUID |

**users_by_phone_number**

| Field | Type |
|---|---|
| phone_number **K** | text |
| user_id | UUID |
| password | text |

# Cassandra-Land Registration Use Case

- Creating a table

```
CREATE TABLE <keyspace name>.<table name> (
    <field name> <field type>,
    // Add additional field descriptions here
    PRIMARY KEY ( <primary key descriptor> )
);
```

# Cassandra-Land Registration Use Case

- Inserting a row into a table

```
INSERT INTO <keyspace name>.<table name>
    ( <column list> )
    VALUES ( <column values> );
```

# Cassandra-Land Registration Use Case

- Selecting all rows from a table
  - Typically wouldn’t do this in production

```
SELECT * FROM <keyspace name>.<table name>;
```

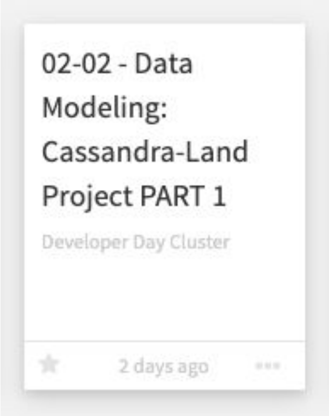
# Cassandra’s Upsert Behavior

- Cassandra does NOT read before writing
- Inserting a row with the same primary key causes an update called an “upsert”
- Similarly, updates to non-existent rows cause an insert
  - Can use a lightweight transaction to prevent an upsert as it does perform a read before writing

```
INSERT INTO keyspace.table IF NOT EXISTS ...
```

# Notebook Data Modeling

# Cassandra-Land Project



02-02 - Data Modeling: Cassandra-Land Project PART 1

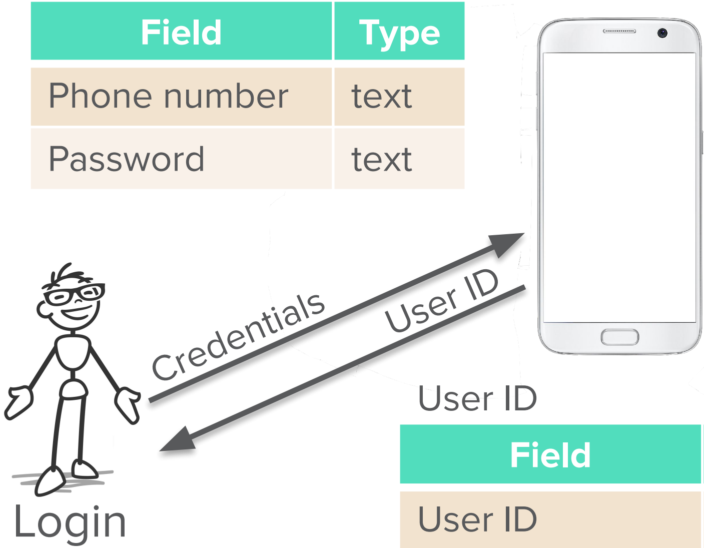
Developer Day Cluster

Part 1

# Cassandra-Land Login Use Case

Credentials

| Field | Type |
|---|---|
| Phone number | text |
| Password | text |



User ID

| Field | Type |
|---|---|
| User ID | UUID |

**users_by_phone_number**

| Field | Type |
|---|---|
| phone_number **K** | text |
| user_id | UUID |
| password | text |

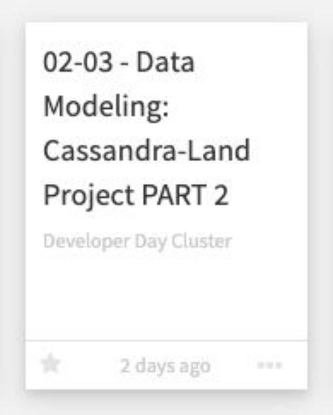
# Cassandra-Land Login Use-Case

- Writing a SELECT statement
  - Must include full partition key
  - Partition keys do NOT support inequalities
  - Not all clustering columns need be specified, but…
  - Any preceding clustering columns MUST be specified

```
SELECT * FROM <keyspace name>.<table name>
    WHERE <query constraints>;
```

# Notebook Data Modeling

# Cassandra-Land Project



02-03 - Data Modeling: Cassandra-Land Project PART 2

Developer Day Cluster

2 days ago

Part 2

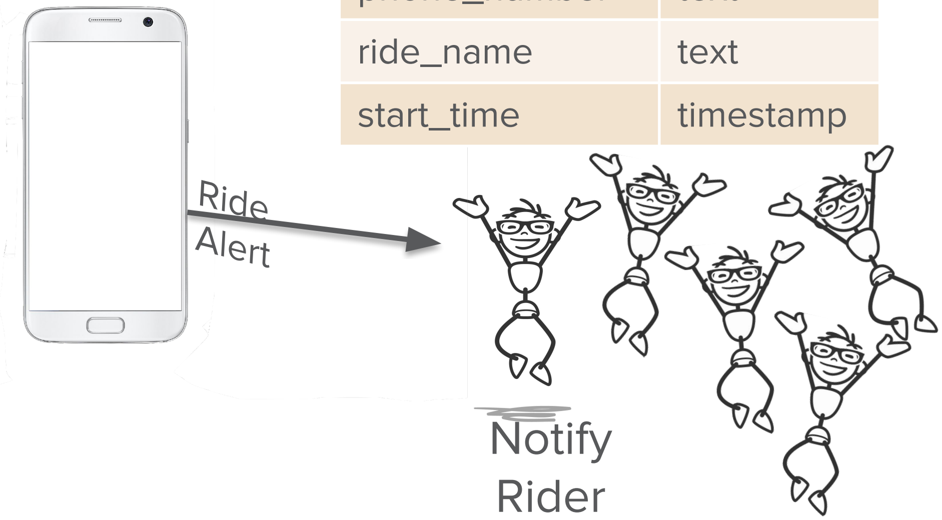
# Cassandra-Land Ride Alert Use-Case

**ride_instances_by_start_time**

| Field | Type |
|---|---|
| start_time **K** | timestamp |
| ride_id **C↑** | UUID |
| user_id **C↑** | UUID |
| ride_name | text |
| phone_number | text |

Ride Alert

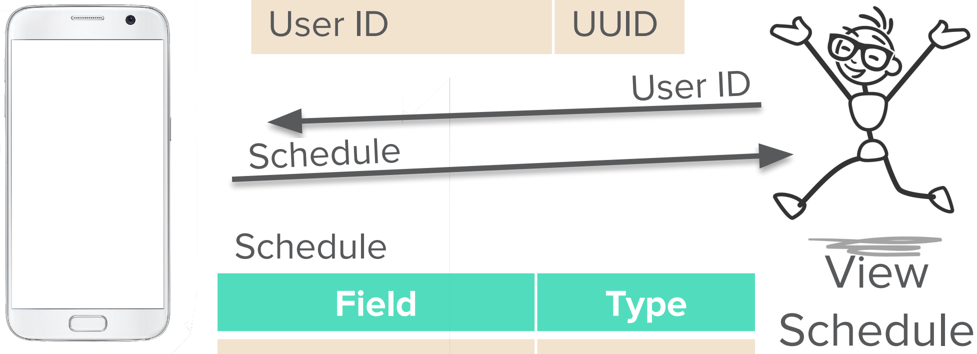
| Field | Type |
|---|---|
| phone_number | text |
| ride_name | text |
| start_time | timestamp |



# Cassandra-Land View Schedule Use-Case

**ride_instances_by_user_id**

| Field | Type |
|---|---|
| user_id **K** | UUID |
| start_time **C↑** | timestamp |
| ride_id | UUID |
| ride_name | text |



User ID

| Field | Type |
|---|---|
| User ID | UUID |

Schedule

| Field | Type |
|---|---|
| start_time | timestamp |
| ride_name | text |

# Cassandra-Land Schedule Ride Use-Case

**ride_list_by_location**

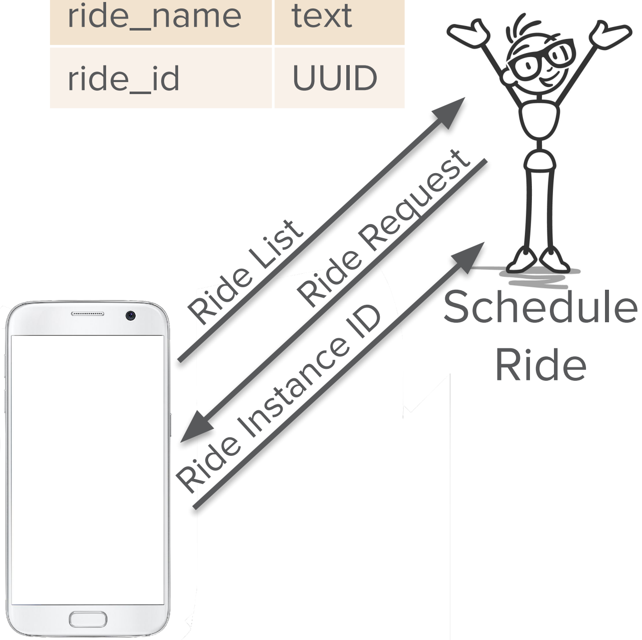
| Field | Type |
|---|---|
| location **K** | text |
| ride_id **C↑** | UUID |
| ride_name | text |
| capacity | int |

**rider_count_by_time_and_ride**

| Field | Type |
|---|---|
| start_time **K** | timestamp |
| ride_id **C↑** | UUID |
| rider_count | int |

Ride List

| Field | Type |
|---|---|
| ride_name | text |
| ride_id | UUID |



Ride Request

| Field | Type |
|---|---|
| user_id | UUID |
| ride_id | UUID |
| start_time | timestamp |

Ride Instance ID

| Field | Type |
|---|---|
| ride_instance_id | UUID |

# Cassandra-Land Table Summary

### users_by_phone_number

| Field | Type |
|---|---|
| phone_number **K** | text |
| user_id | UUID |
| password | text |

### rider_count_by_time_and_ride

| Field | Type |
|---|---|
| start_time **K** | timestamp |
| ride_id **C↑** | UUID |
| rider_count | int |

### ride_instances_by_start_time

| Field | Type |
|---|---|
| start_time **K** | timestamp |
| ride_id **C**↑ | UUID |
| user_id **C**↑ | UUID |
| ride_name | text |
| phone_number | text |

### ride_list_by_location

| Field | Type |
|---|---|
| location **K** | text |
| ride_id **C↑** | UUID |
| ride_name | text |
| capacity | int |

### ride_instances_by_user_id

| Field | Type |
|---|---|
| user_id **K** | UUID |
| start_time **C↑** | timestamp |
| ride_id | UUID |
| ride_name | text |

# Primary Key - What you need to know

- Must have one or more partition key columns
- May have zero or more clustering columns

```
PRIMARY KEY(( <partition key column>,…), <clustering column>,...)
```

# Timestamps

- Format:

```
‘YYYY-MM-DDTHH:MM:SS[.fff]’
```

- Notice the quotes
- Milliseconds are optional
- Examples:

```
‘2020-01-09T11:45:23’
‘2020-01-09T11:45:23.898’
```

# Update Statement

- Can have multiple <assignment>
- IF is optional – causes a lightweight transaction

```
UPDATE <keyspace name>.<table name>
    SET <assignment>
    WHERE <row specification>
IF <condition>
```

# Batch Statement

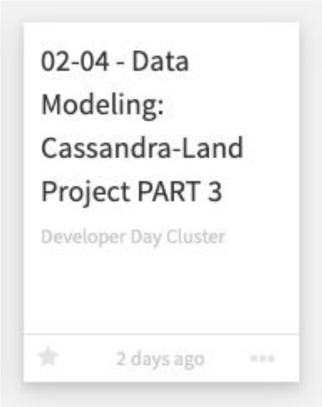
- What you need to know – BATCH

```
BEGIN BATCH
    INSERT statement
    INSERT statement
    …
APPLY BATCH
```

- Once a statement succeeds, Cassandra will ensure all the others succeed
- Can use for inserting into multiple tables

# Notebook Data Modeling

# Cassandra-Land Project



02-04 - Data Modeling: Cassandra-Land Project PART 3

Developer Day Cluster

Part 3

# Cassandra-Land

- How to analyze use-cases to derive a data model
- How to denormalize to maintain performance
- How to use lightweight transactions
- How to leverage batch operations

DATASTAX®

# Thank You