DATASTAX ACADEMY

DS220

05: Optimization and Tuning

DATASTAX ACADEMY

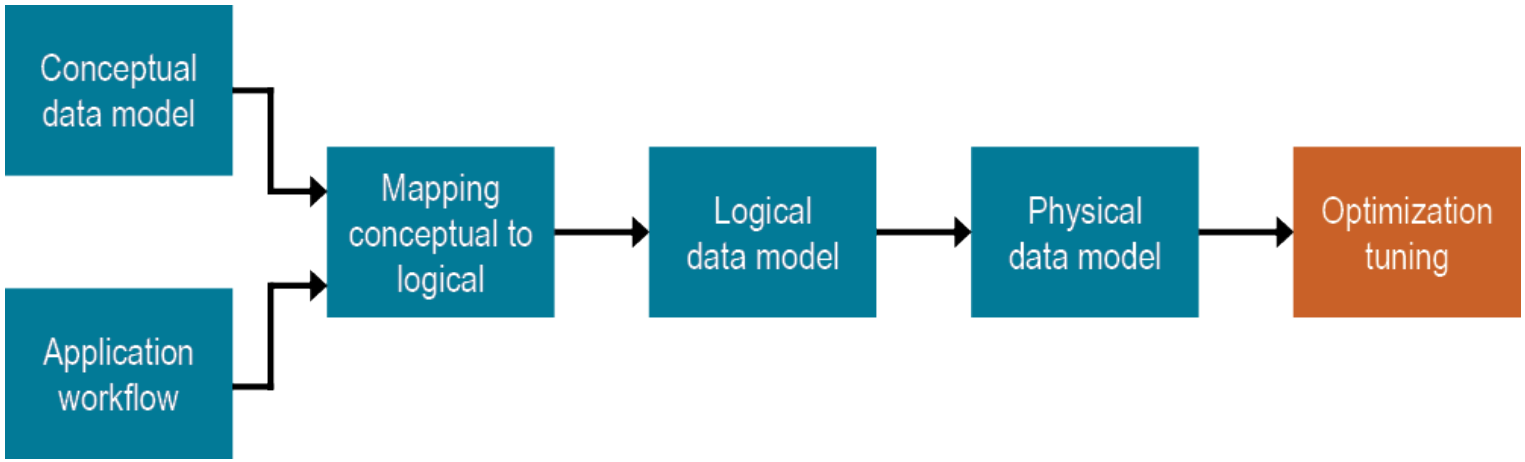
Objectives

Over the next few sections, we will cover:

- Analysis and validation of the data model—is it working?
- Write techniques when using the data model
- Read techniques when using the data model
- Ways to optimize your table keys or tables

Data Modeling Methodology Flow

Where are we now?



DATASTAX ACADEMY

Reasons To Change Data Model

What are the reasons that cause people to revisit their model?

- Requirements change in the domain
- The data model is no longer efficient
- Data is becoming imbalanced
- Unforeseen load on particular nodes leading to hotspotting

DATASTAX ACADEMY

Analysis and Validation

Changing data model or handling new requirements

- Important considerations
  - Natural or surrogate keys?
  - Are write conflicts (overwrites) possible?
  - What data types to use?
  - How large are partitions?
  - How much data duplication is required?
  - Are client-side joins required and at what cost?
  - Are data consistency anomalies possible?
  - How to enable transactions and data aggregation?

DATASTAX ACADEMY

Write Techniques

DATASTAX ACADEMY

Data Consistency with Batches

Schema data consistency refers to the correctness of data copies

- With data duplication, *you* need to worry about and handle consistency
- All copies of the same data in your schema should have the same values
- Adding, updating, or deleting data may require multiple INSERTs, UPDATEs and DELETEs
- Logged batches were built for maintaining consistency
- Have a documented plan!

Batch Example

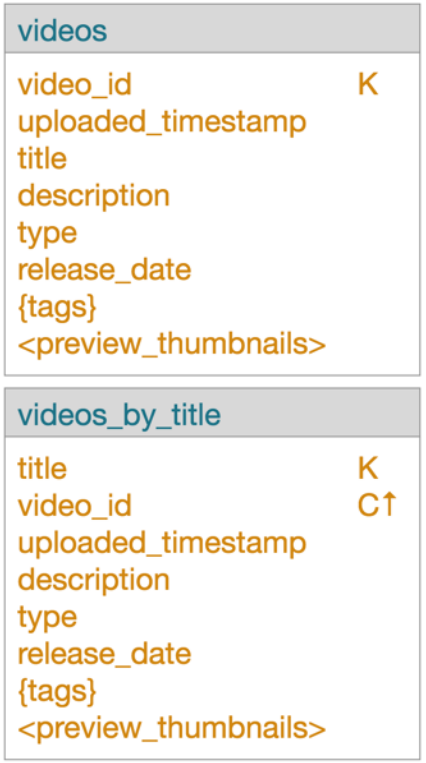
Example of using batch to insert or update a table

- To insert a new video:

```
INSERT INTO videos (video_id, ...) VALUES (1, ...);
INSERT INTO videos_by_title (title, video_id, ...)
     VALUES ('Jaw', 1, ...);
```

- To update the title of a video:

```
UPDATE videos SET title = 'Jaws' WHERE video_id = 1;
INSERT INTO videos_by_title (title, video_id, ...)
VALUES ('Jaws', 1);
DELETE FROM videos_by_title WHERE title = 'Jaw'
AND video_id = 1;
```



DATASTAX ACADEMY

Batch Example, Cont.

Example of using logged batch for data consistency

- To insert a new video:

```
BEGIN BATCH
   INSERT INTO videos (video_id, ...) VALUES (1, ...);
   INSERT INTO videos_by_title (title, video_id, ...) VALUES ('Jaw', 1, ...
APPLY BATCH;
```

- To update the title of a video:

```
BEGIN BATCH
    UPDATE videos SET title = 'Jaws' WHERE video_id = 1;
    INSERT INTO videos_by_title (title, video_id, ...) VALUES ('Jaws',
1);
    DELETE FROM videos_by_title WHERE title = 'Jaw' AND video_id = 1;
APPLY BATCH;
```

DATASTAX ACADEMY

More on Batches

Some batch considerations

- Written to a log on the coordinator node and replicas before execution
  - Batch succeeds when the writes have been applied or hinted
  - Replicas take over if the coordinator node fails mid-batch
- Gets us part of the way to ACID transactions
  - No need for a rollback implementation since Cassandra will ensure that the batch succeeds
  - But no batch isolation—clients may read updated rows before the batch completes

DATASTAX ACADEMY

Misconceptions about Batches

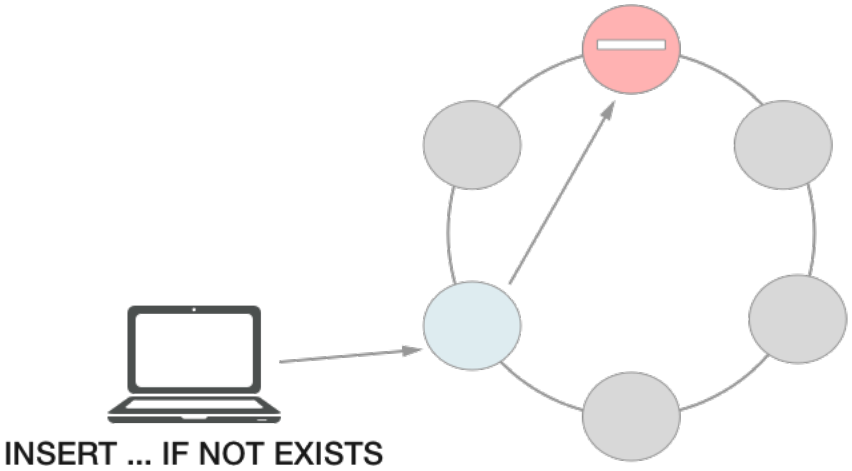
The fine print

- Not intended for bulk loading
  - Rarely increases performance of data load
  - Can overwork the coordinator and cause performance bottlenecks or other issues
- No ordering for operations in batches—all writes are executed with the same timestamp

Lightweight Transactions

Compare and Set (CAS) operation with ACID properties

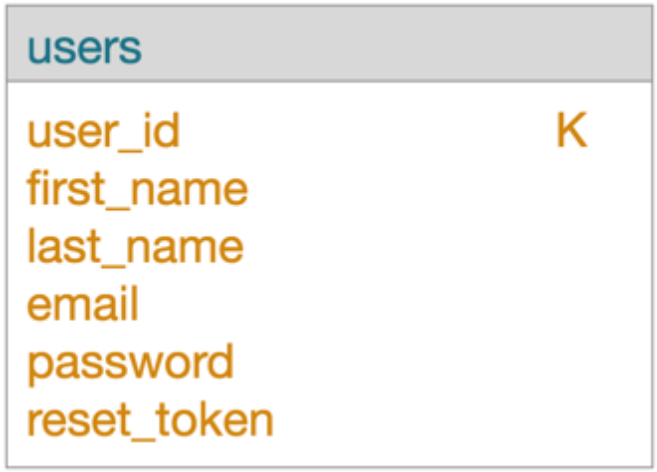
- Does a read to check a condition, and performs the INSERT / UPDATE / DELETE if the condition is true
- Essentially an ACID transaction at the partition level
- More expensive than regular reads and writes



1. Check for partition
2. Partition doesn't exist
3. Write new partition

Using Lightweight Transactions

Create a new user




```
INSERT INTO users (user_id, first_name, last_name, email, password)
VALUES ('pmcfadin', 'Patrick', 'McFadin','patrick@datastax.com', '12345678')
IF NOT EXISTS;
```

Using Lightweight Transactions, Cont.

Generate a reset password token and reset the password

```
UPDATE users
SET reset_token = 12345678-abcd-abcd-abcd-abcd12345678
WHERE user_id = 'pmcfadin';
```

```
UPDATE users
SET reset_token = null, password = 'trustno1'
WHERE user_id = 'pmcfadin'
IF reset_token = 12345678-abcd-abcd-abcd-abcd12345678;
```

DATASTAX ACADEMY

Read Techniques

DATASTAX ACADEMY

Secondary Indexes

What is a secondary index?

- An index on a column that allows a table to be queried on a column that is usually prohibited
  - Table structure is not affected
  - Table partitions are distributed across nodes in a cluster based on a partition key
- Can be created on any column including collections except counter columns

DATASTAX ACADEMY

Secondary Indexes, Cont.

How do they work?

- Secondary index creates additional data structures on nodes holding table partitions
  - Each 'local index' indexes values in rows stored locally
  - Query on an indexed column requires accessing local indexes on all nodes—expensive
  - Query on a partition key and indexed column requires accessing local index on nodes—efficient

Secondary Indexes, Cont.

Secondary indexes in action

- Client sends a query request to a coordinator
- Coordinator sends the request to all nodes—partition key is not known
- Each node searches its local index—returns results to coordinator
- Coordinator combines and returns results

When Secondary Indexes Are Used

Secondary index consideration

When to Use	When Not to Use
Low cardinality columns	High cardinality columns
When prototyping or smaller datasets	With tables that use a counter column
For search on both a partition key and an indexed column in large partition	Frequently updated or deleted columns

Exercise 5.1: Secondary Indexes

DATASTAX ACADEMY

Materialized Views

What is a materialized view?

- A database object that stores query results
- Apache Cassandra™ builds a table from another table's data
  - Has a new primary key and new properties

- Secondary indexes are suited for low cardinality data
- Materialized views are suited for high cardinality data

DATASTAX ACADEMY

Materialized View, Cont.

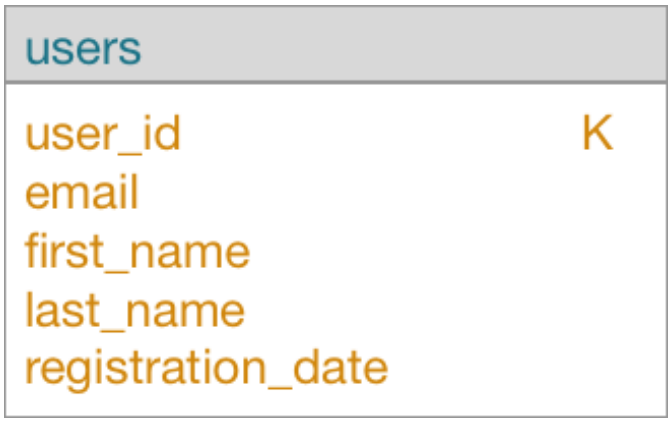
Requirements

- The columns of the source table's primary key must be part of the materialized view's primary key
- Only one new column can be added to the materialized view's primary key
  - Static columns are not allowed

Creating Materialized Views

Given the users table

```
CREATE MATERIALIZED VIEW user_by_email
AS SELECT first_name, last_name, email
FROM users
WHERE email IS NOT NULL AND user_id IS NOT NULL
PRIMARY KEY (email, user_id);
```



Note: normally you would query this table using `user_id`. Creating a materialized view allows us to query by other columns.

DATASTAX ACADEMY

Creating Materialized Views

CREATE materialized view syntax

- AS SELECT: identifies the columns copied from the base table to the materialized view
- FROM: identifies the source table from where the data is copied
- WHERE: must include all primary key columns with IS NOT NULL so that only rows with data for all the primary key columns are copied to the materialized view
- Specification of the primary key columns is crucial
  - The source table USERS uses `id` as its primary key thus `id` must be present in the materialized view's primary key

DATASTAX ACADEMY

Materialized Views Example

Seeing it in action

- Select from the users table using email address
- Note: the primary key of the original users table did not include email

```
SELECT first_name, last_name, email FROM
user_by_email WHERE email =
'iluvbigdata@datastax.com';
```

```
 first_name | last_name | email
------------+-----------+---------------------------
        Joe |   Datafan | iluvbigdata@datastax.com
```

DATASTAX ACADEMY

Materialized Views Caveats

Be aware of the following

- Data can only be written to source tables not materialized views
- Materialized views are asynchronously updated after inserting data into the source table—materialized views update is delayed
- Apache Cassandra™ performs a read repair to a materialized view only after updating the source table

Exercise 5.2: Implementing Materialized Views

DATASTAX ACADEMY

Data Aggregation

Built-in functions provide some summary form of data

Function	Description	Supported
SUM	Total value from a column within selected rows	YES
AVG	Mean value of a column within selected rows	YES
COUNT	Count of selected rows	YES
MIN	Min value of a column within selected rows	YES
MAX	Max value of a column within selected rows	YES

How To Do Data Aggregation

Possible solutions

- Update data aggregates on-the-fly in Cassandra
  - Same technique for linearizable transactions--lightweight transactions
  - Counter type
- Implement data aggregation on client-side
- Use Apache Spark—near real-time batch aggregation
- Use Apache Solr—Stats component

How To Do Data Aggregation

The counter type

ratings_by_video	
video_id	K
num_ratings	++
sum_ratings	++

- Important data type for aggregation
- Increment, decrement, add or subtract the current value
- Counter operations internally requires a read before write
- May not be 100% accurate--it's not an idempotent operation

```
UPDATE ratings_by_video
SET num_ratings = num_ratings + 1,   /* increment num_ratings by 1 */
    sum_ratings = sum_ratings + 5    /* add 5 to the value of sum_ratings
*/
WHERE video_id = 12345678-abcd-abcd-abcd-abcd12345678;
```

DATASTAX ACADEMY

How To Do Aggregation

Implementing data aggregation in an application

- Retrieve data from Apache Cassandra™
- Aggregate data in an application
- Store the result back in Apache Cassandra™

ratings_by_video	
video_id	K
num_ratings	++
sum_ratings	++

videos	
video_id	K
avg_rating	
uploaded_timestamp	
title	
description	
type	
{tags}	
<preview_thumbnails>	
{genres}	

DATASTAX ACADEMY

Exercise 5.3: Implementing Aggregation in Your Data Model

DATASTAX ACADEMY

CAST function

Converts data from one data type to another

- Supported in SELECT statements only
- Converts:
  - From any native data type to text (ASCII and UTF-8)
  - Between numeric types (example: from `int` to `float`, `float` to `int`, etc.)
- Most common use case:
  - Convert timestamp to text for display purposes

CAST function

Example

Without CAST

```
SELECT added_date
FROM videos
WHERE video_id =
6c4cffb9-0dc4-1d59-af24-c960b5fc3652;
```

Result

```
2014-11-06 01:11:50.000000+0000
(time stamp format)
```

With CAST

```
SELECT CAST(added_date AS text)
FROM videos
WHERE video_id =
6c4cffb9-0dc4-1d59-af24-c960b5fc3652;
```

Result

```
2014-11-06T01:11:50.000Z
(Coordinated Universal Time, or UTC)
```

DATASTAX ACADEMY

Exercise 5.4: Using the CAST function

DATASTAX ACADEMY

Table/Key Optimizations

Typical Relational Key

Maintain Uniqueness and minimum number of columns

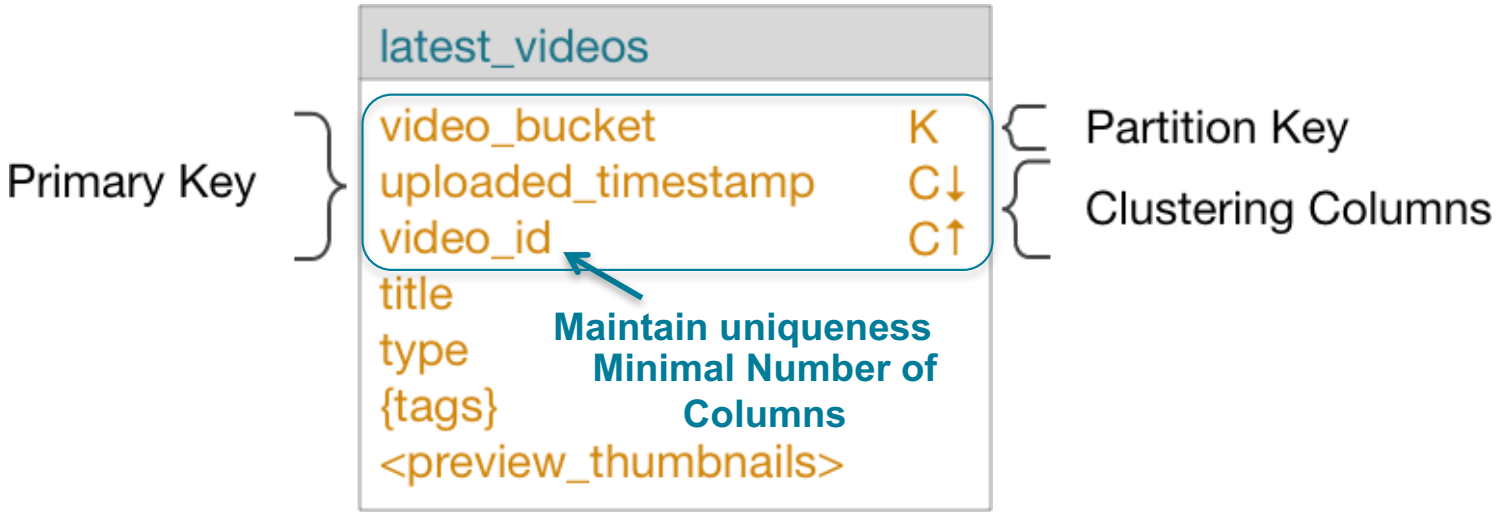
videos	
video_id	K
user_id	K
title	K
description	K
release_date	
uploaded_timestamp	

VS.

videos	
video_id	K
user_id	
title	
description	
release_date	
uploaded_timestamp	

Apache Cassandra™ Key

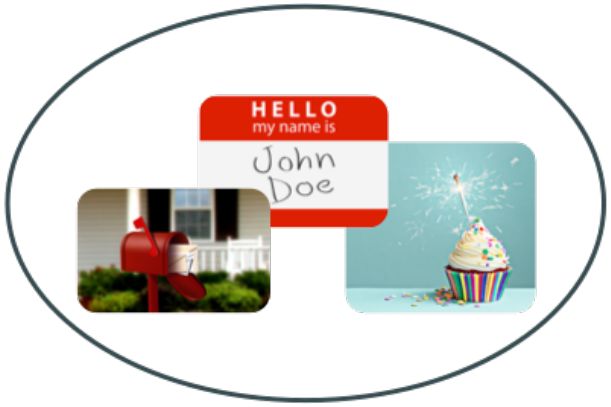
What makes up the key?



Natural Keys

Real world attributes

- They already exist, straightforward to derive, meaningful, and easy to query
- Examples: SSNs, email address, login/passwords, etc.



DATASTAX ACADEMY

Surrogate Keys

What are they?

- Artificial
- Generated
- Meaningless to outside world
- Looks random
- UUID Example:

```
52b11d6d-16e2-4ee2-b2a9-5ef1e9589328
```

Surrogate Keys—Characteristics

- Conflict-free uniqueness
- Immutable
- Uniformity
- Compactness
- Performance

Table Optimizations

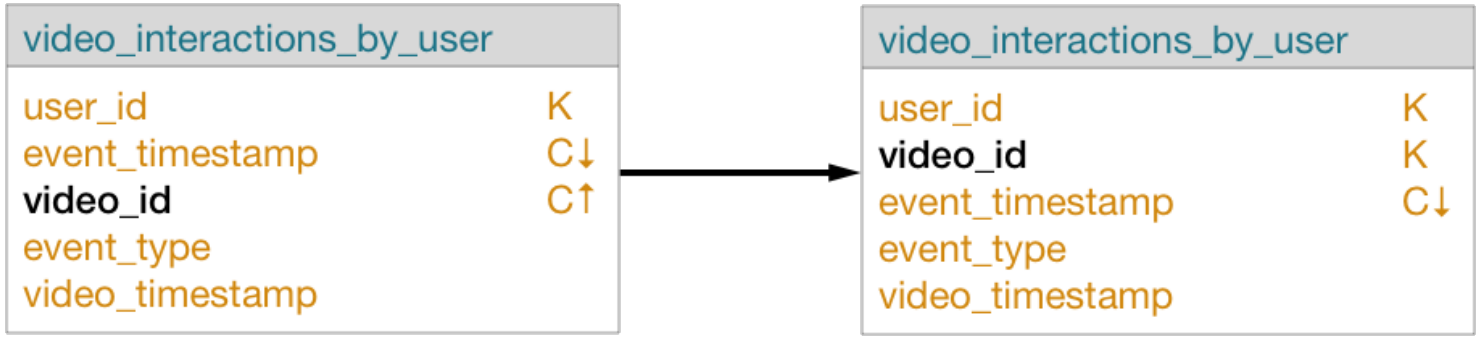
Ways to improve tables

- Splitting partitions—size manageability
- Vertical partitioning—speed
- Merging partitions and tables—speed and eliminate duplication
- Adding columns—speed

Splitting Partitions in a Table

Partitions may grow to large

- Example:
  - Highly active user with 1,000 video interactions (pause, play, seek, etc.) per day will exceed the recommended partition value limit in two months



Splitting Partitions in a Table, Cont.

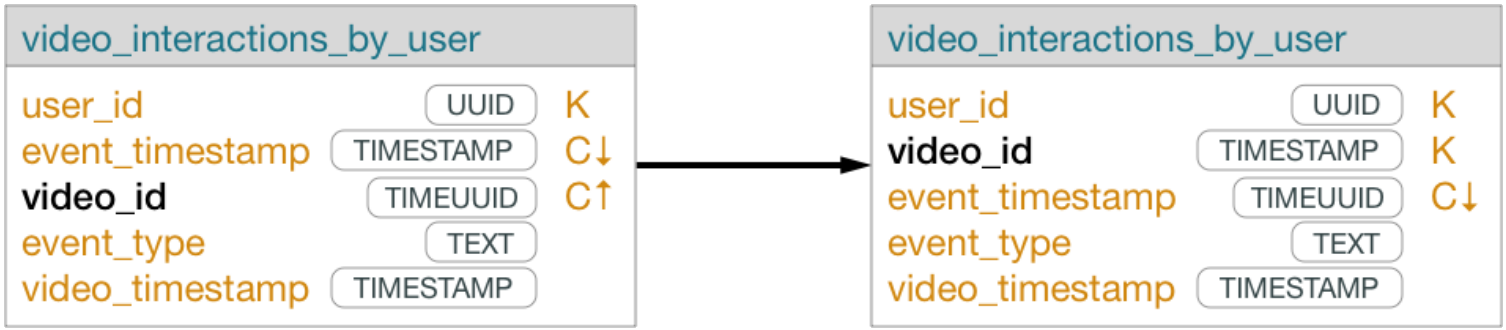
General strategy

- Add another column to a partition key
  - Existing column
  - Artificial column

- Rationale
  - Fewer rows per partition

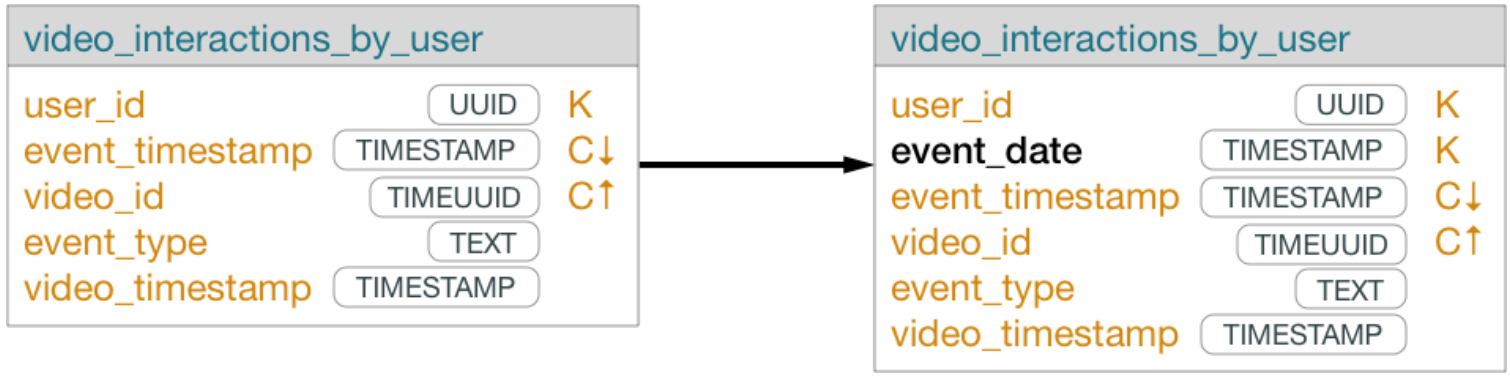
Splitting Using An Existing Column

More convenient



Splitting Using An Artificial Column

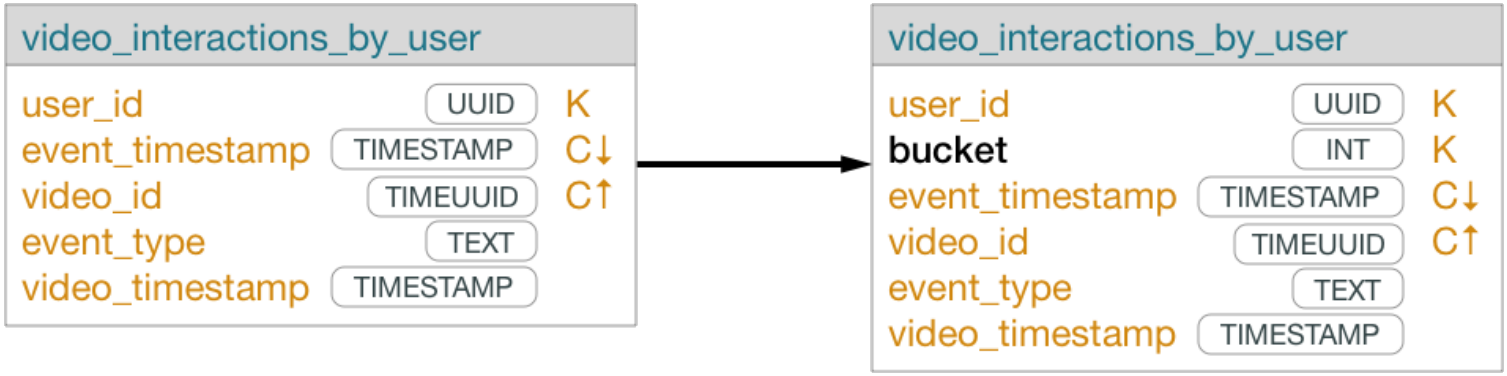
Helps control distribution



DATASTAX ACADEMY

Split Using An Artificial Bucket

Take more control using a “bucket” column



Vertical Partitioning—Splitting Tables

Benefits of vertical partitioning

- Some queries may perform faster
- Table partitions become smaller
- Faster to retrieve and more of them can be cached

DATASTAX ACADEMY

Vertical Partitioning—Splitting Tables

Breaking a table into multiple tables

id	title	...	video_data
1	Insurgent	…	(blob)
2	Interstellar	…	(blob)
3	Mockingjay	…	(blob)

id	title	...
1	Insurgent	…
2	Interstellar	…
3	Mockingjay	…

id	video_data
1	(blob)
2	(blob)
3	(blob)

Merging Tables and Partitions

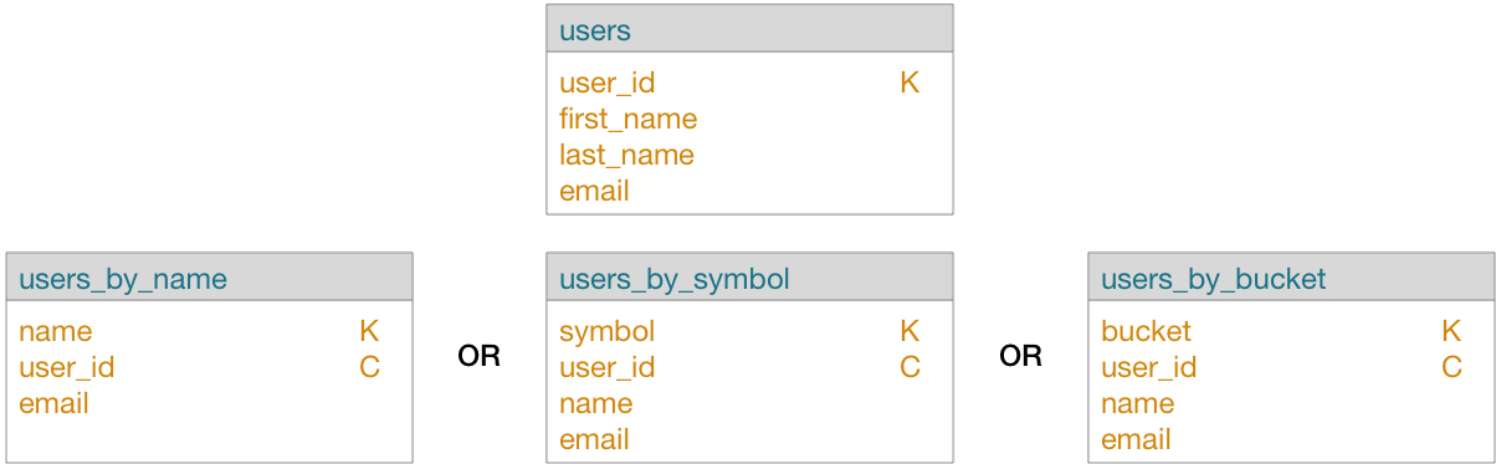
Reverse of vertical partitioning

- Helpful to eliminate duplication
- May result in slower queries

Merging Multiple Partitions

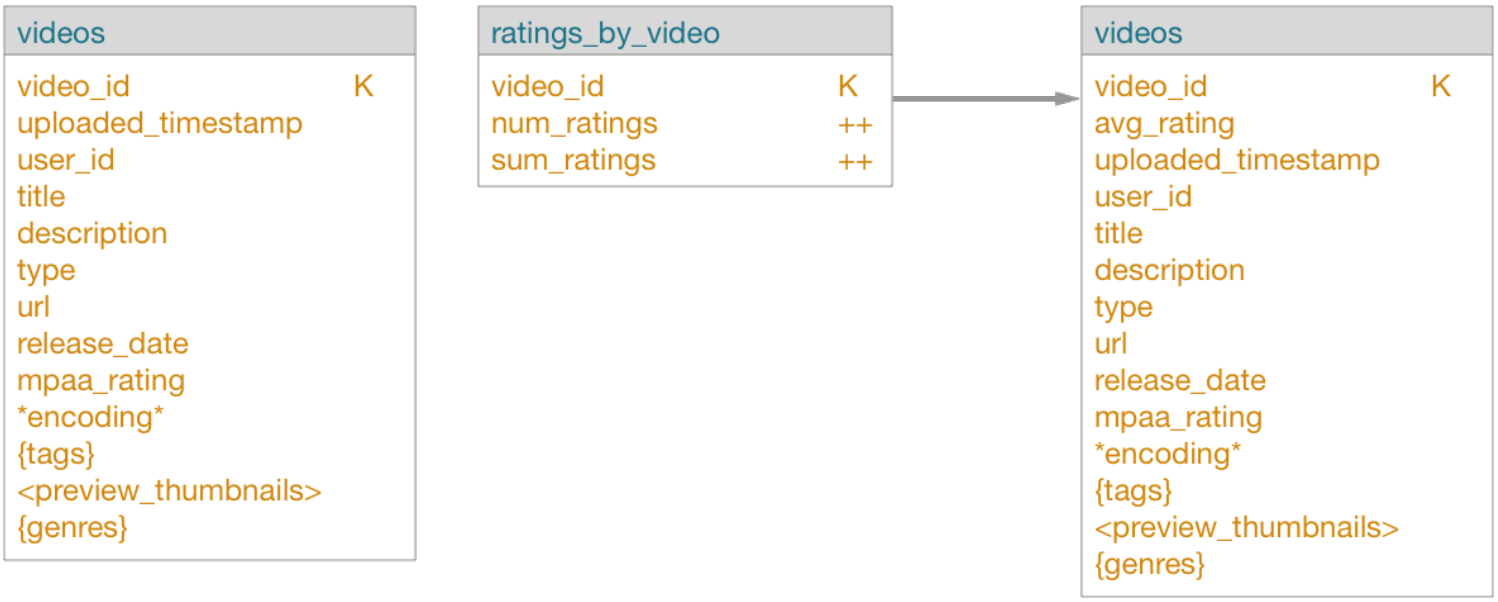
General strategy

- Introduce a new partition key and nest objects into new partitions
- Partition key may consist of existing or artificial columns
- Which of these three is the best choice?



Adding Columns to a Table

Example--Computing an average rating



DATASTAX ACADEMY

Exercise 5.5: Table Optimizations

DATASTAX ACADEMY

Data Model Migration

DATASTAX ACADEMY

Primary Key Changes

Wait! Didn't we say not to change your primary keys?

- In the event that your primary key has to change, you must migrate to a new data model
- But then you have to migrate your data to that new table

- Once complete, you can either:
  - Keep both tables
  - Drop the old table

- Depends on your use case

DATASTAX ACADEMY

Creating New Tables

- Common to keep current schema file in a CQL script
- Schema changes must propagate through the cluster
- On large clusters...
  - Each CREATE TABLE causes a race condition
  - Nodes can become out of sync on schema changes
- Depending on cluster size and load, each CREATE table may take a few seconds to complete
- Be cautious! Schemas not in sync can cause all sorts of havoc in your cluster

Creating New Tables

1. Execute one CREATE/DROP table command at a time
2. Allow time to propagate through the cluster
3. Verify completion via `nodetool describecluster`
4. Then perform the next CREATE/DROP command

DataStax Enterprise Help

- DataStax Enterprise Analytics uses Apache Spark
- Distributed analytics over your entire cluster

- You can use it to migrate your data between tables
- ETL operations are generally straightforward

DATASTAX ACADEMY

Simple ETL Example

- Migrating data from one table to another is straightforward if all you do is change the primary key
- If necessary, you can insert some transformation code between the `cassandraTable()` and `saveToCassandra()` functions
- More details as to how in our DS320 course

```
sc.cassandraTable("keyspace_name","table_name")
.saveToCassandra("keyspace_name", "new_table_name")
```

ETLs With Live Data

- If your application is live, mutations may continue in base table during the ETL after Apache Spark has read it
- *New data could possibly not end up in new tables*
- Plan ahead:
  1. Have your application update both the old and new tables
  2. Perform the ETL
  3. Switch your application to only use the new table
  4. Drop the old table

Other Ways to Migrate Data

- Painful, but...
- CQL
  - Use the CQL COPY command to make a new CSV
  - Transform that CSV using external tool
  - Use COPY to load new CSV file
- DataStax Drivers
  - Use a driver to pull the data, transform, then re-save it
- Both of these approaches are painful, but an option
- (Use Apache Spark instead)

Application Considerations

- Why do you need a different primary key?
  - Partitions too large?
  - Application requirements changed?
  - Do you still need the original table as well for older parts of your application?
- Remember, we build tables to satisfy queries.
- As your application evolves, you may need to add/drop tables
  - Don't drop existing tables until you application no longer depends on them
- Be sure not to create new problems for yourself by making poorly designed partitions

DATASTAX ACADEMY

Data Modeling Anti-Patterns

DATASTAX ACADEMY

Data Modeling Anti-Patterns

What are they?

An anti-pattern is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive.

--Andrew Koenig, Design Patterns

In other words: We've talked about what you should do, let's talk about what you should avoid.

DATASTAX ACADEMY

Query Specific Anti-Patterns

What can go wrong at the query level?

- Queries that do whole cluster or large table scans are expensive—modify your data model to support that query
  - In the event you do want to touch all/most tables in a query might be an awesome use case for DSE Analytics
- Layering IN clauses to get a particular result is non-performant—modify you data model to support that query

DATASTAX ACADEMY

More Query Specific Anti-Patterns

A little bit more to improve the data model

- Queries that require reads before writes excessively are expensive—do not always resort to lightweight transactions, these should be used sparingly
- ALLOW FILTERING is expensive—if you are using more than in an extreme corner case, create a table to support query instead
  - If you KNOW the query is restricted to a single partition, you're okay

DATASTAX ACADEMY

Table Specific Anti-Patterns

What can you improve at the table level?

- Secondary indexes have their uses RARELY—If you need these excessively create tables that support these queries
- Use of non-frozen collections is a huge performance hit—ensure your collection data types are created properly
- Use of String to represent dates and timestamps—use time, timestamp or date (the right tool for the job)

DATASTAX ACADEMY

Keyspace Level Anti-Patterns

How do you improve keyspace level performance?

- Not using TTLs or deletes properly can cause tombstones to build up
- If you are considering increasing read timeouts you should see how changing your data model can improve performance