Materialized Views (MV)

Why Materialized Views ?

- Relieve the pain of manual denormalization

```
CREATE TABLE user(
  id int PRIMARY KEY,
  country text,
  …
);
CREATE TABLE user_by_country(
  country text,
  id int,
  …,
  PRIMARY KEY(country, id)
);
```

Materialzed View In Action

```
CREATE MATERIALIZED VIEW user_by_country
AS SELECT country, id, firstname, lastname
FROM  user
WHERE country IS NOT NULL AND id IS NOT NULL
PRIMARY KEY(country, id)
```

```
CREATE TABLE user_by_country (
    country text,
    id int,
    firstname text,
    lastname text,
    PRIMARY KEY(country, id));
```

Materialzed View Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS]

keyspace_name.view_name

AS SELECT column1, column2, ...

FROM  keyspace_name.table_name

WHERE column1 IS NOT NULL AND column2 IS NOT NULL ...

PRIMARY KEY(column1, column2, ...)
```

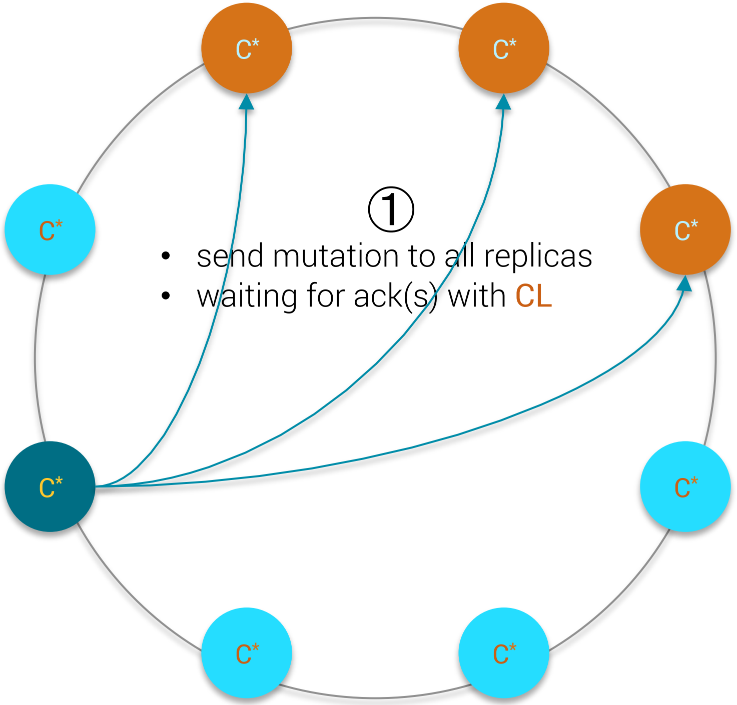
Must select all primary key columns of base table

- IS NOT NULL condition for now
- more complex conditions in future

- at least all primary key columns of base table (ordering can be different)
- maximum 1 column NOT pk from base table

Materialized Views Demo

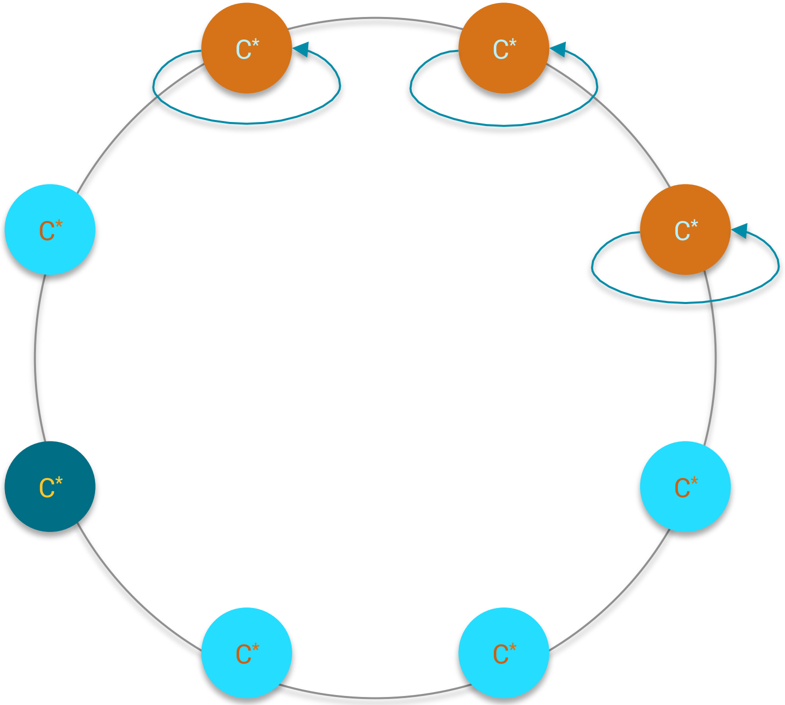
Materialized View Impl



- send mutation to all replicas
- waiting for ack(s) with CL

UPDATE user
SET country='FR'
WHERE id=1

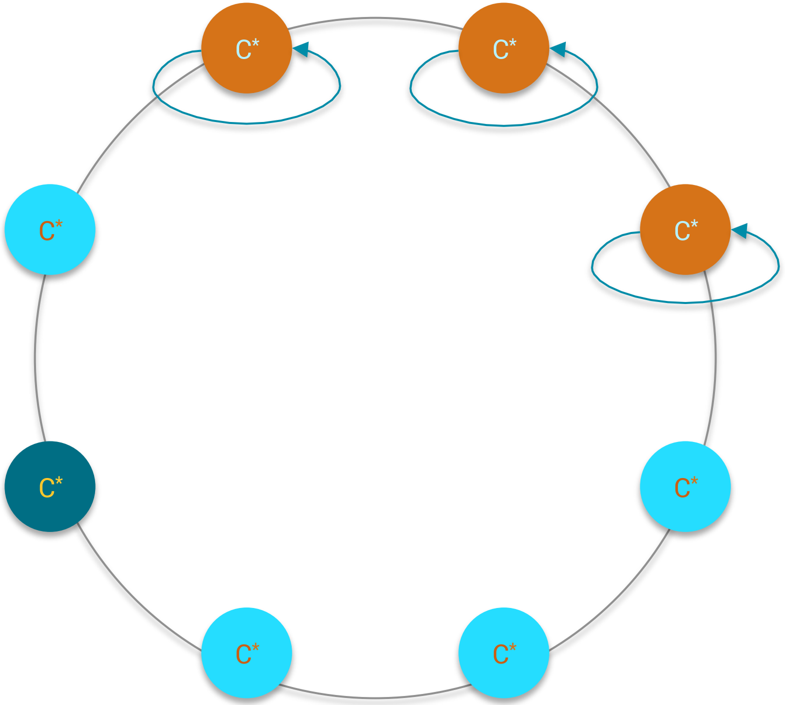
Materialized View Impl



②
Acquire local lock on
base table partition

UPDATE user
SET country='FR'
WHERE id=1

Materialized View Impl

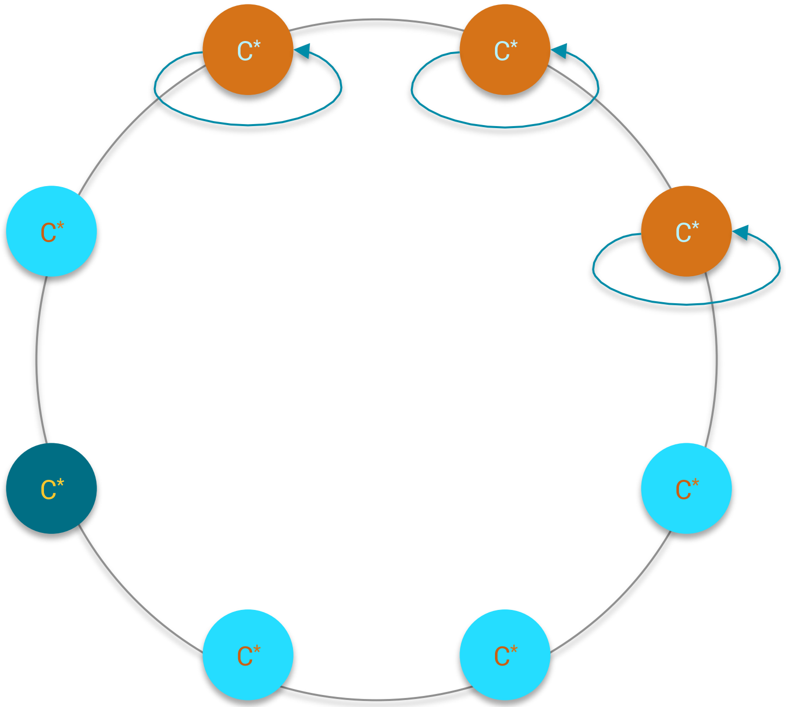


③
Local read to fetch current values
SELECT * FROM user WHERE id=1

UPDATE user
SET country='FR'
WHERE id=1

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

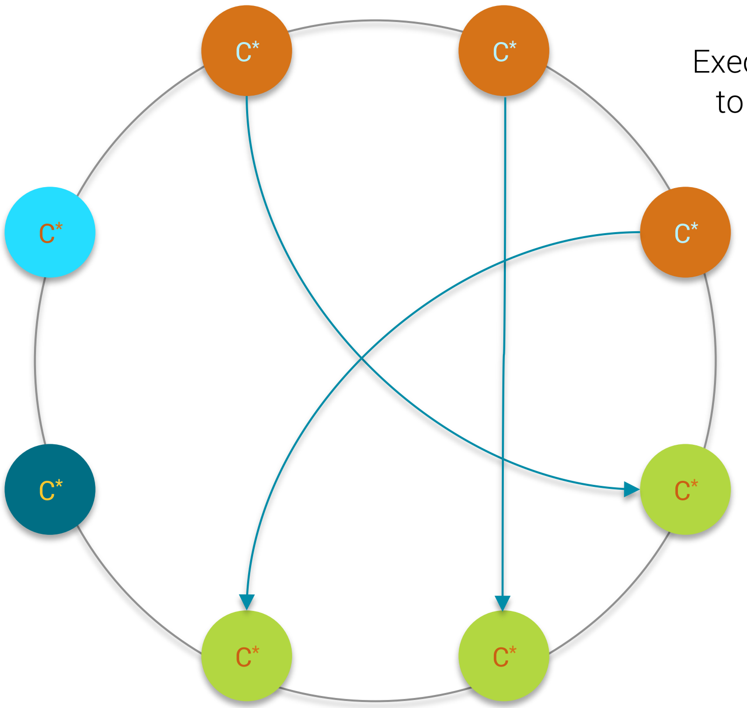


④
Create BatchLog with

- DELETE FROM user_by_country WHERE country = 'old_value'
- INSERT INTO user_by_country(country, id, ...) VALUES('FR', 1, ...)

@doanduyhai

Materialized View Impl

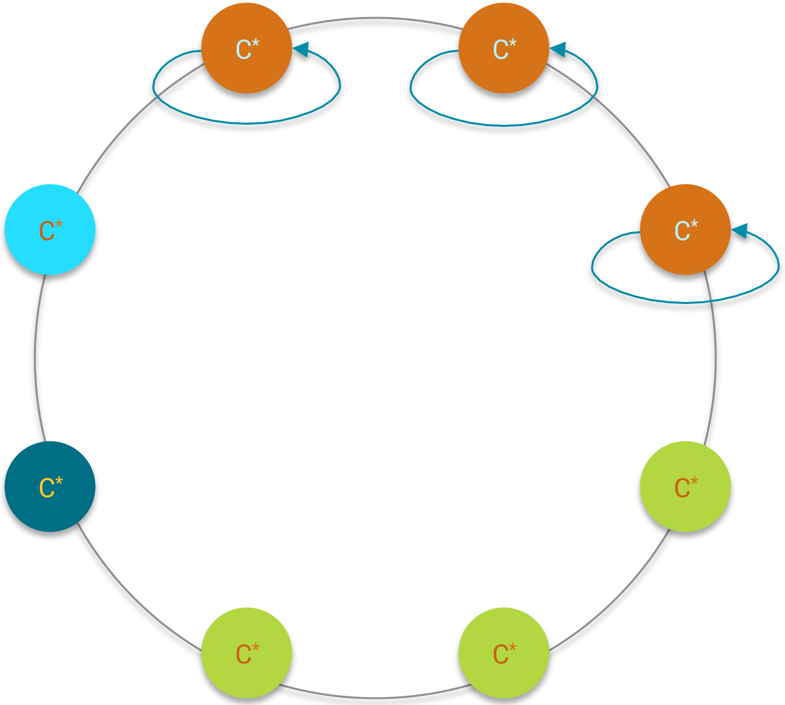


⑤
Execute async BatchLog
to paired view replica
with CL = ONE

UPDATE user
SET country='FR'
WHERE id=1

@doanduyhai

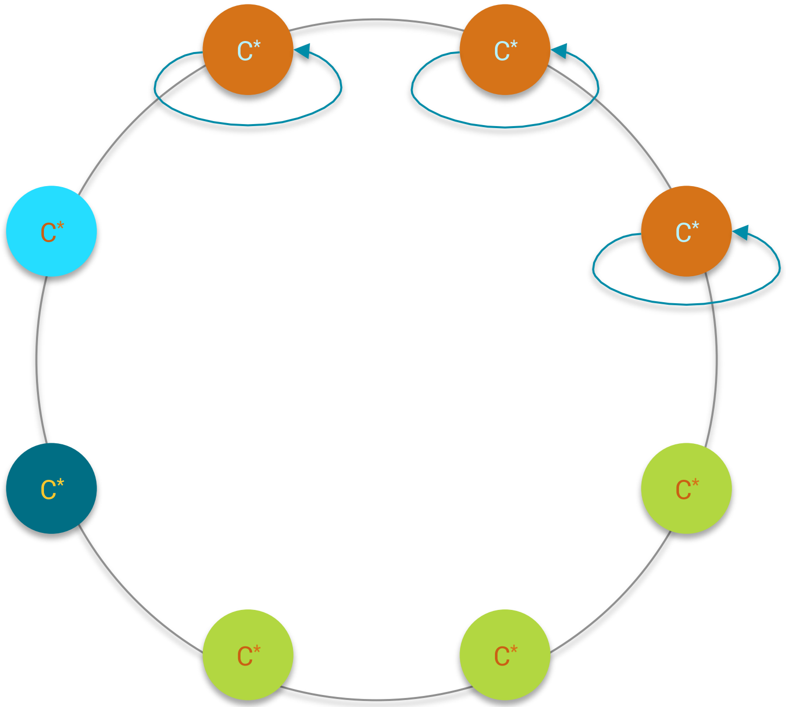
Materialized View Impl



⑥
Apply base table updade locally
SET COUNTRY='FR'

UPDATE user
SET country='FR'
WHERE id=1

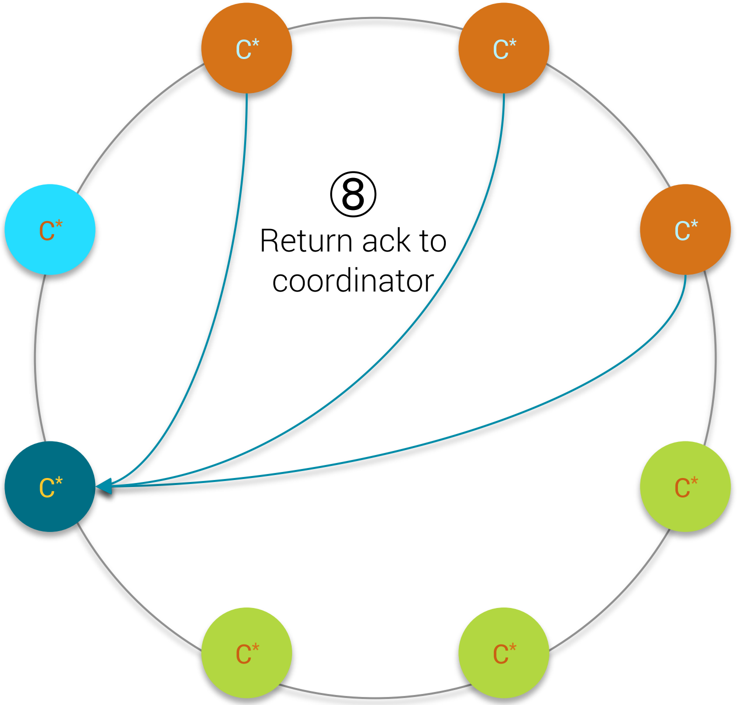
Materialized View Impl



⑦
Release local lock

UPDATE user
SET country='FR'
WHERE id=1

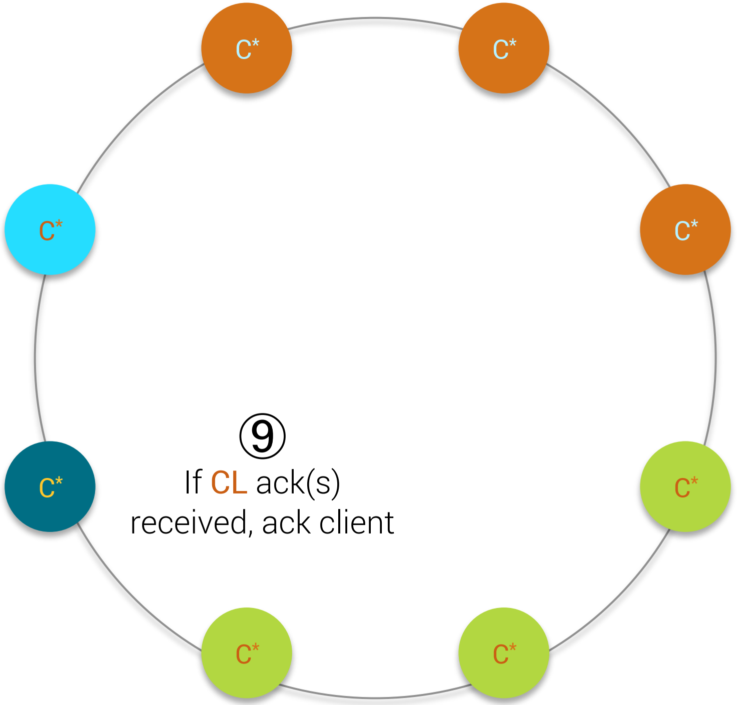
Materialized View Impl



UPDATE user
SET country='FR'
WHERE id=1

Materialized View Impl

UPDATE user
SET country='FR'
WHERE id=1

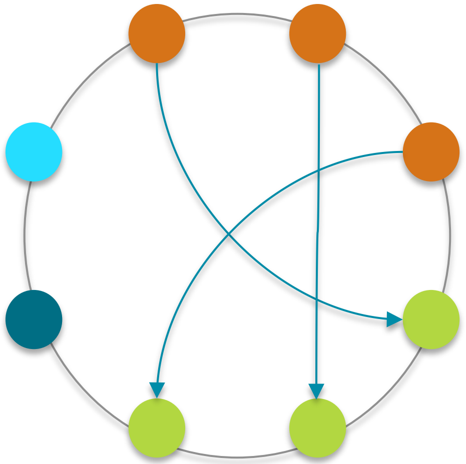


@doanduyhai

Materialzed Views impl explained

What is paired replica ?

- Base primary replica for **id=1** is paired with MV primary replica for **country='FR'**
- Base secondary replica for **id=1** is paired with MV secondary replica for **country='FR'**
- etc …



Materialzed Views impl explained

Why **local lock** on base replica ?

- to provide **serializability** on concurrent updates

Why **BatchLog** on base replica ?

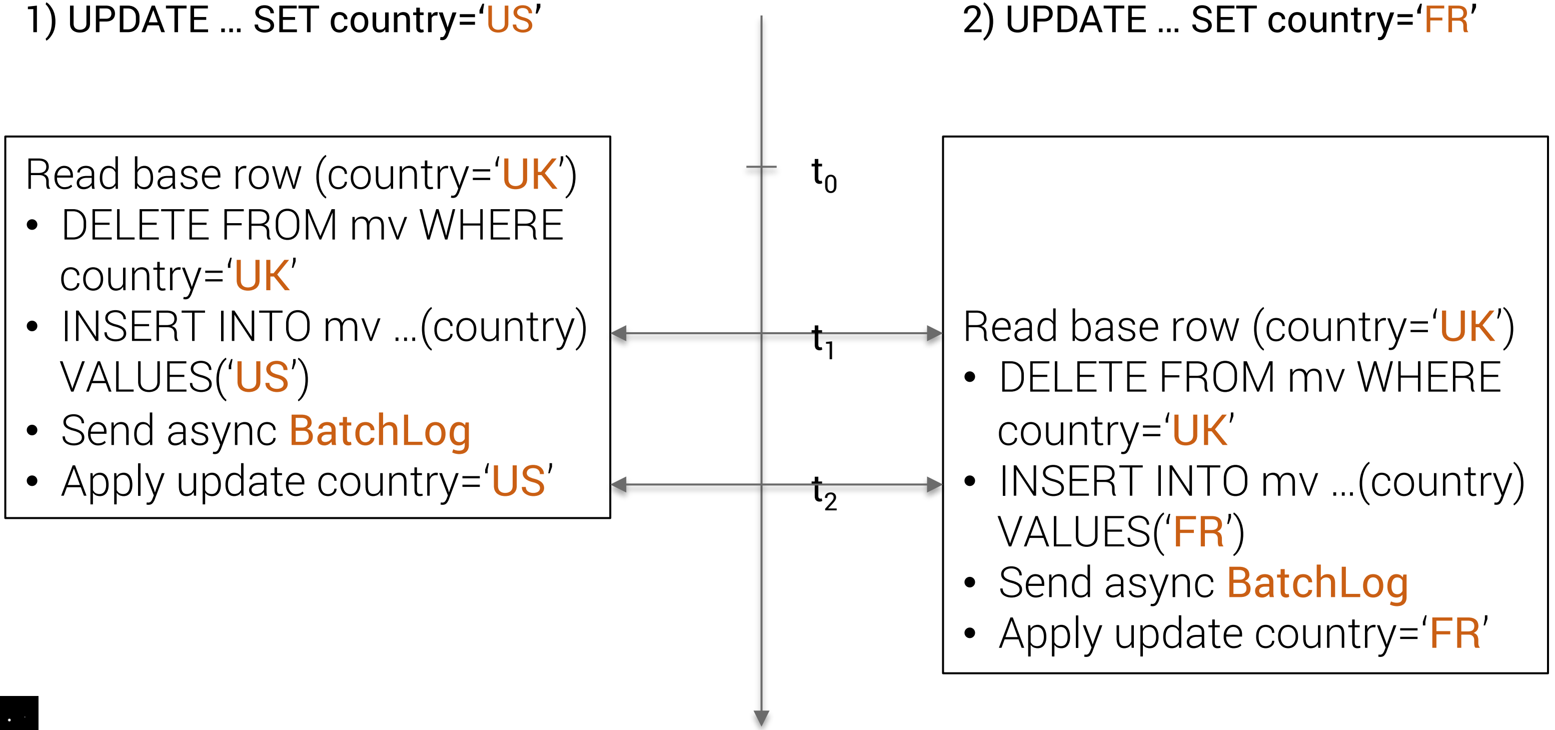
- necessary for **eventual durability**
- replay the MV delete + insert until successful

Why **BatchLog** on base replica uses **CL = ONE** ?

- each base replica is responsible for update of its **paired** MV replica
- CL > ONE will create un-necessary **duplicated mutations**

MV Failure Cases: concurrent updates

without local lock



1) UPDATE … SET country='US'

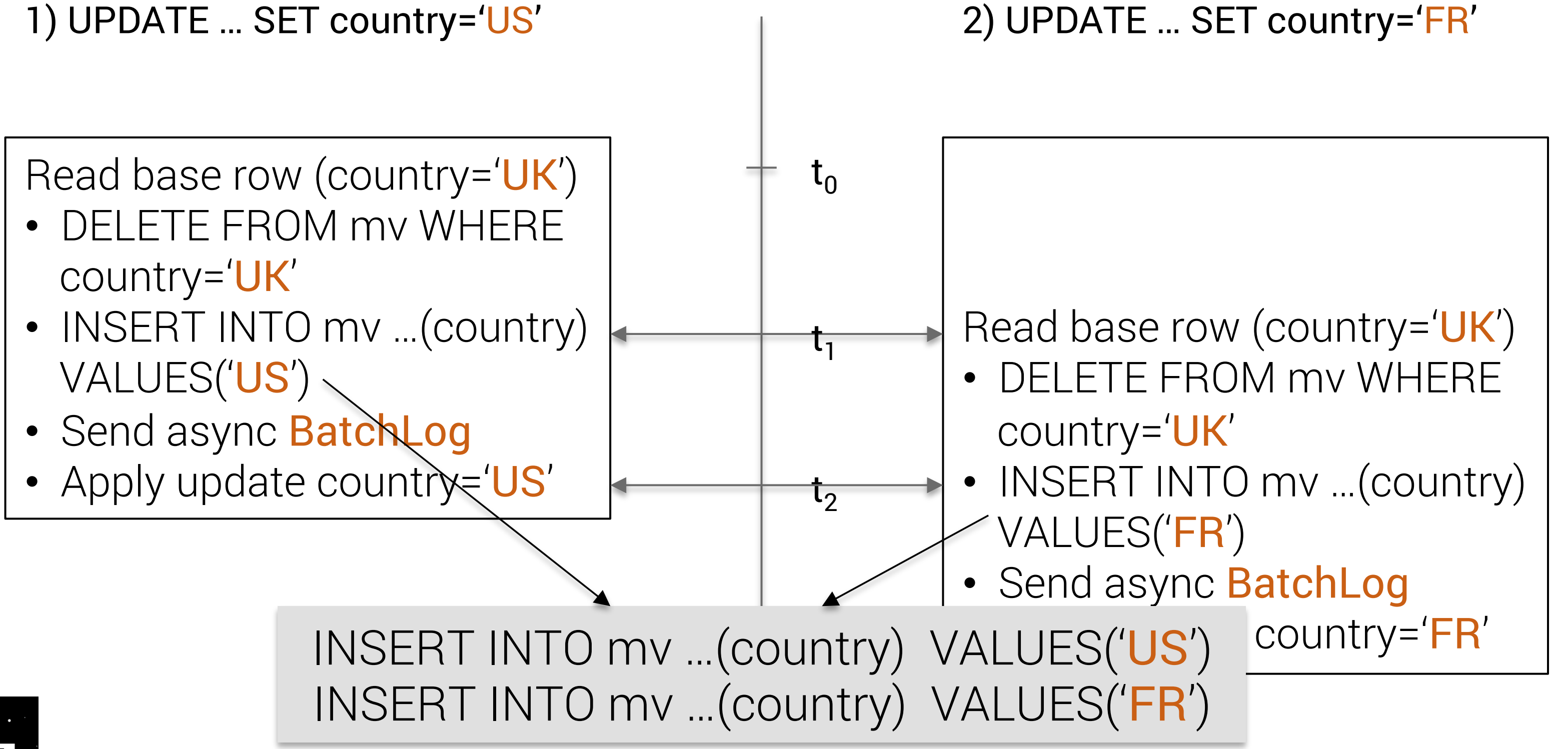
Read base row (country='UK')
- DELETE FROM mv WHERE country='UK'
- INSERT INTO mv …(country) VALUES('US')
- Send async BatchLog
- Apply update country='US'

2) UPDATE … SET country='FR'

Read base row (country='UK')
- DELETE FROM mv WHERE country='UK'
- INSERT INTO mv …(country) VALUES('FR')
- Send async BatchLog
- Apply update country='FR'

MV Failure Cases: concurrent updates

without local lock



1) UPDATE … SET country='US'

Read base row (country='UK')
- DELETE FROM mv WHERE country='UK'
- INSERT INTO mv …(country) VALUES('US')
- Send async BatchLog
- Apply update country='US'

2) UPDATE … SET country='FR'

Read base row (country='UK')
- DELETE FROM mv WHERE country='UK'
- INSERT INTO mv …(country) VALUES('FR')
- Send async BatchLog
- country='FR'

```
INSERT INTO mv …(country)  VALUES('US')
INSERT INTO mv …(country)  VALUES('FR')
```

MV Failure Cases: concurrent updates with Local lock

1) UPDATE … SET country='US'

Read base row (country='UK')

- DELETE FROM mv WHERE country='UK'
- INSERT INTO mv …(country) VALUES('US')
- Send async BatchLog
- Apply update country='US'

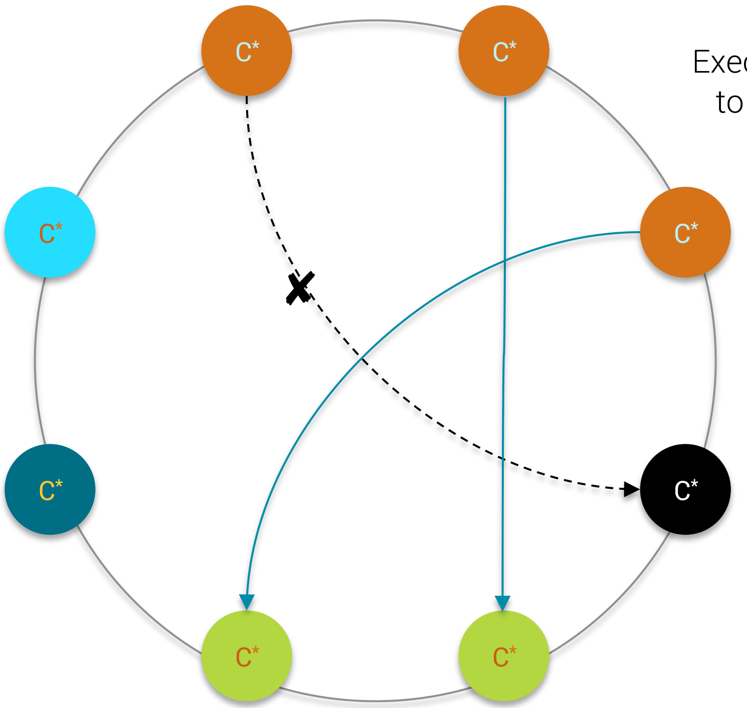
2) UPDATE … SET country='FR'

Read base row (country='US')

- DELETE FROM mv WHERE country='US'
- INSERT INTO mv …(country) VALUES('FR')
- Send async BatchLog
- Apply update country='FR'

MV Failure Cases: failed updates to MV

UPDATE user
SET country='FR'
WHERE id=1

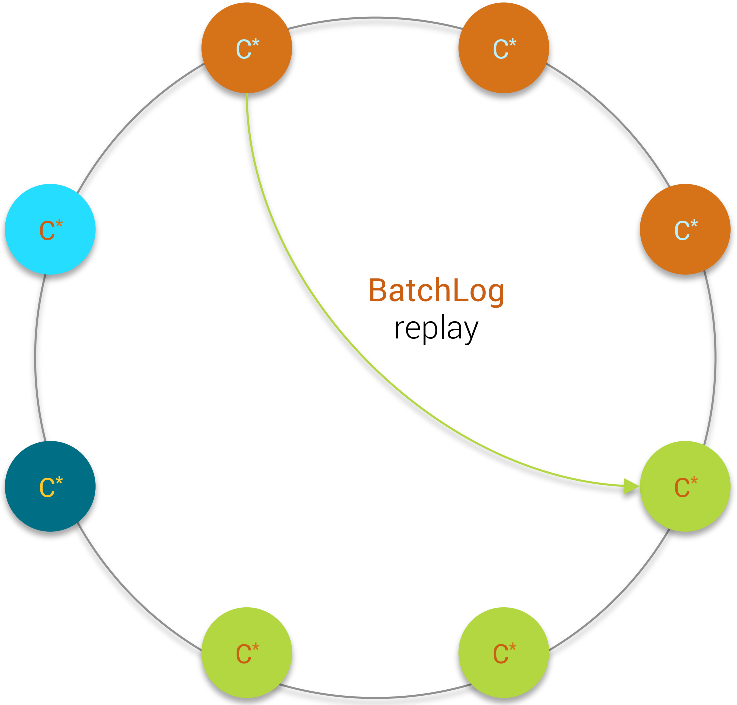


⑤
Execute async BatchLog
to paired view replica
with CL = ONE

MV replica down

@doanduyhai

MV Failure Cases: failed updates to MV



MV replica up

UPDATE user
SET country='FR'
WHERE id=1

Materialized View Performance

- Write performance
  - local lock
  - local read-before-write for MV → update contention on partition (*most of perf hits*)
  - local batchlog for MV
  - ☞ you only pay this price **once** whatever number of MV
  - for each base table update: **mv_count x 2** *(DELETE + INSERT)* extra mutations

Materialized View Performance

- Write performance vs **manual denormalization**
  - MV better because *no client-server network traffic for read-before-write*
  - MV better because *less network traffic* for multiple views (*client-side BATCH*)

- Makes developer life easier → **priceless**

Materialized View Performance

- Read performance vs **secondary index**
  - MV better because **single node read** (secondary index can hit many nodes)
  - MV better because **single read path** (secondary index = *read index + read data*)

Materialized Views Consistency

- Consistency level
    - CL honoured for base table, ONE for MV + local batchlog

- **Weaker consistency guarantees** for MV than for base table.

- Exemple, write at QUORUM
    - guarantee that QUORUM replicas of base tables have received write
    - guarantee that QUORUM of MV replicas will **eventually** receive *DELETE + INSERT*

Materialized Views Gotchas

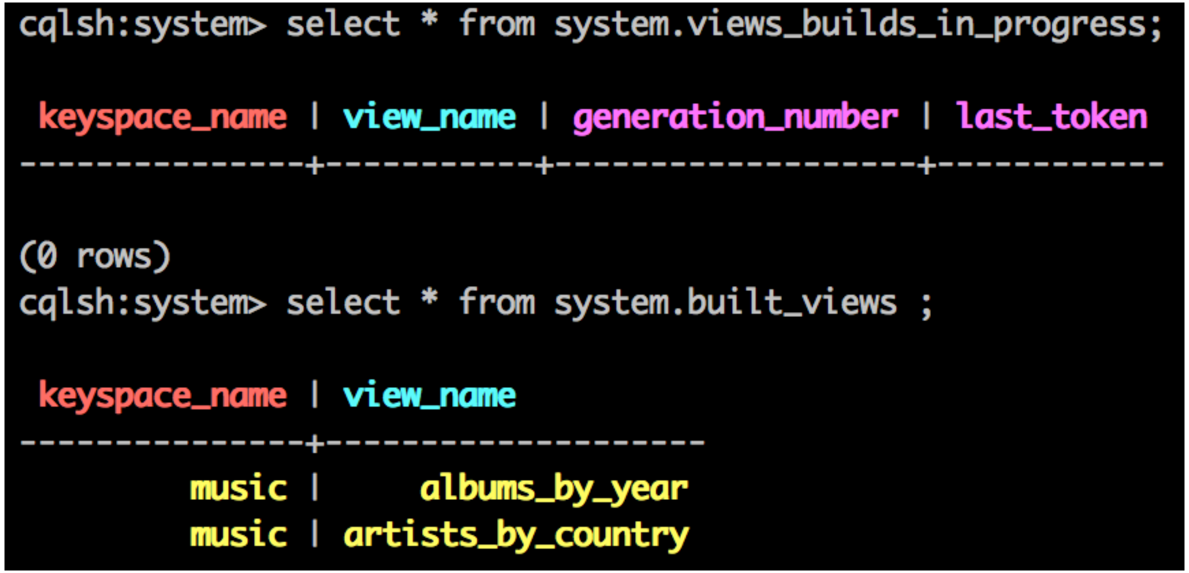
- Beware of hot spots !!! MV **user_by_gender** 😱
- Only 1 non-pk column for MV
- No **static column** for MV view
  - *1:1 relationship* between static column & partition key
  - if MV supports static column → MV has same partition key as base table → uselesss …

Materialized Views Operations

- Repair, read-repair, index rebuild, decomission …
  - repair on base replica (*mutation-based* repair) → update on MV paired replica
  - possible to repair a MV *independently from base table*
  - read-repair on MV behaves as normal read-repair
  - read-repair on base table updates MV
  - hints replay on base table updates MV

Materialized Views Operations

- MV build status ?
  - system.views_builds_in_progress
  - system.built_views
  - data are node-local !!!



```
cqlsh:system> select * from system.views_builds_in_progress;

 keyspace_name | view_name | generation_number | last_token
---------------+-----------+-------------------+-----------


(0 rows)
cqlsh:system> select * from system.built_views ;

 keyspace_name | view_name
---------------+--------------------
         music |      albums_by_year
         music | artists_by_country
```

Materialized Views Schema Ops

- Schema
  - MV can be tuned as normal table (ALTER MATERIALIZED VIEW …)
  - cannot drop column from base table used by MV
  - can add column to base table, initial value = null from MV
  - cannot drop base table, drop all MVs first

Single non-pk column limitation

- Because of Cassandra consistency model
- Because null value forbidden for primary key column

```
CREATE MATERIALIZED VIEW user_by_gender_and_age
AS SELECT country, id, firstname, lastname
FROM  user
WHERE gender IS NOT NULL AND age IS NOT NULL AND id IS NOT NULL
PRIMARY KEY((gender, age) id)
```

APACHE: BIG_DATA EUROPE

@doanduyhai

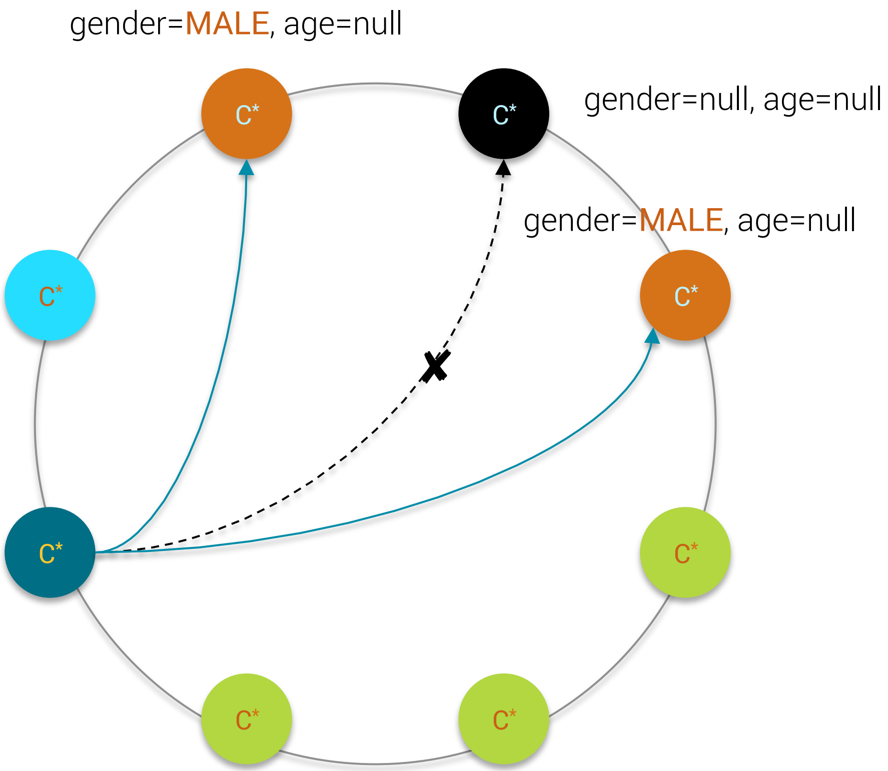
Single non-pk column limitation

- Possible RULE : UPDATE MV ONLY IF ALL COLUMNS IN PK NOT NULL

Multiple non-PK columns in MV PK

CL = QUORUM

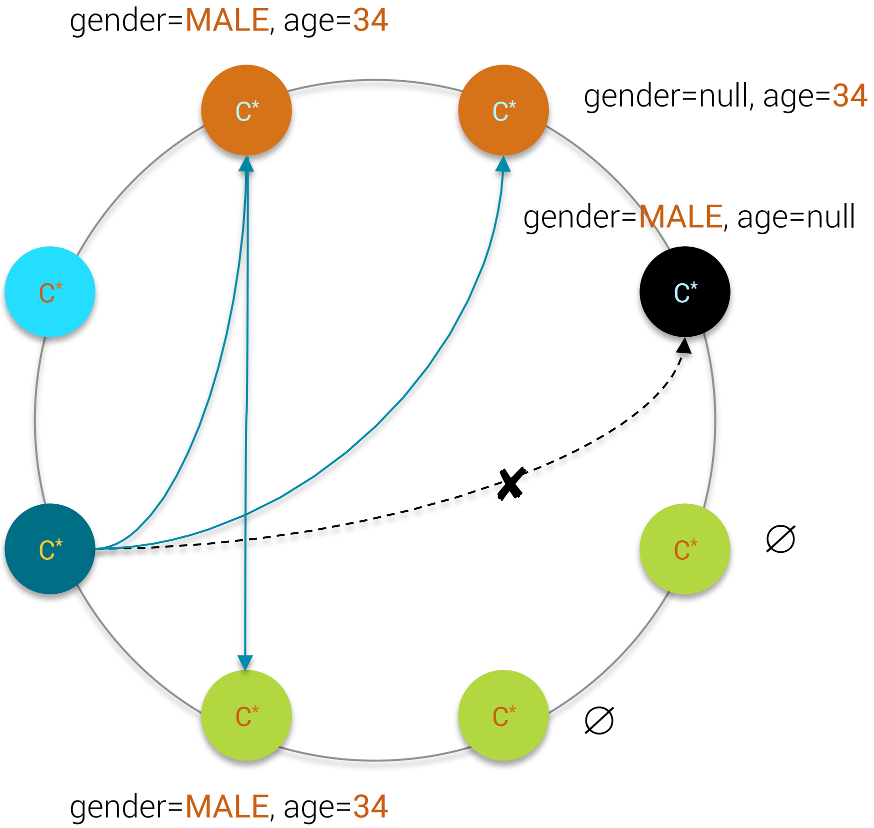
UPDATE user
SET gender='MALE'
WHERE id=1



Multiple non-PK columns in MV PK

CL = QUORUM

UPDATE user
SET age=34
WHERE id=1



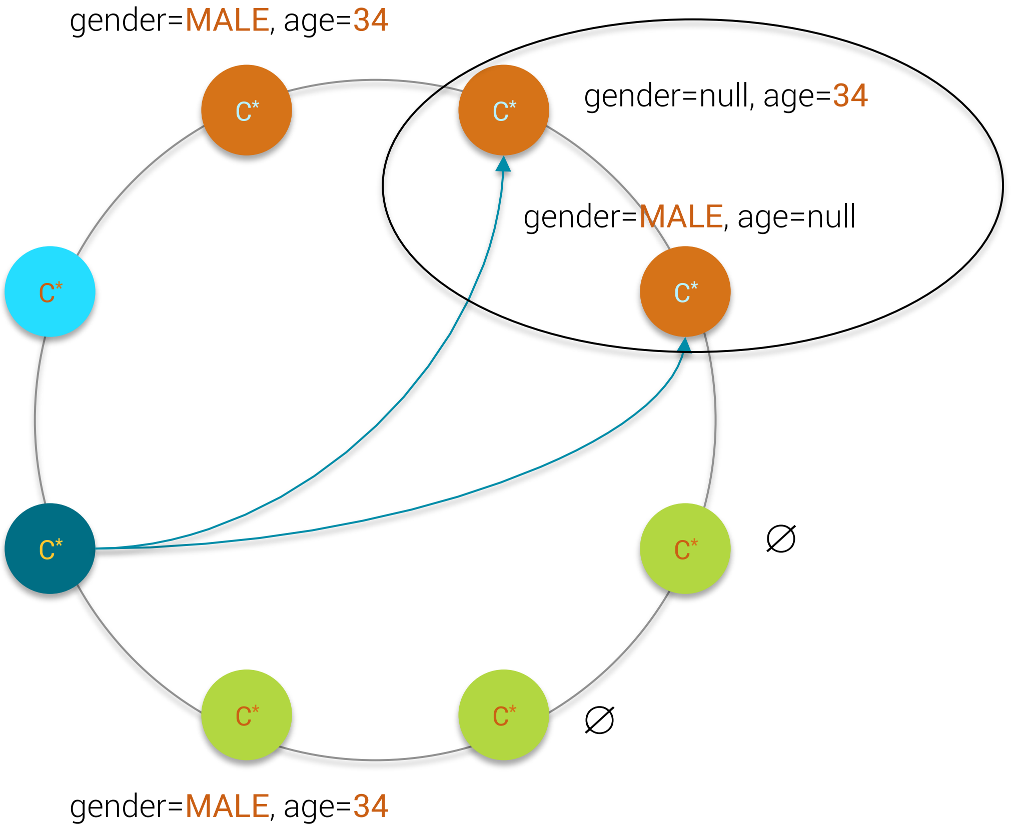
@doanduyhai

Multiple non-PK columns in MV PK

CL = QUORUM

SELECT age,gender
FROM user
WHERE id=1

1	34	MALE



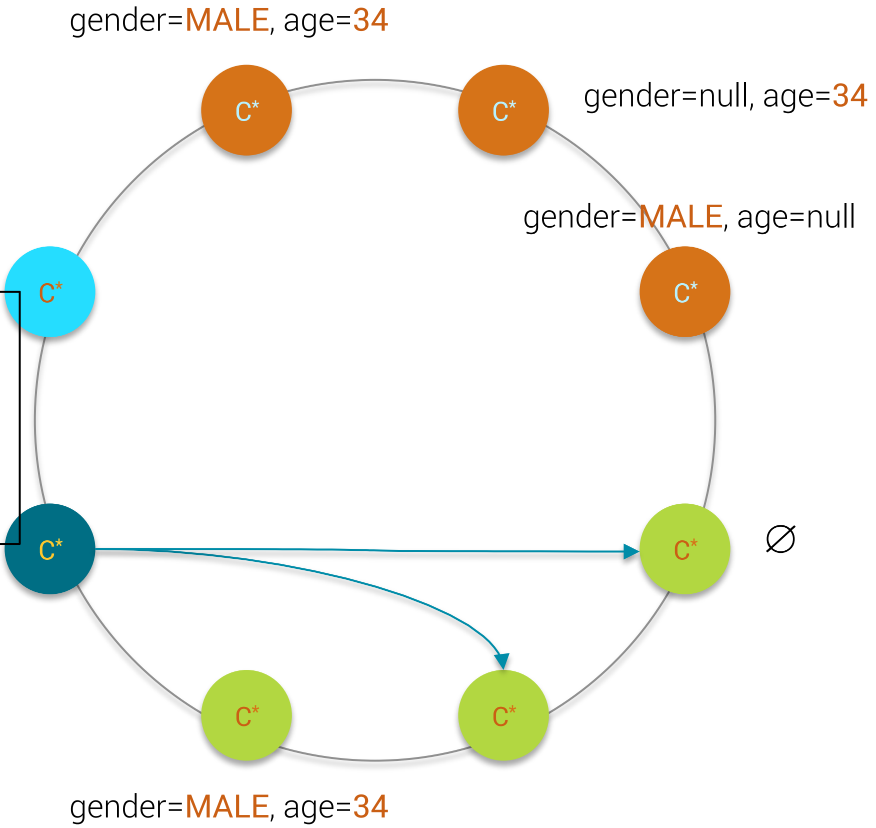
@doanduyhai

Multiple non-PK columns in MV PK

CL = QUORUM

SELECT * FROM
user_by_gender_and_age
WHERE gender='MALE'
AND age=34

∅



Single non-pk column limitation

- Possible RULE 2: **ALLOW NULL VALUE FOR COLUMN IN PK**

```
INSERT INTO user(id, name) VALUES(1, 'John DOE');
…
…
…
INSERT INTO user(id, name) VALUES(1000_000, 'Helen SUE');
```

No age, No gender

Single non-pk column limitation

- Possible RULE 2: **ALLOW NULL VALUE FOR COLUMN IN PK**

(*null*, *null*) single partition with 10^6 users

Living in Danger