C SQL на Cassandra

Александр Пащенко, VK

АЛЕКСАНДР ПАЩЕНКО

Единые
Новые
Облачные
Технологии

Опыт с Cassandra

- Команда монетизации OK
- Платформа OK
- Команда разработки ЕНОТ

Таймлайн внедрения Cassandra в OK:

2007

MS SQL

2010

Cassandra 0.6 в продакшене

2014

Создали Newsql поверх Cassandra2

2018

99% DB - Cassandra

Cassandra 0.6 в продакшене: https://youtu.be/k2efjgRxMp8
Создали Newsql поверх Cassandra2: https://youtu.be/qyTj09e-EM0

Сейчас Cassandra:

- Сотни кластеров

Сейчас Cassandra:

- Сотни кластеров
- Самые большие кластера имеют 1000 нод

Сейчас Cassandra:

- Сотни кластеров
- Самые большие кластера имеют 1000 нод
- 1ТБ данных на 1 ноде

Сейчас Cassandra:

- Сотни кластеров
- Самые большие кластера имеют 1000 нод
- 1ТБ данных на 1 ноде
- Выполнение запроса за 2мс (99 персентиль)

Сейчас Cassandra:

- Сотни кластеров
- Самые большие кластера имеют 1000 нод
- 1ТБ данных на 1 ноде
- Выполнение запроса за 2мс (99 персентиль)
- RPS одного из кластеров > 5 000 000 rps

А расскажите?

- Какие сложности нас ждут
- На что обратить внимание
- Где про это посмотреть

Почему Cassandra:

- Распределенная и надежная

Почему Cassandra:

- Распределенная и надежная
- Высокомасштабируемая

Почему Cassandra:

- Распределенная и надежная
- Высокомасштабируемая
- Производительная

Почему Cassandra:

- Распределенная и надежная
- Высокомасштабируемая
- Производительная
- JAVA

CQL: create table

```
Create table test_table (
   id bigint,
   value text,
   PRIMARY KEY (id)
) …
```

CQL != SQL

```
Create table test_table (
   id bigint,
   value text,
   PRIMARY KEY (id)
) …
```

CQL != SQL

```
CREATE TABLE test.test_table (
id_1 bigint,
id_2 bigint,
id_3 bigint,
value text,
PRIMARY KEY (id_1, id_2, id_3)
) WITH CLUSTERING ORDER BY (id_2 ASC, id_3 ASC)
AND bloom_filter_fp_chance = 0.01
AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
AND compression = {'chunk_length_in_kb': '16', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}
AND crc_check_chance = 1.0
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
```

Primary key:

```
Create table test_table (
  id bigint,
  value text,
  PRIMARY KEY (id)
) …
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  value text,
  PRIMARY KEY (id_1, id_2)
) …
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY (id_1, id_2, id_3)
) …
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY (id_1, id_2, id_3)
) …
```

Primary key = Partition Key + Clustering Key

Primary key:

```
Create table test_table (
   id_1 bigint,
   id_2 bigint,
   id_3 bigint,
   value text,
   PRIMARY KEY (id_1, id_2, id_3)
) …
```

Primary key = Partition Key + Clustering Key

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY (id_1, id_2, id_3)
) …
```

Primary key = Partition Key + Clustering Key

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY (id_1, id_2, id_3)
) …
```

Primary key = Partition Key + Clustering Key

Primary key:

```
Create table test_table (
   id_1 bigint,
   id_2 bigint,
   id_3 bigint,
   value text,
   PRIMARY KEY ((id_1), id_2, id_3)
) …
```

Primary key = Partition Key + Clustering Key

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY ((id_1, id_2), id_3)
) …
```

Primary key = Partition Key + Clustering Key

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY ((id_1, id_2), id_3)
) …
```

Primary key = Partition Key + Clustering Key

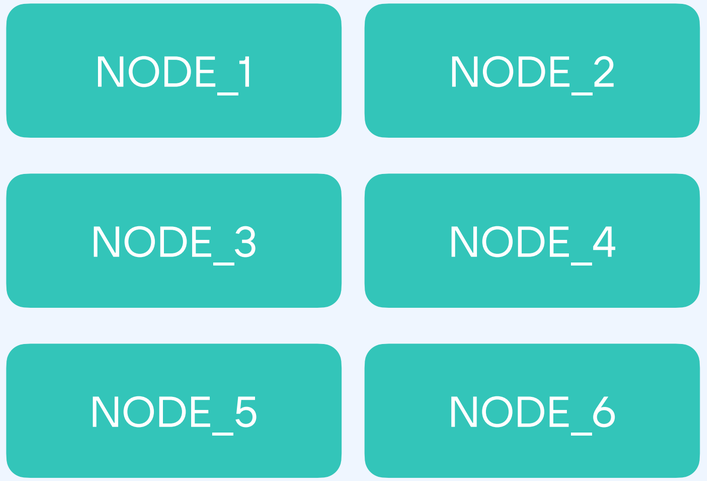
Partition Key - отвечает за выбор ноды

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY ((id_1, id_2), id_3)
) …
```

Primary key = Partition Key + Clustering Key

Partition Key - отвечает за выбор ноды

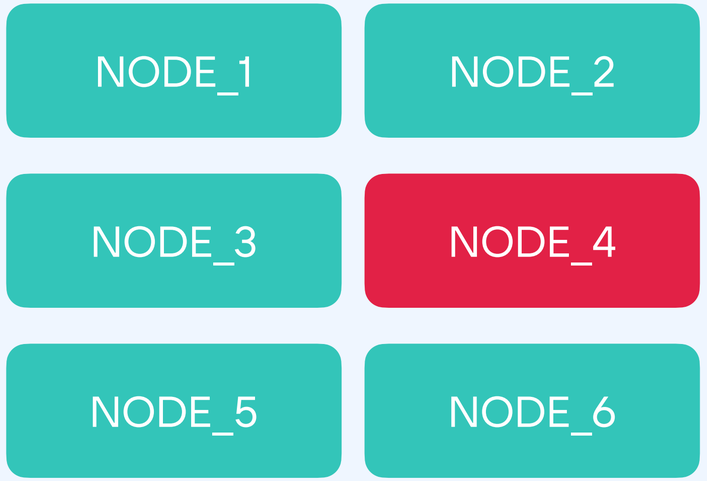


Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY ((id_1, id_2), id_3)
) …
```

Primary key = Partition Key + Clustering Key

Partition Key - отвечает за выбор ноды

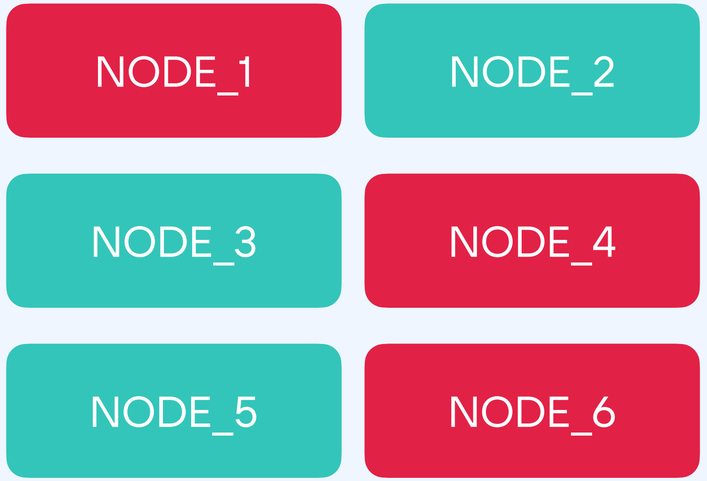


Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text,
  PRIMARY KEY ((id_1, id_2), id_3)
) …
```

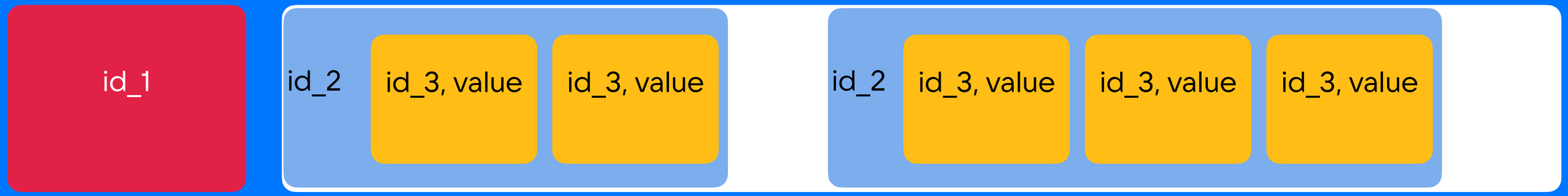
Primary key = Partition Key + Clustering Key

Partition Key - отвечает за выбор ноды



Primary key:

```
Create table test_table (
   id_1 bigint,
   id_2 bigint,
   id_3 bigint,
   value text
   PRIMARY KEY ((id_1), id_2, id_3)
) …
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
      AND id_3 = 3
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
      AND id_3 >= 3
```

Primary key:

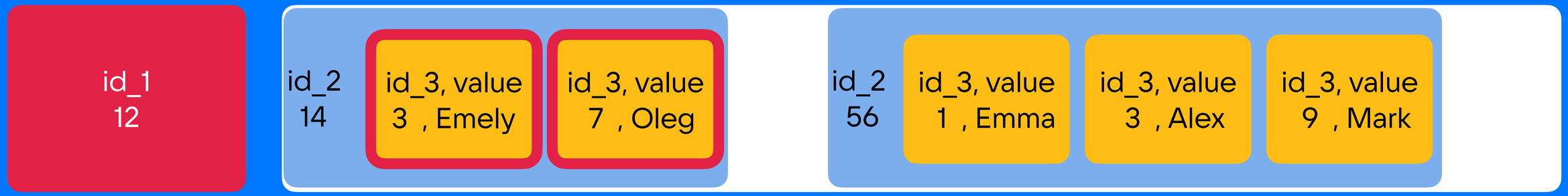
```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
      AND id_3 >= 3
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
      AND id_3 >= 3
```

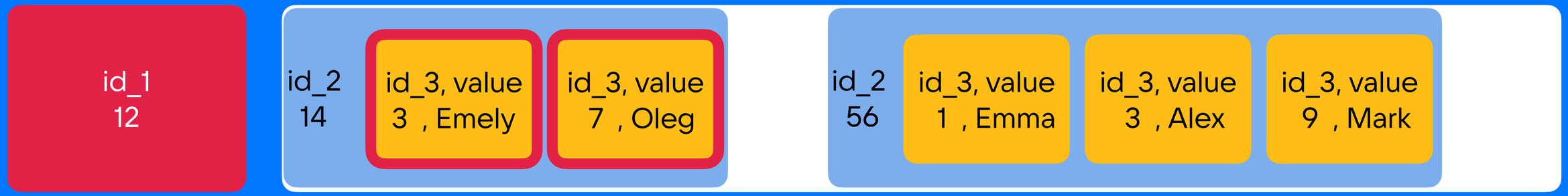


Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
      AND id_3 >= 3
```

```
cqlsh:test> Select * from test_table WHERE id_1 = 12 AND id_2 = 14 AND id_3 >= 3;

 id_1 | id_2 | id_3 | value
------+------+------+-------
   12 |   14 |    3 | Emely
   12 |   14 |    7 |  Oleg

(2 rows)
```



Primary key:

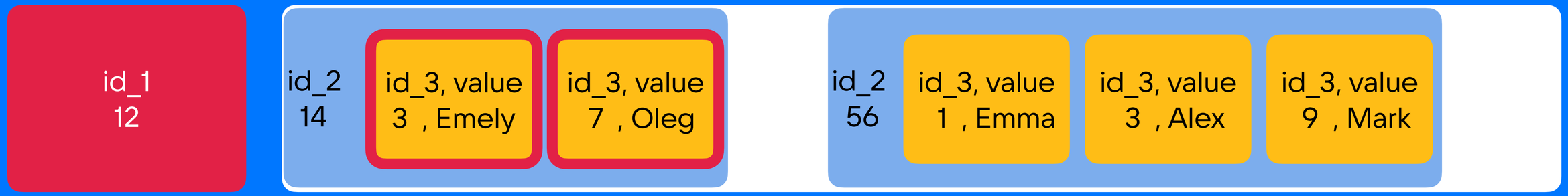
```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
```

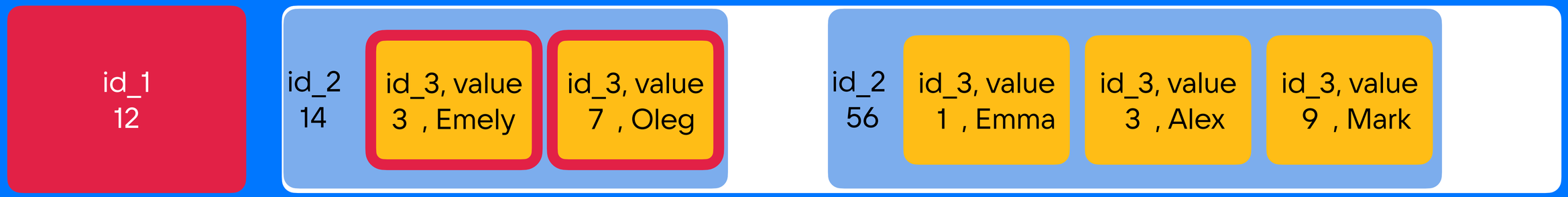


Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 = 14
```

```
cqlsh:test> Select * from test_table WHERE id_1 = 12 AND id_2 = 14;

 id_1 | id_2 | id_3 | value
------+------+------+-------
   12 |   14 |    3 | Emely
   12 |   14 |    7 |  Oleg

(2 rows)
```



Primary key:

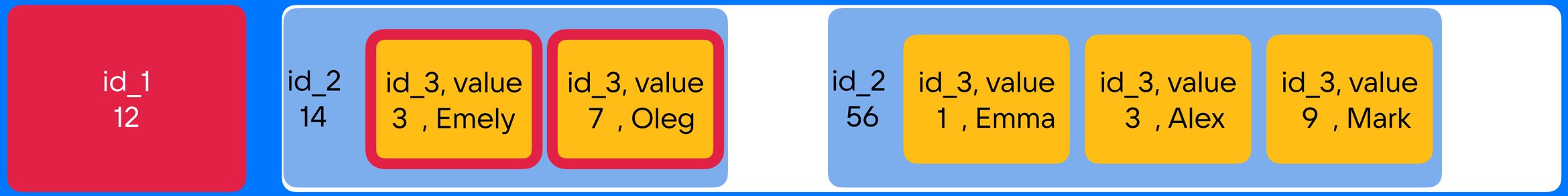
```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

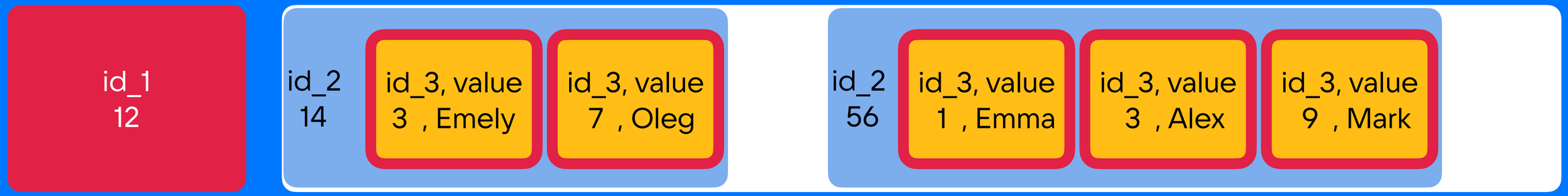
```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
```

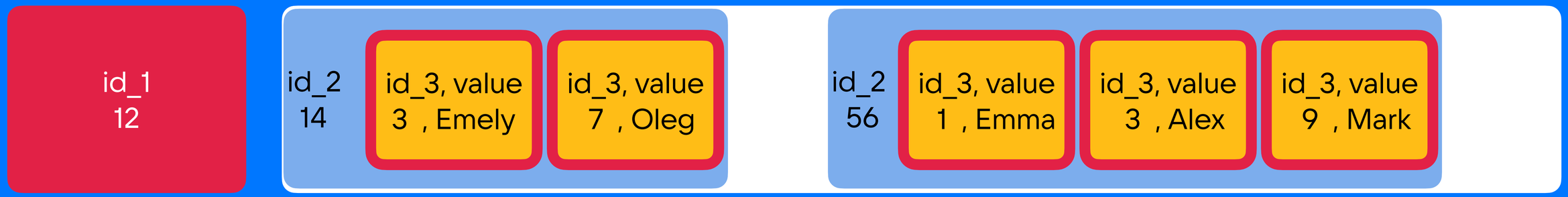


Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
```

```
cqlsh:test> Select * from test_table WHERE id_1 = 12 AND id_2 >= 14;

 id_1 | id_2 | id_3 | value
------+------+------+-------
   12 |   14 |    3 | Emely
   12 |   14 |    7 |  Oleg
   12 |   56 |    1 |  Emma
   12 |   56 |    3 |  Alex
   12 |   56 |    9 |  Mark

(5 rows)
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14
```

ERROR

id_1
12

id_2
14

id_3, value
3 , Emely

id_3, value
7 , Oleg

id_2
56

id_3, value
1 , Emma

id_3, value
3 , Alex

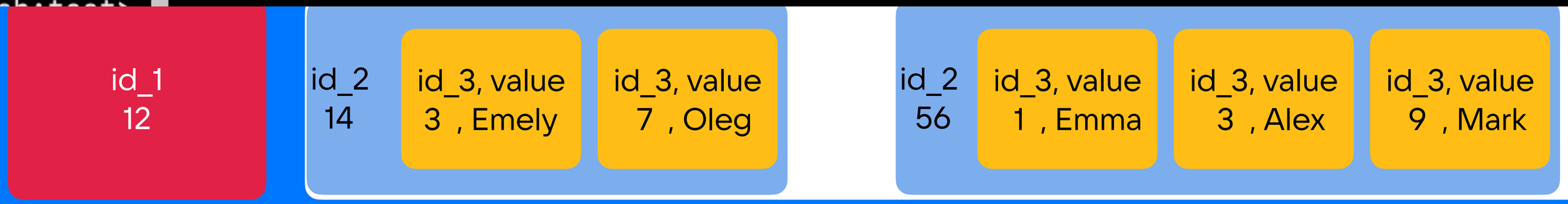
id_3, value
9 , Mark

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14
```

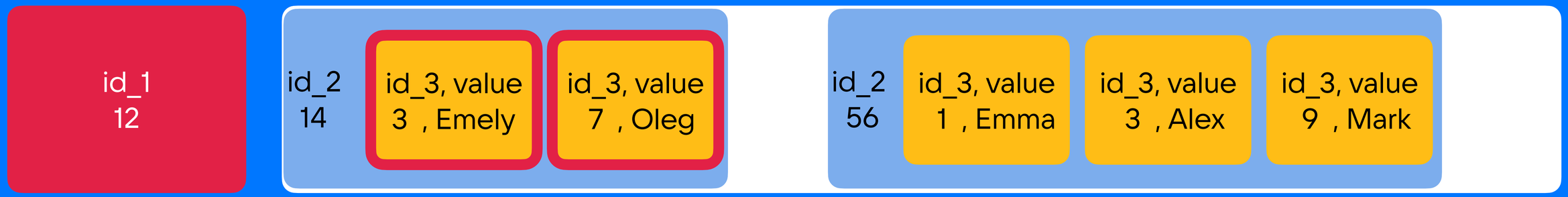
```
cqlsh:test> Select * from test_table WHERE id_2 = 14;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it mig
ht involve data filtering and thus may have unpredictable performance. If you want to execute this query
despite the performance unpredictability, use ALLOW FILTERING"
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14 ALLOW FILTERING
```

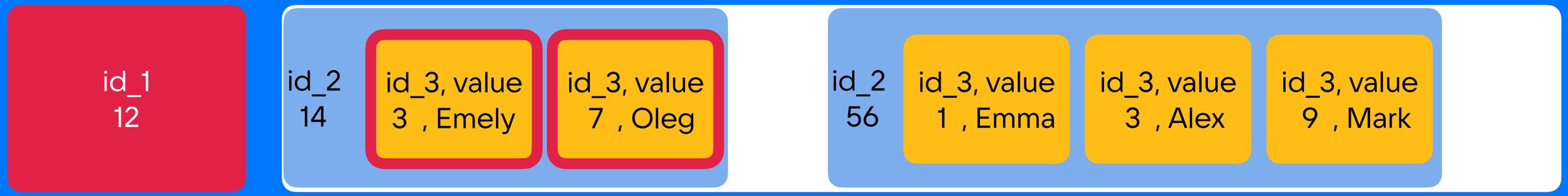


Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14 ALLOW FILTERING
```

```
cqlsh:test> Select * from test_table WHERE id_2 = 14 ALLOW FILTERING;

 id_1 | id_2 | id_3 | value
------+------+------+-------
   12 |   14 |    3 | Emely
   12 |   14 |    7 |  Oleg
   13 |   14 |    9 |   Max
   13 |   14 |   15 | Kelly

(4 rows)
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14 ALLOW FILTERING
```

NODE_1	NODE_2	NODE_3
NODE_4	NODE_5	NODE_6
NODE_7	NODE_8	NODE_9

id_1
12

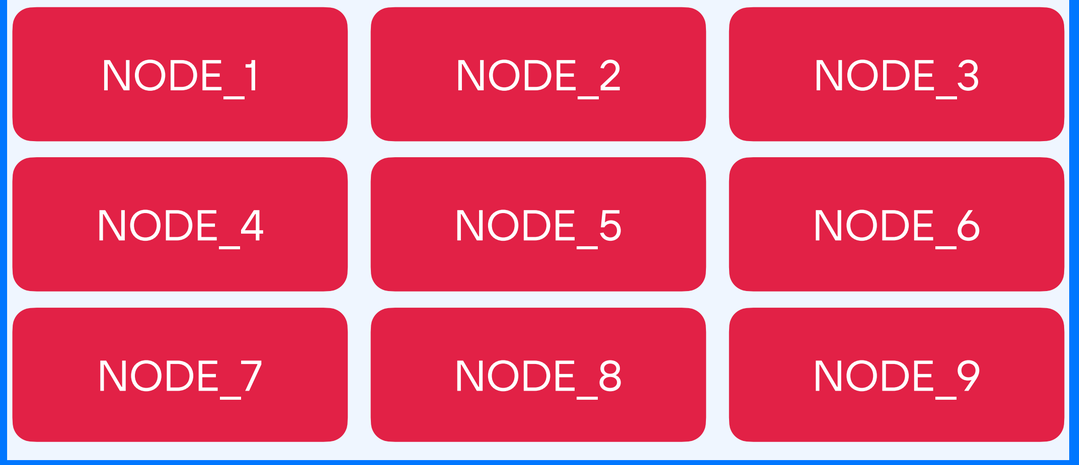
id_2		
14	id_3, value 3 , Emely	id_3, value 7 , Oleg

id_2			
56	id_3, value 1 , Emma	id_3, value 3 , Alex	id_3, value 9 , Mark

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14 ALLOW FILTERING
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

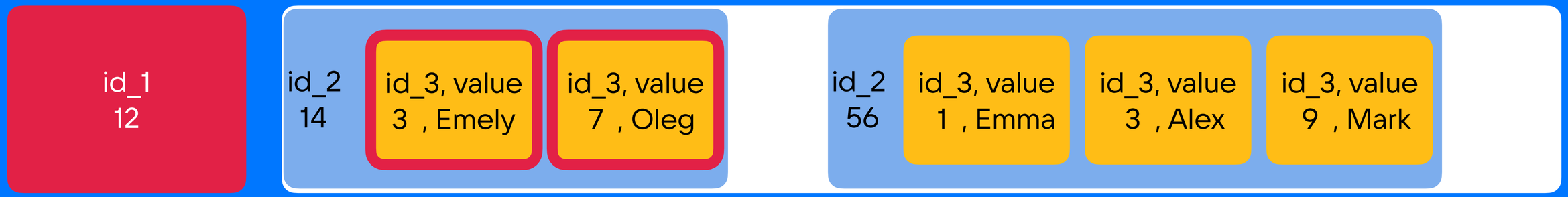
```
Select * from test_table
    WHERE id_2 = 14 ALLOW FILTERING
```

NODE_1	NODE_2	NODE_3
NODE_4	NODE_5	NODE_6
NODE_7	NODE_8	NODE_9

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14 ALLOW FILTERING
```



Поговорим про индексы в кассандра

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14
```

Вторичные индексы:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
CREATE INDEX ON test_table (id_2);
```

Вторичные индексы: find id_2=1

A

id_2	id_1	id_3	value
1	1	0	Value
2	1	0	Value
3	1	0	Value

id_1	id_2	id_3	value
1	1	0	Value
1	2	0	Value
1	3	0	Value

B

id_2	id_1	id_3	value
0	6	0	Value
2	6	0	Value
3	6	0	Value

id_1	id_2	id_3	value
6	1	0	Value
6	2	0	Value
6	3	0	Value

C

id_2	id_1	id_3	value
1	5	0	Value
2	5	0	Value
3	5	0	Value

id_1	id_2	id_3	value
5	1	0	Value
5	2	0	Value
5	3	0	Value

C

id_1	id_2	id_3	value
2	1	0	Value
2	2	0	Value
2	3	0	Value

id_2	id_1	id_3	value
1	2	0	Value
2	2	0	Value
3	2	0	Value

D

id_1	id_2	id_3	value
3	1	0	Value
3	2	0	Value
3	3	0	Value

id_2	id_1	id_3	value
1	3	0	Value
2	3	0	Value
3	3	0	Value

E

id_1	id_2	id_3	value
4	1	0	Value
4	2	0	Value
4	3	0	Value

id_2	id_1	id_3	value
0	4	0	Value
2	4	0	Value
3	4	0	Value

Вторичные индексы: find id_2=1

A

id_2	id_1	id_3	value
1	1	0	Value
2	1	0	Value
3	1	0	Value

id_1	id_2	id_3	value
1	1	0	Value
1	2	0	Value
1	3	0	Value

B

id_2	id_1	id_3	value
0	6	0	Value
2	6	0	Value
3	6	0	Value

id_1	id_2	id_3	value
6	1	0	Value
6	2	0	Value
6	3	0	Value

C

id_2	id_1	id_3	value
1	5	0	Value
2	5	0	Value
3	5	0	Value

id_1	id_2	id_3	value
5	1	0	Value
5	2	0	Value
5	3	0	Value

C

id_1	id_2	id_3	value
2	1	0	Value
2	2	0	Value
2	3	0	Value

id_2	id_1	id_3	value
1	2	0	Value
2	2	0	Value
3	2	0	Value

D

id_1	id_2	id_3	value
3	1	0	Value
3	2	0	Value
3	3	0	Value

id_2	id_1	id_3	value
1	3	0	Value
2	3	0	Value
3	3	0	Value

E

id_1	id_2	id_3	value
4	1	0	Value
4	2	0	Value
4	3	0	Value

id_2	id_1	id_3	value
0	4	0	Value
2	4	0	Value
3	4	0	Value

Вторичные индексы: find id_2=1

A

id_2	id_1	id_3	value
1	1	0	Value
2	1	0	Value
3	1	0	Value

id_1	id_2	id_3	value
1	1	0	Value
1	2	0	Value
1	3	0	Value

B

id_2	id_1	id_3	value
0	6	0	Value
2	6	0	Value
3	6	0	Value

id_1	id_2	id_3	value
6	1	0	Value
6	2	0	Value
6	3	0	Value

C

id_2	id_1	id_3	value
1	5	0	Value
2	5	0	Value
3	5	0	Value

id_1	id_2	id_3	value
5	1	0	Value
5	2	0	Value
5	3	0	Value

id_1	id_2	id_3	value
2	1	0	Value
2	2	0	Value
2	3	0	Value

id_2	id_1	id_3	value
1	2	0	Value
2	2	0	Value
3	2	0	Value

C

id_1	id_2	id_3	value
3	1	0	Value
3	2	0	Value
3	3	0	Value

id_2	id_1	id_3	value
1	3	0	Value
2	3	0	Value
3	3	0	Value

D

id_1	id_2	id_3	value
4	1	0	Value
4	2	0	Value
4	3	0	Value

id_2	id_1	id_3	value
0	4	0	Value
2	4	0	Value
3	4	0	Value

E

Вторичные индексы

- Не поддерживает индексирование по нескольким полям
- Участвует во всех процессах распределения данных вместе с основной таблицей
- Официально не рекомендуется для большинства кейсов

Secondary indexes are tricky to use and can impact performance greatly. The index table is stored on each node in a cluster, so a query involving a secondary index can rapidly become a performance nightmare if multiple nodes are accessed. A general rule of thumb is to index a column with low cardinality of few values. Before creating an index, be aware of when and when not to create an index.

https://docs.datastax.com/en/cql-oss/3.x/cql/cql_using/useSecondaryIndex.html

SSTable Attached Secondary Index (SASI) (3.11.5):

```
CREATE CUSTOM INDEX ON test_table (id_2) USING
‘org.apache.cassandra.index.sasi.SASIIndex';
```

Sasi:

A

id_1	id_2	id_3	value
1	0	0	Value
1	3	0	Value
1	4	0	Value

B

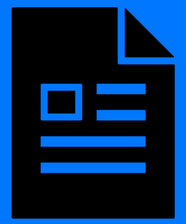
id_1	id_2	id_3	value
6	1	0	Value
6	2	0	Value
6	3	0	Value

C

id_1	id_2	id_3	value
5	1	0	Value
5	1	1	Value
5	3	0	Value

id_1	id_2	id_3	value
2	1	0	Value
2	2	0	Value
2	3	0	Value

C



id_1	id_2	id_3	value
3	0	0	Value
3	2	0	Value
3	3	0	Value

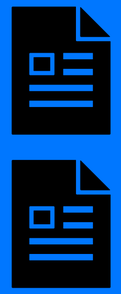
D

id_1	id_2	id_3	value
4	1	0	Value
4	1	5	Value
4	3	0	Value

E

Sasi:

A



id_1	id_2	id_3	value
1	0	0	Value

1	3	0	Value
1	4	0	Value

C

id_1	id_2	id_3	value
2	1	0	Value

2	2	0	Value
2	3	0	Value

B

id_1	id_2	id_3	value
6	1	0	Value

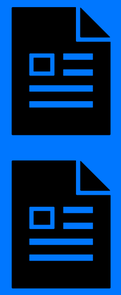
6	2	0	Value
6	3	0	Value

D

id_1	id_2	id_3	value
3	0	0	Value

3	2	0	Value
3	3	0	Value

C



id_1	id_2	id_3	value
5	1	0	Value

5	1	1	Value
5	3	0	Value

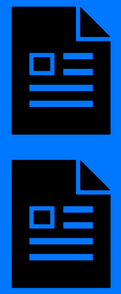
E

id_1	id_2	id_3	value
4	1	0	Value

4	1	5	Value
4	3	0	Value

Sasi: find id_2=1

A



id_1	id_2	id_3	value
1	0	0	Value

1	3	0	Value
1	4	0	Value

B

id_1	id_2	id_3	value
6	1	0	Value

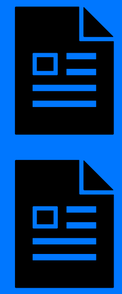
6	2	0	Value
6	3	0	Value

C

id_1	id_2	id_3	value
5	1	0	Value

5	1	1	Value
5	3	0	Value

C



id_1	id_2	id_3	value
2	1	0	Value

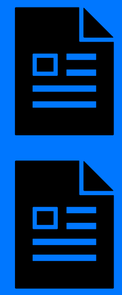
2	2	0	Value
2	3	0	Value

D

id_1	id_2	id_3	value
3	0	0	Value

3	2	0	Value
3	3	0	Value

E

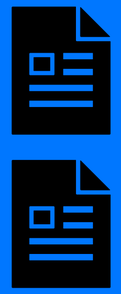


id_1	id_2	id_3	value
4	1	0	Value

4	1	5	Value
4	3	0	Value

Sasi: find id_2=1

A



id_1	id_2	id_3	value
1	0	0	Value

1	3	0	Value
1	4	0	Value

C

id_1	id_2	id_3	value
2	1	0	Value

2	2	0	Value
2	3	0	Value

B

id_1	id_2	id_3	value
6	1	0	Value

6	2	0	Value
6	3	0	Value

D

id_1	id_2	id_3	value
3	0	0	Value

3	2	0	Value
3	3	0	Value

C

id_1	id_2	id_3	value
5	1	0	Value

5	1	1	Value
5	3	0	Value

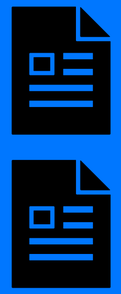
E

id_1	id_2	id_3	value
4	1	0	Value

4	1	5	Value
4	3	0	Value

Sasi: find id_2=1

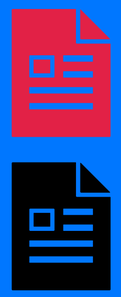
A



id_1	id_2	id_3	value
1	0	0	Value

1	3	0	Value
1	4	0	Value

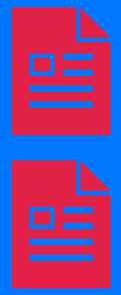
B



id_1	id_2	id_3	value
6	1	0	Value

6	2	0	Value
6	3	0	Value

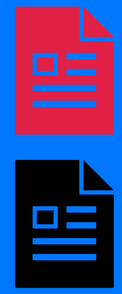
C



id_1	id_2	id_3	value
5	1	0	Value

5	1	1	Value
5	3	0	Value

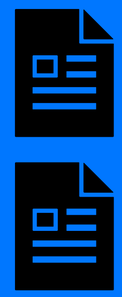
C



id_1	id_2	id_3	value
2	1	0	Value

2	2	0	Value
2	3	0	Value

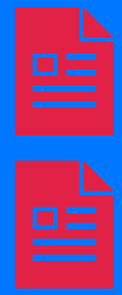
D



id_1	id_2	id_3	value
3	0	0	Value

3	2	0	Value
3	3	0	Value

E



id_1	id_2	id_3	value
4	1	0	Value

4	1	5	Value
4	3	0	Value

Secondary indexes are tricky to use and can impact performance greatly. The index table is stored on each node in a cluster, so a query involving a secondary index can rapidly become a performance nightmare if multiple nodes are accessed. A general rule of thumb is to index a column with low cardinality of few values. Before creating an index, be aware of when and when not to create an index.

https://docs.datastax.com/en/cql-oss/3.x/cql/cql_using/useSecondaryIndex.html

Мы рассмотрели

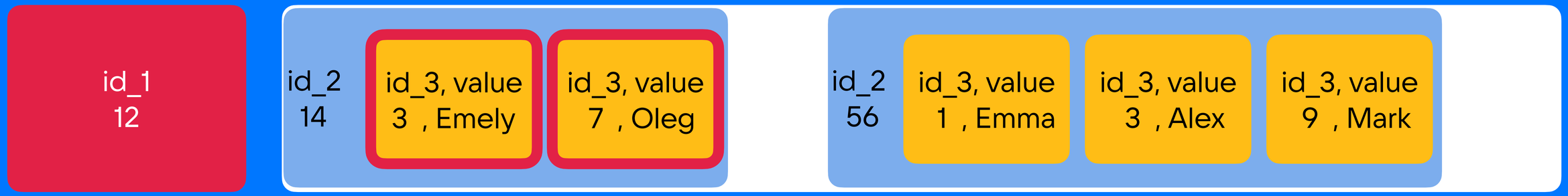
- Allow filtering
- Вторичные индексы
- SASI

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_2 = 14
```

ERROR



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

1. обязательно должен быть partition key в запросе

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 9
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 9
```

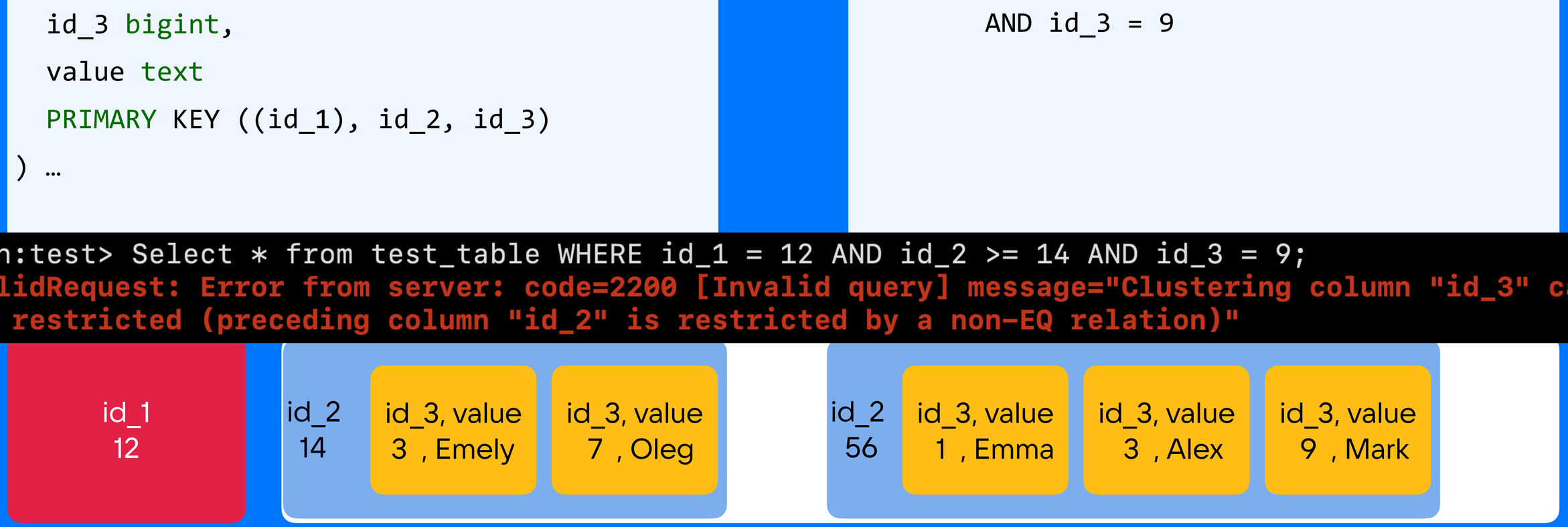
ERROR

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 9
```

```
cqlsh:test> Select * from test_table WHERE id_1 = 12 AND id_2 >= 14 AND id_3 = 9;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Clustering column "id_3" cannot be restricted (preceding column "id_2" is restricted by a non-EQ relation)"
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 9
      ALLOW FILTERING;
```

Primary key:

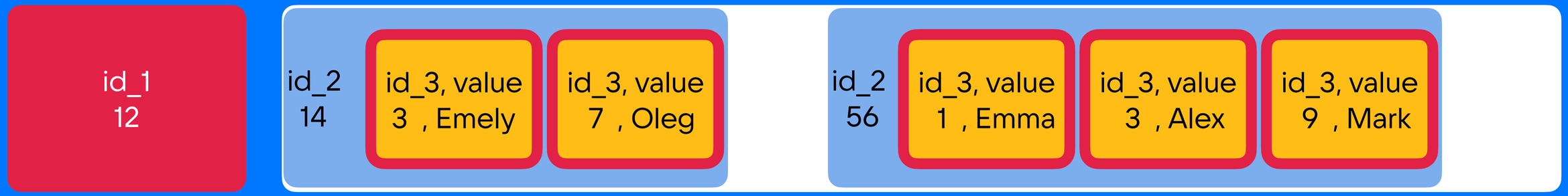
```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 9
      ALLOW FILTERING;
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 9
      ALLOW FILTERING;
```



Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE id_1 = 12
      AND id_2 >= 14
      AND id_3 = 3
      ALLOW FILTERING;
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE value = ‘Alex’
```

ERROR

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

```
Select * from test_table
    WHERE value = ‘Alex’
      ALLOW FILTERING;
```

```
cqlsh:test> Select * from test_table WHERE value = 'Alex' ALLOW FILTERING;

 id_1 | id_2 | id_3 | value
------+------+------+-------
   12 |   56 |    3 |  Alex

(1 rows)
```

Primary key:

```
Create table test_table (
  id_1 bigint,
  id_2 bigint,
  id_3 bigint,
  value text
  PRIMARY KEY ((id_1), id_2, id_3)
) …
```

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Пример 1

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```


```
Create index audit_by_user
ON audit (user_id, timestamp);

Create index audit_by_ts
ON audit (timestamp);
```


```
1) Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
2) SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  id bigint,
  user_id bigint,
  action int,
  timestamp bigint
  PRIMARY KEY (id)
) …
```


```
1) Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
2) SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  id bigint,
  user_id bigint,
  action int,
  timestamp bigint
  PRIMARY KEY ((?), ?)
) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  id bigint,
  user_id bigint,
  action int,
  timestamp bigint
  PRIMARY KEY ((user_id), timestamp, id)
) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp bigint
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```

Жил был audit на SQL

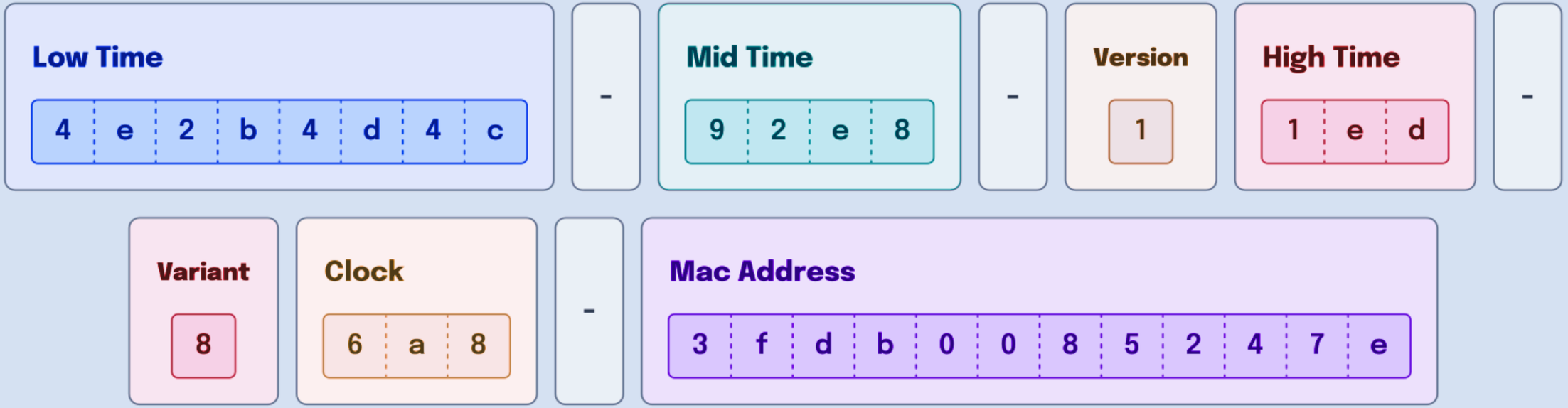
```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp bigint
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```

UUIDv1

EG: 4e2b4d4c-92e8-11ed-86a8-3fdb0085247e



Input Data

Time: 2023-01-13 2:16:35 (UTC)

Mac Address: 3f:db:00:85:24:7e

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Select * from table audit WHERE user_id = ? AND timestamp > ? order by timestamp
```


```
Select * from table c_audit WHERE user_id = ? AND timestamp > ?
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```


```
SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  ? ?,
  timestamp timeuuid,
  PRIMARY KEY ((?), timestamp, user_id,
  action)
) …
```


```
SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```


```
SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Жил был audit на SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```


```
SELECT * FROM audit WHERE timestamp > ? order by timestamp
```


```
SELECT * FROM c_audit_for_all WHERE month = ? AND timestamp > ?
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

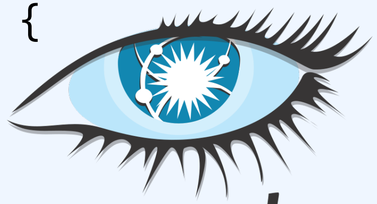
А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

Нужно изучить какой будет размер ряда, т.е. перед проектированием надо узнать сколько событий будет в месяц/день/год

А почему именно месяц? А не день? Или год?

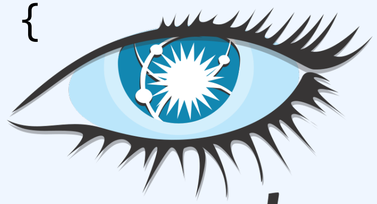
```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  day data,
  timestamp timeuuid,
  PRIMARY KEY ((day), timestamp, user_id,
  action)
) …
```



cassandra

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  day data,
  timestamp timeuuid,
  PRIMARY KEY ((day), timestamp, user_id,
  action)
) …
```



day	timestamp	user_id	action
2.02.2023	11:14	555	1
4.02.2023	15:42	32	2
	19:31	555	4
5.02.2023	08:44	555	1
6.02.2023	12:35	555	1
7.02.2023	18:40	555	1
8.02.2023	09:23	555	1
9.02.2023	09:30	555	1
10.02.2023	19:10	555	1
11.02.2023	04:58	555	1

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  day data,
  timestamp timeuuid,
  PRIMARY KEY ((day), timestamp, user_id,
  action)
) …
```

day	timestamp	user_id	action
2.02.2023	11:14	555	1
4.02.2023	15:42	32	2
	19:31	555	4
5.02.2023	08:44	555	1
6.02.2023	12:35	555	1
7.02.2023	18:40	555	1
8.02.2023	09:23	555	1
9.02.2023	09:30	555	1
10.02.2023	19:10	555	1
11.02.2023	04:58	555	1


```
SELECT * FROM c_audit_for_all WHERE day = 2.02.2023
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  day data,
  timestamp timeuuid,
  PRIMARY KEY ((day), timestamp, user_id,
  action)
) …
```

day	timestamp	user_id	action
2.02.2023	11:14	555	1
4.02.2023	15:42	32	2
	19:31	555	4
5.02.2023	08:44	555	1
6.02.2023	12:35	555	1
7.02.2023	18:40	555	1
8.02.2023	09:23	555	1
9.02.2023	09:30	555	1
10.02.2023	19:10	555	1
11.02.2023	04:58	555	1


```
SELECT * FROM c_audit_for_all WHERE day = 2.02.2023
SELECT * FROM c_audit_for_all WHERE day = 3.02.2023
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  day data,
  timestamp timeuuid,
  PRIMARY KEY ((day), timestamp, user_id,
  action)
) …
```

day	timestamp	user_id	action
2.02.2023	11:14	555	1
4.02.2023	15:42	32	2
	19:31	555	4
5.02.2023	08:44	555	1
6.02.2023	12:35	555	1
7.02.2023	18:40	555	1
8.02.2023	09:23	555	1
9.02.2023	09:30	555	1
10.02.2023	19:10	555	1
11.02.2023	04:58	555	1


```
SELECT * FROM c_audit_for_all WHERE day = 2.02.2023
SELECT * FROM c_audit_for_all WHERE day = 3.02.2023
SELECT * FROM c_audit_for_all WHERE day = 4.02.2023
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  day data,
  timestamp timeuuid,
  PRIMARY KEY ((day), timestamp, user_id,
  action)
) …
```

day	timestamp	user_id	action
2.02.2023	11:14	555	1
4.02.2023	15:42	32	2
	19:31	555	4
5.02.2023	08:44	555	1
6.02.2023	12:35	555	1
7.02.2023	18:40	555	1
8.02.2023	09:23	555	1
9.02.2023	09:30	555	1
10.02.2023	19:10	555	1
11.02.2023	04:58	555	1


```
SELECT * FROM c_audit_for_all WHERE day = 2.02.2023
SELECT * FROM c_audit_for_all WHERE day = 3.02.2023
SELECT * FROM c_audit_for_all WHERE day = 4.02.2023
SELECT * FROM c_audit_for_all WHERE day = 5.02.2023
SELECT * FROM c_audit_for_all WHERE day = 6.02.2023
SELECT * FROM c_audit_for_all WHERE day = 7.02.2023
                                                ...
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

Вводные данные:

- 1 триллион событий в год

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

Вводные данные:

- 1 триллион событий в год


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

NODE_1	NODE_2	NODE_3
NODE_4	NODE_5	NODE_6
NODE_7	NODE_8	NODE_9
NODE_10	NODE_11	NODE_12


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

NODE_1	NODE_2	NODE_3
NODE_4	NODE_5	NODE_6
NODE_7	NODE_8	NODE_9
NODE_10	NODE_11	NODE_12


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

Вводные данные:

- 1 триллион событий в год

Минусы решения:

Не равномерное распределение данных по нодам


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

Вводные данные:

- 1 триллион событий в год

Минусы решения:

Не равномерное распределение данных по нодам

Дорогой репеир


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

Вводные данные:

- 1 триллион событий в год

Минусы решения:

Не равномерное распределение данных по нодам

Дорогой репеир

Тяжелый компакшн


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  year data,
  timestamp timeuuid,
  PRIMARY KEY ((year), timestamp, user_id,
  action)
) …
```

Вводные данные:

- 1 триллион событий в год

Минусы решения:

Не равномерное распределение данных по нодам

Дорогой репеир

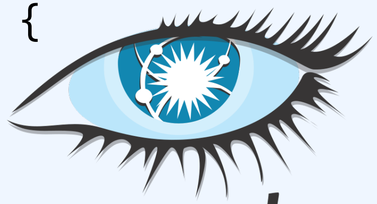
Тяжелый компакшн

Дольше поиск нужного элемента


```
SELECT * FROM c_audit_for_all WHERE year = ? AND timestamp > ? LIMIT 20
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```




```
SELECT * FROM c_audit_for_all WHERE month = ? AND timestamp > ?
```

А почему именно месяц? А не день? Или год?

```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

Calculating partition size

For efficient operation, partitions must be sized within certain limits. Two measures of partition size are the number of values in a partition and the partition size on disk. The maximum number of columns per row is two billion. Sizing the disk space is more complex, and involves the number of rows and the number of columns, primary key columns and static columns in each table. Each application will have different efficiency parameters, but a good rule of thumb is to keep the maximum number of values below 100,000 items and the disk size under 100MB.

Это жил аудит в SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```


```
Create index audit_by_user
ON audit (user_id, timestamp);

Create index audit_by_ts
ON audit (timestamp);
```


```
1) Select * from table audit WHERE user_id = ? AND timestamp > ?order by timestamp
2) SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Но сейчас уже он в Cassandra

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```


```
1) Select * from table c_audit WHERE user_id = ?AND timestamp > ?
2) SELECT * FROM c_audit_for_all WHERE month = ? AND timestamp > ?
```

2 таблицы вместо 1:

Минусы:

- придется поддерживать консистентно 2 таблицы вместо 1

2 таблицы вместо 1:

Минусы:
- придется поддерживать консистентно 2 таблицы вместо 1

Плюсы:
- Сможем выполнять запрос быстро

Поддержка консистентности:

Как не стоит делать:

```
INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
```

Bench:

```
INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
```

Bench:

```
INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
```

Дано:

Cassandra 4.1

Bench:

```
INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
```

Дано:

Cassandra 4.1

6 нод

Bench:

```
INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
```

Дано:

Cassandra 4.1

6 нод

20 клиентов (cassandra-stress с доработками)

Bench:

```
INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
```

Дано:

Cassandra 4.1

6 нод

20 клиентов (cassandra-stress с доработками)

Ops: 100к

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations		

Batching inserts, updates and deletes:

```
BEGIN BATCH
  INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
  INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
APPLY BATCH;
```

https://cassandra.apache.org/doc/4.1/cassandra/cql/dml.html#batch_statement

Batching inserts, updates and deletes:

```
BEGIN BATCH
  INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
  INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
APPLY BATCH;
```

Атомарность?

https://cassandra.apache.org/doc/4.1/cassandra/cql/dml.html#batch_statement

Batching inserts, updates and deletes:

Атомарность?

- В рамках одной партиции атомарность в привычном понимании

https://cassandra.apache.org/doc/4.1/cassandra/cql/dml.html#batch_statement

Batching inserts, updates and deletes:

Атомарность?

- В рамках одной партиции атомарность в привычном понимании
- Иначе гарантирует, что все инструкции DML в пакете будут успешными или нет

https://cassandra.apache.org/doc/4.1/cassandra/cql/dml.html#batch_statement

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations	Сами формируем все изменения каждый раз. Медленно	84к

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations (Logged batches)	Сами формируем все изменения каждый раз. Медленно	84к

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations (Logged batches)	Сами формируем все изменения каждый раз. Медленно	84к
Batching inserts (Unlogged batches)		

Batching inserts, updates and deletes:

```
BEGIN UNLOGGED BATCH
  INSERT INTO c_audit (user_id, action, timestamp)  VALUES (?, ?, ?);
  INSERT INTO c_audit_for_all (user_id, action, month, timestamp)  VALUES (?, ?, ?, ?);
APPLY BATCH;
```

https://docs.datastax.com/en/cql-oss/3.x/cql/cql_reference/cqlBatch.html

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations (Logged batches)	Сами формируем все изменения каждый раз. Медленно	84к
Batching inserts (Unlogged batches)	Можно потерять данные	~100к

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations (Logged batches)	Сами формируем все изменения каждый раз. Медленно	84к
Batching inserts (Unlogged batches)	Можно потерять данные	~100к
MATERIALIZED VIEW		

MATERIALIZED VIEW

Определение: способ автоматического создания таблицы.

MATERIALIZED VIEW

Ограничения

MATERIALIZED VIEW

Ограничения

- MV состоит только из столбцов основной таблицы

MATERIALIZED VIEW

Ограничения

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы

https://cassandra.apache.org/doc/latest/cassandra/cql/mvs.html

MATERIALIZED VIEW

Ограничения

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

https://cassandra.apache.org/doc/latest/cassandra/cql/mvs.html

MATERIALIZED VIEW

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

MATERIALIZED VIEW

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

MATERIALIZED VIEW

```
Create table c_audit (
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

MATERIALIZED VIEW

```
Create table c_audit (
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

MATERIALIZED VIEW

```
Create table c_audit (
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

MATERIALIZED VIEW

```
Create table c_audit (
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
CREATE MATERIALIZED VIEW c_audit_for_all AS
SELECT * FROM c_audit
    WHERE user_id IS NOT NULL
    AND timestamp IS NOT NULL
    AND c2 action NOT NULL
  PRIMARY KEY ((month), timestamp, user_id,
  action);
```

- MV состоит только из столбцов основной таблицы
- Ключ MV содержит все поля из ключа таблицы
- В ключ MV можно добавить максимум одно поле

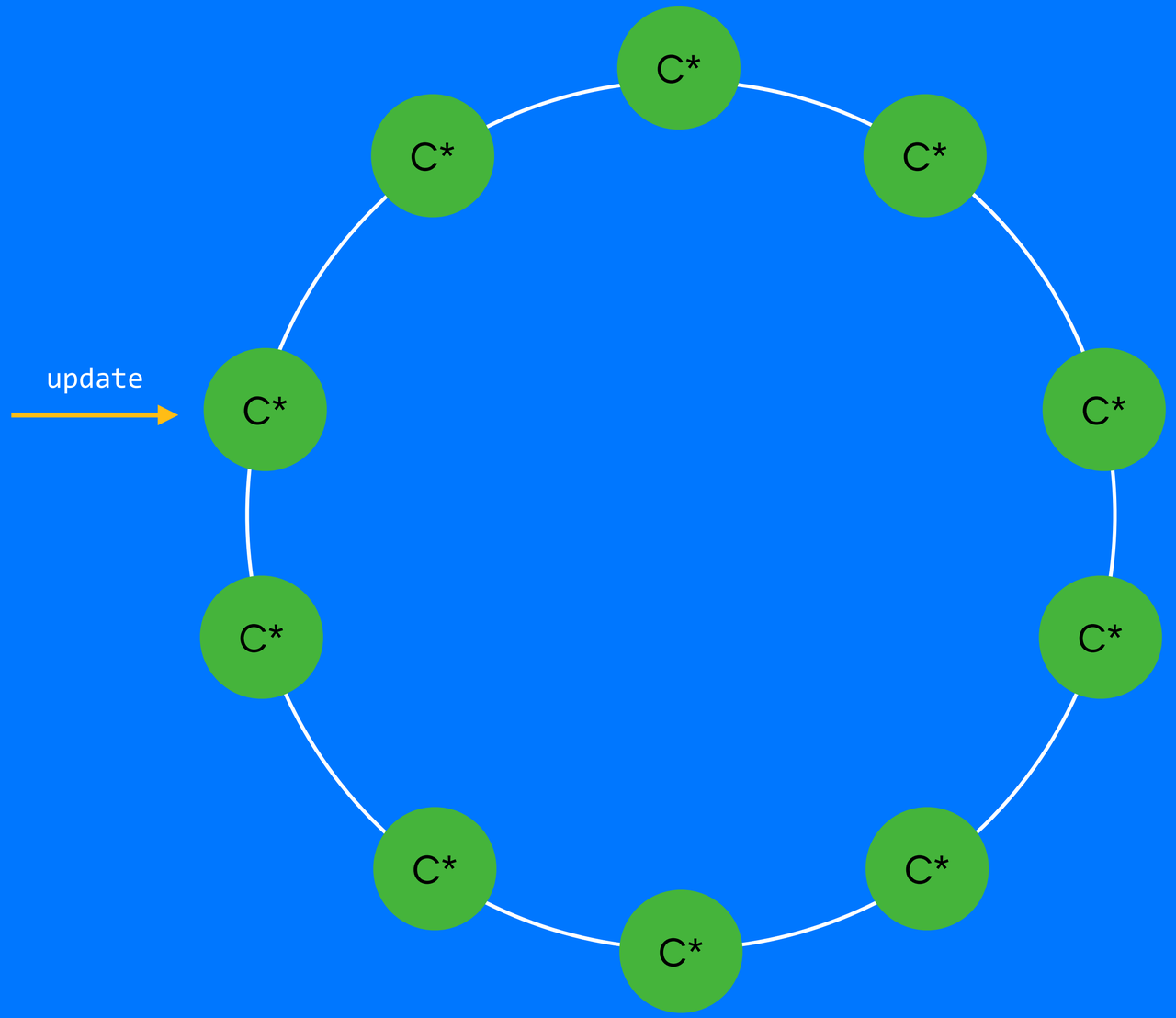
MATERIALIZED VIEW

Минусы

- Ограничения

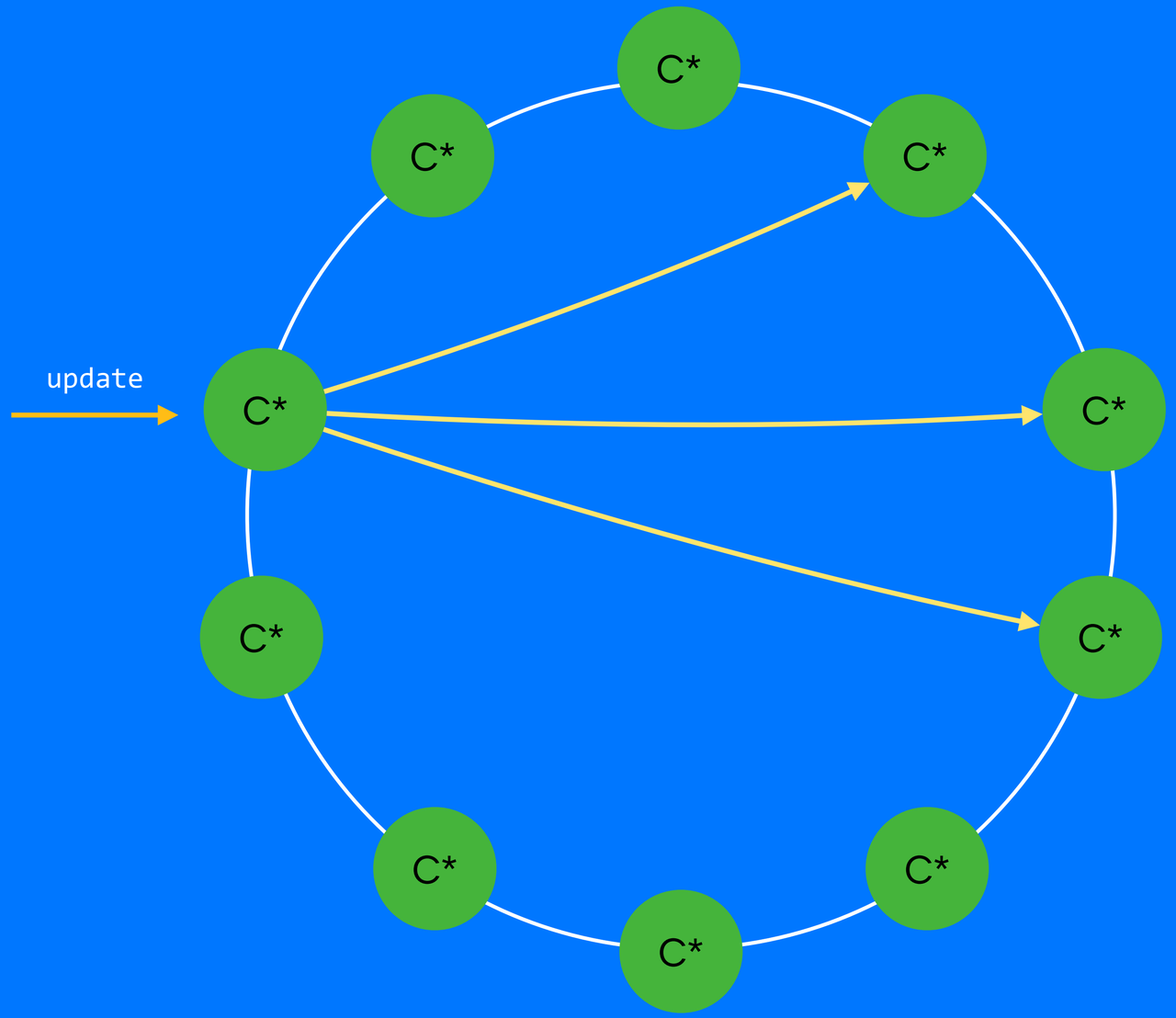
https://cassandra.apache.org/doc/latest/cassandra/cql/mvs.html

MATERIALIZED VIEW INSERT



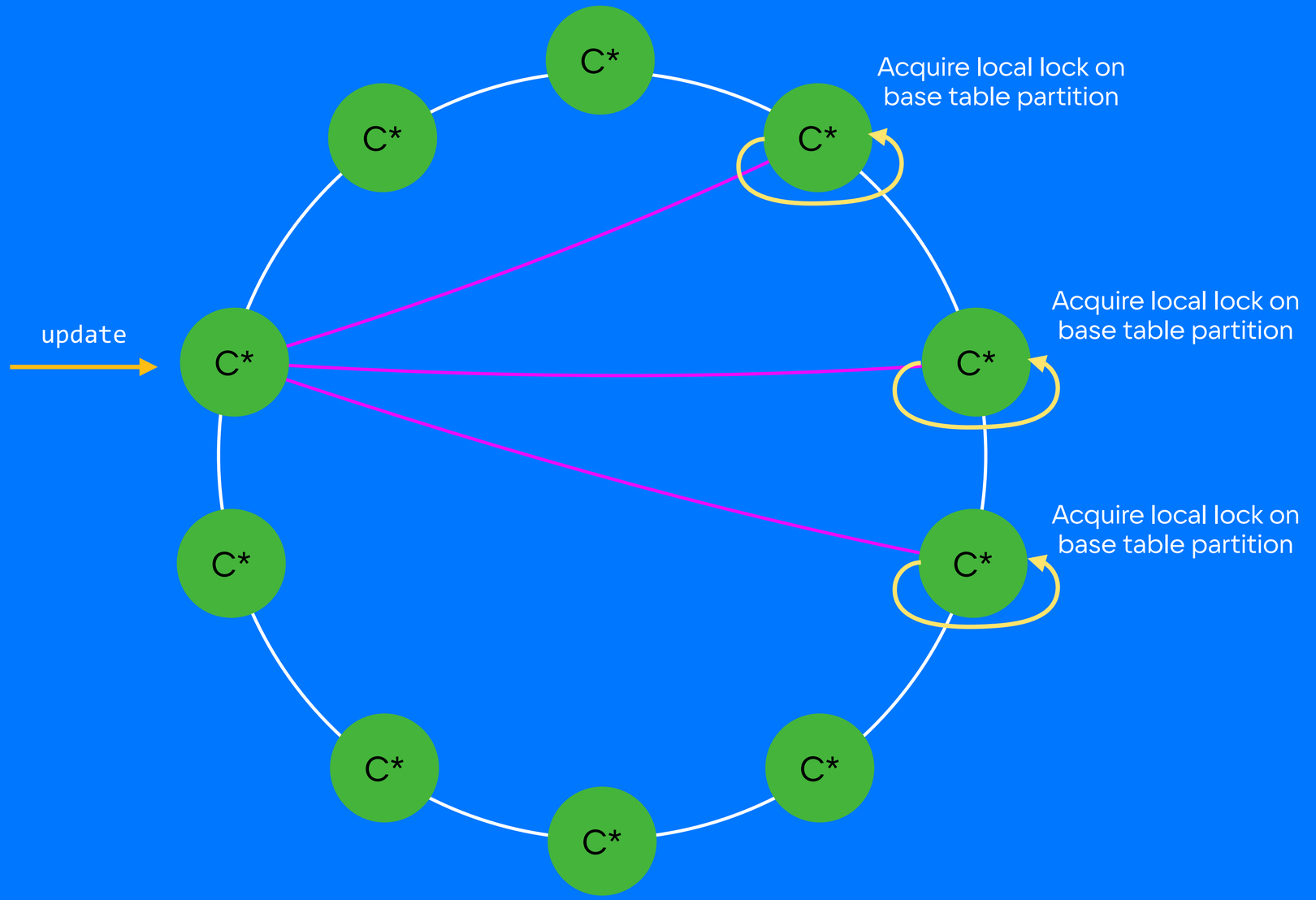
https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW INSERT



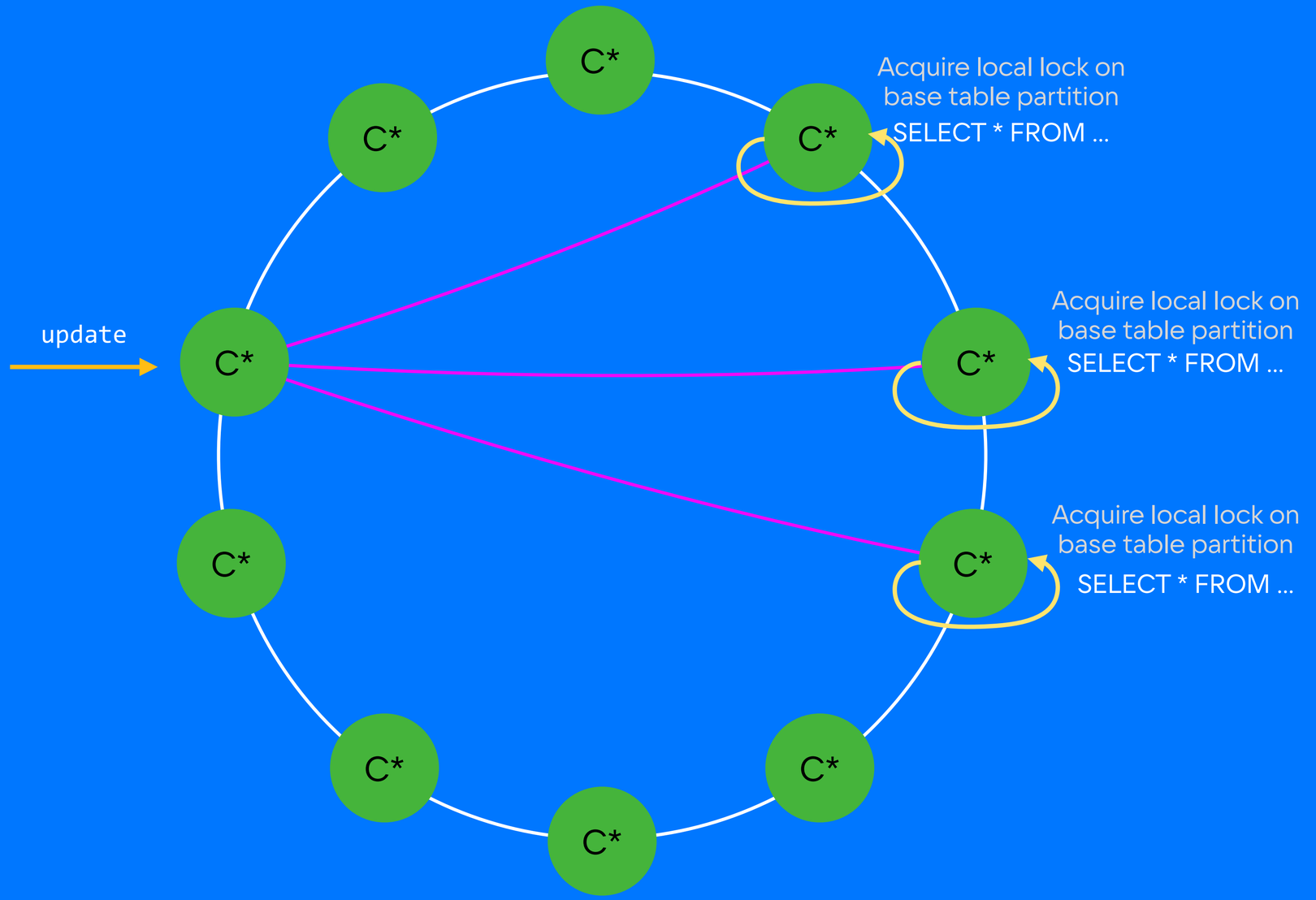
https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW INSERT



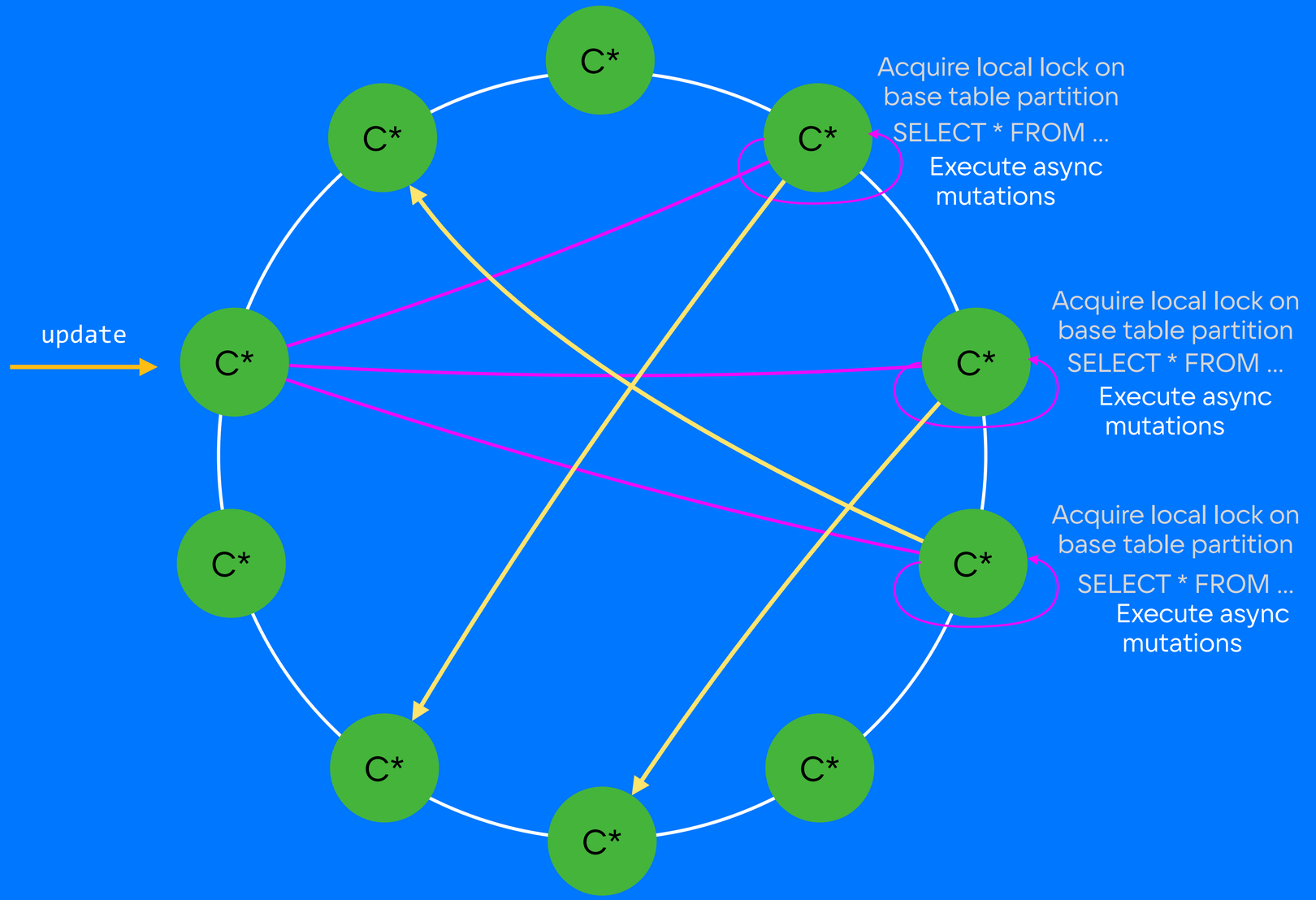
https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW INSERT



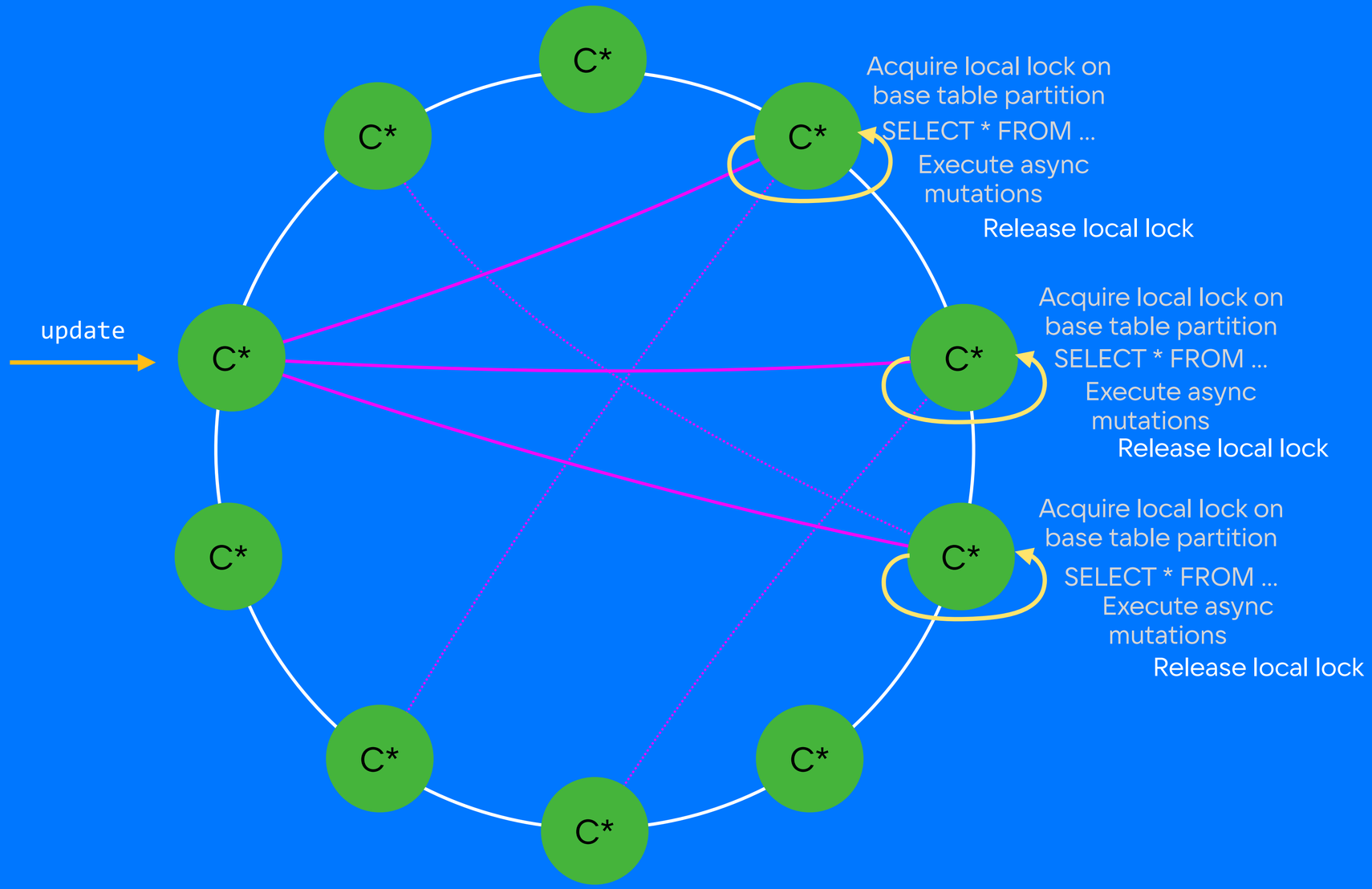
https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW INSERT



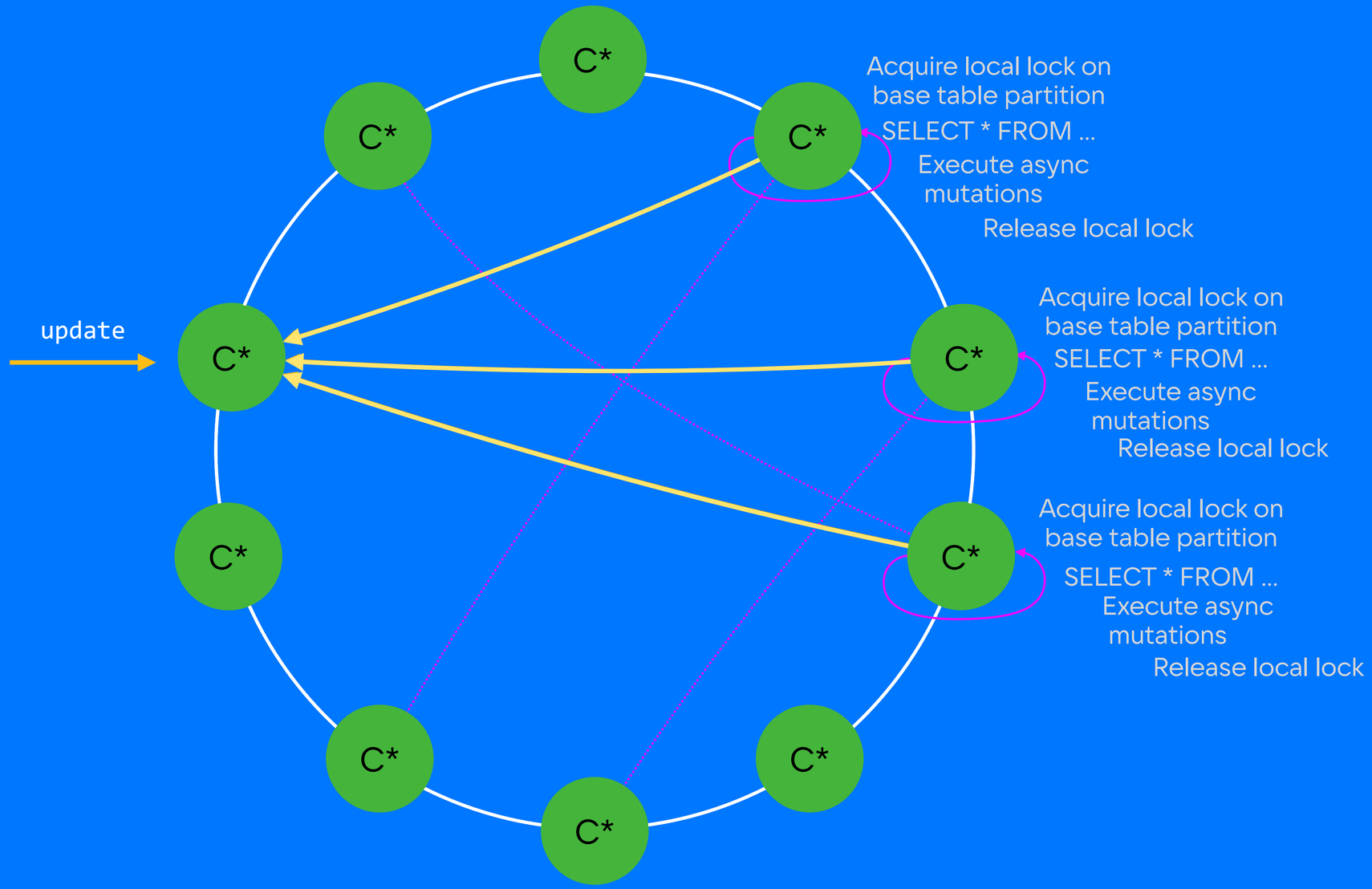
https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW INSERT



https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW INSERT



https://www.doanduyhai.com/blog/?p=1930

MATERIALIZED VIEW

Минусы

- Ограничения
- Eventually update

https://cassandra.apache.org/doc/latest/cassandra/cql/mvs.html

MATERIALIZED VIEW

Минусы

- Ограничения
- Eventually update
- «read before update»

https://cassandra.apache.org/doc/latest/cassandra/cql/mvs.html

MATERIALIZED VIEW

Минусы

- Ограничения
- Eventually update
- «read before update»
- mv_enable_coordinator_batchlog

https://cassandra.apache.org/doc/latest/cassandra/cql/mvs.html

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations (Logged batches)	Сами формируем все изменения каждый раз. Медленно	84к
Batching inserts (Unlogged batches)	Можно потерять данные	~100к
MATERIALIZED VIEW	Есть ограничения. Eventually update. Медленно	68к

Поддержка консистентности:

2 insert подрят: 100к Ops

Способ	Минусы	Bench
Batching mutations (Logged batches)	Сами формируем все изменения каждый раз. Медленно	84к
Batching inserts (Unlogged batches)	Можно потерять данные	~100к
MATERIALIZED VIEW	Есть ограничения. Eventually update. Медленно	68к
EHOT ONLY: GLOBAL INDEX	Недоступно в openSource	100к

Joker<?>

Олег Анастасьев, Одноклассники

За гранью NoSQL: NewSQL на Cassandra

Java conference Joker 2014
Russia, St.Petersburg, 20-21 October 2014

https://youtu.be/qyTj09e-EM0

Audit в Cassandra

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```


```
1) Select * from table c_audit WHERE user_id = ?AND timestamp > ?
2) SELECT * FROM c_audit_for_all WHERE month = ? AND timestamp > ?
```

Это жил аудит в SQL

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```


```
Create index audit_by_user
ON audit (user_id, timestamp);

Create index audit_by_ts
ON audit (timestamp);
```


```
1) Select * from table audit WHERE user_id = ? AND timestamp > ?order by timestamp
2) SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

А еще периодически он чистился.

```
Create table audit (
  id bigint PRIMARY KEY,
  user_id bigint,
  action int,
  timestamp bigint

) …
```


```
Create index audit_by_user
ON audit (user_id, timestamp);

Create index audit_by_ts
ON audit (timestamp);
```


```
1) Select * from table audit WHERE user_id = ? AND timestamp > ?order by timestamp
2) SELECT * FROM audit WHERE timestamp > ? order by timestamp
```

Но сейчас уже он в Cassandra

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) …
```


```
1) Select * from table c_audit WHERE user_id = ?AND timestamp > ?
2) SELECT * FROM c_audit_for_all WHERE month = ? AND timestamp > ?
```

Но сейчас уже он в Cassandra

```
Create table c_audit (
  user_id bigint,
  action int,
  timestamp timeuuid
  PRIMARY KEY ((user_id), timestamp, action)
) … default_time_to_live = 157766400; …
```


```
Create table c_audit_for_all {
  user_id bigint,
  action int,
  month data,
  timestamp timeuuid,
  PRIMARY KEY ((month), timestamp, user_id,
  action)
) … default_time_to_live = 157766400; …
```


```
1) Select * from table c_audit WHERE user_id = ?AND timestamp > ?
2) SELECT * FROM c_audit_for_all WHERE month = ? AND timestamp > ?
```

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам 

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Ограничения:

1. обязательно должен быть partition key в запросе 
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам 

А так же:

- Избавились от поля ID
- Перед созданием таблиц изучили сколько данных будет
- Разобрали способы поддержания консистентности

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

RPS на таблицу не большой

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

RPS на таблицу не большой

При падении SQL сервера, требовались действия админа, и даунтайм платежей мог быть до 1часа пока идет переключение на standby

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

SELECT * FROM payment_request WHERE partner = ? AND partner_transaction_id = ?

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_partner(
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  month data,
  partner int,
  partner_transaction_id text
  PRIMARY KEY
  (partner,partner_transaction_id)
```

SELECT * FROM payment_request WHERE partner = ? AND partner_transaction_id = ?

SELECT * FROM payment_request WHERE partner = ? AND partner_transaction_id = ?

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Множество различных запросов

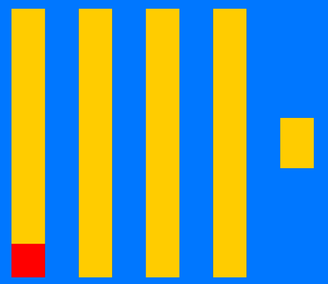
Payment request: DWH

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Payment request: DWH

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

OR



OR

Организовать надёжную синхронизацию данных из C* в OLAP хранилище

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```


```
CREATE TRIGGER kafka_trigger
    ON payment_request
    USING
        'io.smartcat.cassandra.trigger.KafkaTrigger';
```

https://youtu.be/SdQdxp1r8yU

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

https://cassandra.apache.org/doc/latest/cassandra/operating/cdc.html

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

WAL

https://cassandra.apache.org/_/blog/Learn-How-CommitLog-Works-in-Apache-Cassandra.html

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```


```
cdc_enabled
cdc_raw_directory (default: $CASSANDRA_HOME/data/cdc_raw)
```

WAL

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

`cdc_enabled`
`cdc_raw_directory` (default: `$CASSANDRA_HOME/data/cdc_raw`)

```
ALTER TABLE foo WITH cdc=true;
```

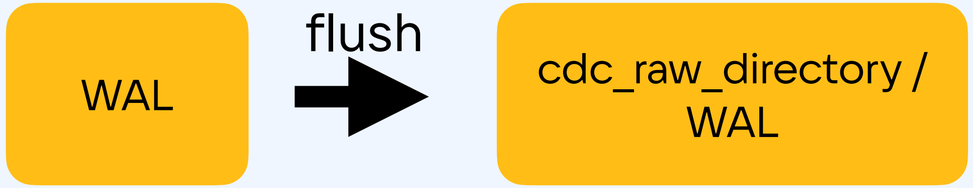
WAL

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

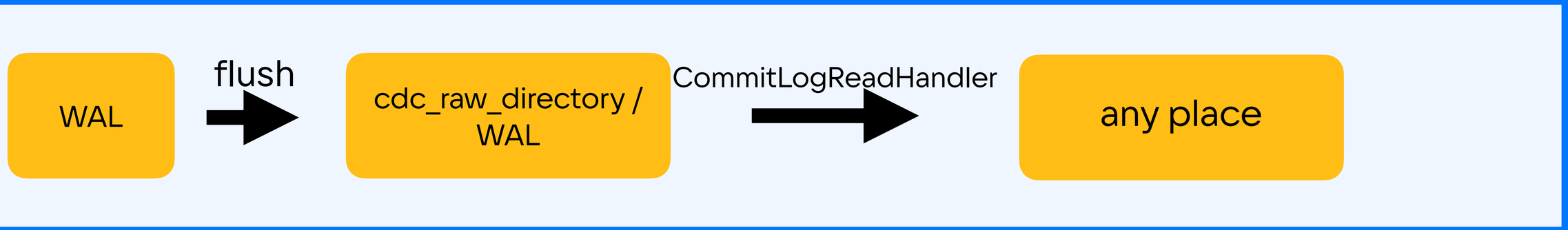
`cdc_enabled`
`cdc_raw_directory` (default: `$CASSANDRA_HOME/data/cdc_raw`)

```
ALTER TABLE foo WITH cdc=true;
```



Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```



Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Debezium Connector for Cassandra - click

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Debezium Connector for Cassandra - click

Datastax CDC → pulsar - click

Change Data Capture(CDC) через Write-Ahead Log

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Debezium Connector for Cassandra - click

Datastax CDC → pulsar - click

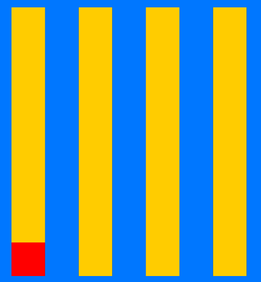
Свой

Payment request: DWH

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

CDC

OR



OR

Payment request

Payment request: разные партнеры

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

1,2,3,4,66: СМС операторы

5,49,764: банки

9000,9001,9002: электронные деньги

Payment request: разные партнеры

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

1,2,3,4,66: СМС операторы

5,49,764: банки

9000,9001,9002: электронные деньги

SELECT COUNT(*) FROM payment_request WHERE partner IN (1,2,3,4,66) AND timestamp > ?

Payment request: разные партнеры

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text
  PRIMARY KEY ((?), ?,
  partner ,partner_transaction_id)
) …
```


```
SELECT  COUNT(*) FROM payment_request WHERE partner IN (1,2,3,4,66) AND timestamp > ?
```

Payment request: разные партнеры

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text
  PRIMARY((partner), timestamp
  partner_transaction_id)
) …
```


```
SELECT  COUNT(*) FROM payment_request WHERE partner IN (1,2,3,4,66) AND timestamp > ?
```


```
SELECT  COUNT(*) FROM payment_request WHERE partner IN (1,2,3,4,66) AND timestamp > ?
```

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner), timestamp,
  partner_transaction_id)
) …
```

SELECT * FROM payment_request WHERE partner IN (1,2,3,4,66)

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner), timestamp,
  partner_transaction_id)
) …
```

SELECT * FROM payment_request WHERE partner IN (1,2,3,4,66)

NODE_1	NODE_2	NODE_3
NODE_4	NODE_5	NODE_6
NODE_7	NODE_8	NODE_9

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner), timestamp,
  partner_transaction_id)
) …
```

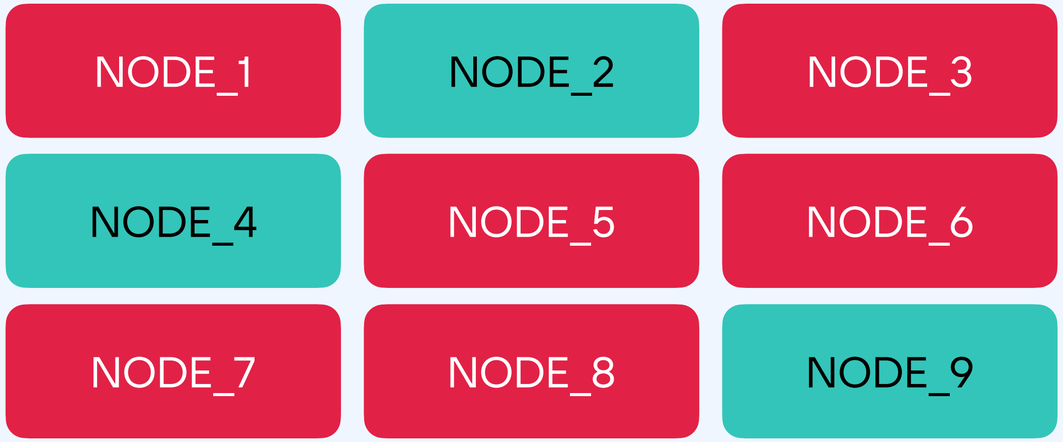
SELECT * FROM payment_request WHERE partner IN (1,2,3,4,66)

NODE_1	NODE_2	NODE_3
NODE_4	NODE_5	NODE_6
NODE_7	NODE_8	NODE_9

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner), timestamp,
  partner_transaction_id)
) …
```

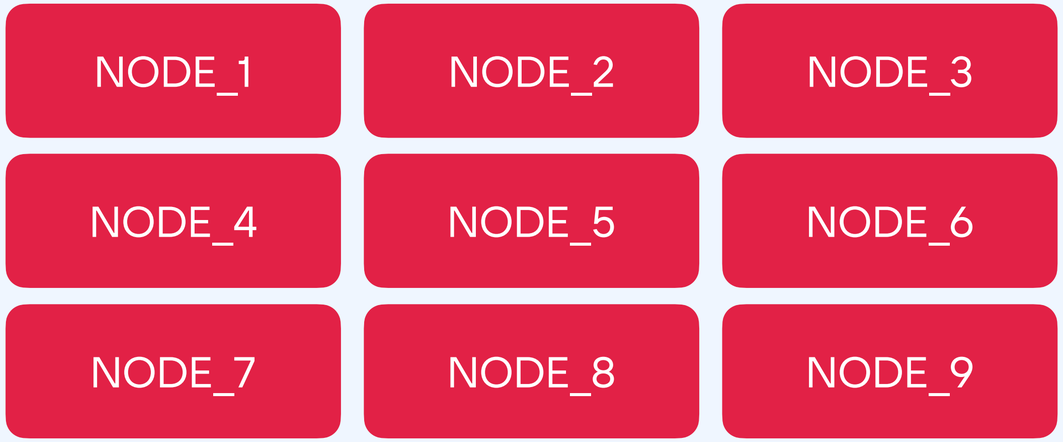
SELECT * FROM payment_request WHERE partner IN (1,2,3,4,66)



Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner), timestamp,
  partner_transaction_id)
) …
```

SELECT * FROM payment_request WHERE partner IN (1,2,3,4,66)



Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY(?)
) …
```

1,2,3,4,66: СМС операторы

5,49,764: банки

9000,9001,9002: электронные деньги

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner int,
  partner_transaction_id text,
  PRIMARY(?)
) …
```

1,2,3,4,66: СМС операторы (1)

5,49,764: банки (2)

9000,9001,9002: электронные деньги (3)

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  partner int,
  partner_transaction_id text,
  PRIMARY(?)
) …
```

1,2,3,4,66: СМС операторы (1)

5,49,764: банки (2)

9000,9001,9002: электронные деньги (3)

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner_type), timestamp,
  partner, partner_transaction_id)
) …
```

1,2,3,4,66: CMC операторы (1)

5,49,764: банки (2)

9000,9001,9002: электронные деньги (3)

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner_type), timestamp,
  partner, partner_transaction_id)
) …
```

1,2,3,4,66: CMC операторы (1)

5,49,764: банки (2)

9000,9001,9002: электронные деньги (3)

SELECT * FROM payment_request WHERE partner_type = ? AND timestamp > ?

Payment request: разные партнеры

```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner_type), timestamp,
  partner, partner_transaction_id)
) …
```

1,2,3,4,66: CMC операторы (1)

5,49,764: банки (2)

9000,9001,9002: электронные деньги (3)

Опять длинный ряд!!!


```
SELECT  * FROM payment_request WHERE partner_type = ? AND timestamp > ?
```

Payment request: разные партнеры

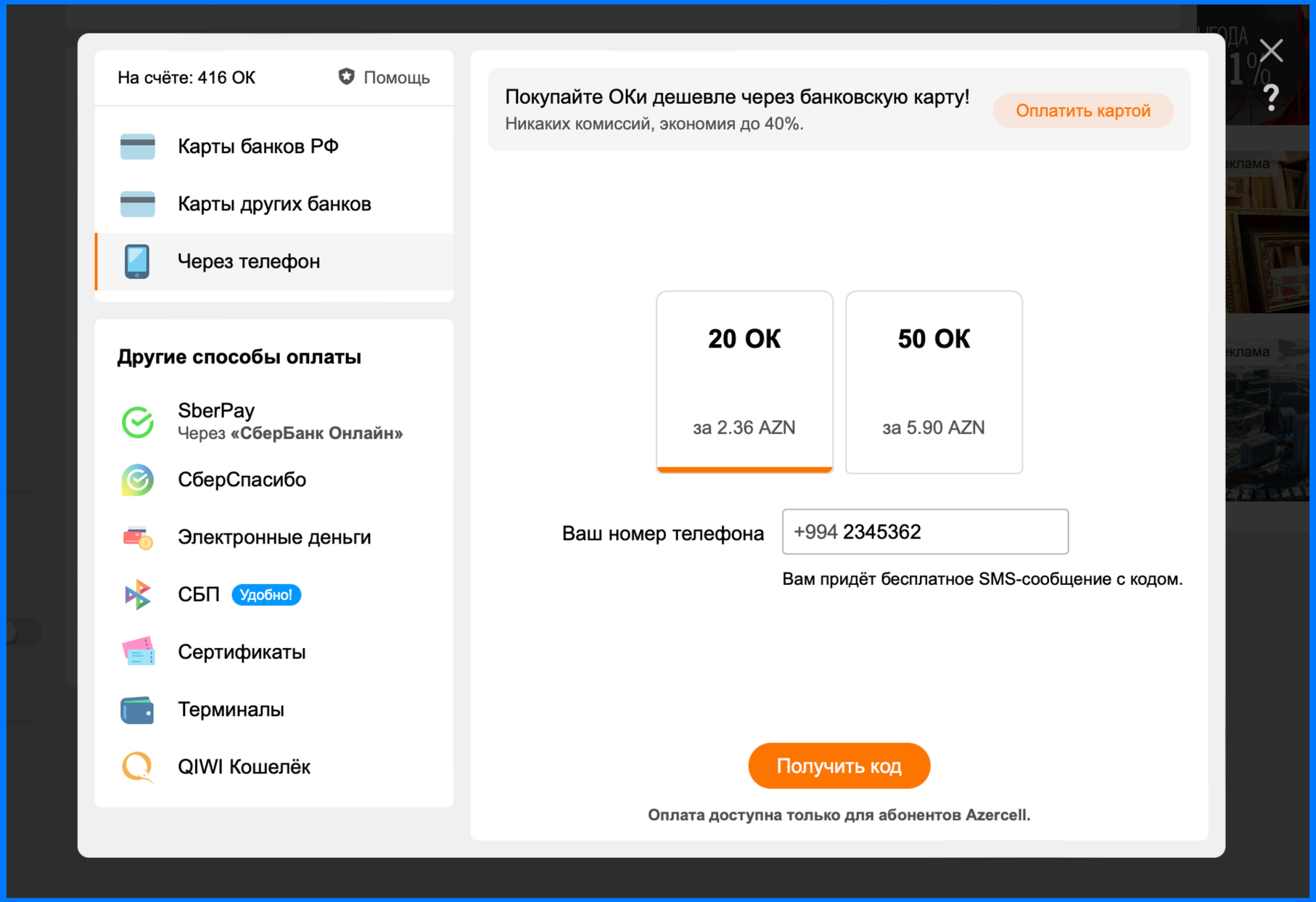
```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```


```
Create table payment_request_by_partner (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  partner int,
  partner_transaction_id text,
  PRIMARY((partner_type, month), timestamp,
  partner, partner_transaction_id)
) …
```


```
SELECT  * FROM payment_request WHERE partner IN (1,2,3,4,66)
```


```
SELECT  * FROM payment_request WHERE partner_type = ? AND month = ? AND timestamp > ?
```

Payment request: поиск по телефону



Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

SELECT * FROM payment_request WHERE phone like %653423

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  month data,
  partner int,
  partner_transaction_id text,
  PRIMARY KEY ((?), ?,
  partner, partner_transaction_id)
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

SELECT * FROM payment_request WHERE phone like %653423

Ограничения:

1. обязательно должен быть partition key в запросе
2. не ограниченные операции (>,>=,<, <=) можно указать в запросе один раз и на последнем ключе из тех, что есть в запросе
3. запрос возможен только по ключам

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY ((phone_postfix),?,partner,
  partner_transaction_id)
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

SELECT * FROM payment_request WHERE phone like %653423

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY ((phone_postfix),?,partner,
  partner_transaction_id)
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  partner, partner_transaction_id)
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  partner, partner_transaction_id)
```

SELECT * FROM payment_request WHERE phone like ? ORDER BY timestamp

SELECT * FROM payment_request WHERE phone_postfix = ?

Payment request: поиск по телефону

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  partner, partner_transaction_id)
```

+7 123 456 78 99

SELECT * FROM payment_request WHERE phone_postfix = ?

Payment request: поиск по телефону

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  partner, partner_transaction_id)
```

+7 123 456 78 99

У нас в базе в среднем у нас 2 разных телефона у которых последние 6 цифр одинаковые.

Т.е. для 567899 в базе: 71234567899, 71259567899

SELECT * FROM payment_request WHERE phone_postfix = ?

Payment request: поиск по телефону

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  partner, partner_transaction_id)
```

Фильтрация на клиенте

+7 123 456 78 99

У нас в базе в среднем у нас 2 разных телефона у которых последние 6 цифр одинаковые.

Т.е. для 567899 в базе: 71234567899, 71234567899

Т.е. при фильтрации на клиенте мы отбрасываем половину или меньше результатов


```
SELECT  * FROM payment_request WHERE phone_postfix = ?
```

Payment request: поиск по телефону

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  phone, partner, partner_transaction_id)
```

Payment request: поиск по телефону

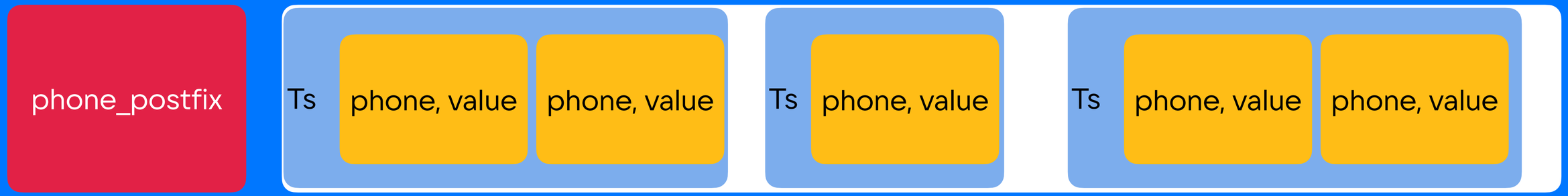
```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  phone, partner, partner_transaction_id)
```


```
SELECT  * FROM payment_request
  WHERE phone_postfix = ?
  AND phone > ?
  AND phone < ?
  ORDER BY timestamp
  allow filtering;
```

Payment request: поиск по телефону

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  phone, partner, partner_transaction_id)
```

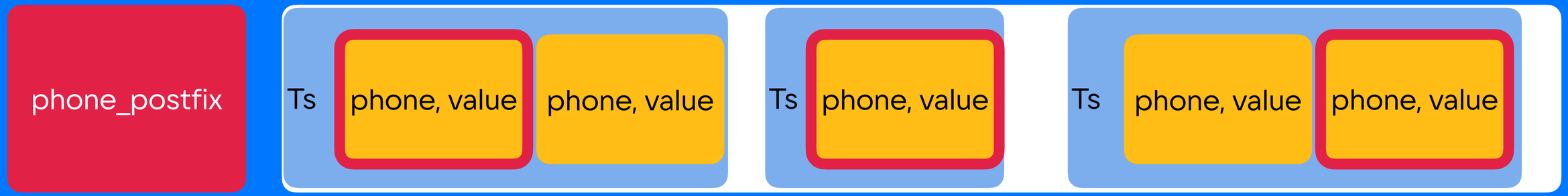

```
SELECT  * FROM payment_request
  WHERE phone_postfix = ?
  AND phone > ?
  AND phone < ?
  ORDER BY timestamp
  allow filtering;
```



Payment request: поиск по телефону

```
Create table payment_request_by_phone (
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ***
  partner int,
  partner_transaction_id text,
  PRIMARY KEY((phone_postfix), timestamp,
  phone, partner, partner_transaction_id)
```


```
SELECT  * FROM payment_request
  WHERE phone_postfix = ?
  AND phone > ?
  AND phone < ?
  ORDER BY timestamp
  allow filtering;
```



«Фильтрация на клиенте» VS «Allow filtering»

Плюсы фильтрации на клиенте:

- Явно видим сложность
- Не зависим от реализации Кассандры
- Сложность не зависит от настроек/версии/багов кассандры

Плюсы «allow filtering»

- Фильтруем на стороне Cassandra, меньше сетевых передач

Payment request: поиск по телефону

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Create table payment_request_by_phone (

user_id bigint,

timestamp bigint,

phone text,

phone_postfix text,

…

partner int,

partner_transaction_id text

PRIMARY KEY ((phone_postfix), timestamp, partner, partner_transaction_id)

SELECT * FROM payment_request WHERE reversePhone like ? ORDER BY timestamp

SELECT * FROM payment_request WHERE phone_postfix = ?

Payment request

```
Create table payment_request(
  user_id bigint,
  timestamp bigint,
  phone CHAR(15),
  partner int,
  partner_transaction_id NVARCHAR(64),
  PRIMARY KEY (partner,
  partner_transaction_id)
) …
```

Payment request

+ CDC

```
Create table
payment_request(
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  month data,
  partner int,
  partner_transaction_id
  text
  PRIMARY KEY ((partner,
  partner_transaction_id))
```

```
Create table
payment_request_by_partner(
  user_id bigint,
  timestamp bigint,
  phone text,
  partner_type int,
  month data,
  partner int,
  partner_transaction_id text
  PRIMARY KEY ((partner_type,
  month), timestamp, partner,
  partner_transaction_id)
```

```
Create table
payment_request_by_phone(
  user_id bigint,
  timestamp bigint,
  phone text,
  phone_postfix text,
  ...
  partner int,
  partner_transaction_id text
  PRIMARY KEY ((phone_postfix),
  timestamp, partner,
  partner_transaction_id)
```

Что мы сделали:

- Решили в конкретном случае как обойти ограничение, что всегда необходим partition key

Что мы сделали:

- Решили в конкретном случае как обойти ограничение, что всегда необходим partition key

```
Create table payment_request_by_phone (
    user_id bigint,
    timestamp bigint,
    phone_postfix text,
    phone text,
    ...
    partner int,
    partner_transaction_id text
    PRIMARY KEY ((phone_postfix),
    timestamp, partner,
    partner_transaction_id)
```

Что мы сделали:

- Решили в конкретном случае как обойти ограничение, что всегда необходим partition key
- Обошли для конкретного кейса когда 2 поля не ограничены

Что мы сделали:

- Решили в конкретном случае как обойти ограничение, что всегда необходим partition key
- Обошли для конкретного кейса когда 2 поля не ограничены

```
Create table payment_request_by_phone (

    timestamp bigint,

    phone_postfix text,

    phone text,

    ...

    PRIMARY KEY ((phone_postfix),
    timestamp, partner,
    partner_transaction_id)
```

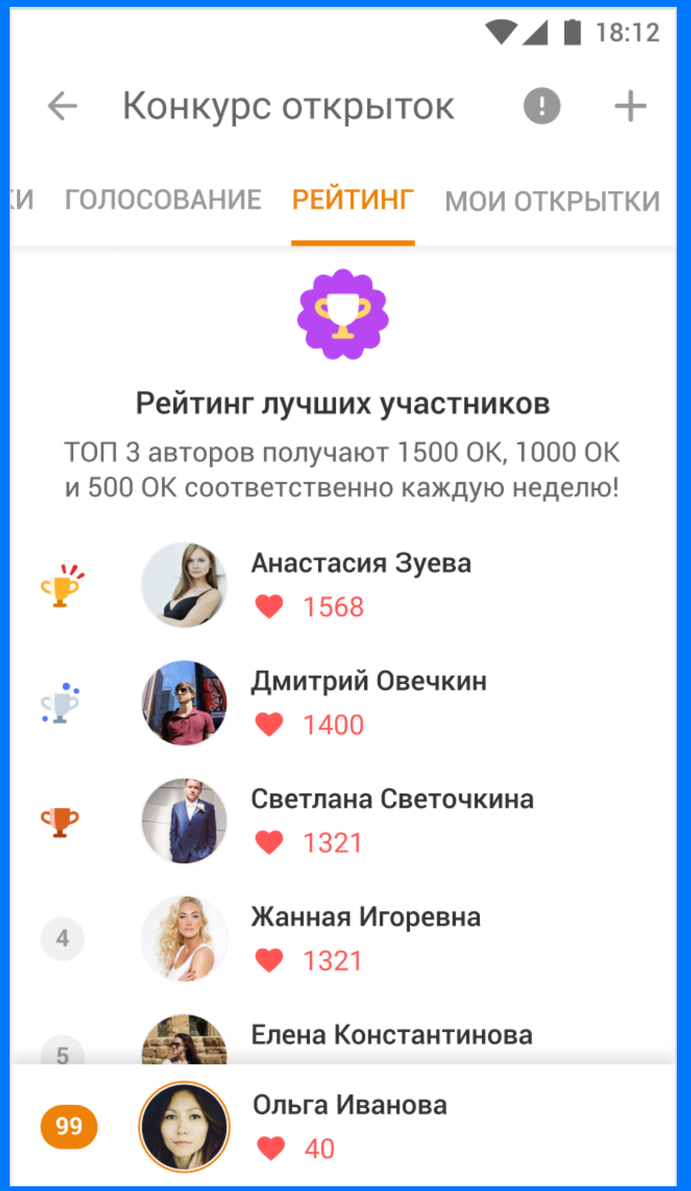
Что мы сделали:

- Решили в конкретном случае как обойти ограничение, что всегда необходим partition key
- Обошли для конкретного кейса когда 2 поля не ограничены
- Посмотрели как можно выгружать в OLAP DB

Удаление в Cassandra

Конкурс: у кого больше лайков, тот user и выиграл

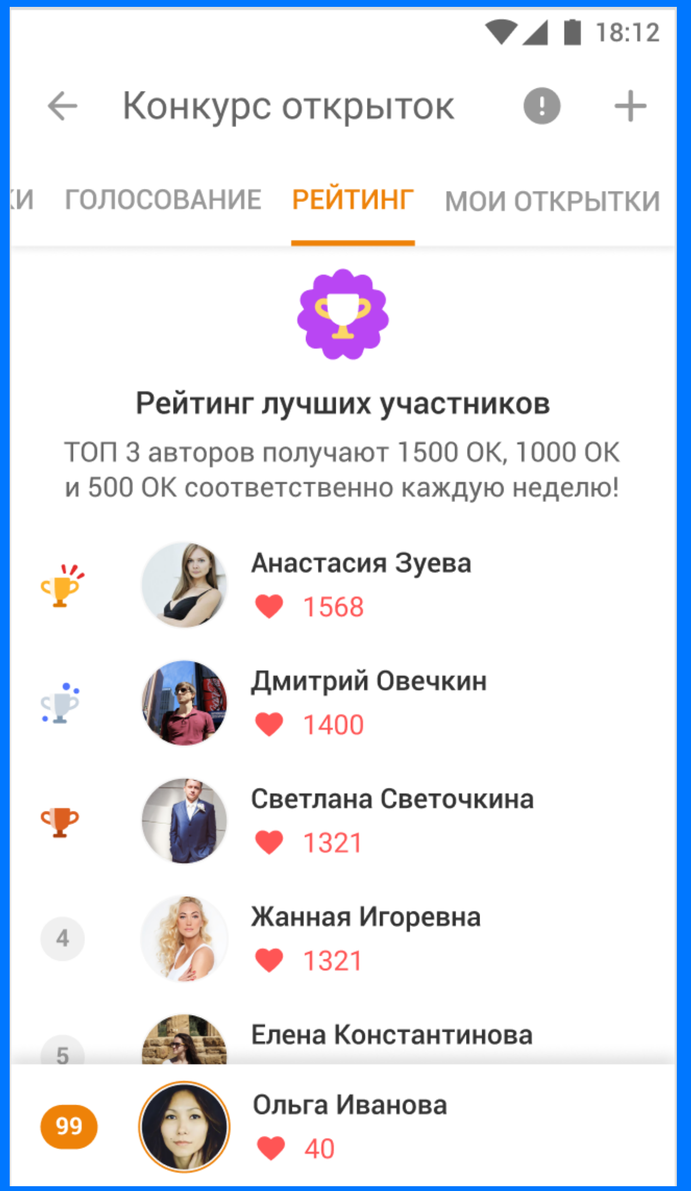
```
Create table likes(
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    user_id)
) …
```



Конкурс: у кого больше лайков, тот user и выиграл

```
Create table likes(
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    user_id)
) …
```

```
Create table
likes_sort(
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```



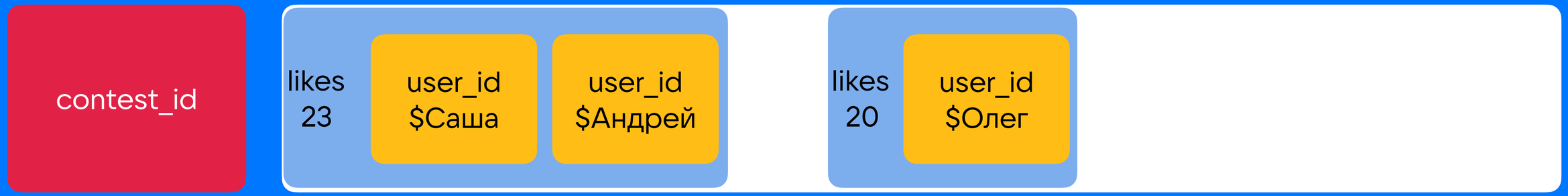
Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Андрей	23
Саша	23
Олег	20

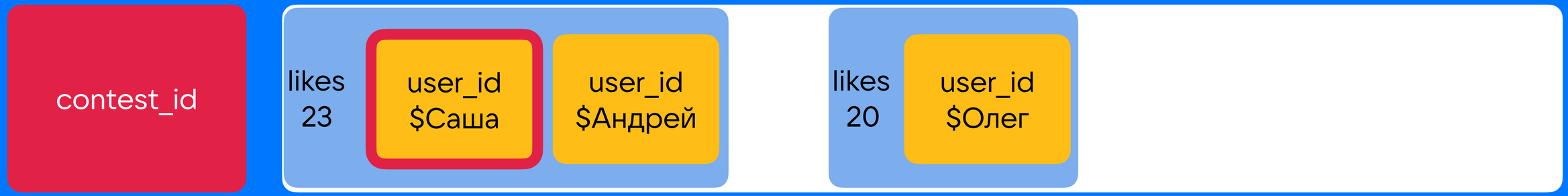


Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Андрей	23
Саша	23
Олег	20

SELECT * FROM likes_sort WHERE user_id = ? likes = ? contest_id = ?

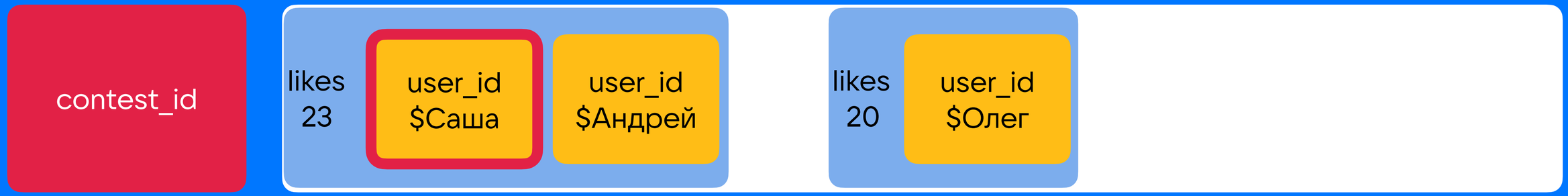


Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Андрей	23
Саша	23 + 1
Олег	20

SELECT * FROM likes_sort WHERE user_id = ? likes = ? contest_id = ?



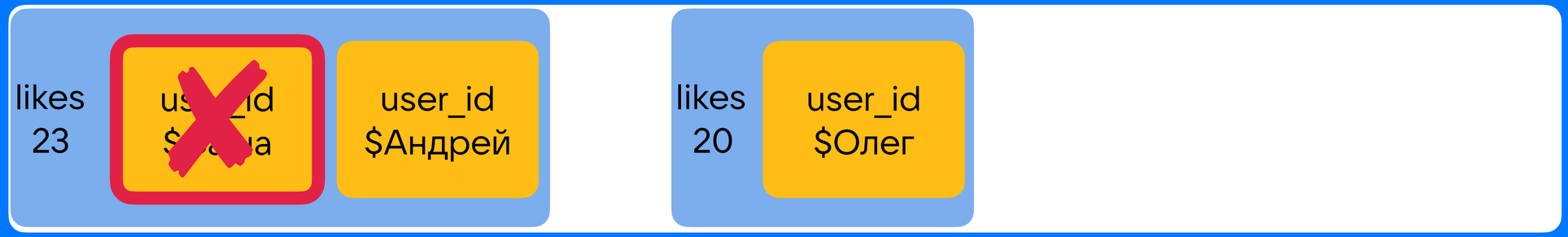
Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Андрей	23
Саша	23 + 1
Олег	20

DELETE FROM likes_sort WHERE user_id = ? AND likes = ? AND contest_id = ?

contest_id

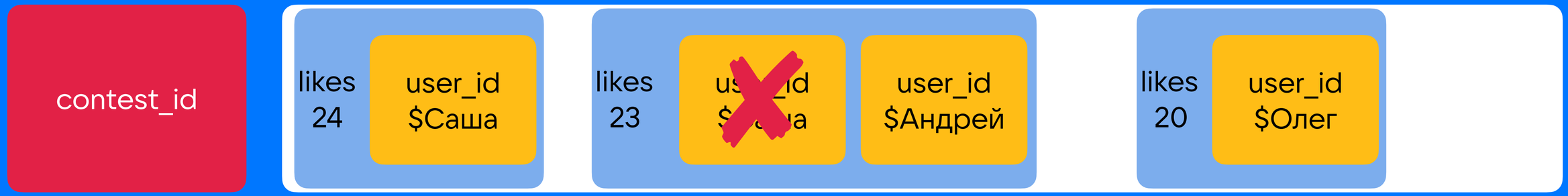


Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Саша	24
Андрей	23
Олег	20

INSERT INTO likes_sort (user_id, likes, contest_id) VALUES (?,?,?)



Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Саша	24
Андрей	23
Олег	20

SELECT * FROM likes_sort WHERE contest_id = ?

Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

gc_grace_seconds (default value is 864000 seconds (ten days))

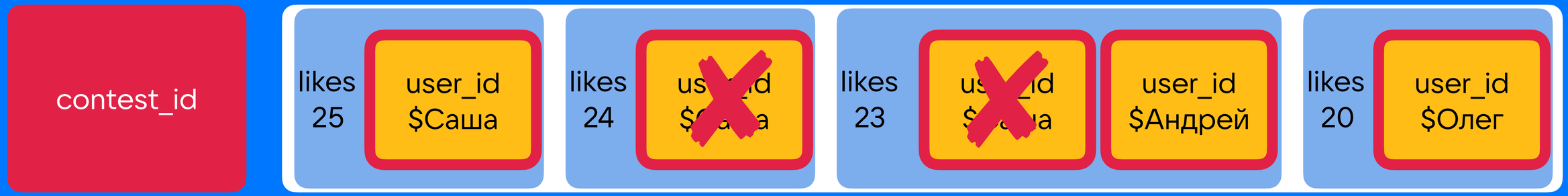
SELECT * FROM likes_sort WHERE contest_id = ?

Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Саша	25
Андрей	23
Олег	20

SELECT * FROM likes_sort WHERE contest_id = ?



Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Саша	10 024
Андрей	23
Олег	20

SELECT * FROM likes_sort WHERE contest_id = ?

Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

user	Likes
Саша	10 024
Андрей	23
Олег	20

SELECT * FROM likes_sort WHERE contest_id = ?

Конкурс: у кого больше лайков, тот user и выиграл

```
Create table
likes_sort {
  contest_id int,
  user_id bigint,
  likes bigint,
    PRIMARY KEY
    ((contest_id),
    likes, user_id)
) …
```

Какие проблемы мы увидим?

- Нагрузка на CPU
- Timeout
- Tombstone failure threshold = 100.000

Конкурс: у кого больше лайков, тот user и выиграл

Условия задачи:

Конкурс: у кого больше лайков, тот user и выиграл

Условия задачи:

- Достаточно иметь топ 1000, а остальным писать место 1000+

Конкурс: у кого больше лайков, тот user и выиграл

Условия задачи:

- Достаточно иметь топ 1000, а остальным писать место 1000+
- Можно обновлять не моментально, а с задержкой в 1 минуту.

Конкурс: у кого больше лайков, тот user и выиграл

`long[] userId`	$Оля	$Олег	$Андрей	$Саша
`long[] likes`	10000	1000	1000	100

Конкурс: у кого больше лайков, тот user и выиграл

`long[] userId`	$Оля	$Олег	$Андрей	$Саша
`long[] likes`	10000	1000	1000	100

Размер структуры:

1000 * (sizeof(long)+sizeof(long)) + sizeof(int) = 16kb

1000 * (8+8) + 4 = 16kb

Конкурс: у кого больше лайков, тот user и выиграл

`long[] userId`	$Оля	$Олег	$Андрей	$Саша
`long[] likes`	10000	1000	1000	100

Размер структуры:

1000 * (sizeof(long)+sizeof(long)) + sizeof(int) = 16kb

1000 * (8+8) + 4 = 16kb

Принятое решение:

Создавать эту структуру, где-то хранить и перестраивать раз в 1 мин

Конкурс: у кого больше лайков, тот user и выиграл

Update

```
Create table likes(
 contest_id int,
 user_id bigint,
 likes bigint,
 PRIMARY KEY ((contest_id),user_id))
```

Конкурс: у кого больше лайков, тот user и выиграл

Update

```
Create table likes(
 contest_id int,
 user_id bigint,
 likes bigint,
 PRIMARY KEY ((contest_id),user_id))
```

```
Select * from likes where contest_id = ? AND user_id > ?
```

Scheduler

```
long[] contentId
long[] likes
```

Конкурс: у кого больше лайков, тот user и выиграл

Update

```
Create table likes(
 contest_id int,
 user_id bigint,
 likes bigint,
 PRIMARY KEY ((contest_id),user_id))
```

```
Create table likes_top(
  contest_id int,
  value blob
  PRIMARY KEY (contest_id))
```

```
Select * from likes where contest_id = ? AND user_id > ?
```

Scheduler

```
long[] contentId
long[] likes
```

Конкурс: у кого больше лайков, тот user и выиграл

Update

```
Create table likes(
 contest_id int,
 user_id bigint,
 likes bigint,
 PRIMARY KEY ((contest_id),user_id))
```

```
Create table likes_top(
  contest_id int,
  value blob
  PRIMARY KEY (contest_id))
```

```
Select * from likes where contest_id = ? AND user_id > ?
```

Scheduler

long[] contentId

long[] likes

Backend

long[] contentId

long[] likes

Get 20

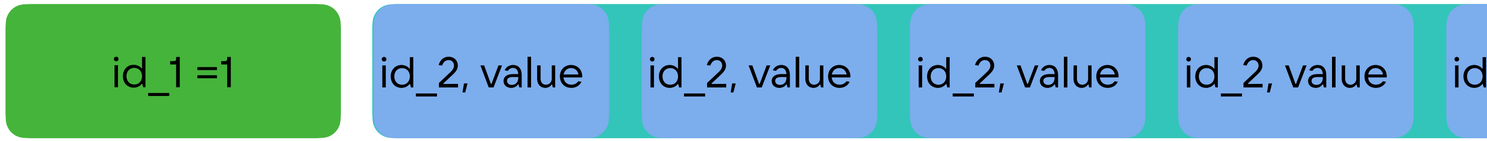
Return with limit 20

Удаления в Cassandra:

- Избегайте удалений
- Если удаляете много,
  то проверяйте, что нет запроса,
  который читает могилки

Удаления в Cassandra:

- Избегайте удалений
- Если удаляете много,
  то проверяйте, что нет запроса,
  который читает могилки

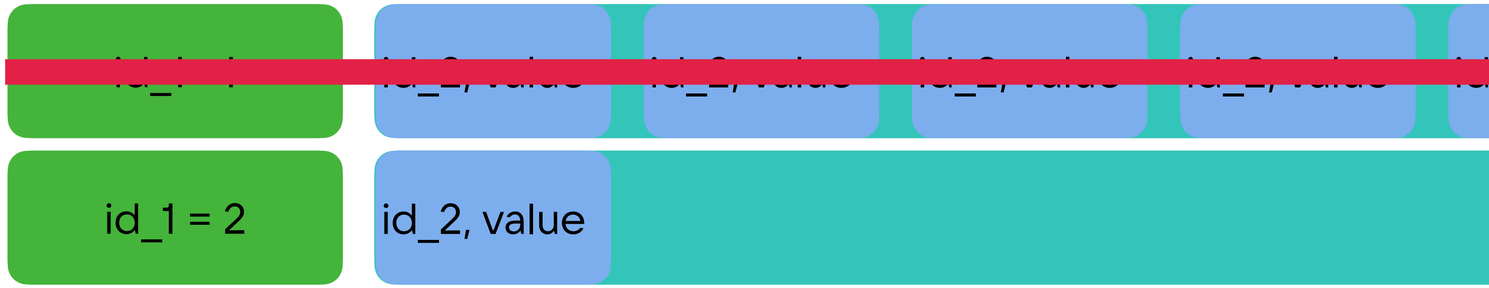


Удаления в Cassandra:

- Избегайте удалений
- Если удаляете много,
  то проверяйте, что нет запроса,
  который читает могилки

Удаления в Cassandra:

- Избегайте удалений
- Если удаляете много,
  то проверяйте, что нет запроса,
  который читает могилки



Выводы

Cassandra позволит

- Повысить доступность
  - 99.9% → 99.9999%
- Ускорить запросы
  - 10мс → 2мс (p99)
- Легко масштабировать
  - 10нод → 300нод на ДЦ

Cassandra требует

- Вдумчивого проектирования
- Денормализации данных
- Аккуратного удаления

АЛЕКСАНДР ПАЩЕНКО

ОК

Единые
Новые
Облачные
Технологии

Дзен