Where and Order By Clauses in Cassandra CQL

Asked 6 years, 1 month ago Modified 3 years, 2 months ago Viewed 39k times

13

5

I am new to NoSQL database and have just started using apache Cassandra. I created a simple table "emp" with primary key on "empno" column. This is a simple table as we always get in Oracle's default scott schema.

Now I loaded data using the `COPY` command and issued query `Select * from emp order by empno` but I was surprised that CQL did not allow Order by on `empno` column (which is PK). Also when I used `Where` condition, it did not allow any inequality operations on empno column (it said only EQ or IN conditions are allowed). It also did not allowed Where and Order by on any other column, as they were not used in PK, and did not have an index.

Can someone please help me what should I do if I want to keep `empno` unique in the table and want a query results in Sorted order of `empno`?

(My version is:

```
cqlsh:demodb> show version
[cqlsh 5.0.1 | Cassandra 2.2.0 | CQL spec 3.3.0 | Native protocol v4]
```
)

cassandra cql cql3

Share Improve this question Follow

edited Dec 28, 2016 at 9:28
Aaron
50.6k ●11 ●111 ●123

asked Feb 29, 2016 at 19:38
Amir S Siddiqui
181 ●1 ●2 ●4

2 Answers

Sorted by: Highest score (default)

There are two parts to a PRIMARY KEY in Cassandra:

- partition key(s)

30

- clustering key(s)

`PRIMARY KEY (partitionKey1,clusteringKey1,clusteringKey2)`

or

`PRIMARY KEY ((partitionKey1,partitionKey2),clusteringKey1,clusteringKey2)`

The partition key determines which node(s) your data is stored on. The clustering key determines the order of the data within your partition key.

In CQL, the `ORDER BY` clause is really only used to *reverse* the defined sort direction of your clustering order. As for the columns themselves, you can only specify the columns defined (and in that exact order...no skipping) in your `CLUSTERING ORDER BY` clause at table creation time. So you cannot pick arbitrary columns to order your result set at query-time.

Cassandra achieves performance by using the clustering keys to sort your data on-disk, thereby only returning ordered rows in a single read (no random reads). This is why you must take a query-based modeling approach (often duplicating your data into multiple query tables) with Cassandra. Know your queries ahead of time, and build your tables to serve them.

```
Select * from emp order by empno;
```

First of all, you need a `WHERE` clause. It's ok to query without it, *if you're working with a relational database*. With Cassandra, you should do your best to avoid unbound `SELECT` queries. Besides, Cassandra can only enforce a sort order *within a partition*, so querying without a `WHERE` clause won't return data in the order you want, anyway.

Secondly, as I mentioned above, you need to define clustering keys. If you want to order your result set by `empno`, then you must find another column to define as your partition key. Try something like this:

```
CREATE TABLE emp_by_dept (
  empno text,
  dept text,
  name text,
  PRIMARY KEY (dept,empno)
) WITH CLUSTERING ORDER BY (empno ASC);
```

Now, I can query employees by department, and they will be returned to me ordered by `empno`:

```
SELECT * FROM emp_by_dept WHERE dept='IT';
```

But to be clear, you will **not** be able to query every row in your table, and have it ordered by a single column. The only way to get meaningful order into your result sets, is first partition your data in a way that makes sense to your business case. Running an unbound `SELECT` will return all of your rows (assuming that the query doesn't time-out while trying to query every node in your cluster), but result set ordering can only be enforced within a partition. So you have to restrict by partition key in order for that to make any sense.

My apologies for self-promoting, but last year I wrote an article for DataStax called [We Shall Have Order!], in which I addressed how to solve these types of problems. Give it a read and see if it helps.

Edit for additional questions:

> From your answer I concluded 2 things about Cassandra:
>
> (1) There is no way of getting a result set which is only order by a column that has been defined as Unique.
>
> (2) When we define a PK (partition-key+clustering-key), then the results will always be order by Clustering columns within any fixed partition key (we must restrict to one partition-key value), that means there is no need of ORDER BY clause, since it cannot ever change the order of rows (the order in which rows are actually stored), i.e. Order By is useless.

1) All PRIMARY KEYs in Cassandra are unique. There's no way to order your result set by your partition key. In my example, I order by `empno` (after partitioning by dept). – Aaron 1 hour ago

2) Stopping short of saying that ORDER BY is useless, I'll say that its only real use is to switch your sort direction between ASC and DESC.

> I created an index on "empno" column of "emp" table, it is still not allowing ORDER BY empno. So, what Indexes are for? are they only for searching records for specific value of index key?

You cannot order a result set by an indexed column. Secondary indexes are (not the same as their relational counterparts) really only useful for edge-case, analytics-based queries. They don't scale, so the general recommendation is not to use secondary indexes.

> Ok, that simply means that one table cannot be used for getting different result sets with different conditions and different sorting order.

Correct.

> Hence for each new requirement, we need to create a new table. IT means if we have a billion rows in a table (say Sales table), and we need sum of sales (1) Product-wise, (2) Region-wise, then we will duplicate all those billion rows in 2 tables with one in clustering order of Product, the other in clustering order of Region,. and even if we need to sum sales per Salesman_id, then we build a 3rd table, again putting all those billion rows? is it sensible?

It's really up to you to decide how sensible it is. But lack of query flexibility is a drawback of Cassandra. To get around it you can keep creating query tables (I.E., trading disk for performance). But if it gets to a point where it becomes ungainly or difficult to manage, then it's time to think about whether or not Cassandra is really the right solution.

EDIT 20160321

> Hi Aaron, you said above "Stopping short of saying that ORDER BY is useless, I'll say that its only real use is to switch your sort direction between ASC and DESC."
>
> But i found even that is not correct. Cassandra only allows ORDER by in the same direction as we define in the "CLUSTERING ORDER BY" caluse of CREATE TABLE. If in that clause we define ASC, it allows only order by ASC, and vice versa.

Without seeing an error message, it's hard to know what to tell you on that one. Although I have heard of queries with `ORDER BY` failing when you have too many rows stored in a partition.

`ORDER BY` also functions a little odd if you specify multiple columns to sort by. If I have two clustering columns defined, I can use `ORDER BY` on the first column indiscriminately. But as soon as I add the second column to the `ORDER BY` clause, my query only works if I specify *both* sort directions the same (as the `CLUSTERING ORDER BY` definition) or *both* different. If I mix and match, I get this:

```
InvalidRequest: code=2200 [Invalid query] message="Unsupported order by relation"
```

I think that has to do with how the data is stored on-disk. Otherwise Cassandra would have more work to do in preparing result sets. Whereas if it requires everything to either to match or mirror the direction(s) specified in the `CLUSTERING ORDER BY`, it can just relay a sequential read from disk. So it's probably best to only use a single column in your `ORDER BY` clause, for more predictable results.

Share Improve this answer Follow

edited Mar 21, 2016 at 21:26

answered Feb 29, 2016 at 20:02

Aaron
50.6k ● 11 ● 111 ● 123

1 Thanks for your detailed answer! I appreciate. From your answer I concluded 2 things about Cassandra: (1) There is no way of getting a result set which is only order by a column that has been defined as Unique, and (2) When we define a PK (partition-key+clustering-key), then the results will always be order by Clustering columns within any fixed partition key (we must restrict to one partition-key value), that means there is no need of ORDER BY clause, since it cannot ever change the order of rows (the order in which rows are actually stored), i.e. Order By is useless. – Amir S Siddiqui Feb 29, 2016 at 20:59

Ok, thanks again. One more thing, I created an index on "empno" column of "emp" table, it is still not allowing ORDER BY empno. So, what Indexes are for? are they only for searching records for specific value of index key? – Amir S Siddiqui Feb 29, 2016 at 21:58

Ok, that simply means that one table cannot be used for getting different result sets with different conditions and different sorting order. Hence for each new requirement, we need to create a new table. IT means if we have a billion rows in a table (say Sales table), and we need sum of sales (1) Product-wise, (2) Region-wise, then we will duplicate all those billion rows in 2 tables with one in clustering order of Product, the other in clustering order of Region,. and even if we need to sum sales per Salesman_id, then we build a 3rd table, again putting all those billion rows? is it sensible? – Amir S Siddiqui Feb 29, 2016 at 22:10

@AmirSSiddiqui edit made. Comments moved to answer text. – Aaron Feb 29, 2016 at 22:31

1 Hi Aaron, you said above "Stopping short of saying that ORDER BY is useless, I'll say that its only real use is to switch your sort direction between ASC and DESC." But i found even that is not correct. Cassandra only allows ORDER by in the same direction as we define in the "CLUSTERING ORDER BY" caluse of CREATE TABLE. If in that clause we define ASC, it allows only order by ASC, and vice versa. – Amir S Siddiqui Mar 21, 2016 at 20:58

4

Adding a redux answer as the accepted one is quite long.

Order by is currently only supported on the clustered columns of the PRIMARY KEY and when the partition key is restricted by an Equality or an IN operator in where clause.

That is if you have your primary key defined like this :

```
PRIMARY KEY ((a,b),c,d)
```

Then you will be able to use the ORDER BY when & only when your query has :

> a where clause with all the primary key restricted either by an equality operator (=) or an IN operator such as :

```
SELECT * FROM emp WHERE a = 1 AND b = 'India' ORDER BY c,d;

SELECT * FROM emp WHERE a = 1 AND b = 'India' ORDER BY c;
```

These two query are the only valid ones.

Also this query would not work :

```
SELECT * FROM emp WHERE a = 1 AND b = 'India' ORDER BY d,c;
```

because order by currently only support the ordering of columns following their declared order in the PRIMARY KEY that is in primary key definition c has been declared before d and the query violates the ordering by placing d first.

answered Jan 12, 2019 at 16:09
Sankar
353 ● 5 ● 16