Difference between partition key, composite key and clustering key in Cassandra?

OrangeDogOrangeDog 26.6k99 gold badges8989 silver badges159159 bronze badges

There is a lot of confusion around this, I will try to make it as simple as possible.

The primary key is a general concept to indicate one or more columns used to retrieve data from a Table.

The primary key may be ***SIMPLE*** and even declared inline:

```
create table stackoverflow_simple (
     key text PRIMARY KEY,
     data text
 );
```

That means that it is made by a single column.

But the primary key can also be ***COMPOSITE*** (aka ***COMPOUND***), generated from more columns.

```
create table stackoverflow_composite (
     key_part_one text,
     key_part_two int,
     data text,
     PRIMARY KEY(key_part_one, key_part_two)
 );
```

In a situation of ***COMPOSITE*** primary key, the "first part" of the key is called ***PARTITION KEY*** (in this example **key_part_one** is the partition key) and the second part of the key is the ***CLUSTERING KEY*** (in this example **key_part_two**)

Please note that the both partition and clustering key can be made by more columns, here's how:

```
create table stackoverflow_multiple (
     k_part_one text,
     k_part_two int,
     k_clust_one text,
     k_clust_two int,
     k_clust_three uuid,
     data text,
     PRIMARY KEY((k_part_one, k_part_two), k_clust_one, k_clust_two, k_clust_three)
 );
```

Behind these names ...

- The **Partition Key** is responsible for data distribution across your nodes.
- The **Clustering Key** is responsible for data sorting within the partition.
- The **Primary Key** is equivalent to the **Partition Key** in a single-field-key table (i.e. **Simple**).
- The **Composite/Compound Key** is just any multiple-column key

Further usage information: DATASTAX DOCUMENTATION

Small usage and content examples
SIMPLE KEY:

```
insert into stackoverflow_simple (key, data) VALUES ('han', 'solo');
select * from stackoverflow_simple where key='han';
```

table content

```
key | data
----+------
han | solo
```

COMPOSITE/COMPOUND KEY can retrieve "wide rows" (i.e. you can query by just the partition key, even if you have clustering keys defined)

```
insert into stackoverflow_composite (key_part_one, key_part_two, data) VALUES ('ronaldo', 9, 'football player');
insert into stackoverflow_composite (key_part_one, key_part_two, data) VALUES ('ronaldo', 10, 'ex-football player');
select * from stackoverflow_composite where key_part_one = 'ronaldo';
```

table content

```
 key_part_one | key_part_two | data
--------------+--------------+--------------------
      ronaldo |            9 |    football player
      ronaldo |           10 | ex-football player
```

But you can query with all key (both partition and clustering) ...

```
select * from stackoverflow_composite
   where key_part_one = 'ronaldo' and key_part_two  = 10;
```

query output

```
 key_part_one | key_part_two | data
--------------+--------------+--------------------
      ronaldo |           10 | ex-football player
```

Important note: the partition key is the minimum-specifier needed to perform a query using a `where clause`. If you have a composite partition key, like the following

eg: `PRIMARY KEY((col1, col2), col10, col4))`

You can perform query only by passing at least both col1 and col2, these are the 2 columns that define the partition key. The "general" rule to make query is you have to pass at least all partition key columns, then you can add optionally each clustering key in the order they're set.

so the valid queries are (**excluding secondary indexes**)

- col1 and col2
- col1 and col2 and col10
- col1 and col2 and col10 and col 4

Invalid:

- col1 and col2 and col4
- anything that does not contain both col1 and col2

Hope this helps.