Apache Cassandra Composite Key₩Partition key₩Clustering key 介绍

在前面的文章《Apache Cassandra 快速入门指南（Quick Start）》我们简单介绍了 Cassandra 的一些基本知识。在那篇文章里面我们使用了下面语句创建了一张名为 iteblog_user 的表：

```
cqlsh> use iteblog_keyspace;
cqlsh:iteblog_keyspace> CREATE TABLE iteblog_user (first_name text , last_name text, PRIMARY
 KEY (first_name)) ;
```

建表语句里面有个 PRIMARY KEY 关键字，我们在初次使用 Apache Cassandra 的时候可能见过诸如 Composite Key、Partition key 以及 Clustering key，这么多种 key 它和上面的 PRIMARY KEY 有什么关系呢？看看本文你就明白了。

Single column Primary Key

在 Cassandra 里面，Primary Key 可以由一列或多列组成，用于从表中检索数据，如果 Primary Key 由一列组成，那么称为 Single column Primary Key，如下语句

```
cqlsh> use iteblog_keyspace;
cqlsh:iteblog_keyspace> CREATE TABLE iteblog_user (first_name text , last_name text, PRIMARY
 KEY (first_name)) ;
```

这种情况 first_name 就是 Single column Primary Key，我们在检索数据的时候需要指定 Primary Key：

```
cqlsh:iteblog_keyspace> select * from iteblog_user;

 first_name | last_name
------------+-----------
        Wu |      Shi
     Zhang |      San
        Li |       Si
```

```
(3 rows)
cqlsh:iteblog_keyspace> select * from iteblog_user where first_name = 'Wu';

 first_name | last_name
------------+-----------
         Wu |        Shi

(1 rows)
cqlsh:iteblog_keyspace> select * from iteblog_user where last_name = 'Si';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this q
uery as it might involve data filtering and thus may have unpredictable performance. If you wa
nt to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

可以看出，如果查询只指定 last_name，是无法运行上面的查询的。

Composite Primary Key

如果 Primary Key 由多列组成，那么这种情况称为 Compound Primary Key 或 Composite Primary Key，如下：

```
cqlsh:iteblog_keyspace> CREATE TABLE iteblog_user_composite (first_name text , last_name tex
t, PRIMARY KEY (first_name, last_name)) ;
```

其中 first_name 称为 Partition key，last_name 称为 Clustering key（也可以称为 Clustering column）。在这种情况下，下面查询的前三条都是合法的，最后一条是非法的。

```
cqlsh:iteblog_keyspace> select * from iteblog_user_composite;

cqlsh:iteblog_keyspace> select * from iteblog_user_composite where first_name = 'iteblog';

cqlsh:iteblog_keyspace> select * from iteblog_user_composite where first_name = 'iteblog' and
 last_name = 'hadoop';

//非法查询
cqlsh:iteblog_keyspace> select * from iteblog_user_composite where last_name = 'hadoop';
```

Partition key 和 Clustering key 也可以由多个字段组成，如果 Partition key 由多个字段组成，称之为 Composite partition key：

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

上面 k_part_one 和 k_part_two 共同组成 Partition key，k_clust_one 和 k_clust_two 共同组成 Clustering key。这种情况下有效的查询包括

```
select * from iteblog_multiple;
select * from iteblog_multiple where k_part_one = 'iteblog' and k_part_two = 'hadoop';
select * from iteblog_multiple where k_part_one = 'iteblog' and k_part_two = 'hadoop' and k_cl
ust_one = 'hbase';
select * from iteblog_multiple where k_part_one = 'iteblog' and k_part_two = 'hadoop' and k_cl
ust_one = 'hbase' and k_clust_two = 'spark';
```

> 小知识 - 使用 Composite partition key 的一个原因
> 其实一个 Partition 对应的 Cell 个数在 Cassandra 里面是有限制的。理论上来说，一个 Partition 的 Cell 个数大约在20亿个（2^{31}）。所以采用了 Composite partition key，我们可以将数据分散到不同的 Partition，这样有利于将同一个 Partition 的 Cell 个数减少。

Partition key & Clustering key & Primary Key 作用

在 Cassandra 里面这么多种 key 都有什么作用呢？总结起来主要如下：

- Partition Key：将数据分散到集群的 node 上
- Primary Key：在 Single column Primary Key 情况下作用和 Partition Key 一样；在 Composite Primary Key 情况下，组合 Partition key 字段决定数据的分发的节点；
- Clustering Key：决定同一个分区内相同 Partition Key 数据的排序，默认为升序，我们可以在建表语句里面手动设置排序的方式（DESC 或 ASC）

示例

这里主要介绍下 Composite Primary Key 中 Clustering Key 是如何决定同一个分区内相同 Partition Key 数据的排序。比如我们的建表语句如下：

```
cqlsh:iteblog_keyspace> CREATE TABLE iteblog_user_composite (
                   ... name text,
                   ... id timeuuid,
                   ... content text,
                   ... PRIMARY KEY (name, id));
cqlsh:iteblog_keyspace>
```

我们往表里面插入以下的数据

```
INSERT INTO iteblog_user_composite(name, id, content) VALUES ('iteblog', NOW(), 'I am www.it
eblog.com');
INSERT INTO iteblog_user_composite(name, id, content) VALUES ('iteblog_hadoop', NOW(), 'I a
m iteblog_hadoop');
INSERT INTO iteblog_user_composite(name, id, content) VALUES ('iteblog', NOW(), 'Hello www.i
teblog.com');
INSERT INTO iteblog_user_composite(name, id, content) VALUES ('iteblog_hadoop', NOW(), 'Hel
lo iteblog_hadoop');
INSERT INTO iteblog_user_composite(name, id, content) VALUES ('iteblog', NOW(), 'Welcome to
 www.iteblog.com');
INSERT INTO iteblog_user_composite(name, id, content) VALUES ('iteblog_hadoop', NOW(), 'We
lcome to iteblog_hadoop');
```

我们使用下面的查询语句将所有数据查询出来：

```
cqlsh:iteblog_keyspace> select name, content, id, UNIXTIMESTAMPOF(id) as time from iteblog_
user_composite;

 name          | content                  | id                                   | time
---------------+--------------------------+--------------------------------------+---------------
 iteblog_hadoop |       I am iteblog_hadoop | 402eace0-5ad7-11e9-a4d4-f5d538b6a60b | 1554
821615278
 iteblog_hadoop |      Hello iteblog_hadoop | 40303380-5ad7-11e9-a4d4-f5d538b6a60b | 1554
821615288
 iteblog_hadoop |  Welcome to iteblog_hadoop | 40a00c50-5ad7-11e9-a4d4-f5d538b6a60b | 1
```

554821616021
iteblog | I am www.iteblog.com | 402de990-5ad7-11e9-a4d4-f5d538b6a60b | 1554821615273
iteblog | Hello www.iteblog.com | 402f4920-5ad7-11e9-a4d4-f5d538b6a60b | 1554821615282
iteblog | Welcome to www.iteblog.com | 4030cfc0-5ad7-11e9-a4d4-f5d538b6a60b | 1554821615292

(6 rows)

从上面的查询可以看出，Partition key 为 iteblog_hadoop 和 iteblog 的数据都是按照 id 进行升序排序的，数据直接是在底层存储就排好序的，查询的时候并不需要做额外的配置，这种排列方式有利于那种只查询最近更新的几条数据的需求。