# The technical details

1. B+-Tree

The B+-tree index is a variant tree for the B-tree. Yet every non-leaf node of the B+-tree only contains navigation information without any actual value. As all leaf nodes are connected by a linked list, it is convenient for traversal process and especially suitable for database storage systems requiring range queries. Fig.1 shows the secondary index structure of B+-tree and oval dotted line part contains the range query results with secondary and primary keys using name *Justin* to *Sean*.

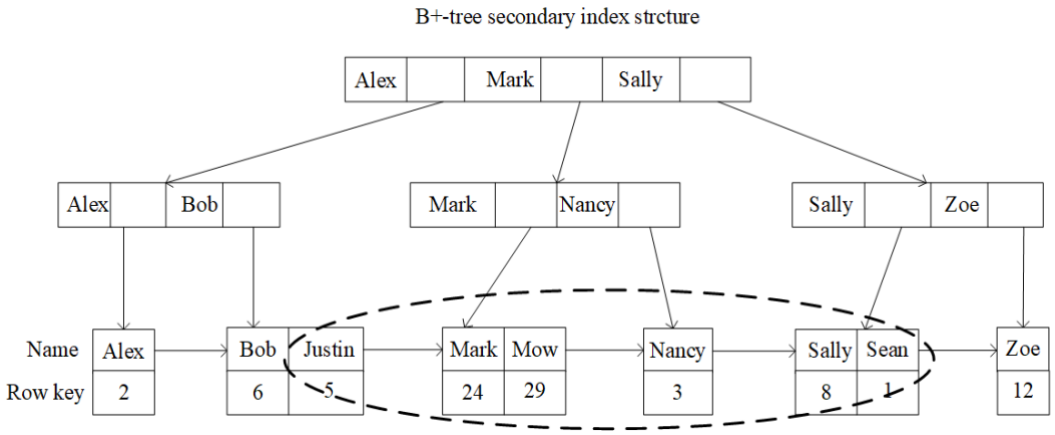


Fig.1 B+-tree secondary index structure to key name

2. Hash Table

A hash table is a data structure that can map keys to values, and it uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. We can construct a simple hash function for primary key value to build a hash table (HTable) as shown in Eq.(1),

$$\mathrm{Hash(key)} = \mathrm{key}\ \%\ \mathrm{m} \tag{1}$$

where key can be chosen from primary key set and m is usually parameterized as the distinct primary key set size.

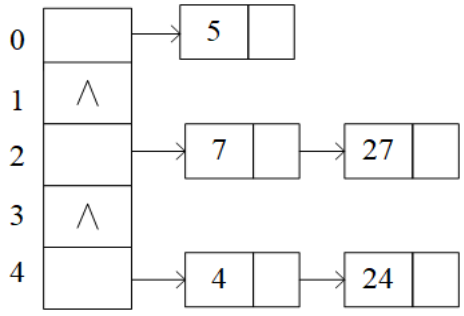


Fig.2 Chaining method to handle hash collisions

In a hash table, now that there will be some primary keys mapped to the same hash value, a chaining is adopted to handle hash collisions as shown in fig.2. So we place the primary key values with the same hash value in the same chaining or linked list. There may exist m linked lists with m hash values, and array T[0...m-1] can be used to store the head pointer of linked lists. Each head is initialized as a null pointer. If the hash value is matched, the new pointer with the primary key will be inserted into the linked list connected to that hash value.

3. Hybrid Index Process

It is hard for the basic structures mentioned above to implement complex queries such as multi-dimensional range query. So we present a novel hybrid index mechanism consisting of secondary index creation step, the primary key collection procedure and the result matching process, which are shown in Fig. 3.

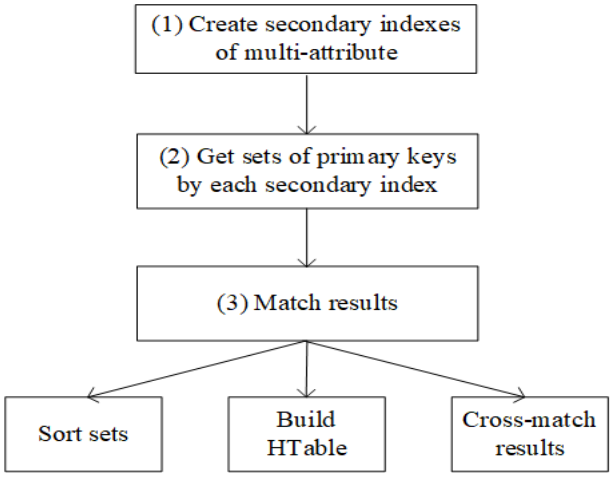


Fig. 3 Query process for multi-dimensional range query

We build different secondary key index according to each key for multi-key-value storage system, and choose B+-tree as the index structure of corresponding queried key. These secondary key query results through secondary index in step(1) are returned. Appropriate primary keys should be collected and grouped into sets separately on the basis of independent secondary key results. The step(3) consists of three parts including key set sorting module, hash table building module, and result cross-matching module. Firstly, we sort the primary key sets by set size that would be obtained in step(2). Then the minimum set is generated. We just employ the hash function in Eq.(1) for the minimum set in the first time. One hash table is produced. Next, the remained primary key sets are mapped to the hash table using the same hash function in sequence based on

preceding results, and during the process the cross-match data result set will become getting smaller. Finally we can obtain the primary keys that satisfy all conditions of the multi-dimensional range query.