**Epipog**

**Specification**

**Index Class Family**

**Dec. 23, 2016**

# 1. Index Interface

The index object is the interface definition for specifying an index. An index is used to index records in a data store. An index consists of mapped values (e.g., hash) of one or more columns in a record and a location in the data store corresponding to the record. Conceptually an index is:

mapped-values location-in-data-store  
 mapped-values location-in-data-store  
 …

An index may be unique (no duplicate entries) or non-unique (duplicate entries permitted).

## Default Methods

The following are default methods defined in the interface. If not overwritten in an implementation, the default method is used in the implementation of the interface.

public default long[] Hash( Data value );

This method generates a pair of hash codes for a data value. If the data value is numeric, both hash codes are the numeric value. In the case of floating point numbers, the value is rounded. For characters, the value is the numeric code for the character, and for boolean it is 1 and 0, respectively. For strings, the first hash value is the java hashCode() function and the second hash value is generated by an internal hash function that generates a different value.

While the hash code generated by hashCode() results in low collisions, the generation of a second independent hash code is used to significantly reduce the likelihood of a collision.

## 1.3 Abstract Methods

The following abstract methods are declared in the interface.

public void Unique( Boolean unique );

This method sets whether the index is unique (no duplicate entries).

public long Add( long hash, long pos, long data );

This method adds an entry into the index. The argument hash is the hash key and the argument data is the hash comparator for validating against collisions. The argument pos is the offset into the data store where the entry (or record) is located.

If unique is set, then the method will check if there is an existing entry. If so, then the entry is replaced with the new entry and the method returns the location of the previous entry in the data store; otherwise -1 is returned to indicate that no entry was replaced.

public ArrayList<Long> Find( long hash, long data );

This method locates an entry in the index. If the arguments hash and data match one or more entries in the index the corresponding pos value (offset in data store) of each matched entry is returned. Otherwise, if no entry is found an empty list is returned.

public ArrayList<Long> Remove( long hash, long data );

This method removes entries matching the hash code and data value. If one or more entries are removed, the offset in the data store of the corresponding entries is returned; otherwise, an empty list is returned.

public long Pos( int nth );

This method returns the offset in the data store for the record at the nth location in the index. If there is no such record, a -1 is returned.

# 2. IndexLinkedList Implementation

## The IndexLinkedList class is an implementation of the Index interface. The implementation is found in the file IndexLinkedList.java. This class defines the methods for implementing a serial linked list index, which is searched sequentially.

## 2.1 Fields

The following fields are defined in the implementation.

private ArrayList<long[]> index = new ArrayList<long[]>(); // in-memory storage

This field holds the index in memory. It is an array list of triplets, where each element is a long data type. The first element is the hash code, the second the offset in the data store of the corresponding entry, and the third is the data validation value.

private boolean unique = false; // required to be unique

This field indicates whether the index entries must be unique or not. If unique, then duplicates are not allowed.

## 2.2 Methods

The following methods are implemented in the implementation.

public void Unique( Boolean unique );

This method sets whether the index is unique (no duplicate entries).

public long Add( long hash, long pos, long data );

This method adds an entry into the index. The argument hash is the hash key and the argument data is the hash comparator for validating against collisions. The argument pos is the offset into the data store where the entry (or record) is located.

If unique is set, then the method will check if there is an existing entry. If so, then the entry is replaced with the new entry, the hash code is changed to 0xFFFFFFFFFFFFFFFF to indicate the entry is dirty, and the method returns the location of the previous entry in the data store; otherwise -1 is returned to indicate no entry was replaced.

Each entry is added sequentially and is not sorted.

public ArrayList<Long> Find( long hash, long data );

This method locates an entry in the index. If the arguments hash and data match one or more entries in the index the corresponding pos value (offset in data store) of each matched entry is returned. Otherwise, if no entry is found an empty list is returned.

The index is searched sequentially.

public ArrayList<Long> Remove( long hash, long data );

This method removes entries matching the hash code and data value. If one or more entries are removed, the offset in the data store of the corresponding entries is returned; otherwise, an empty list is returned.

public long Pos( int nth );

This method returns the offset in the data store for the record at the nth location in the index. If there is no such record, a -1 is returned.