Chapter 11
Online Index Construction and Maintenance

In the preceding chapters, we have assumed that tuples in a relation are accessed via a sparse B-tree index on the primary key of the relation. In this chapter we extend our database and transaction model with read, delete, and update actions based on ranges of non-primary-key attributes. To accelerate these actions, secondary indexes must be constructed. A secondary index, as considered here, is a dense B-tree whose leaf pages contain index records that point to the tuples stored in leaf pages of the sparse primary B-tree index of the relation.

A relation can have many secondary indexes on different attributes or combinations of attributes, thus making it possible for the query optimizer to find efficient execution plans for many kinds of queries. However, this comes with the price of an overhead on updates: when a transaction inserts or deletes a tuple, all the indexes must also be updated within the same transaction.

In database performance tuning, it is common that new secondary indexes are constructed when it is found that some regularly performed SQL statements need acceleration, while secondary indexes that are found to be seldom used in query-execution plans are dropped. Index construction for a large relation is a heavy and time-consuming operation; in many cases we cannot afford the relation to be inaccessible for transactions during index construction.

In this chapter we present an online index-construction algorithm that allows transactions to update the relation while a secondary index is being constructed. When the index construction is completed, the new index can also be used in query-execution plans. We also briefly address the question of online maintenance of the physical clustering of B-tree indexes.
11.1 Secondary-Key-Based Actions

We now assume that our logical database consists of a relation

\[ r(X, Y, V), \]

where \( X \) is the primary key (with unique values as before) and \( Y \) is sequence of attributes (not necessarily with unique values) used as a secondary key on which key-range-read and bulk-delete and bulk-update actions can also be based. Note that insertions cannot, of course, be performed, but using the primary key can be.

The primary-key-based actions now take the following forms:

1. Single-tuple read actions \( R[x, \theta z, y, v] \) for given primary key \( x \)
2. Single-tuple insert actions \( I[x, y, v] \)
3. Single-tuple delete actions \( D[x, y, v] \) for given primary key \( x \)
4. Single-tuple write actions \( W[x, y, v, y', v'] \) for given \( x, y' \) and \( v' \)
5. Bulk-read actions \( R_X[s_X, s_{XYV}] \) for given set \( s_X \) of primary-key ranges
6. Bulk-insert actions \( I[s_{XYV}] \)
7. Bulk-delete actions \( D_X[s_X, s_{XYV}] \) for given set \( s_X \) of primary-key ranges
8. Bulk-update actions \( W[s_X, f, s_{XYV'}v'] \) for given set \( s_X \) of primary-key ranges and function \( f(Y, V) \)

Besides these actions we now allow transactions to contain the following secondary-key-based actions:

(a) Single-tuple read actions \( R[x, y, \theta z, v] \) for given secondary key \( z \)
(b) Bulk-read actions \( R_Y[s_Y, s_{XYV}] \) for given set \( s_Y \) of secondary-key ranges
(c) Bulk-delete actions \( D_Y[s_Y, s_{XYV}] \) for given set \( s_Y \) of secondary-key ranges
(d) Bulk-update (or bulk-write) actions \( W_Y[s_Y, f, s_{XYV'}v'] \) for given set \( s_Y \) of secondary-key ranges and function \( f(V) \)

Here action (a) is only defined for unique secondary keys, and it retrieves the tuple \((x, y, v)\) with the least key \( y \theta z\). Actions (b)–(d) also apply to non-unique keys. Action (b) returns the set \( s_{XYV} \) of tuples \((x, y, v)\) with keys \( y \) in one of the ranges in \( s_Y \), action (c) deletes the tuples \((x, y, v)\) with keys \( y \) in one of the ranges in \( s_Y \) and returns the deleted tuples in set \( s_{XYV} \), and action (d) replaces each tuple \((x, y, v)\) with key \( y \) in one of the ranges in \( s_Y \) by the tuple \((x, y, f(v))\) and returns the set \( s_{XYV'} \) of tuples \((x, y, v, f(v))\).

Expanded histories are used to model the interleaved execution of transactions containing the above secondary-key-based actions besides the primary-key-based ones. This is done along the lines explained in Sect. 10.1. The single-tuple actions generated from actions (c) and (d) are logged in the obvious way with redo-undo log records and their undo actions with redo-only log records, as usual.

The definitions of the isolation anomalies that can appear in an expanded history must be extended so as to cover the secondary-key-based actions. For example, dirty writes can also occur between two secondary-key-based delete or write actions or