RTMP/S copy

REPLACED SHEETS SG

- [x] Description : paged 3 to paged 5
- [x] Claim(s) : paged 14 to paged 17
- [ ] Drawing(s) : paged — to paged —
- [x] Abstract : paged 18 to paged —

CONCURRENT RECORD ACCESS DATABASE SYSTEM

DESCRIPTION

This invention relates to data processing and more specifically to database management of records.

Current data processing systems include database management programs. These programs provide easy access to database tables that each consist of a multiple of records. A relational database program provides access to several different database tables where elements of one database table are contained in another database table. The relational database program allows the user to search, access, and alter data contained in several different database tables using a specific element or field that is common to these database tables. For example, one database table may contain employees' names and employees' serial numbers. Another database table may contain employees' names and employees' salaries. A third database table may contain employees' names and employees' locations. A relational database program may allow the user to access employees' names, employees' serial numbers, employees' salaries, and employees' locations and make changes to these tables without having to individually access each database table separately.

An important aspect of the database programs is the capability to provide fast and efficient access to records in the individual database.

More recent data processing systems provide support to a multiple of users simultaneously enabling each user to access data concurrently.

An index file is commonly used by database management programs to provide quick and efficient access to records in tables. These index files are commonly configured in a B-Tree structure. A reference that discusses the B-Tree is "Efficient Locking For Concurrent Operation On B-Tree" by Lehman and Yao, ACM Transactions on Database Systems, volume 6, number 4, December, 1981 pages 650-670. Other references addressing

B-Tree structures include "The Ubiquitous B-Tree" by Comer, Computing Surveys, volume 11, number 2, June, 1979, pages 121-137; and "Concurrent Operation on B-Trees with Over Taking" by Sagiv, Proceedings ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March, 1985, pages 28-37.

The index file configured as a B-Tree structure consists of a root node with many levels of nodes branching from the root node. The information contained in these nodes include pointers to the nodes at the next level or pointers to records in the database. These pointers include further information termed key record information which may reference the records in the database. The record keys are in an ordered form throughout the nodes. For example, an index tree may exist for an alphabetic listing of employee names. The root node would include reference keyed data that relates to records indirectly or directly referenced by the next level of nodes. The reference keys contain information about the index field, i.e. the alphabetic spelling of the employees name. Therefore, the ordered keys in the root node would point to the next successive level of nodes. In other words, the next successive node may indirectly or directly reference all employees names beginning with A, B, and C.

A next successive node, parallel with the first successive node, may contain employee records whose last name begin with the letters D-M. The last successive node on this level would reference records of employees with last names starting with N-Z. As one searches through the index file tree, a bottom, or leaf, node is eventually reached. The contents of the bottom node include record keys that point to the individual records in storage.

One problem in providing concurrent accesses to database tables occurs when multiple transactions are trying to access a record at the same time. Specifically, when one user wishes to change a record and another user is attempting to access this record, a contention situation occurs. One solution to the contention problem is to provide exclusive

access (or locking) to the records or to the portions of the B-Tree indexes to insure that the index node, or record is not changed while the user is attempting to access it. Locking is addressed in "Index Locking and Splitting" IBM Technical Disclosure Bulletin, volume 25, number 7B, December, 1982, pages 3725-3729; and Locking Protocols for Concurrent Operations on B-Trees", IBM Technical Disclosure Bulletin, volume 19, number 10, March, 1977, pages 3887-3889.

The disadvantage to a locking solution is that a lock, while providing access to one user, prevents access by any other user.

Information in index nodes may be altered during access by several users. These nodes contain the key record information that may be deleted or inserted as records are inserted into or deleted from the database table. Information from one node may be moved to another node during a record insert operation because of limited storage space for a node. The inserting transaction may attempt to move key record information from one node to another in order to accomplish the insertion task. This movement of information during concurrent transaction access may generate error conditions within the concurrent transaction accesses because the index information is being changed.

It is an object of the present invention is to provide concurrent access to the index by several transactions while the index information is being changed without introducing error conditions.

According to the invention, there is provided a method of operating a computer system for effecting simultaneous transactions on a database by multiple users using an index tree of record keys, said tree having a root node and lower level nodes, each node, except those in a lowest (leaf) node referencing one or more nodes in the next lower level, with those in the leaf node referencing records in the database, said method comprising, for a transaction involving insertion or deletion of a current record and its corresponding record key, the steps of:

This Sheet Has Been Replaced



traversing through said nodes from said root node to a leaf node using said record keys in response to an input request;

limiting to other users all but read accesses to a node being traversed and an immediately previously traversed node;

identifying a next successively located record key relative to the record key to be deleted or inserted in the leaf node and limiting all access to the leaf node;

limiting all accesses to a record associated with said next successively located record key;

inserting or deleting said current record and its corresponding record key; and

removing said access limitation to said record associated with the next successively located record key and all access limitations to the traversed nodes.

There is further provided a digital data processing system for effecting simultaneous transactions on a database by multiple users using an index tree of record keys, said tree having a root node and lower level nodes, each node, except those in a lowest (leaf) node referencing one or more nodes in the next lower level, with those in the leaf node referencing records in the database, said system comprising means for traversing through said nodes from said root node to a leaf node using said record keys in response to an input request, means for limiting to other users all but read accesses to a node being traversed and an immediately previously traversed node, means for identifying a next successively located record key relative to the record key to be deleted or inserted the leaf node and limiting all accesses to a record associated with said next successively located record key, means for inserting or deleting said current record and its corresponding record key, and for removing said access limitation to said record associated with the next

This Sheet Has Been Replaced



successively located record key and all access limitations to the traversed nodes.

An embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Figure 1 is a illustration of a database table;

Figure 2 is a illustration of the database table stored in a computer memory;

Figure 3 is a block diagram illustrating a simple B-Tree index;

Figure 4 is an illustration of the storage of index B-Tree node data in a computer memory;

Figure 5 is a flow chart illustrating an initial search through an index B-Tree;

Figure 6 is a flow chart illustrating a fetch through the index B-Tree;

Figure 7 is a flow chart illustrating an insert through the index B-Tree;

Figure 8 is a flow chart illustrating a delete through the index B-Tree;

Figure 9 is a flow chart illustrating a node splitting algorithm; and

Figure 10 is a flow chart illustrating a node collapsing algorithm.

This invention relates to the accessing of records in a database system. The following example is an overly simplified database access problem provided to enable the reader to more easily understand the invention contained herein. The reader should understand, that, in reality, the number of entries and amount of information that is contained in database tables are much greater than illustrated.

Figure 1 illustrates table 10 containing employee names and numbers. Each employee record includes a single employee number. The names in table 10 are listed in alphabetical order. However, it should be apparent to those skilled in the art that this is not how the records are normally stored in the computer memory.

This Sheet Has Been Replaced



Figure 2 is an illustration of the storage of the employee name/number key records in a computer memory 12. Each record 14, 16, 18, 20, 24, and 26 consists of several portions. In record 14, the record first consist of an address portion 28, the employee name 30, and the employee number 32. The address 28 is the actual location of the employee name 30 and the employee number 32 in the computer memory. Table 12 of Figure 2 illustrates that the storage of the names is not in alphabetical order and is not even contiguous.

The database system normally provides an index tree that enables a user to quickly access a specific record in storage. An example of such an index is provided in Figure 3. For simplicity of presentation, the index tree is presented in two levels. Index 40 consist of a root node connected to three nodes 50, 52, and 54. The root node 42 is termed "index identifier" and details the type of information in the index. In this example, the index is an alphabetically ordered (ascending) index of the employees' names. Each employee's name includes a key. In this example, the key is the first letter of the employee's name. The keys are used to identify a next level node to be accessed in searching for a specific employee name record. In the root node 42, the keys consist of letters "D" 44,"F" 46, "NULL" 48. The D key 44 directs the user from the root node 42 (the parent node) to node 50 (a child node). Node 50 contains key record information for direct access of the employees Andrews, Baker, and Chester. However, only a single key is used in the root node. This key relates to the highest level of information (or the highest key record) that might now exist or have existed in the node 50. Likewise the "F" 46 directs a user to node 52 that contains the Edgar and Edwards employee names. Lastly, "NULL" 48 directs the user to node 54. The "NULL" key indicates that node 54 is the last successive leaf node. It should be apparent to those skilled in the art that the index 40 may also contains multiple levels of nodes. The two level index 40 is provided for the sake of simplicity.

The contents of node 50, 52 and 54 as it would appear in a computer memory is illustrated in Figure 4 as columns 56, 58 and 60 respectively.

Column 56 represents how the information contained in node 50 would be stored in a computer memory. Column 56 consist of a set of pointers 62 and data storage area 64. Pointer 66 points to the information containing the employee name Andrew and employee number. Likewise, pointer 68 points to the employee name Baker and related information. Thus, when a transaction accesses a node, the transaction first examines the pointer portion of the node information to determine where the key record information is stored. This pointer information provides direct access to the storage containing the desired information. In multilevel indexing structure including intermediate levels, the node storage would contain pointers that point to addresses for the next successively located nodes according the key record information. The pointers would be ordered according to their respective key information. However, a successive node, such as 50 that contains several employee names is only referenced by a single key from a parent node. The reference key is the highest key record information allowable in the node. In index trees including intermediate nodes, the key in a preceding (or parent) node is a key for the highest record that is referenced by any of the keys.

Figure 5 illustrates a flow chart for accessing record information through the index. It should be understood that in a database management system, the operations illustrated by Figures 5-8 is called during an access transaction. In accordance with the illustrated example, the transaction would be attempting to fetch a record to determine the employee number or to perform some other operation such as record insert or record delete operation.

Figures 3 and 4 also illustrate pointers 51 and 53 that point from one leaf node to a next successively located leaf node. These pointers 72 and 74 direct an accessing transaction to the next successively located leaf node. In this example, the NULL entry 76 in node 60 indicates that node 60 is the last node.

In Figure 5, the access logic starts at step 100 and proceeds to step 102 where the index root node is S-Latched and accessed. An S-Latch

provides limited access to other concurrent users. This limited access provides the other users with the capability to only read the node. No other access, such as the capability to delete or change, is provided. The index identifier node identifies the type of index and provides the initial direction for accessing a record in this example. The index identifier node would identify the index as an alphabetic index in ascending order for the employee names. In step 103, the child node to be accessed would be identified according to the information in the parent node. In step 104, it is determined whether not the operation to be performed is a fetch operation. If the operation is not a fetch operation, in other words the operation is a record insert (or key record insert) or record delete (or key record delete) operation, the program proceeds to step 106 to determine if the next node beneath the parent node is a leaf node or bottom node. If the next node is a leaf node, the program proceeds to the step 110 and acquires an X-Latch on the child node. The X-Latch is an exclusive latch that excludes all other accesses to this node. In other words, the transaction that applies the X-Latch excludes all other transactions from accessing this node.

Returning to step 104, if the operation is a fetch operation or, returning to step 106, if the child is not a leaf node, the logic proceeds to step 108 to acquires an S-Latch on the child node. The logic proceeds to step 112, where it accesses the child node and determines if the key of the record being searched is greater than the key in the child node. If this key is greater, the logic proceeds to step 114 to determine if the tree index structure is latched. If the tree is not latched, the logic proceeds to step 118 wherein the parent and child nodes are unlatched and the tree is latched. The logic then proceeds from step 118 back to step 102 to reinitiate the operation upon the granting of the tree latch. Optimizations are possible to reduce the number of nodes to be accessed when the operation is re-attempted.

In this example, an X-Latch on a tree is provided to indicate to all other accesses that a change in tree structure is being made. If a tree X-Latch access is in progress when a latch is attempted on the tree, the

attempting access must wait till the earlier access is complete. An S-Latch on a tree is provided to all other accesses to indicate that no structure changes are being made but other accesses may concurrently access the index tree. No other changes can occur until the S-Latch is released. Tree traversal may occur regardless of the existence of S-Latches or X-Latches. These tree traversals may include key record deletions or insertions.

In step 114 if the tree is latched, or in step 112 if the key is not greater than the highest key in the child, the logic proceeds to step 116 to determine if the child is a leaf. If the child is not, the logic proceeds to step 115 to unlatch the parent node and then returns to step 103. However, if the child is a leaf, the logic proceeds to step 120 to unlatch the parent and then to step 122, 124 and 126 to determine if the operation is a fetch, insert or delete operation. In this example provided, if neither of these three operations are attempted, the logic will return to the user in step 130. In practice this return would include an error indicator signifying that the operation to be performed is not identifiable by this accessing attempts

If the operation is a fetch operation, the logic proceeds to step 200 illustrated on Figure 6. In step 200, the logic finds the requested key being searched or at least the next highest key. In step 202, the logic then requests a conditional lock on the key record. In this example a conditional lock is requested from managing logic that manages the locks on the record keys. The term "conditional" means that if the lock is not immediately granted, a response will be provided to the requesting accessor indicating that such a lock is not being granted. This response is used in Step 204. If the lock has not been granted, the logic proceeds to step 206 to unlatch the child node and then to step 208 to request an unconditional lock on the key record. In step 208, the accessor then waits until such lock is granted before it proceeds. Once the lock is granted, it proceeds though connector 213 to step 102 in Figure 5 to restart the search. Returning to step 204, if the lock has been granted, the logic proceeds to step 209 to return the key record data. In step 210,

the child node is unlatched. In step 211, the key record is unlocked. It should be apparent to those skilled in the art that step 211 will occur when the transaction is complete or at an earlier time.

In Figure 7, the insert operation is illustrated. It should be apparent to those skilled in the art that the key record to be inserted will be X-Locked, if necessary, before the insert operation begins. This operation would insert a record in the computer memory and provide an inserted key into the index updating the respective node(s) of the index enabling other transactions to access the newly inserted record. In step 300, it is first determined if the key can fit into the bottom or leaf node. If so, the logic proceeds to step 304 to find the next key that is greater than the key to be inserted. If the next key is not in the node, the access is pointed to the next successively located leaf node. If there is no next successively located leaf node, the access finds the null indicator. Then, in step 308, the logic request a conditional X-Lock on the next key record. In step 312, it is then determined whether or not this lock has been granted. If so, in step 318 the new key is inserted on the first bottom node that was reached in step 300. In step 324, the next record and key record are unlocked. Also, any held latches are released in step 329. Then, in step 325, the logic returns. It should be apparent to those skilled in the art that the lock on the inserted key record will be released when the transaction is complete. Returning to step 312, if the lock is not granted, the logic proceeds to step 316 to unlatch the child and then to step 322 to request an unconditional X-Lock on the next key record. The logic then returns through connector 328 to step 102 (Figure 5).

Returning to step 300, if the key will not fit into the leaf node, the logic proceeds to step 302 to request a conditional tree X-Latch. In step 306, it is determined whether or not the conditional latch is granted. If not the logic proceeds to step 310 to unlatch the child node and then to step 314 to request an unconditional tree X-Latch. Upon being granted the unconditional tree X-Latch or, in step 306 if the latch is granted, the logic proceeds to step 320 to perform a node splitting

algorithm. This node splitting algorithm provides a new node containing a portion of the information from the previous node. The node splitting algorithm may also provide additional intermediate nodes wherein the additional nodes would be updated. The parent node is modified to contain the highest key record and a pointer to the new node. A node splitting algorithm is illustrated in Figure 9. Furthermore, node splitting is addressed in Data Structures and Algorithms, by Aho, Hopcroft and Ullman, Addison-Wesley Publishing Company, 1983, pages 170-179. Upon completion of the node splitting algorithm, the logic proceeds to step 326 to unlatch the tree. The logic would then return to step 102 (Figure 5) through connector 327.

Figure 8 illustrates the delete operation. It should be apparent to those skilled in the art that the key record to be deleted will be X-Locked, if necessary, before the delete operation begins. In step 400, the logic finds the next key greater than the key to be deleted. In step 402, the logic requests a conditional X-Lock on this next key record. The logic proceeds to step 404 to determine if the lock is granted. If the lock is not granted, the logic proceeds to unlatch the child node in step 406 and then to request an unconditional lock on the next key record in step 410. The logic then proceeds through connector 431 to step 102 (Fig. 5).

Returning to step 404, if the lock is granted, then the key is deleted in step 408. The logic then proceeds to step 412 to determine if the next node is empty. If not, the logic unlatches the node in step 421, unlocks the next key record in step 413 and returns in step 414. However, if the node is then empty, the logic proceeds to step 416 to request a conditional tree X latch. Then in step 418 the program determines if the latch has been granted. If not, in step 420, the logic unlatches the child node and then in step 422, proceeds to request an unconditional tree X latch. Once the latch is granted, either from step 422 or from step 418, the logic proceeds to step 424 to perform the node collapsing algorithm which removes the empty node and the references to the empty node from the preceding nodes in the index tree. This algorithm is

illustrated in Figure 10. In step 426 the tree is unlatched. In step 429, the deleted key record and the next key record is unlocked. It should be apparent to those skilled in the art that step 429 will occur when the transaction is complete.

Figure 9 illustrates a node splitting algorithm. In this algorithm a new node is added to the index tree to store the newly inserted key record. Additionally all the preceding, or parent, nodes in the index tree may be updated to reflect the addition of this new node and the information contained therein. In step 500, a new node is obtained. In step 502, the new node is X-Latched. In step 504 a portion of the keys from the node that is being split is moved to the new node. It should be apparent that a pointer is also included in the node being split to reference the new node. Additionally, if required, a pointer in the new node is provided to indicate the next successively located node to the new node. In step 506, the node being split and the new node are both unlatched. In step 507, the parent node to the node being split is then X-Latched. In step 508, the parent node is updated. In step 510 the parent node is unlatched. It should be apparent to those skilled in the art, that the updating of the parent node may actually include the splitting of the parent node and the updating of several successively located parent nodes up the index tree.

Figure 10 illustrates a node collapsing algorithm. The purpose of the node collapsing algorithm is to remove an empty node from the index tree to conserve memory and to provide efficient access to keys. In step 600, a node adjacent to the empty node is X-Latched. In step 602, the keys of this adjacent node are then moved to the empty node. In step 604, the pointer in the receiving node the keys is then updated to locate the node with newly loaded keys. In step 606, the next node, now empty is removed. In step 608, the latches for the child node, the node with the newly removed keys, and the adjacent node, that has been removed, are unlatched. The parent node or nodes are then X-Latched in step 610. In step 612, the parent node or nodes are updated to remove the pointer to the removed node and to adjust the key associated with the node that

receives the keys from the removed node. In step 614, the parent nodes are then unlatched. As discussed in Figure 9, the updating of the parent node or nodes maybe a recursive process as would be apparent to those skilled in the art.

The present system provides for a number of database transactions to access an index tree concurrently. This concurrent access is provided in a manner that guarantees that repeated accesses will provide consistent results within a single transaction if deemed necessary. In other words, the procedure previously illustrated provides that a transaction that is accessing an index tree during the change of index tree by another transaction will not return erroneous or invalid information that may result from accessing a portion of the index tree during this change. This is possible because of the tree latch. The tree latch is used in a way to allow concurrent accesses to the tree during the index tree change. However, the transaction protocol requires that when a condition exists wherein an error is possible, the transaction encountering this possible error condition requests a latch on the tree as opposed to returning an error condition or erroneous data. By requesting a latch on the tree, the requesting transaction must wait if another transaction that, is in fact, changing the index tree structure because this earlier transaction would have also requested a tree latch. When the tree latch is finally granted, the requesting transaction will then be assured that no changes are being made to the index tree structure during its access. This procedure is advantageous over excluding access to the index tree in that it allows concurrent accesses and provides a mechanism to insure that transaction will not retrieve information that is being changed during an index tree structure change.



CLAIMS

1. A method of operating a computer system for effecting simultaneous transactions on a database by multiple users using an index tree of record keys, said tree having a root node and lower level nodes, each node, except those in a lowest (leaf) node referencing one or more nodes in the next lower level, with those in the leaf node referencing records in the database, said method comprising, for a transaction involving insertion or deletion of a current record and its corresponding record key, the steps of:

traversing through said nodes from said root node to a leaf node using said record keys in response to an input request;

limiting to other users all but read accesses to a node being traversed and an immediately previously traversed node;

identifying a next successively located record key relative to the record key to be deleted or inserted in the leaf node and limiting all access to the leaf node;

limiting all accesses to a record associated with said next successively located record key;

inserting or deleting said current record and its corresponding record key; and

removing said access limitation to said record associated with the next successively located record key and all access limitations to the traversed nodes.

2. A method according to claim 1, in which, for inserting said current record, the removal of the access limitation to the record associated with the next successively located record key precedes the removal of all access limitations to the traversed nodes.

This Sheet Has Been Replaced



* C L M S *

3. A method according to claim 1 or claim 2, in which, for deleting said current record, the removal of all access limitations to the traversed nodes precedes the removal of the access limitation to the record associated with the next successively located record key.

4. A method according to any one of the previous claims including, for inserting said current record, the steps of:

determing if the record key associated with the current record can be inserted in said leaf node; and

if the key can be so inserted, inserting the key and the current record; or

if the key cannot be so inserted:

providing an indication that the index tree structure is to be changed;

altering the index tree to add a node to receive said current record;

removing the indication of the tree structure change; and

inserting the key and the current record.

5. A method according to any one of the previous claims including, for deleting said current record, the steps of:

determining if the record key associated with the current record is the only record in said leaf node; and

if it is not, deleting the key and the current record; or, if it is:

This Sheet Has Been Replaced

* C L M S *

providing an indication that the index tree structure is to be changed;

deleting the key and the current record;

altering the index tree structure to remove the node; and

removing the indication of the index tree structure change.

6. A digital data processing system for effecting simultaneous transactions on a database by multiple users using an index tree of record keys, said tree having a root node and lower level nodes, each node, except those in a lowest (leaf) node referencing one or more nodes in the next lower level, with those in the leaf node referencing records in the database, said system comprising means for traversing through said nodes from said root node to a leaf node using said record keys in response to an input request, means for limiting to other users all but read accesses to a node being traversed and an immediately previously traversed node, means for identifying a next successively located record key relative to the record key to be deleted or inserted the leaf node and limiting all accesses to a record associated with said next successively located record key, means for inserting or deleting said current record and its corresponding record key, and for removing said access limitation to said record associated with the next successively located record key and all access limitations to the traversed nodes.

7. A system according to claim 6 including, for inserting said current record, means for determing if the record key associated with the current record can be inserted in said leaf node, and, for when the key can not be so inserted, means for providing an indication that the index tree structure is to be changed, means for altering the index tree to add a node to receive said current record, and for removing the indication of the tree structure change prior to insertion of the key and the current record.

This Sheet Has Been Replaced

* C L M S *

8. A system according to claim 6 or claim 7 including, for deleting said current record, means for determining if the record key associated with the current record is the only record in said leaf node and, for when it is, means for providing an indication that the index tree structure is to be changed means for deleting the key and the current record, means for altering the index tree structure to remove the node, and for removing the indication of the index tree structure change.

This Sheet Has Been Replaced