Organization of multi-access in databases

*Natalya Kolyeva**

Ural State University of Economics, Yekaterinburg, Russia

Abstract. A particular, but important problem of freeing database memory in multi-access conditions is considered. A number of solutions to this problem are proposed and their advantages and disadvantages are noted. Three methods of organizing work with free space are considered. The first method assumes two additional modes of capturing pages: "Joint deletion" and "deletion". In the second method, the pages after the last tuple have been deleted are inserted at the end of the free list. The third method involves the actual presentation of the freed pages of the relationship to the free memory pool of the segment and is postponed until the relationship is reorganized.

1 Introduction

Providing multi-access to databases is one of the main requirements for modern DBMS. Users interact with databases through transactions - sets of elementary actions, the effect of which is either fully reflected in the database, or completely rejected. The task of the DBMS is to ensure the serial execution of the transaction system, i.e. perform elementary actions in such a way that the total effect is equivalent to some sequential execution of transactions. In centralized DBMSs, the most common approach is based on a two-phase capture protocol: each database object that is accessed within a transaction is captured in the desired mode, and the accumulated captures are removed only at the transaction completion phase. Currently, a wide range of conversions between object-oriented programs and DBMS are presented [1, p. fifteen]. The levels of captures can be different - from captures of whole relations to captures of elementary field values. The smaller the unit of capture, the greater the degree of asynchronous execution of transactions is allowed by the DBMS. Currently, the most widespread use of protocols of hierarchical capture type [2].

2 Study methodology

When managing memory in a DBMS under conditions of parallel access, additional short synchronization is required, which ensures consistency of data structures at the storage

* Corresponding author: nkoleva@mail.ru

level. In particular, synchronization is required when requesting and discarding external memory pages that host a database.

Thus, in the DBMS there are two levels of synchronization - long-term in relation to small objects (for example, tuples) and short-term in relation to pages of external memory. There are mutual influences between these two levels of synchronization, in particular, deadlock situations are possible in which captures of different levels are involved [3].

The article discusses one of the problems that arise in this case and possible ways to solve it. The problem is as follows. Suppose that the physical organization of the database requires that there are tuples of only one relation in each page of external memory. This is a fairly natural requirement (although not all DBMSs follow it), since it reduces the number of exchanges during sequential scan of a relation. Then, if, as a result of the operation of one or several transactions, some page of external memory is freed, the DBMS should not allow tuples of another relation to be placed in this page until the end of all transactions whose work caused the release of this page. Otherwise, the transactions that freed the page may be rolled back, resulting in tuples of more than one relationship. Transaction rollbacks can be initiated by users or performed automatically to break synchronization deadlocks.

It should be noted that this problem is rarely discussed in the literature sources known to us, although, of course, some of its solutions are present in every DBMS with a similar memory organization. Among the foreign works reflecting the direction of this research, we single out [4, 5].

The article uses a principled approach to memory organization in relational databases and terminology. The terms used, with the exception of the generally accepted ones, are defined. In particular, we refer to the term "segment" as a database partition that contains multiple relationships and corresponds to a storage file.

Memory management is a lower-level subsystem of the DBMS. It implements a structured set of objects on which the relational data model is based, and a fairly simple set of operations on these objects. To indicate objects, identifiers are used that unambiguously correspond to the objects. The memory management subsystem manages the layout of data on external media and is responsible for restoring database contents after failures and for maintaining concurrent data access.

The main control object is an n -ary relation consisting of a variable number of tuples - records with n fields, some of which may have undefined values.

The relationships that make up the database are located in one or more segments of external memory. Segments are, in fact, direct access files with constant block sizes. In DBMS terms, the blocks of a segment file are the pages of a segment. A page is a unit of exchange with external memory and buffering. Each relationship is placed in one segment.

To optimize access to tuples of a relation, indices are supported — logical reordering by individual fields or a set of fields. Index information is contained on separate pages that are in the same segment as the corresponding relation and are linked together in a B-tree.

The tuples of each relation are located in their own set of pages of one segment, i.e. tuples of several relations cannot be located on the same page. To identify tuples, indirect addressing is used: each tuple has its own pointer in the page, the position of which remains unchanged during possible movements of the tuple itself, both inside the page and outside of it. The address of a tuple pointer is a full tuple identifier that allows direct access to it and does not change over the lifetime of the tuple.

The memory management subsystem supports the parallel operation of several user transactions. In transactions, database objects are read and changed: relations are scanned in whole or in accordance with specified intervals of index key values; tuples are added, deleted, or modified. Each operation is accompanied by automatic capture of read or modification objects until the end of the transaction. For this, the protocol of hierarchical captures like [6] is used.

Segment pages, also being concurrently accessed, are briefly grabbed for the duration of an operation within the page to avoid physical mismatch in data structures.

In the course of work, two or more transactions can get into a deadlock in synchronization - a state of endless waiting for each other. A deadlock can be recognized and destroyed by rolling back one or more transactions, i.e. returning them to one of the previous states, when there were fewer capturing ones, as a result of which the rest of the transactions will be able to continue working. When a transaction is rolled back, the subsystem makes changes to the database that are the reverse of those performed by the transaction, and in reverse chronological order.

A transaction can be rolled back even if at the end of its operation a violation of the logical integrity of the database caused by the operation of this transaction is detected. Finally, a transaction can be rolled back to a previously set checkpoint if explicitly requested by the user [7].

The specifics of managing the location of information at the page level affect the efficiency of the DBMS as a whole. A number of links are maintained between pages containing useful information, the use of which speeds up access to the data. In B-trees of indexes, the subsystem supports automatic flow of information between pages of the same level, splitting and merging of pages.

The pages of each relationship are linked in a bidirectional list, which allows to insert a new page or remove a vacant page if necessary, at whatever position in the list is needed [8].

Pages that do not contain useful information form segment free space. It can be supported in a variety of ways. One of the convenient methods that do not require additional memory costs is linking free pages into a list, through the beginning of which both the addition and removal of pages occurs [9-10].

The page is a unit of exchange with external memory. To reduce the number of exchanges when performing operations on database objects, a buffer pool common to all transactions is used. The required page remains in the buffer until it is preempted by another page.

The issues of page replacement in a buffer pool, maintaining free segment space are inextricably linked to transaction synchronization, database change logging, and disaster recovery techniques. In this article, we will consider the relationship between transaction management and the technique of freeing memory, i.e. free space management of segments.

During the operation of a transaction with pages, various operations are performed. Free pages are allocated if necessary; after the last tuple has been removed from the page, it must be returned to the free space of the segment. All changes in the fields of the tuples are accompanied by the corresponding changes in the indices [11].

As already mentioned, when a transaction is rolled back, actions are performed on the data opposite to those that were performed during its direct operation. If during direct operation, a transaction placed a tuple in a new page, during a rollback, it is necessary to retrieve it and return the page to free space, unless other transactions managed to put additional tuples into this page.

Conversely, if, during direct operation, a transaction has deleted a tuple from the page, then on rollback, at least the tuple pointer must be placed in it, even if the page was returned to free space. This is because the page number is part of the tuple ID, which is unchanged from the time the tuple is created until the time it is deleted.

If in the first case there is no problem, then in the second it arises. The fact is that a page freed during the direct operation of a transaction may be occupied by another relation or index by the time the rollback begins. Then, when rolling back, it will become impossible to return the previous tuple to it. A violent release of an index page is possible in principle, but very expensive. The violent release of the relationship page is generally impossible.

In general, the necessary condition that reconciles the release of pages and possible rollbacks of transactions can be formulated as follows: after the elimination of all tuples in it, a page cannot be reused until all transactions that have deleted tuples from it have completed.

3 Study results

Here are several possible methods for organizing work with free space that satisfy the specified condition.

Method 1. Two additional page capture modes are introduced: "joint removal" and "release". Each extraction of a tuple from a page must be accompanied by its capture in the "joint delete" mode. This capture is compatible with read and page modification captures and co-delete captures. A transaction that has deleted the last tuple from a page must, before returning it to free space, grab the page in a "release" mode, which is incompatible with other page-grabbing modes, and actually block until the end of all transactions that deleted tuples from this page. Page captures in the "joint delete" and "release" modes are long lasting and, unlike read and modify captures, are saved until the end of the transaction. The actual freeing of the page, i.e. its inclusion in free space is accompanied by the release of the capture in the "release" mode and should be performed at the final stage of the transaction.

Additional blocking of a user transaction, which is undesirable for both the user and the system as a whole, can be avoided by setting the last capture of the page to release it on behalf of some asynchronous system process. The number of such processes must be at least the number of freed pages.

The considered method is not without its drawbacks. By the time the transaction ends, the freed page can be flushed out of the buffer, and additional reads will be required to include it in the free list.

A significant drawback is, of course, an increase in synchronization overheads, which are already high in a DBMS. The advent of asynchronous system processes in addition to user transactions adds complexity to the memory management subsystem and also introduces additional overhead.

Method 2. After the last tuple has been removed, pages are inserted at the end of the free list, which is maintained on a FIFO basis. In order for the method to satisfy our condition, it is necessary to ensure that the pages are not scooped up from the so-called "danger zone" of the list, where there are pages freed due to the work of still uncompleted transactions.

Thus, the list of free pages maintains three pointers: the pointer to the beginning of the list, by which free pages are selected, the pointer to the end of the list, by which the list is replenished with free pages, and the pointer to the danger zone, with which the start pointer must be compared after being changed due to the allocation of a page. If the values of these two pointers coincide, the allocation of free pages from the list of this segment is terminated until the danger zone pointer changes. This change takes place as follows.

Each transaction at its beginning remembers the current position of the end of the free list of each segment. At any point in time, the effective free-list danger-zone pointer of a segment is the same as the latched free-list end-point of the same segment of the oldest failed transaction. At the end of any transaction, the danger zone pointer can be moved, which in turn can lead to the resumption of the allocation of free pages in a guaranteed safe way.

To compare the values of pointers, one must be able to compare the positions of pages in the free list. One can enter an additional numbering of pages in this list, one can put on

the page the time of its inclusion in the free list, or use the current address of the journal filling as a timestamp, etc. can satisfy any value that monotonically increases in time.

It should be borne in mind that a page can be removed from the dangerous zone of the free list if the corresponding transaction is rolled back. This requires further complication in the organization of the list. When a page is included in the end of the list or when it is removed from the danger zone, additional pumping of the pages adjacent to the list is required to change links in them.

There are several more variations of these two methods with different organization of the free list, but they have the same disadvantages and therefore are not of particular interest.

Method 3. The most rational method seems to be when the actual representation of the freed pages of a relation into the free memory pool of a segment is postponed until the relation is reorganized, and until that time such pages can be used only for the needs of this relation.

As known, relationships "age". Deletion, insertion, and modification of tuples lead to the fact that the degree of page occupancy, generally speaking, decreases, the sometimes required ordering of tuples in relation pages in accordance with the values of a field or a set of fields (clustering) is violated. Ultimately, this leads to an increase in the number of exchanges and, therefore, a decrease in the efficiency of the system.

In order to optimize the location of information in the pages of the database, it is necessary to perform periodic reorganization with closed access to the database and thus in the absence of ongoing transactions. At this stage, all free pages are transferred to the free memory of the segment.

4 Conclusion

Note that the issues discussed concern freeing only relationship pages. There is no reuse issue with index pages. When managing the index, the main task is to reflect the changes occurring in the tuples, and it does not matter which page will be in a given node of the B-tree during the forward and reverse operation of transactions. Released index pages are immediately included in the segment's free list and can be reused for any purpose.

The content of this article can be viewed as an illustration of the increasing degree of complexity of organizing a DBMS when it is necessary to provide a multi-access mode. At first glance, the trivial task of managing free memory turns out to be far from simple, and its strict and habitually "beautiful" solutions are too expensive. Perhaps the above analysis of the problem will be useful to other developers of DBMS or other information systems that must provide multi-access mode.

References

1. Sorokin, Programmye produkty i sistemy, **3(115)** (2016)
2. Putzolo, I. Traiger, *Modelling in Data Base Management Systems* (North Holland, Amsterdam, 2016)
3. HIBERNATE – Relational Persistence for Idiomatic Java, <http://samsongych.com/lib/hibernate/>
4. E. Burtescu, Journ. of Applied Quantitative Methods, **4(4)**, 449 (2016)
5. S. Rohilla, P.K. Mittal, Intern. Journ. of Advanced Research in Computer Science and Software Engineering, **3(5)**, 810 (2013)
6. M.F. Isaeva, M.L. Gluharev, K.A. Vetlugin, Vestnik evrazijskoj nauki, **3(40)** (2017)

7. T. Konnolli, *Proektirovanie, realizaciya i soprovozhdenie. Teoriya i praktika* 3-e izdanie (2003)
8. V.M. Grachev, V.I. Esin, N.G. Poluhina, S.G. Rassomahin, *Kratkie soobshcheniya po fizike FIAN*, **5** (2014)
9. Larionceva E. A., E. YU. Ivanova, *Mashinostroenie i komp'yuternye tekhnologii*, **11** (2011)
10. V.A. Zuev, A.N. Panfilov, A.N. Skoba, *Inzhenernyj vestnik Dona*, **35(2-1)**, 24 (2015)
11. F. Wolfe, K. Michaud, *Rheumatology*, **50(1)**, 16 (2011)