Managing IBM Database 2 buffers to maximize performance

by J. Z. Teng
R. A. Gumaer

The relational data base system, IBM Database 2 (DB2), has a component that manages data buffering. This paper describes the design considerations of the Buffer Manager and the tradeoffs involved in managing the allocation of DB2 buffers to maximize performance.

IBM Database 2 (DB2)[1,2] is IBM's relational data base system for the MVS/XA (Multiple Virtual Storage/Extended Architecture) or MVS/370 operating systems, and the Buffer Manager (BM) is a component within DB2 which manages data buffering between DASD (direct access storage device) and virtual memory. It controls DB2 data in a consistent manner with respect to data integrity and data recovery. It makes the virtual storage copies of DB2 data available to other DB2 components.

IBM uses its own scheduling algorithm to manage the allocation of DB2 buffers to data base data as needed. To move data effectively between DASD and DB2 buffers, BM transfers data in 4K or 32K blocks. BM also provides a look-ahead facility to gain high performance for sequential processing applications. To increase the probability of I/O batching, BM provides a deferred write facility to defer the externalization of committed changes until BM decides that it is required to do so. In addition, BM provides functions to perform data base open/close processing and participates in DB2 restart and recovery processes.

In the following sections we first briefly describe the physical organization of the DB2 data base system and then discuss in detail the design considerations for each of the BM functions mentioned above.

Data base physical organization

The DB2 data is stored in one or more DB2 data bases. A DB2 data base, once defined, is the highest-level logical construct involved in subdividing data. Each data base consists of one or more physical data entities termed "tablespaces" or "indexspaces," which may be made on line or off line. Each tablespace contains rows of one or more DB2 tables. An indexspace contains a DB2 index structure that improves access time to the data stored in an associated table. Within the scope of BM, the tablespace is physically treated the same as an indexspace. Therefore, for ease of reference, the name "pageset" is defined to represent either a tablespace or an indexspace.

Each pageset consists of a set of Virtual Storage Access Method (VSAM)[3] Entry Sequenced Data Sets (ESDSs) and can contain a maximum of 64 gigabytes of data. Each pageset is formatted into fixed-size units termed "pages" with a page size of either 4K bytes or 32K bytes. All pages within a pageset must have the same page size, and they are uniquely identified by their relative page number. DB2 does not allow a table record to span multiple pages. Therefore, it is necessary to use 32K pages when the rows are too large to be contained within 4K pages.

To allow DB2 installations to have better control of the physical organization of their data bases based on different requirements, DB2 provides two kinds of pagesets: partitioned and nonpartitioned (simple). The major differences between a partitioned pageset and a simple pageset are as follows:

Data buffering is dependent on the pattern of references.

Partitioned pageset:

- A partition is a VSAM ESDS data set.
- The maximum partition size can be one, two, or four gigabytes.
- A maximum of 64 partitions (based on the partition size) can be defined.
- Partitions can be on or off line independently, and they can be on different device types.
- A partitioned pageset can only contain a single table, and data may be partitioned by key ranges.
- Partitions can individually be reorganized by DB2 Utilities.

Simple pageset:

- A simple pageset can contain up to 32 VSAM ESDS data sets (two gigabytes each).
- All data sets allocated to a simple pageset must be on the same device type, and they are all required to be on line when accessed.
- A simple pageset can contain multiple tables.

Open/close processing

Before referencing or modifying a page within a pageset, the pageset and its underlying VSAM data set(s) must be opened. BM provides functions to physically or logically open or close pagesets and their underlying data sets. Normally, a pageset will be closed when there are no current users of the pageset. For performance considerations, it is desirable to have frequently referenced pagesets stay open, once opened. To facilitate this, DB2 allows installations to specify CLOSE(YES | NO) when the pageset is created (through an SQL (Structured Query Language) DDL statement). This attribute can also be modified if requirements change. CLOSE(NO) pagesets stay open until the pageset (or its containing data base) is specified in a DB2 STOP DATABASE command or until the DB2 system is terminated.

Buffer pools

DB2 is designed to support large data bases with applications that might require a large number of I/O operations. It is desirable to minimize physical I/O activity whenever possible. Therefore, data base buffering is used by DB2 to transfer data between DASD and virtual memory. The unit of transfer between DASD and the DB2 buffers is a page. DB2 buffers are managed by BM, which supplies the virtual copies of the DB2 data as requested.

Data buffering is dependent on the pattern of references. It is assumed that a requested data page has a high probability of being accessed again within a short period of time. Ideally, the buffer containing the data page will still be available, and no buffer fault will be required. A buffer fault is used to indicate that an access to the data base is needed because a requested page has not been found in the buffer pool.

DB2 employs a multiple buffer pool concept by supporting three buffer pools of 4K buffers and one buffer pool of 32K buffers. DB2 installations are required to select a buffer pool when the pageset (tablespace or indexspace) is created. A facility is also provided by DB2 to allow installations to alter the selection of the 4K-page-size buffer pool after the pageset has been created. This concept of multiple buffer pools allows installations to tune their data base systems by allocating their data bases among the various buffer pools.

To best use available virtual storage, DB2 buffer pools are dynamically constructed during the process of opening the first pageset requiring use of a particular pool. A buffer pool will be deleted when it no longer has users. In a data base system that supports only fixed-size buffer pools, where the buffer pool size has to be predetermined before the system is started, the predefined size may not be able to handle all concurrent processes. Processing units may be abnormally ended because of unavailable buffer resources. To overcome this problem, the DB2 buffer manager allows the installation to define the minimum and

maximum sizes for each buffer pool. BM will use the minimum size to create the buffer pool and provides a facility to expand the buffer pool temporarily when required (e.g., when all buffers are held) if the pool's maximum size has not yet been reached. The temporarily expanded buffer pool will be contracted when the processing unit that caused pool expansion commits or aborts. Note that the DB2 pool expansion/contraction logic involves physically acquiring and releasing storage from the MVS storage pool. Because these operations are expensive, the buffer pool minimum size should be adjusted to avoid excessive use of this function. BM collects various performance monitoring data to assist the installations in determining the buffer pool size attributes. This data will be described in a later section.

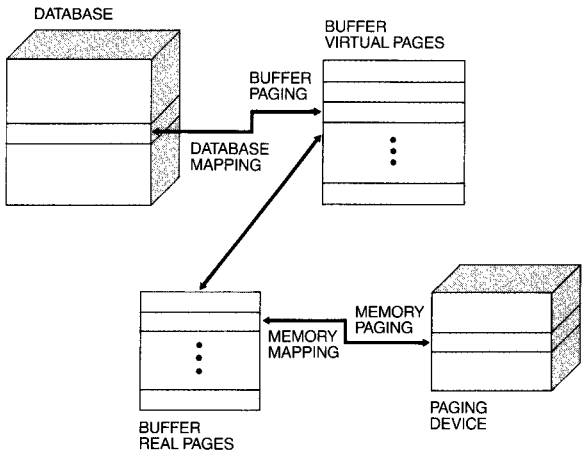
Buffer allocation algorithm

For a virtual memory data base system, the buffer space normally resides in pageable virtual memory. Therefore, double paging (data base paging and memory paging) is expected to occur if there is not enough real memory to back the data base buffers (see Figure 1).

Tuel[4] has done an analysis of buffer paging in virtual storage systems. He concluded that system performance degrades considerably when the virtual buffer requirements are larger than available real storage. His result was based on the assumption that the cost of memory paging is the same as that of data base paging. His experiment was also based on a buffer search algorithm in which the amount of memory paging was heavily dependent on the number of virtual buffers allocated. In reality, the cost of memory paging is less than that of data base paging because high-speed auxiliary storage devices are normally used as the memory paging area. However, memory paging is still expensive and should be avoided. The effect of double paging has led us to design the buffer allocation algorithm of BM to minimize buffer paging as well as memory paging.

Lang[5] has analyzed the behavior of data base buffering using various buffer allocation algorithms. He concluded that the best data base buffer model is to use prefix tables to do a buffer search and to use the Least Recently Used (LRU) algorithm to replace buffers. He further indicated that his results are applicable to data bases of any size and any locality characteristics. For the BM of DB2, it was decided to use LRU as the buffer replacement algorithm and to use a pointer array concept to do a buffer search.

Figure 1 Double paging data base buffer system



The use of a pointer array minimizes the amount of memory paging caused by a buffer search request. Each pointer array element is also called a buffer control block, and it contains the necessary buffer control information and the address of the associated data buffer. To shorten further the path length to do a buffer search, BM uses a hashing technique to locate pointer array elements. This is especially important when large buffer pools are supported. In an attempt to improve concurrency when doing buffer searches, BM serializes on the appropriate hash anchor. Thus, buffers residing on different hash anchors are allowed to be searched or updated concurrently.

As indicated in Lorie's paper,[6] System R[7,8] forces some pages to remain in the buffer pool, independent of how the normal LRU algorithm functions. Each process attempts to keep (i.e., fixing pages and not releasing them) a certain number of pages in the buffer pool for its own look-aside purposes. The process is informed to unfix its look-aside pages when the buffer pool is short of reassignable buffers. DB2 provides a similar facility to improve processing performance when the associated buffer pool is not short of reassignable buffers.

RDS Sort[9] also uses BM functions for sorting temporary work files. Temporary work files are constantly being used or reused as a temporary merging area. Therefore, read I/Os are unnecessary when the initial contents of the data pages can be ignored. To improve sort performance, BM provides a special service which allows pages to be accessed without reading them from DASD.

As mentioned earlier, BM provides a facility to temporarily expand or contract buffer pools. However, due to the potential shortage of virtual storage, it is unable to guarantee that the expansion is always successful when a buffer is required. There are cases where a buffer is required to satisfy a "must-complete" DB2 process and unavailability of a buffer would cause DB2 to be abnormally terminated. In order to support these "must-complete" functions, BM provides a buffer-enqueue facility that enqueues the process, which is waiting for a buffer, until a free buffer becomes available. To avoid the problem of deadlock (i.e., waiting for a buffer while the "must-complete" process is holding all allocated buffers), the "must-complete" DB2 process is limited to holding just one buffer at a time. The "must-complete" status is indicated by BM invokers through the normal page access function.

To overcome the synchronous write problems, DB2 supports two basic write protocols.

Write protocols

Data buffers that have been modified must eventually be written back to their respective data bases. In a data base system that only supports synchronous write protocols, write I/Os must be performed under the process during commit for changes that had been made prior to the commit point. This operation not only prolongs the time to do commit processing because of write I/O delays, but also stops other processes that need to update the same set of pages.

To overcome the synchronous write problems, DB2 supports two basic write protocols. They are designated as "system page write protocol" and "UW (Unit of Work) page write protocol," respectively. UW page write protocol is available only to the DB2 Utility component and is not available to SQL transactions. "Unit of Work" is equivalent to one DB2 thread.[9] Writes can be initiated prior to, during, or after phase 2 of commit,[9,10] depending on whether the system page write or the UW write protocol is used. Selection of a particular write protocol has implications in the areas of locking, logging, and recovery.

System page write protocol and deferred write processing. DB2 data base pages controlled by the system page write protocols of BM are called *system* pages. System pages can be written at any time, except during actual update. Control of these writes is asynchronous to transaction processing, and therefore, the write can occur either before or after the updating transaction commit point. The fact that write processing can occur prior to commit processing requires that undo information be logged so that uncommitted changes residing on DASD can be backed out in cases of transaction abort or a system crash. The redo information is also required to support the possibility of not externalizing committed changes because of the system crash.

System pages are normally enqueued, when updated, on the system Deferred Write Queue (DWQ) for later batched write processing—as opposed to writing them immediately. This delay minimizes the number of write operations required for frequently updated pages. The disadvantage here is that buffers on the DWQ are unavailable for reuse, thereby increasing the potential for a transaction abort because of buffer unavailability. For this reason, the buffer manager will immediately write system pages upon update completion whenever the buffer pool is short of reassignable buffers.

A deferred write processor engine is used to process write I/O requests for pages eligible to be written from the DWQ. Because of the restrictions of the internal I/O driver used by DB2, I/O processing can only be batched on an individual VSAM data set basis. To maximize I/O concurrency, the deferred write processor attempts to schedule asynchronously or reuse as many as possible of the write engines provided by DB2 to perform write operations. The deferred write processor will use itself as a synchronous write engine if all asynchronous write engines are busy.

For ease in relating I/O requests by individual data set, the DWQ is structured as a queue of data-set-related queues. For situations where the process must write out all or certain updated pages (i.e., Checkpoint, Pageset Close Processing), those pages will be dequeued from the DWQ, and writes will be initiated. Otherwise, for situations where the process is not required to write out updated pages, the decision to trigger the deferred write processing is based on the

amount of buffer space dedicated to the deferred write buffers. The objective is to keep a certain amount of buffer space available for read operations and for reassignments by scheduling write operations as required.

UW page write protocol. Data base pages processed by DB2 Utilities for which writes are controlled by the UW write protocols are called UW pages. UW pages, when updated, are controlled exclusively by the transaction process. They are only written during (not prior to) phase 2 of COMMIT. This precludes the need for undo logging since no uncommitted changes will ever appear on DASD that would require backout because of an abort or system crash.

Sequential prefetch

In a data base system, the data referencing behavior has great performance impact on data base buffering. Rodriguez-Rosell[11] had done an empirical study on the data referencing behavior using IMS (Information Management System)[12] as the data base system. He indicated that strong sequentiality was found in data base reference strings. He also pointed out that sequentiality of access is a predictable consequence of data base organization and transaction processing. Therefore, if a process has knowledge of the data organization and decides to access it sequentially, it is desirable to have the data base buffer manager take certain actions to minimize the I/O delays for this type of process. This fact has led BM to provide support for sequential-type processing. The DB2 utilities (e.g., LOAD, IMAGE COPY, . . . , etc.) notify BM whenever sequential processing is to be done.

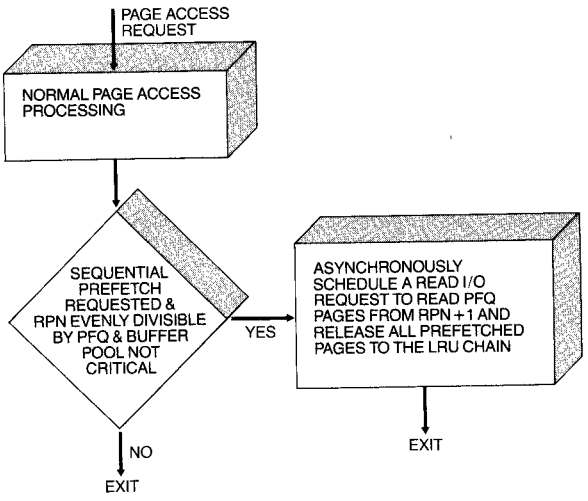
As noted in the paper by Sacco and Schkolnick,[13] a problem exists in cases where the buffer pool is shared by concurrent random and sequential processes. The LRU buffer replacement technique is inadequate to support the transactions performing random access if the sequential process has a higher demand rate for new pages. This results in a higher buffer paging rate for the entire data base system. Therefore, the DB2 Buffer Manager (BM) provides support to minimize the effect of this problem. The support is based on the assumptions that the probability of reaccess to sequentially processed pages is much less than the probability of reaccess to randomly processed pages and that the buffer pool's use is primarily for random processing with only occasional bursts of sequential processing. What basically happens is that a secondary LRU chain called the "Sequential LRU Chain" (SLRU) is employed. Buffers released from sequential processing are placed on the SLRU chain (as well as on the tail of the normal LRU chain). Buffers required to satisfy sequential access requests are obtained from the SLRU chain (if the number of buffers residing on SLRU is greater than the "sequential steal" threshold value "Q"). Otherwise the buffers are obtained from the head of the normal LRU chain. This logic gives buffers recently used for sequential access a higher steal priority for new sequential requests than buffers last used for random access, without altering buffer steal priority for new random requests. This minimizes the monopolization effect of sequential processing. The "sequential steal" threshold value Q is predetermined by BM and is not specifiable by DB2 installations.

Prefetch occurs asynchronously as a side function of the buffer manager page access function.

For sequential processing applications, the data that must be referenced is known ahead of time. It is desirable to have the data available in buffers when it is being referenced. Smith[14] proposed a general prefetching algorithm: If a requested data page is not in the buffer pool, schedule an I/O request to prefetch n pages following the requested page. For a truly sequential process, Smith's proposal can only reduce I/O delays by a factor of n (i.e., batching I/O operations in multiples of n pages). To extend his concept to improve I/O throughput for a truly sequential process, BM provides the asynchronous prefetch facility.

Prefetch occurs asynchronously as a side function of the buffer manager page access function. It is accomplished by scheduling an asynchronous execution unit that acquires necessary buffers from the buffer pool, loads them with the required data, and then releases them to LRU status. Note that the prefetched buffers are not placed on the SLRU chain until they are individually accessed. This is done to minimize the stealing of prefetched buffers that have not yet been accessed.

Figure 2 Sequential prefetch logic flow



To prevent multiple prefetches from being scheduled for the same pages by the same sequential process, the number of consecutive pages prefetched (PFQ) must be a value that is a power of two. BM will only schedule prefetch when the requested page number (RPN) is evenly divisible by the PFQ. Prefetch will start at the currently requested RPN plus one. The last page prefetched is the next sequential page whose RPN is evenly divisible by the PFQ. This ensures that once a prefetch sequence has been initialized, all subsequent pages will have been prefetched prior to access as long as the sequence remains unbroken. Figure 2 shows the logic flow of the BM sequential prefetch algorithm.

Buffer pool performance tuning

For the purpose of studying the behavior of DB2 buffer pools, BM collects the performance monitoring data listed in Table 1. GET is incremented by one each time the BM page access function is requested. The incrementing does not include those page access requests performed within the look-aside buffers of each process (i.e., buffers fixed by each process for future references). Thus, GET tends to be lower than the total number of pages referenced. However, GET is valuable when studying the effectiveness of the BM buffer look-aside capability. RIO is incremented by one each time a requested data page has not been found in the buffer pool, and a read I/O operation is required to access the data base. The ratio RIO/GET could be used as an indication of the efficiency of the buffer look-aside capability. The level of efficiency increases as the ratio approaches zero. Factors that can affect this ratio are

1. Buffer pool size
2. Locality of reference across the mix of applications accessing the data base data assigned to this buffer pool

Increasing 1 and/or 2 could result in the lowering of this ratio (raising the efficiency level of the look-aside capability of the buffer pool).

SWS is incremented by one each time a data base logical record residing in a system page is updated. PWS is incremented by one each time a system page is written to DASD. Each update of a system page does not necessarily result in a page write (multiple updates are allowed per system page write). The ratio PWS/SWS could be used as an indication of the efficiency of the BM deferred write logic. The level of efficiency increases as the ratio approaches zero. User-controllable factors that can affect this ratio are

1. Buffer pool size
2. Level of concurrent buffer pool usage by multiple transaction instances (each accessing different data base data)
3. Transaction rate of transaction instances which update the data base pages

Increasing 1 and/or 3 and/or decreasing 2 could result in the lowering of this ratio (raising the number of updates per system page write).

Similarly, SWU and PWU are used to collect update counts for UW pages. However, UW pages are exclusively held by the process until it commits. There-

Table 1 Performance monitoring data

GET	Number of page access requests received
RIO	Number of read I/O operations performed
SWS	Number of requests received to fix System pages for update
PWS	Number of System pages written to DASD
SWU	Number of requests received to fix UW pages for update
PWU	Number of UW pages written to DASD
WIO	Number of write I/O operations performed

fore, each UW page write can only contain updates by the same process within the scope of a commit, and so the ratio of PWU/SWU should always approximate one.

WIO is incremented by one each time a data base write request is performed by the internal I/O driver. The ratio WIO/(PWS + PWU) could be used to measure the efficiency of batching write I/O operations. The level of efficiency increases as this ratio approaches zero. Factors that can affect this ratio are

1. Buffer pool size
2. Level of concurrent buffer pool usage by multiple transaction instances (each accessing different data base data)

Increasing 1 and/or decreasing 2 could result in the lowering of this ratio (raising the number of pages written per write request to the internal I/O driver).

BM does not perform any direct restart function for DB2.

The role of BM in DB2 restart

BM does not perform any direct restart function for DB2. However, it provides pageset, data set, and page processing services for use by the DB2 Data Manager (DM) component, which controls pageset-related restart processing. Services provided by BM are as follows:

1. "Pageset and data set" log records. These records are produced at DB2 checkpoint time, and whenever a pageset or its data set are opened or closed. They are used by the DM to control pageset/data set open and close during restart.
2. The BM page access protocols are used during restart. These protocols are used by the DM to perform the I/O operations needed to recover DB2 data during restart.

To help DB2 restart from the most recent checkpoint, BM writes checkpoint log records for each recoverable pageset found to be open. This information will be used at restart to reopen those pagesets that were open at checkpoint time. Within each checkpoint log record group, BM supplies a checkpoint log that indicates the log starting point for the associated pageset-related processing. BM provides logic to determine, for each pageset, the most recent checkpoint log record RBA value that precedes the oldest update log record corresponding to those updated pages that may not yet have been externalized. This is required to ensure that these pages are included within the scope of recovery.

The role of BM in media failure recovery

BM provides the following support for media failure recovery:

1. Initial detection of media failures through receipt of I/O error return codes from the internal I/O driver. Whenever this occurs, relevant information is collected and saved to be used by the DB2 RECOVER Utility in recovering DB2 data.
2. A write error page range for each data set. Access to pages within this range is prohibited for processes other than the RECOVER Utility.
3. A service to record special log-range entries for recoverable pagesets that are modified. These log-range entries are used during the RECOVER process to bypass DB2 log records that need not be processed.

Data availability is very critical to the success of a data base system. For large data bases, off-line recovery would be tedious and would cause large amounts of data to be unavailable for an extended period. Thus, it is preferable for the system to do on-line recovery of interrupted transaction processes.

DB2 data pages may be broken by interrupted transaction processes. For this type of broken page condition, DB2 automatically does on-line recovery. Broken pages are detected either at the pageset critical function cleanup process of BM (which is scheduled when data pages could not be written back to DASD for a long period of time due to interrupted transaction processes) or when the pageset is being closed. Prior to initiating the recovery process, BM ensures that no read or write activities are in progress against the associated pageset. Then BM invokes a DM-supported log recovery service to reconstruct the page(s) from the DB2 log.

Concluding remarks

This paper has described the design considerations of the DB2 Buffer Manager and its role in the DB2 system. As noted above, BM plays a very important role in DB2 data integrity and data recovery. It also has great impact on overall DB2 system performance.

DB2 is IBM's first relational data base system running in the MVS/370 and MVS/XA environments. At the moment, there are no published reports on the behavior of DB2 buffer management. It is anticipated that a deeper understanding of this behavior will occur over time. We hope to identify additional enhancements to improve DB2 performance.

We also hope that data processing personnel will now better understand the role of buffer management in a data base system. This knowledge should help DB2 produce a well-planned design for DB2 applications. In addition, the information in this paper should assist DB2 installations in the performance tuning of their DB2 systems.

Acknowledgments

The authors gratefully acknowledge the efforts of R. A. Crus, D. J. Haderle, H. W. Herron, M. M. Roney, A. Shibamiya, and P. S. Worthington in reviewing the paper and for contributing valuable suggestions. The authors also thank DB2 management, specifically M. J. Bohl, M. M. Roney, and G. R. Young, for their support.

Cited references

1. *DB2 General Information Manual,* S370-20, IBM Corporation; available through IBM branch offices.
2. S. Kahn, "An overview of three relational data base products," *IBM Systems Journal* **23,** No. 2, 100–111 (1984, this issue).
3. *OS/VS VSAM Programmer's Guide,* GC26-3838, IBM Corporation; available through IBM branch offices.
4. W. G. Tuel, Jr., "An analysis of buffer paging in virtual storage systems," *IBM Journal of Research and Development* **20,** No. 5, 518–520 (September 1976).
5. T. Lang et al., "Database buffer paging in virtual storage systems," *ACM Transactions on Database Systems* **2,** No. 4, 339–351 (December 1977).
6. R. Lorie, "Physical integrity in a large segmented database," *ACM Transactions on Database Systems* **2,** No. 1, 91–104 (March 1977).
7. M. M. Astrahan et al., "System R: A relational approach to database management," *ACM Transactions on Database Systems* **1,** No. 2, 97–137 (June 1976).
8. M. W. Blasgen et al., *System R: An Architectural Update,* Research Report RJ-2581, IBM Research Division, 5600 Cottle Road, San Jose, CA 95193 (July 1979).
9. D. J. Haderle and R. D. Jackson, "IBM Database 2 overview," *IBM Systems Journal* **23,** No. 2, 112–125 (1984, this issue).
10. R. A. Crus, "Data recovery in IBM Database 2," *IBM Systems Journal* **23,** No. 2, 178–188 (1984, this issue).
11. J. Rodriguez-Rosell, "Empirical data reference behavior in data base systems," *Computer* **9,** No. 11, 9–13 (November 1976).
12. *IMS/360 General Information Manual,* GH20-0765, IBM Corporation; available through IBM branch offices.
13. G. M. Sacco and M. Schkolnick, *A Mechanism for Managing the Buffer Pool in a Relational Database System using the Hot Set Model,* Research Report RJ-3354, IBM Research Division, 5600 Cottle Road, San Jose, CA 95193 (December 1981).
14. A. J. Smith, *Sequentiality and Prefetching in Data Base Systems,* Research Report RJ-1743, IBM Research Division, 5600 Cottle Road, San Jose, CA 95193 (March 1976).

Reprint Order No. G321-5219.

James Z. Teng *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Dr. Teng is an advisory programmer at IBM's Santa Teresa Laboratory. He started his career at Sperry UNIVAC in Roseville, Minnesota, in 1976. He was involved in the design and development of UNIVAC's Functional Mathematical Programming System (FMPS) in the areas of mixed integer programming. In 1978, Dr. Teng joined IBM, where he worked on the Business Information System (BIS) project at the Santa Teresa Laboratory. He was involved in the design and development of various financial systems such as BISPLAN (a tool for project planning), the Cost Measurement System, and the Labor Distribution System. In 1980, he joined the Database 2 development group. Since then, he has been involved in the design and development of the Buffer Manager component. Dr. Teng received a B.S. degree in mathematical statistics from Tamkang University, Taiwan, in 1970. He received an M.S. degree in computer science and a Ph.D. degree in statistics from the University of Wisconsin, Madison, in 1975 and 1978, respectively.

Robert A. Gumaer *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Gumaer joined IBM in 1955 as a customer engineer in Pittsburgh, Pennsylvania. Since transferring to San Jose in 1969, he has worked on the design and development of various data management and data base management products, including VSAM and DB2. Mr. Gumaer is currently an advisory programmer in the Storage Management Department at the Santa Teresa Laboratory.