PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1984,
edited by ICOT. © ICOT, 1984

Design and Implementation of the Relational Database Engine

Hiroshi Sakai, Shigeo Kamiya, Kazuhide Iwata, Masaaki Abe, Akio Tanaka
Toshiba R & D Center
Kawasaki, Japan

Shigeki Shibayama, Kunio Murakami
ICOT Research Center
Tokyo, Japan

Abstract

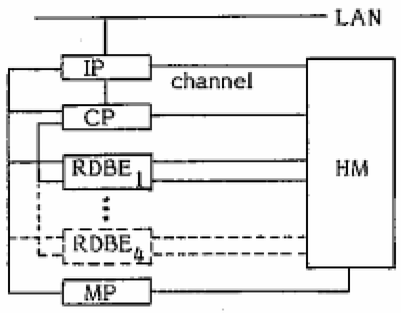
Delta, a relational database machine, is now under development at ICOT (Institute for New Generation Computer Technology) to study knowledge base machines (KBMs). Delta is characterized by its specialized processor, RDBE (Relational Database Engine) and its large semiconductor memory. This paper focuses on the RDBE and describes its design considerations, functions, implementation, and performance analysis.

1 INTRODUCTION

Development of a KBM (Knowledge Base Machine) is one of ICOT's major themes. By KBM, we mean a machine that can manage a large amount of knowledge, offer inference machines simultaneous access to the knowledge and respond quickly to these accesses. Development of relational database machine Delta (Shibayama 84) and research on the connection of inference machines to it are being conducted as a three year project.

The reasons the relational model was adopted are (1) Delta was required to have fundamental data management ability so that knowledge management software would work well on the inference machine, and (2) the relational model seemed more profitable for a logic programming system than any other database models (Codd 70) (Gallaire 78).

In the design of Delta, we aimed at high performance, a large amount of secondary storage, and an intimate interface with logic programming, since Delta was planned to be connected to many inference machines via a local area network (LAN). However, the relational database systems developed for conventional computer systems and commercial database machines are far from meeting our performance requirements. Especially in cooperation with a logic programming system, each attribute is required to be handled at a reasonable speed and processing the null value is to be taken into account. So, we have designed a new database machine for this purpose.



LAN	:	Local Area Network
IP	:	Interface Processor
CP	:	Control Processor
RDBE	:	Relational Database Engine
MP	:	Maintenance Processor
HM	:	Hierarchical Memory

Fig. 1 Delta architecture

2 OVERVIEW OF DELTA ARCHITECTURE

Delta's global architecture is shown in Fig.1. In this figure, the dotted line shows the final configuration. Delta consists of the following components:

(1) An interface processor (IP), which connects Delta to a local area network (LAN) and the Multibus.
(2) A control processor (CP), which provides database management functions, such as concurrency control and database recovery.
(3) A relational database engine (RDBE), which is the key component for processing relational database operations in Delta. The RDBE is implemented by the combination of a general-purpose processor with a specialized processor.
(4) A maintenance processor (MP), which provides functions that enhance Delta's reliability and serviceability.
(5) A hierarchical memory (HM), which provides functions for storing, accessing, clustering and maintaining relations. The HM is implemented using a general-purpose processor as a controller, a high volume semiconductor memory and large-capacity moving-head disks. The HM is connected to other

components through high-speed channels. The HM can be seen as a large semiconductor buffer with a very short latency and fast transfer from the RDBE.

The general Delta command processing sequence is as follows. The IP receives relational algebra level commands, called Delta commands, from a host connected to the LAN, and sends them to the CP. The CP translates these commands into a sequence of internal subcommands, which are then issued to the RDBE and HM to make them cooperatively perform the specified database operation. After execution of the Delta commands, the IP transfers the result stored in the HM to the host via the LAN.

3 DESIGN AND IMPLEMENTATION OF RDBE

3.1 Design Considerations

3.1.1 Basic Idea

The basic idea of RDBE processing mechanisms is that a join operation is performed efficiently by sorting tuples of each relation according to their values and comparing tuples from the relations in a manner resembling a two-way merge operation. This idea is profitable since it is applicable not only to the equi-join operation but also to nonequal join operations and other relational database operations that take two relations. This idea has also been realized in RDBM (Hell 81) and other database machines. The advantages of the RDBE, however, are the following:

(1) The combination of the sorter and merger improves performance as in pipeline processing.
(2) RDBE can process null values and duplicate values efficiently.
(3) The projection operation is performed during another operation.
(4) Parity check and sorting check mechanisms improve reliability.
(5) Data processing by the RDBE's CPU gives RDBE functional flexibility.

3.1.2 Stream Processing

The RDBE was designed to perform relational database operations on data stored in the HM. Two block multiplex channels are provided for each RDBE, so that data may be transferred at high speed. Whenever the RDBE performs an operation, data is transferred from HM to RDBE and from RDBE to HM.

There exist two alternatives. One is to place the RDBE between the HM's semiconductor memory and its moving-head disks, as in VERSO (Bancilhon 82). This would reduce data transfer time and improve system throughput. However, it is difficult to modify the HM hardware and its operating system, since the HM is implemented using a mainframe to achieve a high volume semiconductor memory, large capacity secondary storage, and high speed data transfer. Besides, the 128 Mbyte semiconductor memory of the HM behaves like a disk cache memory.

The other alternative is to make the RDBE access the HM's semiconductor memory directly. This would improve performance if the data is small enough. However, the memory access bus would be a bottleneck, since sort operations, in general, cause frequent access to the key field of each tuple. It is hoped, however, that the HM can be made to perform the operation on small volumes of data, even though this has not yet been adopted.

3.1.3 Processing the entire tuple

The RDBE takes the entire tuple, not only the key field. One alternative is to process the value field, which would reduce the data transfer between RDBE and HM, as well as the required RDBE's memory. The reasons we did not adopt this alternative are (1) processing the entire tuple would become necessary in the HM, (2) the set operation requires comparison of all the fields of a tuple.

3.1.4 RDBE commands

The RDBE offers various kinds of commands necessary for relational database processing. They are classified into the following categories:

(a) Relational algebra commands, such as join, projection, and selection
(b) Sort operation commands in both ascending and descending order
(c) Set operation commands, such as intersection, union and difference
(d) Arithmetic operation commands
(e) Aggregate commands over an entire relation and also over a nonintersecting partition of a relation
(f) Miscellaneous commands specific to the way in which Delta manages data.

The whole list of commands is shown in Fig. 2. The RDBE performs these commands using its hardware modules, the sorter and merger, and also using its general-purpose CPU.

Command name	Comments
PASS	Intratuple operation
JOIN	$=,\neq,<,\leq,>,\geq$,Cartesian product
RESTRICT	$=,\neq$,range
SORT	ascending/descending
AGGREGATE	aggregate operation
UNIQUE	eliminating duplicate tuples
UNION	set operation
INTERSECTION	set operation
DIFFERENCE	set operation
EQUAL	equalty test between relations
CONTAIN	inclusion test between relations
COMPARE	compare attributes of each tuple
ZONE-SORT	for clustering
DELETE	for updating

Fig.2 List of RDBE commands

The sorter and merger were designed to perform intertuple commands, i.e. commands which require comparison between tuples. The sorter and merger are able to compare a field of a tuple with a field of another tuple, i.e., typically an attribute or the entire tuple. The sorter and merger are also able to compare a field of a tuple with constant values, and compare fields within a tuple. The rest of the commands are performed by the CPU itself or the combination of the CPU and the sorter and merger. This decision was made considering the cost/efficiency and functional flexibility of the RDBE.

The internal representation of a tuple is as follows. Each tuple in a relation has the same length (less than 4 Kbytes) and the same number of fields; corresponding fields over a relation have the same data type and length. A field usually has an extra area called a tag, which indicates whether the value is null. The data types are unsigned integer, signed integer, and single-precision floating point. The length of the first two types must be even and less than 4 Kbytes.

3.2 Configuration

The RDBE configuration is shown in Fig. 3. It is designed for high-speed relational database processing by means of pipelined sorting and merging.

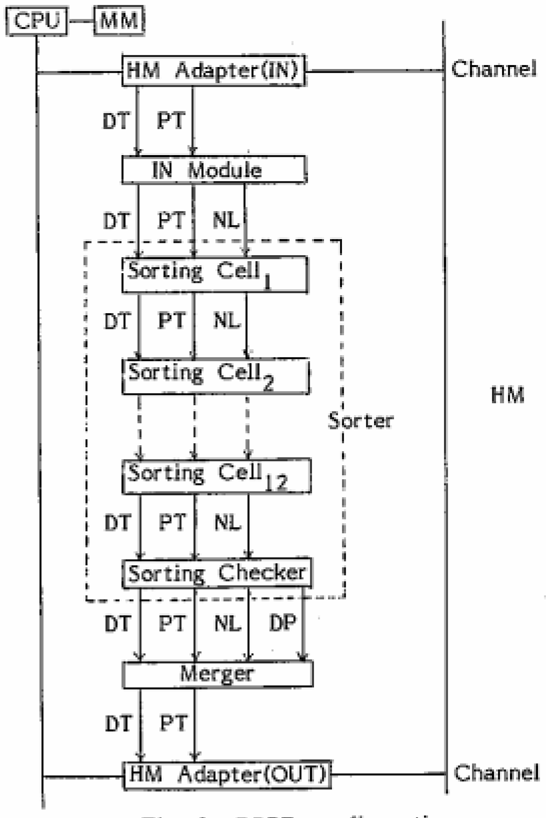


Fig. 3 RDBE configuration

The RDBE is implemented using the following modules:

(1) A general-purpose CPU, which is used as the RDBE controller.

(2) Two HM adapters, which serve as interfaces between the RDBE and HM.

(3) The IN module, which transforms input data into an internal format suitable for the sorter and merger modules. Among these transformations are:
   * Field ordering, which shifts a key field to the head of the tuple
   * data type transformation
   * generation of null value bit signals

(4) A sorter, which generates sorted tuples.

(5) A merger, which performs external sorting and relational database operations using a processing algorithm based on a two-way merge operation.

In Fig. 3, DT,PT,NL and DP stand for data lines, parity lines, a null line and a duplication line, respectively. The null line is used to denote that there is a tuple with a null value key on the data lines. The duplication line is used to denote that there is a tuple having the same key value as the subsequent one on the data lines.

These modules are controlled to run simultaneously. Data transfer is performed in the handshake mode between these modules. Each module is designed to achieve a data processing rate as high as the data transfer rate between the RDBE and HM.

The main data path is from the HM adapter (IN) to the HM adapter (OUT) through the IN module, sorter, and merger. If an RDBE operation takes two relations, as in a join operation, the operation is performed in the following way. The tuples of the first relation, which was originally stored in the HM, pass through the HM adapter (IN), are modified by the IN module, sorted by the sorter, and are finally stored into a buffer of the merger. Then the tuples of the second relation pass through the HM adapter (IN), are modified by the IN module, sorted by the sorter, and are stored into another buffer of the merger. While storing the second tuples, the merger also compares these with the previously stored tuples, and generates the results. They are sent to the HM through the HM adapter (OUT).

If the CPU itself is required to manipulate the data, the result from the merger is sent to the CPU's main memory via the HM adapter (OUT). After the CPU has finished the manipulation, the final result is sent to HM via the HM adapter (OUT).

3.3 Sorter

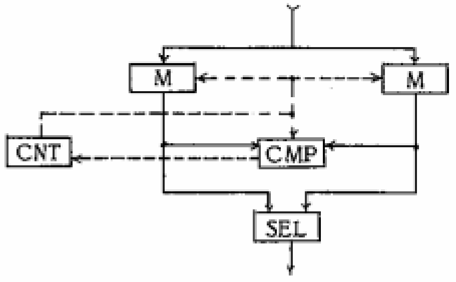
In order to apply a sorter in the RDBE environment, the following conditions must be satisfied:

(1) It receives the original sequence of tuples from the IN module, and sends the sorted sequence to the merger.
(2) The data transfer rates both at its entrance and at its exit are equal to that between the RDBE and HM.
(3) The delay between the ending of input data transfer and the beginning of the output data transfer is small.
(4) It is able to sort a small number of tuples at reasonable speed.
(5) It is able to process absolute values of the standard binary notation up to 4094 bytes long, possibly with the null value signal on.

Various kinds of sorting algorithms have been studied (Knuth 73), and hardware sorters based on them have been proposed and implemented. Tanaka proposed and implemented a sorter based on the heap sort (Tanaka 80). Although it satisfies the above four conditions, it is difficult to implement so as to it satisfy the condition(5).

Our sorter, based on the two-way merge-sort, is similar to Todd's (Todd 78). It is slightly inferior to Tanaka's for the third condition, but it satisfies the condition(5). Our sorter has the following features:
(1) The sorter consists of a linear array of 12 processing elements, called the sorting cell, and one processing element, called the sorting checker; these arrange input data elements in a specified linear order (ascending or descending). Since the data bus consists of 16 lines, the unit size of data, word, is two bytes. The sorting operation is performed by pipeline processing.
(2) The sorter performs only the internal sort operation. The maximum number of tuples that the sorter is able to process is shown by the following expression.
min(2**N, [M/L])
Here, N is the number of sorting cells (currently 12), M is the memory size of the last sorting cell (currently 64 Kbytes), and L is the tuple length.
(3) The sorting cell has two operation modes: the sort mode and the pass mode. The sort mode merges two sorted sequences of tuples into one. The pass mode does not merge, but transfers input data directly to the next cell.
Let C be the number of tuples to be sorted, then [$\log_2 2$(C-1)] of the sorting cells become the sort mode, and the others become the pass mode. The sorter takes (2LC+N-1)T, excluding the time which the sorting checker and the control program take. Here, T is the time required to transfer one byte (currently 330 nsec). For example, 4096 tuples of 16 bytes are sorted in 43 milliseconds.
(4) The sorter processes null values by recognizing the tag field and locates them at the last part of the sorted sequence.
(5) The sorter performs stable sort operations on equal values, i.e., it keeps the original relative order of the input sequence of tuples having the same values.
(6) The sorting checker compares the key field of each tuple with that of the next one, so that it checks the results to increase the reliability of the sorter. It also generates the duplicate signal when the values are the same. Since it takes an additional time of LT, the time required is (2LC+L+N-1)T excluding the software overhead time.

Fig.4 is a block diagram of the sorting cell. It contains two memories, each with a first-in/first-out function (FIFO), a comparator and a control circuit.



M	:	Memory
CMP	:	Comparator
CNT	:	Controller
SEL	:	Selector

Fig. 4 Block diagram of a sorting cell

The sorting cell operation for every two bytes consists of three cycles. They are memory read cycle, another memory read cycle, and compare-transfer cycle. In the first cycle, the word of the first sorted subsequence is read and stored into the register of the comparator. In the second cycle, the same operation for the other subsequence takes place. In the last cycle, the comparator compares them and the selector outputs the smaller or greater word to the (i+1)th cell, according to the ascending or descending mode. In this cycle, the word sent from the (i-1)th cell is stored into the memory. Each cycle takes 220 nsec, and the two-byte merge operation takes 660 nsec.

3.4 Merger

The merger is the central module of the RDBE, which performs relational algebra operations and other operations using a processing algorithm based on the two-way merge-sort operation. These are called merger commands and are classified into the five types of operations listed in Fig. 5. They are characterized by their ability to process null values and duplicate values.

A block diagram of the merger is shown in Fig. 6. The merger consists of an operation section and an output control section.

PASS COMMAND	RESTRICT COMMAND
LOAD	REST-NULL
PASS-1(NOP)	REST-NONULL
PASS-2(UNQ-IN)	REST-EQ
SORT COMMAND	REST-NE
SORT-IN	REST-RANGE
SORT-EX	
UNQ-EX	
COMPARE COMMAND	JOIN COMMAND
COMP-ALL	JOIN-ALL
COMP-NULL	JOIN-NULL
COMP-NONULL	JOIN-NONULL
COMP-EQ	JOIN-EQ
COMP-NE	JOIN-NE
COMP-LT	JOIN-LT
COMP-GT	JOIN-GT
COMP-LE	JOIN-LE
COMP-GE	JOIN-GE

NOP : No operation UNQ-IN : Unique-Internal
EX : External NONULL : Not null

Fig. 5 List of merger commands

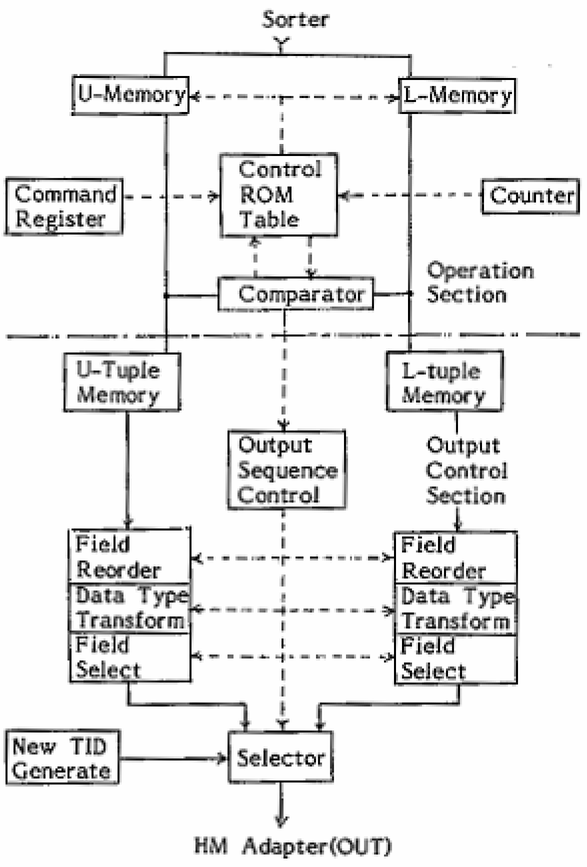


Fig.6 Block diagram of the merger

The operation section contains a comparator, a control ROM table, and two 64 Kbyte memories (U-memory and L-memory) having a FIFO function. This section performs the following steps:

(1) Storing two sorted streams from the sorter into the memories
(2) Reading a tuple from each of two memories simultaneously and providing them to the comparator and the tuple memory in the next section
(3) Comparing the keys of each tuple and detecting output tuples satisfying the conditions of the command

These functions are executed under the control of a 1-Kword * 10-bit ROM table. The address of the ROM table consists of a null signal, duplication signals, the comparison result flag and so on. The output of the ROM table consists of memory address control signals, tuple-selection signals used for the output control section, and an operation-end signal.

The output control section consists of two 16 Kbyte tuple memories, two field-ordering circuits, two field-selection circuits, two data-type-transformation circuits, a new TID (tuple-identifier) generator, a selector and an output sequence controller. This section performs the following functions under software control:

(1) Reordering the fields of an output tuple
(2) Selecting fields of an output tuple
(3) Recovering the original notation of the key field
(4) Adding a new TID to an output tuple

Examples of these functions are shown in Fig.7. Fig.7(a) shows the reordering of the fields of an output tuple. A tuple(1) with five fields (A, B, C, D, E; B is a key field) is rotated to tuple(2) by the IN module, so that the key field is positioned at the head of the tuple, and tuple(2) is rearranged to the original tuple(3) by the merger.

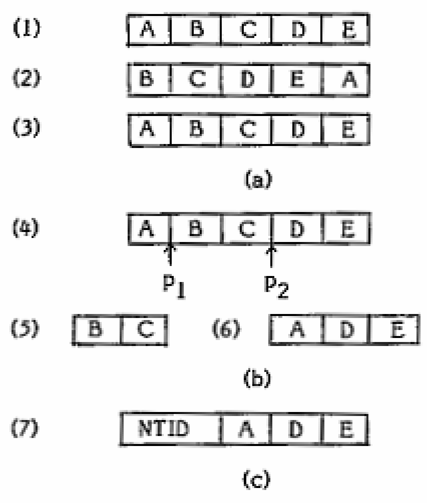


Fig. 7 Functions of the output control section

The selection of fields of an output tuple is shown in Fig. 7(b). In this figure, tuple(4) is projected to tuple(5) or tuple(6) by the assignment of the two pointers, p_1 and p_2.

Fig.7(c) shows the addition of a new TID to an output tuple.

An example of the JOIN-EQ operation is illustrated in Fig.8. JOIN-EQ command is typically used when an equi-join of two relations is performed.

Stream S1 (U-Memory)

(a)

UADR	A1	A2
0	2	B
1	3	C
2	3	D
3	4	A

Stream S2 (L-Memory)

LADR	B1	B2
0	1	T
1	3	S
2	3	V
3	5	U

JOIN A1=B2

(b)

A1	A2	B1	B2
3	C	3	S
3	C	3	V
3	D	3	S
3	D	3	V

(c)

UADR	LADR	U-Tuple	L-Tuple	Result	Output
0	0	2 B	1 T	>	
0	1	2 B	3 S	<	
1	1	3 C	3 S	=	3 C 3 S
1	2	3 C	3 V	=	3 C 3 V
2	1	3 D	3 S	=	3 D 3 S
2	2	3 D	3 V	=	3 D 3 V
3	3	4 A	5 U	<	
END	3				

Fig. 8 Example of processing JOIN-EQ command

Fig.8(a) shows two input streams (S1 and S2) sorted in ascending key-order (A1 and B1); these are stored in the U- and L-memories, respectively. UADR and LADR provide sequence numbers, explaining the address control scheme for each memory. Fig.8(b) shows the output tuples and Fig.8(c) illustrates the execution process. The processing algorithm of the JOIN-EQ command is as follows:

```
If A1>B1 then UADR=UADR and LADR=LADR+1
If A1<B1 then UADR=UADR+1 and LADR=LADR
If A1=B1 then output a matched tuple pair
 and
 if DP of A1 and DP of B1 are on
  then UADR=UADR and LADR= LADR+ 1
 if DP of A1 is on and DP of B1 is off
  then UADR=UADR+1 and LADR=LADR*
 if DP of A1 is off and DP of B1 is on
  then UADR=UADR and LADR=LADR+1
 if DP of A1 and DP of B1 are off
  then UADR=UADR+1 and LADR=LADR+1
```

Here, DP stands for the duplication line and LADR* points to the first tuple of those which have the same values. Adopting this algorithm, the merger is able to perform the JOIN-EQ command on the attributes having duplicate values efficiently.

The operation of the merger is divided into three cycles. These are the four-byte read cycle, compare-transfer cycle and the ROM table read cycle. Each cycle takes 220 nsec and is synchronized with the sorter.

3.5 Data Processing by the CPU

Since the operations performed by the sorter and merger are limited to the intertuple comparison concerning one field (typically one attribute) for each relation, the other operations must be performed by the CPU. These are as follows:

(1) Selection under complex conditions
(2) Arithmetic operations
(3) Aggregate operations

In order to improve the performance, the RDBE has a compiler which generates the native machine instructions into the main memory. The instructions are executed on the tuples generated by the merger and stored into its main memory by the HM adapter (OUT). The result is sent to HM through the HM adapter (OUT).

Since the data processing using its CPU can be overlapped with the sorter and merger operation, a combined operation is able to be performed in one shot. The following is an exemplified query;

```
SELECT *
FROM A, B
WHERE a1=b1 AND a2>b2
```

a join operation with a conjunctive condition. The RDBE is able to perform the join operation in one shot; the equal condition, using the sorter and merger, and the other, using its CPU.

3.6 Control Mechanisms

The RDBE's control mechanisms of the modules, in order to perform an RDBE command, are described in this section.

Since the sorter and merger have limited capacity; i.e. the maximum amount of data which the sorter and merger are able to process in one scan, the CPU controls the modules in a different way. This depends on the category of the RDBE command and the amount of data. They are as follows:

(1) Unary intratuple operation
Arithmetic operations and selection are involved in this category. When the amount of data is so huge that the modules (including the capacity of the CPU's main memory) is not able to process it in one shot, the CPU controls

the modules for each nonintersecting portion (called a substream) of the original data repeatedly.

(2) Sort-type operation

When the amount of data is small enough for the sorter to process in one shot, the CPU indicates the sorter to sort it and the merger to pass it. When the data is not greater than twice the sorter's capacity, the CPU first indicates the sorter to sort one half of it and the merger to store it into its U-memory. Then it indicates the sorter to sort the rest and the merger to merge them. Otherwise, the CPU first controls the modules as in the second case, to generate two partially sorted sequences. Then the CPU indicates the merger to merge them repeatedly. Since each step is based on a two-way merge operation, it becomes inefficient when the amount of data becomes large, in comparison with the multi-way merger (Dohi 83).

(3) Binary operation (type 1)

In a join-like operation, the CPU controls the modules to perform the operation for each combination of the substreams of the relations repeatedly. An alternative is to sort each tuple first and to process the JOIN command on them. However, this is not adopted for two reasons; (1) it takes more time when the amount of data is not eight times greater than the sorter's capacity, and (2) It does not work well when a large number of values are duplicated.

(4) Binary operation (type 2)

In a difference-like operation, the CPU controls the modules as follows. Let Ra and Rb be the original relations, and the operation be to get (Ra-Rb). The CPU first indicates the sorter to sort the first substream of Rb (say Rb1) and the merger to store it into its U-memory. Then the CPU indicates the sorter to sort each substream of Ra and the merger to perform the RESRICT-NE operation between each substream of Ra and Rb1 repeatedly. Since the operations described above generates a temporary result of (Ra-Rb1), the CPU repeats the same operations to generate the final result.

Besides the control mechanisms described above, the CPU is able to control the merger to perform different operations on the same data. This is useful in the RDBE's Delete command. In this command, the RDBE deletes those tuples the key of which match any of the condition values. First, the CPU controls the sorter and merger to store the condition values into the merger's U-memory. Then, for each page, the CPU controls the sorter and merger in two steps; (1) Store all tuples into the merger's L-memory and at the same time, output tuples which unmatch any of the condition values (RESTRICT-NE operation), (2) Output tuples in the L-memory which match one of the condition values (RESTRICT-EQ operation).

Fig.9 illustrates an example. It offloads the HM's tasks necessary to the rollback operation, because the difference between the original page and the deleted page is clear and easy to handle.

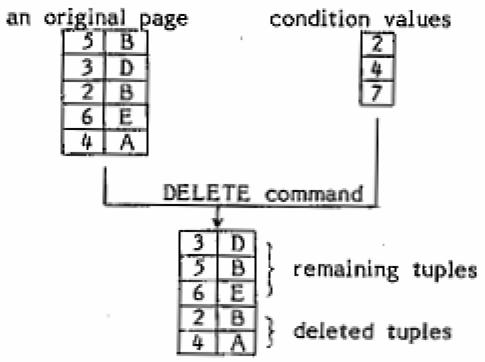


Fig. 9 Example of the DELETE command

3.7 Increasing Reliability

The RDBE has the following features to gain reliability. A parity check mechanism and the sorting checker detect hardware errors with very little increase in processing time. When an error occurs, the CPU resets the active modules and then controls the HM adapters to inform HM to retry the data transfer. HM has only to treat it as an ordinary I/O error.

During the power-up sequence, the CPU controls the modules to perform RDBE operations on certain test data. The main memory, rather than HM, holds the test data, which is provided to the IN module through the HM adapter (IN). The result is stored in the main memory via the HM adapter (OUT) and is checked by the CPU. This mechanism is useful since most of the damage is expected to occur at power-on time.

4 PERFORMANCE ESTIMATION

Since every RDBE module, except the merger, proceeds in a deterministic way, the time required to perform an RDBE operation can be estimated. The activity of the merger, however, depends on the distribution of values in the input tuples, so we present a worst-case estimation of a significant operation, equi-join.

The following list sums up the parameters necessary to estimate performance:

N :	number of sorting cells
M :	merger's U- and L-memory capacity
Cn :	tuple count of the n-th relation
Ln :	tuple length of the n-th relation
Fn :	number of substreams; equal to [(Cn-1)/min(2**N,M/Ln]+1
Snj :	j-th substream of the n-th relation
R :	tuple count of the result
T :	time required to transfer one byte

In a join operation, the CPU controls the modules in the following way:

```
send a request to HM through the HM adapter (IN)
while the first stream is not exhausted
begin
 | get a substream from HM
A| modify it in the IN module
 | sort it in the sorter
 | store it into the U-buffer of the merger
  send a request to HM through the HM adapter
  (IN)
  while the second stream is not exhausted
  begin
 |  get a substream from HM
 |  modify its format in the IN module
B|  sort it in the sorter
 |  compare it with the previous one to generate
 |  the result
  end
end
```

Here, the statements enclosed are executed in parallel.

The total time taken in section A is calculated as follows: The sorter, including the sorting checker, takes $(2S1iL1+L+N-1)T$ for the i-th substream of the first relation. The IN module takes an extra time of $L1T$ and the merger takes an extra time of T. The total time is equal to the following expression:

$\sum(2S1iL1+N+2L1) = 2C1L1+F1(N+2L1)$

The total time taken in section B is calculated in a similar way. The merger, however, takes an extra time of $((S1i+S2j)\max(L1+L2)-S2jL2)T$ for scanning the i-th substream of the first relation and the j-th substream of the second relation. In addition, generating a tuple as a result takes $(L1+L2)T$.

After all, the equi-join operation takes the following time:

$(2C1L1+F1(N+2L1))T +$
$F1(C2L2+(C1+C2)\max(L1+L2)+F2(N+2L2))T +$
$R(L1+L2)T$

When C1 and C2 are equal to 4096, and L1 and L2 are equal to 16 bytes, and the number of the resulting tuples are small, the RDBE takes approximately 128 milliseconds. However, this estimation is unfair because it ignores the following process; accessing the directory, concurrency control, staging the relations into the HM's semiconductor memory from its moving head disks and so on.

5. Conclusions and future plans

We have described the design considerations and implementation of the RDBE. The RDBE was implemented to accommodate it to practical use in the relational database machine Delta.

The RDBE is presently (Aug. 1984) incorporated in the Delta system and is undergoing a system test. At the end of this year, the sorting cell, excluding its memory will be newly implemented using gate-array technology, in order to reduce the volume of the RDBE.

ACKNOWLEDGEMENTS

The present research effort is part of a major research and development project of the fifth generation computer, conducted under a program set up by the Ministry of International Trade and Industry's Agency.

We would like to express our sincere appreciation to the members of the ICOT KBM group for their valuable discussions, and to Dr. K. Mori, director of the TOSHIBA Information Systems Laboratory, who provided the opportunity to conduct the present research.

We would like to thank the TOSHIBA development group, in particular K. Oda, T. Oka and S. Matsuda for their cooperation in designing and implementing the RDBE.

REFERENCES

Boral, H., et al Implementation of the Database Machine DIRECT, IEEE Trans., Software Eng., Vol. SE-8, No.6, Nov., 1982.

Codd, E.F. A Relational Model of Data for Large Shared Data Banks, CACM, Vol.13, No.6, June, 1970.

Dohi, Y., et al. Sorter using PSC Linear Array, International Symposium on VLSI Technology, Systems and Applications, pp.255-259.

Gallaire, H., and Minker, J. (eds) Logic and Data Bases, Plenum Press, 1978.

Hell, W. RDBM-A Relational Database Machine Architecture and Hardware Design, Proc. 6th Workshop on Comp. Arch. for Non-Numerical Processing, Hyeres, France, June, 1981.

Knuth, D.E. The Art of Computer Programming, Vol. 3 / Sorting and Searching, Addition-Wesley, 1973.

Shibayama, S., et al. A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor, ICOT Technical Report, TR-055, 1984.

Tanaka, Y., et al Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer", IFIP 80, pp.427-432, 1980.

Todd, S. Algorithm and Hardware for a Merge Sort Using Multiple Processors, IBM J. RES. DEVELOP., Vol.22, No.5, Sept., 1978.