1987

DESIGN OF AN INFORMATION SYSTEM USING A HISTORICAL DATABASE MANAGEMENT SYSTEM*

N. L. Sarda
University of New Brunswick

DESIGN OF AN INFORMATION SYSTEM USING A HISTORICAL DATABASE MANAGEMENT SYSTEM*

N. L. Sarda
Division of Maths, Engineering and Computer Science
University of New Brunswick

ABSTRACT

A historical database management system (HDBMS) provides facilities for modelling of time and storage and retrieval of history data (i.e., past status). Past research on HDBMS has considered various but individual aspects of the problem and suggested many alternatives. In this paper, we first outline our approach, identifying facilities that are both adequate and practical, and then consider design of a realistic application using those facilities. The application design consists of design of the database schema, storage structures and processing tasks (both query and database maintenance types). The salient features of our approach include schema design with only "current state" perspective, same schema for accessing current and history data, efficient storage structures for history data, and simple extensions to the popular query language SQL.

*This research was supported by a grant from the Natural Sciences and Engineering Research Council

1. INTRODUCTION

Time is an important dimension in all human activities. Events and actions occur continuously over time, modifying current status and generating history. If databases are to model the real world, a database management system (DBMS) must provide concepts and facilities to model time and manage history data. We refer to such a DBMS as an Historical DBMS (HDBMS).

Realizing this need, many recent research efforts have been directed towards studying various aspects of time modelling in database systems. They include formulation of a semantic model (Clifford and Warren 1983) based on "state" concepts, definition of relational algebra operations (Clifford and Tansel 1985) where attributes are stamped with time instants or periods, and techniques for efficient storage and retrieval of data from database containing time information. Snodgrass and Ahn (1985, 1986) made a clear distinction between two time measures: real-world time and (computer) system time. They identified four classes of DBMS based on which (including both) time measure is incorporated in HDBMS.

In our earlier paper (Sarda 1987), we identified main issues which need to be answered satisfactorily. We outlined our approach to HDBMS characterized by:

i) concept of "state" of an entity/relationship; state prevails over a period of time,

ii) real-world time measure (certainly more important than system time as the computer system is merely used as a tool),

iii) separation of history data from current data for efficient storage and retrieval; the separation is mostly transparent to the database users. An added advantage is that the database designer needs to model application requirements based only on "current" perspective,

iv) selectable time granularity, and

v) extension of established query languages such as SQL (Chamberlin et al. 1976) for definition, manipulation and control of data.

Our approach, which is a direct extension of the standard relational data model, is presented more formally elsewhere (Sarda 1987).

The literature pertaining to this area of research is devoted mainly to individual aspects and identification of approaches. As yet, no documented effort has been made to consider a real-life application and model it in all completeness using proposed facilities of HDBMS. Our objective in this paper is to consider practical requirements of a real application and use HDBMS to model those requirements. Thus, the focus of this paper is on a case study. The primary motivation for this exercise is to identify basic but adequate HDBMS support that is efficient and easy to implement.

The paper is organized as follows: In Section 2, we outline the facilities of a proposed HDBMS; Section 3 gives an overview of requirements for a real-world order-processing system. These requirements are analyzed and used to define database schema in Section 4. It also identifies storage structures maintained by HDBMS and access paths desired by the designer. In Section 5, we consider important data processing tasks for the order-processing example and show how they can be programmed using the extended query language. These tasks bring out use of both the current and history data using HDBMS facilities. Finally, Section 6 contains a summary as well as advantages of our approach.

2. OVERVIEW OF HDBMS FACILITIES

2.1 Time Modelling

Our HDBMS (Sarda 1987) is based on the relational data model where a relation scheme defines a real-world entity or relationship type, and where a tuple contains data about one entity or relationship (including the unique identifier or key). In HDBMS, a tuple represents *state* of an entity/relationship during a certain period of time. The time as well as state transitions are automatically managed by HDBMS.

For example, consider the following relation:

BOOK (ISBN, TITLE, PRICE).

With HDBMS, the BOOK relation can contain data of the following kind:

i) The book titled "COBOL PROGRAMMING" with ISBN 07-010125-4 *has* price $25 since March '87 (CURRENT data).

ii) The same book had price $20 from Jan '86 up to March '87 (HISTORY data).

iii) Its price will be (revised to) $30 from Jan '88 (FUTURE data).

The database in HDBMS is segmented into three parts. The current status data are stored in the CURRENT database, the past status data are stored in the HISTORY database, and the data to become effective at some future time are stored in the FUTURE database. *Same* schema describes each database. This segmentation of database is largely transparent to database users. HDBMS selects appropriate segment(s) from the time specification given in the user's query. However, if required and whenever known, users can select a segment for relation BOOK by specifying CURRENT (BOOK), HISTORY (BOOK), or FUTURE (BOOK).

HDBMS maintains a clock of fine granularity. Every "tick" of this clock represents a *time instant*. The value of time instant has the form:

<date> <hour> <minutes> <seconds>

where <date> itself is represented as <day> <month> <year> (e.g., Feb. 1, 1987 as 010287). The current time is denoted by NOW, which can be thought as a "moving" time variable (as in Clifford and Warren 1983).

The designer selects a granularity depending on the needs of application. The granularity may be different for different relations in the database. The chosen granularity is indicated by specifying one or more factors from the above format of time instant; for example, granularity <date> may be selected for relation R and (<date> <hour>) for S.

A *period* is represented by its two boundary instants as $t_1..t_2$ where $t_1<t_2$ (note: the set of all time instants is linearly ordered).

A tuple in the HISTORY database is associated with a period $t_1..t_2$, where $t_2<$NOW. It represents a past state of the tuple which prevailed (i.e. was current) during period $t_1..t_2$. There may be many history states (thus, tuples) for an entity/relationship; they all have the same key value.

A tuple in the FUTURE database has an instant t_4 (>NOW) associated with it. It gives the time when the tuple should be made effective by HDBMS.

Making future data effective corresponds to either the INSERT or UPDATE operation. Each tuple in the FUTURE database can be treated as defining a trigger to be activated at a given time by HDBMS.

Usually, only the current database is updated by database transactions. Update and delete operations produce new history tuples. Insertion (with a new key value) does not affect the history database.

HDBMS provides ways for suppressing history, because every change may not be of interest to the application. The designer may specify maintenance of history (and also future data) for selected relations only. Also, generation of history may be suppressed during certain transactions by adding a SUPPRESS HISTORY clause to UPDATE and DELETE commands (e.g., orders cancelled before any action is taken on them may be DELETEd with SUPPRESS HISTORY).

HDBMS provides facilities for retrospective changes (used mostly for error corrections), although they should be used with caution. A retrospective change may modify the current and many history tuples. An algorithm for retrospective change was outlined in Sarda (1987). It should be noted that a retrospective change overwrites earlier history.

2.2 Query Language

We basically extend SQL (Chamberlin et al. 1976) to permit specifications for time-related aspects. Specifically, the extensions pertain to

i) specification of granularity and of relations for which history or future data are to be maintained; these extensions are used in schema definition,

ii) inclusion of many time-related functions and operations for use in queries, and

iii) a clause by which history can be suppressed during database maintenance operations (update, delete) and the clause by which time contexts can be given for database operations.

The other facilities of SQL can be used without change. For example, the facilities for access control, view definitions and physical storage structures (called images and links) can be used for relations in both the CURRENT and HISTORY databases.

Some extensions are of very simple form. Their formats as given in Table 1 are self-explanatory.

Extended-SQL includes many time-related functions and operations. In the following list, r stands for tuple variable, p's for periods, and t's for time instants.

PERIOD(r) : gives the pair of time-instants representing the period value in r.

START(r) : gives starting time instant of period in r.

END(r) : gives ending time instant of period in r.

level-1(t) : level-1 is a time granularity level such as DATE, MONTH, HOUR, etc. This function extracts the value of corresponding level from time instant t.

Table 1. Extensions of Simple Form

TIME GRANULARITY IS instant-type FOR { ALL | relation-name-list }

KEEP { HISTORY | FUTURE } FOR { ALL | relation-name-list }

The phrase **SUPPRESS HISTORY** may be used with the UPDATE and DELETE command.

The phrase **FROMTIME instant$_1$ [TO instant$_2$]** may be used with any SQL database operation (i.e., query as well as database maintenance). In query operations, projects involved relations to a given period. It may also be used to supply future data or make a retrospective insertion/change.

level-2(t_1,t_2): this function is used to obtain elapsed time in units given by level-2 between time instants t_1 and t_2. Level-2 may be DAYS, MONTHS, etc.

t WITHIN p : is true if time instant t falls within period p.

p_1 OVERLAPS p_2: is true if p_1 and p_2 have some common time instants.

p_1 INCLUDES p_2: is true if period p_2 is wholly contained in period p_1.

Besides these, we can also use the conventional comparison operators (=,<,>, etc.) on time instants.

In addition to the above extensions, we provide for a modified projection operation that is embodied in the SELECT clause, and the concurrent Cartesian product operation that is specified through the extended FROM clause of SQL. These extensions are elaborated below.

i) The SELECT clause of standard SQL specifies projection on attributes given in SELECT. In extended-SQL, time-periods are automatically projected. The resulting tuples are compacted on time before removing duplicates. The compaction is carried out by merging periods of those tuples which have the same attribute values but overlapping or consecutive time periods.

ii) In standard SQL, a Cartesian product is implied between the relations given in the FROM clause. In extended SQL, the word CONCURRENT may optionally be given in the FROM clause as follows:

FROM CONCURRENT relation-list

The concurrent product between relations R and S is defined as follows: a tuple r ϵ R is combined with s ϵ S only if time periods of r and s overlap. In that case, the resulting tuple contains a period which is common to both r and s. The concurrent product is a very useful operation as it allows us to relate data from different relations on the basis of their currency in time.

3. REQUIREMENTS ANALYSIS

We have chosen the order processing system for a large book publishing company as an example. Orders are received from customers, giving quantities for one or more required books. Orders may be prepaid partially or fully. There may be one or more dispatches per order (depending on availability). One invoice is prepared for each dispatch. The company permits payments in installments (thus, there may be one or more payments per invoice). A discount of 10% is given for payments received within 30 days of dispatch. However, interest at 18% is charged for payments pending over 60 days. Orders may be cancelled in part or full (provided dispatches have not been made or have been returned with cancellations). Books damaged in transit may be returned (payments refunded, if necessary) or replaced.

The company holds prices for a period of one quarter. Prices may be revised on the first day of each quarter (starting January 1). Customers are charged by prices prevailing on the day of dispatch.

We have to analyze the above requirements scenario to identify data of "current operational interest" at any point in time. This "current view" will be used by the database designer to define the database schema. Since every historical fact may not be of interest to the company, the designer should also identify historical data of interest and those transactions which produce history data. Attention also needs to be given to the nature of "future" data that might be of importance to this application. Finally, the designer has to select time granularity.

3.1 Current Data

The customer orders which are pending, partially dispatched or partially paid would be considered to be current. Payments and invoices are current if the corresponding order is current. The current data about a book includes its title, author, price, quantity-in-stock and ISBN. The books already published are current (but not the discontinued titles). The current data about customers includes name, address, etc.

3.2 History Data

History data contains data about orders fully processed and paid. Data about dispatches and

payments for all completed orders are of interest to the company (for, say, a period of one previous year; history older than that can be "purged"--this aspect is not dealt with any further). The books discontinued are also of historical interest. Price changes must be noted (necessary for calculation of refunds or returns, etc.). However, changes in stock levels are rapid and are not of interest to the company. Similarly, changes to customer data, such as change of address, are also of no historical interest.

3.3 Future Data

Prices of books for subsequent quarter(s) are often decided in advance. This data must be accepted and enforced by the computerized system. Also, the company announces new titles in advance and may receive orders for them. The orders must be processed as soon as the stocks are available. The system should handle these future data. The concept of "future" is not applicable to customer data for this company.

3.4 Time Granularity

All operational data is timed on the basis of dates. Thus, date (day/month/year) is the required time granularity.

The E-R (entity-relationship model) diagram depicting the current data and their inter-relationships is given in Figure 1.

4. MODELLING USING HDBMS

4.1 Schema Design

To design the application using HDBMS facilities, we specify

i) the database schema for the current database,

ii) time granularity (which may be the same for all relations or different for different relations),

iii) relations for which history is to be maintained, and

iv) relations for which future data needs to be stored.

The relational schema for the current database is easily obtained from the E-R diagram in Figure 1. It contains the following relations:

ORDER (O#, C#).
CUSTOMER (C#, ADDRESS)
BOOK (ISBN,...,PRICE, QOH)
DISPATCH (INV#, O#,COST,...)
ORBK (O#,ISBN,ORQNT)
DSBK (INV#, ISBN, DSQNT)
PAYMENT (INV#, AMOUNT, CODE, ...)

ORQNT and DSQNT give ordered and dispatched quantities, respectively. CODE in PAYMENT will be used to show refunds made to the customer for books returned.

Granularity specification is made by a suitable statement such as

TIME GRANULARITY IS DATE FOR ALL

The relations for which history should be maintained are identified by the following simple specification:

KEEP HISTORY FOR ORDER, BOOK, DISPATCH, ORBK, DSBK, PAYMENT

Finally, we indicate to HDBMS the relations for which future data needs to be recorded:

KEEP FUTURE FOR ORDER, BOOK

It may recalled from Section 3 that future data for BOOK will contain planned price revisions and future titles. Future data for ORDER are orders for future titles.

HDBMS processes the above specifications to define additional attributes (for storing time stamps) and relations (for storing history and future data). Here, six relations for storing history and two for storing future data would be defined by HDBMS. HDBMS permits us to access these relations by generic reference: history of R as HISTORY(R) and FUTURE of R as FUTURE(R).

4.2 Storage Structures

For any application, HDBMS maintains the current history and future data in three independent segments. It is possible to define independent storage structures for relations in each segment so

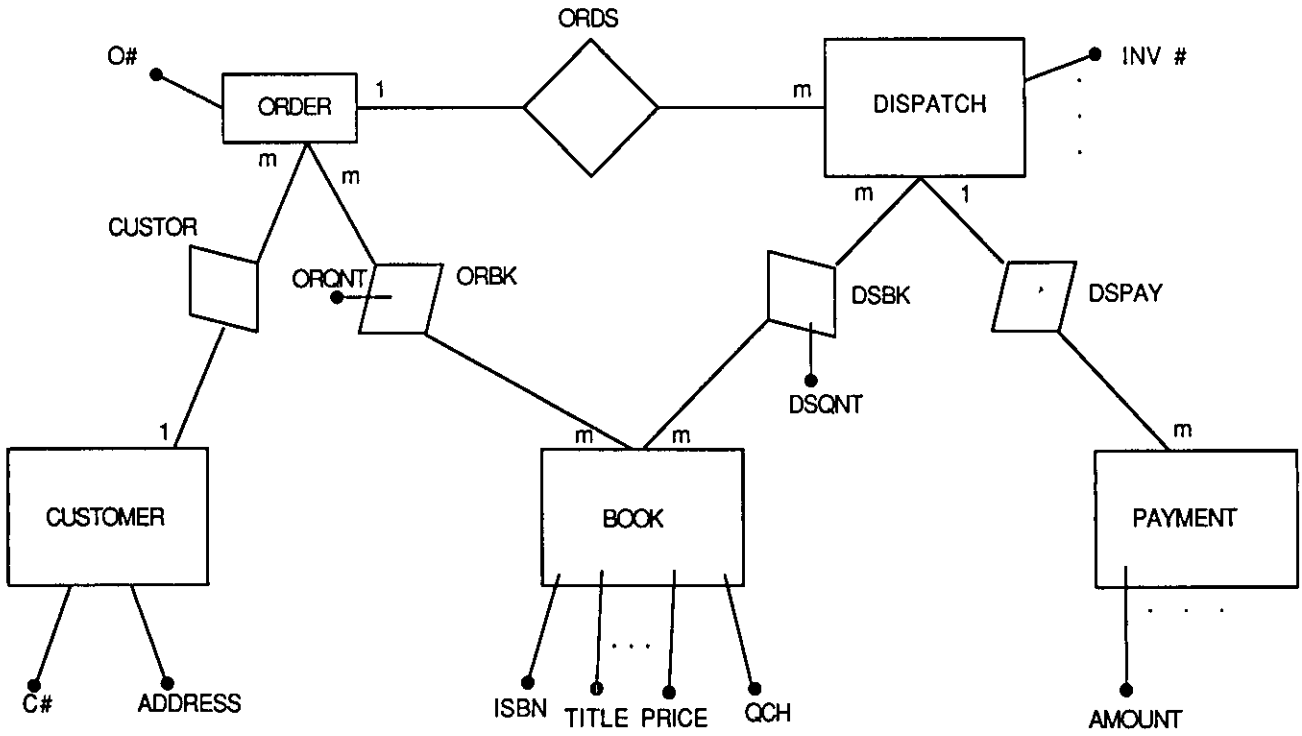

Figure 1. The E-R Diagram for Order Processing System. Relationship-types (1-m, m-m; many-to-many) are also indicated. Only main attributes are shown.

that access to them is efficient. Before deciding access paths, the designer should know that the following internal structures exist by default:

i) The history data by its very nature is ordered chronologically. Every tuple in a history relation carries a time-stamp giving START and END time-instants of the period in which the tuple data was current. The tuples are ordered automatically in increasing sequence of END values. HDBMS uses this built-in order to efficiently locate tuples for a particular period (e.g., binary search method). Because of the inherent sequencing and permanent nature of history data, it is cost-effective to maintain multi-level sparse index on time.

ii) All history tuples pertaining to the same entity or relationship are automatically linked by HDBMS in the reverse chronological sequence. Thus, it is easier for HDBMS to locate all historical states of a given tuple. Moreover, each current tuple contains a pointer to its latest history tuple. HDBMS, thus, has a path from current database to history database. This path does not, however, exist for tuples deleted from the current database. HDBMS keeps a separate index for deleted keys for easy access to their histories. This index also serves HDBMS in ensuring that key values remain unique over the entire life-time of the application.

The following additional access mechanisms may be defined by the designer, keeping in mind predominant usage of database. They are specified in SQL-like form (Chamberlin et al. 1976):

```
CREATE IMAGE I1 ON BOOK (ISBN);
    /* index to BOOK on ISBN */
CREATE IMAGE I2 ON CUSTOMER (C#);
    /* index to CUSTOMER on C# */
CREATE CLUSTERING LINK L1 FROM ORDER
  (O#) TO DISPATCH (O#);
```

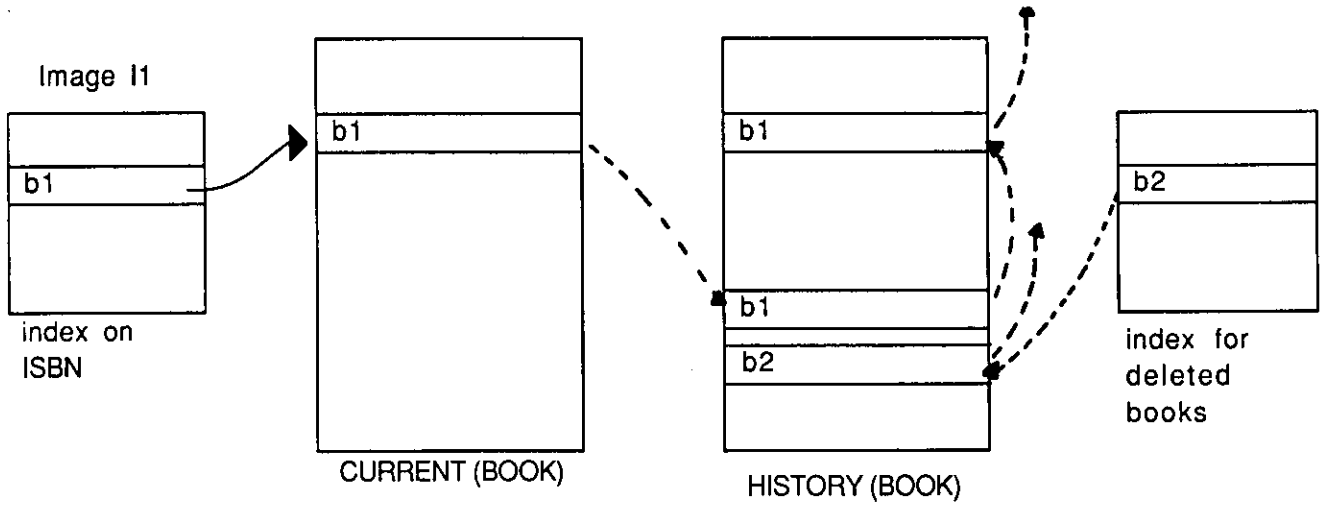

Figure 2. Schematic for Storage of Relations. Broken arrows represent links maintained by HDBMS.

CREATE LINK L2 FROM DISPATCH (INV#) TO DSBK (INV#);

(Note: links basically facilitate the join operation.)

These access paths are created for relations in the current database. Similar statements can be given for creating access paths on history and future relations where relations are specified as HISTORY(R) or FUTURE(R).

Figure 2 shows a schematic of database storage for the BOOK relation. Note that index I1 also acts as the index to history data because of links maintained by HDBMS. Further, the index on deleted books is defined by HDBMS itself.

5. PROCESSING TASKS FOR THE APPLICATION

Having defined schema and access structures, we now consider both the database maintenance and information retrieval type of transactions for our order processing example. Specifically, we consider the following set of transactions as they exercise various aspects of HDBMS facilities:

i) prepare a dispatch for a given order,

ii) process a payment received from a customer,

iii) process a replacement for books returned in damaged condition,

iv) list partially-completed and more than one month old orders,

v) obtain monthly turnover statistics, and

vi) enter future data for price revisions and forthcoming books.

We assume that SQL-like stand-alone as well as higher-level language (e.g., PL/I embedded) interface is provided by HDBMS. However, we will give only database interactions in extended SQL for the above transactions.

i) *Prepare Dispatch*

To prepare a dispatch for an order, we will need to list books ordered along with availability data so that decisions can be made regarding quantities to be dispatched. Only the current database needs to be accessed as follows:

```
SELECT X.ISBN, ORQNT, QOH

FROM   CURRENT (ORBK) X,
       CURRENT (BOOK) Y

WHERE
       X.O# = GIVEN and
       X.ISBN = Y.ISBN
```

(Note: X and Y are tuple variables.)

Based on this data, the dispatch clerk decides on quantities for dispatch and enters data into the database by executing

INSERT INTO DSBK: <INVNO, ISBN, SELQNT> after giving a suitable invoice number to the dispatch. A tuple should also be added to the DISPATCH relation. These SQL commands can be embedded in a suitably-designed PL/I program. The only additional role played by HDBMS in this transaction is to time-stamp inserted tuples.

ii) *Payment Processing*

The primary information required for processing a payment is to determine the number of elapsed days between dispatch and payment (and thereby determine whether it qualifies for a discount or interest charges). The invoice number is known from payment. The following SQL query can be embedded in the payment processing program:

```
SELECT COST, DAYS(START(X),NOW)
FROM   CURRENT(DISPATCH) X
WHERE  X.INV# = GIVEN
```

An index on INV# could be defined to make the above query more efficient. The program should next compute interest/rebate and determine outstanding amount, if any. It may delete order, dispatch and payment tuples if the order is completely processed.

iii) *Replacement Processing*

A dispatch (its INV# known) may be partially or fully returned as damaged. Quantities in original dispatch must be modified (without history) and a new dispatch should be prepared. We will need to execute the following command in replacement processing program:

```
UPDATE   DSBK
SET   DSQNT = DSQNT-RETURNEDQNT
WHERE
      INV# = GIVEN and
      ISBN = RETURNEDBOOK
WITHOUT HISTORY
```

The history of returned books is of no interest to the publisher. However, in realistic design, we will need to provide for "writing off" damaged lots.

If the order was pre-paid, the payment should be carried over to the next dispatch by altering old payment data (without history) and inserting a new tuple for new dispatch. It is quite straight forward to the program for these requirements. Note that UPDATE is applied to the current database by default.

iv) *Partial-orders Listing*

To list orders not fully processed and to also obtain their "age" and outstanding qualities, we may query the current database as follows:

a) as there may be many dispatches for the same order, we first obtain total dispatches made:

```
ASSIGN TO TEMP(O#, ISBN,Q)
   SELECT  O#,Y.ISBN,SUM(DSQNT)
   FROM    CURRENT(DISPATCH)
           CURRENT(DSBK)  Y
   WHERE
           X.INV# = Y.INV#
   GROUP BY O#,ISBN
```

b) subtract dispatched quantities from ordered quantities (in relation ORBK) to find outstanding quantities (SQL query is straightforward for this step), and

c) to list orders more than one month old, we can query as follows

```
SELECT      O#
FROM        CURRENT(ORDER) X
WHERE
            DAYS(START(X),NOW) > 30
```

v) *Turnover Listing*

To list monthly turnover from Jan. 1985 onwards, we use data in DISPATCH (both current and history) as follows:

```
SELECT    MONTH(START(X)),SUM(COST)
FROM      DISPATCH X
WHERE
          START(X) > 010185
GROUP BY
          MONTH(START(X))
```

vi) *Entering Future Data*

a) Price of a book (ISBN known) is to change from July 1, 1987: in extended-SQL, we specify

```
FROMTIME 010787
INSERT INTO BOOK:
    <KNOWN-ISBN,,,NEWPRICE,..>
```

Unspecified fields are taken to mean that their values would not change. HDBMS sets a trigger to be activated on July 1, 1987.

b) Release of a new title (new ISBN value) in August 1987: We add a tuple to the future BOOK relation using

```
FROMTIME 010887
INSERT INTO BOOK: <new-book-data>
```

It is processed by HDBMS in a similar fashion as above.

vii) *Miscellaneous Queries*

We consider more examples to illustrate use of the history data.

a) Obtain price of a given book on a given date; history may have to be accessed if there was a price change. The query can be formulated as

```
SELECT    X.PRICE
FROM      BOOK X
WHERE     X.ISBN = GIVENISBN and
          GIVEN DATE WITHIN PERIOD(X)
```

Note that this query is efficiently processed due to the selected access mechanisms.

b) List titles of 1986 which have now been discontinued:

```
SELECT ISBN,TITLE
FROM   HISTORY(BOOK) X
WHERE
     PERIOD(X) OVERLAPS <010186..311286>
     and ISBN NOT IN
          (SELECT ISBN
          FROM CURRENT(BOOK))
```

c) List current books of 1987 having total dispatches less than $10,000.

We have to calculate the value of a dispatch by using prices prevailing at that time. We need to access both the current and history data of BOOK and DSBK relations.

```
FROMTIME010187 TO NOW
SELECT    X.ISBN, SUM(DSQNT*PRICE)
FROM CONCURRENT BOOK X,
                DSBK Y
WHERE
          X.ISBN = Y.ISBN and
          X.ISBN in (SELECT ISBN FROM
            CURRENT(BOOK))
GROUP BY X.ISBN
HAVING SUM(DSQNT*PRICE) < 10000
```

Query basically joins tuples of BOOK and DSBK on ISBN and matching time periods; the joined result is grouped on ISBN and the cost of dispatch is computed.

6. DISCUSSION AND CONCLUSIONS

The time dimension is important in all human endeavors. Our objective in this paper has been to identify basic but useful support to be provided by a database management system for handling of the time dimension and history data in business applications. Instead of being theoretical or excessively ambitious we wish that the support be without excessive penalty in storage and access costs. We have considered a fairly realistic example to identify the kind of support needed. A number of useful ideas and techniques have emerged as a result of this exercise:

i) System time (maintained by HDBMS) matches real time in most cases when transactions are processed on-line. When they do not, real-time is supplied as part of transaction data/query (e.g., the FROMTIME phrase in the beginning of INSERT or UPDATE statements). Thus, it is sufficient to keep only one time measure.

ii) The state concept seems natural and efficient for history purposes instead of the attribute stamping (Clifford and Tansel 1985) approach which makes tuples unnormalized and of varying (and growing) lengths. It is difficult to manage storage and create access paths for such data.

iii) Not every change would be of interest to an organization. Controlled generation of history data must be provided by HDBMS. This can be achieved at two levels: identify relations for which history should be maintained and provide an optional clause to suppress

generation of history in data manipulation commands.

iv) Separation of data into current and history databases is an important design decision. First, since the same schema describes both databases, the designer needs to formulate database schema only for the current view of data and relationships. The current view does not include history. This approach could reduce schema complexity (Sarda 1984; Sarda 1987). Second, the separation enables us to define independent access mechanisms for history and current data so that their different usage patterns can be taken into account.

v) Certain access structures for history data have been identified. Reverse chronological linked lists can be easily maintained; more over, access paths to current data also facilitate access to their history.

vi) Conventional query languages can be easily extended for access to history (as well as future) data. We need a few time-oriented functions and operations. The reference to history data by the specification HISTORY(R) is primarily intended to narrow down the search quickly when the user is aware of the location of required data.

vii) Out-of-sequence (or retrospective) changes would be rare and mostly for error correction. A need for facilitating this is pointed out in Snodgrass and Ahn (1985, 1986). In our approach, where only one time scale is maintained, a retrospective change overwrites earlier status. We consider this a particularly satisfactory approach because a retrospective change in real-life is never an isolated event. Its processing needs careful thought and possible changes to many related data. It is not sufficient to let HDBMS merely record the retrospective change. To illustrate, consider book b_1 with current price \$35 given by the current tuple

<b_1,.., 35, 010486..NOW>

Assume that b_1 has following 2 history tuples:

<b_1,.., 25, 010985..010186>
<b_1,.., 40, 010186..010486>

To change price to \$45 retrospectively from Feb. 1, 1986, we execute

```
FROMTIME 010286
UPDATE BOOK
SET PRICE = 45
WHERE ISBN = b1
```

When HDBMS executes this, the tuples are affected as follows:

current: <b_1,..,45,010286..NOW>
history: <b_1,..,25,010985..010186>
<b_1,..,40,010186..010286>

This change is effected automatically and permanently (old history is lost). But this change is not sufficient by itself. We must consider consequences of this update in the real world: we must change invoice values for dispatches and even "open up" completed orders. A carefully designed program has to establish these consequences. Such a program should also record "old history," if necessary. The alternative (called "Temporal" database in Snodgrass and Ahn 1986) seems too costly for practical design.

Having identified practical the support required and efficient techniques to provide that, some extensions can be incorporated as suggested below:

i) The designer may specify certain relations to be of the "EVENT" type. Events have a single time instant of interest (i.e., time of "arrival"). For such relations, HDBMS can make START and END times equal when their tuples are put into history.

ii) HDBMS may maintain some control information in history tuples to indicate fields changed, terminal/user identifier etc.

In this paper, we have presented a case study in the design of an application system using the facilities of an HDBMS. Our HDBMS is based on a state-oriented historical relational data model that is presented more formally elsewhere (Sarda 1987). Our objective in this paper has been to demonstrate naturalness and effectiveness of the proposed facilities through a complete and realistic example, as the literature on time modelling lacks such a study. Although a detailed comparison is out of scope, we conclude this paper by comparing our

historical data model with those proposed by other researchers.

A state-oriented approach with a rigorous semantic basis based on intentional-logic was proposed by Clifford and Warren (1976). The model, however, lacked practical considerations. A different formalism, along with a new relational algebra, was defined later by Clifford and Tansel (1985). In the two views presented by Clifford and Tansel, attributes are time-stamped and a large number of concepts and operations are defined. Snodgrass and Ahn (1985, 1986) define two measures of time and four types of databases using a "cubic" view in which time is added as a third dimension to the flat (conventional) relation. They also suggest a few (basically of the "time-slicing" kind) extensions to the query language, QUEL. The cube-oriented conceptualization is also proposed by Ariav (1986). The projection and selection operations are provided. Some of the extensions do not fit into the simple product-select-project framework of SQL, and, hence, are difficult to comprehend.

The model we proposed elsewhere (Sarda 1987) maintains conformity with the standard relational model and our extensions to SQL retain its simple framework. Also, as the present case study demonstrates, the proposed facilities are effective and efficient.

REFERENCES

Ariav, G. "A Temporally-Oriented Data Model." *ACM Transactions on Database Systems*, Vol. 11, No. 4, December 1986, pp. 499-527.

Chamberlin, D. D., Asdrahan, M. M., Eswaran, K. P., Griffiths, P. P., Lorie, R. A., Mehl, J. W., Reisner, P., and Wade, B. W. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control." *IBM Journal of Research and Development*, Vol. 20, No. 6, 1976, pp. 560-575.

Clifford, J. and Tansel, A. V. "On an Algebra for Historical Relational Databases: Two Views." *Proceedings of ACM SIGMOD*, 1985, pp. 247-265.

Clifford, J. and Warren, D. S. "Formal Semantics for Time in Databases." *ACM Transactions on Database Systems*, Vol. 6, No. 2, 1983, pp. 214-254.

Lum, V., Dadam, P., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H., and Woodfill, J. "Designing DBMS Support for the Temporal Dimensions." *Proceedings of ACM SIGMOD*, 1984, pp. 115-130.

Sarda, N. L. "Handling of Time in Database Systems." *Proceedings of the International Computer Symposium*, Taiwan (ROC), 1984, pp. 603-610.

Sarda, N. L. "Modelling of Time and History Data in Database Systems." *Proceedings of the CIPS Congress '87*, Winnipeg, May 12-14, 1987, pp. 15-20.

Sarda, N. L. "Algebra and Query Language for a Historical Data Model." Submitted to *The Computer Journal*, 1987.

Snodgrass, R. and Ahn, I. "A Taxonomy of Time in Databases." *Proceedings of ACM SIGMOD*, 1985, pp. 236-246.

Snodgrass, R. and Ahn, I. "Temporal Databases." *IEEE Computer*, September 1986, pp. 35-42.