A Relational Approach to Querying Data Streams

Carl S. Hartzman and Carolyn R. Watters

Abstract—Data streams are long, relatively unstructured sequences of characters that contain information such as electronic mail or a tape backup of various documents and reports created in an office. This paper deals with a conceptual framework, using relational algebra and relational databases, within which data streams may be queried. As information is extracted from the data stream, it is put into a relational database that may be queried in the usual manner. The database schema evolves as the user's knowledge of the content of the data stream changes. Operators are defined in terms of relational algebra that can be used to extract data from a specially defined relation that contains all or part of the data stream. This approach to querying data streams permits the integration of unstructured data with structured data. The operators defined extend the functionality of relational algebra, in much the same way that the join does relative to the basic operators—select, project, union, difference, and Cartesian product.

Index Terms—Data streams, querying text, relational algebra.

I. INTRODUCTION AND BACKGROUND

Introduction

A DATA stream is a sequence of characters or bits that is too large to be viewed in its entirety. Examples are: electronic mail, continuously monitored digitized sound, the results of a database download being read to load a newly created database, and sequential files containing the daily paperwork at an office in no particular order (the disk storing these files may be read sequentially sector by sector).

Data streams in isolation have varying degrees of structure ranging from no apparent structure for digitized images of white noise to strictly periodic patterns for digitized images of the radio emissions of rotating stars.

Most data streams fall between the two extremes. For example, data streams representing electronic mail with standard headers may have certain patterns that always occur in the same order, although these patterns may be separated by strings of varying length with no apparent pattern—i.e., Message: ... From: ... To: ... Re: ... Message: ... From: ... To: ...

In contrast to the various degrees of structure that isolated data streams may exhibit, it should be noted that they contain no meaningful information until they can be placed in a context. It makes no sense to be wondering what mail is arriving from Berkeley if the stream being observed is the digitized image of rotating star radio emissions. Once the data stream is placed in a context, it can be queried, meaning that information can be extracted from it.

This paper shows that, conceptually, information can be extracted from a data stream using only relational algebra, the same framework used for traditional relational databases. Since the information content of a data stream is not organized according to a predefined scheme known in advance of the querying process, extraction of information from a data stream becomes a twofold process: the progressive discovery of the precise information that can be retrieved from the data stream, and the storage of that information according to a well-defined scheme that may then be queried using standard techniques.

The relational model was chosen as the underlying model for this paper as it is the most widely used and best understood data model, having a long history of rigorous development based on sound theoretical principles.

Using a common conceptual framework to implement the twofold process described above has the advantage of offering a unified and consistent view of the information in both the data stream and the database. Such a view is helpful to the user, who will be employing standard database techniques to incrementally build the database as information is discovered in the data stream.

Since querying a data stream is an incremental process and neither the exact nature nor precise structure of the data is known in advance, questions of precision and recall, analogous to those in the field of information retrieval [11], must be considered. These questions are appropriately noted and may form the basis for future work.

The next section, Section II, of the paper deals with the context of the data stream. Section III defines a special purpose relation for managing the data stream. Sections IV, V, VI, VIII, and IX define special purpose operators, or primitives, used to extract information from the special purpose relation, in terms of relational algebra. Section X illustrates how a database can be incrementally built to store data extracted from the data stream as it becomes known. Section VII details an example that motivates much of the development of the paper and Section XI illustrates how "user friendly" query operators may be built upon the primitives, unifying the notion of querying the data stream with querying the database. The conclusion, Section XII, points to questions about the actual implementation data stream querying, not discussed in the paper which may serve as the subject for future work.

In Section III, it will be seen that a portion of the original data stream becomes resident in the database, at least temporarily, and hence, can be queried. The data stream will, however, invariably contain information that for some reason, probably semantic, will not be contained in the database. When additional information is desired, appropriate portions of the data stream may be recalled for further examination. This can be particularly useful in settings, such as office information systems, where the data stream contains documents or other naturally unstructured data for which all queries and required database attributes are not known a priori.

Background

Work on querying data streams, per se, has not previously appeared in the literature. However, there is a body of work in information retrieval and office information systems that may serve as a background against which this work may be considered.
Attempts to deal with text for retrieval purposes, in office information and bibliographic systems, tend either to ignore the existence of attributes within textual units [6] or to impose some structure on the data within the documents (i.e., author, title, abstract, etc.) so that each document can be assigned attributes upon which to base retrieval [2]. The relational model has been used extensively with such semi-structured textual data [2, 4], but presents some real difficulties in both setting up and querying the data [3]. The thorny question of attributes within larger textual units, such as abstracts, has been left largely untouched.

Stonebraker et al. [12] have looked at the usefulness of the relational model for handling unstructured text from a text editor perspective. The textual unit is broken into well-defined editing attributes: document, sentence, word. This breakdown does not, however, help in the formation of queries about the content of the text.

Textual fact retrieval systems capture the semantic content of the unstructured text through linguistic analysis of the text [7], [1]. The linguistic analysis is directly related to the domain of the knowledge base being constructed. In this paper, a system is being built whose elements do not rely on advanced knowledge of the domain of the data stream. This approach permits the handling of short-lived data extracted for temporary use. The system works in cooperation with the user who interactively supplies any necessary domain specific knowledge.

II. THE CONTEXT

Before proceeding, the notion of context must be understood. The discussion in the Introduction suggests that once the source of the data stream is known, the nature of the information it may contain is known. In a database milieu, the only information that may exist is that which can be stored in some data model. If attention is restricted to the relational model, all possible information is synonymous with all possible relational database schemas. Thus, the context of a data stream will be defined as a set of relational database schemas. This set of schemas would reflect the type of information expected from a data stream of a particular nature. Any schema in the context can be thought of as a known context, in the sense that it represents the current state of knowledge about information in the data stream.

Given a particular understanding of the data stream, represented by a relational database schema or known context, one of the first concerns must be how to compute a relational database instance from the data stream, that is, how to extract the appropriate data from the data stream which can be used to assign values to the attributes in the relations in such a way as to form correct records.

This will be done by introducing a special relation to serve as a mediator between the data stream and the rest of the prospective schema. It will be seen that via this mediator, relational algebra can be used to query the data stream.

As more information is acquired about the nature of the data stream, the known context is replaced by different schemas from the context, each representing a refinement in one's knowledge of the data stream.

III. THE MEDIATOR AND EXTRACTING DATA FROM THE DATA STREAM

The mediator is a relation DATASTREAM whose schema is DATASTREAM (ID#, C1, C2, ..., Cn), where n is large and Ci is of type CHAR(1). Conceptually, the records in DATASTREAM can be viewed as snapshots of the data stream of length n. ID# is a key by which a particular snapshot can be identified. In particular, ID# can be taken to be the shift from the start of the data stream to the start of the snapshot. Thus, two records in DATASTREAM whose ID#'s differ by one represent two snapshots of the data stream of length n shifted from each other by one character.

Before developing general techniques for extracting data from the data stream, a very simple example is given to show how the relation DATASTREAM may be used.

Suppose that it is known that the data stream is electronic mail and that the format of messages is known. By way of example, suppose that every new message has a standard header, the first eight characters of which are 'MESSAGE:' followed immediately by a four digit message id. All message id's could be retrieved from DATASTREAM using standard relational algebra, as follows:

\[
\begin{align*}
\pi_{C9,C10}(C11,C12) & \\
& \sigma_{C1='M',C2='E',C3='s,,C~s,,C~s,,C~s,,C~s,,C~s,} (DATASTREAM) \\
& \sigma_{C9,C10}(C11,C12) & \text{applied to the result projects out all the message id's.}
\end{align*}
\]

A much more sophisticated query on the same data stream would be to find topics of all conferences announced. To facilitate the required data extraction, several operators defined in terms of relational algebra are introduced.

One operator, called clustering, which is denoted by C, can be used to group together all the records in DATASTREAM that pertain to the same message. Then when a message is discovered to have the word 'CONFERENCE' (or maybe 'Conference') in it, the records pertaining to the same message can be searched for the topic.

Another useful operator, SHIFT(j), locates the record in DATASTREAM that corresponds to the snapshot of the data stream shifted by j characters from a given snapshot.

IV. THE CLUSTERING OPERATOR

Suppose that M(ID#) is a relation schema whose instantiation is a list of record id numbers selected from DATASTREAM by some criterion. They might be the record id numbers of those records that begin with 'MESSAGE:' as the first eight characters, selected in a fashion similar to the example above. The clustering operator will assign a common identifier CID# to all records in DATASTREAM that begin with the pattern 'MESSAGE:'.

\[
\begin{align*}
\pi_{MI}(DATASTREAM) & = \pi_{CID#}(DATASTREAM) \\
& \sigma_{C1='M',C2='E',C3='s,,C~s,,C~s,,C~s,,C~s,,C~s,} (DATASTREAM) \\
& \sigma_{C9,C10}(C11,C12) & \text{applied to the result projects out all the message id's.}
\end{align*}
\]

A much more sophisticated query on the same data stream would be to find topics of all conferences announced. To facilitate the required data extraction, several operators defined in terms of relational algebra are introduced.

One operator, called clustering, which is denoted by C, can be used to group together all the records in DATASTREAM that pertain to the same message. Then when a message is discovered to have the word 'CONFERENCE' (or maybe 'Conference') in it, the records pertaining to the same message can be searched for the topic.

Another useful operator, SHIFT(j), locates the record in DATASTREAM that corresponds to the snapshot of the data stream shifted by j characters from a given snapshot.

IV. THE CLUSTERING OPERATOR

Suppose that M(ID#) is a relation schema whose instantiation is a list of record id numbers selected from DATASTREAM by some criterion. They might be the record id numbers of those records that begin with 'MESSAGE:' as the first eight characters, selected in a fashion similar to the example above. The clustering operator will assign a common identifier CID# to all records in DATASTREAM between successive id's in M.

Let D = \pi_{MI}(DATASTREAM); then

\[
\begin{align*}
S(CID#, ID#) & = M \ JOIN_{(M.ID# \leq D.ID#)} D \\
& \pi_{1,2,3,4,5,6,7,8,9,10}(M \ JOIN_{(M.ID# \leq D.ID#)} D) \\
& \JOIN_{\{1,2,3,5,6,7,8,9,10\}} (M \ JOIN_{(M.ID# \leq D.ID#)} M)
\end{align*}
\]

where 1 = 3 and 2 x 4 stand for conditions on the first, third, second, and fourth columns of the cross product M x D x M x M, is a relation whose second column contains all the ID#'s in D greater than or equal to the minimum ID# of M and whose first column contains all the ID#'s in M. Records (x,y) satisfy the following property: y = x and for all z' in M greater than x, y < z'. Thus, all the record ID#'s in D falling between two successive (according to sorted order) values of ID#'s in M have been grouped together and assigned the smaller of the two values. The natural join (using the second column) of this relation with
HARTZMAN AND WATTERS: RELATIONAL APPROACH TO QUERYING DATA STREAMS

DATASTREAM and eliminating one copy of the join column results in a relation CDATASTREAM(CID#, ID#, C1, ..., Cn), where CID# is the same for all ID#'s that have been grouped together. CID# will be called the cluster id. For instance, if

<table>
<thead>
<tr>
<th>M: ID#</th>
<th>D: ID#</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

The process just described can be carried out for any relation R, where M now is a subset of key values of R. Just replace D by R and ID#'s by keys. Then, the cluster operator C may be defined by

\[ C_e(R) = S(\text{key1}, \text{key2}) \text{ JOIN}_\text{int} R(\text{key2}, ...) \]

where \( \theta \) is any Boolean condition selecting rows from R, S is defined as above with key values replacing id#'s, and M in the definition of S contains the key values of the records selected by \( \theta \).

The clustering operator may be applied more than once to produce finer clusterings. For example, if the records of CDATASTREAM for electronic mail are grouped together so that each cluster contains only one message announcing conferences, a finer clustering of CDATASTREAM may be made so that each new cluster contains at most one speaker and the title of his talk.

Suppose that M stores id's, selected from DATASTREAM by some criterion, to be used for a clustering. Suppose that M' stores id's from DATASTREAM to be used for a second clustering. Then, a finer clustering than the one induced by M is a clustering of CDATASTREAM using M' \cup \{0\} and the ID#'s in CDATASTREAM (giving CCDATASTREAM). Two tuples in CCDATASTREAM are in the same cluster if they have the same ordered pair of cluster id's. The final result might appear as follows:

<table>
<thead>
<tr>
<th>M: ID#</th>
<th>M': ID#</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

The first clustering produces clusters with ID#'s

(2, 3, 4, 5, 6)
(7, 8, 9, 10)
(11)
(12, 13).

The pair of clusterings produces clusters with ID#'s

(2)
(3)
(4, 5, 6)
(7, 8)
(9, 10)
(11)
(12, 13).

V. THE OPERATOR SHIFT

The operator \( \text{SHIFT}(j) \) allows attention to be focused on records in DATASTREAM \( j \) characters removed from a given set of records.

Given a relation whose keys are \( K_1, K_2, K_3, \ldots, K_n \) in ascending order and a subset \( K_{j'}, K_{j'+1}, \ldots, K_{j''} \) of these keys, \( \text{SHIFT}(j) \) will compute for each \( K_i = K_j \) the key \( K_{j+j'} \). Let the keys of a relation be stored in a relation B(key) and let A(key) store a subset of the keys stored in B. Define

\[ \text{SHIFT}(1)(A) = \pi_2((A \text{ JOIN}_{(1 < 2)} B)) - \pi_1((A \text{ JOIN}_{(1 < 2)} B) \text{ JOIN}_{(2 < 3)} B)) \]

where the join subscripts refer to the column of the Cartesian product from which it is constructed. The following example will demonstrate the computation of \( \text{SHIFT}(1)(A) \).
whence

\[ \text{SHIFT}(1)(A) \]

\[ \text{SHIFT}(j)(A) \]

may now be defined as

\[ \text{SHIFT}(j)(A) = \text{SHIFT}(1)(\text{SHIFT}(1)(\cdots (\text{SHIFT}(1)(A)))) \]

A backwards shift \( \text{SHIFT}(-j)(A) \) may also be defined based on \( \text{SHIFT}(-1)(A) \) which is defined as

\[ \text{SHIFT}(-1)(A) = \pi_{\theta}(\text{JOIN}(1,2) B) \]

\[ \pi_{\theta}(A \text{ JOIN}(1,2) B) \]

\[ \pi_{\theta}(A \text{ JOIN}(1,2) B) \text{ JOIN}(2,3) B \]

\[ \pi_{\theta}(A \text{ JOIN}(1,2) B) \text{ JOIN}(2,3) B \]

SHIFT(\(j\)), \( j \) positive or negative, is ideally suited for enabling searches forwards and backwards through the data stream based on information found at a particular spot in the data stream. As an example, suppose that conferences in electronic mail are announced by ‘CONFERENCE ON XXX...’ where XXX... is the topic and that it is desired to find conference topics mentioned in the data stream.

If the string ‘CONFERENCE ON ’ is found at the beginning of a DATASTREAM record, then the topic will be located in the first characters of the record in DATASTREAM whose ID# is displaced by a shift of 14 ID#'s from the record in which ‘CONFERENCE ON ’ was located (see Fig. 1).

VI. ELEMENTARY TEMPLATE OPERATORS

Throughout the remaining discussion, no distinction will be made between conditions expressed one character at a time and conditions on strings of characters. In particular, \( \{C1 = 'T' \text{ and } C2 = 'H' \text{ and } C3 = 'E' \} \) will be taken to be the same as \( \{C1 \cdots C3 = 'THE' \} \).

Continuing with the example from Fig. 1, if the records of DATASTREAM are thought of as arranged in ascending order on ID#, then the operation of selecting the record where the topic occupied the first few characters can be viewed in terms of placing a “template” with two windows over DATASTREAM (see Fig. 2). The first window has associated with it conditions that a character string has to satisfy. The record to be recovered is in the second window which is shifted a distance of \( j \) records from the first window. In Fig. 2, the first window is 14 characters wide and has the associated condition that \( C1 \cdots C14 = 'CONFERENCE ON ' \). The information recovered is whatever appears in the second window, which is shifted forward 14 records from the first window.

A template operator is now defined that can be applied to any relation. As with the other special purpose operators previously defined, \( C_{\theta}(R) \) and \( \text{SHIFT}(j)(A) \), this operator can be defined strictly in terms of relational algebra. The operator will be named \( T \) and will be called an elementary template operator, elementary in the sense that it can be used as one component in defining more complicated template operators.

Let \( R(\text{KEY}, \text{A1}, \text{A2}, \cdots, \text{An}) \) be any relation schema, let \( \theta \) be any Boolean condition selecting rows from instantiated \( R \) and let \( B1, \cdots, Bm \) be a subset of the attributes of \( R \). Define the template operator \( T \) by

\[ T_{\theta,j}(R)(B1, \cdots, Bm) = \pi_{\theta,B1,\cdots,Bm}(\text{SHIFT}(j)(\pi_{\text{KEY}}(\text{C}_{\theta}(R)))) \text{ JOIN} \] 

where \( j \) is any integer.

The inner part of the operator retrieves all key values of records satisfying the specified condition \( \theta \). The outer part of the operator shifts the key values \( K \) retrieved in the inner part to \( K+1 \) and retrieves the values of the attributes \( B1, \cdots, Bm \) from the set of records whose keys are elements of the set \( \{K+1 : K \text{ is a key value retrieved by the inner part of the operator} \} \).

Explicitly defining template operators offers a number of conveniences, especially in the setting of data stream processing. As the definition of the template is rather complex, it is easier to refer to a template than to construct the corresponding query. More significantly, the parameters for the corresponding templates can be stored in a relation and the templates retrieved by selecting appropriate tuples. While this is, strictly speaking, an implementation question and not the subject of this paper, a few words on the matter would be of interest and are presented in Section XI.

VII. SIMPLE EXAMPLE USING TEMPLATES

The example of this section illustrates the successive use of templates to extract data from an electronic mail data stream related to the query: find all conference topics of conferences having anything whatsoever to do with Toronto (location, speaker from, editor from, \( \cdots \)). It is assumed that records in the data stream have been grouped together according to the message and that the result of the grouping is as above in CDATASTREAM.
Suppose that it is known that conference titles are generally of the form

...CONFERENCE ON X₁₁X₆X.... ←
...CONFERENCE ON THE X₁₉XXXXXXXXXXX.... ←
...CONFERENCE FOR X₂₀XXXXXXXXXXXX..... ←
...CONFERENCE FOR THE X₂₁XXXXX..... ←

where ← is a carriage return and the numerical subscript denotes the placement of the first X relative to the first C in CONFERENCE. The following discussion is easily extended to include alternatives to the word 'CONFERENCE' such as 'Conference', 'SYMPOSIUM', 'WORKSHOP', etc.

Suppose that \( \theta₁, \theta₂, \theta₃, \) and \( \theta₄ \) are the following Boolean conditions:

\[
\begin{align*}
\theta₁ & : \quad (C₁...C₁₄ = \text{'CONFERENCE ON '}) \text{ and } \neg(C₁₅...C₁₈ = \text{'THE '}) \\
\theta₂ & : \quad C₁...C₁₈ = \text{'CONFERENCE ON THE '}
\theta₃ & : \quad (C₁...C₁₅ = \text{'CONFERENCE FOR '}) \text{ and } \neg(C₁₆...C₁₉ = \text{'THE '})
\theta₄ & : \quad C₁...C₁₉ = \text{'CONFERENCE FOR THE '}
\end{align*}
\]

The relation \( \text{TOP}(\text{CID}\#, \ C₁, ..., Cₙ) \) computed as

\[
\text{TOP} = T_{51,14}(\text{CDATASTREAM})(\text{CID}\#, \ C₁, ..., Cₙ) \\
\cup T_{52,15}(\text{CDATASTREAM})(\text{CID}\#, \ C₁, ..., Cₙ) \\
\cup T_{53,15}(\text{CDATASTream})(\text{CID}\#, \ C₁, ..., Cₙ) \\
\cup T_{54,15}(\text{CDATASTream})(\text{CID}\#, \ C₁, ..., Cₙ)
\]

(7.1)

contains the topics of conferences in the first few characters of each record. Each record also contains superfluous characters, namely those that appear before the first carriage return, ←.

The superfluous characters must now be eliminated from \( \text{TOP} \). This is done by projecting out the desired characters from the records and padding the projected sets of characters with blanks to give them uniform length.

Let \( \text{CONF\_TOPIC(CID}\#, \ \text{TOPIC}) \), where \( \text{TOPIC} = C₁, ..., C₆₀ \) be the schema for the relation that will contain the conference topics of interest. Strictly speaking, \( \text{CON\_TOPIC} \) is not in 1NF insofar as \( \text{TOPIC} \) must be a compound attribute composed of 60 separate one character attributes. Let

\[
\text{TOP}_i = \text{TOP} \quad \text{for } 2 \leq i \leq 61,
\]

\[
\text{TOP}_i = T_{51,i-1} - \sigma_{(C_{i-1} = \text{' '})}(\text{TOP}_{i-1})
\]

(7.2)

\( \text{TOP}_i, k \geq 2, \) is a relation whose records have a CID# and the first \( k-1 \) characters not equal to ← and the \( k \)th character equal to ←. Let \( \text{BLANK}_k \) for \( 1 \leq k \leq 59 \) be a relation with exactly \( k \) blanks. Then, for \( 1 \leq k \leq 60 \),

\[
\text{CONF\_TOPIC}_k = \pi_{\text{CID}\#, \ C₁,..., Cₖ}(\text{TOP}_{k+1}) \times \text{BLANK}_{60-k}
\]

(7.3)

all have records whose \( \text{TOPICs} \) are 60 characters long; \( k \) characters for the conference topic padded by \( 60-k \) blanks.

\( \text{CONF\_TOPIC} \) is then

\[
\text{CONF\_TOPIC} = \text{CONF\_TOPIC}_1 \cup \text{CONF\_TOPIC}_2 \cup ... \cup \text{CONF\_TOPIC}_{60}
\]

(7.4)

The presence of \( \text{CID}\# \) is useful when there are several attributes to be searched for in the messages, such as conference topic and date. The conference topic could then be stored in \( \text{CONF\_TOPIC(CID}\#, \ \text{TOPIC}) \),

where \( \text{TOPIC} = C₁, ..., C₆₀ \),
and the join of the two relations on CID# could be used to associate topics of conferences with the dates of the conferences. This technique will be explored further when discussing the evolution of known contexts.

The CID#'s are also useful when specific information must be selected from the known context based on some characteristic of the data stream. To illustrate this, the full query initially posed at the beginning of the section will be completed—find all conference topics having anything to do with Toronto.

Let TOR(CID#) be a relation storing the CID#'s of those clusters in CDATASTREAM corresponding to messages containing the string 'Toronto'. In particular,

\[ \text{TOR} = \tau_{\text{CID}#}(\sigma_{\text{DATE} = \text{'Toronto'}})(\text{CDATASTREAM}) \]

Then, the records of

\[ \text{TOR} \cap \text{TOP} = \tau_{\text{CID}#}(\sigma_{\text{DATE} = \text{'Toronto'}})(\text{CDATASTREAM})(\text{CID}#). \]

contain the conference topics associated with 'Toronto'.

Carrying along the CID# is also useful when attributes have multiple values. For instance, in CONF_SPEAKERS(CID#, SPEAKER) all speakers at the same conference would have the same CID#.

It is interesting to note that multiple values for attributes may also reflect ambiguity in the information being retrieved. In the example above, conferences may also be announced with the words 'Conference on' pressaging the topic of the conference. Were the above procedure carried out on DATASTREAM (or CDATASTREAM) using the phrase 'Conference on', a string 'Plan to arrive at the Conference on Monday at 9:00 AM...' would give rise to a value 'Monday at 9:00 AM.' for the conference topic in addition to other candidates for the conference topic. Since only one value is expected for the conference topic, the appearance of several possible values introduces ambiguity. In this case, the presence of several values for the conference topic in the same message may profitably be represented using set-valued attributes [10] to signify the ambiguity in the information. Ambiguity is related more to the problem of proper semantic formulation of queries to extract the desired information than to query execution. This problem is reminiscent of the question of precision in the field of information retrieval [11].

VIII. OBSERVATIONS ON THE PREVIOUS EXAMPLE AND THE OPERATOR PAD

The process of finding CONF_TOPIC used in the previous section is illustrative of a general technique that has to be employed when extracting data values from a data stream: first, apply elementary templates, as in (7.1) above, to select rows from CDATASTREAM based on information in other rows; second, from the rows selected, extract the relevant strings of characters [7.2] above; and pad with blanks to fill out the width of the strings to the specification of the attribute for which the strings are values.

Below, an operator PAD is defined that pads the appropriate number of blanks to each string in a set of character strings of varying length so that the resulting set is a set of fixed length character strings. This operator mimics the process in (7.2), (7.3), and (7.4) above. In terms of PAD and templates, the full process of finding CONF_TOPIC is as follows:

\[ \text{PAD}_{\text{TOR} \cap \text{TOP}}(\text{CDATASTREAM})(\text{CID}#, \text{'Toronto'}) \]

where

\[ \text{TOR} \cap \text{TOP} = \tau_{\text{CID}#}(\sigma_{\text{DATE} = \text{'Toronto'}})(\text{CDATASTREAM})(\text{CID}#). \]

\[ \text{PAD}_{\text{TOR} \cap \text{TOP}}(\text{CDATASTREAM})(\text{CID}#, \text{'Toronto'}) \]

IX. COMPOUND TEMPLATES

A compound template operator can now be defined recursively as follows.

1) An elementary template operator \( T()() \) is a compound template operator. (The first pair of brackets takes relation as an argument and the second pair gives the set of attributes returned by the template.)

2) If \( \Sigma()() \) is a compound template operator, then so is \( \text{PAD}(\Sigma)() \).

3) If \( \Sigma_1()() \), \( \Sigma_2()() \), \ldots, \( \Sigma_n()() \) are compound template operators operating on the same relation and outputting attribute values with the same specifications, then \( \cup_{i=1}^{n} \Sigma_i()() \) is a compound template operator.

4) If \( \Sigma_1()() \) and \( \Sigma_2()() \) are compound template operators such that the output of \( \Sigma_1()() \) is contained in the domain of \( \Sigma_2()() \), then \( \Sigma_2()\Sigma_1()() \in \Sigma_2()\Sigma_1()() \) is a compound template operator.

X. KNOWN CONTEXTS: THEIR EVOLUTION AND INSTANTIATION

As mentioned in the Introduction, the known context is a dynamic object. As knowledge about the data stream is acquired by the user, his perception of the data stream changes. Acquisition of knowledge or information results from successive instantiation of
attributes for an envisioned database and changing perception of the data stream implies an expansion of the envisioned database to include new attributes and relationships.

If the known context at any point in time consists of the portion of the envisioned database instantiated up to that time, universal relations 15 form a conceptually well-suited basis from which to describe the evolution of the known context. While there is some controversy as to the broadest extent to which universal relations meet relational database needs, they seem to correspond to databases that are adequate for most purposes 5. Henceforth, attention is restricted to known contexts that are universal relations.

A universal relation schema for a relational database schema is a relation schema containing all the attributes of the database schema. The universal relation may contain null values. Any relation in the database should be obtainable by projection from the universal relation (and elimination of null tuples) and the universal relation should be a lossless join of these projections. A universal relation may have functional dependencies.

A prospective known context can be built up or expanded iteratively through a sequence of known contexts. Let \( K_C \) be a relation that contains only one attribute, the clustering id \( CID# \) for the data stream.

Suppose that \( A_1 \) is an attribute for which values have been extracted from the data stream and that the clustering id’s identifying the regions of CDATASTREAM from which they were extracted have been saved along with them in a relation \( r_1 \) whose schema is \( R_1(CID#, A_1) \). For those \( CID#'s \) in \( K_C \) that have no corresponding value for \( A_1 \), associate a null value. Let \( r_1' \) be the union of the relation \( r_1 \) with these null tuples. Define

\[
K_C' = K_C \Join_{\text{null}} r_1'
\]

In similar fashion, \( K_C \) can be defined from \( K_C_{i-1} \) and values for an attribute \( A_i \).

If attribute values are extracted from the data stream using a refinement of the clustering as indicated at the end of Section IV, the known context should be augmented by the cluster id’s, \( CID#'s \), corresponding to the refinement. The example below illustrates how this can be done by joining \( \pi_{CID#, CID#'}(r(CCIDSTREAM)) \) with the current known context \( K_C \) on \( CID# \) giving \( AK_C \), then joining the new attribute values extracted from CDATASTREAM to \( AK_C \). Attribute values for \( A_{i-1} \) extracted from CDATASTREAM are in a relation \( r_{i-1} \) whose schema is \( R_{i-1}(CID#, CID#'s, A_{i-1}) \) and \( K_C_{i-1} \) is constructed as

\[
K_C_{i-1} = AK_C \Join_{\text{null}} r_{i-1}'.
\]

An analogous process can be carried out for a succession of finer clusterings.

To demonstrate the process described above, suppose that an initial clustering with id numbers \( CID#'s \) identifies records in CDATASTREAM beginning with the word ‘CONFERENCE’. Suppose that another clustering with id numbers \( CID#'s \), identifies records in CDATASTREAM beginning with the character string ‘SPEAKER’. Using the double clustering, representing a refinement of the initial clustering, and appropriate templates, it is possible to pick up all the speakers at each conference from CDATASTREAM (defined in Section IV). If the id numbers identifying rows in CDATASTREAM beginning with ‘CONFERENCE’ are 100, 1500, 2150, … and the id’s identifying rows in CDATASTREAM beginning with ‘SPEAKER’ are 260, 420, 2310, … an instantiation of CDATASTREAM might be based on Fig. 3 as follows.

Assume that first CDATASTREAM was constructed and after it was realized that there was a nonempty set of conferences, the second clustering was made. Then, \( K_C \) would be constructed before CDATASTREAM was formed. The process described above, based on Fig. 3, would yield the following results:

\[
K_C: CID# CONFS-TITLE CID# CID# CONF-TITLE
100 100 PROLOG 260 1500 TUESDAY
1500 1500 null 420 100 PROLOG
2150 2150 DBMS 420 1500 TUESDAY
2310 2150 null 2310 2150 DBMS
\]

\[
K_C': CID# CID# CONF-TITLE AK_C: CID# CID# CONF-TITLE
100 100 PROLOG 260 100 PROLOG
1500 TUESDAY 420 100 PROLOG
2150 DBMS 420 1500 TUESDAY
2310 2150 DBMS
\]

\[
r_1: CID# CID# SPEAKER
260 100 JOHN SMITH
420 100 JAMES JONES
420 1500 null
2310 2150 JUNE SMITH
\]

\[
K_C: CID# CID# CONF-TITLE SPEAKER
260 100 PROLOG JOHN SMITH
420 100 PROLOG JAMES JONES
420 1500 null
2310 2150 JACK JONES
\]

XI. INTERFACING WITH THE USER

While the operators necessary for querying data streams for data values have been developed using relational algebra, they are forbidding in nature. This section briefly discusses a “user friendly” approach to defining and using them. The intention here is to give the flavor of what might be done.

As discussed in Section VI, the essential elements of a query using an elementary template operator are:

1) name(s) for the attribute(s) being instantiated through use of the template,
2) the relation to which the template is to be applied,
3) the Boolean conditions that determine a reference window (window 1 in Fig. 2 in the case of data streams),
4) the shift determining the location of a second window in which the desired data are to be found (window 2), and
5) the attributes to retrieve from the second window.

For data streams, the relation in 2) would be \( (C)\)DATASETS, the Boolean conditions in 3) would relate to string matching or identifying certain patterns in the data stream, and the attributes in 5) would be characters.

The essential information characterizing relatively simple templates for data streams might be stored in a relation similar to TEMPLATES in Fig. 4.

In this illustration, the first record specifies that when 14 characters of the data stream match ‘CONFERENCE ON ’ a string of characters to the first carriage return, up to a maximum of 66 characters, is extracted from the data stream starting 14 characters removed from the first C in ‘CONFERENCE ON ’. The extracted string is an instance of CONF_TITLE.
Once templates are stored in a relation, any standard relational algebra based query language, such as SQL, could be augmented with a statement like

\[ [r] = \text{USE TEMPLATES FOR } \text{attribute} \text{\_name} \]

which would select the records from the TEMPLATES relations that have \text{ATTRIBUTE} = \text{attribute} \text{\_name}, apply them to the data stream, and return the union of the results in \( r \) (if specified). The clustering id's can be retrieved and the appropriate padding with blanks, to meet the specifications of attribute \_name, can be done at the same time.

The language could be further augmented with a statement like

\[ \text{EXPAND UNIVERSE WITH } r \]

which could be used to expand the known context.

Finally, in recognition of the fact that querying the data stream is an evolutionary process, the user must be supplied with a simple mechanism for performing successive clusterings of the data stream. Thus, a statement like

\[ \text{CLUSTER} (n, \text{string}, \text{operator}) \]

might be supplied, that would allow the user to create the \( n \)th cluster level according to some specified comparison of strings in the data stream.

By way of example, the first clustering of a previously unclustered data stream of electronic mail could be achieved using

\[ \text{CLUSTER} (0, \text{MESSAGE:}, =) \]

where each cluster is delineated by the string “MESSAGE:".

Successive clusterings would be performed as needed and USE TEMPLATES FOR attribute \_name would be applied to the most recent clustering.

XII. CONCLUSION

This paper has dealt with the development of a conceptual framework within which to query data streams using relational database techniques. Conceptually, snapshots of the data stream, shifted from each other by one character, have been stored in a relation DATASTREAM and information has been extracted from this relation to instantiate a traditional relational database that may then be queried in the usual fashion. The development of the traditional portion of the database was evolutionary, depending on the information previously extracted from the data stream.
The probable size of the relation `DATASET` makes a
direct implementation of these notions impractical. However,
an actual implementation might involve buffering portions of
the data stream (with no overlap) and extracting information
directly from these portions. The resulting implementation
would involve numerous processes that are not relational, in particular,
specific routines for extracting the data from the data stream that
might be written most efficiently in a special purpose language.
Such supporting routines should, however, be built on top of a
relational database in such a way that the user sees a system that
is relational.

REFERENCES

[1] L. M. Bernstein and R. E. Williamson, "Testing of a natural lan-
guage retrieval system for a full text knowledge base," J. Amer.
11th Annu. CAIS Conf., Halifax, N.S., Canada, May 24–26, 1983,
[4] R. G. Crawford and I. A. MacLeod, "Modular indexing in a re-
vol. 6, pp. 67–75, 1981.
sal relation assumption and its properties," ACM Trans. Database
[7] J. L. Kolodner, "Indexing and retrieval strategies for natural lan-
guage fact retrieval," ACM Trans. Database Syst., vol. 8, no. 3,
Ullman, "System/U: A database system based on the universal
relation assumption," ACM Trans. Database Syst., vol. 9, no. 3,
[9] D. Maier, J. D. Ullman, and M. Y. Vardi, "On the foundations of
the universal relation model," ACM Trans. Database Syst., vol. 9,
no. 2, pp. 283–308, June 1984.
[10] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos, "Extending rela-
tional algebra and relational calculus with set-valued attributes and
aggregate functions," ACM Trans. Database Syst., vol. 12, no. 4,
[12] M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, and A. Gutman,
"Document processing in a relational database system," ACM

Carl S. Hartzman received the B.S. degree from the City College of New York in 1963,
the M.S. degree in mathematics from Purdue University in 1966, and the Ph.D. degree in
mathematics from the University of Colorado in 1970. He obtained formal training in computer
science at the University of Toronto where he received the M.Sc. degree in 1985.

He is currently an Associate Professor with the Department of Mathematics, Statistics, and
Computing Science, Dalhousie University, Halifax, N.S., Canada. His current research interests in computer science
are in the areas of database performance, database design, distributed
databases, and concurrency control.

Carolyn R. Watters received the undergraduate degree in chemistry, the M.L.S. and M.Sc. de-
egrades in computer science from the University of Western Ontario, and the Ph.D. degree in
computer science from the Technical University of Nova Scotia.

She is currently conducting research in text
based systems as an Adjunct Assistant Professor at Dalhousie University.