# Hydrologic data management using RDBMS with differential-linear data storage

G. Tsakalias, D. Koutsoyiannis
*Division of Water Resources, Hydraulics and Maritime Engineering, National Technical University of Athens, 5 Iroon Polytechniou, GR-157 00 Zografou, Greece*

## Abstract

Recently, Relational Data Base Management Systems (RDBMSs) have become popular for handling hydrologic time-series and running hydrologic applications. However, the standard data independence in such systems has many technical disadvantages in storing time-series data. An alternative technique, named Differential-Linear Data Storage (DLDS) technique, has been developed in the framework of the Hydroscope project (a Greek nation-wide database for hydrometeorological information). This technique establishes a standardised representation of hydrologic time-series in a relational database environment, also providing a notable reduction in storage space. Instead of the standard SQL queries (select, insert, delete and update), numerous composite procedures are implemented to facilitate time-series management.

## 1. Introduction

During the last three decades, hydrologic data storage and retrieval has been continuously improved as a result of the evolution of computer technologies in data management. The first systems developed were based on central mainframe computers and programming languages like FORTRAN. Recently, new hydrologic database systems have been developed taking advantage on the available new technologies such as Relational Data Base Management Systems (RDBMSs), graphical user interfaces, etc. Examples of such systems are the National Water Information System II in USA (USGS[5]), CompuMod in Canada (Environment Canada[2]) and Hydroscope in Greece (Tolikas et al.[4]). The latter is a nation-wide distributed database system for hydrometeorological information currently consisting of 12 nodes, each belonging to one of the co-operating or-

ganisations (ministries, universities and research centres). It is based on Hewlett-Packard 9000/s700 UNIX workstations running the INGRES RDBMS.

In a RDBMS the data are organised in tables. Each table may be viewed as a rectangular entity consisting of rows (or records) and columns (or fields). Each row is a separate instance of the same kind of information, while the different fields of the row contain different components of this information. One major feature of RDBMSs is data independence: the system itself controls the details of data storage and retrieval, while the user need not know how the data are stored. Another important feature is that they support the use of special programming languages for data manipulation such as SQL (Structured Query Language), which provides powerful procedures for data access and retrieval (i.e., select, insert, delete and update). Owing to these features, RDBMSs have dominated in several business applications and they have become the state-of-the-art in commercially available database technologies. This has further enhanced their capacity, as they are continually improved offering extensive data handling capabilities and support and development tools. Thus, they currently support distributed databases and provide easy-to-use graphical user interfaces and fourth generation programming languages (such as Windows 4GL) and object oriented techniques (Papakostas et al.[3])

However, RDBMSs have certain disadvantages when used for hydrologic data management (Dodson[1]). These are associated with the time-series character of hydrologic data, their numeric aspect and the complicated calculations needed for certain hydrologic applications. The data independence results in a significant increase of the storage volume, as the date of each data value has to be stored individually with the data value (herein the term "date" means the time of the measurement with the precision offered by the particular RDBMS). In addition, the typical calculations needed for manipulating hydrologic time-series (e.g., interpolation, integration, extraction of maxima or minima) are mainly procedural calculations and thus are not directly supported by non-procedural languages such as SQL. Finally, as it will be described later, the consecutive data values of a time-series are in fact connected to each other such that a possible update of one value may also affect its previous and next data values. This is inconsistent with the data-independence feature of a RDBMS.

Despite these disadvantages, the above mentioned strong features of the commercialised RDBMSs can justify their choice for hydrologic applications. This is the case for the Hydroscope project where the selection of the relational model offered a lot of powerful characteristics that lead to an efficient distributed hydrologic database system. To overcome some of the disadvantages of the relational model, a specific technique, named Differential-Linear Data Storage (DLDS) technique, has been developed and incorporated into the database. The objectives of this technique were (a) to standardise the representation of a time-series in a RDBMS environment; (b) to implement a generalised scheme of a time-series with a varying time step, taking advantage of the data-independence feature of the RDBMS; (c) to cope with the intermittent nature of time-series; (d) to create a set of SQL composite queries that substitute the

typical simple queries in a manner that they are consistent with the data interdependence within a time-series; (e) to provide a set of procedures for typical calculations of a time-series; and (f) to provide an efficient compression technique for time-series data. It is noted that in the Hydroscope system the latter objective (data compression) is also served by other techniques such as integer storage of floating-point values (this is also used in other systems; see Dodson[1]), substitution of the typical date data type offered by the RDBMS with a user-defined data type of a smaller length, and the use of an alternative storage technique for time-series with constant time step, the list storage structure. A brief description of those alternative techniques may be found elsewhere (Papakostas et al.[3]).

## 2. Classification of time-series

Depending on the physical meaning and the observation technique of each hydrologic time-series we can classify hydrologic variables in three categories:

*Instantaneous variables*: Most hydrologic variables such as river stage, river discharge, and temperature are measured (or estimated) and interpreted at a specific time instant (Figure 1a).

*Cumulative variables*: By continuously integrating an instantaneous variable with respect to time, starting at a certain time instant (Figure 1b) we get another type of variable, the cumulative type. A variable of this type is the water volume in a lake or reservoir (the corresponding instantaneous variable being the net inflow into the lake).

*Differential variables*: By intermittently integrating an instantaneous variable with respect to time, starting at numerous arbitrary time instants (Figure 1c) we get a third type of variable, the differential type. Time-series of this type are those of the rainfall depth (each measurement of the rainfall depth is the integral of the intensity for a certain period), the evaporation depth and the sunshine duration.

We can observe that in the first two types the value of the time-series $x(t)$ for any time instant $t$ is unique and has a specific physical meaning. However, in the case of differential variables this is not true, because $x(t)$ depends on the selection of the time instants of discontinuity (i.e., points where the renewal of integration takes place). Consequently, from the point of view of the database manipulation, the differential type needs a different treatment while the other two variable types, which hereinafter will be referred to as non-differential types, can be treated with a common manner. The only difference between instantaneous and cumulative variables is that in the former case given a time period $(t - \Delta t, t)$ the time average

$$x_{\Delta t}(t) = \frac{1}{\Delta t} \int_{t-\Delta t}^{t} x(t)\,dt \tag{1}$$

is meaningful whereas in the latter case it is not. A meaningful quantity for that time period in the case of cumulative variables is the difference

$$x'_{\Delta t}(t) = x(t) - x(t - \Delta t) \tag{2}$$

The corresponding quantity in the case of differential variables is

$$x'_{\Delta t}(t) = x(t) - x(t - \Delta t) + \sum_{t-\Delta t<t_i<t} x(t_i) \tag{3}$$

where $t_i$ is any time instant of discontinuity (renewal of integration).

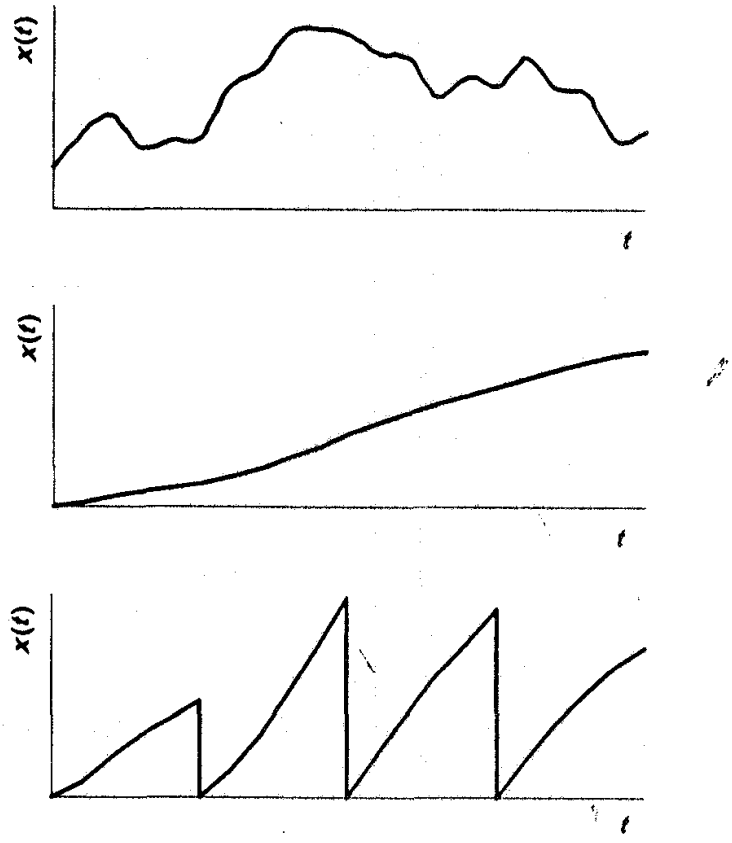


**Figure 1.** Explanation sketch for the classification of time-series: (a) instantaneous type, (b) cumulative type, and (c) differential type.

Because a time-series is never known in continuous time, we will assume that $x(t)$ is known at a set of points $t_j$ only, not necessarily equidistant. Also, we will assume that the distances between consecutive points are the largest distances that can guarantee a valid linear interpolation between them. The series $x(t_j)$ is stored in the database and can be interpreted in two ways, depending on the type of variable (differential or non-differential) as indicated in Figure 2. The technique for the storage and interpretation of $x(t)$ that is described in the next sections, will be referred to as the Differential-Linear Data Storage (DLDS) technique, because of the assumption of linear interpolation and its ability to handle differential time-series.

## 3. Data interdependence

We can observe from Figures 1 and 2 that there is always an interdependence between the separate rows of a time-series in a database table. This is especially apparent in the differential variables where the value at date $t$ has meaning only if we know the date of the previous measurement. Similarly in the case of non-differential time-series we should know when we are not allowed to interpolate any more (when there is a lack of information). For example we see in Figure 2a that we can interpolate (or perform some other calculation) between $t_2$ and $t_3$, but not between $t_3$ and $t_5$ because there is lack of information in that period (e.g., due to instrument malfunction). Thus, it is necessary to add a record with a null value between $t_3$ and $t_5$ (a null is a special value indicating the absence of a valid observation), which will indicate that interpolation is not allowed.

It is obvious that in situations like the described above, simple tasks like the deletion or insertion of a measurement cannot be performed without the consideration of the data interdependence. We cannot simply delete a measurement, but it may be it necessary to replace the deleted record with a null value record to indicate that an interpolation or integration at the measurement's neighbourhood is no longer valid. We also cannot simply insert a value because this will probably alter other values computed by interpolation or integration.

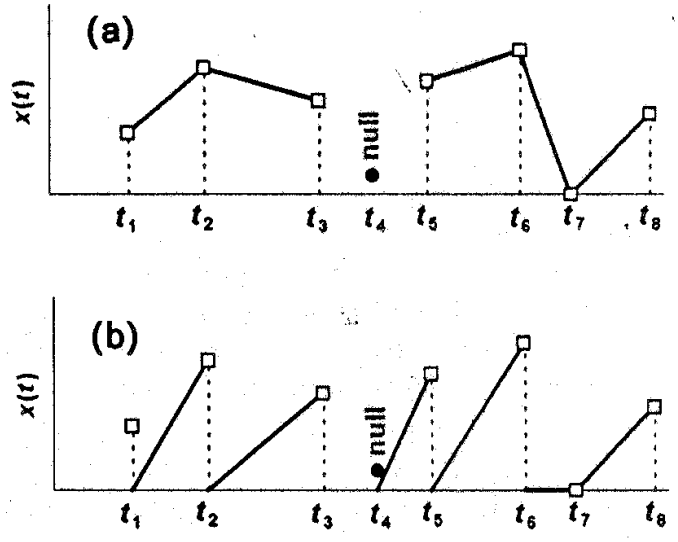


**Figure 2** Explanation sketch for the interpretation of stored data series: (a) non-differential series and (b) differential series

## 4. DLDS chain definition

In relational databases the records are not ordered, thus the notion of the *next* row does not exist. We will explicitly define here the *next* record of the record with date $d_n$ as the record with date $d_{n+1}$ that can be found as:

```
select d_{n+1} = min(date) where date >= d_n
```

In the above notation (as well as in following similar situations) we use an SQL like pseudocode to enhance readability. In a similar way we can define the *previous* date as the date $d_{n-1}$. We will refer to a record with date $d_n$ as the $record_n$.

A DLDS chain is a set of records in a database table representing a part of a time-series and having at least two columns standing for the date and the value of an observation. If $record_1$ is the member of the chain with the minimum date and $record_n$ is the record with the maximum date then: (a) the $record_1$ is the only member of the chain that has a null value; (b) if $d_1 < d_m < d_n$ then $record_m$ is a member of the chain; and (c) the set of records of the chain is not subset of any set of records that form another DLDS chain.

The chain can be interpreted as a differential or a non-differential one (see Figure 2). Obviously a time-series consists of one or more DLDS chains (the number of chains depends on the number of periods with lack of information)

# 5. User-programme interaction

We will examine now how a computer user can manipulate a DLDS chain. Obviously he or she cannot use the conventional procedures used in RDBMSs due to the implicit data interdependence. Despite this, we should consider that the user is familiar with a standard way of interaction with a database front end. The question is if we can keep this type of interaction and use it to manipulate DLDS chains.

We will examine the typical situation where we have a tablefield (an array of fields on the computer screen) that displays time-series data and a user performing actions on the tablefield. We will also assume that the database table (from which the data come from) has two columns only, date and value, and that the column date is a unique key. The tablefield has three columns, date, value and hidden-date. The user can change the value of columns date and value while he or she cannot see the third column.

Each row of the tablefield has a row state (or a destination) associated with it. Every user action modifies this destination. At the end of the session when a *save* command is executed the row destinations will lead to specific database queries. In an independent row format the modification of the row destinations according to user actions and the interpretation of the row destinations to database queries could be briefly described as in Table 1.

In all cases before the insert command the save procedure checks if a record with the same date already exists and provides the user a dialog box to decide if the existing record should be overwritten.

We can observe in Table 1 that a row destination always leads to a unique SQL query. Hence, we want to keep the user-programme interactions unchanged (thus retaining the consistency of the user interface) and use these row destinations to manipulate DLDS chains. This can be done by substituting each single query (update, insert, delete) with a sequence of queries, depending on the chain surrounding the processed row, as we can see in the next section.

Table 1. User-programme interaction

| Previous row state | User action | Tablefield modification | New row state | Database query |
|---|---|---|---|---|
| none | select from the database | the rows appear on the tablefield | unchanged | none |
| unchanged or changed | write new value | | changed | update |
| unchanged or changed | write new date | | new | insert |
| | | an invisible row inheriting the hidden date of the parent row is added | deleted | delete |
| unchanged or changed | delete row | the row becomes invisible | deleted | delete |
| none | insert new row | | undefined | none |
| undefined or new | write new value | | new | insert |
| undefined or new | write new date | | new | insert |
| new | delete row | the row becomes invisible | deleted | delete |

# 6. Mapping of standard SQL to DLDS queries

The mapping of the single queries (update, insert, delete) to composite DLDS queries is given in Table 2. Each composite query is as a set of rules, all triggered each time a row is considered by the save procedure. Each rule has a condition part and an action part. Generally, the rules are not mutually exclusive. The composite queries keep the name of the replaced single queries with the addition of the character '#'. Some columns of Table 2 need explanation:

*Chain conditions*: In order for a rule to perform its action some conditions must hold. The chain conditions refer to the processed row and the surrounding rows of the DLDS chain.

*Rule conditions*: Some rules are mutually exclusive. This is explicitly declared in this column.

*Dialog*: Before a rule's action is performed, a dialog box may appear. This dialog box can be one of two types: *Warning* is just a message providing the user useful information. *Confirmation* provides the user a message and he or she can answer 'yes' or 'no'. If the user answers 'yes' the rule's action will be performed otherwise no action will be taken.

We note here that in the case of the first insertion in the database the rule I6 creates the first DLDS chain. We also note that the actions taken after the update# DLDS query are also DLDS queries.

**Table 2. DLDS queries that replace standard SQL queries**

| DLDS Query | Rule Name | Chain Conditions | Rule Conditions | Dialog | Action |
|---|---|---|---|---|---|
| delete# the record$_n$ | D1 | value$_n$ is not null | | | update value$_n$ to null |
| | D2 | value$_n$ is null<br>value$_{n+1}$ is null | | | delete record$_n$ |
| | D3 | value$_n$ is null<br>value$_{n+1}$ is not null | | confirmation<br>"The chain will be destroyed. You should probably insert a record to maintain the chain. Should the record be deleted?" [yes] | delete record$_n$ |
| insert# a record with date $d_n$ | I1 | exists record$_n$ with date $d_n$ ($d_n$ is a junction of two links at the chain) | | confirmation<br>"Should the record$_n$ be overwritten?" [yes] | delete existing record$_n$<br>insert new record$_n$ |
| | I2 | value$_{n+1}$ is null | not I1<br>not I4 | | insert record$_n$ |
| | I3 | exists record$_a$, $d_a < d_n$<br>exists record$_b$, $d_b > d_n$<br>($d_n$ is in a chain range and lies in the middle of a link) | not I1 | confirmation<br>"The link will be split into the links ($d_a$-$d_n$, $d_n$-$d_b$ thus modifying the chain. Should the new record be inserted?" [yes] | insert record$_n$ |
| | I4 | not exists record$_b$, $d_b > d_n$<br>(the new record lies after the last chain) | not I1 | | insert record$_n$ |
| | I5 | value$_n$ is null<br>not exists record$_a$, $d_a < d_n$<br>(it is the "first" record in the table) | not I1 | | insert record$_n$ |
| | I6 | value$_n$ is not null<br>not exists record$_a$, $d_a < d_n$<br>(it is the "first" record in the table) | not I1 | warning<br>"The record$_{n-u}$ with a null value will be inserted just before[§] this record$_n$ to indicate the start of the chain." | insert record$_n$<br>insert record$_{n-u}$ |
| update# a record with date $d_n$ and hidden date $d_{nh}$ | U1 | | | | delete# record$_{nh}$<br>insert# record$_n$ |

[§] Since time is discrete in DBMSs "just before" means one time unit (u) before