THE MANAGEMENT AND ANALYSIS OF INFRASTRUCTURE TIME SERIES DATA: AN ENVIRONMENTAL TIME SERIES DATABASE

Greg Reilly [1]

ABSTRACT

Until recently, the City of Ottawa did not have a centralized and coherent system to manage their long-term water and sewer time series data. Consequently, it was difficult to perform data management tasks, access data, and do useful analysis.

The City's Water Resources Group initiated the Environmental Time Series (ETS) database project. ETS has organizational and time-saving features that reduce human error and make tasks like data loading, validation, and derivation of new data easy to learn and perform. ETS has a simple and powerful means of deriving data that transparently manages data quality. These features facilitate the management of very large amounts of data.

A well-organized database system with all required data readily available makes for powerful and flexible data analysis. Its ease of use facilitates detailed as well as broad perception of the City's infrastructure behaviour. This minimizes assumptions and maximizes optimization of existing and future infrastructure. In short, it promotes good decision-making.

KEY WORDS

Database, infrastructure, data type, open-architecture, data analysis, query, SQL

INTRODUCTION

The City of Ottawa has a large amount of water-supply, sewer-collection, and weather-related time series data that used to be stored in multiple formats and locations, and managed by multiple software programs.[2] This presented significant problems:

- chaotic organization
- risk of data loss or storage media obsolescence
- limited user access
- no data quality system
- limited activity records and documentation
- limited ability to query and perform data analysis

[1] Water Resources Analyst, Department of Public Works and Services, City of Ottawa, 100 Constellation Cres., Ottawa, K2G 6J8, Canada, 613/560-6065, FAX 613/560-6068, greg.reilly@ottawa.ca.

[2] ETS currently has almost 400,000,000 data items.

- cumbersome and hidden derivation of new data from existing data
- high dependence on individuals' knowledge

ETS was developed to solve these problems.

ETS ORGANIZATION

ETS is an open-architecture database system. It is made up of stand-alone components, and new components can be added. Most importantly, the database itself is a stand-alone entity that third party query and analysis software can freely access without depending on the other components. The current components of ETS are:

- the database
- the front-end application that organizes and executes data management activity
- the calculation engine (.dll file) that performs new data derivation
- the query and analysis tool

The City's IT department administers the database. One of their most important roles is to ensure that it is always backed up. ETS is available to any network user on request.

The database is a normalized relational database. The table structure follows standard rules of normalization, which are central to relational theory. Normalization ensures the mathematically simplest structure by removing data integrity problems such as unnecessary dependencies and complexities. Essentially, it means that each table represents one single independent type of entity. In ETS, the raw_data table is a list of measurements and pertinent information about them. The site table is a list of the locations at which measurements have been made, and the parameter table is a list of the types of measurements that have been made. Normalization helps make a database well-organized, efficient, and flexible. This can mean being able to use it for unexpected purposes.

Table 1 shows a simplified view of the ETS table structure, or data model, with the central tables: raw_data, site, and parameter. The actual data model has more than 50 tables. The tables not shown deal with the data loading, validation, and derivation objects, as well as users, geographic areas, GIS (Geographic Information System) links, execution errors, equipment, and user-defined site properties. There is also a derived_data table which is similar to the raw_data table. It also has a data_value and a confidence_level field.

Table 1: Simplified ETS Data Model

raw_data table

Column Name	Data Type	Column Description
Site_id	Number	Unique site identifier, foreign key to site table
Parameter_id	Number	Unique parameter identifier, foreign key to parameter table
Timestamp	Date/time	Date and time of the measurement
Data_value	Number	Value of the measurement
Confidence_level	Number	Quality indicator: 0=unknown; 1=bad; 2=questionable; 3=good

site table

Column Name	Data Type	Column Description
Site_id	Number	Unique site identifier, primary key for table
Code	Text	Text identifier, a name
Location	Text	Geographic location

parameter table

Column Name	Data Type	Column Description
Parameter_id	Number	Unique parameter identifier, primary key for table
Code	Text	Text identifier, a name
Unit_type_id	Number	Unique unit type identifier, foreign key

Each sequence of rows in the raw_data table with the same site_id and parameter_id represents a time series for that site and parameter.

This data model does not need to change when new sites or parameters are created because the sites and parameters are seen as data, not part of the database structure. When a new site or parameter is added, a row is simply added to the site or parameter table.

This model also makes querying flexible. For example, it is simple to write a short SQL (Structured Query Language) statement that involves multiple sites or parameters. The following is a sample of such a query:

```
SELECT s.code, avg(r.data_value) as average_flow
FROM raw_data r, site s, parameter p
WHERE s.site_id=r.site_id and p.parameter_id=r.parameter_id and p.code='PSFlow' and
  r.confidence_level=3
GROUP BY s.code;
```

This statement averages all good (confidence_level=3) rows in the raw_data table that have the parameter_id for the "PSFlow" (pumping station flow) parameter. It groups the averages for each site name. The result is average flow for all pumping station sites.

If the data model had one separate table for each site, for example, it would not be as flexible. There are many applications for such multiple-site queries and also multiple-parameter queries. Other examples are given in the "Data Analysis" section.

DATA MANAGEMENT

The ETS front-end application organizes and executes data loading, validation, new data derivation, maintenance of sites and parameters, and equipment tracking.

EXECUTABLE ACTIVITY LISTS

All data loading, validation, and derivation activities are organized and carried out through "executable activity lists".

Each type of activity has its own list, so there is a list of data loads, a list of validations, and a list of derivations. Each data load, validation, or derivation is an activity object that contains the information required to perform one activity, (or a group of activities as data loads are lists of lists).

There are different variations, but the basic user steps to perform an activity are: defining the object to perform it; adding it to the appropriate list; and executing it. Once this is done, the list immediately serves as a complete record with the new activity added.

Any object in the list can be executed at any time, so redoing an activity is the same as doing it the first time. Any number of new objects can be added and multiple objects can be selected for being executed, so doing multiple tasks or redoing part or all of the activity history is done in the same way.

Executable lists make all activities openly visible. They act as a history of activity that automatically documents the source of data and details about how, when and by whom it was created. This reduces dependence on individuals' knowledge, and makes learning easier because what has been done by others in the past can be seen and followed. Since executable lists are stored in the database, they are data too and can be queried.

Objects are usually created by copying a similar one and changing a small part of it. This saves time and reduces human error. If a mistake does occur, it is usually easy to see what has been done wrong and redo it correctly. This reduces user anxiety.

This feature has helped make possible the management of large numbers of activity objects and very large amounts of data.

DATA LOADING

ETS has flexible loading of row-column ASCII files. All the information required to load a given file type is stored in a user-defined template object for that file type called a data source. A data source is created and saved in the database for each file type to be loaded. The data source defines the destination site and specifies column to parameter mappings, unit conversions, and time adjustments.

ETS uses an executable list of data load objects to perform data loading. A data load itself is a sub-list that represents one sequence of files. This structure is necessary to handle many files and many file types. A data load's members are objects that have a file name and the name of the data source to use for it. New files are easily added by copying any existing member and editing the file name.

To load data, the user adds file names to the appropriate data loads, sets previously loaded files not to load if necessary, and executes the data loads.

The capability to save and reuse data sources means that the user does not have to define how to load files every time loading is done. The ability to copy a similar object minimizes the risk of using the wrong data source to load a file.

VALIDATION

Data validation assigns a quality indicator (see Table 1) called a confidence level to raw data. This is stored in the confidence_level column of the raw_data table.

All data is kept regardless of quality to allow for possible future re-validation, and because all data is of interest in some analyses. For example, someone may want to know what fraction of a particular site's data is good, or it may be necessary to see all data, good or bad, to help trouble-shoot equipment. Also, some analyses can tolerate data of lesser quality.

ETS uses an executable list of validation objects to perform validation. Validation objects define the confidence level to be applied to raw data for a site and parameter over a range of time.

The user reviews data with charts, uses the features of executable lists to add the required validations, then executes them. Executing them assigns the confidence_level field directly.

DERIVATION

Derivation creates new data from existing data using arbitrarily complex, user-defined text expressions.

ETS uses an executable list of derivation objects to perform derivation. Derivation objects define the calculation expression to create derived data for a site and parameter over a range of time.

The user creates the required derivations and executes them. Normally they are defined once for a given site and then the same ones are used every time new raw data is loaded, but changes can be added at any time and existing derivations can be re-executed, as allowed by the executable list. Executing them creates derived data in the derived_data table. Re-executing overwrites it.

The variables of an expression are the names of parameters or assigned variables. Parameters can represent raw or derived data. They are assumed to belong to the target site unless specified otherwise using [site].[parameter] syntax. The target timestamp is the same timestamp for which the values of the input parameters are retrieved.

Unlike the many scripts and individual programs involved in the legacy systems, the calculation expressions are readily visible for any user to see how data is calculated.

Calculation Engine

Derivation execution is carried out by the calculation engine which is a .dll file. Its functionality can be changed or expanded at any time with C++ programming.

All data items stored in ETS have two parts, the data_value and confidence_level fields of the raw_data and derived_data tables. The calculation engine represents all data items with a data type called the reading type. It has three parts: a value, a confidence, and an error message.

The reading type is used as one uses data types such as floats or complex numbers to write an expression in a programming language. The rules to deal with its parts are transparent. It defines all operations to take confidence into account, so that derived data automatically inherits its confidence from the data it is derived from. There is no need to validate derived data manually, or assign confidence explicitly in calculation expressions. Users just create normal expressions and execute them. This keeps the process simple and minimizes errors in confidence.

The data type is transparent but documented. For standard mathematical operations performed on the data, the confidence of the result is that of the worst of its operands (there are exceptions listed below). For example, bad 2 + good 2 equals bad 4. This is the "minimum confidence rule".

There is however the *option* to manage confidence explicitly, if this is required. Special functions are defined that can act on or assign the confidence of the result. Statements can be conditional on confidence or they can pick out only the best of their arguments. For example, bestavg(arg1, arg2, …, argn) finds the average of only those of its arguments that have the best confidence. An expression can also explicitly assign the confidence of its result. This allows automatic validation.

When a math error occurs, the error message is put into the error part of the reading object and the calculation continues. The error message is carried throughout the calculation and is reported to the user when the derivation is complete. The error message can also be assigned explicitly (see “reading” function in Table 2). This allows custom error messages for specific conditions where a problem is known to occur in a part of an expression.

Engineering functions carry out specific engineering tasks. This is the most likely kind of feature to be a new requirement for the calculation engine but, fortunately, adding functions is the easiest kind of change to make in the calculation engine .dll file. Copying the code from another function and modifying it is reasonably straight-forward.

The following are samples of calculation engine syntax features.

Table 2: Sample Calculation Engine Syntax Features

Syntax	**Meaning**
+ - * / ^ < > <= >= =	Standard operators
avg(x1,x2,...xn)	Average of all x's
best(x1,x2,...xn)	Returns the first x with the best confidence*
bestavg(x1,x2,...xn)	Average of all x's with the best confidence*
bestavgmin(min_num, x1, x2,..xn)	Like bestavg but if # of x’s with best conf is less than min_num, result confidence is 0
iif(condition, then_result, else_result)	Standard “if” function
conf(x)	Value of result is conf. of x; conf. of result is 3
value(x)	Value of result is value of x; conf. of result is 3
reading(val,confidence)	Constructs a reading with value=value(val), conf.=value(confidence), no error message
reading(val,confidence, “error”)	Constructs a reading with value=value(val), conf.=value(confidence), error message=”error”
<<	Variable assignment
;	Statement separator
<parameter name>[n]	Indexing shifts parameter’s timestamp n timesteps; negative n shifts backward; positive shifts forward
circarea(depth, diam)	x-sectional area of water in circular pipe
manningflow(depth, diam, slope, n)	Flow by Manning equation
contflow(depth, diam, velocity)	Flow by continuity equation
pipevolume(inv1, inv2, length, diam, surf)	Volume of still water in pipe; inv1=1st invert, inv2=2nd invert, surf=surface elevation

*Confidence of result is the best of the operands’ confidence.
Notes: where not indicated otherwise, the confidence of the result follows the minimum confidence rule, hence is the minimum of the confidences of the operands. All of the functions and operators return a reading object and all of the functions’ non-text arguments are reading objects.

The following example expression calculates rate of flow into a tank based on the volume of liquid in the tank and the rate of flow out of the tank:

```
bestavg(outflow[0],outflow[-1])+(volume[0]-volume[-1]) / (timestamp[0]-timestamp[-1])
```

There are subtle exceptions to the minimum confidence rule. These occur when a result is not a function of one of its operands. For example, a good zero times anything is a good zero because it doesn't matter how bad the other operand is. Similarly, a good one raised to any power does not depend on that power. However, division does observe the minimum confidence rule: a good zero divided by something *is* a function of that operand because whether it is zero or non-zero makes a difference.

These subtleties extend to the engineering functions since the reading type is used internally for their implementation. For example, "contflow" will return a good zero if the depth of flow is a good zero even if the diameter or velocity is bad.

There are also subtle effects of the minimum confidence rule. For example, the confidence of the "iif" function's result depends on the confidence of the "condition" argument, and the confidence of the "reading" function's result depends on the confidence of the "confidence" argument.

As these rules are encapsulated in the data type, users don't have to refer to them in creating expressions.

DATA ANALYSIS

QUERY AND ANALYSIS TOOL

ETS has a flexible, user-friendly query tool that makes it easy to create, save, and organize queries, charts, and tables. Features include a spreadsheet-like query definition table, an option to show different confidence levels in different colours, a tree structure to organize saved queries and charts, and clipboard data extraction. Data can be extracted in a format similar to the native tables, or a format with a column for each parameter. Timestep resampling can redefine timesteps in different ways in the extracted data.

The query editor executes Oracle SQL, Microsoft JET SQL, and Microsoft Visual Basic for Applications statements and functions. Features include text substitution variables and the ability to create Microsoft Access tables.

There are established queries for sewer inflow and infiltration analysis, frequency distributions, long term trends, flow exceedences, correlation tests, hydraulic model parameters, data coverage, weekly flow patterns, percentiles, outdoor water use, and weather data analysis. There are also queries that provide information about the ETS database itself. For example, there is one that lists all derivations containing a given text substring. It can be used to find how many times a given function is used.

Ad Hoc Query Example

The potential benefit of using real-time control for the City of Ottawa's main sewer trunk (Ottawa Outfall Sewer) was quickly assessed using an ad-hoc query.

The first step was to use the query tool to extract from the ETS database a Microsoft Access table (given the name `data0`) with a field for the timestamp, the depth of flow at a site on the main trunk, and the depths of flow in four combined sewer overflow sites.

The overflows protect the main trunk from overload by diverting flow from its tributaries into the Ottawa River during rainfall events. The following query counts timestamps at which overflow to the river occurred while the main trunk was less than half full. The fields `bol`, `bot`, joo, and rci are the overflow depths and bee is the depth in the main trunk. Overflow depths greater than zero indicate the occurrence of overflow. The 2.44 is the diameter of the main trunk and the .5 is the fraction full.

```
SELECT count(*) as timestep_count
FROM data0
WHERE (bol>0 or bot>0 or joo>0 or rci>0) and bee<2.44*.5;
```

Running the same query without the bee field criterion provided the total number of overflow timesteps without regard to main trunk depth. The first count divided by the second indicated that, for the study period, the main trunk was less than half full 31% of the time that there was overflow.

Doing the same test with different fractions produced the following result:

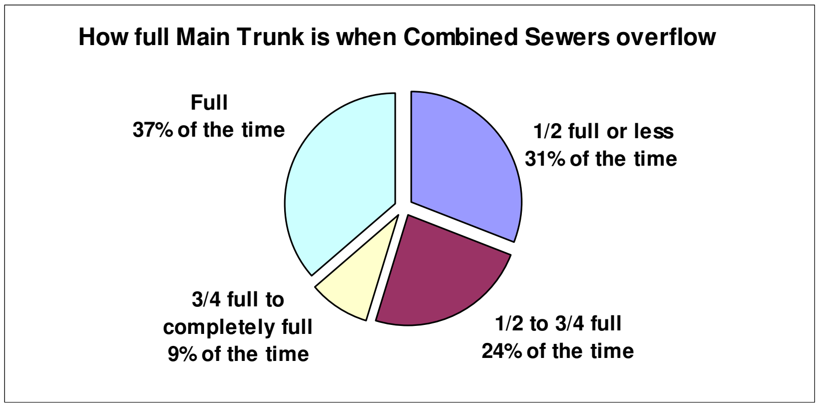

Figure 1: How Full the Main Sewer Trunk is when Combined Sewers Overflow

Since real-time control can reduce overflow depending on unused capacity in the main trunk, these results presented strong support for it. This query convinced City managers to pursue real-time control which will save the City $50 Million by reducing overflows.

Water Consumption Patterns Example

ETS has water consumption data for individual buildings. In ETS, each building is a site. The ETS front end allows users to create properties and assign text values for them to sites. The properties are stored in the property table and the assignments are stored in the site_property table. We use this feature to assign a land use code to each building site.

property table

Column Name	Data Type	Column Description
Property_id	Number	Unique property identifier, primary key for table
Code	Text	Property name

site_property table

Column Name	Data Type	Column Description
Property_id	Number	Property identifier
Site_id	Text	Site identifier
String_value	Text	Value of property assigned to the site

The following query creates an average weekly water consumption pattern for each land use code. This is useful for water distribution system hydraulic modelling. Also, this example shows the flexibility of the data model in its simple use of multiple sites.

```
SELECT
  sp.string_value as land_use_code,
  mod((trunc(rd.timestamp)-to_date('1900/01/01','yyyy/mm/dd')),7) as day_of_week,
  trunc(trunc((rd.timestamp-trunc(rd.timestamp))*1440/60,0)*60) as hour_of_day,
  avg(rd.data_value) as avg_value
FROM raw_data rd, parameter pa, site_property sp, property pr
WHERE
  pa.parameter_id=rd.parameter_id and pa.code='Meter Master Flow' and
  sp.site_id=rd.site_id and pr.property_id=sp.property_id and pr.code='LUCode' and
  rd.confidence_level=3
GROUP BY
   sp.string_value,
   mod((trunc(rd.timestamp)-to_date('1900/01/01','yyyy/mm/dd')),7),
   trunc(trunc((rd.timestamp-trunc(rd.timestamp))*1440/60,0)*60);
```

This query lists an average value for the “Meter Master Flow” parameter for each land use code, each day of the week, and each hour of the day. It uses all sites that have a land use code property assigned to them.

The following chart shows sample results. They have been made dimensionless by dividing by their mean:

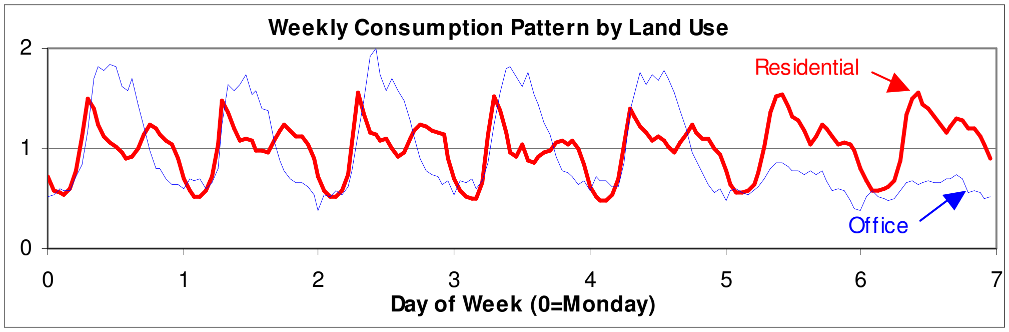

Figure 2: Weekly Land Use Consumption Patterns

Correlation Example

When data exists in the raw_data table for a given site and one or more parameters, those parameters are said to be the set of raw data parameters belonging to that site. In this query, that set of parameters is sampled, two at a time, and a correlation coefficient is calculated from the data values of the two time series corresponding to the sampled parameters. The query lists all possible pairs of raw data parameters belonging to a given site along with the correlation coefficient for each pair.

This is useful for assessing the relationships among different measurements at a site. Also, this example shows the flexibility of the data model in its use of multiple parameters. The code “ss_haw” refers to Hawthorne Shaft, a shaft on a major sewer trunk.

```
SELECT  pa.code as xname,  pb.code as yname,
  sum(a.data_value*b.data_value)-sum(a.data_value)*sum(b.data_value)/count(*) as s
INTO data0
FROM
  raw_data a, site sa, parameter pa,
  raw_data b, site sb, parameter pb
WHERE
  sa.site_id=a.site_id and sa.code='ss_haw' and pa.parameter_id=a.parameter_id and
  sb.site_id=b.site_id and sb.code='ss_haw' and pb.parameter_id=b.parameter_id and
  b.timestamp=a.timestamp
GROUP BY pa.code, pb.code;

SELECT c.xname, c.yname, c.s/(a.s*b.s)^.5 as r
FROM data0 a, data0 b, data0 c
WHERE
  a.xname=a.yname and b.xname=b.yname and c.xname=a.xname and c.yname=b.yname;
```

The first SELECT statement makes the table data0, which lists a calculated value s for all parameter pairs. It lists the parameters' names and the field s calculated as Σxy-Σx*Σy/n, in which x is the first parameter's data values, y is the second parameter's data values, and n is the number of (x, y) data pairs. It includes all possible parameter pairs in either order as well as pairs of the same two parameters. It therefore contains all of the values sxx=Σxx-Σx*Σx/n, syy=Σyy-Σy*Σy/n, and sxy=Σxy-Σx*Σy/n, with each of these on a separate row.

The second SELECT statement does a self-join on three copies of data0 to put sxx, syy, and sxy all on the same row for each parameter pair so that the final calculation, r=sxy/(sxx*syy)^.5, can be made for the correlation coefficient.

With minor modifications, this query could be made to work on one parameter and list the correlation coefficient for all pairs of sites, or work with multiple sites and multiple parameters. It could also easily be adapted to derived data.

CONCLUSIONS

ETS has solved the problems that were identified. It manages a very large amount of data, and makes data access and analysis available to all who need it. Its power and ready availability have given users heightened insight into the behaviour of the City's water and sewer infrastructure.

ACKNOWLEDGMENTS

Shaun Richards, Rick Seeley, and Barbara Fox (all of AJJA Information Technology Consultants), Carlos Narvaez, Louis Julien, Adrien Comeau, Chris Melanson, Gerry Taylor, Ron Rooke, Kevin Cover, and Pat Leblanc (all current or former employees of the City of Ottawa), helped formulate the ideas used in ETS. Shaun Richards designed the data model.

REFERENCES

Korth, Henry F., and Silberschatz, Abraham (1991). *Database System Concepts.* McGraw-Hill, Inc., New York, 694 pp.