SciQL: Array Data Processing Inside an RDBMS

Ying Zhang Martin Kersten Stefan Manegold
Centrum Wiskunde & Informatica
Science Park 123, 1098 XG
Amsterdam, The Netherlands
Ying.Zhang, Martin.Kersten, Stefan.Manegold@cwi.nl

ABSTRACT

Scientific discoveries increasingly rely on the ability to efficiently grind massive amounts of experimental data using database technologies. To bridge the gap between the needs of the Data-Intensive Research fields and the current DBMS technologies, we have introduced SciQL (pronounced as 'cycle') in [15]. SciQL is the first SQL-based declarative query language for scientific applications with both tables and arrays as first class citizens. It provides a seamless symbiosis of array-, set- and sequence- interpretations. A key innovation is the extension of value-based grouping of SQL:2003 with structural grouping, i.e., group array elements based on their positions. This leads to a generalisation of window-based query processing with wide applicability in science domains.

In this demo, we showcase a proof of concept implementation of SciQL in the relational database system MonetDB. First, with the Conway's Game of Life application implemented purely in SciQL queries, we demonstrate the storage of arrays in the MonetDB as first class citizens, and the execution of a comprehensive set of basic operations on arrays. Then, to show the usefulness of SciQL for real-world array data processing use cases, we demonstrate how various common image processing and remote sensing operations are executed as SciQL queries. The audience is invited to challenge SciQL with their use cases.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Arrays; H.2.3 [**Languages**]: Query languages; H.2.8 [**Database Applications**]: Scientific databases

Keywords

SciQL, array query language, array database, scientific databases

1. INTRODUCTION

The array computational paradigm is prevalent in most sciences and it has drawn attention from the database research community for many years. The object-oriented database systems of the '90s allowed any collection type to be used recursively [1] and multi-dimensional database systems took it as the starting point for their design [6]. The hooks provided in relational systems for user defined functions and data types create a stepping stone towards interaction with array-based libraries, e.g., RasDaMan [2] is one of the few systems in this area that have matured beyond the laboratory stage. SciDB [13] is developing an array database system from scratch, with tailored features for the need of the science community. Nevertheless, the array paradigm taken in isolation is insufficient to create a full-fledged scientific information system. Such a system should blend measurements with static and derived metadata about the instruments and observations. It therefore calls for *a strong symbiosis of the relational paradigm and array paradigm*. SciQL has been designed to fill this gap.

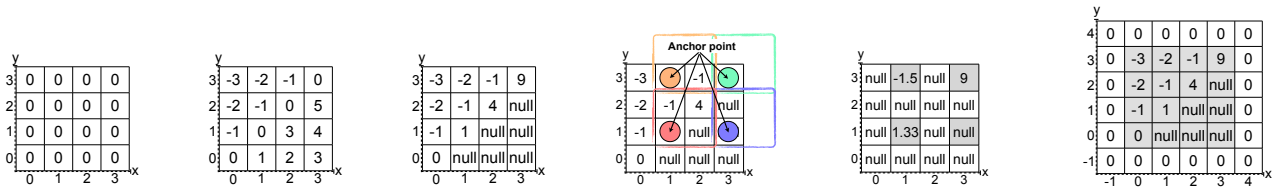
Relational database management systems are the prime means to fulfil the role of application mediator for data exchange and data persistence. However, they have not penetrated the sciences in the same way they have the business world. The mismatch between application needs and database technology has a long history, e.g., [7, 4, 13, 5, 8, 6]. The main problems encountered with relational systems in science can be summed up as *i)* the impedance mismatch between query language and array manipulation, *ii)* the difficulty to write complex array-based expressions in SQL, *iii)* arrays are not first class citizens, and *iv)* ingestion of terabytes of data is too slow. Traditional DBMSs simply carry too much overhead. Moreover, much of the science processing involves use of standard libraries, e.g., LINPACK, and statistics tools, e.g., R. Their interaction with a database is often confined to a simplified data import/export facility. The proposed standard for management of external data (SQL3/MED) [11] has not materialised as a component in contemporary system offerings.

The SciQL Language A query language is needed that achieves a true symbiosis of the table and array semantics in the context of existing external software libraries. This led to the design of SciQL, where arrays are made first class citizens by enhancing the SQL:2003 framework along three innovative lines:

- *Seamless integration* of array-, set-, and sequence- semantics.
- *Named dimensions with constraints* as a declarative means for indexed access to array cells.
- *Structural grouping* to generalise the value-based grouping towards selective access to groups of cells based on positional relationships for aggregation.

A table and an array differ semantically in a straightforward way. A table denotes a set of tuples, while an array denotes an indexed collection of tuples called *cells*. All cells covered by an array's dimensions always exist *conceptually*, while in a table tuples only exist after an explicit insertion. Arrays may appear wherever tables are allowed in SQL expressions, producing an array if the projection list of a SELECT statement contains dimensional expressions.

SIGMOD'13, June 22–27, 2013, New York, New York, USA.



(a) A 4×4 matrix (b) After update (c) After ins/del (d) Array tiling (e) Tiling result (f) Dimension expansion

Figure 1: SciQL array operations

An important operation is to carve out an array slab for further processing. The windowing scheme in SQL:2003 is a step into this direction. It was primarily introduced to better handle time series in business data warehouses and data mining. In SciQL, we take it a step further by introducing an easy to use language feature, called *structural grouping*, to identify groups of cells based on their positional relationships. Each group forms a pattern, called a *tile*, which can be subsequently used to derive all possible incarnations for, e.g., statistical aggregations.

SciQL Implementation and the Demo To assess the usefulness of SciQL for scientific data processing, we have implemented a proof of concept of the language in the relational database system MonetDB [12]. The fact that MonetDB uses BATs (Binary Association Tables) [3], which are physically represented as consecutive C arrays, suggested it as a good basis to implement SciQL. To truly realise arrays as first class citizens of the DBMS, array support has been integrated into all layers of the MonetDB software stack.

The demo contains two parts; both are accompanied by a poster illustrating the main language features of SciQL. In part I, we elaborate individual SciQL features using small arrays. As an application senario, we use the Conway's Game of Life [10]. All play rules are implemented as SciQL queries, capturing language features, e.g., array creation, array data/structure manipulations, and structural grouping. In part II, we show SciQL in action with real-world image processing examples obtained from the European project TELEIOS [14]. We demonstrate how images (e.g., remote sensing images) are stored in MonetDB as arrays (instead of BLOBs) and processed using SciQL queries. A rich set of typical image processing operations, e.g., smooth, resize, rotate and zoom, are expressed as consice SciQL queries and executed directly in MonetDB on the image data. In both parts, the audience has full control of the demo through SciQL queries. In addition, they are highly encouraged to pose new scenario's to be executed by the demo.

Further in the paper, we first briefly describe several main language feature of SciQL in Section 2. Then we shortly depict the SciQL implementation in MonetDB in Section 3. We elaborate the demonstration scenario's in Section 4 and conclude in Section 5.

2. SCIQL OVERVIEW

In this section, we summarise main language features of SciQL. Detailed description of all SciQL features can be found in [15].

Array Definition A SciQL array has one-or-more *dimensions*. A dimension is a measurement of the size of the array in a particular named direction, e.g., "x", "y" or "time". A dimension can have optional *dimension range* constraints, given by `[<start>:<step>:<stop>]`, which is composed out of expressions each producing one scalar value. The interval `[start,stop)` is right-open. A dimension is *fixed* if all three expressions of its dimension range are specified by `<literal>` values; otherwise, it is *unbounded*.

Cells in an array can have zero-or-more *cell values*, which can be of any scalar data types and they may use a DEFAULT clause to initialise their values. Omitting the default implies a NULL as the default value. A cell is called *empty* or a *hole* if all its cell values are NULL. The query below creates the 4×4 matrix shown in Figure 1(a).

```
CREATE ARRAY matrix (
  x INT DIMENSION[0:1:4], y INT DIMENSION[0:1:4],
  v INT DEFAULT 0);
```

Array and Table Coercions One strong feature of SciQL is to switch easily between a TABLE and an ARRAY perspective. Any array is turned into a corresponding table by selecting its attributes. The dimensions form a compound primary key. For example, the previous `matrix` becomes a table using the expression `SELECT x, y, v FROM matrix`. An arbitrary table can be coerced into an array if the column list of the SELECT statement contains the *dimension qualifiers* '`[`' and '`]`' around a projection column: `[<column name>]`. For instance, let `mtable` be the table produced by casting the array `matrix` to a table. It can be turned into an array by picking the columns forming the primary key in the column list as follows: `SELECT [x], [y], v FROM mtable`. The result is an unbounded array with actual size derived from the dimension column expressions `[x]` and `[y]`. The default values of all non-dimensional attributes are inherited from the default values in the original table.

Array Manipulation A cell is given a new value through an ordinary SQL UPDATE statement. A dimension can be used as a bound variable, which takes on all its dimension values (i.e., valid values of this dimension) successively. A convenient shortcut is to combine multiple updates into a single guarded statement. The evaluation order ensures that the first predicate that holds dictates the cell values. The refinement of the array `matrix` is shown in the query below. The cells receive a zero only in the case `x = y`. The results are shown in Figure 1(b).

```
UPDATE matrix SET v = CASE
  WHEN x > y THEN x + y WHEN x < y THEN x - y ELSE 0 END;
```

Arrays can also be updated using INSERT and DELETE statements. Since all cells semantically exists by definition, both operations effectively turn into update statements. The DELETE statement creates holes by assigning a NULL value for all qualified cells. The INSERT statement simply overwrites the cells at positions as specified by the input columns with new values. The results of the following queries are shown in Figure 1(c):

```
INSERT INTO matrix SELECT [x], [y], x * y FROM matrix WHERE x = y;
DELETE FROM matrix WHERE x > y;
```

Array dimension manipulations must be done using ALTER ARRAY statements, since they modify the array schemas. The following statements expand the array `matrix` by 1 in all directions, whose result is shown in Figure 1(f).

```
ALTER ARRAY matrix ALTER DIMENSION x SET RANGE [-1:1:5];
ALTER ARRAY matrix ALTER DIMENSION y SET RANGE [-1:1:5];
```

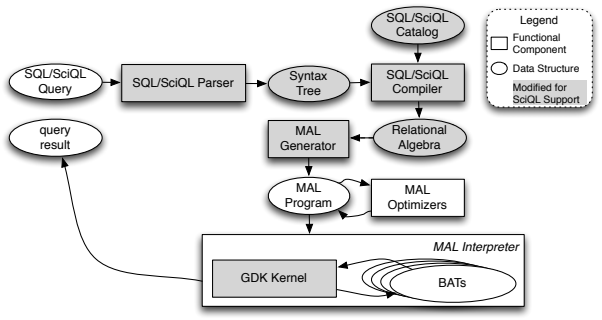


Figure 2: MonetDB architecture extended with SciQL support

Array Tiling A key operation in science applications is to perform statistics on groups. They are commonly identified by a list of attributes or expressions in a GROUP BY clause. This value-based grouping can be extended to *structural grouping* for arrays in a natural way. Large arrays are often broken into smaller pieces before being aggregated or overlaid with a structure to calculate, e.g., a Gaussian kernel function. SciQL supports fine-grained control over breaking an array into possibly overlapping tiles using a slight variation of the SQL GROUP BY clause semantics. Therefore, the attribute list is replaced by a parameterised series of array elements, called *tiles*. The query below tiles the 4×4 array `matrix` in Figure 1(c) with a 2×2 matrix:

```
SELECT [x], [y], AVG(v) FROM matrix
GROUP BY matrix[x:x+2][y:y+2]
HAVING x MOD 2 = 1 AND y MOD 2 = 1;
```

Tiling starts with identifying an anchor point by its dimensional values (e.g., `matrix[x][y]`), which is extended with a list of cell denotations relative to the anchor point (e.g., `matrix[x+1][y]`, `matrix[x][y+1]`, and `matrix[x+1][y+1]`). The tiling operation performs a grouping for every valid anchor point on the actual array dimensions. Unwanted groups are filtered out by specifying extra constrains in the HAVING clause. Figure 1(d) shows the four tiles created. Holes and cells outside the array dimension ranges are ignored by the aggregation functions. Figure 1(e) shows the query results. The value derived from a group aggregation is associated with the dimensional value(s) of the anchor point.

3. IMPLEMENTATION IN MONETDB

SciQL is designed as an extension of SQL that adds arrays as first-class data structures next to tables, and provides relational algebra-like operations on arrays. Thus, several operations on array cell values, e.g., insertion, deletion, selection, projection, join, as well as coercions between tables and arrays, translate directly into relational algebra and MAL with the obvious extensions to handle dimensions. The MonetDB Assemble Language (MAL) is the primary textual interface to the MonetDB kernel. MAL is the target language for all MonetDB query compilers front-ends. More information of MAL can be found in [12]. Other new array-specific operations, such as array creation, dimension modification and tiling, require specific treatment. Figure 2 shows the MonetDB software stack, with the modified components marked in grey. Below we describe how arrays are stored in MonetDB. Details of our implementation can be found in [16].

Array Storage & Creation Adopting the vertically decomposed storage model for relational tables [3], MonetDB/SciQL stores arrays in BATs. Per array, we use one BAT for each dimension and one BAT for each non-dimensional attribute. The creation of persistent database objects has been extended to implement array creation. The main difference with respect to tables is the materialisation of the fixed arrays before their first use. The implementation of this operation introduces two new MAL primitives:

```
command array.series(start:int,step:int,stop:int,N:int,M:int)
        :bat[:oid,:int]
pattern array.filler(cnt:lng, v:any_1):bat[:oid,:any_1];
```

The `array.series()` operator generates a BAT with dimension values based on the dimension range and position in the array definition. The first three arguments are the `start`, `step`, and `stop` parameters of the dimension range. The last two arguments describe the dimension value repetitions inside of a group and of the group as a whole. They are determined by the position of a dimension in the array definition and the sizes of other dimensions. The `array.filler()` operator materialises array cell values with the given value `v`. It creates a BAT with the 'head' columns containing `cnt` consecutive unique OIDs and the 'tail' filled in with the given value. Consider the `CREATE ARRAY` statement in Section 2 that creates a 2D 4×4 `matrix`. The resulting array is stored as the three BATs shown in Figure 3 that are created using the following three MAL statements:

x		y		v	
void	int	void	int	void	int
0	0	0	0	0	0
1	0	1	1	1	0
2	0	2	2	2	0
3	0	3	3	3	0
4	1	4	0	4	0
5	1	5	1	5	0
6	1	6	2	6	0
7	1	7	3	7	0
8	2	8	0	8	0
9	2	9	1	9	0
10	2	10	2	10	0
11	2	11	3	11	0
12	3	12	0	12	0
13	3	13	1	13	0
14	3	14	2	14	0
15	3	15	3	15	0

Figure 3: The `matrix` stored as three BATs after its creation

```
x: array.series(0,1,4,4,1);
y: array.series(0,1,4,1,4);
v: array.filler(16,0);
```

4. DEMONSTRATION OVERVIEW

The demonstration contains two scenario's, which are accompanied by a poster illustrating the main language definitions. The first scenario introduces the audience to the basics of SciQL. The second scenario shows advanced use of SciQL on GeoTIFF image processing. Figure 4 gives an overview of the demo GUI, showing the first scenario. The right-upper text box displays the query to be executed, which can be chosen from the drop-down list. The audience can modify the predefined queries or enter new queries. After execution, the raw query results are displayed in the right-lower text box, and are visualised in the left side.

Scenario I: Conway's Game of Life

The goal of this scenario is to introduce the audience to the possibilities opened up by SciQL in storing and querying array data in RDBMSs. Instead of storing arrays as BLOBs in RDBMSs, and suffering from the limitations and inefficiencies of BLOBs, users can now store arrays directly in an RDBMS side-by-side with the SQL tables. Then, concise SciQL queries can be formulated to express *what* operations users want to apply on the array data, instead of programme *how* they should analyse the data.

The Conway's Game of Life simulates the evolution of cells in a $m \times n$ matrix. All rules of the game are implemented as SciQL queries, e.g., create a game board, initialise the game with living cells, compute the next generation, and clear/resize the board. These queries cover the array definition, array data/dimension manipulation and array tiling features as discussed in Section 2.

The right-lower text box in Figure 4 shows a part of the materialised data of the current game board, as visualised in the left side of Figure 4. A `life` game board is defined as a 2D array (with `x,y`

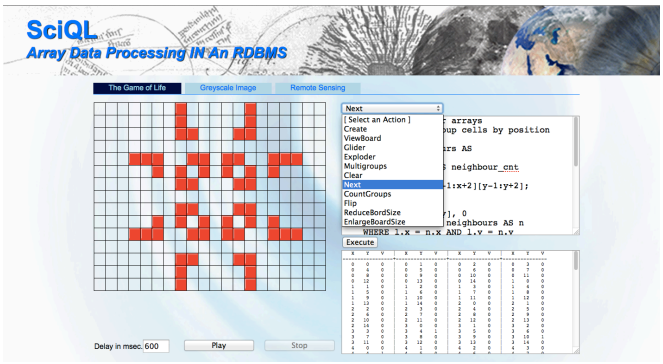


Figure 4: Scenario I: Conway's Game of Life in SciQL

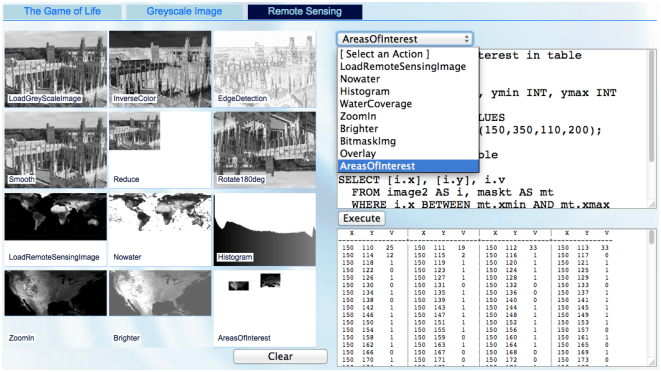


Figure 5: Scenario II: in-DB image processing using SciQL

dimensions) with one integer payload (column `v`) to denote the cell states. Dead and living cells are respectively denoted by 0's and 1's in the `life` array, and visualised by transparent and red-coloured squares. To compute the next generation, a 3×3 tile is created for each cell with this cell as the tile centre. The sum of this tile (subtracting the value of the cell) is the number of living neighbours of this cell, which determines if this cell will be dead or alive in the next generation. In SQL, such query would require a eight-way self-join to associate a cell with all its neighbours.

Scenario II: in database image processing using SciQL

The goal of this scenario is to show SciQL in action using a real-world use case, i.e., how GeoTIFF images are stored as arrays in a DBMS, and how common image processing operations are formulated as SciQL queries and executed directly in the DBMS. We have prepared two GeoTIFF images, a normal grey-scale image of a classic building and a remote sensing image of the earth. The images are loaded into MonetDB using its GeoTIFF Data Vault [9]. Each image is stored as a 2D array with `x,y` dimensions denoting the pixel positions in the image, and an integer column `v` denoting the grey-scale intensities of the pixels.

Figure 5 shows the image processing GUI. A number of queries have been run and their resulting images are displayed in the left column as thumbnails. The first six thumbnails are the results of the queries that have been applied on the grey-scale image, including loading, intensity inversion, building's edges detection, smoothing, resolution reduction and rotation. EdgeDetection is a use case obtained from a TELEIOS user community workshops [14]. It requires computing the differences in colour intensities of each pixel and its upper and left neighbouring pixels. This query can be expressed in a highly concise SciQL query, because SciQL allows addressing the array cells by their relative positions [15].

The second six thumbnails are the results of the queries that have been applied on the remote sensing image. Beside image loading, they include filtering out water areas, compute intensity histogram, zooming in, increasing intensity to make the image brighter and selecting areas of interest given either a bit mask image or rectangular bounding boxes. In the right-lower text box, the raw resulting data of the last executed query, AreasOfInterest, are displayed. This use case demonstrates two main advantages of SciQL. First, instead of retrieving the whole image, one can select only the necessary part of the data for, e.g., displaying, exchanging or further analysis. Second, it shows the combined use of arrays and tables. Here, the bounding boxes of the interested-areas are stored in the table `maskt`. Then, a join between the table and the image array is done to filter out the pixel intensities of those areas.

Finally, the audience can load other GeoTIFF images into the demo and/or propose other image processing operations.

5. CONCLUSIONS

To bridge the gap between the needs of sciences and the available DBMS technologies, we have designed SciQL, a first SQL-based query language with both tables and arrays as the first class citizens. SciQL aims at a true symbiosis of the relational and array paradigm. This demo showcases a proof of concept implementation of SciQL in MonetDB. Users can experience in database array processing through predefined and user defined SciQL queries.

6. ACKNOWLEDGEMENTS

The work reported here is partly funded by the EU-FP7-ICT projects PlanetData (`http://www.planet-data.eu/`) and TELEIOS (`http://www.earthobservatory.eu/`), and the Dutch national project COMMIT (`http://www.commit-nl.nl/`). We would like to thank Milena Ivanova for her work on the GeoTIFF Data Vault.

7. REFERENCES

[1] F. Bancilhon et al., editors. *Building an Object-Oriented Database System, The Story of O2*. Morgan Kaufmann, 1992.

[2] P. Baumann et al. The multidimensional database system RasDaMan. *SIGMOD Rec.*, 27(2):575–577, 1998.

[3] P. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. PhD thesis, UVA, Amsterdam, The Netherlands, May 2002.

[4] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *SIGMOD*, 2010.

[5] J. Gray et al. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.

[6] M. Gyssens and L. V. S. Lakshmanan. A foundation for multi-dimensional databases. In *VLDB*, pages 106–115, 1997.

[7] T. Hey et al., editors. *The Fourth Paradigm: Data-Intensive Scientific Discoveries*. Microsoft Research, 2009.

[8] B. Howe and D. Maier. Algebraic manipulation of scientific datasets. *VLDB J.*, 14(4):397–416, 2005.

[9] M. Ivanova et al. Data vaults: A symbiosis between database technology and scientific file repositories. In *SSDBM*, 2012.

[10] C. G. O. Life. http://en.wikipedia.org/wiki/Conway's_Game_of_Life.

[11] J. Melton et al. SQL/MED: a status report. *SIGMOD Rec.*, 31:81–89, September 2002.

[12] MonetDB, an Open-Source Database System. http://www.monetdb.org/.

[13] M. Stonebraker et al. Requirements for science data bases and SciDB. In *CIDR*, 2009.

[14] TELEIOS, Virtual Observatory Infrastructure for Earth Observation Data. http://www.earthobservatory.eu/.

[15] Y. Zhang et al. Sciql, bridging the gap between science and relational dbms. In *IDEAS2011*, page 10, New York, NY, USA, 2011. ACM.

[16] Y. Zhang et al. An implementation of ad-hoc array queries on top of MonetDB. EU Project TELEIOS (FP7-257662), D5.1, 2012.