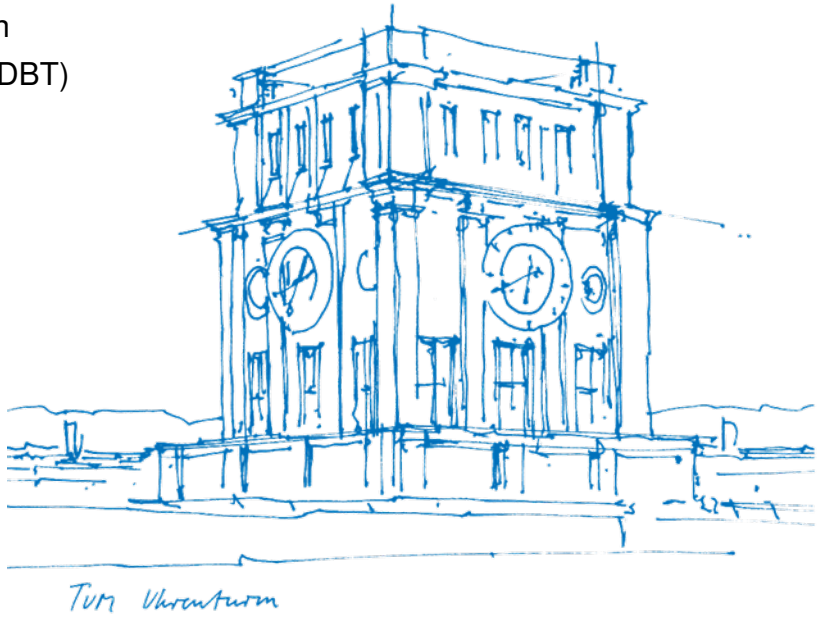
Chair III: Database Systems
Chair XXV: Data Science and Engineering
Department of Informatics
Technical University of Munich

ArrayQL Integration into Code-Generating Database Systems

Maximilian E. Schüle, Tobias Götz, Alfons Kemper, Thomas Neumann
25th International Conference on Extending Database Technology (EDBT)
29th March-1st April, 2022, Edinburgh, UK



Motivation: In-Database Machine Learning

ML2SQL

MLearn: Example 1 TRANSLATE

```
A =
  [[1.1,0.98,87.3,3],
  [0.1,3.15,42.05,3.3],
  [100.5,26.8,10.1,225.1],
  [1097.5,23000,10.1,24850.1]]
X = A[: , 0:2]
y = A[: , 3]
bias[1,len(X,0)] : 1
X = (bias::X.T).T
Xt = X.T
w = (Xt*X)^(-1) * Xt * y
print '%' , w
```

Target Language: Python EXECUTE

```
import numpy as np
  def main():
      A = np.array([ np.array([ 1.1,0.98,87.3,3]),
np.array([0.1,3.15,42.05,3.3]),np.array([ 100.5,26.8,
10.1,225.1]),np.array([1097.5,23000,10.1,24850.1])])
      X = A[ :,0:2 +1]
      y = A[ :,3:3 +1]
      bias = np.full( ( 1,np.size(X ,0 )), 1)
      X = (np.append(bias,X.T, axis=0)).T
      Xt = X.T
      w = np.dot( np.dot( np.linalg.matrix_power(
(np.dot( Xt,X)), (-1)),Xt),y)
      print( '{}'.format(  w))
  if __name__ == "__main__": main()
```

```
$ python < main.py

[[ 0.12238861]
[ 1.9759119 ]
[ 0.98614977]
[-0.00300479]]
```

Motivation: In-Database Machine Learning

ML2SQL

MLearn: Example 1 TRANSLATE

```
1  A =
2    [[1.1,0.98,87.3,3],
3    [0.1,3.15,42.05,3.3],
4    [100.5,26.8,10.1,225.1],
5    [1097.5,23000,10.1,24850.1]]
6  X = A[: , 0:2]
7  y = A[: , 3]
8  bias[1,len(X,0)] : 1
9  X = (bias::X.T).T
10 Xt = X.T
11 w = (Xt*X)^(-1) * Xt * y
12 print '%' , w
```

```
$ python < main.py

[[ 0.12238861]
[ 1.9759119 ]
[ 0.98614977]
[-0.00300479]]
```

- **ML2SQL**: Translate a universal machine learning language into Python or SQL
- **Problem**: first converted into arrays, expensive
- **Goal**: work directly on arrays
- **Solution**: implement ArrayQL syntax draft presented at XLDB 2012 within a relational, code-generating database system

Structure

ArrayQL Algebra
completed syntax draft
added statements for joins

Umbra Integration
First implementation of ArrayQL
Relational representation

Relational Array Representation

ArrayQL Algebra

- **Data Definition Language**: array creation as new array or out of an existing array

```
CREATE ARRAY m (i INTEGER DIMENSION [1:2], j INTEGER DIMENSION [1:2], v INTEGER);
CREATE ARRAY m2 FROM SELECT [i], [j], v FROM m;
```

- **Data Query Language**
  - `SELECT`: indices for the dimensions (in brackets), arithmetic expressions as attributes.
  - `FROM`: specifies the source array.
  - `WHERE`: filters each entry by a predicate.
  - `GROUP BY`: addresses the dimensions preserved after an aggregation.

```
SELECT [i], SUM(v)+1 FROM m WHERE v>0 GROUP BY i
```

- **New**: temporary arrays and joins

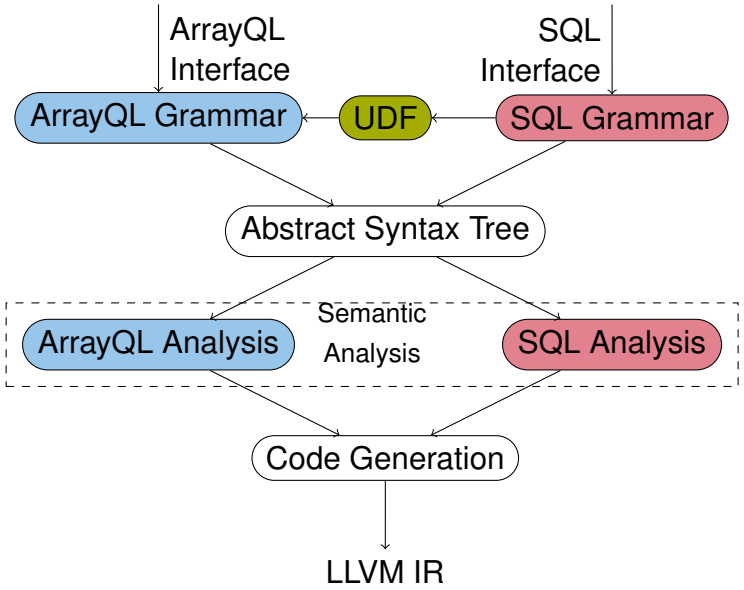
```
WITH ARRAY temp AS (SELECT [i] as j, SUM(v+1) FROM m[j] WHERE v>0 GROUP BY j) SELECT * FROM temp;
SELECT [i] as i, [j] as j, v, v2 FROM m[i+2,j+2] JOIN m2[i-2,j-2];
```

- **New**: **Data Modification Language**

```
UPDATE ARRAY m [1][1] (select [2][1],v FROM m);
UPDATE ARRAY m [1][1:2] (values (1),(2));
```

Umbra Integration

Umbra Integration



- Umbra: a code-generating database system (producer-consumer concept)
- SQL $\rightarrow$ abstract syntax tree $\rightarrow$ semantic analysis $\rightarrow$ operator plan
- Each user-defined function: a separate grammar file/semantic analysis
- Code compiled together
- Only schema known during compile-time $\rightarrow$ relational representation needed

Relational Array Representation

Relational Array Representation

Tabular Representation

$$\begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}$$

Relational Representation

rowno.	col1	...	coln
1	$a_{1,1}$	...	$a_{1,n}$
...	...	...	...
m	$a_{m,1}$	...	$a_{m,n}$

x	y	value
1	1	$a_{1,1}$
...	...	...
1	n	$a_{1,n}$
...	...	...
m	1	$a_{m,1}$
...	...	...
m	n	$a_{m,n}$

```
CREATE ARRAY m (
 i INTEGER DIMENSION [1:2],
 j INTEGER DIMENSION [3:4],
 v INTEGER);
```

⟶

x	y	v
1	3	*NULL*
2	4	*NULL*

- **Goal**: Uniform representation (access from SQL *and* an array query language)
- Either coordinate list (*relational representation*) or tables with row order (*tabular representation*)
- Relational representation: saves memory (sparse arrays) and primitive data types are sufficient even for more than two dimension
- n-dimensional array with m values per cell
  $\rightarrow$ table with $n+m$ attributes
- Attributes for the indices are unique (the primary key)
- ArrayQL: *bounding box*, *validity map* and *content*
  - creation: insert tuples defining the bounding box
  - an entry within the bounding box is valid: if it exists and at least one attribute is not `NULL`.

Relational Array Representation: Mapping to Relation Algebra

Operator	Input	Output	Relational Algebra
apply	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}, f \in (\mathbb{R} \to \mathbb{R})$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\pi_{i_1,\ldots i_n,f(v)}(a)$
combine	$\mathfrak{a}, \mathfrak{b} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$a \fullouterjoin_{a.i_1=b.i_1 \wedge \cdots \wedge a.i_n=b.i_n} (b)$
i. dim. join	$\mathfrak{a}, \mathfrak{b} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$(\mathbb{R}, \mathbb{R})^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$a \bowtie_{a.i_1=b.i_1 \wedge \cdots \wedge a.i_n=b.i_n} (b)$
fill	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\ldots 0_{\lvert a.i_1 \rvert,\ldots,\lvert a.i_n \rvert} \cdots$
filter	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}, p \in (\mathbb{R} \to \mathbb{B})$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\sigma_{p(v)}(a)$
rebox	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}, i_1^l, i_1^u, \ldots, i_n^l, i_n^u \in \mathbb{I}$	$\mathbb{R}^{i_1^u - i_1^l \times \cdots \times i_n^u - i_n^l}$	$\sigma_{i_1^l \le i_1 \le i_1^u \wedge \cdots \wedge i_n^l \le i_n \le i_n^u}(a)$
reduce	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}, f \in (\mathbb{R}^{\lvert i_n \rvert} \to \mathbb{R})$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_{n-1} \rvert}$	$\gamma_{i_1,\ldots,i_n,f(v)}(a)$
rename	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\rho(a)$
shift	$\mathfrak{a} \in \mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}, i'_1, \ldots, i'_n \in \mathbb{I}$	$\mathbb{R}^{\lvert i_1 \rvert \times \cdots \times \lvert i_n \rvert}$	$\pi_{i_1+i'_m,\ldots,i_n+i'_n,v}(a)$

- map nine ArrayQL operators to SQL operators considering the underlying schema
- one relation $a \subseteq \mathbb{I}^n \times \mathbb{R}^m$ with schema $sch(a) = \{\underline{i_1, \ldots, i_n}, r_1, \ldots, r_m\}$ represents one n-dimensional array $\mathfrak{a} \in (\mathbb{R}^m)^{\lvert i_1 \rvert \times \ldots \times \lvert i_n \rvert}$ with m attributes of domain $\mathbb{R}$ as content.
- coordinates $(i_1, \ldots, i_n) \subseteq \mathbb{I}^n$ form the primary key and delimit the bounding box.
- validity map of an array $\mathfrak{a}$: set of indices $d_a \subseteq \mathbb{I}^n$ of valid entries.

ArrayQL for Linear Algebra

$$(a_{ij}) = A \in \mathbb{R}^{2\times 3}$$

```
CREATE ARRAY A (
i INTEGER DIMENSION [1:2],
j INTEGER DIMENSION [1:3],
a FLOAT)
```

$$(a_{ij}) = A$$

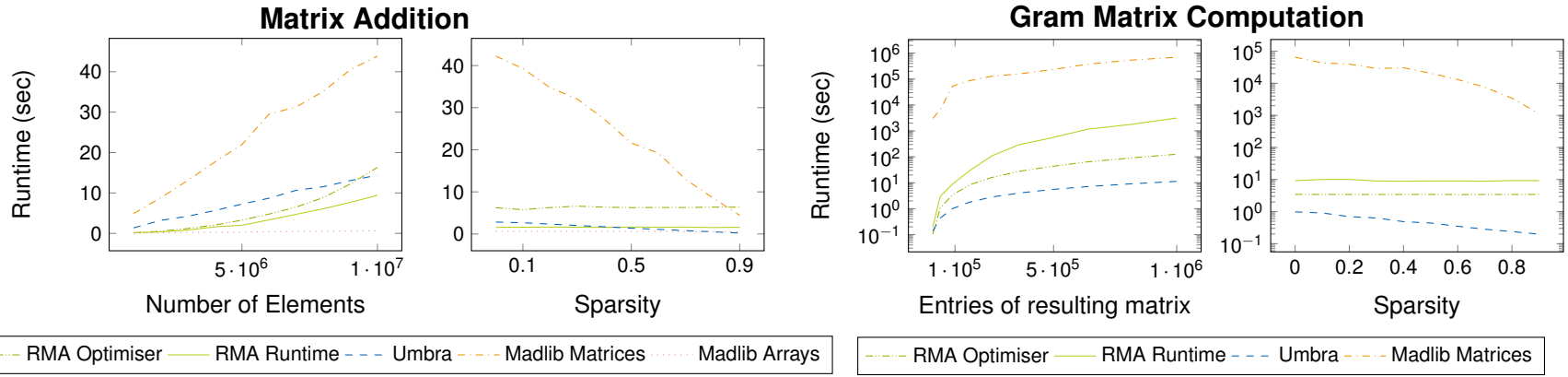
```
SELECT [i],[j], A.a FROM A
```

- Bounding box defines the size of a matrix, the attributes determine the field of which each entry is a member
- Scalar operations: part of the `select`-clause; matrix operations belong to the `from`-clause
- sparse matrices: values of non-existing entries are assumed to be zero (keyword `filled` behind `select`)
- Short-cuts for matrix operations as part of the from clause
- Linear regression: $\vec{w} = (A'^T A')^{-1} A'^T \vec{y}$

```
SELECT FILLED [i],[j],* FROM ((a^T * a)^-1*a)^T*y
```

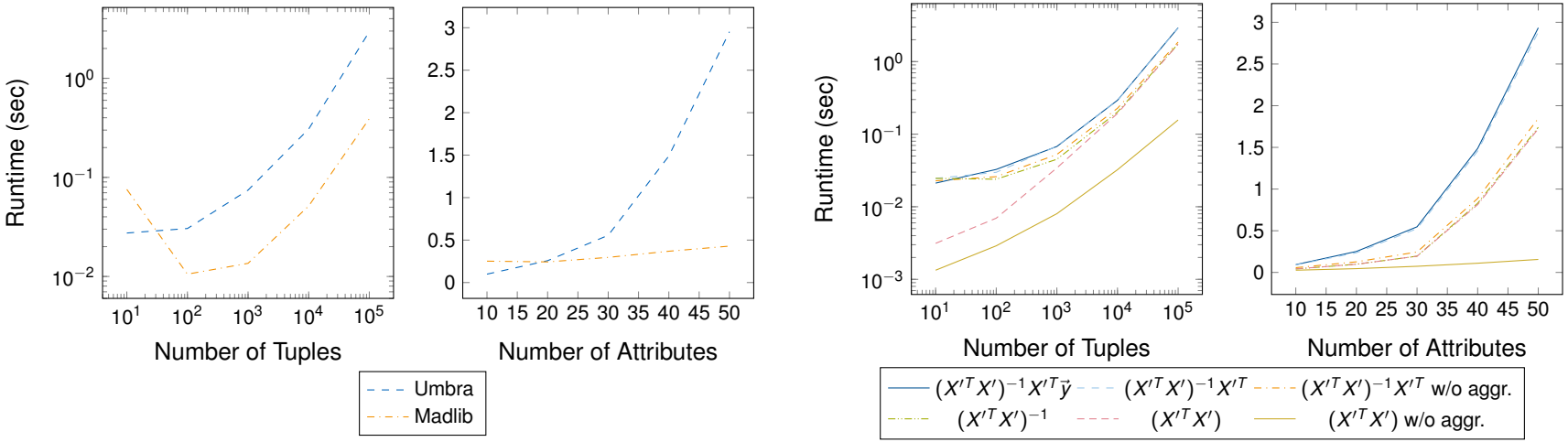
Evaluation

Evaluation: Matrix Algebra



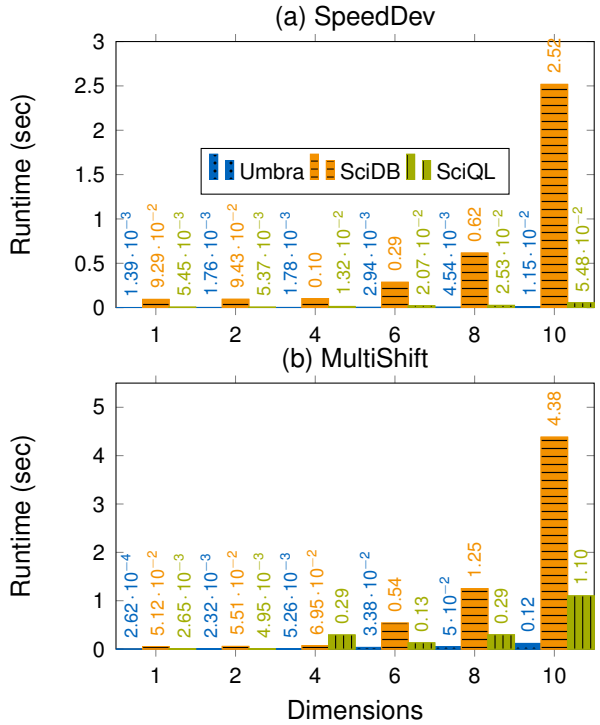
- **System**: Ubuntu 20.04 LTS, six Intel Core i7-3930K CPUs at 3.20GHz, 64 GB main-memory
- Varying either the number of elements in a dense array or the sparsity
- Varying the sparsity: gram matrix XX^T on a resulting matrix with 90000 entries, matrix addition $X + X$ with 10^6 elements
- With increasing size, ArrayQL computes the matrix sum faster than RMA
- MADlib matrices and Umbra benefit from sparse matrices (relational representation)
- Relational representation: comparable performance to existing database extensions for linear algebra

Evaluation: Linear Regression



- **System**: Ubuntu 20.04 LTS, six Intel Core i7-3930K CPUs at 3.20GHz, 64 GB main-memory
- synthetic dataset: ArrayQL using linear algebra only vs. MADlib (dedicated table function)
- Most of the time: aggregation part of each matrix product (summation)
- ability to solve numerical problems when suitable table functions are available

Evaluation: Taxi Data

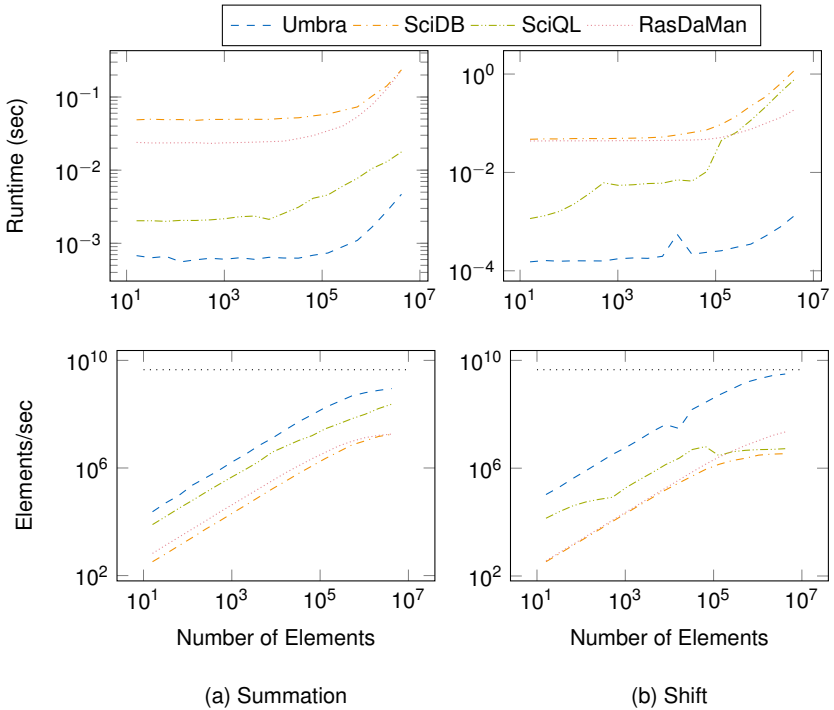



```
SELECT MAX(3600.0*(tmp3.v3-tmp1.v)) as speed
FROM (SELECT pickup_day, AVG(trip_distance/total_duration) as v
 FROM taxiData WHERE total_duration<>0 GROUP BY pickup_day)
 as tmp1, (SELECT AVG(trip_distance/total_duration) as v3
 FROM taxiData WHERE total_duration <> 0) as tmp3;

SELECT [pickup_day] as a, ..., * FROM taxiData[a+1, ...]
```

- **System**: Ubuntu 20.04 LTS, six Intel Core i7-3930K CPUs at 3.20GHz, 64 GB main-memory
- impact of involved dimensions on aggregations
- *SpeedDev*: maximum deviation of the average speed per day compared to the average speed of the whole dataset
- *MultiShift*: shifts all array's dimensions.
- scales linearly with an increasing number of dimensions.

Evaluation: Random Data



(a) Summation (b) Shift

- **System**: Ubuntu 20.04 LTS, six Intel Core i7-3930K CPUs at 3.20GHz, 64 GB main-memory
- two-dimensional arrays with synthetic data and an increasing number of elements.
- summation and shifting the indices
- Umbra the fastest system when performing aggregates
- shifting slows down the performance as all indices have to be changed
- upper constant lines: the maximum throughput of $4.5 \cdot 10^9$ elements per second

Conclusion

- First ArrayQL integration (as another query interface and addressable inside SQL)
- Integration into a code-generating database system
- Introduced a relational array model and an ArrayQL operator translation to relational algebra
- In summary, ArrayQL in Umbra benefits from sparse matrices as well as the performance of an in-memory database system
- Future: ArrayQL to SQL translator

```
CREATE FUNCTION exampletable() RETURNS TABLE (x INT, y INT, v INT) LANGUAGE 'arrayql' AS
'SELECT␣[x],␣[y],␣v␣FROM␣m';
CREATE FUNCTION exampleattribute() RETURNS INT[][] LANGUAGE 'arrayql' AS
'SELECT␣[x],␣[y],␣v␣FROM␣m';
```

Listing 1: ArrayQL as part of a user-defined function returns either an SQL table or an SQL array.

Thank you for your attention!