TriaClick Architectural Overview

By Athanassios I. Hatzis
March, 2019

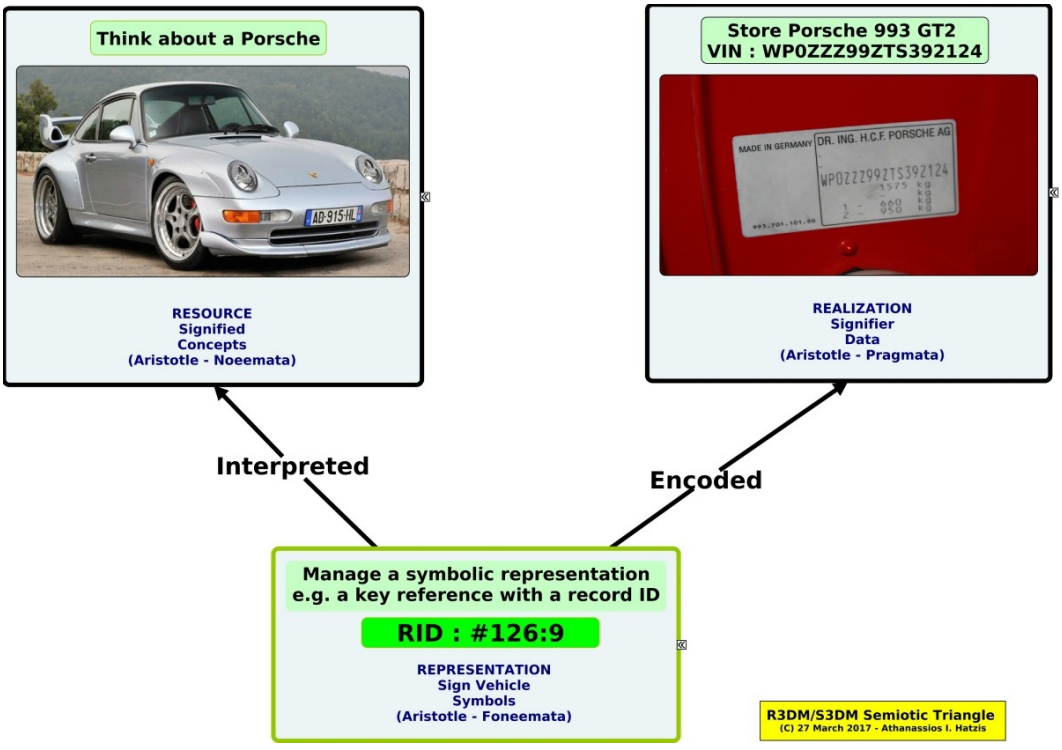

Table of Contents

At A Glance

TriaClick is a specific implementation of TRIADB multi-perspective database management framework on top of MariaDB and ClickHouse DBMS. It is founded on the principles of R3DM/S3DM associative, semiotic, hypergraph technology. This paper addresses specifications of this technology, offers an overview of the software/database engineering problem we try to solve and highlights the key differentiating factors from other solutions. The target audience for this paper consists of IT professionals, preferably with technical expertise in DBMS and data analytics software.

Overview of Target

From TRIADB engineering point of view, storage/graph engines, data modeling, ACID/BASE operations, query language, clustering, migration, integration and eventually analytics should be managed independently. These have to be switchable components in a standardized DBMS architecture. And this is the main reason we selected python as the TRIADB programming language. Python is a glue language and therefore it is relatively easy to build a generic API at a high level that communicates with many different components and unifies them as a solid, easy to use and efficient database management framework.

From the end-user (non technical user) point of view, results should be taken intuitively, self-serviced, with sub-second response in a GUI environment, no matter what API, query language you have underneath. Even the Velocity, Volume and other Vs of data should not be a concern for the end user. And here lies the second reason we chose Python and Jupyter notebooks for the graphical environment. That style of programming allows to built interactively and dynamically a graphical user interface and to change its behaviour and appearence using python commands.

Data Modeling at the Heart of Data Analysis

It is not a coincidence that the roots of this project date back to 2009, when the NoSQL movement started. At that time the author also started his research and thorough investigation of existing data models. Data modeling is at the heart of data analysis process. At a lower level you have to deal with storage, extraction, transformation and loading. At a higher level you manage integration, correlation, aggregation, exploratory data analysis and descriptive statistics. TriaClick extends in both directions to embrace and unify physical, logical and conceptual database layers in a unique and profound way.

Components of TriaClick Framework

From an abstract point of view **TriaClick software components correspond to Computer Science assertion (ABox) and terminological (TBox) components. The main difference from other systems and data modeling specifications is that these components are completely separable**. In the current implementation

MariaDB stores data dictionary information, **i.e. TriaClick metadata objects** and ClickHouse data storage engines are used for processing and querying data, **i.e. TriaClick instances**.

But most important is that representational forms of these two components are interrelated and logically connected through **the cornerstone of our technology, which is semiotic in nature**. TriaClick framework is based on the theory of the semiotic triangle also known as the triangle of meaning or the triangle of reference. In TriaClick **everything is represented and stored with a 3D numerical key vector**.

For example the key (2, 100, 9) represents the attribute "*part name*" (9) of the "*Part*" Entity, in one of the models, "*SupplierPartCatalog*" (100), that exists under TriaClick data modeling system (2). In other words key dimensions are used as hierarchical containers to capture information about the hierarchical relationships in data structures that are inherent in nature (**Fig. 1**). The modeling system has many models. One model has many entities and one entity has many attributes.

```
In[10]: dmf.get(2, 100, None, what='metadata', out='dataframe')
Out[10]:
    dim4  dim3  dim2                  cname       alias ntype    vtype  counter
0      2   100     0  SupplierPartCatalog_ALL  00121_0001    DM      NaN       16
1      2   100     1  SupplierPartCatalog_ALL     SPC_ALL   ENT      NaN        0
2      2   100     2                   prtID       prtID  ATTR   UInt16        0
3      2   100     3                   supID       supID  ATTR   UInt16        0
4      2   100     4                  catChk      catChk  ATTR   String        0
5      2   100     5                 catDate     catDate  ATTR     Date        0
6      2   100     6                catPrice    catPrice  ATTR  Float32        0
7      2   100     7                catTotal    catTotal  ATTR   UInt16        0
8      2   100     8                prtColor    prtColor  ATTR   String        0
9      2   100     9                 prtName     prtName  ATTR   String        0
10     2   100    10                 prtUnit     prtUnit  ATTR   String        0
11     2   100    11              prtWeight   prtWeight  ATTR  Float32        0
12     2   100    12             supAddress  supAddress  ATTR   String        0
13     2   100    13                supCity     supCity  ATTR   String        0
14     2   100    14             supCountry  supCountry  ATTR   String        0
15     2   100    15                supName     supName  ATTR   String        0
16     2   100    16              supStatus   supStatus  ATTR    UInt8        0
```

Figure 1: Data Model Metadata

In the same example each distinct value of the "part name" attribute is also represented with a key, e.g. : (100, 9, 4) is a specific "Fire Hydrant Cap" instance with model, attribute and value dimensions respectively (**Fig. 2**).

```
In[22]: HyperCollection(datmodel, pattern='prtName').cql.Over(value=True, sel=True, pos=True).Where('sel=1').Execute(index='dim3, dim2, dim1').Result
Out[22]:
                                  prtName  sel  pos
dim3 dim2 dim1
100  9    1                   7 Segment Display    1    1
          2                  Acme Widget Washer    1    1
          3       Anti-Gravity Turbine Generator    1    1
          4                    Fire Hydrant Cap    1    1
          5     I Brake for Crop Circles Sticker    1    1
          6    Left Handed Bacon Stretcher Cover    1    1
          7                   Smoke Shifter End    1    1
```

Figure 2: Distinct values of an attribute, i.e. items of a HyperAtom collection

TriaClick Hypergraphs

TriaClick is developed in Python and combines two database interfaces, PonyOrm and Marilyn ClickHouse Python Driver. On top of these interfaces there is the *DataManagementFramework* class that implements an object-oriented console API between databases and TriaClick associative semiotic hypergraph data management framework.

Two hypergraph systems are created; one that handles **TriaClick metadata objects**; and another for **TriaClick instances.** In analogy to the components of TriaClick these are also distinct and completely separable hypergraphs.

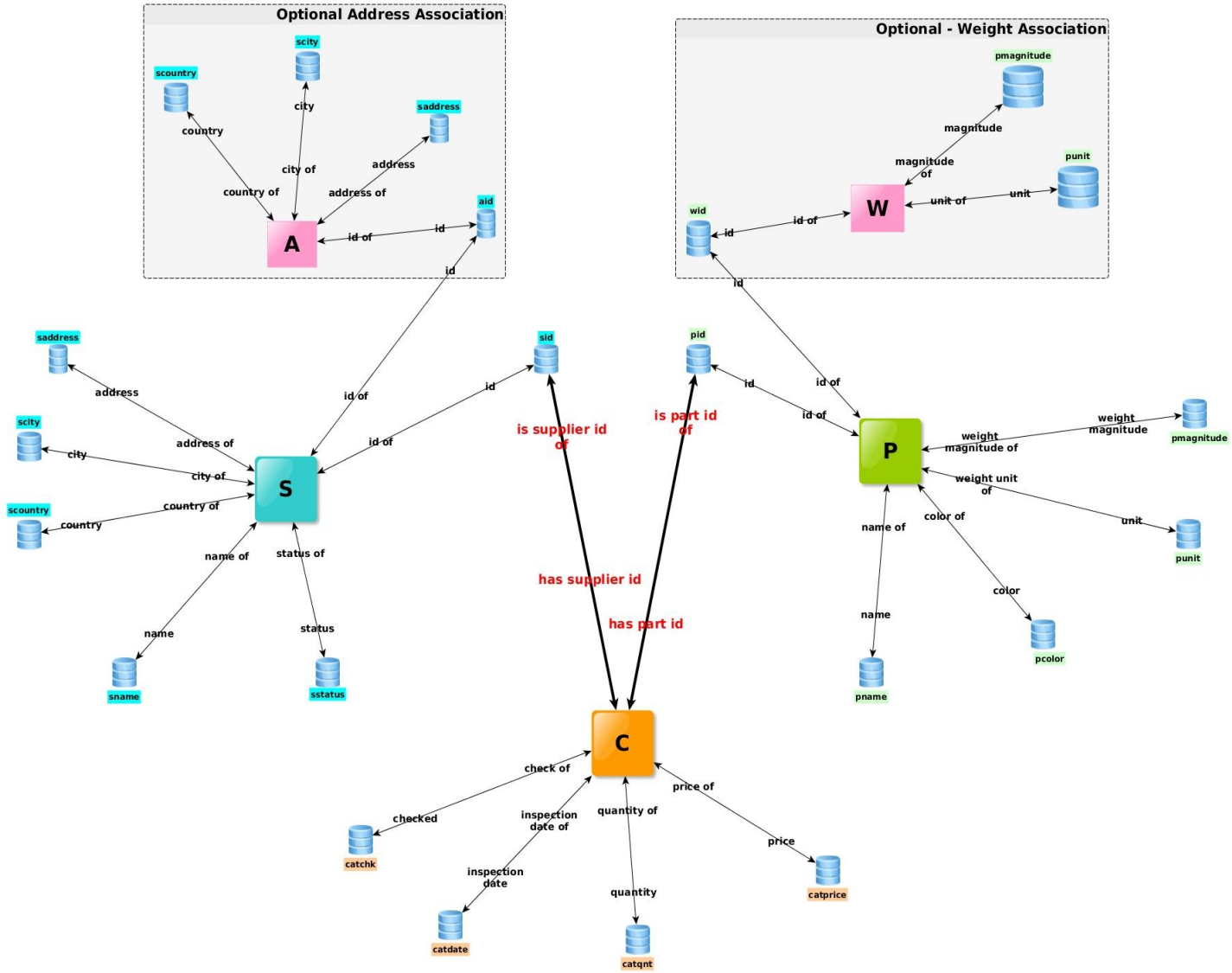

Figure 3: Supplier-Catalog-Part hypergraph data model with bidirectional links

I. **In data dictionary hypergraph (Fig. 3) we have**:

a) **HyperEdges** to represent data model entities, i.e. things with distinct and independent existence and data resources, e.g. TSV/CSV flat files.

b) **HyperNodes** to represent data model attributes, i.e. properties of entities and fields of a data resource, e.g. columns of a file/table.

c) **HyperLinks** to represent bidirectional edges with a Many-To-Many directed relationship from tail nodes (e.g. Entities) to head nodes (e.g. Attributes).

II. **In associative data hypergraph (Fig. 4) we have**:

a) **HyperBonds** to represent instances of associations, i.e. the equivalent data sturucture of a data record (relational tuple).

b) **HyperAtoms** to represent instances of data items (e.g. data values).

c) **HyperLinks** to represent bidirectional edges with a **Many-To-Many directed relationship** from tail nodes (i.e. HyperBonds) to head nodes (i.e. HyperAtoms).

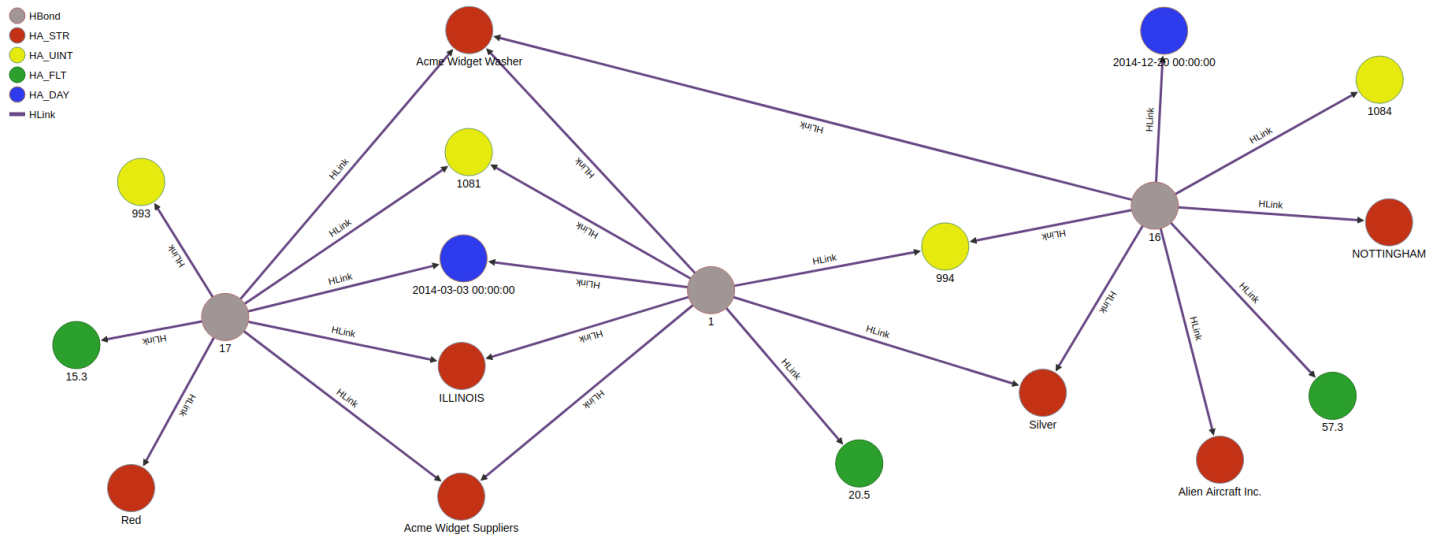

*Figure 4: Part item (***Acme Widget Washer***) with three Catalog entries (grey hyperbonds), two for one Supplier (**Acme Widget Suppliers**) and one for another Supplier (**Alien Aircraft**)*

Quick comparison with graph network data models

Here is a rundown of some of the most important similarities between TriaClick and other Graph Database systems:

- Hypergraph is a generalization of graph network in which an edge can join any number of vertices

- Bidirectional links are similar to edges of a property graph network
- HyperBond/HyperEdge and HyperAtom/HyperNode are similar to RDF subject and object respectively
- Nodes of the graph are represented with logical identifiers (3D numerical keys). Each node knows its type and is directly linked to its neighbours.

The key difference is that in our associative, semiotic, hypergraph technology **the construction of queries or paths for traversal and filtering of data are not dependent on labeling of data, e.g. labeled nodes/edges.**

TriaClick User Environment

The user environment in TriaClick plays the equivalent role of a namespace and a database in other DBMS. Each user of TriaClick has a corresponding database user and default database in both MariaDB and ClickHouse in which he can store data. Since everything in TriaClick is handled with numerical vectors as identifiers (see Fig.1 and Fig.2), **name collisions are avoided and construction of queries and other execution routines can be generalized for any user in any data model, database because they are not dependent on namespace**. From that perspective TriaClick can be considered as a generic database programmable environment.

TriaClick Systems

In this environment *DataModel* and *DataResource* modules are built on top of *DataManagementFramework*. The first implements an interface to construct data models with entities and attributes, to create ClickHouse *MergeTree* table engines for TriaClick hypergraph and to fetch back objects by specifying name patterns or numerical key vectors. The second implements a similar interface to manage data resources. Each data resource can be a data set of many related items, e.g. TSV/CSV flat files, hypergraph files, data model files etc.

```
In[16]: dmf = DataResource(mysql_conn, clickhouse_conn, name='void', alias='void')
In[17]: dmf.get(what='overview', out='dataframe', index='dim4, dim3, dim2')
Out[17]:
                                        cname       alias ntype  vtype  counter
dim4 dim3 dim2
1    0    0              Data  Resources System         DRS   DRS    NaN        3
     121  0                  TRIADB Data Models   TRIADB_DM    DS    NaN        1
     242  0                 Network Hypergraphs      HGraph    DS    NaN        0
     363  0     Supplier_Part_Catalog Combined     SPC_ALL    DS    NaN       16
2    0    0                  Data  Models System         DMS   DMS    NaN        1
     100  0      SupplierPartCatalog_ALL Model  SPC_ALL DM    DM    NaN       16
3    0    0                  Hyper Links  System         HLS   HLS    NaN        2
     77   0                  HyperEdge/HyperNode       HE_HN   HLT    NaN       30
     154  0                  HyperBond/HyperAtom       HB_HA   HLT    NaN        0
```

Figure 5: TriaClick Systems Hierarchy

Both *DataModel* and *DataResource* modules correspond to TriaClick systems at the highest level of framework hierarchy **(Fig. 5).** These two systems together with the *HyperLinks* system are responsible for managing TriaClick metadata dictionary.

TriaClick HyperStructures

The other component of TriaClick, which is based on ClickHouse, is managed and driven with the *HyperStructures* module. The console interface here is implemented with the *HyperCollection* and *AssociativeSet* classes. There are four types of *HyperCollections* in two categories, those that allow duplicates (*Entity*, *Attribute collections*) and those that are sets of distinct elements (*HyperBond*, *HyperAtom collections)*.

A *HyperBond* collection represents a set of *Entity* items (instances) and the *HyperAtom* collection represents the domain set of values for an Attribute (**Fig. 2**). There is a Many-To-Many relationship between HyperBonds and HyperAtoms. This relationship and attribute values for specific data types are stored in ClickHouse MergeTree table engines and these construct the initial state of TriaClick associative semiotic hypergraph engine.

TriaClick AssociativeSet

Hyperbonds and *Hyperatoms* construct associations (**Fig. 6**). An association is the basic construct of *AssociativeSet*, also called *AssociativeEntitySet* because it is always bounded to a single *Entity*. There is a direct analogy of the relational data model with TriaClick data model:

Tuple	**:** An ordered list of data values	===> **Association**	**:**	A set of HyperAtoms that share a single HyperBond
Relation	**:** A set of tuples	===> **AssociativeSet**	**:**	A set of associations
Body	**:** Tuples of ordered values	===> **Body**	**:**	Named tuples, Set of HyperBonds, Sets of HyperAtoms
Heading	**:** Tuple of ordered attribute names	===> **Heading**	**:**	Attributes of an Entity in a TriaClick Data Model

As we can see from the example that is illustrated in Fig. 6, **missing values are ommitted from associations**. **Another important difference of association from the relational tuple is that it does not require a heading and/or ordering of its items.**

TriaClick Chain Query Language (CQL)

AssociativeSet and *HyperCollection* objects take as property a CQL object. CQL is a Python generative class that implements method chaining. TriaClick query is constructed by chaining various operators (methods) and passing required or optional arguments.

For example (**Fig. 6**.), define a projection over the Supplier-Part-Catalog associative set (SPC object). For each association display both *HyperAtom* keys and *HyperAtom* values together with their *HyperBond* key. Use this key as the index of Pandas dataframe. In TriaClick this is an one line command:

SPC.cql.Over(key='**$key**', projection=**aliases**, out='**assocs**', format='**key:value**').Execute(index='**$key**').Result

and the result set can be returned in the form of associations (keys, values, or key-values) or in the classic relational tuples format where missing values are replaced with Python None value, the equivalent of SQL *null* ***(Fig. 6)***.

```
In[18]: # Display Associations by defining a projection with attribute aliases
   ...: aliases = 'supName, supCity, catPrice, catDate, prtName, prtColor'
   ...: SPC.cql.Over(key='$key', projection=aliases, out='assocs', format='key:value').Execute(index='$key').Result
   ...:
Out[18]:
                                                 hatom_keys                                                                                                       hatom_values
$key
(1, 1)    ((13, 1), (8, 6), (15, 1), (9, 2), (6, 8), (5, 1))                            (ILLINOIS, Silver, Acme Widget Suppliers, Acme Widget Washer, 20.5, 2014-03-03)
(1, 2)                   ((13, 1), (15, 1), (9, 5), (6, 8))                                  (ILLINOIS, Acme Widget Suppliers, I Brake for Crop Circles Sticker, 20.5)
(1, 3)           ((13, 1), (8, 3), (15, 1), (9, 1), (6, 14))                          (ILLINOIS, Green, Acme Widget Suppliers, 7 Segment Display, 75.19999694824219)
(1, 4)           ((13, 1), (8, 2), (15, 1), (9, 3), (6, 15))               (ILLINOIS, Cyan, Acme Widget Suppliers, Anti-Gravity Turbine Generator, 124.2300033569336)
(1, 5)           ((13, 1), (8, 4), (15, 1), (9, 3), (6, 15))            (ILLINOIS, Magenta, Acme Widget Suppliers, Anti-Gravity Turbine Generator, 124.2300033569336)
(1, 6)    ((13, 1), (8, 5), (15, 1), (9, 4), (6, 4), (5, 2))              (ILLINOIS, Red, Acme Widget Suppliers, Fire Hydrant Cap, 11.699999809265137, 2014-09-10)
(1, 7)   ((13, 1), (8, 5), (15, 1), (9, 6), (6, 10), (5, 3))  (ILLINOIS, Red, Acme Widget Suppliers, Left Handed Bacon Stretcher Cover, 36.099998474121094, 2014-12-20)
(1, 8)   ((13, 1), (8, 1), (15, 1), (9, 7), (6, 11), (5, 3))             (ILLINOIS, Black, Acme Widget Suppliers, Smoke Shifter End, 42.29999923706055, 2014-12-20)
(1, 9)    ((13, 4), (8, 5), (15, 3), (9, 4), (6, 3), (5, 1))                  (OREGON, Red, Big Red Tool and Die, Fire Hydrant Cap, 7.949999809265137, 2014-03-03)
(1, 10)   ((13, 4), (8, 4), (15, 3), (9, 3), (6, 1), (5, 2))   (OREGON, Magenta, Big Red Tool and Die, Anti-Gravity Turbine Generator, 0.550000011920929, 2014-09-10)
(1, 11)   ((13, 4), (8, 5), (15, 3), (9, 6), (6, 7), (5, 2))               (OREGON, Red, Big Red Tool and Die, Left Handed Bacon Stretcher Cover, 16.5, 2014-09-10)
(1, 12)           ((13, 2), (8, 3), (15, 4), (9, 1), (6, 2))                                                 (MADRID, Green, Perfunctory Parts, 7 Segment Display, 1.0)
(1, 13)           ((13, 2), (8, 5), (15, 4), (9, 4), (6, 5))                                                  (MADRID, Red, Perfunctory Parts, Fire Hydrant Cap, 12.5)
(1, 14)                  ((13, 3), (15, 2), (9, 5), (6, 9))                  (NOTTINGHAM, Alien Aircaft Inc., I Brake for Crop Circles Sticker, 22.200000762939453)
(1, 15)  ((13, 3), (8, 5), (15, 2), (9, 4), (6, 12), (5, 3))                  (NOTTINGHAM, Red, Alien Aircaft Inc., Fire Hydrant Cap, 48.599998474121094, 2014-12-20)
(1, 16)  ((13, 3), (8, 6), (15, 2), (9, 2), (6, 13), (5, 3))             (NOTTINGHAM, Silver, Alien Aircaft Inc., Acme Widget Washer, 57.29999923706055, 2014-12-20)
(1, 17)   ((13, 1), (8, 5), (15, 1), (9, 2), (6, 6), (5, 1))                (ILLINOIS, Red, Acme Widget Suppliers, Acme Widget Washer, 15.300000190734863, 2014-03-03)
In[19]: # Display Tuples
   ...: SPC.cql.Over(projection=aliases, out='tuples').Execute(index='dim3, dim2, dim1').Result
Out[19]:
                              supName     supCity    catPrice     catDate                           prtName prtColor
dim3 dim2 dim1
100  1    1     Acme Widget Suppliers    ILLINOIS   20.500000  2014-03-03                Acme Widget Washer   Silver
          2     Acme Widget Suppliers    ILLINOIS   20.500000        None   I Brake for Crop Circles Sticker     None
          3     Acme Widget Suppliers    ILLINOIS   75.199997        None                 7 Segment Display    Green
          4     Acme Widget Suppliers    ILLINOIS  124.230003        None    Anti-Gravity Turbine Generator     Cyan
          5     Acme Widget Suppliers    ILLINOIS  124.230003        None    Anti-Gravity Turbine Generator  Magenta
          6     Acme Widget Suppliers    ILLINOIS   11.700000  2014-09-10                  Fire Hydrant Cap      Red
          7     Acme Widget Suppliers    ILLINOIS   36.099998  2014-12-20  Left Handed Bacon Stretcher Cover      Red
          8     Acme Widget Suppliers    ILLINOIS   42.299999  2014-12-20                 Smoke Shifter End    Black
          9      Big Red Tool and Die      OREGON    7.950000  2014-03-03                  Fire Hydrant Cap      Red
          10     Big Red Tool and Die      OREGON    0.550000  2014-09-10    Anti-Gravity Turbine Generator  Magenta
          11     Big Red Tool and Die      OREGON   16.500000  2014-09-10  Left Handed Bacon Stretcher Cover      Red
          12        Perfunctory Parts      MADRID    1.000000        None                 7 Segment Display    Green
          13        Perfunctory Parts      MADRID   12.500000        None                  Fire Hydrant Cap      Red
          14       Alien Aircaft Inc.  NOTTINGHAM   22.200001        None   I Brake for Crop Circles Sticker     None
          15       Alien Aircaft Inc.  NOTTINGHAM   48.599998  2014-12-20                  Fire Hydrant Cap      Red
          16       Alien Aircaft Inc.  NOTTINGHAM   57.299999  2014-12-20                Acme Widget Washer   Silver
          17    Acme Widget Suppliers    ILLINOIS   15.300000  2014-03-03                Acme Widget Washer      Red
```

Figure 6: Projection over an AssociativeSet (SPC) and output in the form of associations (keys-values) or tuples

Operators that are already implemented in the current version of TriaClick are:

- **Count, Average, Sum,** (aggregation)
- **Over** (projection)**, Select** (user input)**, Filter** (associative filtering)
- **Where, AND, OR, Between, In, Like,** (conditions)
- **Show** (display ClickHouse SQL query)
- **Execute** (execute operations)

Method chaining is a very popular development approach in python object-relational mappers (ORMs). Same as ORM, CQL provides a high-level abstraction upon the query language of the DBMS, e.g. ClickHouse SQL. That allows the developer to write Python code and to use object methods with a fixed set of arguments instead of writing complex SQL code to create, read, update and delete data and schemas in the DBMS. CQL also makes it theoretically possible to switch an application between different DBMS and also **reuse the same queries in a different project by modifying only the methods' arguments**. And because objects, i.e. data model and data instances, are constructed dynamically from numerical key references that are stored in TriaClick databases; **the "impedance mismatch" problem disappears**.

Interactive Data Exploration

Interactive, guided, ad hoc business query is arguably one of the most important key differentiating features of TriaClick. Metadata specifications, managed by *DataModel* and *DataResource* modules (see TriaClick Systems), are used to map Entities and Attributes, i.e. measures and dimensions, to the underlying physical structures of ClickHouse. The purpose of this is to hide complexity from users and make it easier for them to select data for filtering and aggregation operations.

Logical Inference

Currently user selections are implemented with CQL *Select* operator. After executing a CQL filtering operation TriaClick engine immediately calculates all distinct values for each *Attribute* of an *Entity* that are relevant to the selection. The engine not only filters the Entity, i.e. table, but also all the other attributes, i.e. columns, instantaneously filter themselves based on that selection. This logical inference allow TriaClick engine to show the user/developer not just which data is associated with user's selections but also what data is excluded due to these value selections. And since our logical data model is a HyperGraph (see *TriaClick HyperGraphs*) every data point in the entire dataset is always associated with all other data points at all times. This means that TriaClick engine allows a user to interact with a broader range of data than will ever be possible in SQL. Technical and business users are free to search and explore the dataset based on the visual feedback they get from TriaClick (currently in the form of pandas data frames only).

The main differences from other associative engines is that **processing of interactive, free-form queries is taking place on-disk thanks to the powerful columnar layout of ClickHouse and it is also possible to append or modify data without reloading the whole or part of the original data set**.

Furthermore another key differentiation factor from other SQL based systems is the ability to refine context. In other data models entities and attributes are discrete, disconnected and don't stay in context with one another. Filtering doesn't show the relationship or impact that selection has on objects within an application. In TriaClick associative, semiotic, hypergraph data model **user's iterative interactions perform a progressive query materialization (Fig. 7) that reflects at each step a new context for items of a data subset that are aware of their position and their neighbours**.

```
  ...: dm = DataModel(mysql_conn, clickhouse_conn, key=(2, 200, 0), debug=0)
  ...: # Create HyperBond Collection
  ...: hbcol = HyperCollection(dm, 2, 200, 1)
  ...: # Create Associative Set
  ...: Physicians = AssociativeSet(hbcol, reset=True)
  ...: # Unfiltered State
  ...: Physicians.count_values(coltype='HACOL', attributes='npi, groupTotal, school, org', order='key')
  ...:
Out[3]:
               Attribute  Distinct Values
(2, 200, 2)          npi          1053573
(2, 200, 11)      school              401
(2, 200, 19)         org            72791
(2, 200, 21)  groupTotal              615
In[4]: # First Selection and Initial Filtering
  ...: selection1 = HyperCollection(dm, pattern='school').cql.Select().Where("$value").Like('%UNIVERSITY%')
  ...: Physicians.cql.Filter(selection1).Execute()
  ...: Physicians.count_values(coltype='HACOL', attributes='npi, groupTotal, school, org', order='key')
  ...:
Out[4]:
               Attribute  Distinct Values
(2, 200, 2)          npi           378846
(2, 200, 11)      school              197
(2, 200, 19)         org            47968
(2, 200, 21)  groupTotal              615
In[5]: # Second Selection and Progressive Filtering
  ...: selection2 = HyperCollection(dm, pattern='groupTotal').cql.Select().Where("$value").Between(500, 1000)
  ...: Physicians.cql.Filter(selection2).Execute()
  ...: Physicians.count_values(coltype='HACOL', attributes='npi, groupTotal, school, org', order='key')
  ...:
Out[5]:
               Attribute  Distinct Values
(2, 200, 2)          npi            39088
(2, 200, 11)      school              175
(2, 200, 19)         org              136
(2, 200, 21)  groupTotal              119
In[6]: # Third Selection and Progressive Filtering
  ...: selection3 = HyperCollection(dm, pattern='org').cql.Select().Where("$value='COOK COUNTY'")
  ...: Physicians.cql.Filter(selection3).Execute()
  ...: Physicians.count_values(coltype='HACOL', attributes='npi, groupTotal, school, org', order='key')
  ...:
Out[6]:
               Attribute  Distinct Values
(2, 200, 2)          npi              237
(2, 200, 11)      school               60
(2, 200, 19)         org                1
(2, 200, 21)  groupTotal                1
```

Figure 7: Incremental query in three stages of progressive associative filtering

TriaClick as a State Machine

We can really think of TriaClick engine as a state machine for data sets. When we apply a selection and then ask for filtering the data; the engine will propagate that filter across the data model based on TriaClick hyperlinks. In the current implementation there are two different states for each distinct attribute value: One is the input state, i.e. whether the value is selected or not; and the other is the ouput state, i.e. whether the value is associated with those selected or excluded due to previous selected values (**Fig. 2**).

Epilogue

“ *Ted’s basic idea was that relationships between data items should be based on the item’s values, and not on separately specified linking or nesting. This notion greatly simplified the specification of queries and allowed unprecedented flexibility to exploit existing data sets in new ways. The idea of relying only on value-based relationships was quite a radical concept at that time, and many people were skeptical.* ” - **Don Chamberlin, co-inventor of SQL**

Fifty years ago Codd was surrounded from network and hierarchical models, today IT professionals are also surrounded with graph and document databases. Storage engines, with columnar layout, have been changed to accommodate massive volumes of data on RAM and SSDs with distributed vectorised processing power but data model standards did not follow that evolution. Perhaps **the idea of relying on reference-based associations and logical identifiers is quite a radical concept in present times** but we have demonstrated its novelties and advantages in TriaClick. We believe that associative semiotic hypergraph technology can make the difference in the production of modern DBMS that are driven from a database management framework like TRIADB.