**The Architecture of Data Civilizer**

By

authors

**Abstract**

<<needed>>

**I Introduction**

There is overwhelming anecdotal evidence that data scientists spend at least 80% of their time finding, preparing, integrating and cleaning data sets which they wish to analyze. The remaining 20% of their time is spend doing the analysis tasks, that comprise their job description. One data officer (Mark Schrieber of Merck) estimates the number is 98%, in which case data scientists spend less than one hour a week on tasks in their job description.

In this paper we present the architecture of **Data Civilizer**, a system under construction at MIT, QCRI and Waterloo, whose purpose is to lower the “mung work” faced by data scientists. In the rest of this section we present the environment at Merck to motivate the components of Data Civilizer. We also indicate an architecture diagram of our system. Then, Section 2 – 6 present the components of our system, followed in Section 7 by experiences “in the wild” at both Merck and the M.I.T. Data Warehouse. We expect to present a demo at CIDR of our system running in the MIT environment. Finally, Section 8 draws conclusions and suggests next steps.

* 1. **Merck Environment**

Merck is a large decentralized drug company with about XXX employees, of which YYY are data scientists. An exemplar data scientist would come up with a hypothesis, for example the drug ritalin causes brain cancer in rats weighing more than 1Kg. His first job is identify data sets, both within the Merck firewall and outside that might contribute to resolving this question. Inside the firewall, Merck has some 4000 Oracle databases and countless other repositories in which relevant data might reside. Data Civilizer has a **Discovery** component, discussed in Section 2,which assists the scientist with finding data sets of interest.

Data sets identified by the Discovery module are invariably linked together by intermediary data sets. Hence, the next step is to construct “data stitching” paths among all of the data sets indentified during discovery. This task is the job of the **Data Stitcher**, which is discussed in Section 3. One can think of the output of the data stitcher as one or more **views** on the underlying data. It is now necessary to perform data curation on these multiple views. This entails **extracting** data from source data storage systems, performing **schema integration** on the multiple views, **transforming** data into a common representation, **cleaning** erroneous values from the source data sets, and performing **entity consolidation** on resulting records. Our Data Tamer system [ref] dealt with schema integration and entity consolidation. More recently, we wrote a system DataXformer [ref] which supported transformations, and we examined a collection of data cleaning systems [ref].

Since Merck has a variety of data storage systems and exascale data volumes, it is simply not reasonable to move all data to a central “data lake”. Also, it is not reasonable to perform data curation up front on enterprise data, as was advocated by Data Tamer. Instead Data Civilizer must be a **pull-based system** that does **data curation on demand**, as data scientists need to access data to get their work done. Therefore, Data Civilizer must be based on a polystore architecture [ref], that can pull data out of multiple underlying storage engines on demand. Obviously, data cleaning, data transformation and entity consolidation must be integrated with querying the polystore. In this way, a key technical optimization is to push filters and joins through cleaning operations and into the underlying data storage system wherever possible. The merger of polystores and data curation steps is discussed in Section 4, and we term the resulting system a **Curating Polystore**.

Optimizing such a curating polystore is the subject of the next two sections. Materializing views is very expensive, because of the human effort involved, when automatic algorithms are unsure of what to do. Therefore, we must **estimate** the cost of constructing the MV, because a scientist may not have the budget to pay for the human effort involved. This is the topic of Section 5. It entails constructing a model for how dirty the data in the source data sets actually is. An ancillary topic is to estimate the cleanliness that can be achieved for a given budget for cleaning activities. In this way, a scientist can decide whether or not he wishes to proceed with the project at hand.

Moreover, it is silly to discard expensive-to-construct materialized views (MV) after their initial use by a data scientist. Hence, we assume that they are generally retained for future use. Moreover, future MVs may be based off previously constructed ones or on original data sources. As a result, there may be several ways to construct a new MV, with differing costs and available accuracy (since each existing MV has some given accuracy, as noted above. Therefore, the data stitching problem must be revisited to deal with this materializiation cost/accuracy tradeoff. This is the subject of Sections 6.

Section 7 than turns to update issues. If a source data set is updated, then updates must be incrementally propagated through the data curation pipeline to update downstream MVs. In some cases, the human effort involved may be daunting, and the MV should be discarded rather than updated. Lastly, if a scientist updates his MV, we must **propagate** changes to other descendent data sets, as well as back upstream to data sources.

Section 8 then turns to our workflow system, whereby a data scientist can iterate over our components in whatever order he wishes, undo previous workflow steps, and perform alternate branching from given MVs.

We then turn in Section 9 to the current implementation status of Data Civilizer and indicate initial user experience at the MIT data warehouse, as well as Merck. Section 10 gives conclusions, and outlines our future research plans.

**II Discovery**

Raul

**III Data Stitching**

Dong

**IV Curating Polystore**

Our local polystore system is called BigDAWG. It consists of a middleware query optimizer and executor, and shims to various local storage systems, as noted in [ref-1, ref-2]. Assume that a user has run discovery and stitching to identify a virtual relation (view) that he wishes to query. Also, assume that he has identified the subset of each source table in which he is interested. In other words, he has defined the join path using stitching and the predicates that subset the tables in some other way. This will translate directly to a BigDAWG query. In general, a user will want to retrieve a subset of the view, which is merely an extra predicate(s) attached to the BigDAWG query. In this section we address how to integrate data cleaning and transformation operations with this query plan.

In general, one needs to clean the data prior to performing transformations. For example, if one has a data value, New Yark, and wants to transform it to one of its airport codes (JFK, LGA), then one must correct the data to New York, prior to the airport code lookup. Obviously, cleaning usually entails a human specification of the corrected value or a review of an automatic algorithm. Hence, it is expensive in human resources, which we believe is generally “the high pole in the tent”. As such, a user has to decide how he wants to trade off data cleanliness and cost. Data Civilizer defines two parameters under his control.

1. Minimize cost for a specific cleanliness metric. In this case, the user requires the data to be a certain percentage, P, correct and will spend whatever it takes to get to the point.
2. Maximize accuracy for a specific cost. In this case, the user is willing to spend M$ and wishes to make the data as clean as possible.

Sometimes the user is the one actually cleaning the data. In this case, he can use P and M to quantify the value of his time. In other cases, cleaning is done by other domain experts, who generally need to be paid. In this case, P and M are statements about budget priorities.

Obviously one want to perform expensive cleaning on as few records as possible. Hence, one would like to insert cleaning operations as late in the query plan as possible. Unfortunately, if one runs a query that has the predicate:

…where name = “New York”

then the misspelled city name will not be found, and accuracy will suffer. One solution is to clean the entire source data sets to avoid such errors, an expensive proposition indeed. In Data Civilizer, we support a third alternative; namely more information about the errors in each data set. First, we assume that each data set owner gives us accuracy metrics for each column, namely an estimate for the percentage of the column which is erroneous. Second, the same data set owner is required to specify a “disorder metric”, which indicates the average and variance of the lexical distance between an incorrect value and its ground truth. For example, if salary errors average 5% with variance 2%, then 2/3 of the errors are less than 7% off. If an address field routinely confuses “road” and “street”, but very rarely gets the name of the street wrong, then the lexical distance is again very small. If the penalty is small, then we can insert cleaning “late” in the query plan. On the other hand, if the penalty is large, then we must insert cleaning earlier, of course at much higher cost. The next section analyzes this tradeoff in more detail.

**V Cleanliness Estimation**

Mourad

**VI Enhanced Data Stitching**

Dong

**VII Updates**

Nan

**VIII Workflow**

Zia????

**IX Data Civilizer in the wild**

Raul??