Cloud Data Denormalization of Anonymous Transactions

Aspen Olmsted, Gayathri Santhanakrishnan
Department of Computer Science
College of Charleston, Charleston, SC 29401
e-mail: olmsteda@cofc.edu, santhanakrishnang@g.cofc.edu

***Abstract*— In this paper, we investigate the problem of representing transaction data in PAAS cloud-based systems. We compare traditional database normalization techniques with our denormalized approach. In this research, we specifically focus on transactional data that is not attached to a specific customer. The absence of the relationship in the customer object allows for the vertical merging of tuples. to be stored in aggregate form. The journaling features of the data store, allow full audits of transactions while not requiring anonymous data to be materialized in fine-grained levels. The horizontal merging of objects is also deployed to remove detail lookup data instance records and replace them with business rule objects.**

Keywords-web services; distributed database; modeling

I. INTRODUCTION

In this work, we investigate the problem of representing transactional data in a platform as a service (PAAS) cloud-based system. In traditional client-server architectures, database normalization is used to ensure that redundant data is not stored in the system. Redundant data can lead to update anomalies if the developer is not careful to update every instance of a fact when modifying data. Normalization is also performed to ensure unrelated facts are not stored in the same tuples resulting in deleting anomalies.

Data representation in the cloud has many of the same challenges as data representation in client/server architectures. One challenge data representation in the cloud has that is not shared with client/server is the minimization of data because costs of cloud data storage are significantly higher than local storage. We are thinking of simple costs for disk storage, not true costs. Organizations budget only the costs of disk drives for local storage which are in the tens of dollars per gigabyte but cloud storage can be in the hundreds of dollars per gigabyte per month [1]. Often this storage is expressed as the number of tuples in the data store instead of the number of bytes on the disk drive holding the data. For example, force.com [2] charges for blocks of data measured in megabytes but they calculate usage as a flat 2KB per tuple. Zoho Corporation also tracks data storage by the tuple for serval of their cloud products including Creator [3] and CRM [4]. The tuple count method is used as it is easier to calculate in a multi-tenant system where the physical disk drives are shared by many clients.

In this paper, we present an algorithm that will minimize the number of tuples used to store the facts for a software system. We use a motivating example from a cloud software system developed by students in our lab. The algorithm performs three main operations:

- The horizontal merging of objects – several distinct relations are combined into one.
- The vertical merging of objects – several distinct instances of the same type of facts is combined into one.
- Business rule adoption – instead of storing tuples to represent availability of lookup data, we replace the tables with pattern based business rules

We apply our algorithm to a system in the humanities application domain and show an approximately 500% reduction in tuple storage.

Date [5] invests a good deal of his text on the definition of denormalization. He argues that denormalization is when the number of relational variables is reduced, and functional dependencies are introduced where the left hand of the functional dependency no longer is a super key. In our work, we perform many optimizations. When we horizontally merge relations, then we are performing a true denormalization. Other optimizations such as vertical merging do not fit Date's definition of denormalization. We choose to stay with the term denormalization algorithm as it is a set of steps taken after the normalization process.

The organization of the paper is as follows. Section 2 describes the related work and the limitations of current methods. In Section 3 we give a motivating example where our algorithm is useful and describe our denormalization algorithm. Section 4 describes additional enhancements through the design of business rule objects. Section 5 explores reporting from the denormalized objects utilizing the object version history stored in the journal. Section 6 contains our comparison of the proposed method and the traditional database normalization method. We conclude and discuss future work in Section 7.

.

II. RELATED WORK

Sanders and Shin [6] investigate the process to be followed when denormalization is done on relational data management systems (RDBMS) to gain better query performance. Their work was done before the cloud database offerings became prevalent. In the cloud, database performance is less of an issue to storage requirements because the systems are designed to distribute queries across many systems.

Conley and Whitehurst [7] patented the idea of denormalizing databases for performance but hiding the denormalization for the end user. Their work focuses on merging two relations into one relationship to eliminate the processing required to join the records back together. Their work uses horizontal denormalization to gain performance.

 ISBN: 978-1-61208-460-2

Our work uses both horizontal and vertical denormalization to minimize storage space and increase usability.

Most denormalization research work was done in the late 1990s and was focused on improvement in query performance over the correctness and usability of the data. Recently, folks like Andrey Balmin have looked at denormalization as a technique to improve the performance of querying XML data. Like the previously mentioned research, this work differs from our work in the desired end goal of minimized data storage and end user usage.

In Bisdas' [8] blog, he demonstrates ways that end users can improve data visualization by vertically merging hierarchical data in the Salesforce, data model. He takes advantage of the trigger architecture to create redundant data in the hierarchy. Taber [9] also recommends denormalization to improve data visualization. The problem with both solutions is that data storage requirements are increased while correctness is jeopardized by the redundant data.

In our previous work [10], we study UML models from the perspective of integrating heterogeneous software systems. In the work, we create an algorithm to sort cyclical UML class data diagrams to enable transaction reformation in the integration. In the process, discoveries were made on the freshness of data at different layers in the UML graph. The knowledge is useful in this study when considering anomalies that can happen in response to data updates.

Additional semantics for models can be added by the integration of the matching UML Activity and Class diagrams. UML provides an extensibility mechanism that allows a designer to add new semantics to a model. A stereotype is one of three types of extensibility mechanisms in the UML that allows a designer to extend the vocabulary of UML to represent new model elements [11]. Traditionally these semantics were consumed by the software developer manually and translated into the program code in a hard coded fashion.

Developers have implemented business rules in software systems since the first software package was developed. Most research has been around developing expert systems to process large business rule trees efficiently. Charles Forgy [12] developed the Rete Algorithm, which has become the standard algorithm used in business rule engines. Forgy has published several variations on the Rete Algorithm over the past few decades.

III. DENORMALIZATION

We demonstrate our work using a Tour Reservation System (TRS). TRS uses web services to provide a variety of functionalities to the patrons who are visiting a museum or historical organization. We use the UML specification to represent the meta-data. Figure 1 shows a UML class diagram for an implementation of this functionality. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software systems [13]. In this study, we use data collected by the Gettysburg Foundation on visitors to their national battlefield. The system is modeled and implemented on the force.com [2] cloud platform.

Figure 1 shows a normalized UML class model of reservation transactions of visitors to the Gettysburg National Battlefield. In the model the central object ticket represents a pass for an entry that is valid for a specific date and time and a specific activity. Activities are itinerary items the visitor can be involved in while visiting the battlefield. In the normalized model, each ticket is linked to a specific activity schedule entry that will designate the date and time the pass is valid for entry. Each activity schedule is linked to an activity object that designates the name and location of the activity.

Each ticket is linked to a user in the Gettysburg organization who was responsible for the transaction. Each ticket can be linked to a patron object. In the case of advanced reservations, there will be a valid patron object linked to the ticket. Advanced reservations are transactions that take place through the organization's website or over the phone to a reservation agent. In the case of walk-up transactions, there will not be a linked patron. A walk-up transaction is a transaction that takes place when a visitor arrives on the site without a prior reservation and pays for the ticket at the front desk.

In Figure 1, the multiplicity of the association between the patron and the ticket is a zero or one to many. A multiplicity that can be zero represents anonymous data. Anonymous data is data that does not need to be specified in order for the transaction to be valid. In the example transaction, the patron can remain anonymous but still visit the battlefield and partake in the activities. In the case of the sample Gettysburg data, 60 percent of ticketing transactions were anonymous.

In the case of the force.com [2] PAAS, data storage is charged by the tuple. With an enterprise license to the platform, the organization is granted access to one gigabyte of data storage. The storage is measured by treating every tuple as two kilobytes. This form of measurement allows

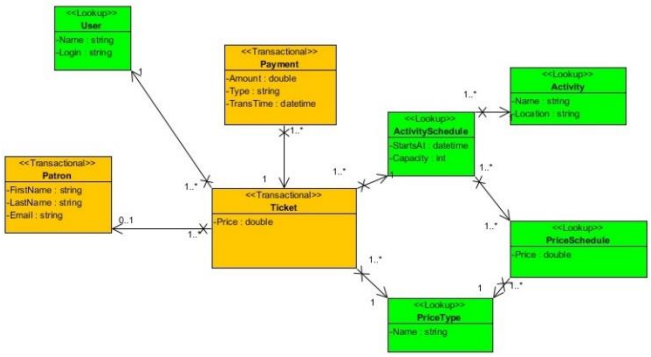

Figure 1. Normalized Transaction Model

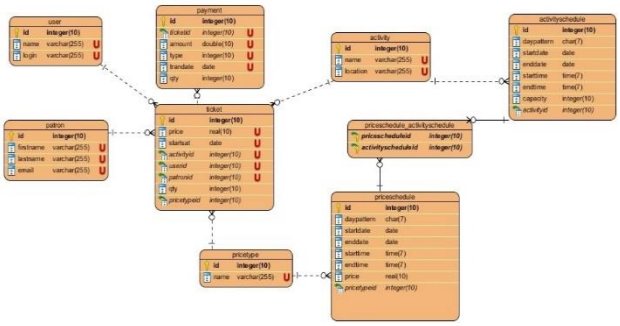

Figure 2 Denormalized Transaction Model

the organization 500,000 tuples in the enterprise data storage option. The data collected for the normalized data model would allow the Gettysburg organization to store around nine months' worth of data. With the anonymous transaction, the ticket and the payment data is only important on the original transaction level for auditing. For example, the accounting department may want to see the details behind a specific ticket agent's cash total for the day. Another example would be the marketing department wants to see the ticket price patterns within an hour of the day.

The force.com [2] platform uses an Oracle relational database to deliver the data storage services but adds a journal feature so history can be stored on all changes to an object over time. This journal can be used at no additional data storage cost. The field level changes stored in the journal would allow aggregate data to be stored for anonymous transactions and still have the detail to perform the audits mentioned earlier.

If an object is used between two other objects where the middle object is the "many" side of the one-to-many relationship and the one side of the other relationship, then the same data can be represented by moving the attributes to the object on the composition side of the relationship. The middle object is then able to be removed, reducing the number of tuples representing the same amount of data. In Figure 1, the "Activity Schedule" object fits this profile and can be horizontally merged with the "Ticket" object. In our previous work [10], we study UML data model freshness requirements and document the relationship between data changes and location in the UML graph. In our findings, we see that middle object nodes are less predisposed to changes than leaf nodes. The lower amount of data changes reduces the change of update anomalies.

In Figure 1, we also designate objects that are updated in transactions differently than objects that are navigated for transactional lookup values. Two stereotypes are added to the diagram:

- Transactional – The classes designated with the orange color and the <<Transactional>> tag are updated during transactional activities.
- Lookup – The classes designated with the green color and the <<Lookup>> tag are not updated during transactional activities. The data in these classes are created by administrative activities. During transactions, the data is searched for the proper values.

The Denormalization Algorithm, Algorithm 1, transforms a normalized model stored in a UML class diagram into a denormalized model represented as an entity-relationship diagram. The algorithm assumes input and output of the models in the XMI [14] format. XMI is a standard exchange format used to represent structural models in a non-proprietary way.

The algorithm first loops through each object in the normalized model and adds the object and attributes as entities in the denormalized entity-relationship diagram. If the object has the transactional stereotype, then the attributes are marked unique. Surrogate identifier fields are added to each object's definition to be used as an auto-incrementing primary key.

The next pass of the algorithm is to find objects that can be eliminated from the middle relationship of two "one-to-many" relationships. The original model, in Figure 1, had an activity schedule object that consumed a lot of data space by storing a lot of tuples to represent the occurrences an activity can take place. We use a stereotype of "PK" applied to attributes in the original model to designate the primary identifier for instances of an object. This designation allows us to shift the attribute down the association and swap the positioning of the objects. In this iteration over the objects, we also look for date-time data

Algorithm 1. Denormalization Algorithm

INPUT: normalizedObjects (XMI representation of UML class diagram)
OUTPUT: denormalizedEntities (XMI representation of denormalized entities)

```
foreach object in normalizedObjects
   add entity to denormalizedEntities
   foreach attribute in object
      add attribute to entity
      if object is transactional
         mark attribute as unique
   add id attribute as primary key
   mark id as autoincrement

foreach entity in denormalizedEntities
   if entity is both a many side and a one side of two
relationships
     and  a lookup object
      foreach attribute in object
         if attribute is PK
            add attributes to many side entity
            if attribute is a datetime type expand date pattern
      swap graph location entity of the one sides

foreach association in normalizedObjects
   if association is one-to-may and many side is transactional
      add foreign key to many side entity
      add quantity field to entity on many side
```

types that are part of the primary key. When we locate an occurrence, we replace the attribute with a date-time specification occurrence. The date-time specification includes a starting date, ending date, starting time, ending time and day of the week pattern.

The final pass of the algorithm adds foreign keys and aggregation counters. The aggregation significantly reduces the count of tuples stored. An example of this is shown in Figure 1. Instead of having an instance of each ticket, we add the quantity field to store the aggregate count for the unique attributes.

Figure 2 shows an entity-relationship diagram of a transformed model of Figure 1. Unique attributes have been applied where aggregations should be performed. The activity schedule entity has been shifted out in the graph, and the quantity fields have been added to the aggregated transactional objects.

IV. BUSINESS RULES

Business rule engines have sprung up to allow the separation of business rules from the core application code. The systems are designed to allow the end users to change the business rules freely without changing the original application code. In 2007, International Data Corporation implemented a survey where they asked 'How often do you want to customize the business rules in your software?'. Ninety percent of the respondents reported that they changed their business rules annually or more frequently. Thirty-four percent of the respondents reported that they changed their business rules monthly [15].

Figure 2 shows two tables that implement business rules:

- Activity Schedule – This table implements the date-time pattern mentioned earlier to store the business rules for when a particular activity is valid.
- Price Schedule – This table implements the date-time pattern mentioned earlier to store the business rules for when a particular price is available.

In each case objects in Figure 1, that inserted instances to represent availability, are replaced with rule instances to represent the availability. So instead of having a tuple per availability instance, a single tuple can represent the pattern. In the case of activity schedules, the example year had over 26,000 instances of availability that were replaced with 30 instances of the business rule.

V. OBJECT HISTORY ANALYSIS

One of the main reasons enterprises develop or purchase software solutions to allow the organization to increase their knowledge of their operations through the analysis of the data collected in the software solution. The denormalization solution presented earlier may limit the data available from the denormalization process. The data is presented to the users through dashboards, reports or exports. A dashboard is presented as a graphical chart to measure where the organization stands compared to a goal. Examples of these would be sales to date compared to same period last year. A report has a set of input parameters that control the data displayed. The data displayed in the report tends to include tables with aggregated values. Exports allow for the exporting of data into a two-dimensional table saved as a comma separated value (CSV) format. In this format, attributes represent the columns of the data. Columns are escaped with double quotes and separated by commas. For our purposes, we will refer to all three categories generically as reports.

Current state and historical comparison are the two categories of reports a user may want to pull in their analysis. In current state reports only the latest version of the object is needed. In historical comparison reports, all versions of an object may be needed depending on the level of aggregation. An example of a historical comparison report would be a report that compares sales for the month compared to sales last year in the same month.

In our work, we developed Algorithm 2 to create an in-memory copy of all historical versions of a specific object. We use code to generate the data and then generate the report output. If the organization wanted to allow end users to report on historical versions, they could modify Algorithm 2 to write records as temporary tuples and then call the reporting tool.

Algorithm 2. History Creation Algorithm

INPUT: object
OUTPUT: collection of object's version history

```
Set thisObject = newest version of object
Set objectVersions = empty list
Set fieldVersions = distinct saveDates values from object journal
Sort fieldVersions by saveDate descending
Set lastDate = maximum(saveDate)
Foreach version in fieldVersions
    If lastDate = version.saveDate
      objectVersions.add(thisObject)
   Set thisObject.[version/attribute] = version.value
   Set lastDate = version.saveDate
Return objectVersions
```

VI. EMPIRICAL RESULTS

The empirical results demonstrate the success of representing the example transaction data with significantly lower cloud storage costs. TABLE 1 shows the tuple counts for the original data model and the denormalized data model. Both data models represent the complete 2014 calendar year of visitor transactions for the Gettysburg National Battlefield. The denormalized model creates a 78% reduction in the number of tuples. In the specific case of the

 ISBN: 978-1-61208-460-2

TABLE 1. EMPIRICAL RESULTS

Table	*Normalized Tuples*	*Denormalized Tuples*
user	*31*	*31*
patron	*17,610*	*17,610*
ticket	*738,981*	*157,780*
activity schedule	*26,697*	*30*
price schedule	*220*	*24*
activity	*17*	*17*
Total	*783,556*	*175,492*

force.com [2] platform, the reduction in number of tuples allows the organization to store nearly three years of transaction data in the data storage provided without additional subscriptions costs. In the minimal data storage provided to an enterprise customer of force.com [2], the organization receives 500,000 tuples. Additional data storage is available to the organization for a monthly subscription price of 2,000 tuples per dollar. Using the normalized data model, the organization would not be able to store a complete year of data without needing to purchase more data storage.

VII. CONCLUSION

In this paper, we propose an algorithm for object denormalization when transforming an application domain object model to a data model used in a cloud PAAS data store. Our solution is based on navigating the relationships in a UML class diagram and horizontally compressing classes between multiple one-to-many relationships, aggregating relationships on anonymous relationships and using temporal offering patterns.

In this research, we studied a specific application domain related to humanities organizations. The algorithms can be applied to similar application domains that contain entity objects representing transactions and customers. Future work needs to test our algorithms in other application domains to ensure the work applies across different application domains.

REFERENCES

[1] brainsell blog, "Salesforce, SugarCRM and SalesLogix — Data Storage Costs Compared," 2016. [Online]. Available: https://www.zoho.com/creator/pricing-comparison.html. [Accessed 03 02 2016].

[2] Salesforce.com, inc, "Run your business better with Force.," 2006. [Online]. Available: http://www.salesforce.com/platform/products/force/?d=70130000000f27V&internal=true. [Accessed 03 02 2016].

[3] Zoho Corporation, "Creator Pricing Comparison," 2016. [Online]. Available: https://www.zoho.com/creator/pricing-comparison.html. [Accessed 03 02 2016].

[4] Zoho Corporation, "Compare Zoho CRM editions," 2016. [Online]. Available: https://www.zoho.com/crm/comparison.html. [Accessed 03 02 2016].

[5] C. J. Date, "Denormalization," in *Database Design and Relational Theory*, O'Reilly Media, 2012.

[6] G. L. Sanders and S. Shin, "Denormalization Effects on Performance of RDBMS," in *Proceedings of the 34th Hawaii International Conference on Systems Sciences*, 2001.

[7] J. D. Conley and R. P. Whitehurst, "Automatic and transparent denormalization support, wherein denormalization is achieved through appending of fields to base relations of a normalized database". USA Patent US5369761 A, 29 November 1994.

[8] A. Bisda , "Salesforce Denormalization Delivers New Power for Nurtures," DemandGen, 29 07 2014. [Online]. Available: http://www.demandgen.com/salesforce-denormalization-delivers-new-power-nurtures/. [Accessed 09 11 2015].

[9] D. Taber, Salesforce.com Secrets of Success: Best Practices for Growth and Profitability, Prentice Hall, 2013.

[10] A. Olmsted, "Fresh, Atomic, Consistent and Durable (FACD) Data Integration Guarantees," in *Software Engineering and Data Engineering, 2015 International Conference for*, San Diego, CA, 2015.

[11] Object Management Group, "Unified Modeling Language: Supersturcture," 05 02 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.1/. [Accessed 08 01 2013].

[12] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. 1, p. 17–37, 1982.

[13] Object Management Group, "Unified Modeling Language: Supersturcture," 05 02 2007. [Online]. Available: http://www.omg.org/spec/UML/2.1.1/. [Accessed 08 01 2013].

[14] Object Management Group, "OMG Formal Versions of XMI," 06 2015. [Online]. Available: http://www.omg.org/spec/XMI/. [Accessed 11 11 2015].

[15] Ceiton Technologies, "Introducing Workflow," [Online]. Available: http://ceiton.com/CMS/EN/workflow/introduction.html#Customization. [Accessed 15 09 2014].

 ISBN: 978-1-61208-460-2