I J C T A, 9(16), 2016, pp. 8257-8265

Rumination on Scaffold Pattern Language for Multi-Tenant SaaS Application Development

Nagarajan Balasubramanian* and Suguna Jayapal**

ABSTRACT

Internet has sophisticated the life of a populist. The cloud is an internet based technology which is location independent. It provides "on demand" service to the customer on pay-per-use model. The multi-tenancy is an essential characteristic in cloud, which has motivated many researchers to contribute their work. Scaffold is a platform, where, programmers can specify and integrate application and database. The pattern is a reusable component used to develop applications. NoSQL document data model has an advantage of storing the values in a tiered storage. In this contribution, a pattern language for the scaffold is proposed, which will be useful for the researchers to integrate and develop multi-tenant aware application.

Keywords: scaffold, pattern, NoSQL, document data model.

1. INTRODUCTION

Cloud computing is envisioned as the next generation revolution in the IT. It has transformed the work-life balance easy and sophisticated. The important characteristics of cloud is multi-tenancy. Researcher around the universe is attracted because of its importance and contributed their research work.

Multi-tenancy is an architectural pattern in which a single instance of a software is run on the service provider's infrastructure and multiple tenant access the same instance. A tenant is the organizational entity which rents SaaS solution.

Scaffold is a platform, in which, the programmer can specify how application and database can be interoperable. A virtual scaffold for multitenant SaaS data model is shown in figure 1. The motivation behind the scaffold, is to address the concern of cloud skeptics, an efficient architectural platform, which will perform the Create, Read, Update and Delete (CRUD) operations during an application development [9].

A pattern language is a reusable and implementation component specific to a virtual scaffold. Patterns provide an independent solution, for challenge related to application and database interoperability [18]. The metadata layer will hold the master copy of data [8]. The pattern language provides a guidance to store and retrieve data to and from a metadata layer, by proper authentication and configuration.

The business layer will execute the snippet code for accessing the data. Thus patterns will be useful component, which helps to easily implement and develop an application by providing efficient interoperability between data layer, metadata layer and business layer.

The rest of the paper is organized as follows. In the next section, a background study on related literature is done. Section 3, elaborates the methodologies used. The theme of methodologies include pattern

* Ph.d., Research Scholar, Research and Development cell, Bharathiar University, coimbatore 641046–India, Email: cloud.nagarajan@gmail.com

** Associate Professor, Department of Computer Science, Vellalar college for Women, Erode, TamilNadu, India, Email: sugunajravi@yahoo.co.in

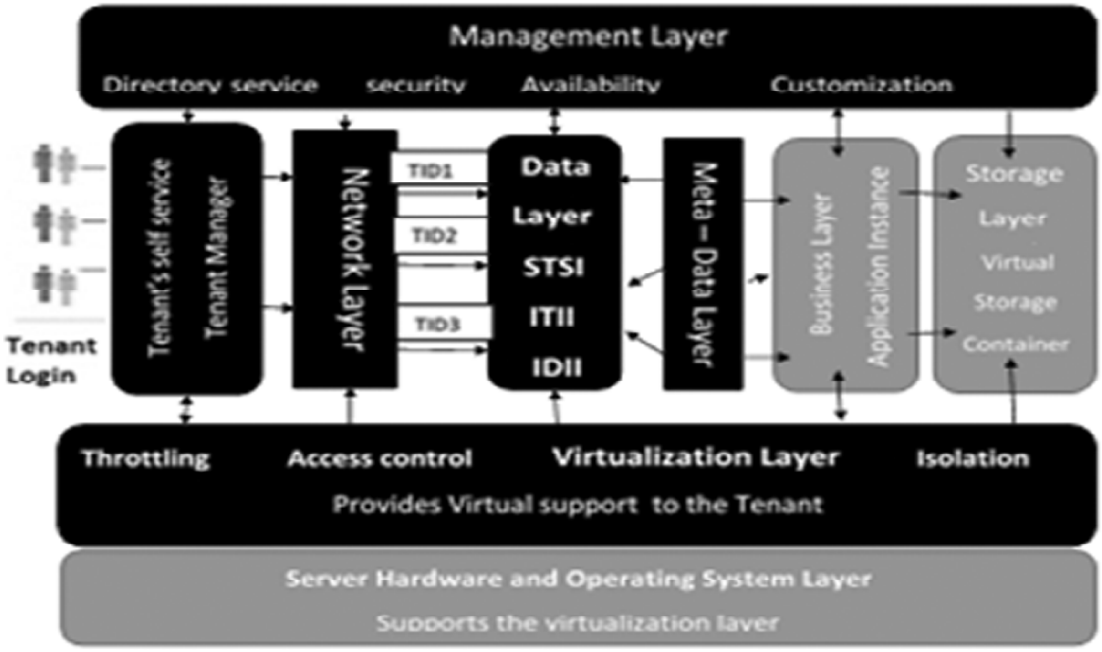

Figure 1: A Virtual Adaptive Scaffold for multi-tenant SaaS Data model [8]

identification and data model classification for designing a pattern language. Section 4 elaborates a complete pattern design with a sketch for the pattern. Section 5 annotates snippet code in document data model, which can used as back-end language for the pattern. Finally, a summary of conclusion with short description of future work is pinpointed

2. RELATED WORK

The literature survey is carried out systematically on contributions related to scaffold, pattern language and data base. The scaffold is a platform where the application and database can be interoperable.

Pieter-jan Maenhaut et al. [10] explained an architectural pattern for multitenant data management. The authors considered both application and data server and argued that load balancer, helps the tenant to login into the system. The authors also claimed that their framework will satisfy the criteria of multi-tenancy.

Steave strauch et al. [16] and [17] eloquent the ways and means by which application data layer can be moved over the cloud. They identified the challenges during implementation of cloud deployment model. An application layer was proposed in [17] and implemented in [16]. They instilled confidentiality pattern for a cloud based application development. The authors claimed that their solution is reusable for any cloud deployment. This contribution is useful and helps to understand how data can be moved from data layer to application layer.

Pattern language helps us to understand and communicate the usage of the design patterns. Tim wellhausen et al. [18] introduced the basic step and essential items to be included in defining a pattern. This contribution is helpful for novice user to understand and write pattern language.

Chirstoph fehling et al. [3] had provided an architectural pattern language. They provided pattern identification and format for a pattern language. The list of catalogue provided by the authors are useful for new pattern development.

Ralph Mietzner et al. [13] combined different multi-tenant patterns and evaluated a set of patterns that can be used to design, develop and deploy process aware service-oriented SaaS applications. The customization provided in [18] is also useful and related to this contribution.

Adewojo et al. [1] argued that tenant – isolated component only will represent a compromised implementation between the shared component and dedicated component They provide UML diagrams and snippet coding in [1] . This contribution helps us to better understand practical implementation of multi-tenant pattern language.

Dean jocobs et al [4] elaborated the basic of multi-tenant structured data bases. Stefan Aulbach et al [17] elaborated the mulit-tenant databases and provides the schema mapping techniques. Both contributions helps to understand and implement structured multi-tenant data bases.

Jing Han et al [6] surveyed on NoSQL database. Yishan Li et al [19] provided a performance comparison on key-value store implementation of SQL and NoSQL.

Bogdan Geroge et al. [2] has provided an elaborate comparison between several NoSQL. While Neal Leavitt et al [9] has entailed that organizations now-a-days are compelled to work with Big Data and hence they are looking for alternative solution to handle the data through NoSQL data models.

The above literatures [2], [6], [19] and [9] creates spur for researcher to trade-off their ideas from relational SQL to non-relational NoSQL.

To conclude with, a combined literature is conducted starting with architecture then writing language pattern and finally with the NoSQL data model. It helps to understand the concepts and creates motivation and spur to provide multi-tenant component gateway pattern.

3. METHODOLOGY

Patterns for Multi-tenant application is involute. Hence, through understanding on pattern identification, format and language will help to understand elusive steps that entails to write an eloquent multi-tenant aware pattern.

3.1. Pattern identification

Identification of a significant concept for a pattern is the first step towards a solution. [3]. It includes significant sections. The context section sets stage where the pattern takes place. The problem section explains what the actual problem is. The force section describes why the problem is difficult to solve. The solution section explains the solution in detail. The consequence section demonstrates what happen when solution is applied [18].

The next step is identification of relevant and irrelevant detail for a pattern [16]. This helps a data layer of a scaffold to hold critical data safe in the meta-data layer.

This can be carried out by pruning the pseudonymous and anonymous data [16], since they are more vulnerable in nature. The critical data can be handled by the data layer either by Pseudonymizer or Anonymization technique.

Pseudonymizer is a technique to provide a masked version of the data to the public while keeping the relation to the non-masked data in private. This enables processing of non-masked data in the private environment when required [16].

Anonymization is a technique to provide a reduced version of the critical data to the public while ensuring that is impossible to relate the reduced version to the critical data [16].

Thus by identification of the critical data, an abstract description for a pattern can be realized [3].

The next step is to decide the pattern format which will include name of pattern, icon, driving question, context, challenges, solution, sketch, result, relation to other pattern, annotations and snippet code [3].

3.2. Data models for Pattern

The reason for pattern identification is to support a scaffold, where database and application can be inter-operable. The data model provide back end support for any scaffold to develop a multi-tenant applications.

The data can be stored in data layer in any one form – Independent data base and independent instance, independent tables and shared database and finally shared instance and shared tables [9]. The SQL can be used to access the data from this tables.

The traditional database is becoming obsolete among the organization because of enormous volume of data [9]. These Big Data compels the organization to concentrate alternative way of storing their data in a "silo" storages. The NoSQL helps to store the data in a tiers with sufficient authentication [9].

The structured data in a relational database is predefined by name, layout and design. They have certain rules to access the data in the database. SQL is a convenient language for structured data but struggling with growth of huge data [7].

Nevertheless of the data volume, data should move through data layer to application layer enabling to develop multi-tenant aware application. The unstructured database are open source and easy to install and work [10].

Any NoSQL should satisfy Consistency, Availability, and Partition tolerance (in short CAP) rule. Consistency[C] equivalent to having a single up-to-date copy of the data, High Availability [A] of the data (for updates) and tolerance to network Partitions [P] [5] [6] .

The data model for the NoSQL include key-value, column-oriented and document and graph stores [6].

Key-value stores are based on key-value pairs, which resembles an associative map or a dictionary. The values are based on the key-value stored and follows two of the CAP rule namely consistency and availability. They are useful for providing use cases and resembles more like a structured database [11].

Column-oriented data base stores are derived from Google Big table, in which the data are stored in a column-oriented way. Prior definition of column is not essential. This increases the flexibility in storing any data type. This data store provide flexible and powerful indexing [11].

Graph database originated from graph theory and use graphs on their data model. It can efficiently store the relationships between different data nodes. These databases are specialized in handling highly interconnected data therefore very efficient in traversing relationships between different entities [15].

In document databases, the database stores and retrieves documents, which can be either XML, JSON or BSON. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values [11].

Document databases follow a master and slave node functioning. Mongo DB is an open source document database, which satisfy the CAP rules [5]. They can be used to write snippet code to establish inter-operability in the scaffold.

4. SCAFFOLD PATTERN LANGUAGE

The scaffold pattern language can be devised by identifying the pattern and then extending the language for the pattern.

Name: Multi-tenant component gateway pattern

Context: This pattern works exclusively for multi-tenant aware application development.

Problem: multi-tenant application has certain criteria to satisfy such as isolation, availability, scalability and customizability. Hence a pattern should be devised to satisfy almost every criteria.

Force: multi-tenancy is an architecture in which single instance of a software application serves multiple tenants. Hence a pattern should be sketched to isolate the tenant, instance is available to everyone, provide provision for scalability and customization for the tenant. The metadata layer should be properly maintained to provide data to the tenant.

Solution: The NoSQL: document data model can be used as a back end to store large amount of data. The inter-operability can be done with the application by writing snippet code in MongoDB. In the metadata layer since the volume of data is enormous, map reduce can be applied to prune the required data by the tenant.

Consequence: When the solution is applied, the tenant can be isolated and can able to search for their information from the Meta layer becomes smooth. The data layer can fetch those reduced data and move them to the application layer.

Relation to other pattern: The **Authorization service pattern**[1] help to identify the tenant. **Federated identity pattern**[1] provides the external identity for a tenant by assigning Tenant ID (TID). **Gatekeeper Pattern**[2] protect applications and services by using a dedicated host instances and prune pseudonymous and anonymous persons using this pattern. **Valet key pattern**[1] will restrict direct access to a client for a specific resource or service. **Compute and Query Responsibility Segregation (CQRS**) **Pattern**[2] can be applied in a scaffold to perform CRUD operation. This pattern segregates the operation that read data (Queries) from the operation that update data (command) by using separate interfaces. This implies that the data models used collaborated with other pattern to provide results in a scaffold.

Event sourcing pattern[2] use an append-only store to record the full series of event that describe history of actions taken on data. **Index table pattern**[2] creates indexes over the fields in data stores that are frequently referenced by query criteria. **Sharding pattern**[2] can be used to divide a data store into a set of horizontal partition. This can be used for the tenant with shared component. **Throttling pattern**[2] can be used to control the consumption of resources. This can be implemented for tenant with dedicated components. **Materialized view pattern**[2] provides prepopulated views. In the NoSQL document data model a subset of data from a big data base can be generated using this pattern.

Variations: The multi-tenant pattern can be implemented in three levels namely **shared component**[1]**, tenant-isolated component**[1] **and dedicated component**[1].

Known uses: Microsoft has published a cloud design pattern using a synchronous communication channel and services using

Windows Azure[2]. Christoph Fehiling et al [4] has published general purpose patterns which were arranged as catalogue[1]

Sketch: The scaffold pattern language for multi-tenant aware application is in

Results: Multi-tenant component gateway pattern helps the tenant to login multi-tenant in three categories namely shared component, isolated-tenant and distributed component. The tenant login into the system will be assigned a proper TenantID (TID) using authentication service pattern. The gatekeeper pattern protects the data and services, while valet key pattern will help authenticated tenant with valid TID to access data from the data layer as a shared data component. The application development is done as an integrated component software. The meta-data layer can hold either table or collections.

Thus multitenant pattern is a consolidation of set of patterns. The NoSQL document data model is used as the backend which can segregate the read data (command) and update (queries) data. The event sourcing will support tiered storage in a virtual container. This will increase the space complexities when big data base exist.

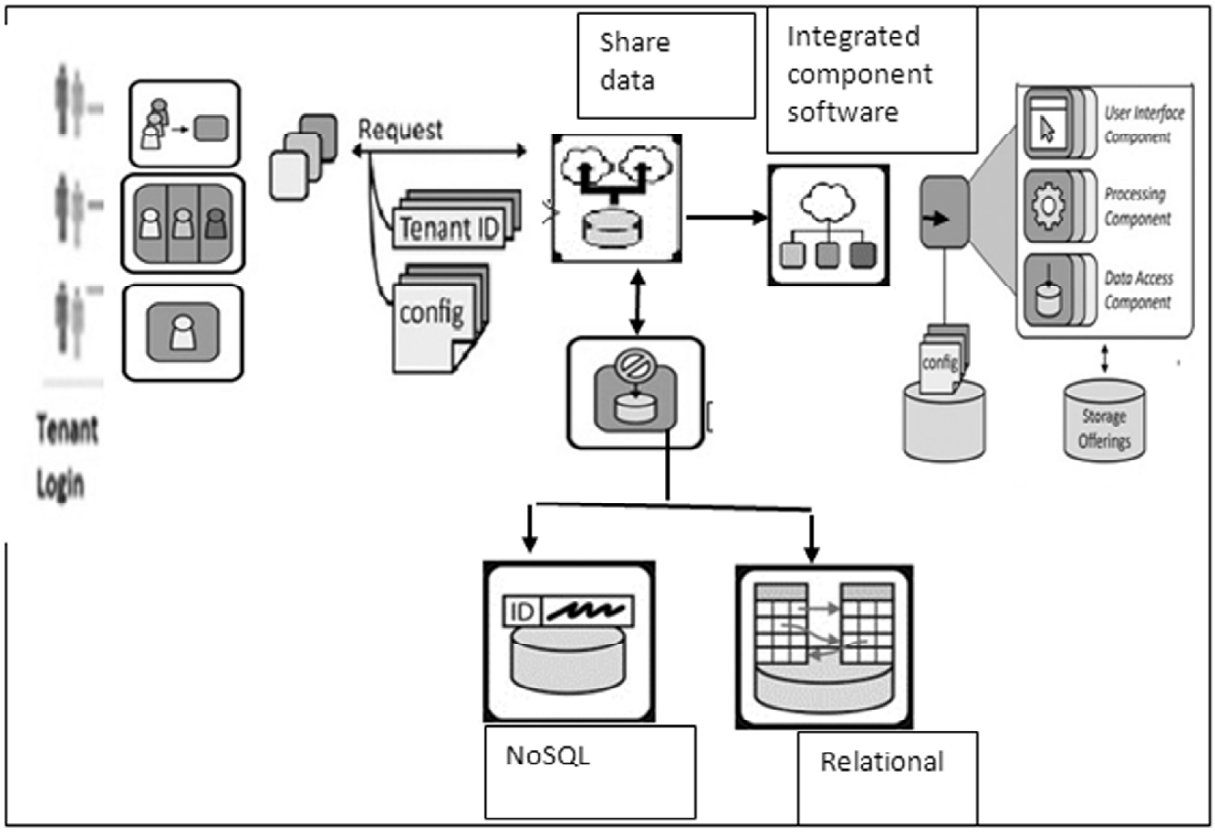

Figure 2: A Sketch for Multi-Tenant Component Gateway Pattern

5. SCAFFOLD PATTERN LANGUAGE

The multi-tenant component gateway pattern can be implemented using three different levels of components namely shared component, tenant-isolated component and dedicated component.

As soon as , the tenant login into scaffold, a unique TID is provided. The master copy of data are stored into meta data layer. The data layer will receive the data whenever the tenant request the data. Those data can be independent database and independent instance, independent table and shared database instance or shared tables and shared database instance [8].

In this contribution, the data model for the multi-tenant data bases is combined with multi-tenant component gateway pattern in figure 2.

To handle enormous amount of data and queries, NoSQL instils document data model, which instantiate ways and means to manage the tenant collection (table in SQL). NoSQL provides, document as a data, in binary representation called BSON (Binary JSON (JavaScript Object Notation)).

The mongoDB is a NoSQL document data model. It has a rich, interactive JavaScript shell, providing a convenient way of querying and updating data as well as perform administrative operations [21].

Tenant can create a new data base table (or collection in document database) using Create statement

db.createcollection(″Tenant1″)

{ ″ok″: 1}

Tenant can insert new documents (or fields) as

> db.Tenant1.insert([{name: ″john″,

... age:34,

... occupation: ″mongodbadmin″,
... company: ″it4u″,
... country: ″India″},
... { name: ″jack″,
... age:35,
... occupation: ″oracleadmin″,
... country: ″UK″,
... company: ″cloud4u″
...}
...]
...)

The updating of tenant data can be done as
db.Tenant1.update({name = ″John″},
{
$set(company:″cloudminds″,
Occupation:″mongoadmin″,
Dataofjoining:″aug162015″,
Performance:″good″
}
}
)

The isolated tenant can be handled using $isolated command in mongoDB. This is helpful to lock the data the tenant specific data.

The tenant with shared component can be handled using sharding, where the data is stored across number of machines. In range based sharding, documents are partitioned across shards according to the shard key value. In hash based sharding, documents are uniformly distributed according to hash of the shared key value. In location aware sharding, documents are partitioned according to a tenant-specific configuration that associates shard key range with specific shards and hardware.

The command query responsibility segregation (CQRS) pattern is implemented along with sharding pattern for the tenant with shared component

```
// simple Command and Query Segregation example
Public class Tenantdatastore {
    // Query method
Public tenant Gettenant1 (int TID) {
// query data storage for specific tenant by TID
// return tenant
}
// command Method
Public void Insert (tenant TID)
// Insert tenant into data storage
```

```
}
Public void updatename (int id, stirng name) {
// find tenant in the data storage by TID
// Update the company, occupation, dateofjoining, performance)
}
}
```

Event sourcing pattern will help implementing the sharding by having separate storage. This language extension entails that the pattern should be extended properly with command and query segregation to run the scaffold, which helps in smooth multi-tenant aware application development.

6. CONCLUSION

This contribution consider a virtual scaffold and provide a pattern language for it. The pattern language provided in this contribution is specific for a multi-tenant SaaS data model. The NoSQL annotation and snippet code is ease in usage. Thus, help to develop application by access the data from the metadata layer. The results can be stored in a virtual container as a tiered storage. The future scope is to extend this pattern language annotations as back end and develop a multi-tenant application in a popular language using the component discussed in this contribution, which should be useful to all walks of people.

NOTES

1. http://cloudcomputingpatterns.org
2. http://womdpwsazure.com

REFERENCES

[1] A.A. Adewojo, J.M. Bass and I.K. Allison . "Enhanced cloud patterns: A case study of multi-tenancy patterns", International conference on information society (I- society 2015) http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=7366858&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D7366858

[2] Bogdan George Tudorica and Cristian Bucur, "A Comparison between several NoSQL databases with comments and notes", http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5993686&url=http%3A%2F%2 Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5993686

[3] Christoph Fehiling, Frank Leymann and Walter Schupeck. "An Architectural Pattern Language of Cloud-base Applications", proceeding of the 18th conference on pattern language of programs, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.644.9198&rep=rep1&type=pdf

[4] Dean Jocobs and Stefan Aullbach , "Ruminations on Multi-Tenant Databases" . In BTW , Volume 103 of LNI , Pages 514-521 , GI , 2007, http://www.db.in.tum.de/research/publications/conferences/BTW2007-mtd.pdf

[5] Eric Brewer , "CAP Twelve years Later : how the "Rules" Have changed" published by IEEE Computer Society , February 2012 , Pages 23–29, http://www.mif.vu.lt/~donatas/PSArchitekturaProjektavimas/Library/EventualConsistency/2012%20Brewer%20CAP%20twelve%20years%20later%20-%20How%20the%20rules%20have%20changed.pdf

[6] Jing Han, Haihong E, Guan Le and Jian Du , "Survey on NoSQl Database" Published by IEEE , 2011 http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6106531&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6106531

[7] Martin Grund, Matthieu Schapranow, Jens Krueger and Anja Bog, "Shared Table Access Pattern Analysis for Multi-Tenant Applications" Published in Advanced management of information for Globalized enterprised symposium , Published by IEEE , 2008, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4721441

[8] Nagarajan Balasubramanian and Dr.J.Suguna , "A Virtual Scaffold for Storage Multi-tenant SaaS Data Models", international journal of Applied Engineering Research , volume 10, Number 20 (2015) pages 40775–40780.

[9] Neal Leavitt, "Will NoSQL Databases Live Upto their promise? "Published by the IEEE computer society , 2010., http://www.leavcom.com/pdf/NoSQL.pdf

[10] Pieter-Jan Maenhaut , Hendrik Moens and Filip De Turck, “Design and Evaluation of a Hierarchical Multi-Tenant Data Management Framework for Cloud Applications” https://biblio.ugent.be/publication/6850410/file/6850411.pdf.

[11] Robin Hecht and stefen Jablonski , “ NoSQL Evaluation : A Use Case Oriented Survey” , International conference on cloud and service computing , published by IEEE , 2011 http://rogerking.me/wpcontent/uploads/2012/03/DatabaseSystemsPaper.pdf

[12] Rabi Prasad Padhy and Deepti Panigrahy , “NoSQL Databases: State-of-the-art and security challenges” , international journal of Recent Engineering Science (IJRES), Volume 16 , October 2015. http://ijresonline.com/archives/volume-16/IJRES-V16P106.pdf

[13] Ralph Mietzner, Tbias Unger, Robert Titze and Frank Leymann, “Combining Different Multi-Tenancy Pattern in Service – Oriented Applications” proceedings of the 13th IEEE Enterprise Distributed Object Conference , Published by IEEE , 2009 , Pages 131-140 . ftp://ftp.informatik.unistuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2009-50/INPROC-2009-50.pdf

[14] Sanjeev Kumar Pippal and Dharminder Singh Kushwaha, “A Simple, adaptable and efficient heterogeneous multi-tenant database architecture for ad hoc cloud” , Journal of Cloud Computing, 2013 . http://link.springer.com/article/10.1186/2192-113X-5/fulltext.html

[15] Stefan Aulbach, Torsten Grust, Dean Jacobs and Jan Rittinger, “Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques”.http://db.in.tum.de/research/publications/conferences/sigmod2008-mtd.pdf

[16] Steve Strauch, Uwe Breitenbuecher, and Tobias Unger, “Cloud Data Pattern for Confidentiality” , Proceedings of the 2nd International Conference on Cloud Computing and Service Science, CLOSER 2012, Portugal, SciTePress , 2012. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.412.6515&rep=rep1&type=pdf

[17] Steve Strauch, Vasilios Andrikopoulos and Frank Leymann, “Using Patterns to Move the Application Data Layer to the Cloud”, Proceedings of the 5th International conference on Pervasive Patterns and Applications, PATTERNS 2013, Spain. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.412.5411&rep=rep1&type=pdf

[18] Tim Wellhausen and Andreas Fieber , “How to write a pattern ? A rough guide for first-time pattern authors” , November 2011. http://dl.acm.org/citation.cfm?id=2396721

[19] Yishan Li and Sathiamoorthy Manoharan, “A Performance Comparison of SQL and NoSQL databases”, published by IEEE, 2013.https://www.researchgate.net/profile/Yishan_Li/publication/261079289_A_performance_comparison_of_SQL_and_NoSQL_databases/links/564fbcf708aeafc2aab3ff73.pdf

[20]] [Manual] MongoDB CRUD Operations, Release 3.2.1. MongoDB Inc, February 2016. https://docs.mongodb.org/manual/MongoDB-crud-guide-master.pdf

[21] A MongoDB White Paper, “MongoDB Architecture Guide” http://s3.amazonaws.com/info-mongodb-com/MongoDB_Architecture_Guide.pdf