Benchmark Proposal for Multi-Tenancy in the Database Layer

Philipp Neugebauer[1], Christian Maier[2] and Alexander Bumann[3]

[1] innoQ Deutschland GmbH, Kreuzstraße 16, D-80331 München
philipp.neugebauer@innoq.com
[2] Information Systems and Services, University of Bamberg
christian.maier@uni-bamberg.de
[3] nubibase GmbH, Silbersteinstraße 14, D-97424 Schweinfurt
a.bumann@nubibase.de

Abstract. The cloud is often utilized with the hope to increase the IT budget efficiency. The cloud service model Software as a Service in combination with its key feature multi-tenancy shines here brightly, but its adoption is complicated by the multiplicity of possible solutions. In detail, multiple multi-tenancy implementations meet many possible database systems requiring a reliable comparison to find the optimal solution. This article briefly explains multi-tenancy, its benefits and implementations. It also indicates through the results of a literature review that no benchmark for multi-tenancy implementations exists. It proposes and describes a benchmark and its setup to gain reliable results of the disk space usage and performance.

Keywords: cloud, SaaS, multi-tenancy, database, benchmark

1 Introduction

An increasing number of companies try to address their business challenges and opportunities with their IT budget more efficiently in response to growing IT budgets or IT budget constraints.[1,2] Restrictively, big business solutions and services do not match with requirements and also budgets of small and middle-sized companies [1]. A possible solution to the problem of small and middle-sized companies is provided by the cloud, especially Software as a Service (SaaS) in combination with multi-tenancy. The application of multi-tenancy enables the better utilization of hardware and economies of scale for IT service providers through the pooling of more tenants on a single server [2, 3] which allows them to provide their services for a smaller price [2]. Small and middle-sized companies can then optimize or even reduce their IT budget through the adoption of these services.

[1] http://diginomica.com/2016/12/21/it-spending-and-staffing-in-2017-computer-economics-on-the-cios-cloud-predicament/
[2] http://www.zdnet.com/article/research-it-budgets-rising-for-many-in-2015/

Multi-tenancy is the primary feature of SaaS and enables the handling of multiple independent tenants in a single instance on a single hardware and software infrastructure while allowing individual configuration to the tenant's needs [3, 4, 5]. It captivates by the combination of efficiency improvement and the enabling of customization which provides a broad range of application targets. The most efficient application of multi-tenancy can improve the served number of tenants from about a dozen to more than hundred per machine and therefore enable huge cost savings [6, 7].

Companies have different possibilities to utilize multi-tenancy and thereby to improve their IT budget efficiency. Various approaches have been proposed in a lot of articles [1, 5, 7, 8, 9] for the adoption of multi-tenancy in the database layer proving an academic interest in the research field. Thus, diverse implementations for different database solutions are available, but even though some of them have been tested, contractionary results have been released which exacerbates the solution selection [6, 8, 9]. Unfortunately, neither a guideline for the selection of the best matching multi-tenancy implementation nor a summary of all implementations exists to simplify the identification of possible solutions. The development of such a guideline requires a reliable benchmark with criterias like disk space usage and performance to test and compare the multi-tenancy approaches to enable the selection of the best matching solution for the scenario.

This article proposes the setup for the benchmark so that the guideline for the selection of multi-tenancy approaches can be created. It covers first the different multi-tenancy implementations and reasons why the existing work cannot be used for the guideline development. In the following, the ideas for the settings and configuration of the benchmark for multi-tenancy approaches are explained. Lastly, the article is concluded and future work areas and plans are presented.

2 Multi-Tenancy Background and Related Work

This section briefly explains the three different levels of multi-tenancy in the database layer: *separate databases*, *shared database* and *shared table* approaches. It approaches their implementation opportunities and lastly justifies the required creation of a new benchmark.

Separate databases provide each tenant its own database instance that is adaptable to their needs but multiple tenants share the same machine [6, 10, 11]. The *shared database* provides each tenant its own tables, but all share the same database process [7, 11]. Lastly, *shared table* approaches [7, 11] preserve data from many tenants in the same tables and each row is marked with a tenant id [11]. Unlike *separate databases* and *shared database*, they represent a category of various implementations most efficiently addressing the problem of multi-tenancy adoption. They avoid data replication, support client customizations and encourage business changes while serving the highest number of tenants through their lowest overhead [6, 10]. On the downside, the solutions often turn the database into a dumb data repository and many features like query optimization or indexes must be reimplemented [11, 12]. Additionally, the security must be handled outside the database and generic columns can only used if sparse tables are handled efficiently [11].

In summary, 22 *shared table* approaches were identified in the literature review. The standardized benchmarking of *shared table* approaches is complicated by the required adaptions of the initial benchmark schema to each approach because they utilize custom schemas. Additionally, the merge of the database tables of all tenants requires an extendable approach schema and therefore the extensibility and the efficiency of its implementation must be determined. The different approach schemas may also vary about the efficiency of their disk space usage so that the disk space usage must be analyzed specifically for *shared table* multi-tenancy approaches. Lastly, it must be checked if the approaches can be implemented in the specific database system or must be excluded for some or all systems. The relational database systems to be benchmarked must be added to the complexity of the missing overview, resulting in a non trivial n to n problem.

All identified and implementable approaches need to be comprehensibly and repeatably compared by their disk space usage and performance to gain the information for the determination of the optimal implementation. Such a test environment is realized in a benchmark which are created in the articles of Krebs et al. [3] and Kiefer et al. [13] but they lack the full support for *shared table* approaches. Krebs et al. [3] developed an extension for the in the meantime as obsolete marked TPC-W[3] and is only able to test the very simple *basic table* approach. The MulTe framework designed by Kiefer et al. [13] can only examine *separate databases* and *shared databases*. Benchmark standards of the Transaction Processing Performance Council (TPC)[4], a community which provides a wide range of database benchmarks to enable effective performance comparisons between them, cannot be applied to *shared table* multi-tenancy approaches since they support only single tenant databases and therefore provide no tenant configurable schema. Hui et al. [2] based their test settings on the TPC-C and TPC-H frameworks but just briefly described them and also optimized them to their test scenario. However, some of their test specifications inspired this proposed benchmark and were adopted. In conclusion, no existing benchmark can be utilized for the testing of *shared table* multi-tenancy scenarios.

3 Benchmark Proposal

This section covers the proposed benchmark. The complete benchmark must be automatically executable and easily extendable to cover new *shared table* approaches to finally develop guidelines for the simple selection and implementation of the best matching approach. In the beginning, only the *shared table* approaches are addressed due to their best cost efficiency. First, the benchmark background is approached by the influences to this proposal. Second, the general setup of the benchmark is explained. Third, the test environment is specified in more detail. The settings for result measurement conclude the section.

Existing research of Hui et al. [2] and Aulbach et al. [8] inspired the setup of this benchmark. Hui et al. [2] contributed the foundation of TPC-H, concurrent access queries and the measurement of disk space usage and performance, while

[3] `http://www.tpc.org/tpcw/`

[4] `http://www.tpc.org/`

Aulbach et al. [8] also considered the concurrency aspect and partially contributed together with Heng et al. [6] and Yaish et al. [14] the query statement numbers. TPC-H was selected as benchmark base because it aims on concurrent data manipulation and data of large volumes and broad industry-wide relevance which are the characteristics of cloud scenarios. Moreover, the service for many customers leads to big data volumes and requires the handling of concurrent accesses. Since TPC-H approaches normal business requirements, has an industry-wide relevance and is easily implementable in relational databases, it also provides a fair benchmark base. The selection is further justified by Kiefer et al. [13] who also based their benchmark on TPC-H. TPC frameworks are currently not applicable to multi-tenancy approaches as already stated, so that only general guidelines of TPC-H could be adopted and were then applied to the *shared table* context: In the beginning, the database tables are created. Afterwards, the data is generated and lastly the test queries are executed.

Mapping these guidelines to the multi-tenancy context results in the initial setup of all tenants' general schema. The test data is generated and populated into the database to ensure the same base data for all tested approaches to avoid any result biasing. In the next step, the specific approach tables regarding the tenants' requirements are created and the base data is mapped into them. In the last step, the concurrent benchmark queries are executed and benchmarked.

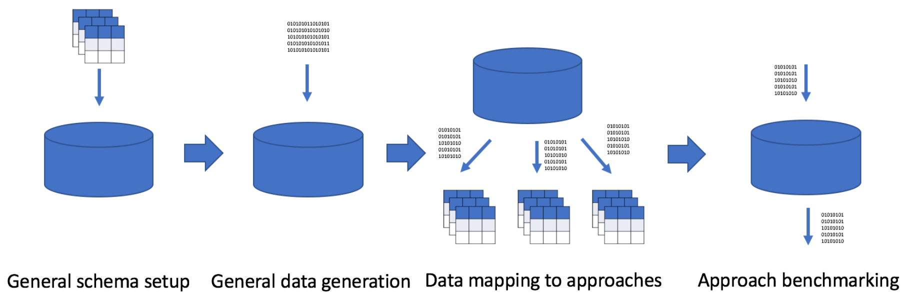

Fig. 1: Benchmark Process

The schema is specifically chosen for the testing of multi-tenancy and comprises only the three tables orders, lineitems and parts to simulate multi-tenant characteristics. The schema of TPC-H is therefore simplified and to multi-tenancy adapted to focus on the determination of the performance in terms of response time allowing for the extensibility of the approaches. The targeted scenario is a service provider who serves multiple small clients by hosting and managing their data and aims for the long-tail of business applications. It does not change the stored data volume of the big single tenant application of the TPC-H but splits them between between many tenants. While only the shared attributes of all tenant groups are displayed in Figure 2, the schemas of the other groups are enriched for their use cases. The benchmark scenario will contain 500 tenants, each having ordered 3,000 times with four lineitems. The catalog contains 187,500 parts so that the benchmark will run against more than 7.5 million records in total. The tenant number is based on the assumption that the usage of *shared table* approaches must provide a significant increase of served tenants to warrant the effort for its adoption. The other ratios are taken from the TPC-H benchmark whose selection is justified in section 2. The lineitem table is used to regard the different tenant requirements and has a customized column length for each tenant group. Five tenant groups are introduced in the tests meeting different business requirements which results in varying table column numbers. Therefore, also different ratios of these groups will be considered to represent multiple business scenarios. The first group uses only a very small subset of the possible attributes, the second employs just slightly less attributes than the third and normal group which utilizes the general schema. The fourth group lacks a bit more attributes than the normal group and the fifth group utilizes much more attributes than all other groups to satisfy their business requirements.

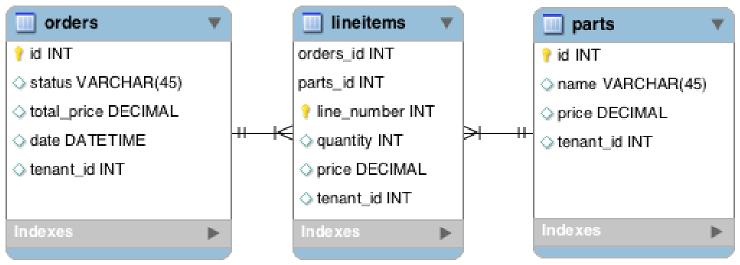

Fig. 2: Evaluation Schema

The measurement and resulting metrics are conducted with the execution of SELECT, INSERT, UPDATE and DELETE queries. SELECT queries must be more often executed to provide helpful results leading to 100 and 1,000 record statements, while INSERT, UPDATE, DELETE queries allow already good comparisons at 50 and 100 records. As already stated, these query numbers were already used and contributed by Heng et al. [6], Aulbach et al. [8], and Yaish et al. [14] and are therefore considered a fair benchmark base. Specifically, Heng et al. [6] run 1,000 record SELECTs, Aulbach et al. [8] executed 50 and 1,000 record SELECT queries as well as 50 INSERT and 100 UPDATE statements, while Yaish et al. [14] performed 100 record queries. The benchmark queries are inspired by the queries of TPC-H but primary focus on the performance testing of the characteristics of *shared table* approaches or multi-tenancy instead of specific business logic. Therefore, the queries of TPC-H were analyzed and their characteristics like JOINs and aggregations taken to model these benchmark's queries which are listed below. In consequence, queries must be executed on both the attributes shared by all tenant groups and custom extension attributes since they are a key feature for the enabling of multi-tenancy and their impact needs to be analyzed. Same and different table queries must be executed to reveal the load balancing capability of the multi-tenancy approaches. The seven benchmark queries are in the following introduced and reasoned for their capability to expose the support for specific multi-tenancy features. Placeholders like x, y or custom_column will be replaced by real values in the executed queries, but are used to exemplify the targets of the queries.

Query 1: All line items of a specific tenant with a custom column value of y
Reason: Same table query aiming at extension support and specifically index support for extension fields

```sql
SELECT * FROM lineitems WHERE tenant_id = x AND
    custom_column = y
```

Query 2: All line items with summed prices, grouped by the order of a tenant and ordered by their sum and quantity
Reason: Different table query to reveal aggregation and join efficiency and load balancing capabilities

```sql
SELECT orders.id as order_id, orders.tenant_id, orders.
    total_price, sum(lineitems.price) as lip, sum(
    lineitems.quantity) as liq FROM orders, lineitems
WHERE orders.id = lineitems.orders_id AND orders.
    tenant_id = lineitems.tenant_id
GROUP BY orders.id ORDER BY lip DESC, liq DESC
```

Query 3: All parts that are used by an order and fulfill a custom value comparison ordered by their usage count
Reason: Different table query to determine aggregation and index support as well as load balancing capability and extension field efficiency

```sql
SELECT parts.id, COUNT(orders.id) as oc FROM parts,
    orders, lineitems
WHERE orders.id = lineitems.orders_id AND parts.id =
    lineitems.parts_id AND custom_column > x
GROUP BY parts.id ORDER BY oc DESC
```

Query 4: All closed orders of the last year filtered through LIKE
Reason: Same table query for complex clause and index support

```sql
SELECT * FROM orders WHERE orders.date >
    current_timestamp - interval '1 year' AND orders.
    status LIKE 'clo\%'
```

Query 5: Inserting new orders and line items
Reason: Different table query for determining load balancing capability

```sql
INSERT INTO orders VALUES ...
INSERT INTO lineitems VALUES ...
```

Query 6: Updating orders and line items
Reason: Different table query for determining load balancing capability

```sql
UPDATE orders SET colum=new_value WHERE tenant_id = x
UPDATE lineitems SET colum=new_value WHERE tenant_id = x
```

Query 7: Delete parts of a specific tenant
Reason: Same table query for performance determination

```
DELETE FROM parts WHERE tenant_id = x
```

A practically relevant SaaS scenario requires additionally the concurrent execution of tenants' queries which is met through queries of a configurable number of tenants considering the tenant groups' ratio. The minimum concurrent load will be 10% of the tenants and can be increased up to 50%. As result, the strength and weaknesses of the approaches regarding the disk space usage as well as the response time of reads and writes can be demonstrated and analyzed.

4 Conclusion

A benchmark of multi-tenancy approaches and its results enable the creation of a guideline which simplifies the decision to adopt the best approach and thus avoids costly wrong decisions. This article indicates through the results of a literature review that no benchmark of multi-tenancy implementations exists. Therefore, it proposed the setup for a reliable and useful multi-tenancy benchmark for *shared table* approaches to close the identified information gap. The benchmark is inspired by existing work and seized their ideas for a general *shared table* benchmarking scenario to ensure meaningful results for all identified approaches in relevant scenarios. Future research does not need to collect basic information about multi-tenancy anymore and can either extend the proposed benchmark or start its development. The development focuses on relational database systems like MySQL and PostgreSQL, but the support for NoSQL databases is considered as a possible and useful option because the latter enables the utilization of new technologies which cannot be realized in relational databases. The benchmark will be theoretically tested against the findings of Sim et al. [15] and Huppler [16]. Beside the inclusion of *separate databases* and *shared databases*, the benchmark could also consider the handling of metadata and middleware in future to take these effects into account.

References

[1] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. "Native Support of Multi-tenancy in RDBMS for Software as a Service". In: *Proceedings of the 14th International Conference on Extending Database Technology*. EDBT/ICDT '11. Uppsala, Sweden: ACM, 2011, pp. 117–128.

[2] M. Hui, D. Jiang, G. Li, and Y. Zhou. "Supporting Database Applications as a Service". In: *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*. 2009, pp. 832–843.

[3] R. Krebs, A. Wert, and S. Kounev. "Multi-tenancy Performance Benchmark for Web Application Platforms". In: *Proceedings of the 13th International Conference on Web Engineering*. ICWE'13. Aalborg, Denmark: Springer-Verlag, 2013, pp. 424–438.

[4] T. Kwok, T. Nguyen, and L. Lam. “A Software As a Service with Multi-tenancy Support for an Electronic Contract Management Application”. In: *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2*. SCC ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 179–186.

[5] H. Yaish, M. Goyal, and G. Feuerlicht. “An Elastic Multi-tenant Database Schema for Software as a Service”. In: *IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC 2011, 12-14 December 2011, Sydney, Australia*. IEEE, 2011, pp. 737–743.

[6] L. Heng, Y. Dan, and Z. Xiaohong. “Survey on Multi-Tenant Data Architecture for SaaS”. In: *International Journal of Computer Science Issues*. Vol. 9. 2012.

[7] W. Lehner and K.-U. Sattler. “Virtualization for Data Management Services”. English. In: *Web-Scale Data Management for the Cloud*. Springer New York, 2013, pp. 13–58.

[8] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold. “A Comparison of Flexible Schemas for Software As a Service”. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’09. Providence, Rhode Island, USA: ACM, 2009, pp. 881–888.

[9] S. Pippal and D. Kushwaha. “A Simple, Adaptable and Efficient Heterogeneous Multi-tenant Database Architecture for ad hoc Cloud”. English. In: *Journal of Cloud Computing* 2.1, 5 (2013).

[10] F. Chong, G. Carraro, and R. Wolter. *Multi-Tenant Data Architecture*. MSDN Library, Microsoft Corporation. 2006.

[11] D. Jacobs and S. Aulbach. “Ruminations on Multi-Tenant Databases”. In: *BTW Proceedings, volume 103 of LNI*. GI, 2007, pp. 514–521.

[12] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper. “Extensibility and Data Sharing in Evolving Multi-tenant Databases”. In: *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*. Ed. by S. Abiteboul, K. Böhm, C. Koch, and K. Tan. IEEE Computer Society, 2011, pp. 99–110.

[13] T. Kiefer, B. Schlegel, and W. Lehner. “MulTe: A Multi-Tenancy Database Benchmark Framework”. In: *Selected Topics in Performance Evaluation and Benchmarking*. Vol. 7755. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 92–107.

[14] H. Yaish, M. Goyal, and G. Feuerlicht. “A Proxy Service for Multi-tenant Elastic Extension Tables”. In: *T. Large-Scale Data- and Knowledge-Centered Systems* 9070 (2015), pp. 1–33.

[15] S. E. Sim, S. Easterbrook, and R. C. Holt. “Using Benchmarking to Advance Research: A Challenge to Software Engineering”. In: *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society. 2003, pp. 74–83.

[16] K. Huppler. “The Art of Building a Good Benchmark”. In: *Performance Evaluation and Benchmarking*. Springer, 2009, pp. 18–30.