ATLANTIS PRESS

Advances in Computer Science Research, volume 87

3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)

A New Table Structure based on Universal Table Layout Schema-Mapping Technique

Linfeng Yin [1, a], Xiaocong Zhou [1, b], Xinming Wang [2, c]

[1] School of Data and Computer Science Sun Yat-sen University Guangzhou, China.

[2] School of Computer, South China Normal University Guangzhou, China.

[a] yinlf3@mail2.sysu.edu.cn, [b] isszx@mail.sysu.edu.cn, [c] roadlit@gmail.com

Abstract. In SaaS (Software-as-a-Service) systems, the database is typically implemented using the Universal Table Layout schema-mapping technique, in which all tenant data is mapped to a table, thus valid indexes cannot be created directly in the table. To enable to create valid indexes directly in the table, this paper proposes a new table structure for better database performance. In this paper, the experiments are designed to verify the conclusions of this paper. The experimental results show that the table structure of this paper can obtain better database performance.

Keywords: SaaS System, Universal Table Layout, valid index, database performance.

1. Introduction

In the SaaS (Software as a Service) system [1], the management of the application is the responsibility of the service provider. Tenants only need to use the provided service remotely through the network [2]. Tenants do not need to develop and maintain their own system, which greatly reduces the overall cost. Due to the low cost of using the SaaS system, more and more SMEs (Small and Medium-Sized Enterprises) are attracted. IDC estimates that SaaS service fees in the global market reached $3.98 billion in 2006 and $14.5 billion in 2011, with a compound annual growth rate of 30% [3]. The rapid growth of SaaS services has had a considerable impact on the software market [4].

The SaaS system aims to reduce the cost of tenant through economies of scale. The multi-tenant structure can minimize the overall cost of the SaaS system. In a multi-tenant structure, multi-tenant services are unified into one operating system [5]. For highly scalable web applications, application servers are often stateless, so the multi-tenancy of services is always reflected in the database layer [6].

The multi-tenant structure reduces the cost of the SaaS system, but it also brings some problems. The first is the contention of shared resources [7], the second is to reduce security, and finally the structure is difficult to support the scalability of the application. Scalability is necessary for enterprise applications, and many hosted business services provide platform to build and share such extensibility [8, 9, 10].

To implement a multi-tenant structure, most managed services use query transformation to map multiple single-tenant logical schemas of the application to a multi-tenant physical schema of the database. In the case of a certain workload, this method is scalable. The limit depends on the number of tables the database can process, which depends on the amount of available memory [7]. To alleviate the limited problem of tables that the database can handle, it is necessary to share tables among tenants, which will affect the ability of the tenant to extend the application. A better solution is to map the logical table to a generic table, such as the Universal table [11]. Based on this method, a widely used schema-mapping technique, namely Universal Table Layout [12], is introduced. This schema-mapping technique maps tenant data to a table. Since each data row belongs to a different logical table, valid indexes cannot be directly created in the table to speed up the execution of the SQL [13, 14]. To solve this problem, the index function is usually implemented indirectly by creating a new index table. But this will have one more table query, which will affect the database performance.

To reduce the performance loss of the database, this paper proposes to modify the table structure of the Universal Table Layout schema-mapping technique, and create a valid index directly in the table. This article creates multiple tables to store tenant data based on index type and counts of tenant logical tables. Improving efficiency of SQL execution by adding index fields to tables.

Although the table implementation method in this paper will reduce the sharing degree of data, when the shared data is enough, the performance of the database is reduced to a level that the tenant can't bear. To ensure the system performance, the database will inevitably reduce the data sharing degree. Therefore, the impact of the proposed method on data sharing is within acceptable limits. If the amount of tenant data managed by the database system is large enough, the impact on data sharing will be further reduced.

This paper proposes to modify the table structure of the Universal Table Layout schema-mapping technique, which not only improves efficiency of SQL statements execution, but also simplifies SQL statements translation. In the table structure of the Universal Table Layout schema-mapping technique, the index table is mainly used to record the row ID of the logical table data stored in the physical table, thereby indirectly implementing the index function. If you execute a logical SQL statement, you need to translate the logical SQL statement into a SQL statement. In the way of implementing the index function indirectly through the table, first, you need to manipulate the index table and then manipulate the data table, which makes the difficulty of SQL translation more difficult than the translation of a single table.

To verify the validity of the proposed method, two experiments were organized in this paper. In the first experiment, *benchmark SQL* [15] is used to test the performance of the database under the two table structures. The test is run under a variety of different data scales. The experimental results show that the table structure of this paper can obtain better database performance. In the second experiment, the paper explores the performance of the four database operations, e.g. add, delete, update and insert. The test is performed under three indexes, namely, primary key, unique index, and normal index. The experimental results show that the table structure of this paper can obtain better performance.

Generally speaking, the main contributions of this paper are as follows:

1) This paper proposes a new table structure, which adds index fields directly to tables, which improves efficiency of SQL execution and simplifies SQL translation.
2) This paper proves the validity of our method through experiments. The experiments verify the conclusion of this paper from the whole and the part.

The remaining chapters of this paper are organized as follows: Section 2 introduces the table structure of Universal Table Layout schema-mapping technique in detail; Section 3 describes the research motivation of this paper, and specifically describes the table structure of this paper; Section 4 mainly completed the experiments, and analyzed the experimental results; Section 5 is the summary and outlook.

2. Table Structure of Universal Table Layout

This section focuses on the table structure of Universal Table Layout schema-mapping technique. This schema is derived from Universal Relation [16], which was originally proposed as a conceptual tool for developing queries and is not intended to be implemented directly.

Under the Universal Table Layout schema-mapping technique, it mainly includes three data tables *mt_tables*, *mt_fields* and *mt_data*. The *mt_*tables is used to record the correspondence between the tenant ID and the logical table, where the tenant ID, the table name, and the table ID are mainly stored; the mt_fields is used to record the field information of the logical table, where the field name, column number of field, and field type are stored; mt_data is used to store tenant data, including table ID, and row data. The structure of these three tables is shown in Table 1, Table 2, and Table 3.

Table 1. Table Structure of mt_tables

Field	Type	Illustrate
tableId	int	table_ID
tableName	varchar	table_name
tenantId	int	tenant_ID

Table 2. Table Structure of mt_fields

Field	Type	Illustrate
id	int	primary key
tenantId	int	tenant ID
tableId	int	logical table ID
value0	varchar	data column
.....	.....	.....
value499	varchar	data column

In mt_data, there are id, tableId, tenantId, and many reserved fields, which are used to store logical table row data. 500 fields are reserved in the most popular SaaS system, salesforce.com, to accommodate row data for any logical table.

In the table structure of Universal Table Layout schema-mapping technique, since the table mt_data stores data of different tenant logical tables, a valid index cannot be directly created in mt_data to quickly operate the data table. Therefore, to improve the efficiency of SQL execution, the index table is usually created to indirectly implement the index function. The row ID of the logical table index data row in the table mt_data is recorded in the index table.

Table 3. Table Structure of mt_data

Field	Type	Illustrate
id	int	primary key
dataType	varchar	field type
fieldName	varchar	field name
fieldNum	int	column number
tenantId	int	tenant ID
tableId	int	table ID
isIndex	int	index column

Table 4 is an index table structure. In the index table, mtDataId is the row Id of the logical data row in mt_data, and intValue1 and stringValue1 are respectively used to store integer and string index column of the logical table. There are zero or more intValue fields and stringValue fields in the index table.

Table 4. Table Structure of index table

Field	Type	Illustrate
id	int	primary key
tenantId	int	tenant ID
tableId	int	table ID
mtDataId	int	record ID
intValue1	int	index field
.....	.....	.....
intValueN	int	index field
stringValue1	varchar	index field
.....	.....	.....
stringValueN	varchar	index field

In logical tables, there are mainly three types of indexes, namely, primary key, normal index, and unique index. According to logical table, multiple index tables are created. For example, there are three logical tables whose primary keys are composed of three *INT* type columns, two *INT* type columns, and one *INT* type column. When creating index tables, three index tables need to be created to store corresponding index data. For normal indexes and unique indexes, the index table is similar to the primary key, and will not be described one by one.

3. Research Motivation and Methods

3.1 Research Motivation

In the Universal Table Layout schema-mapping technique, tenant data is mapped to a table. To improve the efficiency of SQL statements execution, the index function is often implemented indirectly by creating an index table. This approach speeds up SQL statement execution, but performs poorly compared to indexes directly in the table. This is because when this approach executes SQL statement, you need to operate the index table before operating the table. Therefore, the database performance is not high.

To improve the efficiency of SQL statements execution, this paper proposes to modify the table structure based on the table structure of the Universal Table Layout schema-mapping technique, and put data of same index type and counts in logical table into a physical table, which can create index directly in the physical table.

3.2 Research Methods

This part describes in detail how to modify the table structure of the Universal Table Layout schema-mapping technique to achieve the purpose of improving efficiency of SQL statements execution and reducing the difficulty of logical SQL translation. This paper proposes to create multiple data tables according to the index type and counts of logical tables, and map data of the logical tables to the corresponding table. The number of tables is determined by index types and counts of logical table, and the number of tables must be limited because the number of index types and counts in logical table is limited. The tables proposed in this paper are mainly divided into two categories, one for storing data with a primary key in logical tables and one for storing data without a primary key in logical tables. The table structure of the two types of tables is as follows.

Table 5. Table Structure of Data Table

Field	Type	Illustrate
id	int	primary key
tenantId	int	tenant ID
tableId	int	table ID
index1	varchar	normal index
……	……	……
indexN	varchar	normal index
unique1	varchar	unique index
……	……	……
uniqueN	varchar	unique index
value0	varchar	data column
……	……	……
value499	varchar	data column

Table 6. Table Structure of Data Table

Field	Type	Illustrate
tenantId	int	tenant ID
tableId	int	table ID
pk	varchar	PK
index1	varchar	normal index
……	……	……
indexN	varchar	normal index
unique1	varchar	unique index
……	……	……
uniqueN	varchar	unique index
value0	varchar	data column
……	……	……
value499	varchar	data column

The table structure of Table 5 is used to store data without a primary key in the logical table. The table structure of Table 6 is used to store data with primary key in the logical table. In these two table structures, the unique index column and the normal index column can have zero or more columns, which are determined according to the logical table. This paper defines the index column in the table as a varchar type, which can store any type of data to the index data column.

In the logical table, if the primary key is a composite primary key composed of multiple columns, when the data of logical table is stored in the physical table, the primary key needs to be concatenated into a string by a separator and stored in the *pk* column of the table. For the unique composite index and the normal composite index, the same method is used for processing.

In logical tables, if the logical SQL statements is executed using a partial prefix column of the composite index, the composite index will speed up execution of SQL statements because the index usage follows the leftmost prefix principle. The index implementation method proposed in this paper does not affect the use of partial prefix columns in the composite index to speed up the execution of SQL statement, but only needs to use the database keyword to achieve this function, such as the use of the *LIKE* keyword in MySQL.

The table structure in Table 5 needs to retain the primary key id, and the table structure in Table 6 uses the logical table primary key and the tableId as the primary key. It is not necessary to retain the primary key id, and directly use (tableId, pk) as the primary key of the table. The following is an example of the specific storage, as follows, there are 10 tables, the table structure is as follows.

table1(id1, id2, index1, filed1);
table2(id1, id2, index1, index2, field1);
table3(id1, id2, field1);
table4(id1, id2, id3, index1, field1);
table5(id, index1, field1);
table6(id, index1, index2, field1);
table7(id, uniqueIndex1, index1, field1);
table8(id1, id2, id3, uniqueIndex1, index1, field1);
table9(id1, id2, id3, uniqueIndex1, uniqueIndxe2, index1, field1);
table10(index1, field1);

In the logical tables, the id column is primary key, the index column is normal index, the uniqueIndex column is unique index, they consist of at least one field. The data in the 10 logical tables above will be mapped to 6 tables, whose structure is as follows.

①mt_data_pk1 (tenantId, tableId, pk, index1, value0, ..., value499)
②mt_data_pk2 (tenantId, tableId, pk, index1, index2, value0, ..., value499)
③mt_data_pk3 (tenantId, tableId, pk, value0, ..., value499)
④mt_data_pk4 (tenantId, tableId, pk, uniqueIndex1, index1, value0, ..., value499)
⑤mt_data_pk5 (tenantId, tableId, pk, uniqueIndex1, uniqueIndex2, index1, value0, ..., value499)

⑥mt_data1 (id, tenantId, tableId, index1, value0, ..., value499)

The data of TABLE1, TABLE4 and TABLE5 will be mapped to ①, TABLE2 and TABLE6 to ②, TABLE3 to ③, TABLE7 and TABLE8 to ④, TABLE9 to ⑤, TABLE10 to ⑥. The data in logical tables are mapped to corresponding data tables if index type and number of logical table is the same as the data table one.

When the data of logical table is mapped to the physical table, the index data column of logical table should be mapped to the index column of the physical table, and mapped to the corresponding data column, if there is a logical table below.

userInfo (stuNo, name, age, address)

stuNo is the primary key. If data of the table is mapped to the physical table, it will be mapped to the table of the following table structure.

mt_data (tenantId, table1*, pk1, value0, value1, ..., value499)*

pk1 will store the data corresponding to stuNo, value0 will store the data corresponding to stuNo, value1 will store the data corresponding to name, value2 will store the corresponding data, and value3 will store the data corresponding to address.

4. Experiments

In experiment 1, the *benchmarkSQL* would be used to perform performance tests. It simulates a wholesaler's cargo management environment, the system processes the order request and tests the number of requests that the database can process per unit time. Before testing the performance of the database, we should translate the logical SQL into physical SQL. The experimental result is shown in FIG 1.

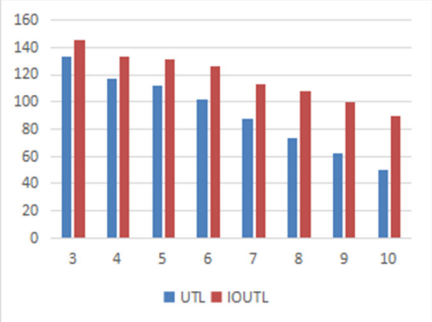

Fig 1. Database Throughput

In FIG 1, the blue bar indicates the result of table structure of Universal Table Layout schema-mapping technique, and the red bar indicates the result of table structure in this paper. From these results, it can be found that the throughput of table structure presented in this paper is better, because SQL statements execution does not need to operate the index table. In Universal Table Layout schema-mapping technique, when we execute SQL statements, we need to get row ID from index table, and then execute them based on row ID. Under the table structure of this paper, SQL statements are executed directly, which is more efficient.

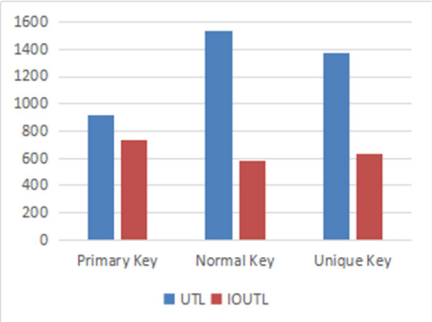

Fig 2. Select SQL Execute Time

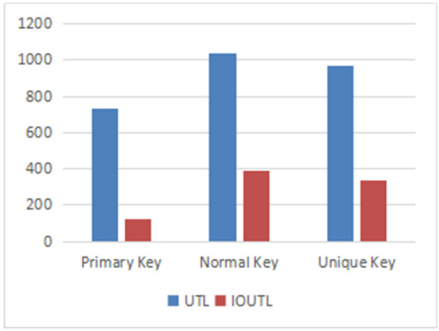

Fig 3. Update SQL Execute Time

In experiment 2, we calculate the execution time of 100 add, delete, update, select statements separately. The shorter they execute, the higher the performance. And the experimental results are shown in FIG 2, 3, 4 and 5, which correspond to select, update, delete, insert operations respectively. Through experimental results, we can conclude that the table structure presented in this paper gain better performance on four kinds of operation. For select, update, delete operations, we tested performance under three indexes (normal index, primary key and unique index). In the insert operation, for the table structure of Universal Table Layout schema-mapping technique, we tested the runtime of both the update index table and the non-update index table.

In the table structure of this paper, better database performance because we do not need to operate the index table. which will save SQL statements execution time.

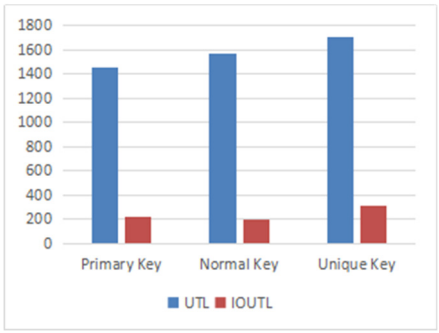

Fig 4. Delete SQL Execute Time

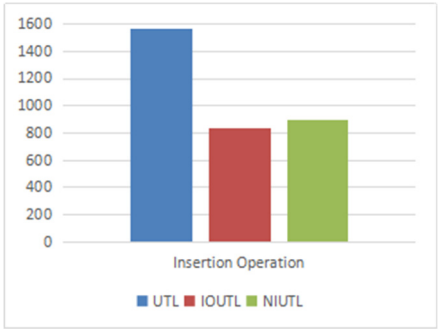

Fig 5. Insert SQL Execute Time

5. Conclusion

In this paper, based on table structure of Universal Table Layout schema-mapping technique, this paper proposes a new table structure. This table structure of this paper has better performance. To verify the conclusions of this paper, two experiments are organized. First, this paper tests performance through a real-world business environment to verify overall performance. Second, this paper verifies the performance from four operations, namely, *ADD*, *DELETE*, *UPDATA* and *SELECT*. This proves that the performance of the table structure of this paper is better than the table structure of Universal Table Layout schema-mapping technique.

Although the table structure of this paper can get better database performance, for some extreme cases, the table structure in this paper is not suitable. If the index type and counts of all logical table are different, in this case, if the tables are created by the method proposed in this paper, the sharing degree of the data will be seriously reduced. Of course, these extremes will hardly occur. We will continue to explore whether there is a better table structure for managing tenant data, ensuring the efficiency of tenant operations while keeping costs as low as possible.

Acknowledgements

This work is supported by the Important Science and Technology Specific Project of Guangdong Province of China(Grant No. 2016B030305006).

References

[1]. Kim W, Lee J H, Hong C, et al. An innovative method for data and software integration in SaaS[J]. Computers & Mathematics with Applications, 2012, 64(5):1252-1258.

[2]. Aulbach S, Jacobs D, Kemper A, et al. A comparison of flexible schemas for software as a service[C]// ACM SIGMOD International Conference on Management of Data. New York: ACM, 2009:881-888.

[3]. E. TenWolde. Worldwide Software on Demand 2007-2011 Forecast: A Preliminary Look at Delivery Model Performance, IDC No. 206240, 2007. IDC Report.

[4]. Aized Amin Soofi, M. Irfan Khan, et al. Security issues in SaaS delivery model of Cloud Computing[J]. International Journal of Computer Science and Mobile Computing, 2014,3(3):15-21.

[5]. Hui M, Jiang D, Li G, et al. Supporting Database Applications as a Service[C]// IEEE International Conference on Data Engineering. New York: IEEE Computer Society, 2009:832-843.

[6]. J. R. Hamilton. On designing and deploying internet-scale services[C]// In Proceedings of the 21th Large Installation System Administration Conference, Texas: USENIX, 2007:231–242.

[7]. Anatomy of MySQL on the GRID. http://blog.mediatemple.net/weblog/ 2007/01/19/anatomy-of-mysql-on-the-grid/.

[8]. NetSuite NetFlex. http://www.netsuite.com/portal/products/netflex/main. shtml.

[9]. Salesforce App Exchange. http://www.salesforce.com/appexchange/about.

[10]. WebEx. http:// www. webex.com/.

[11]. Aulbach S, Grust T, Jacobs D, et al. Multi-tenant databases for software as a service: schema-mapping techniques[C]// ACM SIGMOD International Conference on Management of Data, Canada: Vancouver, 2008:1195-1206.

[12]. D. Florescu and D. Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Technical report, Inria, France, 1999.

[13]. The Force.com Multitenant Architecture. https://www.salesforce.com.

[14]. Schwartz B. High Performance MySQL[J]. Oreilly Media, 2004(11).

[15]. TPC-C on-ling transaction processing benchmark. https://sourceforge. net/ projects / benchmarksql/.

[16]. D. Maier and J. D. Ullman. Maximal objects and the semantics of universal relation databases. ACM Trans. Database Syst., 8(1):1–14, 1983.