Dynamic Constraint Definition Method in Cloud Databases

Hequn Xian, Jing Li, and Xiuqing Lu
College of Information Engineering, Qingdao University
Qingdao, China
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
E-mail: xianhq@126.com

***Abstract*— A dynamic database constraint definition method is presented based on metadata design, which facilitates client-side configuration of cloud database schema and consistency definition. The proposed scheme simplifies the conceptual schema designing process for cloud databases and offers more data administrative privileges to the user. It also enables the user to create new data tables and generate standard data access interfaces dynamically. A typical application of our method is introduced which solve the problem of dynamic factor management in a teaching management database. The implementation and application show that our method is effective and applicable.**

Keywords- dynamic constraint; cloud database; metadata; user-defined constraint

I. INTRODUCTION

Traditional database application development involves requirements analysis, conceptual model design, logical model design, development, testing and other steps. The predefined database structure approach is widely used. For applications based on cloud databases, the relational schema is pre-designed, the applications can only perform operations such as record adding, deleting and updating. There is no way for the users to change the database schema without changing the codes and deploying them to the cloud again. Despite of its advantages like high consistency and fine formalization, such predefined static manner cannot meet the need for new data constraint definition or special business logic changing, which incur even more cost in a cloud database context. Therefore, a dynamic database schema designing method, allowing users to change the relational schema and define data constraints after deployment, can enhance the flexibility and scalability of the database system. Various application requirements can be fulfilled without post deployment development, which can only be achieved via dynamic design techniques.

In this paper, we devised a metadata based design method, which realizes dynamic constraint definition without executing traditional SQL statements against the cloud database server. Data administrators can define multi-value constraints with user interfaces and apply them to columns when they create new tables.

In the following sections of this paper, we present the scheme of database design with dynamic constraint definition and automatic data access interface generation. Then we introduce an application of our techniques in a teaching management database system.

II. RELATED WORKS

When designing the conceptual schema of a future information system, it is crucial to define a set of constraints that will guarantee the consistency of the subsequent database once it is implemented and operational [1]. Ravi et al. present an interactive way of eliciting database constraints. Luca et al. study the problem of security and consistency for cloud databases. They propose an alternative architecture that eliminates the need for a trusted intermediate server [2]. In the field of dynamic databases, Sergey introduces a new paradigm with virtualization techniques [3]. Zhao et al. apply dynamic database techniques to web mapping service in the Internet [4]. Dynamic database design techniques are also used in genetic information processing [5]. Ralf et al. study the application of dynamic database design in mobile applications [6]. Peter et al. identify the problem of entity resolution as the process of matching records, they propose an approach that adaptively adjusts similarities between records depending upon the values of the records' attributes and the time differences between records [7]. The problem of auditing inference based disclosures in dynamic databases is studied in [8], a sound and complete algorithm to determine a suspicious query set for a given domain knowledge is presented. Peter et al. address the problem of resource discovery model in multi-provider cloud databases. A software abstraction layer is used to discover the most appropriate infrastructure resources for a given application by applying a two-phase constraints-based approach to a multi-provider cloud environment [9].

III. DYNAMIC CONSTRAINT DEFINITION

In a typical information management system based on cloud databases, privileges are usually categorized into several roles. The goal of our design is to allow users with data administrative privileges to dynamically define database schema, including creating new tables and data access interfaces, so that users with ordinary data access privileges can read and edit tuples in the dynamically created tables. Constraints are needed to restrict the values that the users are allowed to update to a certain column of tuples.

In a traditional database application design, creating a constraint requires SQL statement execution. In the cloud database scenario, it is inappropriate even for the administrative users to execute SQL statements directly to create tables and to enforce constraints. To access data in a dynamically created table, user interfaces are needed, which should be generated with the table accordingly.

Our design replaces the traditional constraint definition method with a dynamic approach for cloud databases. We devise a set of pre-defined metadata tables to record the information about dynamically created tables, available constraints and the interrelation between columns and constraints.

Three metadata tables are involved in our scheme as shown below.

TABLE I. METADATA RELATION FOR USER-DEFINED RELATIONS

Column	Data Type	Description
MRID	Int32	Relation ID
MRalias	Nvarchar(20)	User alias
MRcolumn	Int32	Number of col
MRdate	Datetime	Creation time
MRnote	Nvarchar(50)	Other description

In table I, each tuple represents a dynamically created table. MRID is a unique auto-increasing value assigned to each table, which is also used to construct the table name when the system performs CREATE TABLE statement. MRalias is a user-defined meaningful alias for the table. MRcolumn denotes the number of columns in the table. MRdate is the date on which the table is created. Further description about the table is recorded in MRnote.

TABLE II. METADATA RELATION FOR USER-DEFINED ATTRIBUTES

Column	Data type	Description
MAID	Int32	ColumnID
MAalias	NVarchar(20)	User alias
MRID	Int32	Owner relationID
MAConstraint	Int32	Constrain ID
MAnote	NVarchar(50)	Other description

In table II, each tuple represents a column in a dynamically created table. MAID is a unique auto-increasing value for every column. MAalias is the column's user-defined alias. MRID is the foreign key referring to the table that the column resides. A zero value in the MAConstraint field indicates no constraint for the column, and a non-zero value means there is a constraint for the column when users edit the values in the tuples.

TABLE III. METADATA RELATION FOR USER-DEFINED CONSTRAINT

Column	Data type	Description
MCID	Int32	Constrain value ID
MCalias	NVarchar(20)	Constrain value alias
MAConstraint	Int32	Constrain group ID
MCValue	NVarchar(50)	Other description

Tuples in table III represent constraint values, which are divided into groups by their values in the MAConstraint attribute. A constraint consists of a group of values, and can be bound to one or more columns in any dynamically created tables. Setting a tuple's value of MAConstraint in table II to a certain MAConstraint value in table III means that only the chosen constraint values are valid for that column. However, administrative users can configure the values in any constraint group, resulting in an instantaneous enforcement of valid values for all the columns bound with the constraint group.

IV. IMPLEMENTATION AND APPLICATION

We implement the method with Microsoft .NET framework and embedded it in an actual cloud database application, which is a teaching management database system for a university. The teaching management system consists of six primary modules, which are system configuration, status data collection, database maintenance, factor management, assessment and data query. The core module is data collection module which facilitates the comprehensive assessment of teaching activities in all teaching unit. The assessment is base on a set of quantified factors. The summary of all the factors multiplied by their corresponding weight is the final score of each teaching unit. The choosing and configuration of the factors embraces the principles and emphasis of teaching management.

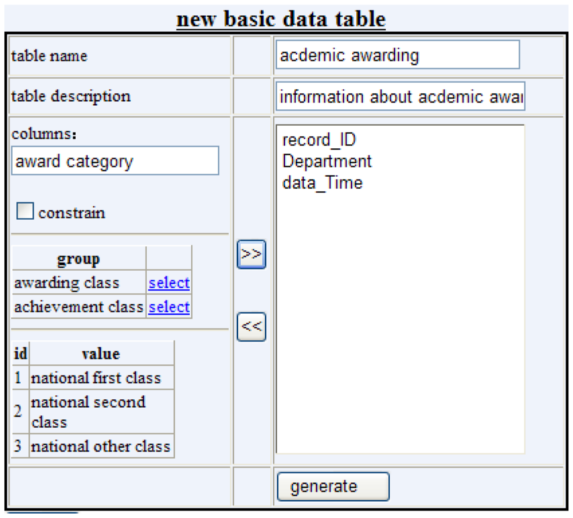
There are more than 50 factors in the system for now, divided into 4 categories. Each factor has its own quantifying method and calculation formula. It may be a calculated result from many other factors or from basic data tables. With the development of teaching management and the adjustment of long term goals of the university, the factors might inevitably be reconfigured. New factors are bound to be added, and new tables will be created. In that case, the database schema and all the pre-define application logic will have to be changed, resulting in a post deployment development, which would introduce cost in both human resource and financial resource.

Our method solves the problem of factor management by allowing administrative users to create new factor tables or base data tables. Constraints can be defined and applied at any point in the process. Ordinary users with only data access privileges can read and edit data in those tables.

The system is developed with Microsoft Visual Studio, and the code is written in ASP.NET with C# as the script language. We implement the database design in Microsoft SQL Server 2008 Express.

The name of the dynamically created table is generated from the maximum value of table IDs in the metadata, preceded by a short string with two characters to conform to the naming regulation. In the teaching management database, the preceding string "MR" is used for a factor information table, and "DR" is used for a base data table.

The graphic user interface for creating new tables and attaching constraints is designed as shown in Figure 1. An administrative user can add as many columns to the candidate column list as he wants, and column names can be defined at will.



new basic data table

table name	acdemic awarding
table description	information about acdemic awai
columns: award category ☐ constrain	record_ID Department data_Time

group	
awarding class	select
achievement class	select

id	value
1	national first class
2	national second class
3	national other class

Figure 1. The user interface for dynamic relation creation.

Columns can be removed from the candidate column list before the final confirmation of the operation, which is triggered as the "generate" button is clicked. While adding new columns, the user can attach some existing constraint to the column by selecting the group from the constraint list in the form. Constraint values are listed as the group is selected.

When a new table is created, its corresponding data access interfaces are automatically generated. The column with user defined constraint is generated as a template field in a grid-view component. When a tuple is edited, the template field shows as a drop down list component. The optional values in the drop down list are bound to the values belonging to the specific value group in the metadata table. So, the user cannot input other arbitrary values into the tuple, they can only choose one of those optional values to fill the column. To generate the column list, a list box component in the GUI is traversed, which holds the candidate columns the user wants to build into the new table. Figure 2 shows some of the C# codes for the dynamic data definition process.

```
DataTable dt = new DataTable();
SqlDataAdapter da = new SqlDataAdapter( "select
max(MRID) from MTI ", strConnection);
da.Fill(dt);
string sqlcmd = "create table MR" +
dt.Rows[0][0].ToString()+ "(id int identity primary key";
for (int i = 0; i < ListBox1.Items.Count; i++)
  sqlcmd = sqlcmd + ","+ListBox1.Items[i].Text+"
varchar(50)";
sqlcmd = sqlcmd + ")";
SqlCommand comm = new SqlCommand();
comm.CommandText = sqlcmd ;
comm.ExecuteNonQuery();
```

Figure 2. Key codes for dynamic data definition.

After a table is created, a tuple is inserted into the metadata to record the information of the newly generated table. The value of attribute "MRalias" depends on the user's input. The value of attribute "MRcolumn" is set to be the actual number of columns the user enlisted in the GUI. The value of attribute "MRdate" is automatically filled with current date and time. The value of attribute "MRnote" is inputted by the user for further description of the table. Figure 3 shows some of the key codes.

```
comm.CommandText = "INSERT INTO MTI(MRalias,
MRcolumn, MRdate, MRnote) values(@MRalias,
@MRcolumn, @MRdate, @MRnote)";
comm.Parameters.AddWithValue("@MRalias ",
"MR"+Convert.ToString( MAXID+1));
comm.ExecuteNonQuery();
```

Figure 3. Key codes for metadata update.

To browse or to update tuples in a dynamically create table, users can choose the table's name from a drop down list component. A shared grid view component is designed to show and edit the tuples in the selected table. Figure 4 shows some of the xml formatted codes in the automatically generated data access interfaces that enforce the value constraint on the client side.

```
<asp:TemplateField HeaderText="col3">
<EditItemTemplate>
<asp:DropDownList ID="DDL1" runat="server"
selectedValue='<%# Bind("col3")%>'
DataSourceID="SqlDS1" DataTextField="MCValue"
DataValueField=" MCValue "></asp:DropDownList>
<asp:SqlDataSource ID="SqlDS2" runat="server"
ConnectionString="<%$ ConStr:ConnectionString%>"
SelectCommand="SELECT [MCValue] FROM [MTIII]
where [MAConstraint]=4 union select NULL from
[TblIII]"></asp:SqlDataSource> </EditItemTemplate>
<ItemTemplate><%# Eval("col3") %></ItemTemplate>
</asp:TemplateField>
```

Figure 4. Key codes for dynamic generated data access interface.

Among the codes in the above figure, the union operation of a null value to the available constraint values is indispensible. Because the valid values in a constraint group do not include any null value, which is rather common in most of the newly created tables. Exceptions will be raised if the null value is not taken into consideration when enforcing a constraint.

Compare to our method of dynamic approach, the former design belongs to the static design approach, which involves several pre-defined empty table templates. These templates are taken as raw materials for new table definition. The defect is obvious, that the designer cannot foretell how many columns are there in the newly defined tables. The preserved empty templates are waste of system resources, and they may cause difficulties in database management. So, unlike static database design approaches, our design grants the privileges of relation schema editing to the application user (only one or two administrative users, of course). Thus, dynamic database schema altering can be carried out without any post deployment develop or any coding task.

V. Conclusions

In this paper, we propose a cloud database design scheme with dynamic constraint definition and automatic data access interface generation. By metadata design, we give the application users the ability to create new tables and to define arbitrary constraints on columns of the tables. When users edit tuples in those newly defined tables, our dynamically generated data access interfaces will prevent the users from entering invalid values according to the corresponding constraints. We apply the proposed method in a teaching management database system and solve the problem of dynamic factor adding and managing. Our design not only fulfills the need of the application requirement, but also improves the flexibility and usability of the system. Common short comings of dynamic database design such as poor formalization and management complexity are avoided in the presented database system. The successful deployment and operation of the system demonstrate the correctness and advantage of our design, which facilitates the flexible processing of teaching management information.

References

[1] Ravi Ramdoyal, Jean-Luc Hainaut, Interactively Eliciting Database Constraints and Dependencies, Advanced Information Systems Engineering, Lecture Notes in Computer Science Volume 6741, 2011, pp 184-198

[2] Luca Ferretti, Michele Colajanni, Mirco Marchetti, Supporting Security and Consistency for Cloud Database. Lecture Notes in Computer Science Volume 7672, 2012, pp 179-193

[3] Sergey Savinov,Khuzaima Daudjee.Dynamic database replica provisioning through virtualization.In Proc. of ACM CloudDB '10. New York, NY, USA:ACM,2010: 41-46

[4] Zhao, Fujun,Zhang, Jingfa,Cao, Dai-Yong.Dynamic database connection and dynamic web map service for internet mapping. In Proc of IGARSS '05. New York, NY, USA:IEEE International,2005:3167 – 3169

[5] Swertz, Morris.Towards dynamic database infrastructures for mouse genetics. In Proc of BIBE 2008. New York, NY, USA:IEEE International,2008:1 –7

[6] Ralf Mühlberger.Dynamic database generation for mobile applications. In Proc of ER 2002 Workshops, ECDM, MobIMod, IWCMQ, and eCOMO. Berlin , Germany:Springer Berlin Heidelberg,2003:195-204

[7] Peter Christen, Ross W. Gayler . Adaptive temporal entity resolution on dynamic databases. Advances in Knowledge Discovery and Data Mining. Lecture Notes in Computer Science Volume 7819, 2013, pp 558-569.

[8] Vikram Goyal, S. K. Gupta, Manish Singh, Anand Gupta. Auditing inference based disclosures in dynamic databases. Secure Data Management Lecture Notes in Computer Science Volume 5159, 2008, pp 67-81

[9] Peter Wright, Yih Leong Sun, Terence Harmer, Anthony Keenan, Alan Stewart, Ronald Perrott. A constraints-based resource discovery model for multi-provider cloud environments. Journal of Cloud Computing.June 2012, 1:6,Open Access