ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

IJESRT
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Enhanced Dynamic Schema Binding Using Hashing Algorithm

Ms. Alka[1], Mr. Mahesh Singh[2]
[1]Student, AITM, INDIA
[2]Asst. Professor, Department of Computer Science Engineering, AITM, INDIA
alka2513@gmail.com,

Abstract

The Data Meta Structure employs a database to persist agent and tag information. The tables and fields contain are known as the database schema. The attributes present in these tables are the basis of queries from the users. Hence, there may arise, a need for adding new fields to existing tables or adding new tables to support new functionalities iqueries. These additions to the database are known as a schema update. This paper describes logic that enables dynamic update of the schema through the Dynamic Schema Binding in the Data Meta Structures.

Keywords: DataBase Schema, Message Encryption, Schema Updates, Test plan

Introduction

The Dynamic Schema Binding did not provide any support for migrating from older schema versions to newer ones using dynamic key generation. This meant that users running code expecting a newer schema version than the one present in the database would not be able to exploit the extended functionality that the newer schema was intended to provide(as the database simply did not contain the fields/tables added as part of newer schema).Many users, potentially using varied schema versions in their code, can cause security issues. Initializing databases to update their schema would result in the undesired outcome of all the data being erased and data theft issue is always there. Hence, there is a need for a systematic way of addressing schema mismatches and if required, updating the schema with no loss of existing data and to avoid unauthorized access. Information and status would, therefore, augment to the usability of the application.

The objective of this paper is to design and implement an update logic enabling dynamic schema migration, implementing a frame work that facilitated the addition schema updating code and designing an intuitive and informative interface to the database agent and to insecure the security with the help of dynamic schema generation using security algorithm.

Design

Background

Schema update support is a problem that is commonly encountered in applications that employ data to support business logic. The primary issue faced in such scenarios is the need for supporting newdata isanalysis/descisionswhich,inturn,potentially require new tables/fields to be added to the existing schema. This is especially true for distributed entities connecting to a single database. Resolving schema mismatches under such circumstances is a non-trivial problem. Another challenge faced by architects and developers of such systems is to support the update of schema dynamically (i.e. while the application is running) without causing any loss of existing data and the algorithm mentioned below to generate the dynamic schema models.

In order to address the database vulnerabilities, algorithm based on stream ciper theory has been proposed. A pseudorandom keystream is used to generate ciper text by utilizing the Substitution box values. This encryption algorithm is based on XOR operation and the steps for the algorithm are described below to generate the dynamic schema models.

i. **Calculate Passkey Numeral for Encryption for Schema Structures:**
    - Random number is generated between 1024 and 999999.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

	0	1	2	3
0	(EP1[b0] XOR c0) * c0	(EP1[b1] XOR c1) * c0	(EP1[b2] XOR c2) * c0	(EP1[b3] XOR c3) * c0
1	(EP2[b0] XOR c0) * c1	(EP2[b1] XOR c1) * c1	(EP2[b2] XOR c2) * c1	(EP2[b3] XOR c3) * c1
2	(EP3[b0] XOR c0) * c2	(EP3[b1] XOR c1) * c2	(EP3[b2] XOR c2) * c2	(EP3[b3] XOR c3) * c2
3	(EP4[b0] XOR c0) * c3	(EP4[b1] XOR c1) * c3	(EP4[b2] XOR c2) * c3	(EP4[b3] XOR c3) * c3

- Length of number is calculated.
- Sum of ASCII value of digits of number are calculated.

Thus, Passkey Numeral= Numeral Length + Sum of ASCII value of digits.

ii. Calculate a0, a1, a2 and a3 parameters:

	EP1 parameter	**EP2 parameter**	**EP3 parameter**	**EP4 parameter**
0	a0 XOR a1	EP1 + 15	a2 XOR a3	EP3 + 55
1	a0 XOR a2	EP1 + 25	a1 XOR a3	EP3 + 65
2	a0 XOR a3	EP1 + 35	a1 XOR a2	EP3 + 75
3	a2 XOR a3	EP1 + 45	a1 XOR a3	EP3 + 85

- a1= Sum of digits at odd positions of passkey numeral
- a2= Product of digits of passkey numeral
- a3= (Passkey numeral) mod (256)

iii Calculate b0, b1,b2 and b3 parameters:

In order to compute b0, b1, b2 and b3 values, encryption parameters EP1, EP2, EP3 and EP4 are required which are computed using Table 5.1:

Table 5.1: Encryption Parameters

iv Calculate c0,c1,c2 and c3 parameters:

c0 = ((EP1[b2] XOR EP2[b2]) * a0) + b2 (5)

c1 = ((EP1[b1] XOR EP3[b1]) * a1) + b1 (6)

c2 = ((EP1[b0] XOR EP4[b0]) * a2) + b0 (7)

c3 = ((EP2[b3] XOR EP3[b3]) * a3) +b3 (8)

v Calculate Substitution box (S-box) values using Table 5.2

Table 5.2: S-Box values

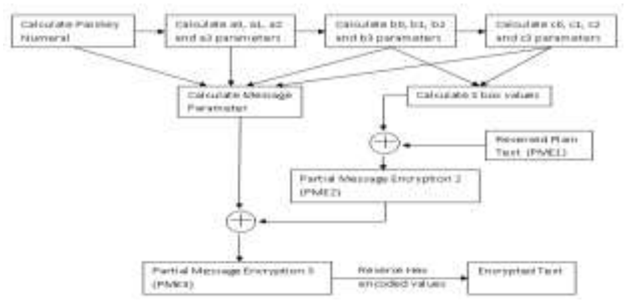
vi Calculate Message parameter:

Message Parameter = Passkey Numeral (obtained in step i.) + Randomly generated key between 1024 and 9999 + Average of a0, a1, a2 and a3 parameters (obtained in step ii.) + Average of b0, b1, b2 and b3 parameters (obtained in step iii.) + Average of c0, c1, c2 and c3 parameters (obtained in step iv.)

vii Message Encryption

- Reverse the plaintext to be encrypted to obtain Partial Message Encryption 1 (PME1).
- Perform PME1 XOR S-box [index] (obtained in step E) operation to obtain Partial Message Encryption 2 (PME2).
- Perform PME2 XOR Message parameter (obtained in step F) operation to compute Partial Message Encryption 3 (PME3).
- Reverse hex encoded value of PME3 to compute Partial Message Encryption 4 (PME4) which is the resultant encrypted text.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852


Calculate Parsley Numeral
Calculate a0, a1, a2 and a3 parameters
Calculate b0, b1, b2 and b3 parameters
Calculate c0, c1, c2 and c3 parameters
Calculate Message Parameter
Calculate S box values
Reserved Plain Text (PME1)
Partial Message Encryption 2 (PME2)
Partial Message Encryption 3 (PME3)
Reverse Hex encoded values
Encrypted Text

Requirement

Display more information pertaining to schema version, server and port running the databases and the status of the connection.Design now messages and the logic to determine database status based on schema mismatches.Design and the implement logic to perform schema update.

High level Design

In order to detect schema mismatches and keep track of the general working of the database a status attribute was included in the Dynamic Schema Binding. To retrieve this information from the Dynamic Schema Binding, two new messages were incorporated into the application. The database panel would query the Dynamic Schema Binding for the status of the database by sending a DBStatusRequest message. On receiving this message, the Dynamic Schema Binding would check for any schema mismatches or other anomalous condition in the database. The Dynamic Schema Binding would then compose a DBStatusUpdate message reflecting the database status and the same to the database panel. Status thus acquired would be displayed to the user by the panel

The version of schema is persisted in the database within the TagCentricAppInfo table. We will refer to this as the database schema version. The schema version is also hard-coded into the code as a Dynamic Schema Binding attribute. This is referred to as the code schema version and indicates the format of schema that the code expects. Whenever a Dynamic Schema Binding connects to a database, the database schema version can be read and compare with the code schema version of that Dynamic Schema Binding. A mismatch is detected if these two schema version are not equal.

Whenever a schema update is issued by the user, the database panel should forward the command to the Dynamic Schema Binding. The update command is embedded into the DBInitialize message. A flag in the DBInitialize is set to indicate to the Dynamic Schema Binding that an update and not an initialize was required. Once the update logic is executed, the Dynamic Schema Binding multicasts a DBStatusUpdate message to all the database panels to indicate the new schema version and status.

Detailed Design

Design of update architecture

The update logic architecture was designed based on the schema version detected from the database. The crucial part of the design was to decide the course of action for different scenarios of schema mismatch. The following design was adopted to deal with the various scenarios.

If no version was read from the database, it would be a sign that the database was not populated with the middleware schema. Hence under such a scenario the user is requested to initialize the database to set up the necessary tables and the data for the application to be operational.

If the database schema version is found to be higher than the one present in the code, the user is allowed to access the database and operate in a normal fashion. The reason for implementing such a strategy is to provide backward compatibility for users employing code with old schemas. The user is, however, informed that s/he processes old code. If, on the other hand, the code schema version is detected to be higher or newer than the one read from the database, the user is forced to either update the schema or initialize the resulting in newer schema being populated in the database. Such an action is a necessary as the some queries may need the fields/tables belonging to the new schema in order to execute.

To communicate the update command issued by the user or the version, database information and database status, separate messages were added in to the application. Whenever a database agent is launched or alternately whenever a database panel connected to an already launched agent,the panel would send a DBStatusRequest to the Dynamic Schema Binding. The Dynamic Schema Binding, on receiving such a request would compose a DBStatusUpdate message to the concerned panel. This update message is designed (the design of the messages is covered later) to contain all the relevant

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

information which is then displayed to the user by database panel.

In case a user needs an update and issues the same, the panel sends a DBInitialize message to the Dynamic Schema Binding. This message is designed to indicate the nature of the command issued by the user (initialize or update). On receiving the DBInitialize the Dynamic Schema Binding ascertains the course of action to be taken. Once the update is completed, the agent composes a DBStatusUpdate message to indicate the latest attributes of the database. This time, however, the Dynamic Schema Binding publishes or multi-casts this message to all the existing panels so as to maintain a unified and coherent view of the database for all the users.

Design of update logic

The user in made invoke the update logic the current code schema version is found to be higher than that of the database schema version. The update logic is designed to update to accept a schema version number. This number is the database schema version. Since the code schema version is manually updated by the developers whenever additions to the schema are made, the code is cognizant of the latest version that it expects in the database. Hence , when the old schema number detected in the database is passed to the update logic it updates the schema in cycles till the current version of schema resides in the database.

The updating of schema is done one step at a time. In each step the current schema is converted to the next higher one. Therefore , the number of such steps or cycles executed depends on the difference in schema version in database and code. For instance if the code schema version was 4 and the database schema version was 1,three steps would be required to update the schema to version 4. Hence, the update logic works for any version of schema no matter how old it is.

Design of user interface

The user Interface, incorporated into the database administration panel was designed to be more informative to the user. Separate object were used to indicate the various attributes of the database. The user interface is designed to show the version number(the database version number is to be specific), the JDBC connection URL (to indicate the type, machine and the port of the database) and the status of the database connection. The status field in particular was designed to display information that indicate problems, if any and the next step to be taken by the user. The interface was also designed to disable certain operations under certain status to avoid inconsistent results. For instance, a user is not allowed to query the database whenever an update is deemed necessary etc.

Design of messages

The DBStatusRequest was designed to simplify query for the database status. The DbStatusUpdate was designed to contain attributes necessary to display information on the panel. The attributes incorporated into the message were version, connection URL and status. Six different status and their scenarios were also included in the DBStatusUpdate. The message also contained the necessary mechanisms to read or write these attributes.

To indicate the operation requested by the user, the DBInitialize message had an additional flag added to it. The flag is set true if the user requests an update. On receiving this message, the Dynamic Schema Binding can access the flag to ascertain if an update operation is required. If not the Dynamic Schema Binding proceeds to initialize the database.

Implementation

The update logic is implemented as a block of switch case statements. The switch statement takes the database schema version as argument. The case statements start with 1 through to the least version. The case statements lack the break command and hence, the flow of execution flows through till the last case statements irrespective of where they start. Hence, the database is updated to the latest version once the update logic is finished executing. Each case statement makes a call to it's respective convert schema method. The convert schema methods follow a common nomenclature of convertSchemaxtoy where x is the older version and y is the next higher schema version. For example, convertSchema2to3 converts the database from schema version 2 to 3. In addition to adding to the code for schema update in the convert schema fucntions, the same is the incorporated into the initialization method as well. After each convert schema statements, the case block also updates the schema version number in the TagCentricApplInfo to reflect the update in schema.

The user interface in database panel in implemented using a Box Layout. Each attribute and it's value are implemented as separate panels and

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

added to the entire tab in the database panel. The initialize, terminate and update buttons are added to a separate component which in turn is added to the panel. The tablesathat display the value for the versions, connection and status attributes are dynamically updated as and when the panel receives a DBStatusUpdate message from the Dynamic Schema Binding. The user interface is also programmed to control the enabling/disabling of the various buttons based on the status of the database received.

The messages DBStatusrequest and DBStatusUpdate were implemented as separate classes just like the other messages in the edu.uark.rfid.messages package. The DbStatusRequset is a plain message class with a unique multicast address. The DBStatusRequest has public static attributes that helps in indicating the status of the database. The status itself is an integer variable that varies from 0 to 6 depending on the actual status. The Dynamic Schema Binding and the database panel then access these public members to determine and set the status. The DBStatusUpdate also contains an integer version number and a connection attribute which is an object of the type String. The message also has accessors and modifiers to read and set the values for these attributes.

The DBInitialize message had an update flag added to it. The flag in of the Boolean type and can take true or false as it's value. If an update is issued by the user, this flag is set to true through the modifiers implemented into the message. If an initialize is issued, the flag retains it's default false value. The flag is then accessed by the Dynamic 3chema Binding. If a true is read, the Dynamic Schema Binding invokes the update logic, else it invokes it's initialize method.

Test Plan

Unit tests were conducted using AlkaSQLowncustomized database. The new fields tables added as part of the update were checked manually through the database administration tool (in case of postresql). Also ,data present in the database prior to the update were checked for after execution of the update to ensure that no data was erased. All the different scenarios leading to the different status of the data base were tested exhaustively. Care was takento cover the entire functionality (as well as the usability) of this module of code.

Results and discussion

The unit Test were successful and the outcomes of all the tests were as expected. No data was found to be lost after the update operation and the database was in a consistent state irrespective of the operation performed. All the different scenarios resulting from the varied database status worked as expected. The user interface was found to informative and coherent. Interoperability of multiple instances of the application did not produce inconsistent results.

Conclusion

We are applying security on tables not on the whole database. If we apply security on database then every time, encrypt and decrypt schema then efficiency decreases.

References

1. *Elmasri, R. and Navathe, S. – Fundamentals of Database Systems – 2nd Edition. Benjamin Cummings Publishing Company Inc., Redwood City, CA (1994), 25-26.*
2. *Gibbs, S. J. and Tsichritzis, D. - "A Data Modeling Approach for Office Information Systems", ACM Transactions on Office Information Systems, vol.1, no. 4, (1983), 299-319.*
3. *Hull, R. and King, R. - "Semantic Database Modeling: Survey, Application and Research Issues", ACM Computing Surveys, vol. 19, no. 3, (September 1997), 105-133.*
4. *King, R. and McLeod, D. - "A Database Design Methodology and Tool for Information Systems", ACM Trans. on Office Information Systems, vol.3, no. 1, 1985.*
5. *Lerner, B. S. and Habermann, A. N. - "Beyond Schema Evolution to Database Reorganization", Proceedings of the ECOOP, pp. 67-76, Oct. 21-25, 1990*
6. *Manber, U. - "Introduction to Algorithms - A Creative Approach", Addison Wesley Pubs. Reading, Mass., 1989.*