ISSN:0975-9646

An Efficient Reuse of Legacy C-ISAM through Java Native Interface

Raphael O. Anumba

CTT Research Lab
29626 Teasedale Place, Castaic, CA 91384,USA

Abstract— **The access methods to C-ISAM database table was explored and implemented for use by any Java application. Here, we presented some of the stages through which data can be retrieved and modified via Java Native Interface (JNI), using a series of instance methods to encapsulate the raw manipulation of the inherent binary record of C-ISAM database records. We particularly addressed the issue of binary data access and conversion, using a runtime field table routine. The paper went ahead to show the use of the static schema class approach as an alternative.**

Keywords— **C-ISAM, C, Java, JNI, Database**

I. INTRODUCTION

C-ISAM[8] is well known for its simplicity and its inherent design to bypasses the overhead of a relational database management system thereby provides fast, efficient access to database records.

- Provides quick data retrieval using B+ tree-index architecture.
- Supports multiple languages through global language support (GLS).
- Offers flexible indexing options that let you build indexes on multiple fields, a single field or parts of a field.
- Includes efficient mechanisms for preserving data integrity.[7]

To speed up the development of new application in Java programming language, there is an eminent need to use the C-ISAM legacy database tables. Here we presented an in-depth approach of safely accessing C-ISAM using a JNI[1] to access the complex binary records. JNISAM, a collection of Java access classes made it possible to modularize and segregate the various operations ranging from configuration to record field access methods.

Not only that the use of JNISAM gave insight and easy way to handle C-ISAM records, it also shaded light to an endless possibility of the use of Indexed Sequential files in an unorthodox ways to achieve speed and reliability. It may also serve as an extended hash table for very large and multiple data sets.

The benefits of JNISAM software development approach includes the portability of Java and JNI which means that JNISAM will ported to any C-ISAM platform, and any internal change in JNISAM is transparent to the recompiled JNI implementation.

II. STRUCTURE OF C-ISAM

A. Record layout

To describe the structure of a C-ISAM record we assume and employee record *emprec* with 115 record length. If the record was made up of fields *p_empno, p_empname, p_SSN, p_age* and *p_allowance*; and filed width of *4, 20, 9, 2* and *8* respectively, then the C language equivalent record definition follows:

```
char emprec[115];
char *p_empno       = emprec+ 0;
char *p_empname     = emprec+ 4;
char *p_SSN         = emprec+24;
char *p_age         = emprec+33;
char *p_allowances  = emprec+35;
```

B. Abstract representation

A record R in a file is of 4-tuple $R(p, t, w, r)$, where p is the pointer to a filed in R and t is the type of the field, w is width or size of each field and r is the number of times a filed can repeat such that for $r=4$, there exists pointes p_0, p_1, p_2, and p_3.

Each field can be of type $t=\{l, s, c, f\}$ where l is of type *long* of 4 bytes, s is short of 2 bytes, c is character of 1 byte and d is double of 8 bytes.

TABLE I
RECORD DEFINITION

i	p	t	w	r	Θ
0	emp_no	l	4	1	0
1	emp_name	c	20	1	4
2	SSN	c	9	1	24
3	age	s	2	1	33
4	allowances	d	8	10	35
				Record length L	**115**

A transformation of R can be made to find the offset Θ of p such that Θ_i of p_i will be

$$\theta_i = \begin{cases} 0 \quad for\ i = 0, \\ w_{i-1} \cdot r_{i-1} + \theta_{i-1}\ for\ i > 0 \end{cases}$$

The length L of a record R will be

$$L = \sum_{i=0}^{n} w_i \cdot r_i = w_n \cdot r_n + \theta_n$$

TABLE I can be seen as a two dimensional matrix:

$$[p_i, t_i, w_i, r_i]_{i=0}^{n},$$

such that for i=2, the field definition will be (SSN,c,9,1);

III. JNISAM STRUCTURE

Without modifying the underlying table operations of the exiting C-ISAM, JNISAM provided many functions that mostly perform one-to-one operations and in some cases provided a JNISAM function to encapsulate multiple C-ISAM operations

A. Configuration functions.

Configuration functions provide means for the creation of and dropping of C-ISAM table. The two main class functions can respectively be represented as

$$JNISAM \cdot create(fname, [p_i, t_i, w_i, r_i]_{i=0}^{n}) \text{ and}$$
$$JNISAM \cdot drop(fname).$$

Where *fname* is the name of the table to be created or dropped.

B. Table access functions.

C-ISAM table can be accessed by calling a JNISAM *open* function and deaccessed by calling a JNISAM *close* function. To open a table we used the JNISAM class method as:

$$F \leftarrow JNISAM \cdot open(fname)$$

This will open a table with name *fname* and return an object *F* that will be used to perform all file related operations. The *close* function is a table specific operation, hence:

$$F \cdot close();$$

C. Record access functions.

Access to a C-ISAM record *R* can be gained by a *find* or *create* method. While *create* method returns a record with all initialized field, the *find* method returns the first record that met some query *criteria*.

$$R \leftarrow F \cdot create()$$
$$R \leftarrow F \cdot find([criteria])$$

With a newly created record *R* of file *F,* the *insert* method will add *R*, after necessary field modification, to the C-ISAM table. The fields of a retrieved record, *R* of file *F,* may be modified and written back to the C-ISAM table using the *update* method.

$$F \cdot insert(R)$$
$$F \cdot update(R)$$

A record can be removed from the C-ISAM file with the *delete* method if the record *R* met the query *criteria.*

$$R \leftarrow F \cdot delete([criteria])$$

D. Field access functions.

With an instance of a record *R* the field value *V* of type *String, int, short or double,* can be retrieved using *getString, getInt, getShort or getDouble* method respectively.

$$V_t \leftarrow R \cdot get_{<t>}(p) \Longrightarrow \begin{Bmatrix} f \leftarrow getField(p) \\ \theta \leftarrow f.getOffset() \\ w \leftarrow f.getWidth() \\ t \leftarrow f.getType() \\ \textbf{return}\ b.read_{<t>}(\theta, w) \end{Bmatrix}$$

Where *f* is a field object in *R* accessible by name *p,* and *b* is also an object in *R* holding the byte array of record from C-ISAM record.

To update fields of a newly created or query retrieved record, we used a *set* method that can take the name of a field *p* and a value *V* of any type *String, int, short or double*. We hence abstract the operations as:

$$R \cdot set(p, V) \Longrightarrow \begin{Bmatrix} f \leftarrow getField(p) \\ \theta \leftarrow f.getOffset() \\ w \leftarrow f.getWidth() \\ t \leftarrow f.getType() \\ b.write(conv(V, t), \theta, w) \end{Bmatrix}$$

Just like the *set* method above, the values *f, Θ, w* and *t* are retrieved in the same manner, but we used a *conv* method to convert the value *V* to the field type *t* before writing to the underlying byte array.

An existing record in some situation may require the reset of the entire field before the *update* method; we achieved this by implementing a *clear* method on the record object *R*.

$$R \cdot clear()$$

IV. CONCLUSION

JNISAM has been efficiently implemented to use the capabilities of Java Native Interface (JNI) to access the low level C-ISAM byte records by querying, retrieving and extracting fields to be used directly in Java applications.

The underlying C-ISAM field and methods were not change in any form, rather a Java Native Interface were implemented to wrap one or more C-ISAM member to achieve a tread safe methods.

Apart from the normal access to native C-ISAM, other advantages of the use JNISAM includes an instant sorting of very large files, the use as external or extended Hash table and can be used educationally as an experimental key-value database.

REFERENCES

[1] Sheng Liang, *Java Native Interface: Programmer's Guide and Reference*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.

[2] *Java native interface specification (release 1.1),* Sun Microsystems. Jan. 1997.

[3] G. Tan, A. W. Appel, S. Chakradhar, A. Raghunathan, S. Ravi, and D. Wang. "Safe Java native interface," In Proc. 2006 IEEE International Symposium on Secure Software Engineering, pp. 97--106, Mar. 2006.

[4] C. J. Date. *An introduction to database systems: vol. 1 (5th ed.)* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA 1990

[5] C. J. Date. *An Introduction to Database Systems - Volume II.* Addison-Wesley Publishing Company, 1985.

[6] Rob Gordon. *Essential JNI: Java Native Interface*, Prentice Hall PTR. 1998

[7] *http://www-03.ibm.com/software/products/en/imformixcisam/*

[8] *Informix C-ISAM Index Sequential Access Method - Programmer's Manual (File Manager Version 5.0)* Informix 1991