Vandana Publications

Volume-4, Issue-1, February-2014, ISSN No.: 2250-0758

International Journal of Engineering and Management Research

Available at: www.ijemr.net

Page Number: 81-89

Key and Value Paired Data using Java Hash Table

Ankur Saxena[1], Ankur Chaurasia[2]

[1,2]Amity Institute of Biotechnology, Amity University, Noida, Uttar Pradesh, INDIA

ABSTRACT

This paper proposes use of a Java HashTable mechanism for Online Shopping Cart Application over the Web Application. A hash table is a data structure in java that can offer rapid storage and retrieval of data. Hash tables are extensively used to implement tables that associate a set of key and values, as they provide O(1), to perform operations such as query, insert and delete operations. A leading implementation for string keys is the cache conscious array hash table. However, at moderate collisions are quite frequent which not only increases the access time, but also decrease the performance in the deterministic. Due to this deterministic performance, the hash table degrades. In some systems, it is very difficult to keep the hash operations more deterministic. In recent trends, more research papers have been proposed, which employs a new and fast hash functions to implement hash tables and to avoid collisions.

Keywords: *Hashing, HashTable, Data Structure, Synchronization, Tomcat, Shopping cart.*

I. INTRODUCTION

A Data Structure is an arrangement of data in a computer's memory or even disk storage. It has a difficult way of storing and organizing data in a computer so that it can be used efficiently. Different kind of Data structure is used in different applications. Usually, efficient data structures are a key to designing efficient algorithms. [1]

Hashing is used to build, search or delete data from the table. Hashing is so commonly used in computing that one might expect hash functions to be well understood, and that choosing a suitable function should not be difficult. [2] The basic idea behind hashing is to fetch out a column or field in a record, which we can call as a key, and convert it through some algorithm to a numeric value, known as the hash key. Hash key shows the position to either store or retrieve an item from the table. For many if not most applications of computer science, performance is important. Sometimes, even small performance gains can save a lot of time and memory. Improving performance can be done at different levels. [3] This numeric value may vary in between range of 0 to n-1, where n is the maximum number of slots in the table. The algorithm to convert a key to a hash key is commonly known as a hash function. This can be used whenever access to the table is needed. Hashing is a popular technique because the expected retrieval time is effectively a constant. Unfortunately, hashing is often avoided in real-time applications because the worst-case retrieval time is proportional to *n*, implying that some retrieval operations may be unacceptably slow. For these applications, perfect hashing schemes are desirable. Such schemes have, to date, been suitable only for small, constant tables. This is because they require either tremendous computational effort to construct the table, or require much more storage than that required to actually hold the elements in the table. [4]

For example:

One common method of determining a hash key is the division method of hashing. The formula that will be used is:

Hash key = key % number of slots in the table

Considering that the given table is having 8 slots:

Hash Key=key % table size

0 = 64 % 8
4 = 12 % 8
2 = 34 % 8
1 = 41 % 8
6 = 30 % 8

[0]	64
[1]	12
[2]	34
[3]	
[4]	
[5]	41
[6]	
[7]	30

The division method is mostly a reasonable strategy, providing the key happens to have some undesirable properties. For example: If the table size is 10 and all of the key values are ending with zero. Hence the hash key of each key value will fall in 0. In this case, choosing the hash function and the table size should be done carefully. The best table sizes are prime numbers.

One problem though is that keys are not always numeric. In fact, it's common for them to be strings.

One possible solution: add up the ASCII values of the characters in the string to get a numeric value and then perform the division method.

```
int hashValue = 0;

for( int j = 0; j < stringKey.length(); j++ )
hashValue += stringKey[j];

int hashKey = hashValue % tableSize;
```

Hashing is a method for storing and retrieving data from a database. It is used to insert, delete, and search for records based on a search key value. To implement the hash table, these operations need to have constant time. In fact, a good hash system typically shows at only one or two records for each search, insert, or delete operation.

Hashing Techniques

- Hashing provides very fast access to records on certain search conditions.
- The search condition key on a single field, called the hash field.
- The main aim behind hashing is to provide a function 'h' called a hash function (or) randomizing function, that is applied to the hash field value of a record and yields the address of the disk block in which the record is stored. [5]

The shopping cart is similar to the original grocery store shopping cart. Basically, it is a means by which a customer can read a list of products and mark off the selections they want. When finished ordering, the customer indicates that they are ready to checkout. This is where the total order is placed and confirmed. Also, the customer will enter their shipping information at the checkout. Shopping cart software is the programming that allows a website to build a catalog of products and its database and integrate it into the website pages. The shopping cart is one of the most important parts to having a smooth eCommerce transition.

The features of this Application are divided into three parts.

A. Visitor Features

- Browse all the product of Shopping Cart
- View Product Details like price, quantity and features.
- Become a Member through Registration process

B. Registered Member Panel

- Login to Shopping Cart.
- Manage Account
- My Profile
- My multiple Orders
- Purchase Product
- Logout

C. Admin Panel

- Login to administrator
- Administrator User Management
- Customer Management
- Product Management
- Price Chart Management
- Order Manager
- Shipping Management

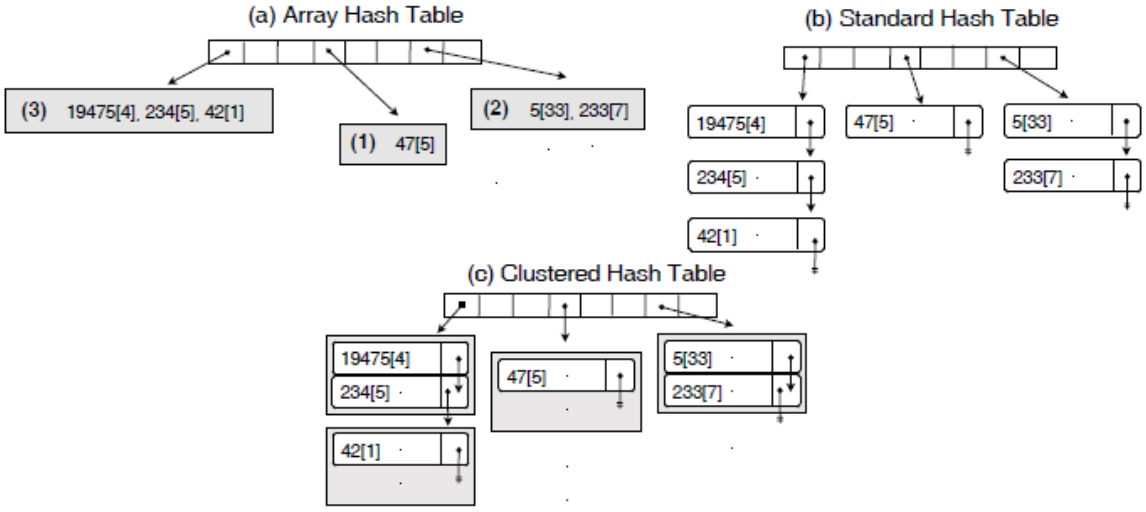


Figure 1: *Some integers (keys) along with their payload data (square brackets) are inserted into an array hash, a standard-chain hash, and a clustered-chain hash table. Each dynamic array starts with the no. of keys (in parenthesis).*

Purpose		Software Used
Front end tool	:	Java
Back end tool	:	My SQL 5.x
Operating system	:	Linux
Web Server	:	Apache tomcat.

Browser Compatibility: Internet Explorer 7.0, 8.0, Firefox 3.x, Google Chrome, Safari (Windows)

II. BACKGROUND

When distinct inputs hash to the same value a 'collision' is said to occur. When the hash value is used as a key (for example, as an index into a table) some method needs to be adopted for dealing with collisions. In what follows it is assumed that the technique known as 'separate

chaining' is employed, and that there is effectively infinite memory available for storing the chains.

The algorithm which led to the present investigation was:

```
Algorithm ACK:
m1 = 171
mi = BITS(77.mi-1 + 153, 8) for 2 £ i £ 16
h0 = 0
hi = hi-1 + XOR(ci, mi) for 1 £ i £ 16
hi = hi-1 for i > 16
H = BITS(hn, 8)
```

The mask array elements mi in this algorithm is stored as 'signed bytes'.

The process used to generate these pseudo-random numbers is a linear congruential generator. The routine ACK was used to hash 36 376 identifiers collected from a large number of programs written in C. As a check that there was not some curious property of C identifiers which did not hold generally, it was also used to hash 24473 words from a Unix dictionary. It had been expected that this distribution would be more or less flat, and its marked U-shape was surprising. [2]

The array hash table can be easily adapted to store, delete, and retrieve fixed-length keys, namely 32-bit or 64-bit integers. In this paper, we only consider 32-bit integers but the algorithms we describe below are readily adaptable to 64-bit integers.

To search for an integer (a key) in an array hash table, we first hash the key to acquire a slot. If the slot is empty, then the key does not exist in the hash table and the search fails. Otherwise, we scan the array acquired, a key at a time, until a match is found or until we exhaust the array—in which case, the search fails. On successful search, the key and its payload data are not moved to the front of the array due to the high computational costs involved (Askitis 2007); as we show in later experiments, cache-efficiency more than compensates.

When a search fails, we can then insert the key into the hash table. Insertion proceeds as follows: if the slot was empty, a new 12 byte array is allocated and assigned. The first 4 bytes in the array are reserved to store the number of entries (which is initialized to 1). The next 8 bytes store the key (4 bytes) followed by its payload data, a 4 byte integer. Otherwise, we resize the existing array by 8 bytes (4 bytes for the key and 4 bytes for its payload data) using the *realloc* system call. The key and its payload are then appended to the array and the number of entries in the array is incremented by 1, completing the insertion process.

Deletion proceeds in a similar manner: the key is hashed, the acquired array (if any) is searched and assuming the key is found, we overwrite the key and its payload data by sliding the remaining keys (if any) with their payload data one entry towards the start of the array. The number of entries in the array is then decremented by 1. The array can then be resized using the *realloc* system call to shrink the array by 8 bytes.

Figure 1 shows an example of an array hash. As shown in Figure 1, we interleave keys with their payload data. Alternatively, we can separate keys from their payload by storing the keys first followed by their payload data (in order of occurrence). However, preliminary experiments using this array configuration yielded no significant gains in performance. [6]

III. IMPLEMENTATION

HashTable: HashTable is a part of java utilities and is a concrete implementation of a Dictionary. Hashtable is now integrated into the collections framework and it stores key-value pairs and it is synchronized.

When using a HashTable, we specify an object that is used as a key, and the value that you want linked to that key. Objects with non-null value can be used as a key or value. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

The HashTable constructors are shown here:

Hashtable( ): This Constructor create default constructor of HashTable

Hashtable(int size): This form of constructor creates a hash table that has an initial size specified by size

Hashtable(int size, float fillRatio): This form of constructor creates a hash table that has an initial size specified by size and a fill ratio specified by fillRatio.

This ratio must be between 0.0 and 1.0, and it determines how full the hash table can be before it is resized upward

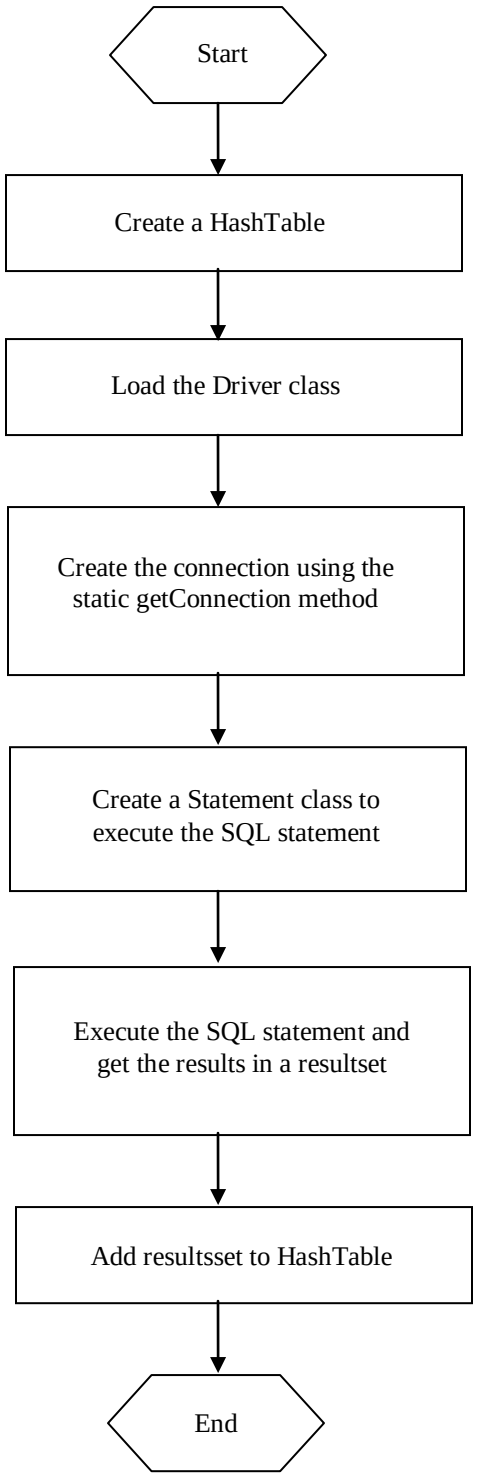
The capacity of the hash table is set to twice the number of elements in m. The default load factor of 0.75 is used.

Methods of hash table are given below:

Methods	Description
clear()	This method clears the hash.
clone()	This method creates a clone of the hashtable
contains(Object)	This method returns true if the specified object is an element of the hashtable.
elements()	This method returns an enumeration of the elements.
keys()	This method returns an enumeration of the hashtable's keys.
put(Object, Object)	This method puts the specified element into the hashtable, using the specified key.
remove(Object)	This method removes the element corresponding to the key.
size()	This method returns the number of elements contained in the hashtable.
toString()	This method converts to a rather lengthy String.

Table 1: Methods of Java HashTable

A. Flowchart:



B. Shopping Cart Architecture

Shopping Cart is given in the Figure 2. Tomcat Server was used for this study. This server is configured by URL: http:\\localhost:8080. Tomcat is an open source servlet developed by the Apache Software Foundation (ASF). [7] Tomcat implements the Java

Servlet and the Java Server Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run. [8]

Business concept model is a mind map that relates the terms generated in the business process like no of product, product name and other important terms.

C. Method Of Hash Table Data Structure:

Storing key and value pair using HashTable

```
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.Map;

public class HashTableDemo3
{
        public static void main(String[] args)
        {
        Hashtable<Integer,String>
        book=new Hashtable<Integer,String>();
        hTable.put(new Integer(1), "Java");
        hTable.put(new Integer(2), "J2ee");
        hTable.put(new Integer(3), "Jsp");
        hTable.put(new Integer(4), "C++");
        hTable.put(new Integer(5), "Servlet");

        Set s =hTable.entrySet();
        Iterator i=s.iterator();

        while(i.hasNext())
        {
        Map.Entry m=(Map.Entry)i.next();
        int key = (Integer)m.getKey();
        String value=(String)m.getValue();
        System.out.println("Key :"+key+"    value
        :"+value);
        }
        }
}
```

Business concept model is a mind map that relates the terms generated in the business process like number of product, product name and other important terms. (Figure 3)

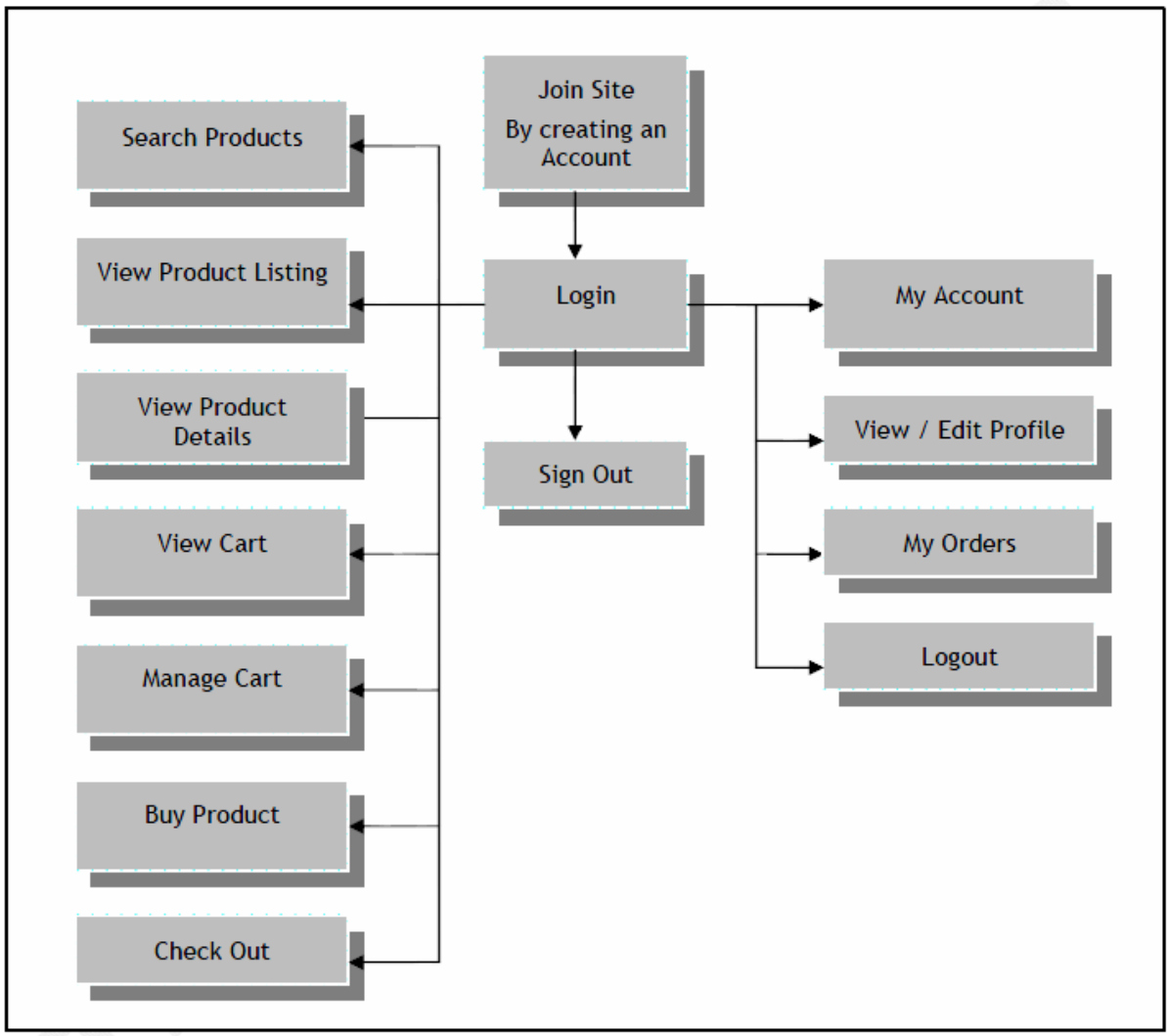


Figure 2: Architecture of the Shopping cart application

Get HashTable keys from values

```
class Ankur
{
ResultSet rs;
Hashtable hash;
Vector v;
static int i;
Enumeration e;

public static void main(String args[] )
{
keyhash gs = new keyhash();
System.out.println("Hello Ankur");
gs.getresultset();
}
```

```
public static void main(String args[] )
{
keyhash gs = new keyhash();
System.out.println("Hello Ankur");
gs.getresultset();
}

public void getresultset()
{
Hashtable hash = new Hashtable();
Vector v = new Vector();
boolean conn = dbConnect();

// Establish database connection here
```

```
        if(conn == true)
        {
                try
                {
                PreparedStatement getArticles =
                db.con.prepareStatement
                 ("select * from book where id = ?"
                  // sample query)
                 String artid;
                 getArticles.setInt(1, 4);
                rs = getArticles.executeQuery();

                while (rs.next() == true)
                {
                key = rs.getString(1);
                value = rs.getString(2);
                hash.put(key,value);
                //Hashtable populated
                }
                fillvector(hash,v);
                getkeys(v);
                rs.close();
                }
                catch(Exception ex)
                {
                 ex.printStackTrace();
                 }
        }
}

public void fillvector(Hashtable hash,Vector v)
{
int j=0;
boolean success;
Enumeration e = hash.keys();

        while(e.hasMoreElements())
        {
        String key = (String)(e.nextElement());
        String value = (String)hash.get(key);

        if(value.matches("????"))
        //Put the value here to retrieve the
        // corresponding keys
        {
        v.addElement(key);
        //Add the corresponding keys to the vector
        }
        }
}

public void getkeys(Vector v)
{
        Enumeration ev= v.elements();
        while (ev.hasMoreElements())
        {
        System.out.println(ev.nextElement());
        // Print the keys related to a single value
        }
}
```

Storing into database using Hash Table:

```
Hashtable<Integer,String>
book=new Hashtable<Integer,String>();
try
{
PreparedStatement ps = dbConn.prepareStatement
("INSERT INTO book (BLOBcolumn) VALUES
(?)");
        ps.setObject(1, hrm, Types.OTHER);
        ps.executeUpdate();
        ps.close();
}
catch (Exception e)
{
System.out.println("ERROR storing HashTable into
db: " + e.toString());
}
```

HashTable with JDBC

```
Hashtable<Integer,String>
book=new Hashtable<Integer,String>();
//create a HashTable
try
{
// Load the Driver class
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
//Create the connection using the static getConnection
method
Connection
connection=DriverManager.getConnection
("jdbc:odbc:cart");
 //Create a Statement class to execute the SQL
statement
statement = connection.createStatement();
String emp = "select * from book";
//Execute the SQL statement and get the results in a
Resultset
ResultSet resultset = statement.executeQuery(book);

while (resultset.next())
hrm.add(resultset.getString(1));
//Add resultsset  to HashTable
}
catch (Exception e)
{
}
```

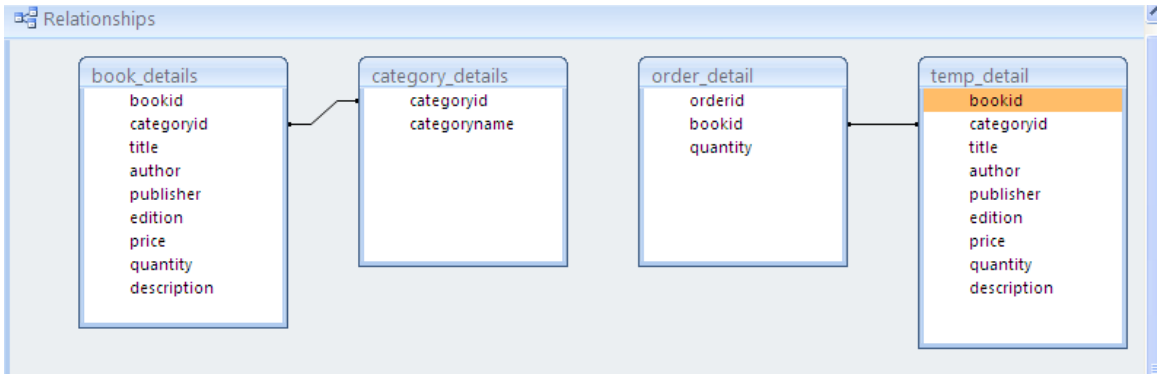


Figure 3: Business Type Model

IV RESULTS

This application can run on any server. For this study Tomcat server was used. Figure 4 shows the user interface after login

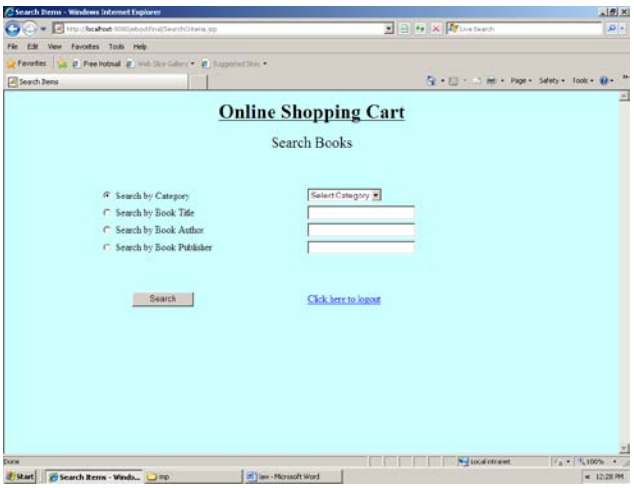


Figure 4: User Interface after login

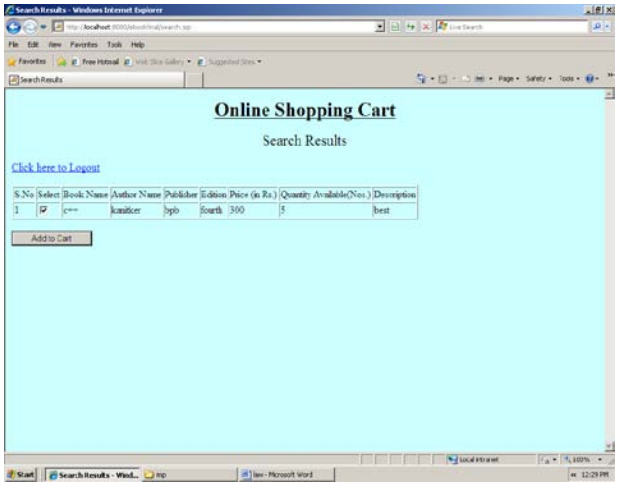


Figure 5: Search result of product

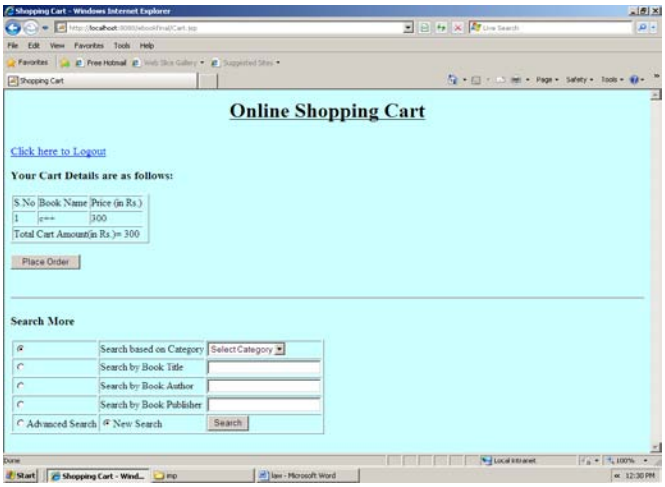


Figure 6: Excerpt from the Database

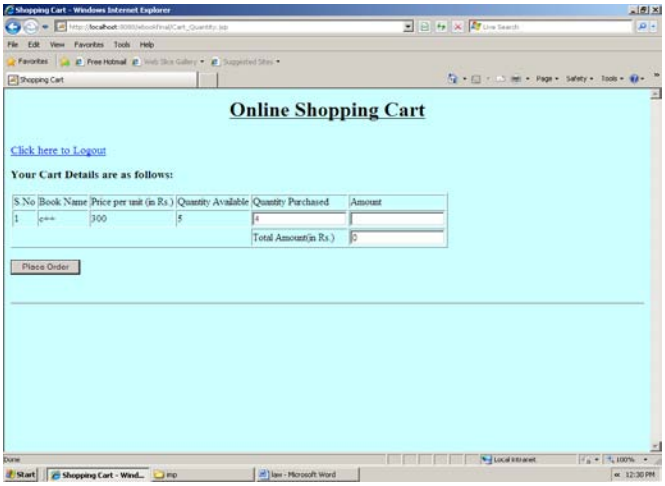


Figure 7: Place Order after shopping

V. CONCLUSION

We hope that Shopping Cart using java HashTable will become a successful interface for business owners seeking to open up small, online products. We see that it has the potential to benefit a whole spectrum of business owners, from those technically experienced, including developers, to those with little or no technical background. Its stability, customizability, and user-friendly interface make it an ideal choice for facilitating online transactions and creating a satisfying customer experience.

REFERENCES

[1] Handling Of Synchronized Data Using JAVA/J2EE. Ankur Saxena 177-194(2011) International Journal of Management, IT & Engineering

[2] Selecting a Hashing Algorithm B. J. MCKENZIE, R. HARRIES AND T. BELL *Department of Computer Science, University of Canterbury, Christchurch, New Zealand*

[3] Analysing and Improving Hash Table Performance *Using usage analysis to improve performance for cache* Tom van Dijk t.vandijk@student.utwente.nl

[4] Practical Perfect Hashing *G.V. Cormack1, R.N.S. Horspool2, M. Kaiserswerth3* School of Computer Science McGill University

[5] Hashsort - A Novel Hashing Function Based On Sorting Technique To Resolute Collision S. Muthusundari [1], R.M. Suresh [2] [1]research Scholar, Sathyabama University,Chennai,India[2] Principal, Jerusalem Engineering College, Chennai, India E-mial: nellailath@yahoo.co.in , rmsuresh@hotmail.com

[6] Fast and Compact Hash Tables for Integer Keys Nikolas Askitis School of Computer Science and Information Technology, RMIT University, Melbourne 3001, Australia. Email: naskitis@cs.rmit.edu.au

[7] Securing Confidential Data using Java/J2EE Ankur Saxena, Ruchi Jakhmola (2011) International Journal of Science Technology and Management

[8] Java Server Pages by Ankur Saxena MJP Publication.