Analysing Schemaless Design in Marketplace Application Using HBase

Ade Hodijah[1,*] Urip Teguh Setijohatmo[2]

[1,2]*School School of Information Engineering, Politeknik Negeri Bandung, West Java40559, Indonesia*
[*] *Corresponding author. Email: adehodijah@jtk.polban.ac.id*

ABSTRACT
A marketplace in this research is an online application for buying and selling products like e-commerce, where Seller is able to have more than one store. A seller in a marketplace application may create a store with a variant description based on the products for sale. HBase is One of NoSQL schemaless database, that means the store represented by row can have a different number of a field which makes complexity when it designed in the relational database. This study explores how to design a schemaless in the marketplace by implementing a test case scenario to examine a prototyping application is suitable with unstructured data of the store.

Keywords: *Marketplace application, HBase, Unstructured data, Schemaless*

1. INTRODUCTION

HBase is a column-oriented database built on top of the Hadoop Distributed File System or HDFS [1]. Schemaless in HBase means columns schema is not fixed, which defines only Column Families [2], while the number of columns in a table of a relational database is grouped in the same structure [3]. A mechanism to access data in each row in HBase uses Column Family and Column Quantifier as Key-Value pairs, where each cell of it can have many values with a timestamp as a marker or a differentiator [4].

There are many studies applied HBase in Big Data cases [5, 6, 7, 8] as a solution than used relational databases to handle unstructured data. In this research, HBase is applied in marketplace application with an emphasis on how to analyze and implement data of store as unstructured data by modeling in schemaless design.

Some of the problems in this research are: (1) what is the unstructured data model of marketplace application; (2) what is the mechanism for saving the unstructured data in HBase; (3) how to implement the unstructured data uses Java programming.

1.1. Our Contribution

This paper presents an analysis of marketplace applications based on a non-relational database approach using HBase with schemaless design exploitation.

1.3. Paper Structure

The following figure shows the stages of this research in detail.

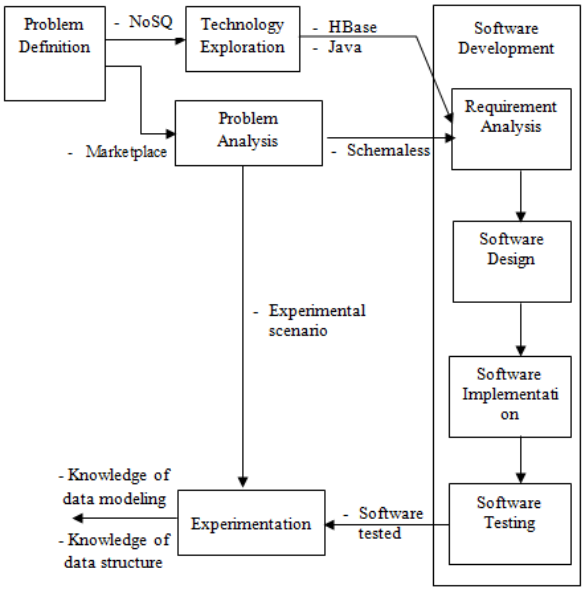

Figure 1. Research methodology

2. HBASE DATA MODEL

The column-oriented data model is a table that consists of many rows and each row consists of many columns where each row can have a different column

and the value of a particular column in a particular row can be more than one, so a timestamp is needed as a marker [9]. There are two concepts of an attribute as a differentiator for each row in designing table in HBase, namely, is a RowKey and columns are organized into one or more Column Families (CF). These Key-Value pairs are used to access each cell of data in HBase [9]. The cell in HBase is a combination of the RowKey, CF, and version (Timestamp).

```
A table = 1 Column Family (CF)
A row = 1 RowKey + 1 CF + 1 Column Qualifier (Column)
A cell = 1 RowKey + 1 CF + 1 Column + 1 Timestamp

Key →Value

[RowKey, CF: Column, Timestamp] →Cell or tuple
```

Figure 2. Key-Value pairs in HBase construction

To distinguish which table is meant, the construction becomes [Table, RowKey, CF: Column, and Timestamp] → Value. The following is the schema of the Student table.

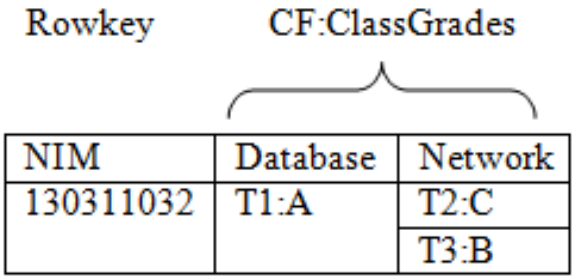

Figure 3. An conceptual of HBase data model instances

The process of creating tables is executed by using the create Table () method of HBaseAdmin class [2].

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(config);
HTableDescriptor tableDescriptor = new
HTableDescriptor(TableName.valueOf("StudentTable"));
tableDescriptor.addFamily(new HColumnDescriptor
        ("ClassGrades"));
admin.createTable(tableDescriptor);
```

Figure 4. Create table operation

Column-oriented is intended to deal with distributed big data, so the table is separated based on the CF and the Column itself as a qualifier, and each table is a separate HFile from one another [9].

RowKey	ColumnKey	Timestamp	Value
130311032	ClassGrades:Database	T1	A
130311032	ClassGrades:Network	T2	C
130311032	ClassGrades:Network	T3	B

Figure 5. An physical of HBase data model instances

The data type of the RowKey and Columnkeys are bytes, and the version is a long integer [9]. The version is stored in decreasing order where the most recent values of the cell are found first when reading a table in HBase. This mechanism allowing history data of particular value for (RowKey, ColumnKey) is stored in the same cell where the update process uses the same method as the input process.

```
HTable hTable = new HTable(config, "StudentTable");
 Put p = new Put(Bytes.toBytes("130311032"));
p.add(Bytes.toBytes("ClassGrades"),
    Bytes.toBytes("Network"),Bytes.toBytes("B"));
hTable.put(p);
```

Figure 6. Input and update operation

There is another method to keep only one value is stored in a cell by doing the delete process first, as seen in Figure 7, then followed by the input process, as seen in Figure 6.

```
HTable table = new HTable(conf, "StudentTable");
Delete delete = new
Delete(Bytes.toBytes("130311032"));
delete.deleteColumn(Bytes.toBytes("ClassGrades"),
Bytes.toBytes("Network"));
```

Figure 7. Delete operation

To get C and B of network values for NIM 130311032 in the Student table uses scan() method.

```
HTable table = new HTable(config, "StudentTable");
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("ClassGrades"),
Bytes.toBytes("Network"));
ResultScanner scanner = table.getScanner(scan);
for (Result result = scanner.next(); result != null;
result = scanner.next());
```

Figure 8. Scan operation

In a relational database, "StudentTable" is a database name, "ClassGrades" is a table name, and "Network" is a field name. To facilitate the scanning process that will be carried out for all pairs of field (Column Qualifier) and table (Column Family) values in one HBase database uses a unique name when storing data uses put() method. Avoid using the same name, e.g., "BasicInfo" for more than one schema. In this research, "SellerBasicInfo" was used for Seller and "CustomerBasicInfo" used for Customer.

3. RESULT

Each row in HBase consists of many columns that can have different columns, namely the schemaless. A differentiator in each row formed by the RowKeys and the ColumnKeys. A new CF is provided to store

additional data. Data modeling of marketplace application in this research consists of 4 tables with pastry as a business domain. There is unstructured data that the Seller may create as characteristic of the created shop or product [10]. An example is a Seller might provide various account bank numbers for the convenience of consumers making payments. Another example is the promotion of product prices based on variations in the production date and expired date or other unstructured data that may be added by the Seller in-store specifications or product information to attract the Consumer.

3.1. Data Design

- Customer

RowKey	CustomerBasicInfo				
CustomerNumber	CustomerUsername	CustomerPassword	CustomerName	CustomerPhone	CustomerAddress

Figure 9. Customer schema

There is just one table in the Customer schema that was created in HBase, which consists of username, password, name, phone, address columns. This data is used for the needs of shipping goods. The data structure of Seller schema, as seen in Figure 10.

```
public class Customer {
    private String customerUsername;
    private String customerPassword;
    private String customerName;
    private String customerPhone;
    private String customerAddress;
```

Figure 10. Customer data structure

- Seller

RowKey	SellerBasicInfo					Bank	
SellerNumber	SellerUsername	SellerPassword	SellerName	SellerPhone	SellerAddress	BankNumber	...

Figure 11. Seller schema

There are two tables in Seller schema that was created in HBase. First is the SellerBasicInfo table that consists of the username, password, name, phone, address columns. Second is the Bank table that consists of BankNumber and other BankNumber's columns of Seller that might be created by Seller later, so the possibility that a Seller has more than one bank account number can be stored with the same RowKey for all of these bank accounts in HBase. The data structure of the Seller schema, as seen in Figure 12.

```
public class Person {
    private int sellerNumber;
    private String sellerUsername;
    private String sellerPassword;
    private String sellerName;
    private String sellerPhone;
    private String sellerAddress;
    private List<int>
    bankNumberReference = new ArrayList<>();
```

Figure 12. Seller data structure

The value of the BankNumber field in Seller schema **refers** to BankNumber (RowKey) in Bank schema.

RowKey	BankBasicInfo	
BankNumber	BankName	BankAccount

Figure 13. Bank schema

The data structure of the Bank schema, as seen in Figure 14.

```
public class Bank {
    private int bankNumber;
    private String bankName;
    private String bankAccount;
```

Figure 14. Bank data structure

- Product

RowKey	ProductBasicInfo							ProductAdditional
ProductNumber	ProductionDate	ProductName	ExpiredDate	Type	Weight	Price	PhysicalStock	Picture	...

Figure 15. Product schema

There are two tables in the Product schema that was created in HBase. First is the ProductBasicInfo table that consists of production date, name, expired date, type, weight, price, physical stock, picture columns. Second is the ProductAdditional table that

might be created by Seller later as additional information or other information according to the sales strategy of the product where the product might be sold by more than one Seller. The data structure of the Product schema, as seen in Figure 16.

```
public class Product {
    private int productNumber;
    private Date productionDate;
    private String productName;
    private Date expiredDate;
    private String type;
    private String weight;
    private double price;
    private int physicalStock;
    private String picture;
    private List<String>
    productAdditional = new ArrayList<>();
```

Figure 16. Product data structure

The value of the ProductAdditional Column in Product schema contains string as other Product attributes but allowing to save more than one value using a list of string data types, namely product additional.

There are two categories of product stock, namely PhysicalStock, as seen in Figures 15 and 16. Another one is availableStock, as seen in Figure 19. The availableStock (client-side) represents the value of PhysicalStock (server-side) when the Customer selects the product and places it in the shopping cart. The actual value of the product stock is PhysicalStock, which will be taken from HBase and will replace the availableStock value when the Customer checkout. This causes the possibility of failure of a purchase transaction, where the value of PhysicalStock is 0 (zero), which means that the goods have been sold out. The cause of the failure is due to the time lag between the purchase process, and the checkout is long enough, so allowing other Customers to checkout at the period of the time lag. This often happens in flash sales.

- Purchase

RowKey		PurchaseDetails				Payment	
PurchaseDateTime	CustomerNumber	ProductNumber	PurchasePrice	PurchaseAmount	...	TotalPayment	BankNumber

Figure 17. Purchase schema

There are two tables in the Purchase schema that was created in HBase. First is the PurchaseDetails table that consists of product, price, purchase amount columns. The second is the Payment table that consists of total payment and BankNumber. The data structure of Product schema, as seen in Figure 18.

```
public class Purchase {
    private Date purchaseDateTime;
    private int customerNumber;
    private double totalPayment;
    private int bankNumber;
    private List<Cart>
    Cart = new ArrayList<>();
```

Figure 18. Purchase data structure

Schemaless model in Purchase schema is designed to record the occurrence of a customer purchasing for more than one item, where the purchase transaction data for one item **embedded** Cart object, which consists of ProductNumber, price, and PurchaseAmount columns. A list of purchases made by a Customer is stored in an array of a list, namely Cart if the value of PhysicalStock of Product is greater than zero, then it assigned to availableStock value as seen in Figure 19.

```
public class Cart {
    private int productNumber;
    private int availableStock;
    private double price;
    private int purchaseAmount;
```

Figure 19. Cart data structure

There are two columns as RowKey in Purchase schema, namely PurchaseDateTime and CustomerNumber, so that a Customer can make purchases of goods for more than one transaction on the same day. While the consideration of bank data is made into a separate table (see Figure 13), it aims to make it easier to store one Column (BankNumber) on the Payment table in Purchase schema. This is because there is a possibility that the growth in purchasing transactions occurs more frequently than getting a transfer list (BankName and BankAccount), which may be required by a Seller from a successful sales transaction as follows.

RowKey		PurchaseDetails				Payment		
TransactionDateTime	ConsumerNumber	ProductNumber	PurchasePrice	PurchaseAmount	...	TotalPayment	SellerBankName	SellerBankNumber

Figure 20. Purchase schema with canceled Payment design

The PurchasePrice column in Purchase schema aims to store the possibility of different prices for the same product when there is a discount for a product that enters the expiration date according to ProductionDate and ExpiredDate in the Product schema.

3.2. User Interface Design

Schemaless design is needed to provide additional data that might be inputted by the user. There are three tables applying schemaless to store unstructured data. First is the Bank table in Seller schema. The second is ProductAdditional in Product schema, and the last is PurchaseList in Purchase schema. Source code to save the additional bank, as seen in Figure 21.

```
//accepts column family name, qualifier name, value
p.add(Bytes.toBytes("Bank"), Bytes.toBytes
        ("BankName"), Bytes.toBytes(bankName));
p.add(Bytes.toBytes("Bank"), Bytes.toBytes
        ("BankAccount"), Bytes.toBytes(bankAccount));
// Saving the put Instance to the HTable.
hTable.put(p);
```

Figure 21. Add new BankNumber in Seller schema

There are three steps to add a new BankNumber value in the Seller schema. First, create BankNumber in Bank schema. Second, save BankName and BankAccount in Bank schema. The last is to store BankNumber in Seller schema, which refers to Bank schema. The user interface design it is seen in Figure 22.

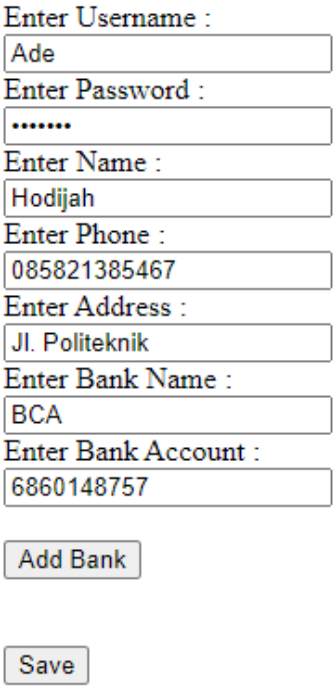

Figure 22. Schemaless in addition to the Seller's bank

Adding bank data can be done by a Seller by filling in the Enter Bank Name and Enter Bank Account values after Seller presses the “Add Bank” button, as seen in Figure 23, while “Save” button is used to submit the Seller data and stored it in HBase.

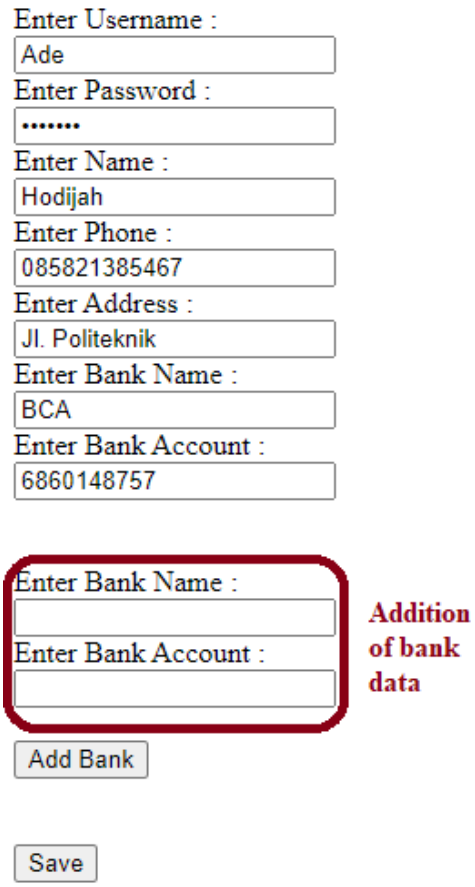

Figure 23. Addition of bank data

The Other ProductAdditional is different from the schemaless of bank data, where a new column qualifier that contains more than one value. HBase uses “addColumn” method to store the unstructured data [2].

```
// Instantiating configuration class.
Configuration conf = HBaseConfiguration.create();
// Instantiating HBaseAdmin class.
HBaseAdmin admin = new HBaseAdmin(conf);
// Instantiating columnDescriptor class
String newProductAdditional = "ProductAdditional".
        concat(String.valueOf(additionalNumber));
HColumnDescriptor columnDescriptor =
        new HColumnDescriptor(newProductAdditional);
// Adding column family in student table
admin.addColumn("marketplace", columnDescriptor);
```

Figure 24. Add new ProductAdditional[n] column

User interface design of ProductionAdditional as seen in Figure 25.

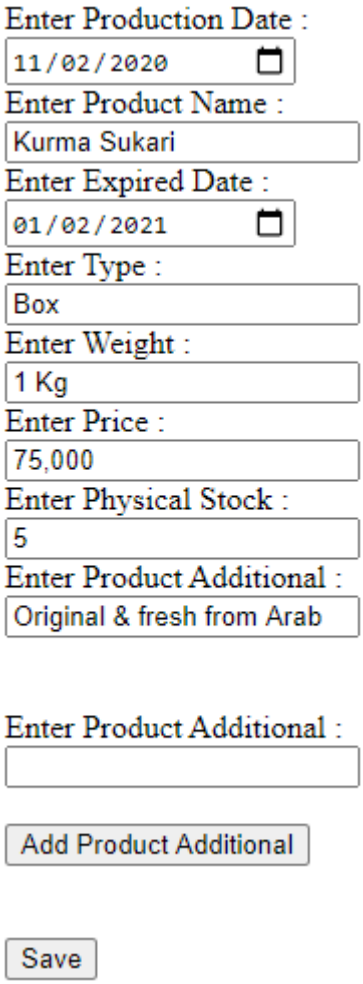

Figure 25. Addition of ProductAddtional data

Source code to save new ProductAdditional data, as seen in Figure 26.

```
//accepts column family name, qualifier name, value
p.add(Bytes.toBytes("ProductAdditional"), Bytes.toBytes
        (newProductAdditional), Bytes.toBytes
        (productAdditional));
// Saving the put Instance to the HTable.
hTable.put(p);
```

Figure 26. Add new ProductAdditional in Product schema

The last schemaless design is PurchaseList, which consists of adding three static columns like Bank data and needs to create a new column qualifier for each static Column like ProductAdditional data. First, create a new column qualifier, as seen in Figure 27.

```
// Instantiating configuration class.
Configuration conf = HBaseConfiguration.create();
// Instantiating HBaseAdmin class.
HBaseAdmin admin = new HBaseAdmin(conf);
// Instantiating columnDescriptor class
String newPurchaseList = "PurchaseList".
        concat(String.valueOf(additionalNumber));

HColumnDescriptor columnDescriptor =
        new HColumnDescriptor(newPurchaseList);
// Adding column family in student table
admin.addColumn("marketplace", columnDescriptor);
```

Figure 27. Add new PurchaseList[n] column

Source code to save new PurchaseList data as seen in Figure 28.

```
//accepts column family name, qualifier name, value
p.add(Bytes.toBytes(newPurchaseList), Bytes.toBytes
        ("ProductNumber"), Bytes.toBytes
        (productNumber));
p.add(Bytes.toBytes(newPurchaseList), Bytes.toBytes
        ("PurchasePrice"), Bytes.toBytes
        (purchaseprice));
p.add(Bytes.toBytes(newPurchaseList), Bytes.toBytes
        ("PurchaseAmount"), Bytes.toBytes
        (purchaseAmount));
// Saving the put Instance to the HTable.
hTable.put(p);
```

Figure 28. Add new PurchaseList in Purchase schema

The purchase transaction of a certain item can occur simultaneously, so multithreading is required to be able to manage these events. This simultaneously means a product can be purchased by more than one Customer at the same time, and it is assigned in the Purchase data structure (see Figure 18) when the Customer performed to check out.

To ensure changes in the PhysicalStock value of a certain item are carried out by one purchase transaction, synchronization is required as a result of the simultaneous transaction. Following is a simultaneous illustration of one product that is bought by more than one Customer at the same time, as seen in Figure 29.

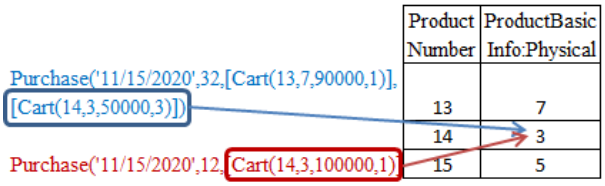

Figure 29. The simultaneous transaction

Figure 29 shows Cutomer_32 and Customer_2 purchased ProductNumber_14 on November 15, 2020. Since Customer_32 had successfully checked out, then Customer_12 would have failed to perform the checkout process.

The synchronized method to handle the simultaneous transaction conducted in two steps as follows:

1. Check the value of PhysicalStock
2. IF PhysicalStock >= purchaseAmount THEN PhysicalStock = PhysicalStock – purchaseAmount

The synchronized method is seen in Figure 30.

```
public synchronized boolean updateQuantityItem
    (int productNumber,
    int purchaseAmount) throws IOException {

    int physicalStock = 0;
    try {
        Scan scan = new Scan();
        FilterList filters = new FilterList(FilterList.
            Operator.MUST_PASS_ALL);
        Configuration conf = HBaseConfiguration.create();
        HTable table = new HTable(conf, "marketplace");
        Filter getPhysicalstock = new PrefixFilter(Bytes.
            toBytes(String.valueOf(productNumber)));
        filters.addFilter(getPhysicalstock);
        scan.setFilter(filters);
        ResultScanner scanner = table.getScanner(scan);
        for (Result result : scanner) {
            byte[] vPhysicalStock = result.getValue(Bytes.
                toBytes("ProductBasicInfo"), Bytes.
                toBytes("PhysicalStock"));
            physicalStock = Bytes.toInt(vPhysicalStock);
        }
        scanner.close();
        if(physicalStock >= purchaseAmount) {
            int updateStock = physicalStock -
                purchaseAmount;
            Delete delete = new Delete(Bytes.toBytes
                (String.valueOf(search)));
            delete.deleteColumn(Bytes.toBytes
                ("ProductBasicInfo"),
                Bytes.toBytes("PhysicalStock"));
            table.delete(delete);
            Put p = new Put(Bytes.toBytes(String.valueOf
                (search)));
            p.add(Bytes.toBytes("ProductBasicInfo"),
                Bytes.toBytes("PhysicalStock"), Bytes.
                toBytes(updateStock));
            table.put(p);
            table.close();
            return true;
        }
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
```

Figure 30. Synchronized update PhysicalStock

To get physicalStock value of the product uses the Filter method [9]. Furthermore, to update PhysicalStock perform delete operation first then followed by put operation. Following is a test of the PhysicalStock update process that occurs when purchasing products with a value of productNumber is “rowKue0” that occurs simultaneously by different Customers, namely “rowPembeli0” up to “rowPembeli7”. The testing is performed using the thread function.

```
public static void main (String[] args) throws IOException {
    ThreadTest t = new ThreadTest();
    new Thread(new ThreadTest("rowPembeli0", "rowKue0", 1)).start();
    new Thread(new ThreadTest("rowPembeli1", "rowKue0", 2)).start();
    new Thread(new ThreadTest("rowPembeli2", "rowKue0", 3)).start();
    new Thread(new ThreadTest("rowPembeli3", "rowKue0", 4)).start();
    new Thread(new ThreadTest("rowPembeli4", "rowKue0", 5)).start();
    new Thread(new ThreadTest("rowPembeli5", "rowKue0", 6)).start();
    new Thread(new ThreadTest("rowPembeli6", "rowKue0", 7)).start();
    new Thread(new ThreadTest("rowPembeli7", "rowKue0", 8)).start();
}
```

Figure 31. Thread test

4. CONCLUSION

There is a need for adding data that may be done by users in the marketplace application. It needs to be supported by the selection of database technology, which is also flexible in making a record in a database table. In this research, the difference in the number of different attributes is known as unstructured data which is supported by the schemaless database design feature in HBase.

REFERENCES

[1] Spaggiari, J.M., O’Dell, K., Architecting HBase Applications, O’Reilly, 2016. http://2.droppdf.com/files/XiiBn/architecting-hbase-applications.pdf

[2] tutorialspoint.com

[3] ibm.com

[4] Shehata, N.S., Big data with Column oriented NoSQL Database to overcome the drawbacks of relational database, International Journal Advanced Networking and Applications, Vol.11, Issue.5, pp. 4423-4428, February, 2020. ISSN: 0975-0290

[5] M. Radoev, “A Comparison between Characteristics of NoSQL Databases and Traditional Databases,” vol. 5, no. 5, pp. 149–153, 2017.DOI: http://dx.doi.org/10.13189/csit.2017.050501

[6] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, “Comparison between relational and NOSQL databases,” no. May, 2018. DOI: http://dx.doi.org/10.23919/MIPRO.2018.8400041

[7] C. Gyorodi, R. Gyorodi, and R. Sotoc, “A Comparative Study of Relational and Non-Relational Database Models in a Web- Based Application,” vol. 6, no. 11, pp. 78–83, 2015. DOI: http://dx.doi.org/10.14569/IJACSA.2015.061111

[8] Oliveira, F., Oliveira, A., & Alturas, B..”Migration of Relational Databases to NoSQL - Methods of Analysis”, Mediterranean Journal Of Social Sciences, 9(2), 227-235, 2018. DOI: http://dx.doi.org/10.2478/mjss-2018-0042

[9] hbase.apache.org

[10] Hutami, Rr. R., Rahmah, Z, "Pengaruh Atribut Toko Online Terhadap Perilaku Pembelian Online Konsumen Lazada Indonesia", KINERJA, 149-160, 2016. DOI: http://dx.doi.org/10.24002/kinerja.v20i2.841