# A COMPARATIVE STUDY ON BIG DATA HANDLING USING RELATIONAL AND NON-RELATIONAL DATA MODEL

Jannatul Maowa, A. H. M. Sajedul Hoque, Rashed Mustafa and Mohammad Osiur Rahman

Dept. of Computer Science & Engineering, University of Chittagong, Chittagong, Bangladesh

## *ABSTRACT*

*Unstructured data, which is the heart of big data, are generating in every moment due to the revolution of Internet. Relational data model is not the right tool to handle such unstructured data because it has limitation of scalability, accessibility, flexibility, agility and in addition redundancy in database. In order to resolve those flaws of relational data model, a number of non-relational data models have been invented. Among those data models, this paper focuses on document oriented data model on some cosmetic products, explores the different shortcomings of relational data model compared to non-relational data model and suggests the best data model for cosmetic product.*

## *KEYWORDS*

*Big Data, Unstructured Data, NoSQL*

## 1. INTRODUCTION

Data play a vital role in all sectors and functions of the global economy by discerning patterns and making better decisions. In the last decade the volume, velocity and variety of recorded information have increased drastically and the existing data storages and processing tools were not able to handle that sort of data, which implies the Big Data tools nowadays. Big data can include both structured and unstructured data, where unstructured data does not follow any specified format. Traditional relational data model fails to manage and analyse the huge volume of unstructured data. According to International Data Corporation (IDC) survey, unstructured data takes a lion's share in digital space and approximately occupies 80% by volume compared to only 20 for structured data [1]. Thus the use cases of unstructured data are expanding rapidly. The most popular applications to handle unstructured data are storing Log Data, pre– Aggregated Reports, Hierarchical Aggregation, Product Catalogue, Inventory Management, Category Hierarchy, Metadata and Asset Management and Storing Comments [2]. This paper analyses different merits of non-relational data model compared to relational data model by examining basic patterns and principles for a cosmetic E-Commerce product catalogue system.

The rest of the paper is organized as follows. Section II describes the basics of unstructured data. Polyglot persistence is described in section III. The Product Catalogue handling system is presented in section IV. Section V and VI illustrates the comparative study of relational and non relational data model on sample product catalog and Section VII illustrates driving philosophy of non-relational over relational data model. Finally, section VIII concludes this paper.

## 2. UNSTRUCTURED DATA

Unstructured data (or unstructured information) means information that have an undefined data model. Unstructured information is typically text-heavy, but may hold data such as dates, numbers, and facts as well. The exponential enhancement of unstructured data presents both issues and opportunities in research. It does not conform to strict schemas or data which do not have the same attributes all the time. These data are generated via call centre logs, emails, documents on the web, blogs, tweets, customer comments, customer reviews, and so on. Due to increasing use of the number of smart phones and tablets in the society, these data are created everywhere, all the time. Different business organizations are using such data to summarize, understand and make sense for making better business decisions. In order to handle such type of schema less entities, NoSQL [3] (most interpreted as "not only sql") database system is used. Generally speaking NoSQL systems could be a proper solution over RDBMS for handling use cases where data is unstructured. In computing, NoSQL is a broad class of database management system identified by its non-adherence to the widely used relational database management system model. That is, NoSQL databases are not primarily build on tables, and as a result, generally do not use SQL for data manipulation [4]. It is a non-relational database management system which is different from traditional relational database management systems. This type of data may not require fixed schema for storing, avoid join operations and typically scale horizontally. It is designed for distributed data stores where storing very large scale of data needs. There are four types of NoSQL storage such as Column-oriented database (such as Apache Cassandra, HBase, Google Big Table and so on), Document store (such as MongoDB, CouchDB, Redis, BaseX and so on), Key value store (such as Berkley DB, MemcacheDB and so on) and Graph store (such as Neo4j,Titan,ObjectDB and so on). Among those this paper will look on document store NoSQL database system.

## 3. POLYGLOT PERSISTENCE

Polyglot Persistence is a greatly popular addiction in the NoSQL database domain, because it obviously makes sense to use "the right data model" for each specific division of an architecture or application. Simply it means picking the right tool for the right use case. Current e-commerce web applications are so smart and do not depend on only one database management system. They are handling different types of data with different DBMS. Looking at a Polyglot Persistence example, an e-commerce platform will deal with many types of data (i.e. shopping cart, inventory, completed orders, etc) illustrated in fig. 1[5]. Instead of trying to store all this data in one database, which would require a lot of data conversion to make the format of the data all the same, store the data in the database best suited for that type of data. In this e-commerce application, shopping cart and session related data are stored in Key-Value store NoSQL database, different orders are stored in document store NoSQL, RDBMS is used for inventory and graph store NoSQL is used for storing customer social graph.

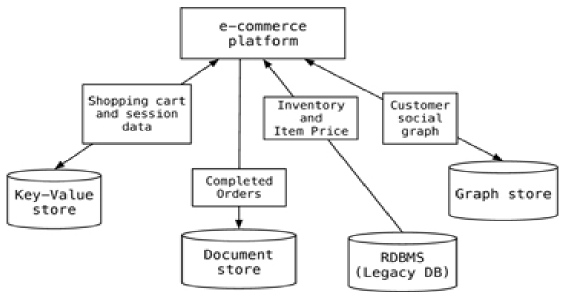

Fig.1 Polyglot persistence of e-commerce platform

## 4. Product Catalogue Handling System

Product Catalogue System imports, exports, creates, and manages online catalogues in every e-commerce site. Among different product catalogue related use cases Insert Product Item is very common use case, which is illustrated in fig. 2.

*UC: Insert Product Item*
*Actors: sales man*
*Pre Conditions:*
- *1. The user is logged into the system*
- *2. The user is on the Home page*

*Main Scenarios:*
- *1. User activates the insert function.*
- *2. system prompts for p_id , name , price etc.*
- *3. User enters the required values.*
- *4. System checks the validity of the values.*
- *5. System displays a form containing selected field for user judgement.*
- *6. User clicks*

*'Submit' Post Condition:*
- *The values are saved by the system and a confirmation message is displayed.*

*Alternative scenario:*
- *4.a. the entered values is not valid*
  - *4.a.01 System displays an error message.*

Fig. 2 Use case to Insert Product Item

Product catalogues must have the capacity to preserve various types of objects with different sets of attributes. The data model behind the use case in fig. 2 may be relational (structured) or non-relational (unstructured). The objective of this paper is to retrieve the better model among non-relational and different types of relational models for aforementioned use case. The following section examines several of these options of relational and then describes the preferred non-relational database solution.

## 5. Relational Data Model Of Product Catalogue

A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to re-organize the database tables [6]. In order to implement the product catalogue related use cases like view product catalogue, insert product, update product catalogue, few related tables should be created. There are different approaches to build the schema of the table, which has been described below.

### 5.1 Concrete Table Inheritance

Concrete Table Inheritance uses one table for each class in the hierarchy. Each table contains columns for the class and all its ancestors, so any fields in a superclass are duplicated across the tables of the subclasses [7]. That is, in this approach each product category should have an individual and independent table. This paper is based on the two cosmetic products: Eye and Face, whose schemas are shown in fig. 3[8].

*Product_Eye (P_Id, Name, Colour, Price, Rating )*

*Product _Face (P_Id, Name, Amount, Price, Rating )*

Fig. 3 Schema for Eye and Face Products

In this approach, each table is self-contained and does not have any unnecessary fields. For example, since the feature amount is redundant in Product_Eye table, it is not included in that table. Two sample tables on the above schema are shown in table 1 and 2 respectively. It does not need any join operation to view or create any reports in any application.

Table 1. Product Eye

| P_Id | Name | Colour | Price ($) | Rating |
|---|---|---|---|---|
| 1001 | Shadow | Green | 20 | 5 |
| 1002 | Liner | Black | 40 | 2 |
| 1003 | Mascara | Black | 70 | 1 |
| 1004 | Brow | Vivid | 50 | 3 |

Table 2. Product Face

| P_Id | Name | Amount (ml) | Price ($) | Rating |
|---|---|---|---|---|
| 2001 | Foundation | 200 | 200 | 4 |
| 2002 | Powder | 100 | 150 | 4 |
| 2003 | Blush | 50 | 300 | 2 |
| 2004 | Face kits | 300 | 500 | 3 |

Concrete Table Inheritance approach is a clumsy approach, as the database application based on this approach is static, not dynamic. That is, it is fully product-oriented approach which implies tight binding with application. As a result no general database can be built and module independency will be lost in software model. In addition, if an application has multiple products, multiple tables should be created in this technique, which increases the complexity to manage tables. Moreover, a data model must be built before developing the related software. If a new product is wanted to add or delete through this method, then the whole data model will have to be redesigned which increases time complexity. These limitations of the Concrete Table Inheritance demand new technique called Single Table Inheritance (STI).

## 5.2 Single Table Inheritance

Single-table inheritance (STI) is the practice of storing multiple types of values in the same table, where each record includes a field indicating its type, and the table includes a column for every field of all the types it stores [9]. This model uses a single table for all products with its own attributes as columns. The main advantage of this technique is any item can be added inside the table without creating new table any time. After combining the Product Eye and product Face tables, a new table named Product has been formed illustrated in table 3.

Table 3. Product

| P_Id | Name | Colour | Price ($) | Amount (ml) | Rating |
|---|---|---|---|---|---|
| 1001 | Shadow | Green | 20 | Null | 5 |
| 1002 | Liner | Black | 40 | Null | 2 |
| 1003 | Mascara | Black | 70 | Null | 1 |
| 1004 | Brow | Vivid | 50 | Null | 3 |
| 2001 | Foundation | Null | 200 | 200 | 4 |
| 2001 | Foundation | Null | 200 | 200 | 4 |
| 2003 | Blush | Null | 300 | 50 | 2 |
| 2004 | Face kits | Null | 500 | 300 | 3 |

Using this approach general table of an application can be easily built, where it does not matter the type of product. This application can make reports by executing single queries on one table with no joint operation. That's why this approach is better than concrete table inheritance. Building table using Single Table Inheritance is a good approach if the attributes in the child tables are same. Otherwise null values will be appeared in combined table which implies redundancy in database. For example since the Product Eye and Product Face table have two different attributes: Amount and Colour, null values are seen in table III. Null values will be linearly increased with the increasing of dissimilar attributes of child tables, which kill huge memory spaces.

## 5.3 Multiple Table Inheritance

Due to the redundancy of in table created by STI approach, a most efficient technique named Multiple Table Inheritance (MTI) is used to build table in database, where all tables are in normalized form. In this approach, one generic table with all common attributes of all products is created and individual tables with different attributes are created. Each table should have a primary key and each table except generic table must have a foreign key which refers to general table. The equivalent structure of schema in fig. 3 using MTI is shown in fig. 4.

*Product (P_Id, Nam, Price, Rating)*

*Product_Eye (E_Id, Colour, P_Id)*

*Product _Face (F_id, Amount, P_Id)*

Fig. 4 Schema using MTI

Here Product table is generic table and Product Eye and Product Face table has been formed with its distinct attributes: Colour and Amount. Both concrete tables have the same foreign key P_Id which refers to the generic table Product. Product, Product Eye and Product Face tables are shown in table 4, 5 and 6respectively. Formed tables using MTI are the combination of generic and concrete tables. It is more space-efficient than STI, as all null values shown in STI approach have been completely omitted. Since MTI approach has no hard restriction to build for each product, MTI is more flexible than CTI. That is, if any upcoming product which has only common features, no different feature, it is not necessary to build an individual table for that product, only needs to insert information into generic table. However, this model does require an expensive JOIN operation to obtain all relevant attributes relevant to a product.

Table 4. Product

| P_Id | Name | Price ($) | Rating |
|---|---|---|---|
| 101 | Shadow | 20 | 5 |
| 102 | Liner | 40 | 2 |
| 103 | Mascara | 70 | 1 |
| 104 | Brow | 50 | 3 |
| 105 | Foundation | 200 | 4 |
| 106 | Powder | 150 | 4 |
| 107 | Blush | 300 | 2 |
| 108 | Face kits | 500 | 3 |

Table 5. Product Eye

| **E_Id** | **Colour** | **P_ID** |
|---|---|---|
| 1001 | Green | 101 |
| 1002 | Black | 102 |
| 1003 | Black | 103 |
| 1004 | Vivid | 104 |

Table 6. Product Face

| **F_Id** | **Amount (ml)** | **P_Id** |
|---|---|---|
| 2001 | 200 | 105 |
| 2002 | 100 | 106 |
| 2003 | 50 | 107 |
| 2004 | 300 | 108 |

## 6. NON-RELATIONAL DATA MODEL OF PRODUCT CATALOGUE

The main principle of non-relational data model is the organization of data has no logical structure like table in relational model. Among different types on non-relational data model this paper examines the document model used in Mongo DB and Couch DB. In this approach data are stored in document which contains one or more fields, where each field contains a typed value, such as string, date, binary or array [10].

```
{
P_id: "1001",
type: {
        product_type: "Makeup",
        product_name: "Shadow"
        }
title: "Eye Z You", details:{
          used_for: "Eye", color:
          "Green", shades: 6,
          portabe: "Yes"
          packaging: "Special"
          },
price: 20
}
```

```
{
P_id: "1002",
type: {
        product_type: "Makeup",
        product_name: "Liner"
        }
title: "Fluideline", details:{
          used_for: "Eye", gel:
          "Formula", wearing:
          "Long" precise: "Yes"
},

price: {
        original_price: 50,
        new_price:40
        }
}
```

(a) (b)

Fig. 5. Eye documents (a) Shadow (b) Liner

Each document is treated as object in object oriented programming and as record in relational database. Fig. 5 presents the documents of two products: liner and shadow.

## 7. Driving Philosophy Of Non Relational Over Relational Data Model

The advantages of document-oriented database over relational data model have been explained below.

### 7.1. Reduce Redundancy

It has been observed that both documents in fig. 5 are for the same product called Makeup, but their structures are different. Each document includes only the relevant fields. For example, color field is relevant for Eye Z You product, but not for Fluid line product. This is not possible in relational model. It does not matter if it is relevant or not. Any irrelevant field in relational model holds null value which prompts redundancy. Since there is no scope of holding null value to a specific field in document-oriented non-relational data model, it reduces redundancy completely from database.

### 7.2. Increase the Flexibility

In relational data model, it is not possible to add any extra field out of its predefined fields as required. But in non- relational data model, anytime any field can be added. That is, non-relational database is vertically scalable. For example, the next upcoming product Fluid line has an extra field length, which does not belong to current product, this field can be easily added in its document without any difficulties.

### 7.3. Unlimited Fields

Relational database has hard limit for the number of fields. For example MySQL and Oracle supports maximum 4096 and 1000 columns in one single table respectively [11][12]. There are some data sets which has more than 4096 features. For example Amazon Access Samples Data Set has 20000 fields [12] which cannot be handled using relational database management system. Non-relational database has no such type of restriction; unlimited number of fields can be added.

### 7.4. Simplify Data Access

In non-relational data model, every document is complete with its all required fields. That is, each document does not refer to other documents for other fields illustrated in fig.5, which is happening in relational data model by imposing foreign keys. As non-relational data model has no foreign key and does not need any JOIN operation, inserting, deleting, updating and searching operations on this database are very simple.

### 7.5. Increases the Agility

According to the documents in fig. 5, every document has its own structure which is not predefined and that structure will be fixed up during the building of application. This is the beauty of non-relational data model which implies the dynamic schema. This dynamic nature of the schema increases the speed of development or agility. A core principle of agile development is adapting to evolving application requirements: when the requirements change, the data model also changes. This is a problem for relational databases because the data model is fixed and defined by a static schema. So in order to change the data model, developers have to modify the schema. By comparison, a NoSQL document database fully supports agile development, because it is schema-less and does not statically define how the data must be modelled.

## 8. CONCLUSIONS

The proliferation of multiple non-relational databases is created new dimension of data management landscape. Instead of having to force structures onto their data, organisations can now choose NoSQL database architectures that fit their emerging data needs, as well as combining these new technologies with conventional relational databases to drive new value from their information. The data of an organization are scaling in horizontal or vertical direction. To manage those data is big challenge. Unstructured databases are better to solve those problems. Therefore, with NoSQL, enterprises are better able to both develop with agility and operate at any scale – and to deliver the performance and availability required to meet the demands of Digital Economy businesses. Data model behind the use cases of an application can be varied. When the data of a use case has certainty about the features, relational data model is perfect for that. Otherwise, non-relational data model is suitable. A new general data model can be invented for all sorts of use cases in future.

## REFERENCES

[1] Dr. Goutam Chakraborty, Analysis of Unstructured Data: Applications of Text Analytics and Sentiment Mining, paper-1288(2014)

[2] Use Cases, https://docs.mongodb.com/ecosystem/use-cases/, 8 June 2016.

[3] NoSQL, DBMS, TechTarget, http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL, 29 June 2016

[4] Gaurav Vaish, Getting Started With NoSQL, p.7, 9 June 2016.

[5] James Serra's Blog, Big Data and DataWarehousing, http://www.jamesserra.com/archive/2015/07/what-is-polyglot- persistence/, 9 June 2016.

[6] Relational database, SQL Server Platforms, Search SQL Server, http://searchsqlserver.techtarget.com/definition/relational-database, 11 June 2016.

[7] Inheritance, http://propelorm.org/Propel/documentation/09- inheritance.html , 11june 2016

[8] MAC Cosmetics,http://www.maccosmetics.com/products/13838/Products/Makeup/Eyes/Liner,11 June 2016.

[9] Refactoring our Rails app out of single-table inheritance, https://about.futurelearn.com/blog/refactoring-rails-sti/, 12 june 2016

[10] The Mongo DB white paper, Top 5 Considerations When Evaluating NoSQL Databses, August 2015

[11] C.10.4 Limits on Table Column Count and Row Size, https://dev.mysql.com/doc/refman/5.7/en/column-count-limit.html, 22 June 2016

[12] Logical Database limits, https://docs.oracle.com/cd/B28359_01/server.111/b28320/limits003.html, 22 June 2016

[13] Amazon Access Samples Data Set, http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples,22 June 2016