BIG DATA TECHNOLOGY – CAN WE ABANDON THE TEACHING OF NORMALISATION?

Bernadette M Byrne[1], David Nelson[2], Renuga Jayakumar[1]

[1] *School of Computer Science, University of Hertfordshire (UNITED KINGDOM)*
[2] *Faculty of Computer Science, University of Sunderland (UNITED KINGDOM)*

Abstract

The design and implementation of Relational Databases is a well-established process. Many traditional text books for teaching database systems begin their introductory chapters with the ANSI/SPARC three level architecture diagram. Generally, this diagram consists of a conceptual model with external user views, a logical model in the target DBMS followed by a physical database design and implementation stage. However, with the advent of Big Data and so called "schemaless" databases and NoSQL databases this appears to have turned this process on its head. Or has it? In this paper, we will compare the data structures of a traditional relational database with a document based schemaless database (MongoDB) at the logical and physical database design stage. We will use the traditional concepts of the data definition language, data manipulation language and data control language of both types of database. We also briefly compare the query languages and the purposes of both types of databases. We will answer the question in the title of the paper "Big Data Technology – can we abandon the teaching of Normalisation?"

Keywords: Big Data, Relational Databases, NoSQL, MongoDB, SQL, NoSQL, Normalisation.

1 INTRODUCTION

Relational databases have been the mainstay of traditional data processing since the mid-1980s. Schools in the United Kingdom (UK) teach relational databases using the Microsoft Access database and the topic is assessed by the main examining bodies. In many Universities in the UK the undergraduate practical teaching of Relational Database Management Systems (RDBMS) is carried out using the Oracle Database Management System (DBMS). Modelling for Relational Databases has a well-established process. Databases using the relational model of data traditionally use normalisation to decide on a suitable set of data structures to implement in the target relational database [1]. Many undergraduate traditional text books for teaching Data Modelling begin their introductory chapters with the ANSI/SPARC three level architecture. Generally, this architecture consists of a conceptual model with external user views, a logical model in the target DBMS and then a physical database design stage. For conceptual modelling usually one of the many variations of Entity Relationship (ER) diagrams is used [2]; or in more recent years a Unified Modelling Language (UML) class diagram. Peter Chen [3] put forward the Entity Relationship model which later became a conceptual modelling tool for relational databases (as well as for other types of databases). There has been subsequent extensions to this model to take in the concepts of specialisation and generalisation [4].

In a relational database the logical model is a set of tables in at least third normal form (arrived at by normalisation or a top down approach) [5]. The logical model is then implemented in the RDBMS of choice normally using SQL data definition language (DDL), the database is queried using SQL data manipulation language (DML) and multi-user access can be granted using SQL Data Control Language (DCL). This whole process is well-established. This process has been extant since Codd put forward his watershed paper in 1970 [1]. Normalisation is covered in all standard database textbooks (sometimes in torturous detail) and as C. J. Date says is a "scientific principal which can be brought to bear on the database design problem" [6].

In recent years we have seen the advent of Big Data, NoSQL databases and so called "schemaless" (or semi-structured) databases. Schemaless gives the impression that the conceptual and logical design levels are not necessary and NoSQL (which is actually an abbreviation for *Not Only* SQL) gives the impression that the SQL query language is redundant. NoSQL databases are mostly open source, non-relational, distributed, and designed for large volumes of data stored across many clusters supporting replication and partitioning, parallel processing and what is usually called horizontal scaling. When teaching Big Data we usually start with the three (or now four V's). Volume, Variety,

Velocity and more recently Veracity. The **Volume** of the data is huge, the data can come from various sources and be different types of data (**Variety**), the data is generated quickly (for example tweets) and flows in quickly (**Velocity)** and lastly the data has to be meaningful and "clean" (**Veracity**). The data is mined and analysed to help with business decision making, for sentiment analysis and to discover trends. The volume of the data is typically too large to be managed in a regular RDBMS with its tight structure and the rigidity of ACID (atomicity, consistency, isolation and durability) properties for transactions.

The NoSQL data model has been broadly divided into four categories: document store; key-value store; column-store; and graph. At the time of writing MongoDB is one of the most popular document store schemaless databases in use. As with many NoSQL databases, MongoDB does not have an end-user friendly query language similar to SQL but relies heavily on JavaScript notation. This is maybe not an issue as the end-users may be mainly data analysts (or programmers producing reports for business analysists) whereas with traditional on-line transaction processing (OLTP) databases there could be large groups of end-users. The DCL, DML and the DDL are much less user-friendly in MongoDB than with SQL and require 3GL skills. However, both types of databases have different purposes. In this paper we will compare the data structures of a traditional relational database with a document based schemaless database (MongoDB) at the logical and physical database design stage. We will use the traditional concepts of the data definition language, data manipulation language and data control language of both types of database. We also briefly compare the query languages and the purposes of both types of databases.

2 BIG DATA, SCHEMALESS DATABASES AND NOSQL DATABASES

'Not only' SQL is abbreviated to NoSQL, giving the impression that SQL is now redundant. NoSQL databases are mostly open source, non-relational, distributed, and designed for large volumes of data stored across many clusters supporting parallel processing. Facebook, Twitter and Amazon all use different types of Schemaless Databases. At the time of writing there are four main types of NoSQL/Schemaless database.

2.1 The first main type of schemaless database is the Document Store

The information will be stored in the form of document. The document structure will be similar to an XML document. The NoSQL database document is semi-structured, de-normalised and hierarchical. Table 1 is an example of a data structure in a document store database. MongoDB, CouchDB and RavenDB are examples of document store databases.

Table 1. *Example Data Structure for Document Store Database.*

EXAMPLE 1	**EXAMPLE 2**
{	{
policyID :123456	policyID :123457
policyHolderName: John Smith	policyHolderName: Alice Maud
policyHolderAddress: 123, Dukes Avenue	policyHolderAddress: 12, Arundel Road
policyPostcode : HA1 3UJ	policyPostcode : HA1 5IJ
policyHolderCity: London	policyHolderCity: London
vehicleID: BD51 SMR	vehicleID: CD51 M6J
vehicleModel: Coupe	vehicleModel: Coupe
vehicleYear: 2010	vehicleYear: 2007
}	}

2.2 The second type of schemaless database is the Key-Value pair.

This is the simplest form of data model used by a NoSQL database. Each key has been associated with a simple value or complex data. Table 2 is an example of a key-value pair. Project Voldemort, Riak and Redis are examples of simple key-value pair database systems.

Table 2. *Example Key-Value Pair.*

KEY	VALUE
policyID	123457
policyHolderName	Alice Maud
policyHolderAddress	12, Arundel Road
policyPostcode	HA1 5IJ
policyHolderCity	London
vehicleID	CD51 M6J
vehicleModel	Coupe
vehicleYear	2007

2.3 The third type of schemaless database is the Column Store

As with relational databases, the values are stored in a table format but the column database supports unstructured data such as images, text or complex data. It provides more flexibility than a normal RDBMS which supports normalised, structured and flat data types. Cassandra, Apache and HBase are examples of column store systems.

Table 3. *Example Column Store for table storing Teacher details.*

ROW_KEY	PERSONAL_DATA		PERSONAL_DETAILS		CODE_DETAILS	
TeacherID	Name	Telephone	YearOfBirth	Gender	DepartmentId	Code
1	John	Smith	1977	M	A1	1234
2	Jack	Black	1969	M	A1	5678
3	York	Cox	1970	F	A2	9876

2.4 The fourth type of schemaless database is theGraph Store

Graph database systems store information about networks of data. Transport links and social media relations are examples of applications which could easily be handled with this type of database. Figure 1 is an example of a graph database. Neo4j is an example Graph type NoSQL database.

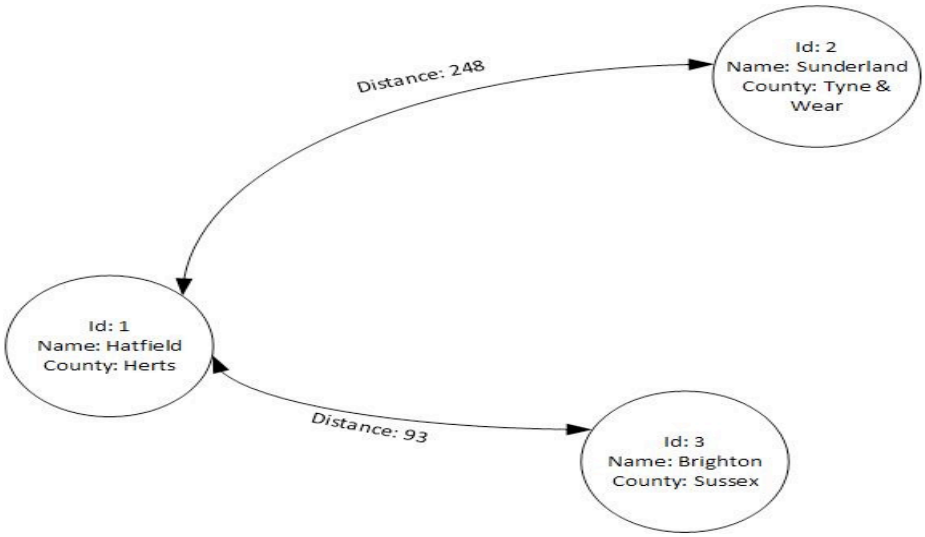

Figure 1. Example Graph Database.

3 MONGO DB – AN EXAMPLE OF A DOCUMENT STORE NOSQL DATABASE

MongoDB is an open-source, NoSQL database originally developed during 2007 by a software company called 10gen which in 2013 changed its name to MongoDB. MongoDB uses a document structure similar to the JSON document model to represent its schema. In this paper we concentrate on comparing MongoDB to SQL. Similar to SQL commands in a classic RDBMS MongoDB supports DDL, DML and DCL statements. The following part of the paper will compare the SQL commands between RDBMS and similar statements in MongoDB.

Table 4 gives a comparison of the main structures and functions in SQL and MongoDB. An SQL table can be compared with a MongoDB collection, and an SQL row can be compared with a MongoDB row or document. A column is comparable to a field and SQL joins can be compared to embedded documents or linking in MongoDB.

Table 4. *Comparison of Main Structures and Functions.*

SQL	**MongoDB**
Table	Collection
Row	Document or Row
Column	Field
Joins (linking the table)	Embedded documents or linking by object-identity
Primary Key	Row key (Automatically set)

3.1 A comparison of the Data Definition Language in MongoDB and in SQL

Table 5 gives a comparison of Data Definition Language commands in MongoDB and in SQL. All SQL examples have been written using Oracle SQL 11g syntax. The version of MongoDB used was version 3.4.

Table 5. *Comparison of DDL.*

SQL	**MongoDB**

CREATE TABLE <tablename>
(
  column_name1 data type 1,
  column_name2 data type 2,
…….
  column_nameN data type N
);
SQL EXAMPLE:
CREATE TABLE employee(
  employeeID VARCHAR(6) PRIMARY KEY,
  empAge NUMBER(3),
  empDesignation VARCHAR(15));
	db.createCollection(“collectionname”)
	or
	db.collectionname.insert
	Primary key has been created automatically. First insert command will generate a collection automatically.
	MongoDB EXAMPLE:
	db.createCollection(“employee”)
	or
	db.employee.insert(
	{
	  employeeID : “LOB123”,
	  empAge : 24,
	  empDesignation : “Manager”,
	}

ALTER Table <tablename>
  ADD column_name data_type;
 	
ALTER TABLE <tablename>
  DROP COLUMN column;
	Updates happened at document level rather than collection level. MongoDB does not enforce changes at structure.
	db.collectionname.update()
	$set is used to add a new column and $unset is used to drop a column within a MongoDB document.
	{multi:true} is optional. It represents multiple changes in a document that satisfies the criteria.

SQL EXAMPLE:
ALTER Table employee
ADD D_O_J Date;
 	
ALTER TABLE employee
DROP COLUMN empAge;
	MongoDB EXAMPLE:
	db.employee.update(
	  {},
	  {$set: {D_O_J :new Date()}},
	  {multi: true}
	)
	db.employee.update(
	  {},
	  {$unset: {empAge : 24}},
	  {multi: true}
	)

DROP TABLE <tablename>
SQL EXAMPLE:
DROP TABLE employee;
	db.collectionname.drop();
	MongoDB EXAMPLE:
	db.employee.drop();

Foreign keys allow us to relate tables where we have a relationship between tables
 	
SQL EXAMPLE:
CREATE TABLE project(
projectId VARCHAR(6) PRIMARY KEY,
startDate DATE,
endDate DATE);
CREATE TABLE employeesOnProject
(employeeID VARCHAR(6)
  REFERENCES employee(employeeId),
projectID VARCHAR(6)
  REFERENCES project(projectId),
  PRIMARY KEY(employeeID, projectID));
	We can use embedded documents or object referencing to implement this. There are no similar constructs in MongoDB as a foreign key.

3.2 Data Manipulation Language Comparison

Table 6 gives a comparison of Data Manipulation Language commands in MongoDB and in SQL.

Table 6. *Comparison of DML.*

SQL	**MongoDB**
INSERT INTO employee (employeeID, empAge, empDesignation) VALUES ('LOB123', 24, 'Manager');	db.employee.insert({ employeeID : "LOB123", empAge : 24, empDesignation : "Manager", })

UPDATE employee
SET empDesignation = 'Asst. Manager'
WHERE employeeID= 'LOB123';
	db.employee.update(
	{
	employeeID : "LOB123"
	},
	{
	$set :{empDesignation: "Asst. Manager"}
	}
	)

DELETE FROM employee WHERE employeeID= 'LOB123'; or DELETE FROM employee;	db.employee.remove({ employeeID : "LOB123" }) or db.employee.remove({})
SELECT * FROM employee;	db.employee.find(); find() is a method used to search a document in MongoDB.
SELECT * FROM employee WHERE empAge > 20;	db.employee.find({empAge: {$gt: 20}}) $gt is one of many comparison operators
INSERT INTO project VALUES ('PRJ001', '15-MAY-2017', '30-SEP-2017'); INSERT INTO employeesOnProject VALUES ('LOB123', 'PRJ001');	db.project.insert({ projectID: "PRJ001", start_date: "15-MAY-2017", end_date: "30-SEPT-2017", employees: [{employeeID: "LOB123", empAge : 24, empDesignation : "Manager"}, {employeeID: "LOB124", empAge: 18, empDesignation : "PA"}] })
The above is an example of linking projects and employees as a many-to-many relationship using foreign keys.	The above is an example of creating a document containing an array of nested documents, i.e. a project containing multiple employees. Whereas in SQL we can use foreign keys to enforce data integrity, this is not enforced in document stores.

3.3 Data Control Language Comparison

The first usual administration task, to perform in the MongoDB shell, is to add users to configure access control. DCL statements are used to control the user's authority in the database so that a user could execute any granted operation in the schema, such as for creating objects (DDL), retrieving data with select (DML) or granting privileges to other users (DCL). Table 7 shows the DCL commands in SQL and MongoDB.

Table 7. *Comparison of DCL.*

SQL		**MongoDB**
GRANT	GRANT SELECT ON company.employees TO user99;	db.grantRolesToUser("user99", [{ role: "readWrite", db: "company"}], { w: "majority" , wtimeout: 4000 });
REVOKE	REVOKE SELECT ON employees FROM user99;	db.revokeRolesFromUser("user99", [{ role: "read", db: "company" }]);

4 EVALUATION

One cannot fail to notice the difference between the two databases as regards the DDL, DML and the DCL. The DCL, DML and the DDL are much less user-friendly in MongoDB than with SQL. One of the big advantages of SQL and RDBMs when they first appeared was that they were user-friendly in that they did not need expert programming knowledge to use. This was mainly because they were non-procedural in nature unlike their predecessors which were navigational and program dependent. The fact that they were non-procedural made them much easier to use. In fact SQL was described as a fourth generation language [7]. There was much discussion at the time that SQL would see the death of programmers and 3GL languages [7]. This obviously did not happen and what did happen is that these 4GLs were used by traditional programmers to speed up development and also by end-users (which caused its own problems). Today we can give students an SQL workbook in a laboratory session and they can learn and achieve much in an hour with no 3GL programming skills necessary.

Also, as stated earlier, because we can create a document in MongoDB which contains nested documents this obviously raises the question of redundant data – data stored more than once if these documents are nested in many documents. This of course is against the principles of normalisation where one of the main aims is to eliminate redundancy. Also if we have redundancy of data this can also lead to update and deletion problems. (For example if Department details are nested many times in Employee rows/records and the Department closes you have to delete each occurrence. This would not happen with normalised tables). The structure and modelling of Big Data seems almost perverse to those of us brought up on the tightly controlled relational databases with normalisation and referential integrity. For a database that requires full data integrity all of the time a schemaless database is not appropriate because there can be repeated data which can lead to update and deletion problems which normalisation aims to eliminate. The question is "Are we bothered?" given the use to which Big Data is put.

There will undoubtedly be attempts to graft on an SQL-like query language as a front end to NoSQL databases, but history tells us that this will cause the usual impedance mismatch problems which also happened with Object Relational databases [8], [9]. MongoDB will also now allow references between collections with linking and embedding (rather similar to how with Codasyl DBMS' we could add extra links to Hierarchical DBMS' back in the 1970s).

5 CONCLUSION

Relational databases have been successful partly due to their user-friendly query language – Structured Query Language (SQL), and because they had other advantages over the other large program dependant and navigational databases which were prevalent at the time; such as Hierarchical databases and Network databases.

In this paper we have compared the DDL, DML and DCL of a traditional relational database with a document based schemaless database. Document databases, of which MongoDB is one, are mainly procedural in nature, whereas SQL is largely non-procedural. At the moment MongoDB is not end-user friendly. This is maybe not an issue as the end-users may be mainly analysists, whereas with on-line transaction databases there could be large groups of end-users. To describe the NoSQL databases as schemaless is also a misnomer as you have to have a plan and you need to know what data you need and what are the best structures to use bearing in mind that MongoDB is hierarchical in nature and you need to decide on the nature of the hierarchy and what data you can embed in documents. Atenzi et al. comment in [10] that NoSQL systems make no distinction between the logical and physical schema. If this can be supported we have indeed turned the ANSI/SPARC architecture on its head!

However, in many ways we are comparing chalk with cheese. Both types of databases, relational and NoSQL, have very different purposes. We need to ensure that the new technologies which make up Big Data are as effectively embedded into the curriculum for university teaching and learning as they have been for relational databases, and, need to be taught alongside relational databases. We cannot stop the teaching of Normalisation yet!

ACKNOWLEDGEMENTS

We would like to acknowledge the work of Michalakis Ntallas. (MSc Software Engineering Computer Science Masters Project, University of Hertfordshire 2016), “An investigation into the application of database query languages in schema-less databases.”

REFERENCES

[1] E.F. Codd, “A Relational Model for Large Shared Data Banks,” *Communications of the ACM,* vol. 13, no. 6, pp. 377-387, 1970.

[2] I.Y. Song, M. Evans, E. Park, “A Comparative Analysis of Entity-Relationship Diagrams,” *Journal of Computer and Software Engineering*, vol. 3 no. 4, pp 427-459, 1995.

[3] P.P. Chen, “The Entity-Relationship Model: Towards a Unified View of Data,” *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9-36, 1976.

[4] R. Elmasri, S. Navathe, *Database Systems – Models, Languages, Design, and Application Programming.* Pearson Sixth editon pp. 224-238, 2011.

[5] B. Byrne, “Good Top-down design methodologies tend to produce fully normalized designs anyway,” *Proceedings of Teaching, Learning and Assessment in Databases (TLAD),* 2003.

[6] C.J. Date, *An Introduction to Database Systems, 7th Edition*. Addison-Wesley, 2000 p 328.

[7] J. Martin, Application Development Without Programmers. Prentice-Hall, 1981.

[8] D.M. Bowers, C. Newton, K. Ireland, K. Waugh, “A Classification of Object-Relational Impedance Mismatch,” *DBKDA '09 Proceedings of the 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications,* pp. 36-43, 2009.

[9] B. Byrne, M. Garvey, “User Defined Types and Nested Tables in Object Relational Databases”, *UK Academy for Information Systems (UKAIS) Annual Conference, University of Gloucester,* 2006.

[10] P. Atzeni et al., “The Relational Model is Dead, SQL is Dead, and I Don’t Feel So Good Myself,” *SIGMOD Record,* vol. 42 no. 2, pp. 64-68, 2013.