Journal homepage: www.sabauni.net/ojs

Saba Journal of Information Technology and Networking (SJITN)

Article

Using XML as NoSQL Database in .NET and Comparing it with SQL Database

Dr.Abdualmajed Al-Khulaidi

Sana'a University, Yemen

Article info

Article history:

Accepted Jan, 2016

Keywords:

NoSQL Database,
SQL Database,
XML,
HTML .

Abstract

This scientific paper intends to explore the use of XML as a database as being a NoSQL type and the functions it carries out. Some of its important functions are to store and transfer data. It also tackles another point as well; how to use XML database in .NET by using C# language. To clarify such points, examples are provided to show the processes of adding, reading, deleting, updating and searching for data by using XML in C#. The paper further draws a comparison between NoSQL and SQL databases. The comparison is conducted to incorporate a number of points: database structuring, type of data they store, querying, scaling, support, reliability, and need for storing complex data and querying for it.

* Corresponding author: Dr. Abdualmajed Al-Khulaidi
E-mail: alkhulaidi@mail.ru

1. Introduction

Before initiating this paper, it is important to introduce the types of databases. There are several types among others are: network database, hierarchical database, relational database, object-oriented database …etc. Most famous of which is the relational database as its fame is similar to that of Oracle, SQL Server, MySQL, Access. More importantly is that relational databases store or organize data in tables, which are linked in the form of relational models. Another set of database, OOP Database, relies on having objects in their structure [1,2,3,4,5,6]. One more type is Document Store or NoSQL database in which data is stored in the form of encoded files in XML or JSON format or in the form of Word or PDF format.

XML is not a programming language, yet it is designed to transmit and store data. It is a family of Extensible Markup Language which also includes the well-known HTML language. This language is a subset of the Standard Generalized Markup Language SGML which first appeared in 1960, that is, about 30 years before the World Wide Web came to function. It assists in marking up and coordinating as well as organizing exchanged documents and emails via the Net. The family of marking up languages is featured by using tags in the form of parenthesizes such as < >, and having a tree-like structure. While it lacks any instructions or processes, its function is limited to mark up contents with certain codes that are understood by browsing software. It is known that HTML is a descriptive language used to display data on a web page. While some think that XML language is different from HTML, it is possible to argue that what links both languages is that they are subsets of the same language. It is well known that HTML uses a very limited number of tags which enables browsers to interpret these tags or codes automatically in order to coordinate the proper display of data on a web page.

XML files do often offer benefits in terms of not needing mysql databases or other similar databases. The benefit is incorporated in making a file writable and enabling it to store information that is transmitted to it. Via using XML files, it is possible to save and display data in an HTML page without the need for any programming languages. An example of XML which allows the saving of information is a CSS file permitting the saving of styles.

1.1 XML Tree Structure

XML documents are formed as element trees.
An XML tree starts at a root element and branches from the root to child elements [7,8].
All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

1.2 An Example XML Document

```
<?xml version=”1.0” encoding=”UTF-8”?>
<bookstore>
  <book category=”cooking”>
    <title lang=”en”>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category=”children”>
    <title lang=”en”>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category=”web”>
    <title lang=”en”>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

1.3 XML Functions

XML functions could be classified into three types [9,10]:

A benefit could be obtained from XML within one website:

1. *This benefit is often two faced.*

The use of XML files as databases:

This could be through either through the use of

XML documents to store data or the conversion of a group of documents into a database by using one of the following applications.

2. Making use of XML files when exchanging data among two or more websites

We often hear and realize the great cooperation in information and data sharing among various websites. Such a feature allows us to easily post our favored links in Delicious on our Facebook and Twitter pages and other social networking websites. The users of Good reads website also display their updates on Twitter and Facebook. A sufficient example for data/ information sharing among websites is Friend Feed which displays summaries of most social networks. All of that is being done through XML, that is, either through RSS files or other XML files.

3. Making use of XML when sharing data on computers or other devices and various applications

Let's take Twitter as an example. I can log in to my Twitter account and read my friends' updates on Twitter through:
www.twitter.com, my mobile phone, Desk/Laptop Computer. All of these means deal with the same database, and the same updates are displayed. Yet, each means/ device is programmed by a different language and functions differently. All of this is done by the benefits and features offered by XML.
2. Using XML Files as NoSQL Databases in .net
NoSQL is a new model or type of database management system. It follows a different model than the older and traditional models which follow the model of linked or tied tables (relational databases). The most outstanding difference between these two models is the use of tables. Unlike relational database, NoSQL does not consider tables as the foundation element for constructing database. For such a reason, NoSQL is used as an alternative or substitute for SQL language in terms of dealing with data.
XML is one model of a NoSQL database and it is not a programming language. Yet, we could state that it takes the form of textual writings, composed with known and specific rules. It is very similar to HTML in terms of writing its codes by using tags. Unlike HTML, XML is not limited in terms of the use of words; a designer could use any word to create the root and nodes and sub-nodes. In addition, XML is used to store data regardless of how they are displayed. An XML file is of a small size compared to database files and is easily accessed and dealt with. Some of its functions may include the storing and sharing of data among databases. A table could be exported to an XML file while an XML file could be imported. In addition, an XML file is composed of a root, nodes and sub-nodes. The attribute of a node could be specified, a matter which is known to be "Attribute." Such a linguistic item represents the characteristic of a node or (an element). In .net (dot net), XML is dealt with through LINQ. LINQ is from Microsoft to unite the means of using data irrespective of data sources. The Net Framework library has offered a means to easily deal with such files through XML or XML. LINQ libraries. These two libraries provide objects and functions/subroutines enabling us to deal with XML through creating a file, or updating the file or even deleting or performing search in it.
Let's take a look at the following example, illustrating how to use XML in C#.

Required Libraries:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using System.Xml;
using System.Xml.Linq;
```

We design a Class by the name of Emp as follows:

```
class Emp
    {
      public int ID { get; set; }
      public string Name { get; set; }
      public string Gender { get; set; }
      public int Age { get; set; }
      public string Imgstring { get; set; }
```

```
        }
```

General Variables:

```
private XElement _xel,_empinfo, _id, _name, _age,_image;
private readonly XmlDocument _doc = new XmlDocument();
 private XmlNode _node;
private readonly List<Emp>  _emp = new List<Emp>();
 private int _pos;
 private const string Xpath = @"D:\emp.xml";
  private string _imgstr = "";
```

Function Filling Field

```
private   void Fill(int pos)
{
id_txt.Text = _emp[pos].ID.ToString(CultureInfo.InvariantCulture);
 name_txt.Text = _emp[pos].Name;
age_txt.Text = _emp[pos].Age.ToString(CultureInfo.InvariantCulture);
pic.Image = StringToImage(_emp[pos].Imgstring);
("Male             if (_emp[pos].Gender == "
Male_rdbtn.Checked = true;
            else
                Female_rdbtn.Checked = true;
        }
```

A Function for Converting an Image into a Textual Series to be Easily Saved in an XML File:

```
private static string ImageToString(Image img)
        {
    var m = new MemoryStream();
  img.Save(m, System.Drawing.Imaging.ImageFormat.Jpeg);
  return Convert.ToBase64String(m.ToArray());
        }
```

A Function for retrieving an image from its textual format:

```
private static Image StringToImage(string imgstring)
        {
var imgbytes = Convert.FromBase64String(imgstring);
 var m = new MemoryStream(imgbytes);
      return Image.FromStream(m);
        }
```

A Code for Creating an XML File and Adding a Node to the File:

```
private void write_btn_Click(object sender, EventArgs e)
{
var found = false;
if (File.Exists(Xpath))
{
_xel = XElement.Load(Xpath);
foreach (var t in _emp)
{
found = t.ID.ToString(CultureInfo.InvariantCulture) == id_txt.Text;
}
}
else
_xel = new XElement("Info");
if (!found)
{
var gender = Male_rdbtn.Checked ? "Male" : "Female";
_empinfo = new XElement("EmpInfo");
_empinfo.SetAttributeValue("EmpInfo", "EmployeeInformation");
_id = new XElement("EId", id_txt.Text);
_name = new XElement("EName", name_txt.Text);
_name.SetAttributeValue("Gender", gender);
_age = new XElement("EAge", age_txt.Text);
_image = new XElement("EImage", _imgstr);
_empinfo.Add(_id, _name, _age, _image);
_xel.Add(_empinfo);
_xel.Save(Xpath);
_pos = _xel.Nodes().Count() - 1;
read_btn_Click(sender, e);
}
else
{
MessageBox.Show(@"This sequence is present ,choose another sequence ", "");
}
}
```

At this point, it is important to check if the file is available or not. If so, we read it only. Then, we make sure of the non-recurrence of the sequence for more than once and add the element to the file. If there is no file, then we create one file by creating its major root with the name Info and a sub-node with the name EmpInfo to include the following nodes: (ID, Name, Age, Image). While Name node has Gender as an attribute the EmpInfo node has an attribute as EmpInfo.

There are, for sure, various ways used in adding an element to an XML file. We have chosen XElement to be included in the field of XML.Linq names. This is the easiest way to create a file and add elements and the attributes of a specific element of those elements.

Content Reading Code:

```
private void read_btn_Click(object sender, EventArgs e)
{
srch_btn.Enabled = true;
_emp.Clear();
if (!File.Exists(Xpath)) return;
_doc.Load(Xpath);
var xmlElement = _doc["Info"];
if (xmlElement != null)
_node = xmlElement["EmpInfo"];
while (_node != null)
{
var element = _node["EId"];
if (element != null)
{
var xmlElement1 = _node["EName"];
if (xmlElement1 != null)
{
var element1 = _node["EAge"];
if (element1 != null)
{
var xmlElement2 = _node["EImage"];
if (xmlElement2 != null)
_emp.Add(new Emp
{
ID = int.Parse(element.InnerText),
Name = xmlElement1.InnerText,
Gender = xmlElement1.Attributes["Gender"].InnerText,
Age = int.Parse(element1.InnerText),
Imgstring= xmlElement2.InnerText ;
}
}
}
}
_node = _node.NextSibling;
}
Fill(_pos);        }
```

There are also various ways to read a file. Here, we use two objects, being XmlDocumnet and XmlNode, both of which are within the field of XML names. To do so, we first apply the main node and the node following it to the XmlNode variable object. That is, EmpInfo node is applied to the variable Node through which we pass to the node which it contains and extract its contents. Then, we move to each node by using NextSibling attribute, that is, the next node. Yet, reading a node content is done through InnerText attribute after mentioning the name of the node. To reach the attribute of a specific node, we first mention the name of the node followed by the name of the attribute, and then InnerText. As you may see, it is a very simple to reach the content of any node in a file.

Content Update Code:

```
private void update_btn_Click(object sender, EventArgs e)
{
var xmlElement = _doc["Info"];
if (xmlElement != null)
_node = xmlElement["EmpInfo"];
while (_node != null)
{
if (_node["EId"] != null && _node["EId"].InnerText == id_txt.Text)
{
_emp[_pos].ID = int.Parse(_node["EId"].InnerText = id_txt.Text);
_emp[_pos].Name = _node["EName"].InnerText = name_txt.Text;
if (Male_rdbtn.Checked)
{
_emp[_pos].Gender = _node["EName"].Attributes["Gender"].InnerText = Male_rdbtn.Text;
}
else
{
_emp[_pos].Gender = _node["EName"].Attributes["Gender"].InnerText = Female_rdbtn.Text;
}
_emp[_pos].Age = int.Parse(_node["EAge"].InnerText = age_txt.Text);
_emp[_pos].Imgstring = _node["EImage"].InnerText = ImageToString(pic.Image);
}
_node = _node.NextSibling;
}
_doc.Save(Xpath);
read_btn_Click(sender, e);
}
```

Deletion Code:

```
private void delete_btn_Click(object sender, EventArgs e)
{
var xmlElement = _doc["Info"];
if (xmlElement != null)
_node = xmlElement["EmpInfo"];
while (_node != null)
{
var element = _node["EName"];
if (element != null && element.InnerText == name_txt.Text)
{
if (_node.ParentNode != null) _node.ParentNode.RemoveChild(_node);
_emp.RemoveAt(int.Parse(id_txt.Text) - 1);
}
_node = _node.NextSibling;
}
_doc.Save(Xpath);
_pos--;
if (_pos < 0)
_pos = 0;
Fill(_pos);
}
```

We delete a specific node according to the name by using this line:
node.ParentNode.RemoveChild(node);

The above line deletes EmpInfo node which belongs to the main node Info. When Empinfo node is deleted, all other embedded nodes will be deleted including their contents or elements.

The Search Code:

```
private void srch_btn_Click(object sender, EventArgs e)
{
if (!File.Exists(Xpath)) return;
_doc.Load(Xpath);
var xmlElement = _doc["Info"];
if (xmlElement != null) _node = xmlElement["EmpInfo"];
while (_node != null)
{
if (_node["EName"].InnerText == Srch_txt.Text.Trim())
{
id_txt.Text = _node["EId"].InnerText;
name_txt.Text = _node["EName"].InnerText;
string gender = _node["EName"].Attributes["Gender"].InnerText;
age_txt.Text = _node["EAge"].InnerText;
pic.Image = StringToImage(_node["EImage"].InnerText);
_pos = int.Parse(id_txt.Text) - 1;
if (gender == "ذكر")
Male_rdbtn.Checked = true;
else
Female_rdbtn.Checked = true;
}
_node = _node.NextSibling;
}}
```

Reporting Code:

```
private void button2_Click(object sender, EventArgs e)
{
var rf = new reportform();
rf.rv.LocalReport.EnableExternalImages = true;
var ds = new Microsoft.Reporting.WinForms.ReportDataSource("emp", _emp);
var param = new List<Microsoft.Reporting.WinForms.ReportParameter>
{
new Microsoft.Reporting.WinForms.ReportParameter("id",
_emp[_pos].ID.ToString(CultureInfo.InvariantCulture)),
new Microsoft.Reporting.WinForms.ReportParameter("name", _emp[_pos].Name),
new Microsoft.Reporting.WinForms.ReportParameter("age",
_emp[_pos].Age.ToString(CultureInfo.InvariantCulture))
};
var im = StringToImage(_emp[_pos].Imgstring);
const string temp = "c:\\Pics\\temp.jpg";
im.Save(temp);
param.Add(new Microsoft.Reporting.WinForms.
ReportParameter("image", "file:///c:\\Pics\\temp.
jpg"));
rf.rv.LocalReport.DataSources.Add(ds);
rf.rv.LocalReport.SetParameters(param);
rf.ShowDialog();
File.Delete(temp);        }
```

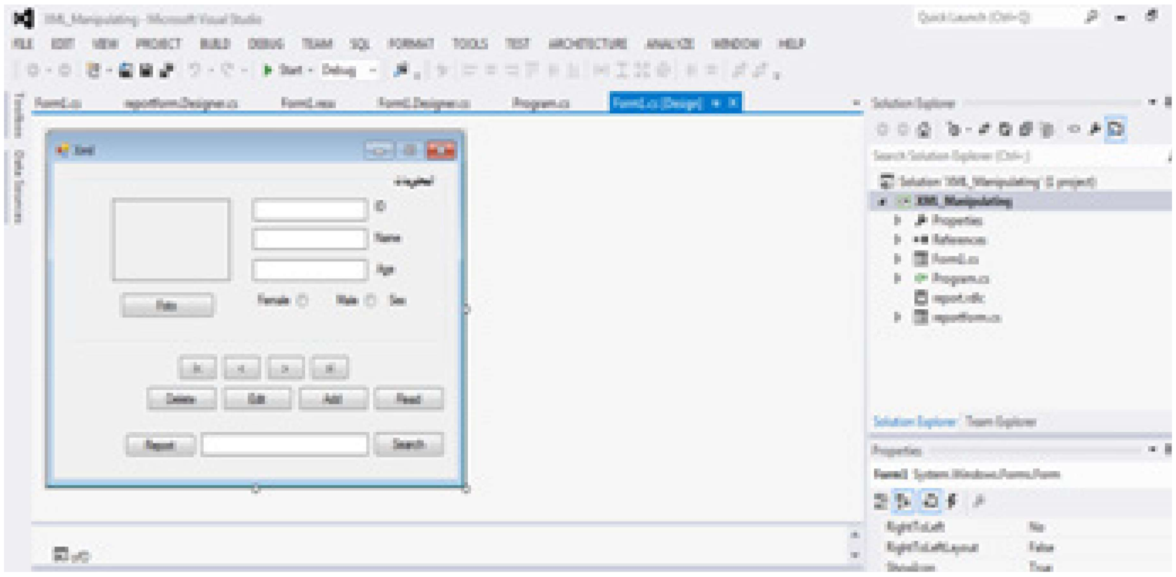
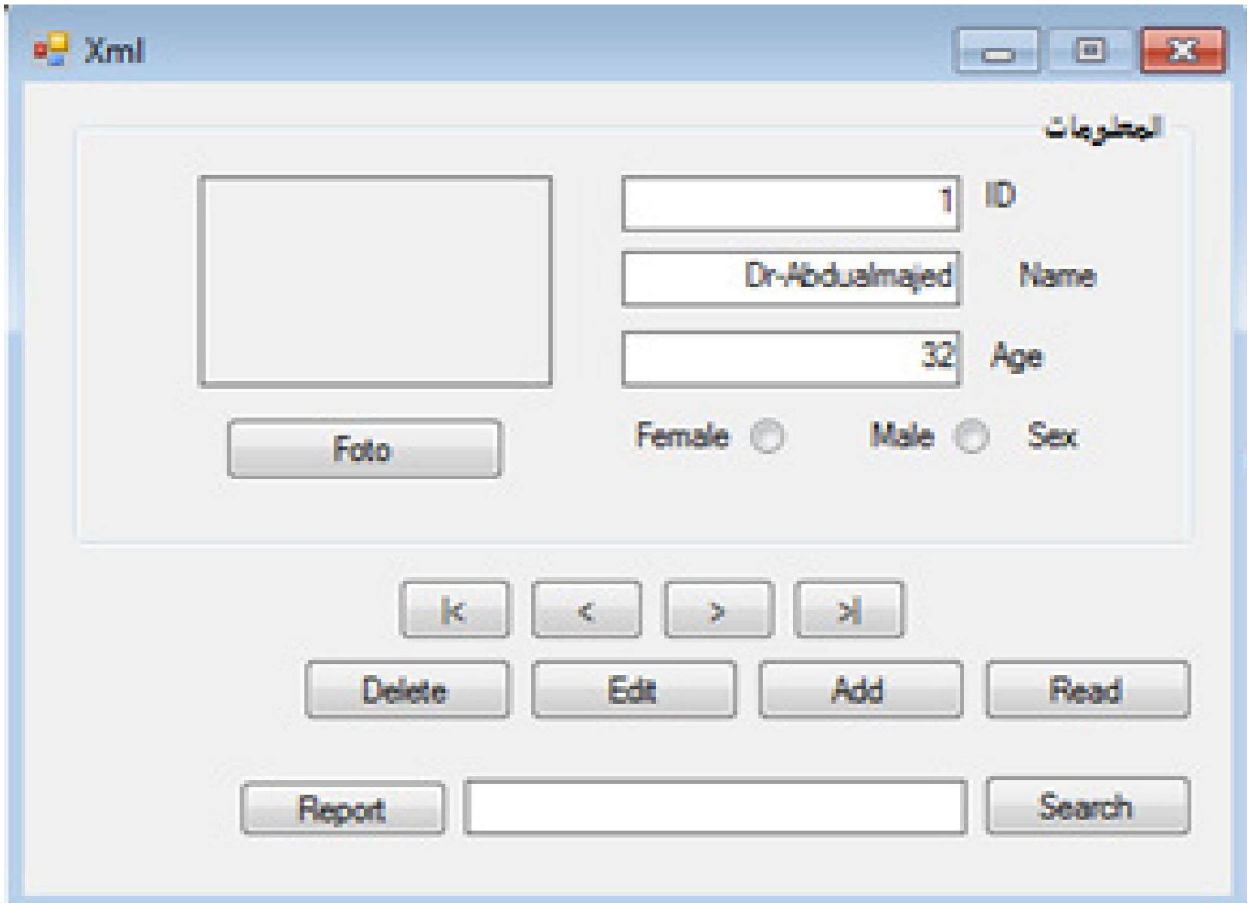

XML_Manipulating - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL FORMAT TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Form1.cs reportform.Designer.cs Form1.resx Form1.Designer.cs Program.cs Form1.cs [Design]
Solution Explorer
Solution 'XML_Manipulating' (1 project)
XML_Manipulating
Properties
References
Form1.cs
Program.cs
report.rdlc
reportform.cs
Properties
Form1 System.Windows.Forms.Form
RightToLeft No
RightToLeftLayout False

Fig. 1: Illustrating a program in C# by using (NoSQL) XML database



المعلومات
ID 1
Name Dr-Abdualmajed
Age 32
Female Male Sex

Fig. 2: Implementation of the Program in C# by using (NoSQL) XML database

Just like XML database, an NoSQL database could deal with Asp.net in terms of designing and developing dynamic websites instead of dealing with SQL database such as Oracle, Mysql, and Mssql server.

4.Comparing NoSQL Database with SQL Database

Before drawing a comparison between NoSQL databases and SQL databases, let's take a look at the different database management systems.

Fig. 3: Showing the Way of Comparing NoSQL and SQL Databases

4.1 Relational Database Management Systems (SQL)

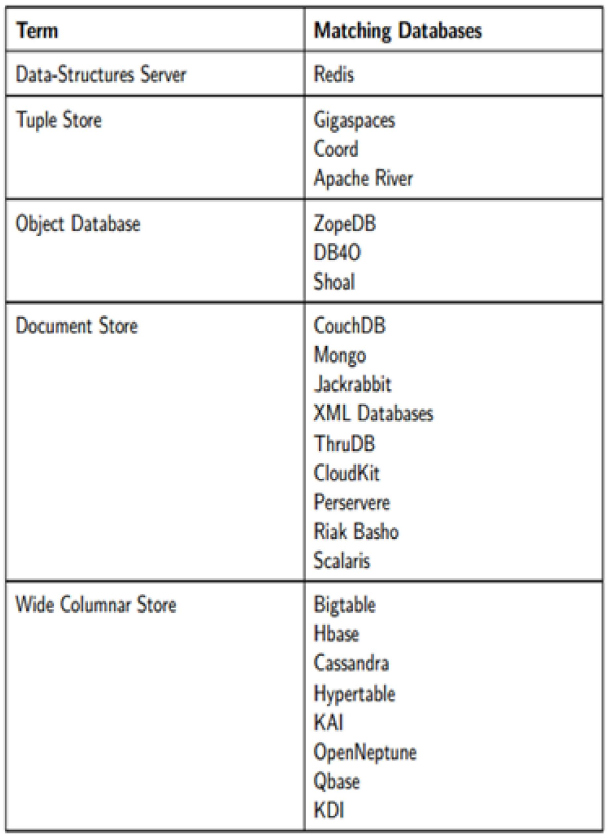
Relational Database Systems took its name after the model it is based on -The Relational Model, which was discussed earlier. These systems are and will remain for quite some time the best option to keep data reliable and safe. Not only that, they are also efficient. Relational database management systems require defined and clearly set schemas.

These schemas are much like tables; columns to contain a certain amount of information and to present the type of information in each record in addition to rows presenting the inputs.
Some of the most common relational database management systems include the followings [11]:
• SQLite: A very powerful and embedded relational database management system.
• MySQL: The most popular and commonly used RDBMS.
•PostgreSQL: The most advanced, SQL-compliant and open-source objective-RDBMS.

4.2 NoSQL Database Systems

NoSQL database systems do not come with a model as the one used (or needed) by structured relational solutions. There are many applications; each application operates very differently and serves specific needs. Either these schema-less solutions allow an unlimited formation of inputs or entries, rather take, a very simple form but extremely efficient for operating as useful key based value stores[12,13].

NoSQL databases do not have a common way to query the data (i.e. similar to SQL of relational databases) and each solution provides its own query system.
Examples of NoSQL databases are Jackrabbit Mongodb, XML, Riak, CouchDB, and Cassandra.



Term	Matching Databases
Data-Structures Server	Redis
Tuple Store	Gigaspaces Coord Apache River
Object Database	ZopeDB DB4O Shoal
Document Store	CouchDB Mongo Jackrabbit XML Databases ThruDB CloudKit Perservere Riak Basho Scalaris
Wide Columnar Store	Bigtable Hbase Cassandra Hypertable KAI OpenNeptune Qbase KDI

Fig. 4: Classifications NoSQL Database

- MongoDB: Cross-platform document-oriented database system that eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas making the integration of data in certain types of applications faster and easier [10].
- Cassandra: Highly scalable, high performance distributed database designed to handle large amounts of data across many commodity Serv-

ers, and that provide high availability with no single point of failure [14].

- XML database is a data persistence software system that allows data to be specified, and sometimes stored, in XML format. These data can be queried, transformed, exported and returned to a calling system [3]. XML databases are a flavor of document-oriented databases which are in turn a category of NoSQL database (meaning Not (only) SQL).

4.3 A Comparison of SQL and No-SQL Database Management Systems

In order to reach a simpler, understandable conclusion, let's analyze the differences between both SQL and No-SQL database management systems:

4.3.1 Structure and Type of Data Being Kept:

SQL/Relational databases require a structure with defined attributes to maintain or keep the data, unlike NoSQL databases which usually allow free-flow operations.

4.3.2 Querying:

Regardless of their licenses, all relational databases apply the SQL standard to a certain degree and thus, can be queried using the Structured Query Language (SQL). NoSQL databases, on the other hand, do not apply a unique way to operate the data they manage.

4.3.3 Scaling:

Both solutions are easy to scale vertically (i.e. by increasing a system's resources). However, being more modern (and simpler) applications, NoSQL solutions usually offer a much easier means to scale horizontally (i.e. by creating a cluster of multiple devices or machines).

4.3.4 Reliability:

When it comes to data reliability and safe guarantee of performed transactions, SQL databases are still the best option

4.3.5 Support:

Relational database management systems have a long history of strong performance and application. They are extremely common and their support is very easy to find both free of charge and/or paid. If an issue or problem arises, it is therefore much easier to solve such a problem than in the case of recently-popular NoSQL databases - especially if the said solution under focus is complex in nature (e.g. MongoDB).

4.3.6 Complex data keeping and querying needs:

By nature, relational databases are the ultimate solution for complex querying and data keeping needs. They are much more efficient and definitely excel in this domain.

4. 4The primary differences between SQL and NoSQL

4. 4.1SQL Tables and NoSQL Documents

SQL databases provide a store of related data tables. For example, if you run an Online book store, book information can be added to a table named book:

ISBN	Title	author	format	price
9780992461225	JavaScript: Novice to Ninja	D a r r e n Jones	ebook	29.00
9780994182654	Jump Start Git	Shaumik-Daityari	ebook	29.00

Every row is a different book record, the design is rigid; you cannot use the same table to store different information or insert a string where an integer is expected.

NoSQL databases store JSON-like field-value pair documents, e.g.

```
{
  ISBN: 9780992461225,
  title: “JavaScript: Novice to Ninja”,
  author: “Darren Jones”,
  format: “ebook”,
  price: 29.00
}
```

Similar documents can be stored in a collection, which is analogous to an SQL table. However, you can store any data you like in any document;

the NoSQL database won’t complain. For example:

```
{
 ISBN: 9780992461225,
 title: “JavaScript: Novice to Ninja”,
 author: “Darren Jones”,
 year: 2014,
 format: “ebook”,
 price: 29.00,
 description: “Learn JavaScript from scratch!”,
 rating: “5/5”,
 review: [
  { name: “A Reader”, text: “The best JavaScript
book I’ve ever read.” },
   { name: “JS Expert”, text: “Recommended to
novice and expert developers alike.” }
}
```

SQL tables create a strict data template, so it’s difficult to make mistakes. NoSQL is more flexible and forgiving, but being able to store any data anywhere which can lead to consistency issues.

4. 4.2SQL Schema and NoSQL Schema-less

In an SQL database, it’s impossible to add data until you define tables and field types in what’s referred to as a schema. The schema optionally contains other information, such as —

• **Primary keys** — unique identifiers such as the ISBN which apply to a single record

• **Indexes** — commonly queried fields indexed to aid quick searching

• **Relationships** — logical links between data fields

• **Functionality** such as triggers and stored procedures.

Your data schema must be designed and implemented before any business logic can be developed to manipulate data. It is possible to make updates later, but large changes can be complicated.

In a NoSQL database, data can be added anywhere, at any time, there’s no need to specify a document design or even a collection up-front, for example, in MongoDB the following statement will create a new document in a new book collection if it’s not been previously created:

```
db.book.insert(
 ISBN: 9780994182654,
 title: “Jump Start Git”,
 author: “ShaumikDaityari”,
 format: “ebook”,
 price: 29.00
);
```

(MongoDB will automatically add a unique _id value to each document in a collection. You may still want to define indexes, but that can be done later if necessary.)

A NoSQL database may be more suited to projects where the initial data requirements are difficult to ascertain. That said, don’t mistake difficulty for laziness: neglecting to design a good data store at project commencement will lead to problems later.

4.4.3 SQL Normalization and NoSQL Denormalization

Presume we want to add publisher information to our book store database, a single publisher could offer more than one title so, in an SQL database, we create a new publisher table:

id	name	country	Email
SP001	SitePoint	Australia	feedback@sitepoint.com

We can then add a publisher_id field to our book table, which references records by publisher.id:

ISBN	Title	author	format	price	publisher_id
9780992461225	JavaScript: Novice to Ninja	Darren Jones	ebook	29.00	SP001
9780994182654	Jump Start Git	Shaumik Daityari	ebook	29.00	SP001

This minimizes data redundancy; we’re not repeating the publisher information for every book only the reference to it. This technique is known as normalization, and has practical benefits. We can update a single publisher without changing book data.

We can use normalization techniques in NoSQL. Documents in the book collection —

```
{
 ISBN: 9780992461225,
 title: "JavaScript: Novice to Ninja",
 author: "Darren Jones",
 format: "ebook",
 price: 29.00,
 publisher_id: "SP001"
}
```

— reference a document in a publisher collection:

```
{
 id: "SP001"
 name: "SitePoint",
 country: "Australia",
 email: "feedback@sitepoint.com"
}
```

However, this is not always practical, for reasons that will become evident below. We may opt to denormalize our document and repeat publisher information for every book:

```
{ISBN: 9780992461225,
 title: "JavaScript: Novice to Ninja",
 author: "Darren Jones",
 format: "ebook",
 price: 29.00,
 publisher: {
  name: "SitePoint",
  country: "Australia",
email: "feedback@sitepoint.com"}}
```

This leads to faster queries, but updating the publisher information in multiple records will be significantly slower.

4. 4.4 SQL Relational JOIN and NoSQL

SQL queries offer a powerful JOIN clause. We can obtain related data in multiple tables using a single SQL statement. For example:

```
SELECT book.title, book.author, publisher.name
FROM book
LEFT JOIN book.publisher_id ON publisher.id;
```

This returns all book titles, authors and associated publisher names (presuming one has been set). NoSQL has no equivalent of JOIN, and this can shock those with SQL experience. If we used normalized collections as described above, we would need to fetch all book documents, retrieve all associated publisher documents, and manually link the two in our program logic, this is one reason denormalization is often essential.

4. 4.5 SQL and NoSQL Data Integrity

Most SQL databases allow you to enforce data integrity rules using foreign key constraints (unless you're still using the older, defunct MyISAM storage engine in MySQL). Our book store could
•ensure all books have a valid publisher_id code that matches one entry in the publisher table, and
•not permit publishers to be removed if one or more books are assigned to them.

the schema enforces these rules for the database to follow, it's impossible for developers or users to add, edit or remove records, which could result in invalid data or orphan records.

The same data integrity options are not available in NoSQL databases; you can store what you want regardless of any other documents. Ideally, a single document will be the sole source of all information about an item.

4. 4.6 SQL and NoSQL Transactions

In SQL databases, two or more updates can be executed in a transaction — an all-or-nothing wrapper that guarantees success or failure. For example, presume our book store contained order and stock tables, when a book is ordered we add a record to the orderTable and decrement the stock count in the stock table. If we execute those two updates individually, one could succeed and the other fail — thus leaving our figures out of sync, placing the same updates within a transaction ensures either both succeed or both fail.

In a NoSQL database, modification of a single document is atomic, in other words, if you're updating three values within a document, either all three are updated successfully or it remains unchanged. However, there's no transaction equivalent for updates to multiple documents, there are transaction-like options, but -at the time of writing this article- these must be manually processed in your code.

4. 4.7 SQL vs NoSQL CRUD Syntax

Creating, reading updating and deleting data is the basis of all database systems. In essence —
•SQL is a lightweight declarative language. It's

deceptively powerful, and has become an international standard, although most systems implement subtly different syntaxes.

•NoSQL databases use JavaScripty-looking queries with JSON-like arguments! Basic operations are simple, but nested JSON can become increasingly convoluted for more complex queries.

Table 3 : Comparison between SQL and NoSQL

SQL	NoSQL
insert a new book record	
INSERT INTO book (\`ISBN\`, \`title\`, \`author\`) VALUES ('9780992461256', 'Full Stack JavaScript','Colin Ihrig& Adam Bretz');	db.book.insert({ ISBN: "9780992461256", title: "Full Stack JavaScript", author: "Colin Ihrig& Adam Bretz"});
update a book record	
UPDATE book SET price = 19.99 WHERE ISBN = '9780992461256'	db.book.update({ ISBN: 9780992461256' },{ $set: { price: 19.99 });
return all book titles over $10	
SELECT title FROM book WHERE price > 10;	db.book.find({ price: { &gt;: 10 } }, { _id: 0, title: 1 }); The second JSON object is known as a projection: it sets which fields are returned (_id is returned by default so it needs to be unset).
count the number of SitePoint books	
SELECT COUNT(1) FROM book WHERE publisher_id = 'SP001';	db.book.count({"publisher.name": "SitePoint"}); This presumes denormalized documents are used.
return the number of book format types	
SELECT format, COUNT(1) AS \`total\` FROM book GROUP BY format;	db.book.aggregate([{ $group: { _id: "$format", total: { $sum: 1 } }});This is known as aggregation: a new set of documents is computed from an original set.
delete all SitePoint books	
DELETE FROM book WHERE publisher_id = 'SP001'; Alternatively, it's possible to delete the publisher record and have this cascade to associated book records if foreign keys are specified appropriately.	db.book.remove({"publisher.name": "SitePoint"});

The second JSON object is known as a projection: it sets which fields are returned (_id is returned by default so it needs to be unset).
count the number of SitePoint books
SELECT COUNT(1) FROM book
WHERE publisher_id = 'SP001'; db.book.count({
"publisher.name": "SitePoint"});
This presumes denormalized documents are used.
return the number of book format types
SELECT format, COUNT(1) AS \`total\`
FROM book
GROUP BY format; db.book.aggregate([
{ $group:
{ _id: "$format",
total: { $sum: 1 }
}});This is known as aggregation: a new set of documents is computed from an original set.
delete all SitePoint books
DELETE FROM book WHERE publisher_id = 'SP001';
Alternatively, it's possible to delete the publisher record and have this cascade to associated book records if foreign keys are specified appropriately. db.book.remove({"publisher.name": "SitePoint"});

4. 5 SQL and NoSQL: High-Level Differences

•SQL databases are primarily called as Relational Databases (RDBMS); whereas NoSQL database are primarily called as non-relational or distributed database.

•SQL databases are table based databases where as NoSQL databases are document based, key-value pairs, graph databases or wide-column stores, this means that SQL databases represent data in form of tables which consists of n number of rows of data whereas NoSQL databases are the collection of key-value pair, documents, graph databases or wide-column stores which do not have standard schema definitions which it needs to adhered to.

•SQL databases have predefined schema whereas NoSQL databases have dynamic schema for unstructured data.

•SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable; SQL databases are scaled by increasing the horse-power of the hardware. NoSQL databases are scaled by increasing the databases servers in the pool of resources to reduce the load.

•SQL databases use SQL (structured query language) for defining and manipulating the data which is very powerful; In NoSQL databases queries are focused on collection of documents, sometimes it is also called UnQL (Unstructured Query Language). The syntax of using UnQL varies from database to database.

•SQL database examples: MySql, Oracle, SQLite, Postgres and MS-SQL. NoSQL database examples: MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb.

•**For complex queries:** SQL databases are a good fit for the complex query intensive environment whereas NoSQL databases are not good fit for complex queries. On a high-level, NoSQL don't have standard interfaces to perform complex queries, and the queries themselves in NoSQL are not as powerful as SQL query language.

•**For the type of data to be stored:** SQL databases are not best fit for hierarchical data storage, where NoSQL database fits better for the hierarchical data storage as it follows the key-value pair way of storing data similar to JSON data. NoSQL database are highly preferred for large data set (i.e for big data). Hbase is an example for this purpose.

•**For scalability:** In most typical situations, SQL databases are vertically scalable. You can manage increasing load by increasing the CPU, RAM, SSD, etc, on a single server. On the other hand, NoSQL databases are horizontally scalable. You can just add few more servers easily in your NoSQL database infrastructure to handle the large traffic.

•**For high transactional based application:** SQL databases are best fit for heavy duty transactional type applications, as it is more stable and promises the atomicity as well as integrity of the data. While you can use NoSQL for transactions purpose, it is still not comparable and sable enough in high load and for complex transactional applications.

•**For support:** Excellent support are available for all SQL database from their vendors, there are also lot of independent consultations who can help you with SQL database for a very large scale deployments. For some NoSQL database you still have to rely on community support, and only limited outside experts are available for you to setup and deploy your large scale NoSQL deployments.

•**For properties:** SQL databases emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance)

•**For DB types:** On a high-level, we can classify SQL databases as either open-source or close-sourced from commercial vendors. NoSQL databases can be classified on the basis of way of storing data as graph databases, key-value store databases, document store databases, column store database and XML databases

5. Conclusion

It is safe to argue that NoSQL represents a very powerful and new means for data storage and retrieval in an optimized and smooth manner. Applying NoSQL is easier to deal with data than using SQL in this regard.

Criticizing any one of the SQL's will not help the aim of this work. If there is a buzz of NoSQL these days, it doesn't mean that is a silver bullet to all your needs. Both technologies (SQL and NoSQL) are best in what they do. It is up to a developer to make a better use of them depending on the situations and needs.

NoSQL databases, by using an unstructured (or structured) kind of approach, aim to eliminate the limitations of strict relations, and accordingly, offer many different ways to maintain and efficiently use data for specific usage cases (e.g. full-text document storage).

References

[1] XML Media Types, RFC 7303. Internet Engineering Task Force. July 2014.

[2] XML 1.0 Specification. World Wide Web Consortium. Retrieved 2010-08-22.

[3] XML and Semantic Web W3C Standards Timeline (PDF). 2012-02-04.

[4] Fennell, Philip (June 2013). "Extremes of XML". XML London 2013: 80–86.doi:10.14337/XMLLondon13.Fennell01. ISBN 978-0-9926471-0-0.

[5]M. Murata, D. Kohn, and C. Lilley (2009-09-24). "Internet Drafts: XML Media Types". Internet Engineering Task Force. Retrieved 2012-02-29.

[6] Extensible Markup Language (XML) 1.1 (Second Edition) . World Wide Web Consortium. Retrieved 2010-08-22.

[7] Pilgrim, "The history of draconian error handling in XML". Archived from the original on 2011-07-26. Retrieved 18 July 2013.

[8] Jon Bosak, "Closing Keynote, XML 2006". 2006.xmlconference.org. Archived from the original on 2007-07-11. Retrieved2009-07-31.

[9] T. Bray, C. Frankston, A. Malhotra, "Document Content Description for XML", http://www.w3.org/TR/NOTE-dcd.

[10] Obasanjo, D. (2013). Building scalable databases: Denormalization, the NoSQL movement and Digg. Retrieved July 15th.

[11] Software Engineering Radio Podcast, Episode 165: NoSQL and MongoDB with Dwight Merriman by Robert Blumen and Dwight Merriman: http://www.se-radio.net/2010/07/episode-165-nosql-and-mongodb-with-dwight-merriman/

[12] Heise SoftwareArchitekTOUR Podcast (German), Episode 22: NoSQL – Alternative zu relationalen Datenbanken by Markus Völter, Stefan Tilkov and Mathias Meyer: http://www.heise.de/developer/artikel/Episode-22-NoSQL-Alternative-zu-relationalen-Datenbanken- 1027769.htm.

[13] Zawodny, J. (2009). NoSQL: Distributed and Scalable Non-Relational Database Systems. Linux Magazine web portal, http://www.linux-mag.com/id/7579.

[14] RadioTux Binärgewitter Podcast (German), Episode 1: NoSQL by Dirk Deimeke, Marc Seeger, Sven Pfleiderer and Ingo Ebel: http://blog.radiotux.de/2011/01/09/binaergewitter-1-nosql