Abey Tedla Worku

Web Crawler and Information Management

Helsinki Metropolia University of Applied Sciences
Degree Bachelor of Engineering

Degree Programme - Media Engineering

Thesis

Date 12 May 2011

Author(s) Title Number of Pages	Abey Tedla Worku Web crawler and information management 44 pages + 3 appendices
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	Application Development
Instructor(s)	Ahti Ahde, Project Manager Kari Aaltonen, Senior Lecturer

This project was made for a company that provides a networking website for people who are interested in music. The objective of this project was to develop an application which collects and stores data from two distinct websites namely, http://www.muusikoiden.net and http://www.imperiumi.net. Both websites contain rich information about artists and their gig schedule, which was the main interest of the company. Another goal was to compile one data table that contains all the information which was gathered from the above mentioned sites.

The application was developed using a web crawler integrated with Document Object Model Interface, which enables different programming languages to access any websites by representing the components in a more structured form. This crawler was made to iterate for every page in the websites. After parsing, the stored data was analyzed by the application to detect if there were any name prefixes, and remove those prefixes on every occurrence. Finally, a complete table with all the required information was build.

Web crawler is a powerful programming application which can help for locating and accessing information from any web documents. And by combining the information it is possible to innovate or make useful analysis.

Keywords	Web crawler, interface, document object model, serialize, database management system, indexing, normalizing

Contents

1 Introduction

LoudRevolution is a company which made a networking site for music enthusiasts. It has three major user groups, fans, venues and artists or bands. Artists can register to this networking website and send multiple gig requests to the venues that they are interested to work with and at the same time the venues will have the privilege to check artists' profiles. LoudRevolution has a rich source of information in cooperating with facebook, MySpace and other networking sites. This yields to a great prediction about the upcoming concert or gig. For elaboration, let us examine a scenario which demonstrates what the service actually offers. An artist registers and creates a profile to this network and if the artist has a Facebook, MySpace, own website or other useful links it can also be added as substance to his or her profile. Then the artist sends a gig proposal to a couple of venues. The software analyzes the data and makes a rating. This rating can be useful for predicting the number of fans that can attend a gig, in short it tells us how famous an artist is in that specific region. These information came in handy, especially for venues because they can decide whether to accept the gig request or change some settings accordingly. Venues will get a chance to predict the amount of crowd they will have in that concert, and using this feature will help them to minimize the extra cost they might face if the use the traditional way of organizing a concert.

The objective of this project was to build a crawler that fetches data from two specific web sites, http://www.muusikoiden.net and http://www.imperiumi.net and save it on one common data table. These web sites provide rich information about music artists and their gig schedule. After collecting the desired data, a database was arranged accordingly to save the data. The next step was data purification, since we are dealing with tons of data. Also its sources are two different sites and it should be considered if there is a relation between each data entry. This relation could be same data entry or redundancy or it could be a data being registered in a different format. When considering artists, they may register their names in different forms. The purpose of this project was to create an application which collects, studies and merge the data to one data table without losing any information.

2 Web Crawler

2.1 What Is a Web Crawler

As a definition, a web crawler is a computer program that surfs through URLs (uniform resource locator) and uses a systematic way to view or even save data from the desired web site. Crawlers read information from web sites providing entries for search engine index. Web crawler works by indexing the required data that are scanned from the certain page. Indexing is a method that helps a program like crawlers or web surfers to find the specific or related information of a keyword. Web crawlers are also known as web spiders, web robots, automatic indexer or Bot. Almost all major search engines have similar application as web service. To mention some of the well-known web crawlers, Yahoo! Slurp, Bing Bot, Google Bot and Alta Vista can be included and they are the most powerful search engines up to now.

Almost everything on the web can be crawled. Most search engines keep their crawlers always running so that they can store web contents into a database. In crawling there are three basic steps. They first start browsing the page which is followed by indexing content of the site and finally they start to go through links. This process continues until they finish visiting all the links from the web page. [1]

Web crawlers are mostly used for search engines. Usually they start with a list of links to be visited which are called seeds, and as the crawler goes through these links it can detect all the hyperlinks on that specific web page. These hyperlinks are added to the list of URLs to be visited and they can be referred as crawler frontier. This can be done by indexing the HTML (Hyper Text Markup Language) attribute. *href* is an attribute that confirms a link is contained next to it. Using this method we can fetch all emails or links that are listed on a certain web page. [2]

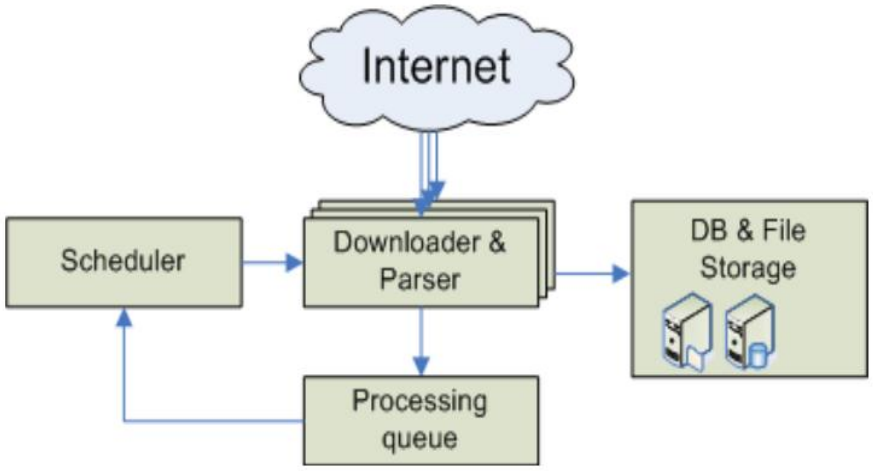


Figure 1. Standard Architecture for a web crawler [2]

Figure 1 shows how a web crawler operates. First it browses through net and collect links and these links will be pending until the scheduler calls for it. Each link will be loaded and their content will be parsed. Finally they will store the required information to the database. [2]

The other basic use of a web crawler, which is the back bone for this project, is to save information from a given web page by using Document Object Model. If we are interested in a specific data and analyze or manipulate the data, then it is recommended to implement this part of a web crawler service. The reason why using document object model is very important is that we can crawl any HTML web pages and we can access whichever data we desire. In general, it is a way of organizing pages as an object so that it can correlate with other programming languages and give the required output. The implementation will be discussed in detail in chapter 3.

2.2 History

In 1994 a student named Brian Pinkerton from University of Washington started developing a web crawler. Even though it is a web service, back then he was making it just as a simple desktop application. For the first time on 20 April 1994, the crawler was introduced to web and it was consisted of a database from 4000 different web pages. A few months later it had served over 1 million queries with a satisfying result. The quick success of web crawler and response of internet woke the attention of different organizations. On first of December 1994, the first two companies DealerNet and StarWave contributed money for further development of this web service.

Web crawler suits best for search engine software. Almost all search engines use a crawler to navigate the web by following links and downloading their contents. They send these contents to an indexer. These feature abled companies such as Google to have a massive collection of information about a web page. The indexed contents are arranged in an inverted file structure and data such as web page address will be saved in a database. When a user sends a query, the system processes by collecting relevant information and provides a close match back to the user. [3,89]

Google has one of the most powerful search engines in the whole world. These days Google crawlers browse and index already over trillions of web pages. The contents are stored to several servers in which by the end of 2009 it reached over 500,000 separate servers. Now Google serves web searches from all over the world each day and night, and handles over billions of queries, and gives a response for each request. [3,81]

2.3 Privacy and Legal Issue

There have been a couple of issues which were against web crawlers, because of their quick ability to retrieve data and the depth that they can dig through online documents. Crawlers can send multiple requests to servers in a second and they can also download large sized files. These processes hinder a server not to achieve the highest performance. In conclusion:

- Crawlers use a large bandwidth while they operate. They can also work for a long time and creating hectic on web pages.
- Crawlers take server resources and slow them down.
- Crawlers which are not well developed might have a chance even to crush a server.

Many individuals and web site owners have complained about having a privacy constraint on crawlers but the problem is that it is not easy to put a privacy restriction on a data which is already accessible to public. But still the website owners wanted to have some kind of agreement on how crawlers should not access data beyond a certain period and for prohibition of linking a personal data to other databases. Nevertheless, there is no such restriction that binds crawlers up to now.

As a solution web pages are recommended to use the robot rejection protocol which is known as robots.txt. It is a protocol which identifies and blocks crawlers, but it does not include a recommendation or any solution for applications which visit frequently to the same server. However, implementing this interval control is the best way of avoiding server overload. Now a days search engines have a "Crawl-delay" parameter in the robots.txt file which will slow the number of seconds on the program to delay between requests.

3 Document Object Model

3.1 Programming Interface

DOM interface helps to represent HTML, XHTML (Extensible Hypertext Markup Language) and XML (Extensible Markup Language) as an object including their properties and methods. It defines a structured way of accessing and manipulating HTML web pages. Since DOM (Document Object Model) is interface independent, it can be accessed by any programming language but for this project I used PHP.

This interface cannot function without a document to act on. The first step for a DOM application is turning the document into an object. Objects are self-contained bundles of data. There are a number of variables associated with the document which are called properties of the object. Objects can also execute instructions and manipulate document properties. These instruction sets are known as methods.

The DOM can be divided in four standards level, 0, 1, 2 and 3.

Level 0 is almost an extinct standard and does not exist as a recommendation of most documentations including W3C. It used to be applicable for Explorer and Netscape. [4;5,26]

Level 1 consists of two basic sub levels. The Core level 1, which is a low-level interface that can represent any structured document and it is recommended for XML documents representations. The HTML Level 1 provides additional features to Core Level. It uses higher level interface to represent web documents and provides a better way to view. Node, Attr, Element, and Text interfaces are among the interfaces introduced in Level 1. One should not forget that all the interfaces contain attributes and methods that help to interact with XML and HTML documents. The extension also enables to work on the previous level. Many of these web based features share widespread and consistent browser support. [4;5,26]

Level 2 which is mostly supported by Mozila has additional XML- and HTML-specific extensions. It contains six different specifications: Views, Events, Style, Traversal and Range (handy for doing browser-based YSIWYG editing), and the DOM 2 HTML. At this level, it starts to show a difference on how these standards are applied across browsers. Mozilla browsers such as Firefox have been updating their specifications along with W3C but Internet Explorer bases its specification on Level 2 which was applicable in 1999. Since then, IE has gone in a different direction and has not made any changes on the event handling structures. The following are the new features that Level 2 interface support. [4;5,26]

- It introduced very important methods such as getElementById which was not included on the previous Level.
- It gives a privilege for programs to dynamically access the document and even change the content. This feature could also be implemented to style sheets documents.
- It enables interfaces to handle events. This could be the most considerable upgrade from this Level. To mention some of the popular events, EventTarget, EventListener, Event, DocumentEvent, MouseEvent, MutationEvent are included. Unfortunately, this Level does not implement an interface for KeyBoard events.
- It provides the opportunity to navigate dynamically through the document. Identifying range of contents is also another interface provided by this Level. DOM range allows, creating new content, modifying, insertion, fragment and more features.

Level 3 adds additional extensions to the core and event handling. It contains five different specifications which are, DOM3 Core, Load and Save, Validation, Events, and XPath. Some of the new features introduced at this level are handling KeyBoard events, serializing a DOM document in to XML, addition of lots of properties and methods are worth to mention. Being a new version, only a few browsers support Level 3 specifications. [4;5,26]

3.2 Document Representation

DOM tree is a structured form for XML or HTML document. Each tag in these documents is represented as elements, and tags inside another tag are child elements in this tree structure. There are 21 types of nodes in DOM and an element is among one of them. Those familiar to HTML tags are elements, attributes, text, document and comment. White spaces between tags are also considered as a character of text node with IE being an exception, so when we use DOM objects we could get different results according to which browser we use. [5,26-28]

Let us take a simple HTML snippet to see how it is represented in the DOM tree.

```
<html>
        <head>
             <title>Title</title>
        </head>
              <body>
                <div class="div class"> text/content
                <p>text</p>
                <p> text</p>
                </div>
              </body>
</html>
```

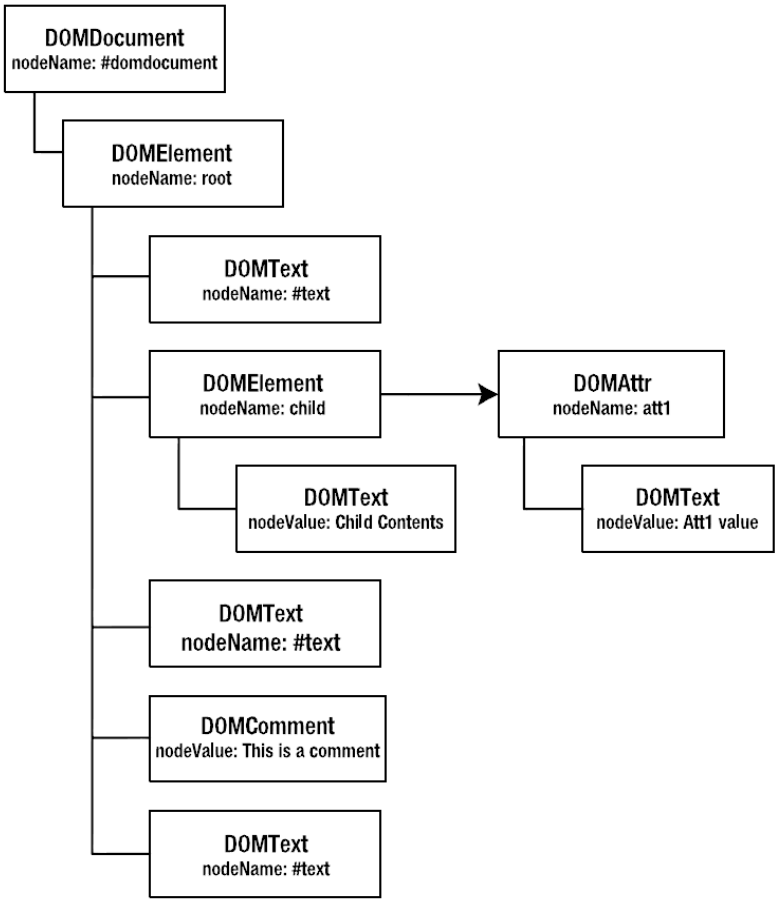


Figure 2. Structure diagram [5,183]

The above diagram shows only the five node types which are regularly used in HTML. The list of those not presented above include,

- DocumentType: Document type node
- Entity: Entity node
- EntityReference: Entity reference node
- Notation: Notation node
- ProcessingInstruction: PI node
- CDATASection: CDATA section node
- DocumentFragment: Document fragment node

Having seen how DOM views the HTML document, we can go deeper to node objects. Each node type is directly related to its own object but also we have a few more objects that do not inherit from a node. CharacterData, NodeList, NameNodeMap and DOMImplementation fall under this category. Using the above objects and their properties provided by the interface, we can access any node on the tree but this does not mean we have a full right to modify the document. [6,182]

Loading a document occupies some memory space, and a specific reference will be allocated automatically. The reference remains engaged until the object related to the document is destroyed and this can be done by calling unset() function. The objected document is saved in the memory with UTF-8 (Unicode transformation format), so to interact with the document tree we must use a compatible format. UTF-8 is an encoding format to a data. As we load or save a document, the DOM extension automatically encodes the data. Characters which do not support the UTF-8 format should be encoded using one of string conversion functions. [6,188]

3.3 Major Functionalities

To read a document using DOM, the first step would be to create an empty document class; this could be performed like any other PHP instances. We use the keyword *new* and then the class name DOMDocument. Once we create the class, we can save, load and also create other node types using the method that are provided with the classes. Creating a new class has two parameters, version and encode. For both parameters it is not compulsory to instantiate. The version parameter has its default, the value of which is 1.0, but for the encoding we have to specify the value otherwise it stays omitted. The DOM extensions have great features of detecting the encoding of the data and loading it appropriately using built-in methods but if a document has special characters, the data should be converted to UTF-8 manually. [6,189]

Loading a document follows after instantiating a new class. Using a key word *Load()* to load from a resource or LoadXML() to load write the XML inside the method itself, for example DOMDocument::LoadXML('<root><child> some text </child></root>'). In

the former cases we have to state the location to the document either by specifying a string that contains the documents or the URL for online documents. The Load() method has two parameters: the first one is the resource locator and the other one being a parser option. Loading without parser option will remove blank spaces between nodes because it sets to a default value 'LIBXML_NOBLANKS' and if we want to keep the white space we use 'preservewhitespace' property. These properties do not apply for HTML documents. Loading HTML has its own methods just like XML does, LoadHTML() and LoadHTMLFile(). Basically these methods load the HTML file to the DOM tree and then they are treated as any DOM document. To view the output we use saveXML() and saveHTML() methods depending on our document type. These methods do not need parameters especially saveHTML(). But saveXML() can accept one parameter which must be derived from the same document class, otherwise the output will be serialized and displayed as a string. [6,190]

The following code demonstrates how to instantiate and load an XML document:

```
$dom = new DOMDocument('1.0', 'ISO-8859-1');
        outputs <?xml version="1.0" encoding="ISO-8859-1"?>.
$dom->preserveWhiteSpace = FALSE;
        escape white spaces between node
$dom->load('xmldata.xml', LIBXML_NOBLANKS);
        If we set preserve white space value to TRUE we will override and omits
        the blank space
$output = $dom->saveXML($child);
        outputs only the nodes which belongs to the $child variable
```

After loading, the next step would be accessing each node. The three most important and more commonly used pieces of node information are, node type, node name and node value. Using DOMNode class and its derived properties, we can fetch all the above information, but if they are not provided by the nodes, the return value will be null as illustrated below: [6,195]

- Node type - to get the type of a node we use the property nodeType but first we load the document. [7,179]

$root = $dom->documentElement;

$root->nodeType;

The above code outputs an integer. Each integer corresponds to a node type, as illustrated in Table 1.

- NodeName and NodeValue have the same implementation in code aspect. We can use NodeName property either for a single element or for the whole document class. The first one returns the tag name of the node whereas the second one prints #Document. Unlike NodeName we use Node value only for element level. If we try to get the value of the whole document, it will return a null value: [6,195]

  $root->nodeName ……<tag name>

  $dom->nodeName……..Document

  $root->nodeValue……node value or text

Node type	Number
Element	1
Attribute	2
Text	3
CDATA section	4
Entity reference	5
Entity	6
PI (Processing Instruction)	7
Comment	8

Document	9
Document type	10
Document fragment	11
Notation	12
HTML document	13
DTD	14
Element declaration	15
Attribute declaration	16
Entity declaration	17
Namespace declaration	18
XInclude start	19
XInclude end	20
DocBook document	21

Table 1. List of all node types. [7,179-180]

As discussed in the above section, when we use the nodeType function it will return an integer. We could have a nodeType value as an integer which varies from 1 to 21. Each integer represents a certain type of node type. Table 1 shows the list of all node types and their numeric value.

3.4 Accessing Child Nodes

Child nodes are one level down or deeper from the root, and to fetch the value we use ChildNode property. This property returns a node list containing all the nodes. To check if a node has a child we use hasChildNode() method which returns a Boolean depending on their presence. One thing we should always remember is that in some browsers like FireFox, empty white spaces are considered as nodes. To solve this we first check if it is only a text node and then print out the value. [6,196;13]

```
if ($root->hasChildNodes()) {
        $children = $root->childNodes;
                foreach($children as $node) {
                if ($node->nodeType != XML_TEXT_NODE) {
                print $node->nodeName."\n";}}
```

Figure 3. Prints child node value. [6,196]

The code snippet in Figure 3 will iterate through all the child nodes that exist and print out their value.

Now we move to the next step which is accessing specific nodes. For this purpose we use getElementByTagName () method. The return value is a DOM list object of nodes that concedes under the specified tag name. Since we are looking for the values, we have to extend further and locate. For example, if we want to fetch from the first node of the list we use item() property as shown below. [7,192]

```
getElementByTagName('$name')->item(0)->firstChild->nodeValue ;
```

By changing the item number, we can iterate throughout the document and get all node values.

Another method which enables us to reach for a specific node is getElementByID (). Its implementation is similar to getElementByTagName ().

4 Database Management

4.1 Evolution of Database System

Database Management System (DBMS) is a collection of data which are interrelated to each other and sets of programs that enable us to interact with data. Database is just any collection of data and DBMS helps us to store and retrieve data in both convenient and efficient way. Storing and manipulation of data was a huge concern at the early stage of computer, the first general purpose DBMS was introduced by the mid of 1960s. Charles W. Bachman, computer scientist, designed a system which was based on network data model and he named it integrated data store. [8,5]

By the end of 1960s a company International Business Machines (IBM) developed a database management system which was a different way of representing data. This representation framework was based on hierarchal data model and it is still used in many institutes. In the meanwhile, IBM and American airlines developed a system which enabled customers to make a reservation and access the same data at a same time through a computer network. Even today many online ticketing services use this system. [8,6]

In 1970 another new way of data representation was discovered by Edgar Codd, who was working at IBM laboratory. It was based on relational data model. The fundamental concept behind the model was based on mathematics set theory and predicate logic. The idea was that a database consists of a series of relations that can be manipulated using non-procedural operations that return tables. In mathematical model reasoning such data is done by two valued predicate logic, it can have either true or false value and no other third value and this led to a concept of data with null values. The model guides designers to create a consistent and logical representation of information. This was a major step for the evolution of DBMS and Edgar Codd won the 1981 Turing Award for his work. [8,6]

The ability to store data and metadata made xml as one of database types. Metadata is the structure that describes the data in the database. Having the information about the data and the data itself, XML is famous for being self-descriptive. The metadata holds the attributes and elements all together and in addition it keeps the hierarchy of the document. As a comparison, the elements and attributes can relate to data tables and the fields on a relational data model; whereas hierarchy of the document is equivalent to relations between data tables. XML might store a huge amount of data in just one file with a long list of nodes and this feature raised issues like [9,246-248]

- for not being easily manageable
- not having a clear relationship visualization
- allow data redundancy
- scalability issue
- performance

Currently data in XML files are stored to a more advanced database and by using some extra installation, databases like SQL store XML data types. Since many other formats like PDF (Portable Document Format), Word and HTMLs can be converted to XML, we can store those documents to the database by using XML as presented in Figure 4.

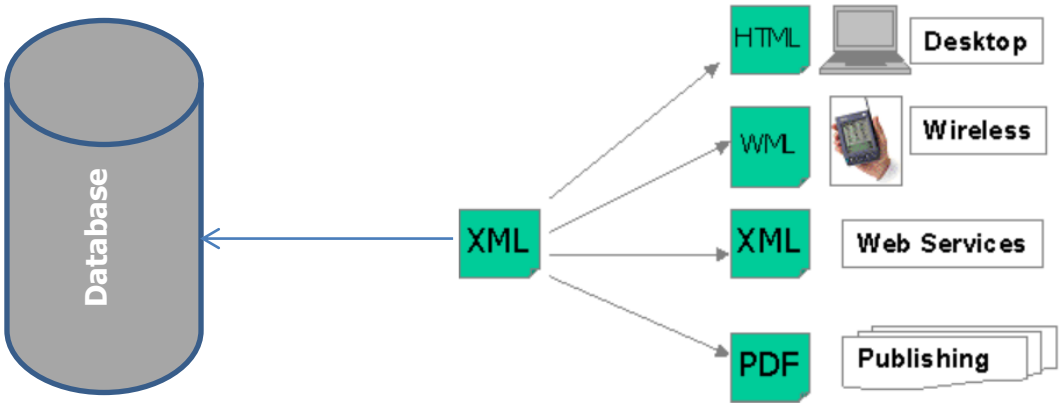


Figure 4. Other formats conversion to XML [10]

4.2 DBMS and File Management System

Both DBMS and file management system are used to save, organize, retrieve and manipulate data. File systems are a relatively traditional way of organizing data. They are collections of raw data files, which are saved in a hard-drive. DBMS are a set of applications that enable us for managing data which are stored in a database. Database management system is used not only for organizing databases, it also allows to search, create, merge, update and many other functionalities. We should not forget that at the end of the day all data are saved in some sort of files. Both systems are used for managing data but using a file system has some limitations, especially when regarding to an electronic data. DBMS discovered a bit later and it was successful by introducing flexibility on working with data and solving lots of problems in the file management system.

In File System- data are directly stored in a set of files and if there is only one table data saved in a file it is called a flat file. Flat file saves the data in text files in which each line of the text files holds only one record and each field may be separated by commas or tabs but can never have more than one table. However, some database programs like Microsoft Access can import them to other databases. Using file system, locating data and going through each record is a tiresome and tremendous work. There are two options of using file system, one is to read each record line by line, which is impossible if we have for instance 50 Gb of data, and the other one is to write a program that searches or fetches data automatically with the help of the file management capabilities of the operating system. Doing this needs advanced programmer skills. Some of the limitations of file systems are, [11]

- do not support multiuser access
- process one data at a time
- are applicable to limited and smaller databases
- have limited functionality
- have data redundancy issue
- do not support decentralization of data.

The structure of a file system is shown below in Figure 5.

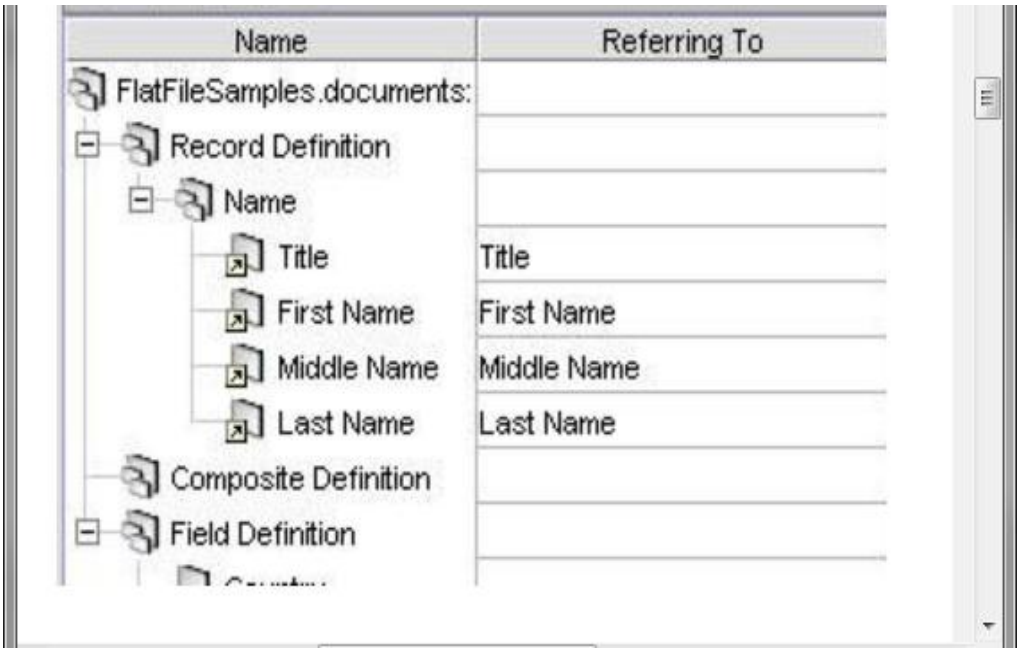


Figure 5. Flat File System [12]

Database management systems are typically made for controlling of database creation, retrieving of data and for maintenance purpose. Unlike file systems, they are sets of applications or computer programs and can be installed in physical hard drive. There are lots of DBMSs that are popular and which have powerful functionality, such as Oracle, MySql, SQL, PLSQL, DB2, Access and MsSql. All these modeling languages create a way for the user to manipulate data depending on their proper access right: in other words, it can be controlled by an admin and it can provide a different privilege for different user. DBMSs are divided in many parts based on their approach, hierarchal, context, associative, network and object. Other than solving the problems in the file system, DBMSs also provide backup, flexibility and way more facilities to users which is why they are considered as an advanced movement in organizing database. To mention the major positive aspects of DBMS are [11]

- supporting transaction of data
- storing data, update, retrieve without showing the details to the end user
- recovery service
- security
- data communication and forming relation between tables
- user accessible catalog.

A typical DMBS architecture is shown below in Figure 6

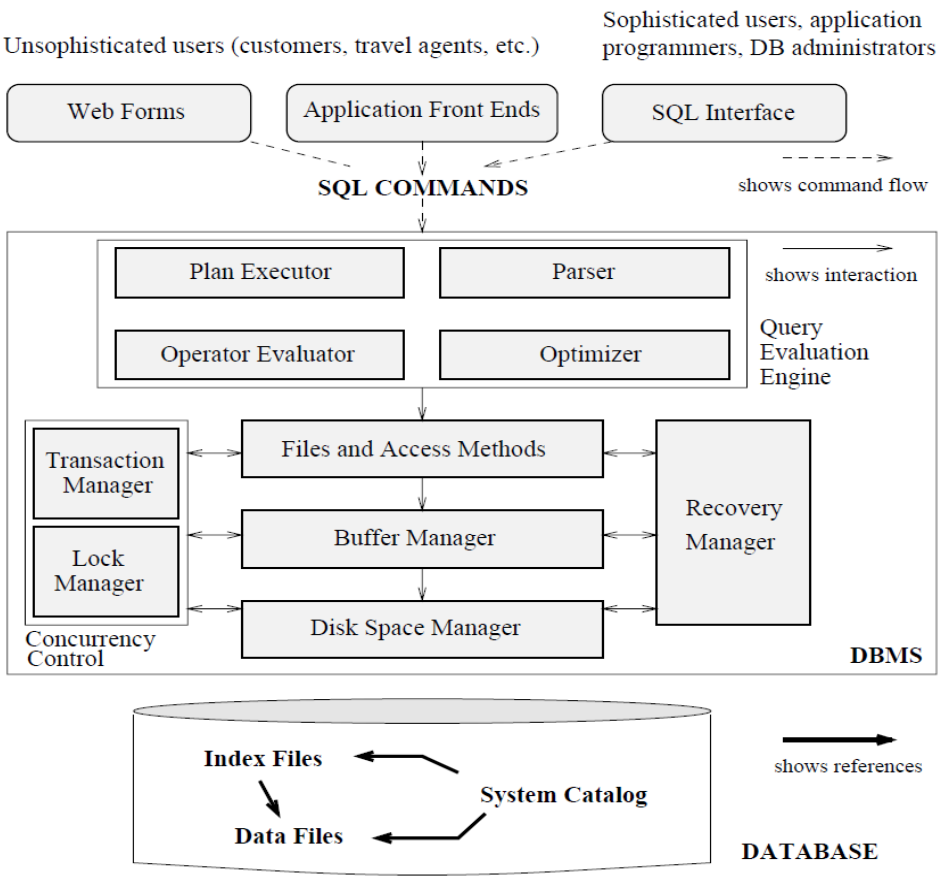


Figure 6. Architecture of DBMS. [8,20]

As we can see from Figure 6, DAMS can be accessed by multiple users, web forms or even some application such as web crawlers and other advanced software programs. All queries which are sent by the above users to a database are handled through the management system.

The process starts with a user or other program sending a SQL (Structured Query Language) command to the database. This will pass through the parser. The parser checks if the SQL is well written and breaks it down into specific signature of information which might contain routine name, order of parameters, number of parameters, types of function, owners name and others. This parsed information will be passed to the

query optimizer, which organizes the SQL statements for their execution as it is shown in Figure 6. [8,19]

This optimizer is a software that models the database how to execute by using the information gained from the parser. Modeling the plan is made by considering targets such as, the output efficiency, running cost, time span to carry out the order and easiness to process. The optimizer generates an execution plan which fetches the data rows that are needed to process the query. [10,19]

Execution plans are the blueprint for evaluating queries. Performing these plans take a considerable resource of CPU (Central Processing Unit), therefore execution plans are saved in a memory so that they can be reused. For each query the optimizer creates an estimated plan but just before executing they will be checked against the saved plans to see if they match. This is a very useful feature when we are dealing with huge load of queries. The plans are represented as a tree of relational operators; these relations operators work by expressions of numeric and string values. By using the operators it is possible to compare between values, and they always return a Boolean. [10,19]

The code that executes the relational operators is located on top of the file and access methods layer. The method layer includes different types of software which helps for supporting the concept of a file. File in DBMS is a collection of pages or records. This layer typically supports files that consists an unordered pages which is called heap. The method layer does not only keeps the page tracking, it also organizes the details of records that are in the page. These all process mentioned above are considered as a query evaluation engine. [8,19]

Next to file and access method we get the buffer manager. Buffer mangers retrieve pages from the database based on the methods received from the above layer. To explain best what kind of task carried out at this level, I will take a similar case presented in Raghu's DBMS book [8]. A database might contain millions of pages but out of those only thousands of the pages have a data; if a query is sent that scans the entire file, it would be impossible to bring all the data to the main memory. So the DBMS is responsible for the task and brings the pages as required and in a mean while

it also decides which pages should be exchanged with the coming new data containing pages, this process is called replacement policy. [8, 208]

The last level in DBMS software architecture is the disk space management. It ensures for the proper usage of disk and keeping track of which pages are allocated. It also provides commands that enable us to read or wright data on those pages. The dick manager supports the concept of a page as a unit of data. This way it will be easy to sequentially access blocks of data. The whole process and details will be hidden so that higher levels of the software think of data as a collection of pages. [8,207]

All the above stages and processes together build a complete database management system.

4.3 Layers of Data Abstraction

Even though main purpose of databases is for storing, they have a special way of presenting data according to user's privilege and specific query. When we say user, it could mean any application or a person that accesses the database for fetching or storing data. The capability of presenting data without showing the details is called user views. For example by using an Excel program, users can browse to file directories and access, modify or share data; these data should appear in a physical memory in a same directory as they are shown by the Excel program. If the user deletes or hides a certain row and saves the Excel, then with the next user, the data will be presented in a manner in which the first user saved it. The other way is to assign a physical file for each user but the result of this is that when one user updates the previous data, the other user's information will be out of date. In a database system we can show each user a different view of data according to their previous set without altering the original. This is possible because views do not actually store data, they just reflect changes made by the user or show an output based on user preference. [13,3]

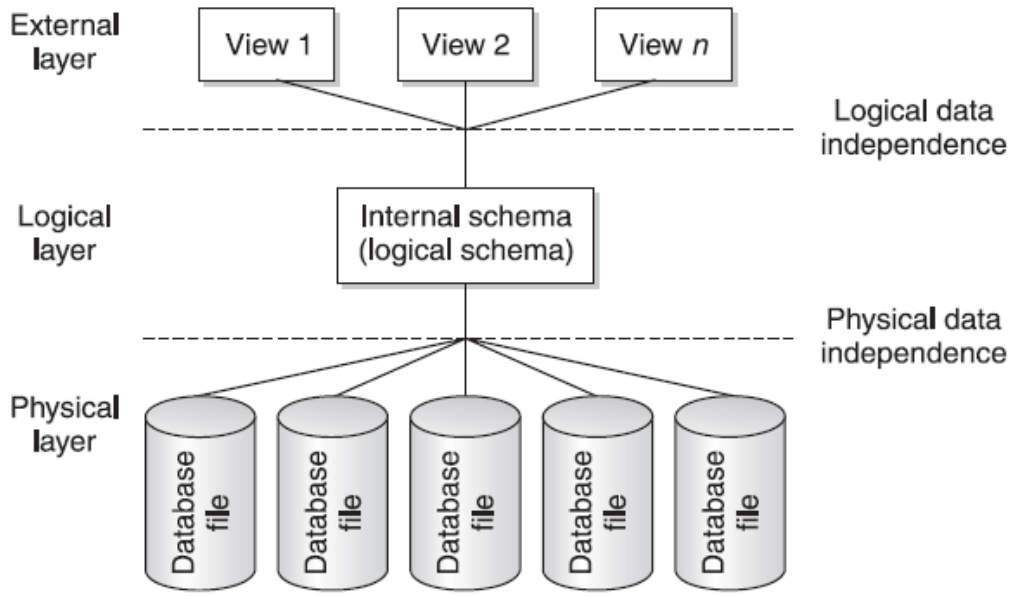


Figure 7. Layers Of Data Abstraction.[13,4]

The above architecture shown in Figure 7 was first developed by ANSI/SPARC (American National Standards Institute-Standards Planning and Requirements Committee). It became a start for most modern DBMS. The three basic layers that exist in current database management systems are external, logical and physical layers. [13,3]

The Physical Layer

The physical layer stores all the data files for the database system. With few exceptions like Microsoft Access, almost all current DBMSs store their database in multiple data files. These files are stored in different drives so that they can process side by side for maximum performance. In the case of Microsoft access or for similar management systems which stores their database in a single Physical file, it will have a great drawback on database scaling; that is why it is not recommended for enterprises with large database to use. This layer handles all the necessary processes such as opening,

closing, reading and writing of a file so an ordinary user should not have to worry about the details such as the exact location of the files or the name of the files. Once a data administrator handles the implementation and configuring of database software, the multiple users can work on the system. For example, with a word processing tool or spreadsheets the user must consciously save the document and choose the file name and location of storage. In DBMSs the physical files are handled by the system so users never need to memorize locations of files. [13,4]

The Logical Layer

The logical layer is one level higher than the Physical layer. This layer defines what data are stored to the database and what kind of relationship they have but as we discussed before, physical layers deal with how data are stored. Logical layer exist as an abstract from the lower file level. Unlike physical layers, they do not have a real appearance in the operating system. Using small numbers and structures this layer represents the real data files. Even though implementation of those structures for a physical layer might be complex, ordinary users do not need to know the detail process; but for administrators they might have to decide what information to store to which database. This layer is sometimes referred as a schema because it represents data files as a common structure. [13,5;14,17]

External Layer

Sometimes the external layer is also known as views. This layer is the highest level of abstraction which describes the database. It is an interface between user and DB (database) system. It connects all users and application programs that access the DB and take queries to process. Unlike others, these layers actually interact with the user; they store queries that provide users with a customized subset of the data from tables in the DB. For users external layers appear as table and act the same way as tables do, data is not stored on this layer but those queries which help to access the data are saved in them. Views are filtered form of the whole database. A user who needs to access a part of the DB should not be provided with all the DB tables, so views will

show a part of information that are specific for the user. We can have different views from one specific database. Some of external layer functionalities are [13,5;13,66;14,16]

- interacting with user
- having simplified interface
- representation of database
- showing only user requested data
- increasing database performance
- hiding complicated details that happens behind the scene.

4.4 Database Models

Database models are representations of real world information. They can be implemented through the data management system and make the data structure easy to understand for database users. Therefore data models serve as an interface to users and at the same time they hold the structure for database system. It is considered as a bridge between a real world and database system. Since data models contain information about the relation of data and structure of the database, they are considered to exist in the logical or conceptual level. Based on how the data is viewed by the system, we have different types of data models. [15,36;15,39]

The Hierarchical Model

The hierarchical model was discovered in 1960, back then it was applicable for handling large and complicated data. In big projects like NASA Apollo rocket, they used hierarchal model as data structure. This model is successive to file system. As the name hierarchy model is a self-defined, this model has levels. The top level is considered as a Root and the next step down is Root Children; another step down from root children is called a Child. In reverse we can refer to the Root level as a Parent for Children, the structure is illustrated in Figure 8. The development of this model helped for creating different and advanced models. By the end of 1970 database professionals

arranged for a meeting which led for the birth of Network Model as it had become evident that hierarchal model had some drawbacks like they [16,36]

- had only one type of relation between levels either it is a child or it is a parent
- were tedious to implementation
- were tiresome to manage
- did not have structural independence
- had only one-to-many relationship.

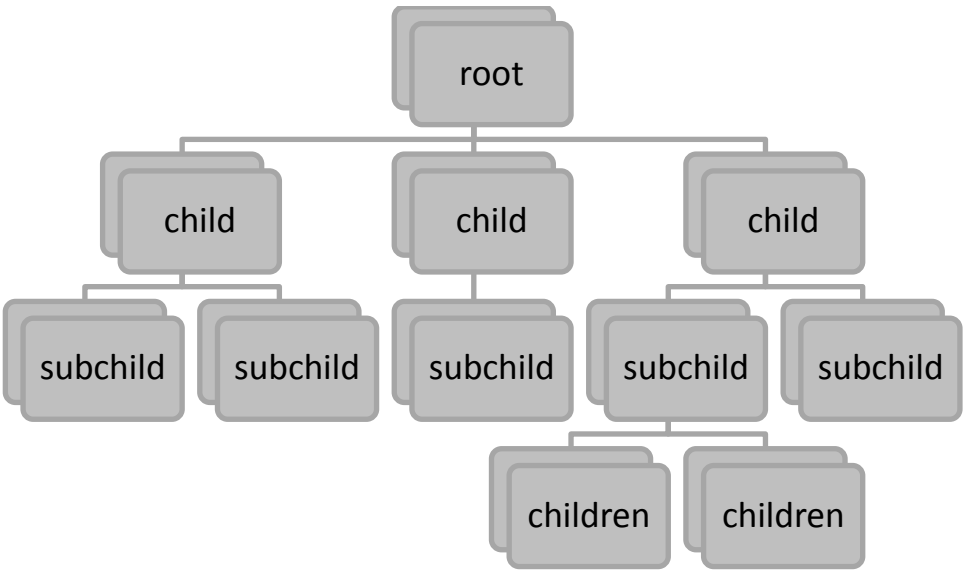


Figure 8. Hierarchical Model. [16,36]

The Network Model

Introduced right after hierarchal model, the Network Model shown in Figure 9 was the first to have a standard database structure. It has important improvements especially by introducing a new type of relation to the database structure. This means rather than having only one Parent to each Child, in network model a child can have multiple parents too. In this model the relation between each record is called a set. A set must have two record types which are titled Owner and Member. These features helped applications to access a data segment. A program can access owner record and the members contained within that set. In Hierarchy model there is only one parent, so it would be difficult to get the desired data. The other plus for the network model is that it was able to handle complex data; it increased performance and efficiency of data-

base. Even though it has improved many functionalities from previous model, it had some shortcomings, such as [16,37]

- as databases grow, using this model becomes too bulky.
- small structural change might be a disaster for application programs using the database.
- it has limited data independency.

Updating or insertion of data requires a lot of work for programmers. The Figure below illustrates the structure of a network model.

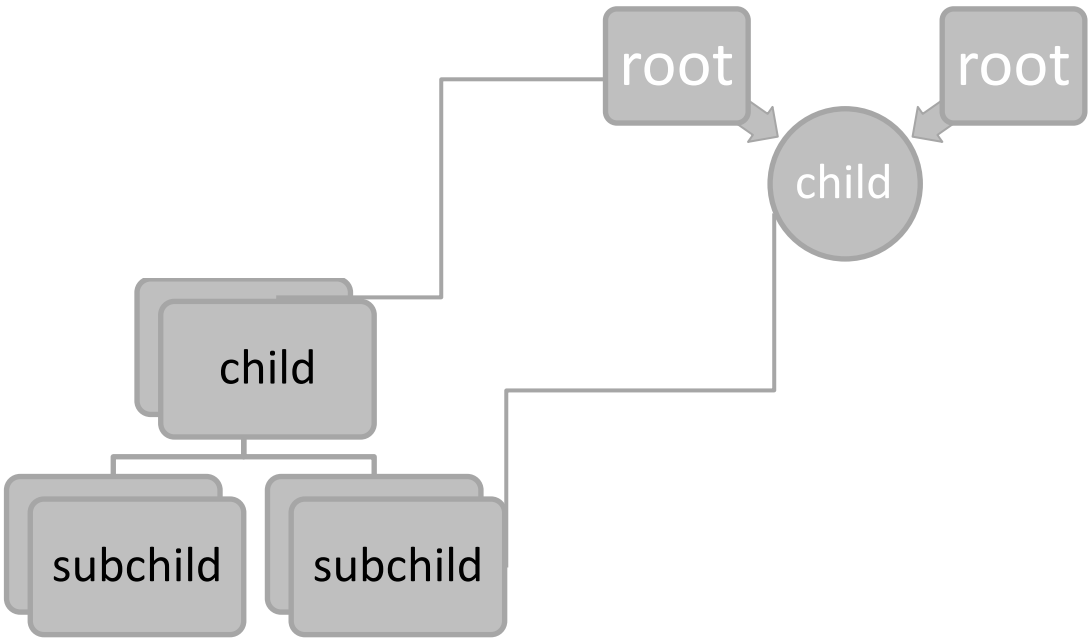


Figure 9. Network Model. [16,37]

Relational Model

Relational Model was discovered in 1970 by a British scientist named E. Codd. The model was developed based on a mathematical concept called relation. By that time computers had a very small power to implement sophisticated and complex models. As performance of computers increased and their running costs decreased, relational models became popular. In addition to providing some new features, this model performs all the functionalities which were provided by previous models. As a revolution it

managed to hide all the complex processes and was able to provide an enhanced view to users. Tables, which represent the relation model, contain rows and columns and it defines the connection between tables by the intersection of attributes. Therefore if there is a common attribute in a table, we have a relation. This helped to reduce the redundancy problem in a database. It also defines what kind relation tables have with each other; it could be one-to-one or one-to-many. The other big step was to introduce a structured query language which created a more convenient environment. This language takes order from users and arranges the database accordingly rather that restricting them to stick a certain structure. Now updating, deleting and insertion of data became handy. Because of all the processes are taken care at the background, users do not need to worry about the physical sides of the database. The figure below shows the relation between two tables namely Agent and Customer in a database. [16,38-39]

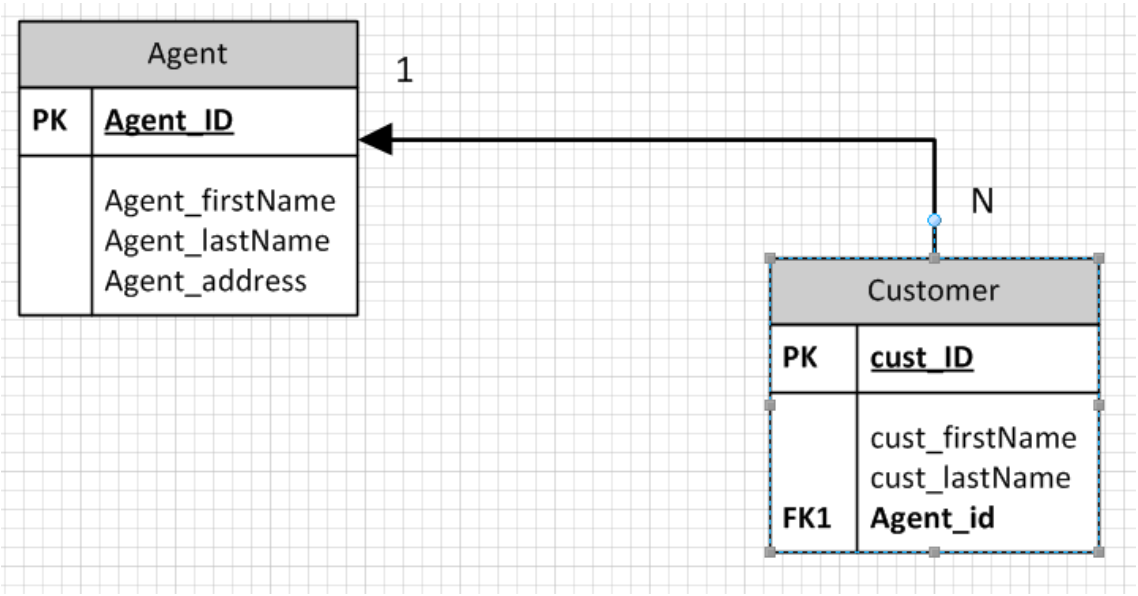


Figure 10. Relational Model. [16,39]

4.5 Indexing

Indexing is an essential term used in general database terminology that is accustomed to enhance the performance of a data retrieval mechanism by helping speed up query execution and the overall database performance. This procedure helps the query processor not to look at every row of each table for each query processed during the search. The underlying foundation for the quick search of the records of the database is the creation of what are called 'fields' of the database table. These fields are the means by which the indexes are created. Indexes can be built on one or more columns of a table. A fast and efficient searching of rows or a set of rows without scanning at the entire table is possible with these entities. The size of the table created for the purpose of indexing is less than the actual size of the table where the index originates.

Most relational databases support primary, unique, normal, and full-text indexing types where the most common type is the primary key index. This key is used to identify each record known as a row in the dataset. It can have a value of an integer that automatically increments for every data insertion. It is also possible to index using a pre-existing value, which makes each record unique to one another.

Unique indexes insure that no duplicate values are inserted enforcing the integrity of the table. They allow one primary key per table with multiple unique indexes. If further optimization of a database is required, a more advanced type of index must be implemented. Normal indexes operate by indexing the columns of the table to provide the query processor an improved way of looking up a value and displaying the output in a relatively less unit time. Normal indexes can be categorized as either Single-Column Normal Indexes or Multiple-Column Normal Indexes depending on the number of columns they have incorporated. Texts that are stored in CHAR, VARCHAR, or TEXT data types can be successfully indexed by a full-text type of indexing schema. The architecture of indexes can be grouped as either clustered or non-clustered. [17,208]

Clustered index

Records are inserted into a database table randomly by default. However, a clustered index changes the physical location of each row based on the columns it has been created. Indices created for these columns are pointers to the records where the actual records are distributed in a seemingly unordered manner. This process vastly speeds the data fetching process as there is no need to scan through the entire record looking for results. [17,209]

Non-clustered index

A Non-clustered index is a scan of the whole index and it is created consecutively looking for the search term. A non-clustered index search first navigates the index from the root node, through the intermediate node(s) that has been stated as 'Leaf nodes' (as it is shown in Figure 11), to the leaf node, and finally to the row. The importance of this kind of a search process is the tendency to be narrow, meaning it has only few columns to work with. The search will be completed once the required row has been found. The columns are found in the index because the index covers the needs of the query. Figure 11 shows a non-clustered indexing structure. [19, 1319]

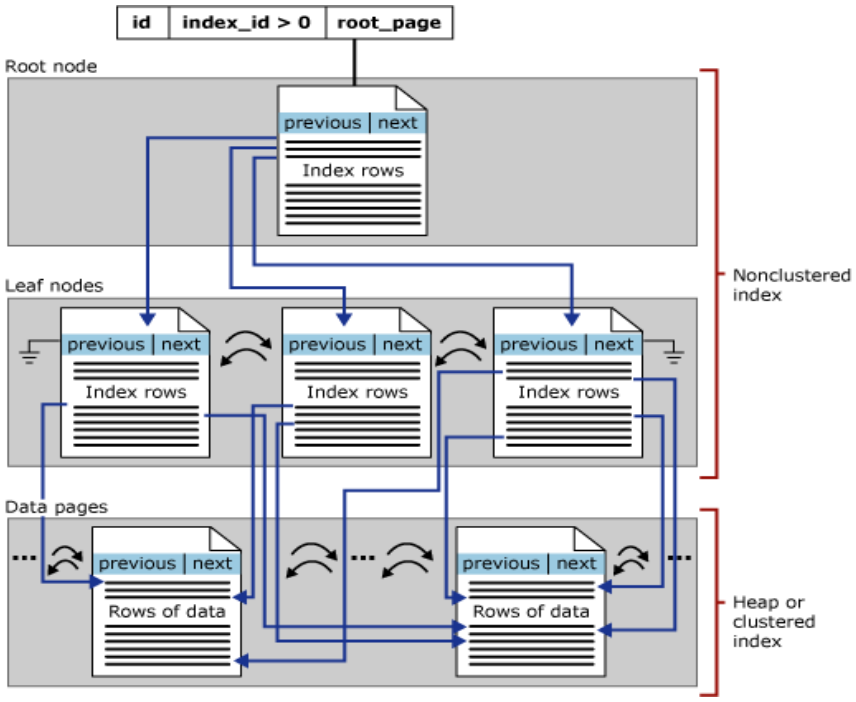


Figure 11 - Structure of a non-clustered index in a single partition. [18]

The table below illustrates the pros and cons of clustered and non-clustered indexes.

	Clustered	Non-Clustered
Pros	• Fast to return large range of data • Fast for presorted results	• Wide keys do not reflect on other indexes • Frequently updated key columns do not reflect on other indexes • Can be assigned on different FileGroup • Many non-clustered indexes per table • Smaller size than clustered indexes due to column subsets
Cons	• Frequently updated key columns reflect on non-clustered indexes • Wide keys increase the size of the non-clustered indexes • Only one clustered index per table	• Generally slower than clustered indexes due to bookmark lookup (except for covering indexes). • Not recommended for returning large data sets (except for covering indexes).

Table 2. Clustered and non-clustered indexing comparison

4.6 Database Normalization

Normalization in general sense it is the process of breaking down a collection of database fields into logical tables, and reduce the level of data redundancy in the database. Therefore, in this section we will see how normalization is taken place and its advantages and disadvantages.

Why do we normalize?

In modern database management system, the concept of database normalization is the backbone in order to achieve cost effective and secured database system. Database normalization ensures that our data tables reside properly and logically. Properly normalized database has benefits such as, better security, economical disk space use, faster speed of queries, and effective database update.

Normalizing a Database

As defined above, normalizing yields an optimum performance of a database. Logically designed database should eliminate the repetition of data stored into our database. The most common practice of normalizing a database is using *Normal forms* –which is measuring the extent of normalization applied in a given database. [20,229-231]

The most common types in normalization handling process are:

- First Normal Form
- Second Normal Form
- Third Normal form

First Normal Form

First normal form is the process of dividing database fields into more manageable tables. Once the tables have been created, the next step is assigning a primary key in all tables which makes every row uniquely identified. Mostly primary keys are assigned for fields that need to be uniquely represented in each row and the most common examples for this purpose are ID, SSN (Social Security Number) and GUID (Globally Unique identifier).

Second Normal Form

The second normal form takes the first normal form one step ahead and assures that each non-key attribute is associated functionally with the primary key. In order to get the desired result in 2NF, the first normal form must be accomplished properly. More specifically second Normal form emphasizes the functional dependencies such as key->value bases of two attributes at any given time.

Third Normal Form

The third Normal form is taking place when the following preconditions are fulfilled:

- The database must be in 2NF (Second Normal form).
- Non-key attributes are not transitively dependent.

Just as the rest normalization process, this one also plays an important role in reduction of data redundancy and inconsistencies of data that may happen. [20,231]

Drawback

As mentioned above, data fields are normalized to certain levels in order to increase the data integrity, and enhance the speed of query. However, there is one crucial drawback, which hinders the activity of basic hardware components such as CPU, input Out (I/O) and memory use. A normalized database requires efficient CPU and memory, especially during data query, where the process demands more resources in order to provide the requested data coming from various tables and which are joined together till they meet the requested data.

There is a solution for this problem, which is done by recombining the normalized tables together; as a result the number of tables that are needed during fetching data will be reduced. This practice obviously increases the performance of the database. However, this method is somehow expensive and it requires much more effort because we have to make sure that each field is updated consistently during data processing. Such method is called *denormalization.* [20,237]

5 Web Crawler Application

5.1 Purpose and Requirement

Purpose

The purpose of the project was to fetch specific data from http://www.muusikoiden .net and http://www.imperiumi.net using web crawler. These two websites provide information about artists and their gig schedule. The target of my application is to save all artist names, location of the gig, URL to an artists' personal webpage if there is one, date and any other additional information if it provided by the artist. When data from each website is stored to specific data table, it should perform merging. Because an artist can register to both websites, merging should handle duplicity of entities.

Requirement

Below the requirements for the web crawler application will be presented:

- The crawler goes through each page by incrementing the gig ID number
- The application skips pages that have ID but no gig information.
- While crawling, the application checks if there are missing data and send a null value to the database.
- On reload the crawler fetches only new entries.
- Information concerning the source should be saved in the database so that it will be easy to track back the original data.
- The application should analyze the collected data and rank the possible prefixes and suffixes.
- By user choice, the application should get rid of those prefixes and suffixes of artist names.
- While merging the two tables, the application must take care of data redundancy and at the same time it must keep basic information for back tracing.

5.2 General System

The application starts to operate when a request from a user is sent through a browser. Then the application browses both websites and collects specific data and stores it to database. A list of gathered information will be sent to the browser and it will display the result. In the meanwhile, data will be analyzed by the system and given rank for the possible prefixes and suffixes. The ranking is based on their repetition. After deciding the extensions (prefix and suffix), they will be removed from artist names and they will be saved to a separate table. The reason for removing those extensions is that, artists might register to both sites in a different form. When tables are merged, we could get two data rows with similar name or information. The final step would be merging and having one complete table which has complete information about artists and their gig schedule. Figure 12 shows the activities that are executed while using the application.

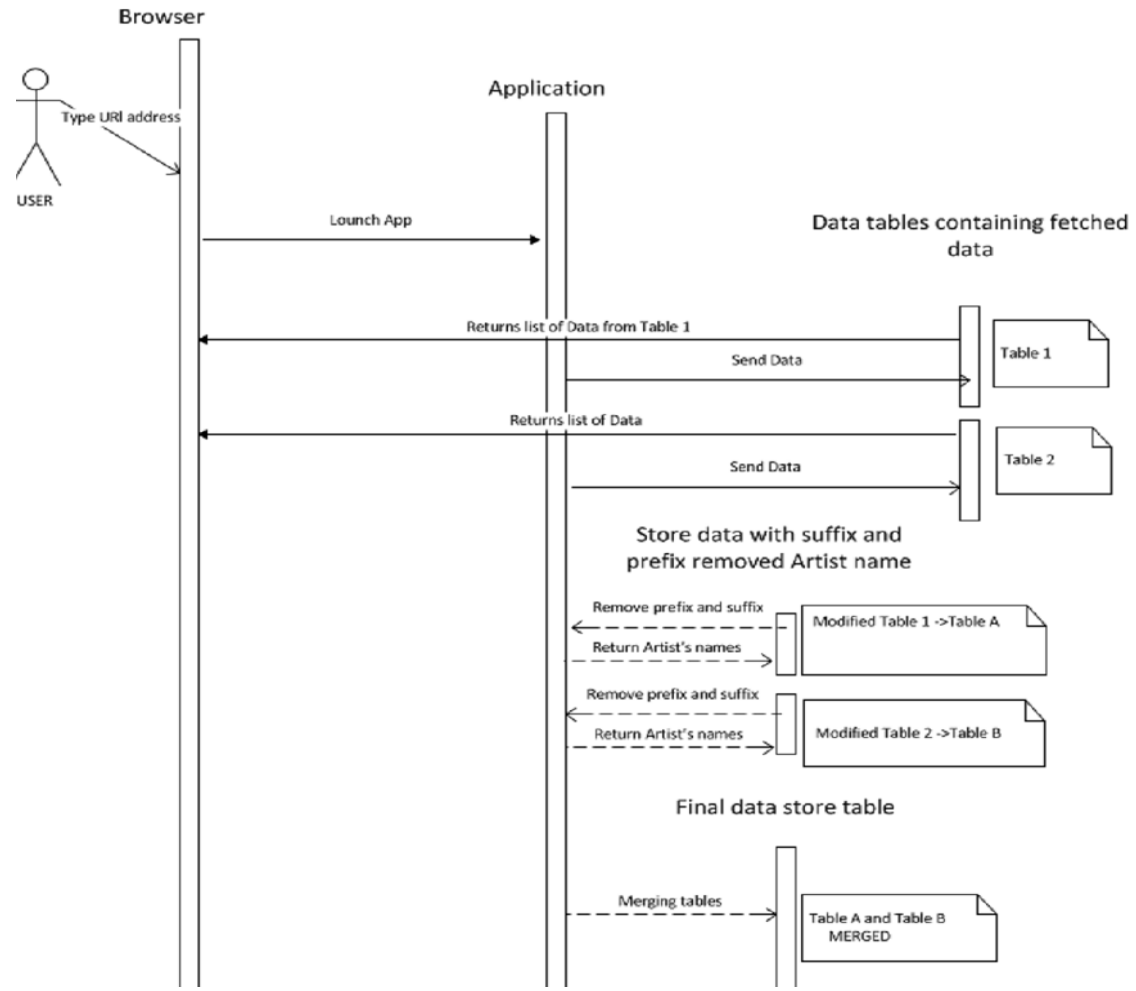


Figure 12. Activity diagram

Figure 12 shows overall flow of the application, the process starts from the top left corner where a user loads the application on a browser. Next the crawled list of information will be stored in two different data tables. The data will be sent to the user and displayed on the browser.

The next step is processing of artist name. This is the stage the application analyzes all the stored names and suggests a list of possible prefixes and suffixes. Then removing of those extensions from the original artist name will be carried-out and finally merging of the two data tables will take place.

5.3 Application Development

This web crawler application consists of two major classes which are database.php and crawler.php. Database.php contains methods which enable us to connect to the database. The crawler.php enables us to view the list of fetched data and to call some methods from the database class.

The very first step is to include the necessary classes like config and database. Then a database instance is created to crawler.php. Next, a loop that crawls to each page is made. The loop works by loading webpage addresses and calling a read page function. As mentioned before, by incrementing the gig ID number you can access all pages. This step is applicable only for these specific websites because others have their own arrangements. The code snippet below demonstrates the process.

```
for($x = 0; $x < 1000; $x++){

                    $page_start = microtime(true);
                    echo "<h4>page number : ".$x."</h4>";
          readPage('http://muusikoiden.net/keikat/'.$x, 50*$x);
                    $page_end = microtime(true);
                    $pagetime = $page_end - $page_start;
                    echo "Pagetime: ".$pagetime." seconds<br />";

$db->registerGig2($performer,$location,$url,$date,$addinformation,$x);

                    unset($performer);
                    unset ($location);
                    unset($url);
                    unset($date);
                    unset($addinformation);
```

Figure 13. Iterating readPage function

As shown in Figure 13, each gig has its own page and the loop calls readPage function for 1000 gig pages. The function returns the list of artist name, location of gig, date of performance. So by calling register gig function, all information can be sent to the database. Using the built-in function microtime, the time elapsed to load those pages can be displayed. After saving data, variables must be cleared using unset as shown in Figure 13.

Read-Page function first instantiates the DOM class and then loads the page. It is also possible to declare the document standard as it is shown in Figure 14. The function takes the website address as a parameter and it will be executed in the loop section shown in Figure 13.

```
function readPage($uri){

  $dom = new DOMDocument('1.0', 'iso-8859-1');
  @$dom->loadHTMLFile($uri);
  $tables = $dom->getElementsByTagName("table");

     if(isset($tables)){
           $rows = $tables->item(2)->getElementsByTagName('tr');

           $performerRow = $rows;
           $locationRow = $rows->item(0)->getElementsByTagName('a');
           $dateRow = $rows->item(0)->getElementsByTagName('span');

           $performer =$performerRow->item(3)->nodeValue;
           $location =$locationRow->item(0)->nodeValue;
           $date = $dateRow->item(0)->nodeValue;
           $additionlInfo = $performerRow->item(4)->nodeValue;

      }else echo "gig table note found!!";

}
```

Figure 14. Read page function

As it is shown in Figure 14, the first three lines of code execute the new document creation and loading of the web address. The next code will look for a table tag through the webpage. The isset() function will check if a table exists and it starts to parse. As we can see form the code in Figure 14, accessing a specific data cell is implemented by studying the structure of the web page. By changing the item number we can get different results.

At this stage we have all the required information of two tables in the data base. The next step will be to check if there is an extension on artist names because we need only names without their title like Mr, Ms, Dr and so on. The algorithm used for separating name extensions was, that if there is a space or period in between first names then the text before those characters (space, dot) should be considered as a prefix. The algorithm works the same way for the suffixes.

```
$artists1=$db->getAllNameTable1();

for($i=0;$i<count($artists1);$i++){

            $exploded= explode(".",$artists1[$i]);
            $prefix=$exploded[0];
            $db->registerPrefixes($pre[0]);
            echo $prefix."<br />";

            $sufPosition=count($exploded)-1;
            $suffix= ".".$exploded[$sufPosition];
            $db->registerSuffixes($exploded[$suffix]);
            echo $suffix."<br />";

}
```

Figure 15. Separate prefixes and suffixes

After calling all the artists from the database as shown in Figure 15, a built-in function called explode() will be used. This separates artist names if they contain a dot (.). The explode function divides a text and stores it as an array. The size of the array depends on how many portions the name has. For example, if the is name Dr.John.(fi), it has three portions which are separated by dot (.) namely, Dr, John and (fi). So the explode function will have an array of size three. Based on the concept discussed earlier, the first array is addressed as a prefix, the second would be artist name and the last array is taken as country or suffix. After separation, the next step is to send a collection of extensions to the database which is performed by the method register-prefixes. When there is a same prefix or suffix registered in the database more than once, a counter will record the number of repetition so that the higher repetition number we have, the higher the probability of it being a prefix or a suffix. This process will iterate for the entire artist name.

Now we have a list of probable extensions in our database with their rating. At this stage the user can see the highly rated prefixes and suffixes, and he or she can choose the value of rating to be considered as an extension. For example as shown in Figure 16, if a user chooses the rating (count) number to be eight on a prefixes table, then the application will consider the extensions Ms and Mr as prefixes. The database below will help to visualize this matter.

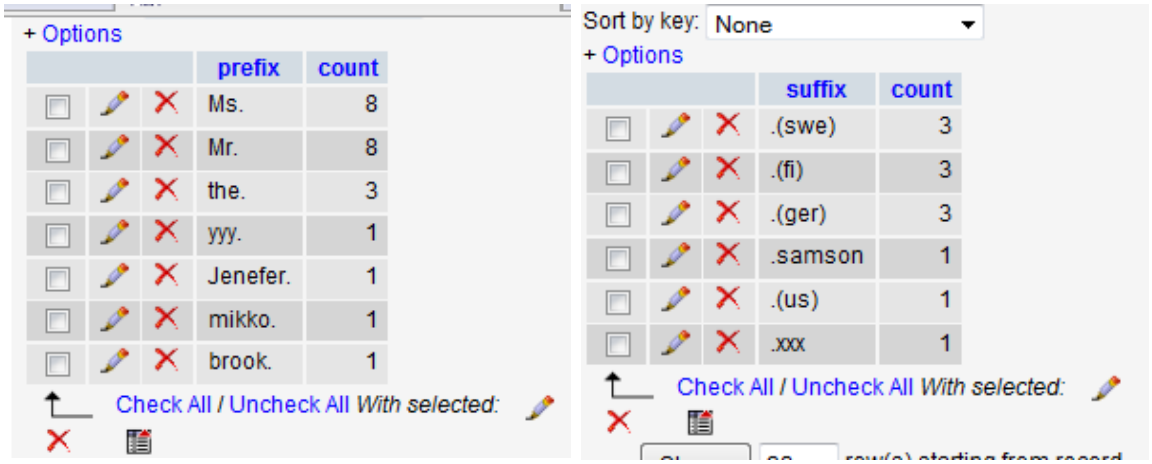


Figure 16. Prefixes and Suffixes with their repetition value

Figure 16 is taken from a demo database and it shows the list of extensions and how many times they have appeared while running the explode function to the demo list of names.

After selecting the value count, as it is shown in Figure 16, that should be considered as an extension then the application starts to list the prefixes and suffixes from the database and remove them from the artist's names. Using the built-in function sub-str_compare () we can first check if the selected prefix matches with the beginning of the artists' names.

```
function check_pre_fix($artist,$db){
          $pre_fixes = $db->getMostRepeatedPrefix();

 foreach($pre_fixes as $pre_fix){
                     $count_prefix = strlen($pre_fix);
                     $count_artist = strlen($artist);
                     $result_pre= sub-
str_compare($artist,$pre_fix,0,$count_prefix,true);

   if($result_pre==0){
            $trimartist_pre = $artist;
            $pre_trimed_artist=substr($trimartist_pre,$count_prefix);
            $artist = $pre_trimed_artist;
            return $artist;
        }else{
           echo "<br />dont match <br />";
             }
           }
        }
```

Figure 17. Remove prefixes

As shown in Figure 17, all the repeated prefixes will be called from the database first and then assigned to $prefixes. Using foreach loop it picks one prefix at a time and then tries to match it with the artist's name. To demonstrate this lets us take an artist name Mr.Jhon

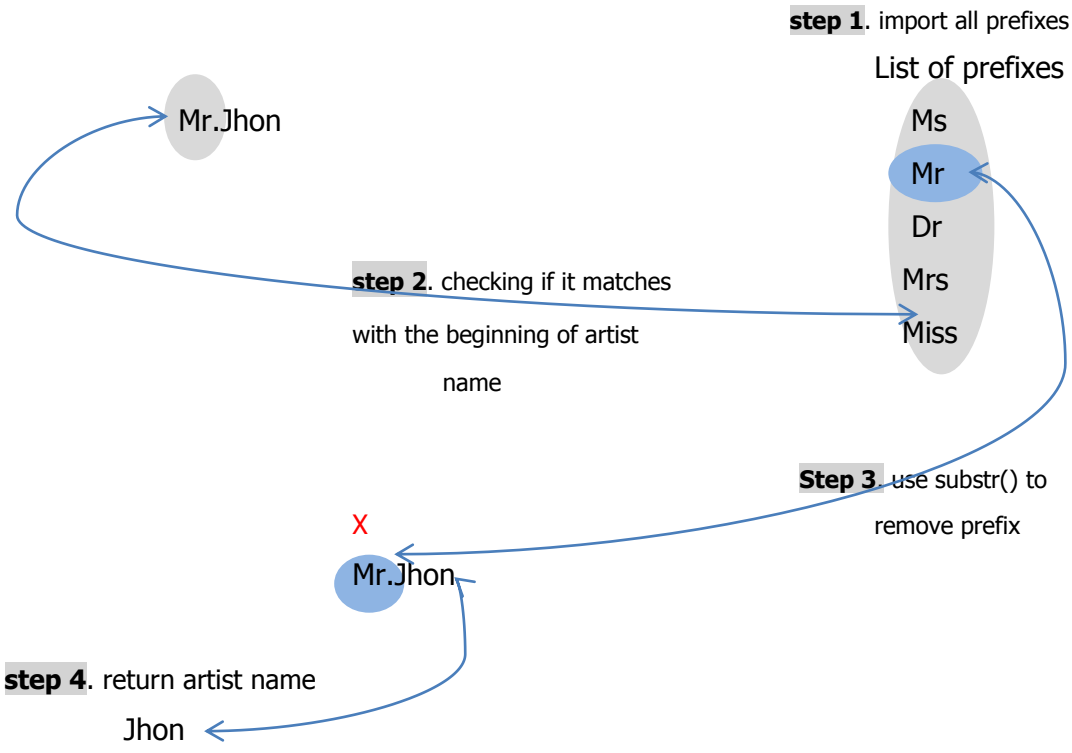


Figure 18. Steps on removing prefixes

Figure 18 shows how the code in Figure 17 snippet performs with a demo name Mr.Jhon. With suffixes, we use the same procedure but instead of comparing the extension at the beginning, we start counting from the end. The complete code for this part is shown in appendix 1. At this stage we have artist names with their extensions removed; now it will be much easier to merge both data tables because if there is a redundancy in artist names, it can be registered only once.

Merging data tables

The last step would be combining the two data tables in to one complete table without losing any information. Since the database for this specific project is too big to show in this paper, I made demo tables to demonstrate the process that went through the database. The tables can be found in appendix 1.

6 Conclusion

Currently collecting information from the net is a very serious business area, especially when you are providing services such as search engines. Web crawlers have been very successful in gathering information. Almost everything which is on the web can be accessed by this programming application and it has been playing a great role in everyday life. Due to the rapid growth and development of the internet and together with the aid of crawlers, it is almost a click away to read or to visit any web source. This has worked as a back-bone for different researches and great inventions. Although there are different ways of gathering information from the net, we can take the whole process one step further by using Document Object Model interface. The structured representation of web documents provided by DOM interface makes it possible to access specific portion of documents.

The crawler application with the extended feature made in this study was to gather specific data from two separate websites which are http://www.muusikoiden.net and http://www.imperiumi.net, and process their information. The data contains information about artists and their gig schedules. Data modification was made based on prefixes and suffixes of artist names. The application analyzes all the data and proposes a list of probable candidates for prefixes and suffixes. These name extensions had to be removed from original artist names because it was necessary to have one final data table with all the information in it. The problem was that the artist could register with and without a prefix at both websites which led to data redundancy.

This project was carried out to meet the requirements which were provided by the company. I used the best possible algorithm for the solution and the result was successful. In the future this application can also be upgraded and may lead to inventing and exploring different ways of data crawling from web documents.

References

1 Wp Themes Planet. How does a web crawler works [online]. URL: http://www.wpthemesplanet.com/2009/09/how-does-web-crawler-spider-work/. Accessed 26 April 2011.

2 NAZOU. Web crawler[online]. URL:http://nazou.fiit.stuba.sk/home/?page=webcrawler. Accessed 2 April 2011.

3 Levene M. An introduction to search engines and web navigation. 2nd ed. John Wiley and Sons. New Jersey. 2010.

4 Guisset F. DOM Levels[online].Mozila developer network. URL: https://developer.mozilla.org/en/DOM_Levels. Accessed 5 April 2011.

5 Snook J, Gustafson A, Langridge S and Webb D. Accelerated DOM scripting with Ajax, APIs, and libraries. Apress: New York; 2007.

6 Richards R. Pro PHP XML and Web services. Apress: New York; 2006.

7 Trachtenberg A. Upgrading to PHP 5. O'Reilly: Sebastopol, CA;2004.

8 Ramakrisnhan R and Gehrke J. Database Management System.3rd ed. McGraw-Hill: New York; 2003.

9 Powell G. Beginning XML Databases. Wiley publishing.inc: Toronto; 2007.

10 Cambridge Docs. Why convert document to XML [online]. URL: http://www.cambridgedocs.com/resources/whitepapers/id35.htm. Accessed 26 April 2011.

11 SQL server central. File system and DBMS [online]. URL:http://www.sqlservercentral.com/articles/Miscellaneous/dbmsvsfilemanagementsystem/1047/. Accessed 20 April 2011.

12 Top Bits.com. Flat file [online]. URL: http://www.tech-faq.com/flat-file.html. Accessed 20 April 2011.

13 Oppel AJ. Database demystified. The McGraw-Hill Companies:OH, USA; 2004.

14 Silberschatz K. Database system concepts. 4th ed. Foxit Software Company: Fremount, California; 2004.

15 Ponniah P. Data modeling fundamentals: a practical guide for IT professionals. New Jersey: A John Wiley & Sons, INC. 2007.

16 Rob P, Coronel C. Database systems: design implementation and management. 8th ed. Cengage Learning: Andover, Hampshire UK; 2009.

17 Kriegel A. Discovering SQL: a hands-on guide for beginners. Wiley Publishing Inc, Indianapolis, IN; 2011.

18 Non-clustered Index Structures [online].
URL: http://msdn.microsoft.com/en-us/library/ms177484.aspx. Accessed 4 May 2011.

19 Nielse P, White M and Parui U. Microsoft® SQL Server®2008 Bible. Wiley Publishing, Inc; Indianapolis, IN; 2008.

20 Stephens R, Plew R and Jones A. Sams teach yourself SQL in one hour a day. Upper Saddel River, NJ: Pearson Education inc; 2009.

Appendix

Checks for artist prefix and suffix

```
function check($artist,$db){

            $check_pre_fix=check_pre_fix($artist,$db);
            $check_suf_fix=check_suf_fix($artist,$db);
            //$check_both=check_both($artist);
            if(isset($check_pre_fix)&&isset($check_suf_fix)){
                        //echo "we have both";
                        $suf_removed=check_suf_fix($artist,$db);
                        $both_removed=check_pre_fix($suf_removed,$db);
                        return $both_removed;
            }if(isset($check_pre_fix)&&!isset($check_suf_fix)){
                        //echo "prefix only";
                        return $check_pre_fix;
            }if(!isset($check_pre_fix)&&isset($check_suf_fix)){
                        //echo "suffix only";
                        return $check_suf_fix;
            }if(!isset($check_pre_fix)&&!isset($check_suf_fix)){
                        //echo "clean";
                        return $artist;
            }
}
```

Checks for artist suffix and remove

```
function check_suf_fix($artist,$db){

            $suf_fixes = $db->getMostRepeatedSuffix();

            foreach($suf_fixes as $suf_fix){

                        $count_suffix = strlen($suf_fix);
                        $count_artist = strlen($artist);

                        $result_suf=                                    sub-
str_compare($artist,$suf_fix,($count_artist-
$count_suffix),$count_artist,true);

                        if($result_suf==0){

                                    //echo "<br>we have suffix";
                                    $trimartist_suf = $artist;
                                    //trim suffix from atrist

            $suf_trimed_artist=substr($trimartist_suf,0,($count_artist-
$count_suffix));
                                    $artist =$suf_trimed_artist;
                                    return $artist;
                        }

            }

}
```

Checks for artist prefix and remove

```
function check_pre_fix($artist,$db){

            $pre_fixes = $db->getMostRepeatedPrefix();
                        foreach($pre_fixes as $pre_fix){
                        $count_prefix = strlen($pre_fix);
                        $count_artist = strlen($artist);

                        $result_pre=                                    sub-
str_compare($artist,$pre_fix,0,$count_prefix,true);

                        if($result_pre==0){

                                    //echo "<br>we have prefix";
                                    $trimartist_pre = $artist;
                                    //trim prefix from atrist

            $pre_trimed_artist=substr($trimartist_pre,$count_prefix);
                                    //echo    "<br    />artist    name    :-
".$pre_trimed_artist;
                                    $artist = $pre_trimed_artist;

                                    return $artist;
                        }

            }

}
```

Table 1	Table 2

key: None
ns

		id	name	info
✎	✕	1	the.leul.(swe)	info table 1 the.leul(swe)
✎	✕	2	Ms.abey.(fi)	info table 1 Ms.abey(fi)
✎	✕	3	Mr.xxx.(ger)	info table 1 Mr.xxx(ger)
✎	✕	4	yyy.(swe)	info table 1 yyy(swe)
✎	✕	5	Mr.Brook.(fi)	info table 1 Brook(fi)
✎	✕	6	Jenefer.(fi)	info table 1 Ms.Jenefer(fi)
✎	✕	7	the.Kanye.(us)	info table 1 the.Kanye(us)
✎	✕	8	Ms.Beyonce	info table 1 Ms.Beyonce
✎	✕	9	Mr.hey.(swe)	info table 1 Mr.hey(swe)
✎	✕	10	mikko	info table 1 mikko
✎	✕	11	Ms.hey.(ger)	info table 1 Ms.hey

Check All / Uncheck All *With selected:* ✎ ✕

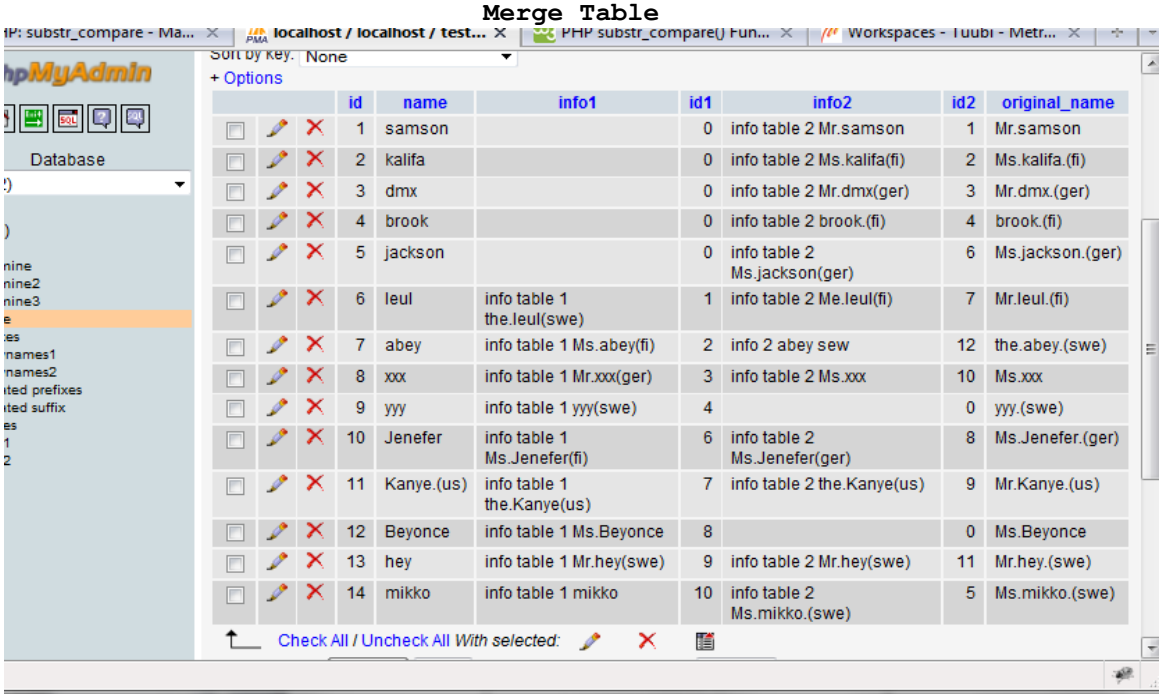
Show : 30 row(s) starting from record #

horizontal mode and repeat headers

y: None

		id	name	info
	✕	1	Mr.samson	info table 2 Mr.samson
	✕	2	Ms.kalifa.(fi)	info table 2 Ms.kalifa(fi)
	✕	3	Mr.dmx.(ger)	info table 2 Mr.dmx(ger)
	✕	4	brook.(fi)	info table 2 brook.(fi)
	✕	5	Ms.mikko.(swe)	info table 2 Ms.mikko.(s
	✕	6	Ms.jackson.(ger)	info table 2 Ms.jackson(
	✕	7	Mr.leul.(fi)	info table 2 Me.leul(fi)
	✕	8	Ms.Jenefer.(ger)	info table 2 Ms.Jenefer(
	✕	9	Mr.Kanye.(us)	info table 2 the.Kanye(us
	✕	10	Ms.xxx	info table 2 Ms.xxx
	✕	11	Mr.hey.(swe)	info table 2 Mr.hey(swe)
	✕	12	the.abey.(swe)	info 2 abey sew

Merge Table



	id	name	info1	id1	info2	id2	original_name
	1	samson		0	info table 2 Mr.samson	1	Mr.samson
	2	kalifa		0	info table 2 Ms.kalifa(fi)	2	Ms.kalifa.(fi)
	3	dmx		0	info table 2 Mr.dmx(ger)	3	Mr.dmx.(ger)
	4	brook		0	info table 2 brook.(fi)	4	brook.(fi)
	5	jackson		0	info table 2 Ms.jackson(ger)	6	Ms.jackson.(ger)
	6	leul	info table 1 the.leul(swe)	1	info table 2 Me.leul(fi)	7	Mr.leul.(fi)
	7	abey	info table 1 Ms.abey(fi)	2	info 2 abey sew	12	the.abey.(swe)
	8	xxx	info table 1 Mr.xxx(ger)	3	info table 2 Ms.xxx	10	Ms.xxx
	9	yyy	info table 1 yyy(swe)	4		0	yyy.(swe)
	10	Jenefer	info table 1 Ms.Jenefer(fi)	6	info table 2 Ms.Jenefer(ger)	8	Ms.Jenefer.(ger)
	11	Kanye.(us)	info table 1 the.Kanye(us)	7	info table 2 the.Kanye(us)	9	Mr.Kanye.(us)
	12	Beyonce	info table 1 Ms.Beyonce	8		0	Ms.Beyonce
	13	hey	info table 1 Mr.hey(swe)	9	info table 2 Mr.hey(swe)	11	Mr.hey.(swe)
	14	mikko	info table 1 mikko	10	info table 2 Ms.mikko.(swe)	5	Ms.mikko.(swe)