Database Metadata Requirements for Automated Web Development

Mohamed Ahmed Mgheder

PhD

2009
Database Metadata Requirements for Automated Web Development

A case study using PHP

Mohamed Ahmed Mgheder

BSc. MSc.

Submitted for the degree of Doctor of Philosophy

Department of Computing

School of Computing, Informatics and Media

University of Bradford

2009
I am very grateful to my Lord Almighty ALLAH who helped me and guided me throughout my life and made it possible. I could never have done it by myself!

I would like to acknowledge with great pleasure the continued support, and valuable advice of my supervisor Mr. Mick. J. Ridley, who gave me all the encouragement to carry out this work. I owe my loving thanks to my wife, and my children. They have lost a lot due to my research. Without their encouragement and understanding it would have been impossible for me to finish this work. My special gratitude is due to my parents and brothers and sisters for their loving support.
Abstract

The Web has come a long way. It started as a distributed document repository and quickly became the spring board for a new type of application. Propped on top of the original HTML+HTTP architecture, this new application platform shifted the way the architecture was used so that commands and functionality were embedded in the form data of Web requests rather than in the HTTP command conveying the request. This approach enabled Web requests to convey any type of data, not just document operations. This is occurring because the Web provides such a powerful platform on which to create applications. This is occurring because web development methods are still evolving toward the structure and stability required taking on this enormous new role.

As the needs of developers change, certain themes that arise more frequently than others become embedded into new environments to support those needs. Until recently, Web application programming has largely been done with a set of keywords and metaphors developed long before the Web became a popular place to program. APIs have been developed to support Web specific features, but they are no replacement for fundamental changes in the programming environment itself. The growth of Web applications requires a new type of programming designed specifically for the needs of the Web.

This thesis aims to contribute towards the development of an abstract framework to generate abstract and dynamic Web user interfaces that are not developed to a specific
platform. To meet this aim, this thesis suggests a general implementation of a prototype system that uses the information in database metadata in conjunction with PHP. Database metadata is richer in providing the information needed to build dynamic user interfaces. This thesis uses PHP and the abstract library ADOdb to provide us with a generalised database metadata based prototype. PHP does not have any restrictions on accessing and extracting database metadata from numerous database management systems. As a result, PHP and relational database were used to build the proposed framework. Additionally, ADOdb was used to link the two mentioned technologies. The implemented framework in this thesis demonstrates that it is possible to generate different automatic Web entry forms that are not specific at any platform.
List of Author’s Publications Related to this Thesis


## Contents

Chapter 1 .......................................................................................................................... 1

1.0 Introduction.................................................................................................................... 1

1.1 Motivations.................................................................................................................... 2

1.2 Aims and Objectives .................................................................................................... 3

1.3 Thesis Outline............................................................................................................... 4

Chapter 2 .......................................................................................................................... 5

Web Technologies.............................................................................................................. 5

2.0 Introduction.................................................................................................................... 5

2.1 HTML........................................................................................................................... 5

2.1.1 The Evolution of HTML............................................................................................ 6

2.1.2 Structure of an HTML Document.............................................................................. 12

2.1.3 Forms......................................................................................................................... 13

2.1.3.1 HTML Form Controls.......................................................................................... 16

2.1.4 Cascading Style Sheets (CSS).................................................................................. 19

2.2 Client-side processing ................................................................................................. 25

2.2.1 JavaScript................................................................................................................ 25

2.3 Server-side processing ............................................................................................... 28

2.3.1 CGI.......................................................................................................................... 29

2.3.2 FastCGI.................................................................................................................... 30

2.3.3 PERL......................................................................................................................... 30

2.3.4 Java Servlets............................................................................................................ 31

2.3.5 Java Server Pages (JSP).......................................................................................... 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.6</td>
<td>Active Server Pages (ASP)</td>
<td>34</td>
</tr>
<tr>
<td>2.4</td>
<td>PHP</td>
<td>35</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Introduction</td>
<td>35</td>
</tr>
<tr>
<td>2.4.2</td>
<td>PHP history</td>
<td>36</td>
</tr>
<tr>
<td>2.4.3</td>
<td>PHP General Features</td>
<td>38</td>
</tr>
<tr>
<td>2.4.4</td>
<td>PHP Syntax</td>
<td>40</td>
</tr>
<tr>
<td>2.4.5</td>
<td>PHP data types</td>
<td>43</td>
</tr>
<tr>
<td>2.4.6</td>
<td>PHP Libraries</td>
<td>44</td>
</tr>
<tr>
<td>2.5</td>
<td>PHP and Database</td>
<td>45</td>
</tr>
<tr>
<td>3.0</td>
<td>Introduction</td>
<td>50</td>
</tr>
<tr>
<td>3.1</td>
<td>Relational Database</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>SQL Overview</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Database Metadata</td>
<td>54</td>
</tr>
<tr>
<td>3.4</td>
<td>Database-independent programming interfaces</td>
<td>58</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Introduction</td>
<td>58</td>
</tr>
<tr>
<td>3.4.2</td>
<td>JDBC</td>
<td>59</td>
</tr>
<tr>
<td>3.4.2.1</td>
<td>JDBC Characteristics</td>
<td>60</td>
</tr>
<tr>
<td>3.4.2.2</td>
<td>JDBC Components</td>
<td>62</td>
</tr>
<tr>
<td>3.4.2.3</td>
<td>JDBC driver types</td>
<td>65</td>
</tr>
<tr>
<td>3.4.3</td>
<td>PERL DBI</td>
<td>65</td>
</tr>
<tr>
<td>3.5</td>
<td>Metadata in APIs</td>
<td>70</td>
</tr>
</tbody>
</table>
4.5.1 Related Work to User Interface to Databases ........................................... 99
4.5.1.1 Automatic generation of graphical user interfaces for interactive database applications ...................................................................................................................... 100
4.5.1.2 Accessing Relational Databases from the World Wide Web .............. 101
4.5.1.3 Automatically Generating World Wide Web Interfaces to Relational Databases. .......................................................................................................................... 102
4.5.2 Related work to User interface to metadata ........................................... 103
4.5.2.1 Metadata tables to enable dynamic data modelling and web interface design: the SEER example ........................................................................................................................ 103
4.5.2.2 Developing Web Entry Forms Based on Metadata ......................... 105
4.5.2.3 Developing an Abstract Representation for User Interface Elements Based on Database Metadata ........................................................................................................ 106
4.6 PHP .............................................................................................................. 107
4.6.1 Books and papers on PHP ........................................................................ 109
4.7 Conclusion ...................................................................................................... 111

Chapter 5 ........................................................................................................... 112
Framework Design and Architecture ................................................................. 112
5.0 Introduction .................................................................................................... 112
5.1 Framework Overview ................................................................................... 112
5.2 Prototype Features ....................................................................................... 114
5.3 Three-tier Connection ................................................................................ 115
5.3.1 The Web Browser .................................................................................... 116
5.3.2 The Web Server ....................................................................................... 116
List of Figures

Figure 2.1 HTML Form example ........................................................................................................... 19
Figure 2.2 Browser output of CSS example ............................................................................................ 21
Figure 2.3 CSS rule structure .................................................................................................................. 22
Figure 2.4 Comparison between traditional Web application and Ajax Web application models ........................................... 27
Figure 2.5 JSP MVC architecture .......................................................................................................... 33
Figure 2.6 PHP interaction process captured from [33]. ....................................................................... 42
Figure 3.1 The JDBC architecture ......................................................................................................... 64
Figure 3.2 The DBI architecture ........................................................................................................... 67
Figure 3.3 Metadata architecture .......................................................................................................... 70
Figure 3.4 Abstraction layer generation ............................................................................................... 76
Figure 3.5 ADOdb flow process ............................................................................................................ 81
Figure 4.1 Rails Model-View-Controller Framework captured from [67] ............................................... 88
Figure 4.2 CakePHP typical flow adopted from [70] .......................................................................... 90
Figure 4.3 Summary of the Different Categories of Web Development Tools adopted from [61] ................................................................. 94
Figure 4.4 DB2 WWW System overview adopted from [81] ............................................................. 102
Figure 4.5 A subset of the SEER data model captured from [3] ................................................................ 104
Figure 5.1 Metadata levels in the logical model .................................................................................. 114
Figure 5.2 Three-tier solution ............................................................................................................... 115
Figure 5.3 Architecture of the Framework .......................................................................................... 118
Figure 5.4 System components diagram ................................................................. 120
Figure 6.1 Part of the metadata retrieved from the database table in Listing 6.1. ....... 124
Figure 6.2 Extracted metadata from table listname ............................................. 127
Figure 6.3 Extracted metadata from table namelist .......................................... 128
Figure 6.4 Metadata information extracted from the table in Listing 5.2 ............. 138
Figure 6.5 Web interface example ..................................................................... 139
Figure 6.6 User Interface Design ....................................................................... 141
Figure 6.7 Database name selection form ......................................................... 142
Figure 6.8 Table name selection form ................................................................. 143
Figure 6.9 HTML Web form generated on the fly from metadata. .................... 144
Figure 6.10 HTML Web entry form generated from database metadata on the fly,
invoked in IE Web browser displayed using CSS ......................................... 145
Figure 6.11 HTML Web entry form generated from database metadata on the fly,
invoked in Fire Fox Web browser displayed using CSS .................................. 146
Figure 6.12 HTML Web entry form generated from database metadata on the fly,
invoked in Google Chrome Web browser displayed using CSS ..................... 147
Figure 6.13 Client side data validation ............................................................... 149
Figure 6.14 Required data validation ................................................................. 150
Figure 6.15 Data validation warning messages .................................................. 151
Figure 6.16 HTML Web entry form generated from MySQL database metadata on the
fly ......................................................................................................................... 154
Figure 6.17 HTML Web entry form generated from Oracle database metadata on the fly
.......................................................................................................................... 156
Figure 6.18 Metadata from a view .................................................................160

Figure 6.19 Form generated from a view .....................................................161
List of Tables

Table 2.1 HTML vs. XHTML ................................................................. 11

Table 3.1 Information schema .................................................................. 57

Table 3.2 Table Metadata Functions .......................................................... 69

Table 4.1 Rails and Struts like implementation in PHP ................................. 86

Table 4.2 The metadata table that represents the SEER data model adopted from [3]. 105

Table 4.3 PHP and Perl functions per intention captured from [84] ............... 110

Table 6.1 Defined generic data types .......................................................... 125
List of Listing

Listing 2.1 HTML document structure. ................................................................. 13
Listing 2.2 CSS example.................................................................................. 21
Listing 2.3 Example of Inline Styles............................................................... 22
Listing 2.4 Example of an Embedded Style Sheet........................................... 23
Listing 2.5 Linking to an External Style Sheet ................................................ 24
Listing 6.1 PostgreSQL database table’s structure ...................................... 122
Listing 6.2 Code snippet of ADOdb metadata methods ............................... 123
Listing 6.3 PostgreSQL table’s structure ......................................................... 126
Listing 6.4 PostgreSQL domain structure ..................................................... 127
Listing 6.5 PostgreSQL table with domain type .......................................... 128
Listing 6.6 PostgreSQL example to show a variety of defaults ................. 138
Listing 6.7 MySQL database table’s structure .............................................. 153
Listing 6.8 Oracle database table’s structure ............................................... 155
Chapter 1

1.0 Introduction

The Internet has changed our world. Gathering information, buying goods and services, finding people, making travel reservations, and more can now be done from the comfort of your own chair. It is changing the nature of the way business is done. People can search for the best prices on virtually any product or service. Special-interest communities can stay in touch with one another and researchers can learn of scientific and academic breakthroughs worldwide.

The Internet mixes both computing and communications technologies. It makes information instantly and conveniently accessible worldwide. Individuals and small businesses can receive worldwide exposure on the Internet.

The World Wide Web (WWW) is one of the applications of the internet; allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animations, audios or videos) on almost any subject.

Most computer applications in the past were executed on computers that were not connected to one another. Today’s applications can be written to communicate with hundreds of millions of computers. Building software quickly, correctly and economically remains a subtle goal at a time when demands for new and more powerful software are soaring.

A modern web site is not just a web server; it also includes a way of storing and
Introduction

querying data, a means of processing the requests from the user and creating documents with the appropriate information. There are many options open to the Web developer, but not all of them as open and general as others. We should not only consider creating a site with dynamic content, we need to be sure that we can still provide the content independently of the changes in hardware or software technology. Many approaches have been made in using database metadata to automatically generate and improve web interfaces[1] and [2]. Most of these approaches do not use database metadata but rather rely on an external model of the database to provide information about the database[3] and [4], or to the approach common when using Web Development Frameworks where the information is essentially stored in the application rather than in the database[5].

1.1 Motivations

Database Management Systems (DBMSs) are the tools that manage all the information, since software that combines database with the Web are rapidly becoming essential web site development tools. The number of products that connect databases to the Web is increasing faster than any other segment of internet software. One of the most important things to consider when developing a Web application is portability. Given the rapid pace of change in the Web world, it doesn't do to bind the code too tightly to a specific operating system, RDBMS or programming language. PHP has enough built in support for a majority of database systems. Unfortunately each
supported database has different functions to do the same things. This causes problems as far as code portability goes. The database abstraction library allows switching between one RDBMS and another, without requiring either a code rewrite or a long retest cycle.

This thesis will focus on PHP, and the ADOdb library, also written in PHP, this library will function as a wrapper around the PHP script, exposing a set of generic methods to interact with a database server. These generic methods are responsible for mapping to the native API for each corresponding database.

1.2 Aims and Objectives

The aim of this thesis is to play a part towards the development of a generic approach that can be used to generate automatic and dynamic Web application systems, using database metadata. Based on the existing Web technology and the related work the following points are intended to be accomplished throughout the rest of this research.

- To perform an extensive literature review about the existing web technologies used in Web application development.
- To investigate the potential resources of database metadata in order to verify to what extent these resources can provide us with sufficient information to accomplish the main aim.
- To investigate the development of dynamic web database interfaces in PHP.
- To make use of database metadata and the abstract library ADOdb features to make our prototype as generic as possible.
1.3 Thesis Outline

The rest of the thesis is structured as follows:

- Chapter 2: This chapter discusses the latest development in Web technologies. The purpose of this chapter is providing a survey of the available technologies and to examine the suitable ones to implement the proposed prototype.

- Chapter 3: This chapter conducts an overview of the most recent development in database technologies in order to assess the suitability of various technologies in the implementation of the framework system to be developed during this research.

- Chapter 4: This chapter conducts a literature review that is related to user interfaces to database. A number of academic research papers and commercial products related to user interface to database are discussed. This chapter is split into four sections. The first section discusses the Application Centered Approach. The second section discusses External schema approach. The Third section deals with the issue of Database Centred Approach. The last section discusses related work to PHP.

- Chapter 5: This chapter starts with an overview of the prototype features. Framework layers and architecture are demonstrated in detail.

- Chapter 6: This chapter shows how the system is connected to a database. Automatically generating Web user interface using metadata is discussed and illustrated. Finally a framework evaluation is demonstrated.

- Chapter 7: This chapter concludes the thesis and points out possible future work.
Chapter 2

Web Technologies

2.0 Introduction

The rapidly growing amount of information made available on the internet drives the need for having tools to help maintain and structure this information. Some of this information is already located in databases that are accessed via other web applications, specifically Web Forms that are developed using standard HyperText Markup Language (HTML) [6].

Web Form based applications are implemented using HTML, add to that client side scripting such as JavaScript [7] which is used as data validation, and server side technologies such as PHP [8], and JSP [9] that are used for more complex processing and functionality.

It is now easy to develop dynamic Web based applications that use databases using these technologies. In the following sections we summarise the historic development and focus on features such as forms, used later.

2.1 HTML

The World Wide Web, or WWW is built out of Web pages, and those pages are formatted as HyperText Markup Language, or HTML.

HTML is theoretically designed to structure documents and make their content more
accessible, not to format documents for display purposes. HTML does provide many different ways to define the appearance of documents: font specifications, line breaks, and multicolumn text are all features of the language.

HTML has grown considerably since its first implementation. It was initially developed for the creation of WWW pages, HTML proved useful for many other, originally unforeseen tasks, and has therefore evolved over time into service of the creation of corporate intranets, for spicing up e-mail and news messages, and even for developing Graphical User Interfaces (GUIs) for stand-alone and net centric applications[10]. The original HTML language allowed only hyperlinks and some basic formatting of ASCII text. Later we have the addition of forms (which allow the user to transmit data to the browser), tables, and frames etc.

2.1.1 The Evolution of HTML

- **HTML**

At Level 0, HTML offered a platform-independent means of marking data for interchange. The concept was that servers would store and supply data and clients would retrieve and display it. The only required element was the title element, and many older pages still remain that start with a title and then go straight into the text. There are no separate elements for html, head and body. A `<P>` tag was often used at the end of each paragraph to separate paragraphs.

There were six levels of headings, but with the expectation that each level would be used only once and that the levels would be used sequentially.
• **HTML 1.0**

Along with title, the head element could contain the attributes of isindex, link, and base, giving the document a context within a larger universe. Along with the anchor element, the image element allowed GIF (CompuServe’s Graphics Interchange Format) files to be displayed within the text if the browser supported that format. The horizontal rule was also introduced, beginning the slide down the slippery slope of giving the author some control over the document's final appearance to the user.

Level 1 also introduced forms, which make it possible for authors to have input fields on their nodes that enable feedback from users and open the door to considering interaction through Common Gateway Interface (CGI) scripting.

• **HTML 2.0**

Increasing demands on HTML led the newly formed World Wide Web Consortium (W3C) to introduce HTML 2. Level 2 used the form element with input, select, option, and textarea, plus the `<BR>` element for line breaks.

HTML 2 also added the meta element for detailed document description, which also provided a possibility for indexing and cataloguing the contents. It also changed the descriptions of the head and body sections and the anchor, base, lists, image, link, and title elements.

• **HTML 3.0**

HTML 3 was proposed as an attempt to address the competing demands for a markup language that operated across all platforms and for a page description language that was
acceptable to all software companies. The theory was that HTML 3 would be fully SGML-compliant [11], but enable hints to browsers on how to display certain text. Elements proposed for HTML 3 included a fig element that supported text flow around figures. Support was also suggested for mathematical equations and a table element to help format tabular data without the hassle of the pre element. The align attribute was added to several elements, including img, p, and hr, enabling authors to provide for left, right, or centre justification. On the character level, HTML 3 also proposed some new logical elements, including tags for definitions, quotations, language, inserted text, and deleted text. Some of the proposed physical tags were underlined text, bigger and smaller text, and subscript and superscript. Additional attributes were also proposed for background images, tabs, footnotes, and banners. Consideration was also given to style sheets in HTML 3, by removing the majority of display-oriented elements from HTML and placing them in style sheets.

- **HTML 3.2**

HTML 3.2 repeated the things that were accepted/used widely which were proposed on the previous version (HTML 3.0), and offered new elements and attributes to Web pages e.g. animation, colours, and sound. HTML3.2 added the script and style tags to make room for scripting languages and style sheets. It formalized practices such as colours for backgrounds, text, and links and sizing, alignment, and spacing for images. It made it possible to create attractive, dynamic Web pages that complied with an established standard.
• **HTML 4**

HTML 4 enables separating physical styles from the content mark up by relying on style sheets (see section 2.2.4). Many see the style sheet solution as the best way to suggest a document's appearance yet still enable full use of HTML as an SGML-compliant method of bridging the gap between users and operating systems. The W3C has introduced the object element, along with enough flexibility for it to serve in other roles in the future. The new standard for HTML also makes use of other W3C proposals for the Web [12]. The HTML 4 standard includes the style, div, and span elements for incorporating style sheets, but the actual style sheet specification is a separate entity.

• **HTML 5**

The HTML 5 draft was originally developed by the Web Hypertext Application Technology Working Group (WHATWG) [13]. It introduces and enhances a wide range of features including form controls, APIs, multimedia, and structure, to enable more interactive and exciting websites and applications. HTML 5 gives web developers a lot of options and new features for effectively and easily handling input fields and form submissions. It is aim is to provide a better way to describe the content displayed on a web page, enable more complex content types, improve media and web application support, and increase the interoperability of HTML documents.
XHTML was created by the World Wide Web Consortium (WC3) in 1996, amongst its aims was to provide a common markup language for wireless devices and other small devices with limited memory [14]. It is a widely supported open technology that is used for data exchange between any types of application that can understand eXtensible Markup Language (XML)[15]. XML was developed as a substitute for SGML and therefore XHTML is to XML as HTML was to SGML that is a language for Web documents based on a meta language. The combination of XML with Hypertext Markup Language (HTML) produces a web presentation language that is flexible (extensible) based on the needs of the application it is being used for. XHTML is a reformulation of HTML following the more stringent rules of XML, which is a powerful language that allows web authors to create their own customized tags. XHTML, unlike XML, offers a finite set of predefined tags to choose from. XHTML is similar to HTML 4.01, with just a few more rules dictating how it must be written. Notice that the tag name is written in lowercase, which is a requirement of XHTML; tag names are not case-sensitive in HTML, but they must be lowercase in XHTML (that’s one of those more stringent rules that distinguishes XHTML from HTML) see Table 2.1.
Web Technologies

<table>
<thead>
<tr>
<th>HTML</th>
<th>XHTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag and attribute names are not case-sensitive.</td>
<td>Tag and attribute names must be written in lowercase.</td>
</tr>
<tr>
<td>Some attributes can be minimized, and attribute values do not require quotes.</td>
<td>All attributes must have a specified value, and the value must be quoted.</td>
</tr>
<tr>
<td>Some elements do not require closing tags, and empty elements should not be closed with a trailing slash.</td>
<td>All elements must be closed, either with a closing tag for nonempty elements or with a trailing slash for empty elements.</td>
</tr>
</tbody>
</table>

Table 2.1 HTML vs. XHTML

The Syntax of a Tag

Every HTML tag consists of a tag name, sometimes followed by an optional list of tag attributes, all placed between opening and closing brackets (< and >). The simplest tag is nothing more than a name appropriately enclosed in brackets, such as <head> and <i>. More complicated tags contain one or more attributes, which specify or modify the behaviour of the tag.

Tag attributes, if any, belong after the tag name, each separated by one or more tab, space, or return characters.

Their order of appearance is not important. A tag attribute's value, if any, follows an equal sign (=) after the attribute name.

<input type=text name=filename size=24>
Some tags indicate empty elements, which are elements that do not hold any contents. Empty elements do not require a closing tag but are instead self-closed in HTML. For example, the `<BR>` tag causes a line break; it has no effect otherwise on the subsequent portion of the document and, hence, does not need an ending tag. However, in XHTML this tag is written with a trailing slash at the end of it `<BR/>`.

In XHTML documents, all attribute values, whether they are strings or numbers, must be enclosed in quotation marks. For example, when setting attributes such as height for an image element, the numbers must be surrounded by quotation marks.

### 2.1.2 Structure of an HTML Document

An HTML document consists of text, which defines the content of the document, and tags, which define the structure and appearance of the document. The structure of an HTML document is simple, consisting of an outer `<html>` tag enclosing the document head and body Listing 2.1:
Each document has a head and a body, delimited by the `<head>` and `<body>` tags. The head is where you give your HTML document a title and where you indicate other parameters the browser may use when displaying the document. The body is where you put the actual contents of the HTML document. This includes the text for display and document control markers (tags) that tells the browser how to display the text. Tags also reference files, including graphics and sound, and indicate the hot spots (hyperlinks and anchors) that link your document to other documents.

### 2.1.3 Forms

Forms in an HTML page provide an interface between a user and some hidden servers, databases, and applications. They enable the user to access information or provide inputs to some data repository.

HTML forms provide a set of input objects that can be used to build complex and functional forms and user interfaces. These objects include radio buttons, image maps, input fields, and scrolled lists.
It is permissible to create one or more special form sections in the HTML document, bounded with the `<form>` and `</form>` tags. Inside the form, you may put predefined as well as customized text-input boxes allowing for both single and multi-line input. It is possible to insert checkboxes and radio buttons for making choice selections and special buttons such as submit and reset to send the form contents to be processed by a server script or to reset the form.

Forms are supported by almost every browser and make it possible to create documents that collect and process user input, and to formulate personalized replies. It gives an automated way to interact with HTML document readers.

**The `<form>` Tag**

You can place a form anywhere inside the body of an HTML document with its elements enclosed by the `<form>` tag and its respective end tag `</form>`.  

 `<form>` defines a form

 End tag:

`</form>`; never omitted

There are no special layout rules for form elements, so you need to use other HTML elements, like tables and style sheets, to control the placement of elements within the text flow.

At least two special form attributes must be defined, which provide the name of the form's processing server and the method by which the parameters are to be sent to the server.
A third, optional attribute lets you change how the parameters get encoded for secure transmission over the network.

**The action attribute**

The required action attribute for the `<form>` tag gives the URL of the application that is to receive and process the form's data.

A typical `<form>` tag with the action attribute looks like this:

```html
<form action="URL of the script">
    ........
</form>
```

The example URL tells the browser to contact the web server and pass along the user's form values to the application.

**The method attribute**

The other required attribute for the `<form>` tag sets the method by which the browser sends the form's data to the server for processing. There are two ways: the POST method and the GET method. With the POST method, the browser sends the data in two steps: the browser first contacts the form-processing server specified in the action attribute, and, once contact is made, sends the data to the server in a separate transmission.

On the server side, POST-style applications are expected to read the parameters from a standard location once they begin execution.
Once read, the parameters must be decoded before the application can use the form values. Each particular server will define exactly how the POST-style applications can expect to receive their parameters.

The GET method, on the other hand, contacts the form-processing server and sends the form data in a single transmission step: the browser appends the data to the form's action URL, separated by the question mark character.

The common browsers transmit the form information by either method; some servers receive the form data by only one or the other method. You indicate which of the two methods POST or GET your forms-processing server handles with the method attribute in the <form> tag. Here's the complete tag including the GET transmission method attribute:

```html
<form method=GET action="script">
...
</form>
```

### 2.1.3.1 HTML Form Controls

There are three types of tags for constructing the user interface: `<INPUT>`, `<SELECT>`, and `<TEXTAREA>`. Any number of these tags can be put between the `<FORM>` and `</FORM>` container tags. The most used form tag is the `<INPUT>`. This tag can accept any data type specified by the `type` attribute.
Among a dozen input types see Figure 2.1, the list below shows the most commonly used ones:

1. **Text**: is the default input type, gathers a simple line of text and this can include numeric digits, letters and special characters. The attributes NAME (this is required), SIZE, VALUE, MAXLENGTH can be used with TEXT. This is constructed as:

   \[\text{<input type="text" name="any name">}\]

2. **PASSWORD**: This control is similar to a text field; it’s a single-line field and will usually appear as a rectangular box with a white background and an inset border. But unlike a regular text field, a password field obscures the entered text, usually as a series of asterisks (*) or solid dots. This offers a bit of added security and privacy. The layout of this field is as:

   \[\text{<input type="password" name="any name">}\]

3. **CHECK BOX**: displays a simple check box that can be checked or left empty; possible attributes to include with this type NAME (required), VALUE, and CHECKED (defaults the check box as checked). Check boxes are useful when there is a list of options, more than one of which can be selected at a time. They are constructed as:

   \[\text{<input type="checkbox" name="name1" value="value1">}\]

   \[\text{<br}>\]

   \[\text{<input type="checkbox" name="name1" value="value2">}\]
4. **RADIO BUTTON**: is a more complex version of a check box, allowing only one of a related set to be chosen. More than one radio button can be grouped together by using the NAME attribute; this keeps all buttons in the same group under one NAME. The same attributes in the CHECK BOX field can be used with the RADIO BUTTON.

RADIO BUTTONS are constructed as:

<input type="radio" name="any name" value="any value">.

5. **SUBMIT BUTTON**: displays a button with the preset function of sending the data in the form to the server for processing. The VALUE attribute with SUBMIT is used to provide text other than the default value which is “Submit Query” for the button. The layout of SUBMIT BUTTON is as:

<input type="submit" value="any text">.
2.1.4 Cascading Style Sheets (CSS)

In 1995, the World Wide Web Consortium (W3C) started publicizing a work-in-progress called Cascading Style Sheets (CSS). By 1996, it had become a full recommendation, with the same weight as HTML itself [16]. Style sheets are the way
web developers manage the overall presentation of their web sites fonts, colours, and so on from a single page to huge collections of documents.

From its earliest origins, HTML focused on content over style. Developers are encouraged to worry about providing high quality information, and leave it to the browser to worry about presentation. However, while use of the HTML `<font>` tag and related attributes like colour produce accurate presentation effects; style sheets, when applied bring consistency and order to whole document collections, as well as to individual documents.

Style sheets extend that presentation with several additional effects, including colours and a wider selection of fonts. But most importantly, style sheets let the developer control the presentation attributes for all the tags in a document. Users have a great degree of control over presentation; depending on their browser and the web site, a user may choose from various style sheets provided by the designers, may remove all added style and view the site using their own settings. CSS text-indent is a great way to indent paragraphs without having to use preformatted HTML tags, `<pre>`, or inserting spaces manually (`&nbsp;`). It is possible to define indentation with exact values or percentages [17]. Here is an example of paragraph indentation using CSS Listing 2.2:
Web Technologies

Listing 2.2 CSS example

Figure 2.2 shows the output of the above example on the browser.

```html
<html>
<style>
P { text-indent: 20px; }
H5 { text-indent: 30%; }
</style>
<body>
  <p>This is a paragraph that uses a text indentation with the value of 20px. This is the recommended usage of text indentation.</p>
  <h5>This is a paragraph that uses a text indentation with the value of 30%.</h5>
</body>
</html>
```

Figure 2.2 Browser output of CSS example

CSS uses rules to say how a document should appear. These rules usually live in a separate document rather than in the page with the content, thus keeping the presentation rules separate from the structural and semantic markup.

Each CSS rule is made up of two parts:

- **A selector** to indicate which elements a rule applies to.

- **Declarations** indicating the properties of an element you want to change, such as its typeface or color, and the value you want this to be, such as Arial or red. Declarations are very similar to attribute names and their values as shown on Figure 2.3.
Style sheets can be included to documents in a few ways, each with its own benefits and some drawbacks:

- **Inline Styles**

CSS declarations can be included within the optional style attribute of each element in HTML. Inline styles are not constructed as rules, and there is no selector because the properties and values are attached directly to the element. An inline style is the most specific of all because it applies to exactly one element and no others as in Listing 2.3.

```
<h2 style="color: red;">Inline Styles</h2>
<p style="color: gray;">This is an example of inline style!</p>
```

Listing 2.3 Example of Inline Styles

Using inline styles should be avoided because they mix presentation with the structural markup. They are also highly redundant, forcing you to declare the same style properties again and again to maintain consistent presentation.
• **Embedded Style Sheets**

The style rules can be impeded within the head element of the document, and those rules will be used only for the document in which they reside. An embedded style sheet is contained within the style element, as shown in Listing 2.4.

```html
<html>
  <head>
    <title>CSS Example</title>
    <style type="text/css">
      h2 { color: red; }
      p { color: gray; }
    </style>
  </head>
  <body>
    <h2>Embedded Style Sheets</h2>
    <p>This is an example of embedded style sheets!</p>
  </body>
</html>
```

**Listing 2.4 Example of an Embedded Style Sheet**

Embedding a style sheet in the head of a document does further separate presentation from a structured content, and those rules will be applied throughout that document, but it is not an efficient approach if styling more than one page at a time. Other documents within the same website would require embedded style sheets of their own, so making any future modifications to the site’s presentation would require updating every single document in the site.

• **External Style Sheet**

The third and best option is to place all the CSS rules in a separate, external style sheet, directly connected to the documents.
An external style sheet is a plain-text file that can be edited using the same text editor that created the HTML documents, saved with the file extension css. This approach completely separates presentation from content and structure they are not even stored in the same file. A single external style sheet can be linked from and associated with any number of HTML documents, allowing the entire website’s visual design to be controlled from one central file.

An HTML document links to an external style sheet via a link element in the document’s head. Listing 2.5 shows a simple example:

```html
<html>
<head>
<title>CSS Example</title>
<link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<body>
<h2>External style sheets</h2>
<p>This is an example of external style sheet!</p>
</body>
</html>
```

**Listing 2.5 Linking to an External Style Sheet**

When a browser downloads and begins processing the document, it will follow that link to retrieve the external style sheet and process it as well, automatically following its rules to render the page. An external style sheet is downloaded only once and then cached in the browser’s memory for use on subsequent pages. In order for this to happen the reference to the style sheet should be absolute not relative URL.
2.2 Client-side processing

Client-side Web page processing is achievable by downloading compiled programs, installing and executing them on the client workstation or by writing scripts within the HTML Web page commands interpreted by the client browser. There are two kinds of client-side processing:

1- Client-side scripts for example JavaScript[18], VBScript [19] and Jscript [20].
2- Compiled programs on the client workstation e.g. Java applets [21] and ActiveX controls [22].

2.2.1 JavaScript

JavaScript is the most commonly used client-side scripting language and is supported by most browsers. It uses loosely typed variables and is designed to work mainly within Web browsers. JavaScript allows embedding commands in an HTML Web page; when a user downloads the page, JavaScript commands will be evaluated. These commands can be triggered when the user clicks page items, manipulates fields in an HTML form. Although JavaScript cannot be used by a Web page to interact with a remote database, it is often used to validate user inputs entered on HTML forms submitted for processing by a server-side program. JavaScript’s main advantage is that it lets the page react dynamically to the changes of the browser without needing to access the server.
A recent development in Web applications using JavaScript, AJAX stands for Asynchronous JavaScript And XML [23]. AJAX uses JavaScript to send and receive data between a web browser and a web server. The AJAX technique makes web pages more responsive by exchanging data with the web server behind the scenes, instead of reloading an entire web page each time a user makes a change. The server responses that AJAX uses can be programmed in any server side language and that could include PHP. AJAX applications are theoretically browser and platform independent. In practise this depends on how standard the JavaScript is, and the browsers ability to parse XML. AJAX works in most modern browsers and does not require any proprietary software or hardware. Though originally considered an acronym for Asynchronous JavaScript + XML, the term is now used simply to encompass all the technologies that allow a browser to communicate with the server without refreshing the current page.

With AJAX, only the necessary data is transferred back and forth between the client and the web server see Figure 2.4. This minimizes the network utilization and processing on the client.

In the AJAX-enabled scenario, the web application can validate the entered data by making server calls in the background, while the user keeps typing. For example, after the user selects a country, the web browser calls the server to load; on the fly; the list of cities for that country, without interrupting the user from his or her current activity.
Figure 2.4 Comparison between traditional Web application and Ajax Web application models.

In an AJAX application, only the user interface elements that contain new information will be updated. The rest of the user interface should be unchanged. This means that you do not have to send as much information down the line, and you are not left waiting for a response because the previous page is already operational. This model enables continuous operation of a web page, and it also means that work done on the page does not have to follow a straight, predictable pattern.

In an AJAX application, the server can leave a notification when it’s ready, and the client will pick it up when it wants to. Or, the client can poll the server at regular intervals to see if the server’s ready, but it can continue with other operations in the meantime. As a result, the user can continue to use the application while the client requests information from the server in the background. When the new data turns up, only the related parts of the user interface need to be updated.
2.3 Server-side processing

With server-side processing, the Web server receives the dynamic Web page request, performs all the necessary operations to create the page, and then sends it to the client for display in the client’s browser.

The Web server is responsible for handling user input, it starts a program that processes it or just passes it to an already running program, gets the results and sends them back to the user. Server-side processing offers several advantages over client-side technologies, among them:

- Minimizes network traffic by limiting the need for the browser and server back and forth to each other.
- Makes for quicker loading time since, in the end, we’re only actually downloading a page of HTML.
- Avoids browser-compatibility problems.
- Can provide the client with data that does not reside at the client.
- Provides improved security measures, since we can code things that can never be viewed from the browser.

There are two main styles of server-side processing:

1- Compiled programs such as CGI scripts[24], Java Servlets[25] and ASP.NET[26].

2- Server-side scripts such as Hypertext Pre-Processor (PHP), Perl and Active Server Pages (ASP).
This section reviews the most commonly used server-side technologies that are used in developing Web applications and the main focus will be on PHP chosen to implement our prototype system.

2.3.1 CGI

The Common Gateway Interface (CGI) provides a standard interface between the web server and external applications. These applications are referred to as gateway programs, CGI programs, and CGI scripts. CGI programs are the code that accepts data from the Web server and does some processing with that data. They provide a common environment and a set of protocols for external applications to use while interfacing with the Web server. This extends the range of functions the Web server has including features provided by a potentially limitless number of external applications. CGI scripts can be written in any language supported on the Web server host machine. Some of the widely-used languages are Perl, C, C++, BASIC and for simple tasks even shell scripts. CGI is a generic interface for calling external programs to query databases, generate customized graphics, or perform any other server-side task. Despite the fact it has a simple interface and supported by almost every Web server; it has many disadvantages such as:

- It is fairly slow and can require a fair knowledge of the operating system and the programming language.
• There is a new process for every new request.
• Session tracking is not supported in CGI programs.

2.3.2 **FastCGI**

FastCGI is a fast, open, and secure Web server interface that solves the performance problems inherent in CGI without introducing any of the new problems associated with writing applications to lower-level Web server APIs [27]. FastCGI’s main aim is to minimize the cost of migrating CGI applications, to support both single-threaded and multi-threaded application programming, to generalize the roles that gateway applications can play beyond CGI’s "responder" role, and to allow a server to handle more web page requests at once. FastCGI processes are persistent; they need to be started prior to serving any requests, and after finishing one, they wait for a new request instead of terminating. A process manager is needed to control all processes created by FastCGI. This makes FastCGI applications difficult to develop, because special directives need to be included in the web server configuration files.

2.3.3 **PERL**

PERL (Practical Extraction and Report Language) is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, Web development, network programming, and GUI development[28].
Web Technologies

The language has support for multiple programming models including (procedural, object-oriented, and functional styles), built-in support for text processing, and a large collection of modules.

Perl is open-source software, supported by most of the operating systems. It has the following features:

- Perl is popular because of its text manipulation capabilities and support for rapid development cycle.
- Perl’s database integration interface (DBI) supports different databases platforms including Oracle, Sybase, PostgreSQL, MySQL and others, which makes Web-database integration easy.
- Perl has good libraries for working with HTML, XML.
- Perl's CGI module, part of Perl's standard distribution, makes handling HTML forms simple.
- The Perl interpreter can be embedded into other systems.

In spite of these features Perl has grown a complex structure of additions and extensions. The format and syntax of Perl can make a Perl script hard to read and modify.

2.3.4 **Java Servlets**

Servlets are programs written in Java that extend the functionality of the application server in which they run [25]. Writing Web applications with servlets has
many advantages. For every request, a new process is generated to handle it. A servlet can include every Java class, so standardized access to databases via JDBC and several XML parsers can be used by the developer. Another advantage of servlets is that they provide classes and methods devoted to both user-space and server-space session tracking. Also since they are written in Java; servlets are widely portable. Servlets are supported by most commercial web development tools. Moreover, servlets have access to all of the benefits of the Java platform itself, including reusable class libraries and platform independence. Even with the services provided by the Java Servlet API, however, the platform has a drawback in terms of convenience and maintainability when it is used to produce dynamic web pages.

2.3.5 Java Server Pages (JSP)

Java Server Pages, JSP, for short is a Java-based technology that simplifies the process of developing dynamic web sites [9]. With JSP, designers and developers can quickly incorporate dynamic content into web sites using Java and simple markup tags. The JSP platform lets the developer access data and Java business logic without having to master the complexities of Java application development. JSP helps effectively separate presentation from content by applying the Model-View-Controller (MVC) design pattern [29].
When a JSP page is requested by a web browser, the page causes code to run and decide, on the fly, what content to send back to the browser. Such dynamic content allows for the construction of large and complex web applications that interact with users.

JSP is a simple way to mix Java code and HTML text to produce dynamic web pages. It helps separate Java code from presentation text, giving nonprogrammers a way to produce functional and dynamic web pages. JSP is not even limited to the production of dynamic HTML-based content. For instance, it can be used in wireless and voice-driven applications [30].

The browser submits requests to a servlet, which accesses business objects to create content. That content is stored in a Java bean, which is accessed by a JSP page. That JSP page subsequently presents the content, typically in HTML see Figure 2.5.

Separating content generation from presentation is beneficial because Java code, for the most part, is restricted to content generation.
This encapsulation of Java code lets software developers concentrate on servlets and business objects and lets web page authors focus on corresponding JSP pages.

In this MVC architecture, where business objects represent the model (data), servlets represent controllers (which handle requests), and JSP pages are views of the model. The MVC architecture, which originated in Smalltalk back in the stone age of computing [26], has stood the test of time because it separates business and presentation logic. That separation allows for pluggable components, resulting in flexible, reusable, and adaptable software.

MVC encapsulates three general abstractions that are presented in most graphical applications: models, views, and controllers. By encapsulating what other architectures intertwine, MVC applications are much more flexible and reusable than their traditional counterparts.

JSP extends the Servlet platform and provides most of the advantages of servlets. Like servlets, JSP pages have access to the full range of Java code’s capabilities. JSP also provides several standard HTML-like tags to access back-end logic, and it lets developers design new tags.

2.3.6 Active Server Pages (ASP)

ASP is Microsoft’s server-side script engine for generating Web pages dynamically and is supported by Microsoft’s Web server and the Internet Information Server (IIS) [31]. ASP scripts can connect to any ODBC-compliant database.
ASP files are text files that can contain plain text, HTML tags and script blocks which are executed on the server.

ASP is a powerful technology, allowing you to do any of the following:

1. Display HTML pages on the fly.
2. Process and store data submitted through HTML forms.
4. Retrieve/Edit/Store data from databases like MS SQL Server, MySQL, PostgreSQL, MS Access and Oracle.

Since ASP is implemented as an ISAPI (Internet Server API) application integrated into Microsoft Information Server (IIS), which implies that ASP technology is basically restricted to Microsoft Windows based platform. However, ASP technology can be targeted to other platforms via third-party porting products.

2.4 PHP

2.4.1 Introduction

PHP is an open-source, cross-platform and server scripting language for writing web-based applications, and as such it has the potential to change faster; individuals can participate in the development of the language. Many web programmers select PHP because a number of free, open-source portal and bulletin board software modules are available.
PHP allows the embedding of program logic in HTML pages, which enables serving dynamic web pages [32]. It has a structure like JSP/ASP with no complex set up, ease of syntax, which is a mix of Perl, C++, and Java. It is un-typed scripting language like PERL which leads to ease of use and wide adoption. Although PHP was originally designed for small sites, it’s now striving to grow and achieve speed and efficiency. It is used mainly in server-side application software, but can be used from a command line interface or in standalone graphical applications. PHP is free and open source software, runs in all major Web server platforms, and is available for many different operating systems like Unix, Windows and Mac OSX, and for a number of web servers, including Apache and IIS. It has a rapid development cycle, supportive community, a simple syntax, and can be easily extended.

2.4.2 PHP history

PHP/FI

PHP/FI, an acronym of Personal Home Page / Forms Interpreter was created by Rasmus Lerdorf in 1995, primarily as an uncomplicated set of Perl scripts for tracking accesses to his online resume [8]. As more functionality was required, Rasmus wrote a much larger C implementation, which allowed users to develop simple dynamic Web applications and access databases.
PHP 2

In 1996 PHP version 2 was released which had the basic functionality that PHP has today. This version included the ability to embed HTML and form processing. The syntax was similar to Perl but was more limited, simpler, and less consistent.

PHP 3

PHP 3.0 was named plain 'PHP' in 1997, with the meaning being a recursive acronym - PHP: Hypertext Preprocessor. It was the first version that closely resembles PHP as we know it today. It provided end users with a concrete infrastructure for lots of different databases, protocols and APIs, PHP 3.0's extensibility features attracted dozens of developers to join in and submit new extension modules. The other features introduced in PHP 3.0 were the object oriented syntax support and the more powerful and consistent language syntax.

PHP 4

PHP 4 released in 2000, its design goals were to improve performance of complex applications, and improve the modularity of PHP's code base. It was based on a new engine, named ‘Zend Engine’ [33]. In addition to the highly improved performance of this version, PHP 4 included other key features such as support for many more Web servers, HTTP sessions, more secure ways of handling user input and several new language constructs. PHP 4 provides improved efficiency over PHP 3 by compiling
scripts before executing them. It comes with a rich function library that includes support for accessing mail services, directories, and databases.

**PHP 5**

In 2004, PHP 5 powered by the new Zend Engine II included new features such as, numerous object oriented functions such as explicit constructors and destructors, object cloning, class abstraction, and variable scope. It supports try/catch exception handling which offers an excellent means for standardizing error-reporting logic. Improved XML and Web Services support

**PHP 6**

The big change in PHP 6 is support for Unicode, which is to say that PHP can now handle characters in many different languages in the world [34]. Beyond Unicode support, PHP 6 cleans up a lot of garbage that was left in PHP 5 even though the recommendation was not to use such things. The two biggest removals are the “Magic Quotes” and “register globals” features.

2.4.3 **PHP General Features**

* Practicality

A useful PHP script can consist of as little as one line; unlike other languages such as C, there is no need for the mandatory inclusion of libraries. The libraries in PHP are automatically found as long as they have been included in the PHP build on the Web server [35].
For example, the following represents a complete PHP script, the purpose of which is to output the current date, in this case one formatted like April 2, 2009:

```php
<?php echo date("F j, Y");?>
```

PHP is a loosely typed language, meaning there is no need to explicitly create, typecast, or destroy a variable, although you are not prevented from doing so. PHP handles such matters internally, creating variables on the fly as they are called in a script, and employing a best-guess formula for automatically typecasting variables. PHP will also automatically destroy variables and return resources to the system when the script completes.

- **Power**

PHP major versions were accompanied by enormous additions to PHP’s default libraries. PHP can do the following:

- Create and manipulate Macromedia Flash, image, and Portable Document Format (PDF) files
- Communicate with the Lightweight Directory Access Protocol (LDAP)
- Parse even the most complex of strings using both the POSIX and Perl-based regular expression libraries.
- Communicate with a wide variety of protocols, including IMAP, POP3, NNTP, and DNS, among others.
• **Possibility**

PHP developers are rarely bound to any single implementation solution. It offers users of differing skill set the opportunity not only to immediately begin performing complex string operations but also to quickly port programs of similar functionality (such as Perl and Python) over to PHP [36]. PHP offers comprehensive support for functional programming and the object-oriented paradigm.

• **Price**

Since its beginning, PHP has no usage, modification, and redistribution restrictions. Open-source software users are freed of the vast majority of licensing restrictions that are used in the commercial software, users are largely free to modify, redistribute, and integrate the software into other products.

Because the source code is freely available for anyone to examine, security holes and potential problems are rapidly found and fixed.

2.4.4 **PHP Syntax**

PHP code is saved as plain text in ASCII format, so we can write our PHP pages in almost any text editor. PHP code is script code embedded in an HTML page, which is executed on the server before being sent to the browser. When using PHP the server actions are as follows:

1. Read the request from the browser.

2. Find the page in the server.

3. Perform any instructions provided in PHP to modify the page.
4. Send the page back across the Internet to the browser.

You add PHP code to a Web page by using tags, similar, but not identical, to other tags in the HTML file. The PHP code section is enclosed between `<? PHP` and `?>` tags with the following form:

```
<HTML>
<html>
<?php
 echo "Text generated by PHP";
?>
</html>
</HTML>
```

All statements between the two PHP tags are passed to PHP by the Web server and are processed by the PHP preprocessor. After processing, the PHP section is discarded. If the PHP statements produce output, the output is sent back to the Web server, which then sends the HTML and the output from the PHP sections to the browser. The browser does not see the PHP section, only its output (if there is any output).

PHP can produce various output forms other than HTML/XHTML e.g. pdf, images, and XML outputs like SVG.

When the Web server gets the file and sees the .php extension, it checks for PHP tags. When it finds the PHP tag, it executes the PHP echo statement instead of sending it to the browser. The echo statement used to output everything that is between the double quotes ("}). Only the output from the PHP section, which is *Text generated by PHP*, is sent on to the browser. In the browser window, you see the output at the location in the
page where you added the PHP section. Even if you view the source in your browser, you only see the output, not the PHP code see Figure 2.6.

![Diagram of PHP interaction process](image)

**Figure 2.6** PHP interaction process captured from [33]

PHP has several built-in arrays called super global arrays, which are accessible from anywhere within the executing script and provide a substantial amount of environment specific information. With these arrays it is possible to retrieve details about the current user session, the user’s operating environment, the local operating environment, and more. PHP creates some of the arrays, while the availability and value of many of the other arrays are specific to the operating system and Web server.
They all begin with a dollar sign followed by an underscore e.g. \$_POST and \$_GET. They contain information passed from forms through the post and get methods, respectively. The super globals are all associative arrays, and the keys of \$_POST and \$_GET are automatically derived from the names of form elements. For example if you have a text input field called name in a form; PHP automatically creates an array element called \$_POST['name'] when the form is submitted by the post method or \$_GET['name'] if the get method is used.

2.4.5 PHP data types

Even though PHP is weakly typed, it uses the following eight data types:

**Integer**: This is a whole number, such as 1, 25, 42, or 2006. Integers must not contain any commas or other punctuation as thousand-separators. You can also use hexadecimal numbers, which should be preceded by 0x (e.g., 0xFFFFFF, 0x000000).

**Float or Double**: This is a number that contains a decimal point, such as 9.99. Like integers, floating-point numbers must not contain thousand separators.

**String**: A string is text of any length. It can be as short as zero characters (an empty string), and it has no upper limit.

**Boolean**: This type has only two values: true or false.

**Array**: An array is a variable that is capable of storing multiple values, although it may contain none at all (an empty array). Arrays can hold any data type, including other arrays.
Web Technologies

**Object:** PHP has powerful object-oriented capabilities.

**Resource:** When PHP connects to an external data source, such as a file or database, it stores a reference to it as a resource. Typically this interaction takes place through *handles*, which are named at the time a connection to that resource is successfully initiated. These handles remain the main point of reference for that resource until communication is completed, at which time the handle is destroyed. These handles are of the resource data type. Examples of such functions include fopen(), pg_connect(), and pdf_new(). Variables of type resource do not actually hold a value; rather, they hold a pointer to the opened resource connection.

**Null:** This is a special data type that indicates that a variable has no value. This data type recognizes only one value, Null.

2.4.6 **PHP Libraries**

PHP’s great flexibility and power depend greatly on its hundreds of functions, which consist of built-in functions and functions available in a PHP extension [37]. Many functions are available in PHP through the use of extensions.

Most PHP functions are contained in the PHP extensions. Several core extensions are compiled into PHP and are always there by default. Other standard extensions are included in the PHP distribution, but you must add them to PHP before you can use their functions.
In addition, many extensions are written and made available by individuals. Many of these are part of PEAR, the PHP Extension and Application Repository [9]. There are a range number of libraries for different sorts of applications, some of these libraries are mentioned below:

**GD library**: It provides functions so you can output graphics, as well as HTML. You can create and manipulate images in several formats, such as JPEG, GIF, PNG, and others.

**PDF extension**: This is a library of functions that are used to create a PDF document (rather than HTML). You can set fonts, write text to the document, and add graphics.

**cURL**: This library allows communication with many kinds of servers, such as HTTPS, Telnet, FTP, LDAP, and others.

### 2.5 PHP and Database

PHP is particularly strong in its ability to interact with databases. PHP provides support for the most widely used databases systems [38] such as PostgreSQL, MySQL, Oracle, MsSQL, Informix and Sybase.

In addition, PHP supports ODBC (Open Database Connectivity), a standard that allows you to communicate with even more databases, such as Access and IBM DB2.

PHP simplifies connecting to the database and communicating with it. It connects to the database, passes your instructions to the database, and returns the database response to you.
PHP can create web-based user interfaces that interact with most of database systems such as MySQL, Oracle and PostgreSQL [39].

Before anyone can interact with the database, they must first open a connection to it. Each connection is represented by a single variable (connection handle). In PHP multiple connections can be open at once, each with its own handle.

PHP also supports persistent database connections. Persistent connections stay open beyond the life time of the page request. PHP maintains a list of currently open connections and if a request is made for a new database connection with the same connection parameters as one of the open connections in this list, a handle to the already opened connection is returned instead. We illustrate PHP database connectivity with examples using PostgreSQL [40].

**Establishing and Closing a Connection**

Before interacting with the PostgreSQL server, you need to successfully connect to it and choose a database, passing along any necessary credentials. Likewise, once you’ve finished using the database, you should close the connection to recover system resources.

This section shows you how to establish a new connection, choose a database, and subsequently close the connection.

**pg_connect()**

```php
pg_connect(string connection_string [, int connect_type])
```
PostgreSQL requires parameters to be submitted as a single string, denoted by `connection_string`. Several individual parameters are recognized in this string, including:

- `dbname`: The name of the database to connect to.
- `host`: The server location as defined by a hostname, such as localhost.
- `password`: The connecting user’s password.
- `port`: The port on which the server operates. By default, this is 5432; therefore, you need to specify this parameter only if the destination server is operating on another port.
- `user`: The connecting user.

For example, to connect to a localhost database named `test` using user `name` that is assigned a password of `secret`, the following command would be used:

```shell
$con = pg_connect("host=localhost user=name password=secret dbname=test");
```

Because PostgreSQL by default pools its connections, to save system resources, if a subsequent connection request is made within the same script using the same parameters as those used to initiate an already-open connection, PostgreSQL uses the existing connection rather than opening a new one.

You can override this behavior with the optional `connect_type` parameter, by passing in the value `PGSQL_CONNECT_FORCE_NEW`.

**pg_pconnect()**

```php
pg_pconnect(string connection_string [, int connect_type])
```

The `pg_pconnect()` function operates identically to `pg_connect()` in every way, supporting all of the parameters described previously, except that using it will result in
the connection remaining open even after the script completes execution. If a subsequent attempt is made to open a new connection consisting of the same connection parameters as those used in the original connection, then that persistent connection will be reused.

**pg_close()**

`pg_close([resource connection])`

Although database connections opened during the execution of a script are automatically closed once the script completes, rigorous programming practice is always encouraged by explicitly closing such resources once they are no longer needed.

**pg_query()**

`pg_query([resource connection,] string query)`

The `pg_query()` function is responsible for sending the query to the selected database, returning a result resource on success, and FALSE otherwise.

While the `pg_query()` function does indeed execute the query, you cannot do anything with the result without using other functions.

When executing SELECT statements, you might use `pg_fetch_row()`, `pg_fetch_array()`, `pg_fetch_object()`. To find the rows returned by a SELECT you could use `pg_num_rows()`. For INSERT, UPDATE, or DELETE statements, you can use `pg_affected_rows()` as alternative to `pg_num_rows()`.

**pg_insert()**

`pg_insert(resource conn, string table, array assoc_array [, int options])`
The pg_insert() function inserts the values found in assoc_array into the table specified by table. The number of values found in assoc_array should equal the number of columns in table or be acceptable because the database can supply other values e.g. defaults or serials.

**pg_update()**

\[pg\_update(resource\ conn, string\ table, array\ data, array\ conditions [, int\ options])\]

The pg_update() function modifies those rows located in the table named table according to the conditions specified by the conditions array, updating the row per the keys and corresponding values specified by data.

**pg_delete()**

\[pg\_delete(resource\ conn, string\ table, array\ assoc\_array [, int\ options])\]

The pg_delete() function deletes the rows in table where the column values equal those found in the array assoc_array.
Chapter 3
Database Development

3.0 Introduction

Databases and database technology are having a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, law, education, and library science, to name a few. A database is a collection of related data organized for quick search and retrieval. By data, we mean known facts that can be recorded and that have implicit meaning. However, the common use of the term database is usually more restricted. A database has the following implicit properties:

• A database represents some aspect of the real world;

• A database is a logically coherent collection of data with some inherent meaning;

• A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

This chapter is structured to introduce the basic principles of Relational Databases (3.1), SQL (3.2), the general principles of database Metadata (3.3), and finally programmatic access to databases (3.4) and more independent access via Data Abstraction Layer (DAL) (3.5).
3.1 Relational Database

There are a number of ways databases have been implemented over the past two decades, but they fall into several basic categories. In general the power and flexibility of a database are directly proportional to the complexity of developing and implementing it. Databases provide a direct way of tapping the knowledge that already exists in an organization and providing it to interested parties with the click of a mouse.

Databases come in many shapes and sizes:

- A hierarchical database
- A relational database
- An object-oriented database
- A hybrid database

All of these different types of databases are in use for a variety of purposes; it is most common to use relational databases for Web applications due to their wide acceptance, stability and speed.

A relational database is a database that is perceived by the user as a collection of tables, where a table is an unordered collection of rows [41]. Each row has a fixed number of fields, and each field can store a predefined type of data value, such as an integer, date, or string.

Relational databases are the most common DBMS. A key characteristic of a relational database is the absolute separation between physical and logical data.
Data is accessed through the associated logical model to avoid supplying physical storage locations and to reduce the limitations imposed by using physical information. Relational databases allow the definition of relationships and integrity rules between data sets. The main focus of the relational database design is that related information should be stored either in the same place or in a separate place that is related to the original in some meaningful way. It is also designed around the principle that data should not be duplicated within the database.

### 3.2 SQL Overview

SQL stands for **Structured Query Language**; it is a query language that interacts with a DBMS [3]. It allows data retrieval as sets of records, performing of complex computations on the data and data access without supplying physical access plans. SQL is not a complete programming language usable to build complex applications. It is commonly used within a host language that offers specific features for building complete applications. It enables data definition, manipulation and management, access protection, and transaction control. SQL handles many relational database objects, including tables, indexes, keys, rows, and columns. The American National Standards Institute (ANSI) in 1986 released a formal standard named *SQL-86*. ANSI updated that standard in 1989 to *SQL-89* and again in 1992 to *SQL-92*. The *SQL-92*, addresses modern environment needs. It contains new features such as support for dynamic SQL and for an advanced technique to access result sets called scrollable cursors.
In 1999 SQL-99 added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features. In their next standard version SQL: 2003, ANSI introduced XML-related features, window functions, standardized sequences [42]. The ANSI standard defines three programmatic interfaces to SQL:

- **Modules**: Separate compiled modules may define procedures and then call them from a traditional programming language.

- **Embedded SQL Statements**: SQL statements are embedded directly into the program source code and mixed with the language statements.

- **Application Program Interface (API)**: this method provides the programmer with a standard set of functions that can be invoked from the software.

While embedded SQL was the most popular choice a few years ago, it is not the best answer to the problem of querying databases in client-server environments. It is static in all senses of the term, and this limitation makes it unsuitable for newer software architectures.

SQL language is usable for a variety of purposes, including:

- Querying a database by entering SQL text directly
- Querying a database within a program
- Defining data organization
- Administering data
- Accessing multiple data servers
- Managing transactions
SQL is a language designed to use relations to transform inputs into required outputs. It has two major components:

- Data Definition Language (DDL) for defining the database structure.
- Data Manipulation Language (DML) for retrieving and updating data.

The SQL language supports a set of commands used to define, store, manipulate, and retrieve data [43]. The following are the basic SQL commands used to build SQL clauses for data manipulation:

- **SELECT** To query data in the database.
- **INSERT** To insert data into a table.
- **UPDATE** To update data in a table.
- **DELETE** To delete data from a table.

### 3.3 Database Metadata

A fundamental characteristic of a relational database is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.

This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called metadata, and it describes the structure of the primary database [44].
The catalog is used by the DBMS’s own software and also by database users who need information about the database structure. A general-purpose DBMS software package is not written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with any number of database applications for example, a university database, a banking database, or a company database-as long as the database definition is stored in the catalog.

DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions.

Database metadata is high-level information, or "data about data," stored within a database describing that database. This information is extremely useful for dynamically building SQL statements or even generating dynamic views of the database contents.

The metadata stored by a database, and the way in which it's stored, varies widely between different database systems.

A relational database must provide access to its structure through the same tools that are used to access the data. Each database includes a set of system catalog tables, which describe the logical and physical structure of the data [45].

These tables contain information about the definitions of database object such as user tables, views and indexes. They are created when the database is created, can be queried by any user but cannot be explicitly created or updated. Some DBMS distinguish the system tables by giving them a special prefix like “pg_” for PostgreSQL system tables.
There are two common problems when trying to query the system tables directly, they can be complex and difficult to query, the queries are not portable to other types of database, and not guaranteed to remain constant over version update within one database.

Since the Information Schema is defined as views in the standard, a DBMS can use the Information Schema directly or map its own System Tables to the Information Schema definitions [46]. It is defined in the SQL standard and can therefore be expected to be portable and remain stable unlike the system catalogs, which are specific to a particular RDBMS e.g. PostgreSQL and are modelled after implementation concerns. The Information Schema views do not, however, contain information about DBMS-specific features for example to inquire about PostgreSQL-specific features you need to query the system catalogs or other PostgreSQL-specific views.

The Information Schema consists of a set of views, exposing metadata in relational format. This allows the execution of SELECT statements to retrieve or to format metadata. We are allowed to query the Information Schema, but we are not allowed to change its structure or modify its data see Table 3.1.
The Information Schema is written on top of System Catalogs and it is generally more readable than System Catalogs.

For example a query that will return the names of the fields of a table using System Catalogs in PostgreSQL:

```sql
SELECT a.attname from pg_class c, pg_attribute a, pg_type t
WHERE c.relname=' table name '
  AND a.attnum > 0
  AND a.attrelid = c.oid
  AND a.atttypid = t.oid
```
Using Information Schema:

```sql
SELECT column_name
FROM information_schema.columns
WHERE table_name = 'table name'
```

### 3.4 Database-independent programming interfaces

#### 3.4.1 Introduction

In addition to cheaper and more powerful computer hardware, free database packages have become available, such as MySQL [47] and PostgreSQL [48]. These packages give all computer users the ability to use powerful database technology in their everyday lives.

This trend towards mixing and matching database technology has some important downsides. Having replaced a centralized database with a cluster of workstations and multiple database types, companies are now faced with hiring skilled administration staff or training their existing administration staff for new skills. In addition, administrators now need to learn how to glue different databases together.

It is in this climate that a new order of software engineering has evolved, namely database independent programming interfaces. Therefore, the task of coordinating and porting software has rapidly gone from being relatively straightforward to extremely difficult.
Database-independent programming interfaces help developers by giving them a single, unified interface with which they can program. This shields the developer from having to know which database type they are working with, and allows software written for one database type to be ported far more easily to another database. For example, software written for PostgreSQL will generally work on Oracle with little modification.

Database-independent programming interfaces help not only developers. Administrators can also use them to write database-monitoring and administration software quickly and portably, increasing their own efficiency and the efficiency of the systems and databases they are responsible for monitoring.

Databases are becoming more important in the corporate landscape, and powerful interfaces are required to stop these resources from flying apart and becoming disparate fragments of localized data. This joining process can be aided by the use of database-independent programming interfaces, such as the JDBC and DBI, especially when used in conjunction with efficient high-level data-processing languages such as Java and Perl.

3.4.2 JDBC

Java Database Connectivity (JDBC) is an API used for connecting to relational databases and other data sources from Java [49]. It provides methods for querying and updating data in a database. JDBC achieves its goals through a set of Java interfaces, each implemented differently by individual vendors.
The set of classes that employ the JDBC interfaces for a particular database system is called a JDBC driver.

Hence, a Java program with JDBC function calls can access any RDBMS that has a JDBC driver available.

Because Java is object-oriented, its function libraries are implemented as classes. Before being able to process JDBC function calls with Java, it is necessary to import the JDBC class libraries, which are called java.sql. These are sometimes called the data sources accessed by the Java program. These data sources could be stored using RDBMSs from different vendors and could reside on different machines. Hence, different data source accesses within the same Java program may require JDBC drivers from different vendors.

The JDBC interface provides the application with a set of methods that enable database connections, queries, and result retrievals [50]. It is the interface between specific database drivers and the Java user application, applet, or servlet. The functions a user application can call are methods of connection, statements, or results object classes.

### 3.4.2.1 JDBC Characteristics

- JDBC is a “call-level” SQL interface for Java [51]. This interface is totally independent of the available database management systems. It is a low-level application programming interface (API) that allows a Java program to issue SQL statements and retrieve their results.
It also provides methods for error and warning messages management.

- JDBC does not restrict the type of queries passed to an underlying DBMS driver. An application may use as much SQL functionality as desired. The underlying drivers are authorized to claim JDBC compliance on the condition they fully support ANSI SQL-92 Entry Level. SQL-92 Entry Level conformance is widely supported today and guarantees a wide level of portability.

- JDBC may be implemented on top of common SQL level APIs, in particular on top of ODBC.

- JDBC provides a Java interface that stays consistent with the rest of the Java system. Provides a cell defined bridge between the philosophies expressed by the impedance mismatch between the object-oriented world (Java in SQL).

- The JDBC mechanisms are simple to understand and use. This simplicity does not mean that functionality suffers.

- JDBC uses strong, static typing whenever possible. This approach allows for performing more error checking at compile time. However, it should not be a limitation to JDBC’s usage.

- One functionality, one method; this concept has been adopted, as opposed to many other DBMS SQL level APIs, to keep it simple yet powerful for the beginner as well as the experienced developer.
3.4.2.2 JDBC Components

The JDBC architecture is illustrated in Figure 3.1.

The following are JDBC components:

**Application:** The user application invokes JDBC methods to send SQL statements to the database and retrieve results. It performs these tasks:

- Requests a connection with a data source
- Sends SQL statements to the data source
- Defines storage areas and data types for the result sets
- Requests results
- Processes errors
- Controls transactions: requests commit or rollback operations
- Closes the connection

**Driver Manager:** Its primary purpose is to load specific drivers for the user application. It may also perform the following:

- Locate a driver for a particular database
- Process JDBC initialization calls
- Provide entry points to JDBC functions for each specific driver
- Perform parameter and sequence validation for JDBC calls

**Driver:** The driver processes JDBC methods invocations, sends SQL statements to a specific data source, and returns results back to the application.
When necessary, the driver translates and/or optimizes requests so the request conforms to the syntax supported by the specific DBMS. It will:

- Establish a connection to a data source
- Send requests to the data source
- Perform translations when requested by the user application
- Return results to the user application
- Format errors in standard JDBC error codes
- Manipulate cursors if necessary
- Initiate transactions, if explicitly required

**Data Source:** The data source consists of the data the user application wants to access and its associated parameters that is, the type of DBMS and network layer (if any) used to access the DBMS.
Figure 3.1 The JDBC architecture
3.4.2.3 JDBC driver types

**Proprietary Database Drivers:** They process JDBC calls and send SQL statements to the data source. They may be “native-API partly-Java” or “native-protocol all-Java.” A native-API driver forwards the calls to a locally installed library, usually developed in C and provided by the database vendor. It may be a Dynamic Link Library (DLL) or a so-called shared library. A native-protocol all-Java driver implements in Java all the layers necessary to communicate with the database. They are fully portable because they do not use local libraries or other native code.

**Bridge Drivers:** This driver creates a bridge between JDBC and another Call Level Interface (CLI). For example, the JDBC-ODBC Bridge is a bridge driver. It processes JDBC calls and, in turn, calls ODBC functions that will send SQL statements to the ODBC data source [52].

**DBMS-independent all-Java Net drivers:** These drivers use a DBMS-independent published network protocol. They are very portable because they are 100 percent Java. The JDBC interface defines the possible interactions between the user application and the driver manager.

3.4.3 PERL DBI

In their efforts to change from database specific interface to generic interface, the developers of Perl introduced the standard database interface for Perl DBI (DataBase Interface) [53].
DBI is a database-independent package that provides a consistent set of routines regardless of what database product in use. The design of DBI is to separate the DBD (DataBase Driver) modules from the programmer's API, so any DBI program can work with any database, or even with multiple databases by different vendors at the same time.

The DBI architecture as shown in Figure 3.2 is split into two main groups of software: the DBI itself, and the drivers [54]. The DBI defines the actual DBI programming interface; routes method calls to the appropriate drivers, and provides various support services to them. Specific drivers are implemented for each different type of database and actually perform the operations on the databases. All the information that passes between the DBI and its drivers is standard Perl data types, thereby preserving the isolation of the DBI module from any knowledge of databases. The separation of the drivers from the DBI itself makes the DBI a powerful programming interface that can be extended to support almost any database available today. Drivers currently exist for many popular databases including Oracle, Informix, mSQL, MySQL, Ingres, Sybase, DB2, PostgreSQL and others.

Drivers are also called database drivers, or DBDs, after the namespace in which they are declared. For example, Oracle uses DBD::Oracle, PostgreSQL DBD::pg, and so on.

The DBI defines three main types of objects that you may use to interact with databases. These objects are known as handles.
There are handles for drivers, which the DBI uses to create handles for database connections, which, in turn, can be used to create handles for individual database commands, known as statements.
Table 3.2 shows which metadata functions are available directly for each language. Some functions are missing for different languages for example in JDBC there is no direct function to get the number of rows in the result set like the one in PHP

\textit{pg\_num\_rows}.

But you can get the number of rows in a number of ways, if you are using a scrollable result set, \textit{rs}, you can call the methods \textit{rs\_last} and then \textit{rs\_getRow} to find out how many rows \textit{rs} has. If the result is not scrollable, you can either count the rows by iterating through the result set or get the number of rows by submitting a query with a \texttt{COUNT} column in the \texttt{SELECT} clause.

There is no direct function in PHP to get the primary key like the one available in JDBC through the \textit{getPrimaryKeys()} function. However in order to get the primary key in PHP is by using the \textit{pg\_meta\_data} function to get an array of the table metadata information and then fetch the array for the element that holds the primary key.
<table>
<thead>
<tr>
<th>Metadata function</th>
<th>Server info</th>
<th>Host info</th>
<th>DBNAME</th>
<th>User</th>
<th>Port</th>
<th>Schema</th>
<th>Tables</th>
<th>Column Metadata</th>
<th>DBMS</th>
<th>Database</th>
<th>Query/Statement</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JDBC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>number</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>DBI</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>type</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>ADODB</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>size</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>PHP / PostgreSQL</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>unique</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>PHP / MySQL</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>null</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 3.2 Table Metadata Functions
3.5 Metadata in APIs

Developing dynamic Web applications requires using database metadata in an efficient way. Metadata defined as data about data not only helps effectively managing resources, it also helps in finding the needed data and determines how to use it. In addition, metadata provides a structured description of database information resources and services.

As shown in Figure 3.3 the metadata architecture showing metadata retrieval using the API metadata functions at one level, but if the library is incomplete we can build it up with wrappers or use of queries on information schema and System catalogs.

![Figure 3.3 Metadata architecture](image-url)
3.5.1 **PHP and Metadata**

There are a number of PHP’s PostgreSQL functions supports retrieving database metadata. These functions can be divided into two groups:

**Resource Connection functions**

One of the most important functions is `pg_meta_data()` function, which returns an array containing information on each field name, type, length, default value, and support for NULL values.  

The following code snippet illustrates using this function:

```php
// make a connection
$dbh = pg_connect("host=localhost dbname=test user=postgres");

// get table information
$meta = pg_meta_data($dbh, "table-name");
```

There’s also a `pg_parameter_status()` function, which can be used to retrieve the current values of server variables, such as ‘max_connections’ or ‘server-version’.

Here’s an example of how you can use this function using the above connection handle `$dbh`:

```php
// get value of 'server_version' variable
echo "Server version: " . pg_parameter_status($dbh, 'server_version');
```
Resource Result functions

These functions work on the result resource, returned by `pg_query()`, `pg_query_params()` or `pg_execute()` (among others). When retrieving or modifying data, it is important to know how many rows were selected or modified. Two functions are available to perform these tasks, `pg_num_rows()` and `pg_affected_rows()`, respectively.

**pg_num_rows()**

This function returns the total number of rows found in the resource result, or -1 should an error occur. The following code illustrates how to use this function:

```php
$result = pg_query($query);
echo "Number of Rows: ".pg_num_rows($result);
```

This outputs the number of rows in the result set.

**pg_affected_rows()**

The `pg_affected_rows()` function returns the total number of rows affected by a DELETE, INSERT, or UPDATE query, retrieving this number from the resource result.

For example:

```php
$result = pg_query($query);
echo "Total number of affected rows: ".pg_rows_affected($result);
```

This returns the number of the affected rows in the result set.
Database Development

There are more functions for retrieving information about fields such as to name few:

\[ \text{pg\_field\_is\_null} \] to test if a field is SQL NULL.

\[ \text{pg\_field\_name} \] returns the name of a field.

\[ \text{pg\_field\_num} \] returns the field number of the named field.

\[ \text{pg\_field\_size} \] returns the internal storage size of the named field.

\[ \text{pg\_field\_type} \] returns the type name of a field.

\[ \text{pg\_field\_table} \] returns the name or oid of the tables field.

3.5.2 JDBC and metadata

JDBC allows clients to discover a large amount of metadata information about a database (including tables, views, columns, stored procedures, and so on) and any given ResultSet via metadata classes. Most of JDBC’s metadata consists of information about one of two things:

- **DatabaseMetaData** provides information about the database as a whole: table name, table indexes, database product name and version, and actions the database supports. The DatabaseMetaData interface provides methods for retrieving various metadata associated with a database. This includes enumerating the stored procedures in the database, the tables in the database, the schemas in the database, the valid table types, the valid catalogs, finding information on the columns in tables, access rights on columns, access rights on tables etc. The peace of code below illustrates the use of this interface.
DatabaseMetaData mtdt = Connection.getMetaData();

mtdt.getDatabaseProductVersion();

System.out.println("Driver name: " + mtdt.getDriverName());

JDBC will retrieve the driver name for example JDBC-ODBC Bridge.

- ResultSetMetaData gets information about the types and properties of the columns in a ResultSet object.

The following piece of code illustrates creating a ResultSetMetaData object.

    ResultSet rs = statement.executeQuery("SELECT * FROM customer");

    ResultSetMetaData rsmd = rs.getMetaData();

The ResultSetMetaData interface offers the following methods for finding out about a result set’s metadata.

- Information on ResultSet Columns
The ResultSetMetaData interface is a rich interface providing many methods for obtaining information about the columns of a result set. For example:

    getColumnCount(); : gets number of columns in a ResultSet.

    getColumnName(int column);: gets column’s name in a ResultSet.

- Column Properties
The ResultSetMetaData interface provides many methods for obtaining information about the properties of the columns of a result set such as:

    isNullable(int column);
3.6 Database Abstraction Layer (DAL)

3.6.1 Introduction

The Database Abstraction Layer functions as a wrapper around the code, exposing a set of generic methods to interact with a database server. Its first goal is to hide the complexity of the specific database implementation from the developer. It also centralizes the application’s access to database, which results in cleaner and manageable code [55]. Database abstraction layers serve to decouple the application logic from that used to communicate with the database. By passing all database-related commands through this generalized interface, it became possible for an application to use one of several database solutions, provided the database supported the features required by the application, and the abstraction layer offered a driver compatible with that database. A graphical depiction of this process is found in Figure 3.4. Most database abstraction layers incorporate a generic set of data types, which get converted into native data types of the selected database system.

PHP's database access functions are not standardised. Every database extension uses a different and incompatible API. This creates a need for a database class library to hide the differences between the different databases so we can easily switch databases. The advantage of using a database abstraction layer instead of calling the database’s native functions directly is code independence; see sections (3.4.2) and (3.4.3). The disadvantage of database abstraction layers is that you can lose access to features that are database-specific. The code might also be less efficient due to the overhead imposed by the abstraction layer.
Figure 3.4 Abstraction layer generation
A number of different database abstraction layers have been developed for PHP; the most significant ones are PEAR DB, and ADOdb. Both of them are open source, maintained and used by a large community of users.

3.6.2 PEAR DB

PEAR DB is one of the component tools of PEAR (PHP Extension and Application Repository), an object-oriented Database Abstraction Layer for PHP [56]. It is installed by default in most PHP distributions.

The PEAR DB module supports database access based on a two-level design:

- The top level provides an abstract interface that hides database specific details and is the same for all databases supported by PEAR DB.
- The lower level consists of individual drivers, where each driver supports a special database server.

The PEAR DB architecture implements database support through two files that are used for all database systems, and a third file that is used according to the specific engine:

- The primary file is DB.php, this implements DB class which creates a database connection object.
- DB/common.php implements the DB_common class which forms the basis for database access.
Database Development

- DB/driver.php implements the DB_driver class which is selected according to the database to be used, for example DB/pgsql.php for the PostgreSQL engine.

This architecture implementation is structurally similar to the one used in Perl DBI.

3.6.3 ADOdb

Active Data Object database (ADOdb) is one of the most full-featured and efficient PHP abstraction libraries available today [57]. It is easy to learn, especially if you have Window's programming experience, as it uses many ADO conventions. It supports a wide variety of database systems, including MySQL, PostgreSQL, Oracle, Interbase, Access ODBC and others.

Many popular web applications such as phpLens, PostNuke and many others are using ADOdb as their database abstraction layer [58]. ADOdb has unique features such as:

- Provides code to handle inserts and updates that can be adapted to multiple databases quickly.
- Support for PHP5 iterators and exceptions included.
- Methods are provided for date handling, string concatenation and string quoting characters for differing databases.
- A metatype system is built in so that you can equate types such as CHAR, TEXT and STRING across different databases.
- Supports the creation of databases, tables, indexes portably using an XML.
- SQL monitoring and schema creation using XML.
- Provides extensive portability support such as date and type-handling and portable schema creation.
- Support many enterprise features such as database backed sessions (with session expiry notification), SQL code generation, pivot tables, SELECT LIMIT emulation for all databases, performance monitoring.
- Provides a class named Adodb_Active_Record which exposes a separate object for every table in the database by which you can perform different operations.

When using ADOdb, at least two files are loaded. First is adodb/adodb.inc.php, which contains all functions used by all database classes [59]. The code specific to a particular database is in the adodb/driver/adodb-database server.inc.php file and this implementation is similar to Java JDBC and Perl DBI:DBD.

The ADOdb flow process is shown in Figure 3.5:

1. The first step is to include the abstraction layer in the script.

   ```php
   include ('adodb.inc.php');
   ```

   Note that the ADOdb library doesn't consist of just this file - in fact, there are over thirty different files included with the library, many of them drivers for different databases.
By including the main class file, as above, then it will invoke the appropriate drivers or additional classes as required.

2. Next, create a Connection object using NewADOConnection($driver) function.

\[
$\text{db} = \text{NewADOConnection ("pgsql")};
\]

The parameter passed to the object constructor tells ADOdb which type of database you're trying to connect to. In this case, we will use the argument "pgsql" to connect to a PostgreSQL database server.
3. Next open up a connection to the database. This is accomplished via the Connect() method, which must be passed a set of connection parameters.

   $db->Connect("localhost", "username", "password", "databasename");
The line of code above attempts to open up a connection to the PostgreSQL using parameters the host named "localhost", the username, password and database name. It is possible to use a Data Source Name (DSN) that contains all of the information necessary to connect to a database. The DSN format is:

$driver://$username:$password@hostname/$database

4. Next the object's Execute() method can be used to execute SQL queries on that database.

$query = "SELECT * FROM tablename";
$result = $db->Execute($query) or die("Error in query: $query. " . $db->ErrorMsg());

Successful query execution returns a new object containing the results of the query. The special ErrorMsg() method, which can be used to obtain the last error message generated by the system.

5. The result object returned in the previous step exposes methods and properties that can be used to extract specific fields or elements from the returned result set.

while (!$result->EOF)

In this case, the object's MoveNext() method is used, in combination with a "while" loop, to iterate through the returned result set and display individual fields (these individual fields are accessed as array elements of the object's "fields" property). This data is then printed to the output device. ADOdb also offers a number of alternative methods to process a result set.
For example, you can retrieve the result set as a string-indexed associative array, where the keys are field names and the values are the corresponding field values. You can also fetch each row as an object, whose properties correspond to the field names, via ADOdb's FetchNextObject() method.

6. Finally the Close() method is used to close the database connection.

   $db->Close();

ADOdb comes with a number of utility functions that provide a useful information on the executed query. The most useful of these are the RecordCount() and FieldCount() methods, which return the number of rows and columns in the result set respectively. Also it is possible to obtain further information on each field with the FetchField() method, which returns an object containing detailed information on the field properties, including its name and type.

Finally, ADOdb has a method to obtain a list of databases on the server called the MetaDatabases() method, and a method to get a list of tables within the current database named the MetaTables() method.
Chapter 4

Related Work

4.0 Introduction

Web-based applications are programs that run on Web servers and use Web pages as the user interface. For the average user this new kind of software will be easier, cheaper, more mobile, more reliable, and often more powerful than non Web-based applications. With Web-based software, most users will not have to think about anything except the applications they use. Recently, there has been a strong trend towards providing web-based interfaces to databases. There are a number of reasons supporting this movement, including:

- Web browsers are common interfaces for browsing data
- Web-based applications can easily be added into an existing web site
- Web (HTML) interfaces are easily created and modified

Many attempts have been presented in several academic papers and articles to address these Web-based software concerns.

In this chapter we review different approaches to using databases on the Web, in section 4.1 we discuss the Application centred approach, section 4.2 examines the External schema approach, the DBMS Application development tools are discussed in section 4.3, the automated development of object oriented user interfaces is discussed in section 4.4, the database centred approach is considered in section 4.5, and in section 4.6 we talk about the development of PHP. Our focus will be on automated interfaces to databases using metadata and study of PHP.
4.1 Application Centered Approach

4.1.1 Introduction

In the early days of web applications, all the code was heaped together into a single document, which ran when you accessed it from a browser. However, as online applications got longer and more complex, it became a good idea to split out the presentation code from the rest of the code to make maintenance and debugging easier. That led to a whole new breed of applications. But even that architecture in turn has been superseded by the addition of the model for data handling.

We note the rise in recent years of a number of Web frameworks perhaps most significantly Ruby on Rails [11] and then for PHP a number of competing frameworks that are essentially clones of this approach such as CakePHP [60], Akelos [61] and PHP on Trax [62] applying the use of the MVC design pattern and RoRs use of “Convention over Configuration”. Frameworks are a set of objects, functions and elements which provide a certain functionality to help in the development of an application. Frameworks can be of different shapes and sizes. There can be micro frameworks which only control the most basic application's architecture. On the other hand frameworks like Ruby on Rails control almost every aspect of development. Every application has its own needs and some original functionalities are required. Developers need a certain amount of freedom, so frameworks shouldn't be too restrictive and should allow certain configuration.
The Rails and Struts [63] [64] style is followed in many PHP frameworks [65] for example see the extended list of projects doing a similar job in Table 4.1.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Modeled after</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phrame</td>
<td>Java Struts</td>
<td><a href="http://www.phpwact.org/php/mvc_frameworks/phrame">http://www.phpwact.org/php/mvc_frameworks/phrame</a></td>
</tr>
<tr>
<td>Akelos PHP Framework</td>
<td>Ruby on Rails</td>
<td><a href="http://www.akelos.org/">http://www.akelos.org/</a></td>
</tr>
<tr>
<td>Biscuit</td>
<td>Ruby on Rails</td>
<td><a href="http://ripcord.co.nz/biscuit/">http://ripcord.co.nz/biscuit/</a></td>
</tr>
<tr>
<td>Cake</td>
<td>Ruby on Rails</td>
<td><a href="http://cakephp.org/">http://cakephp.org/</a></td>
</tr>
<tr>
<td>CodeIgniter</td>
<td>RoR</td>
<td><a href="http://codeigniter.com/">http://codeigniter.com/</a></td>
</tr>
<tr>
<td>Fusebox</td>
<td>ColdFusion</td>
<td><a href="http://www.fusebox.org/">http://www.fusebox.org/</a></td>
</tr>
<tr>
<td>Kolibri</td>
<td>Apache Struts</td>
<td><a href="https://launchpad.net/kolibri">https://launchpad.net/kolibri</a></td>
</tr>
<tr>
<td>PHP on Trax</td>
<td>Ruby on Rails</td>
<td><a href="http://www.phpontrax.com/">http://www.phpontrax.com/</a></td>
</tr>
<tr>
<td>PhpMVC</td>
<td>Java Struts</td>
<td><a href="http://www.phpmvc.net/">http://www.phpmvc.net/</a></td>
</tr>
<tr>
<td>Sapphire / SilverStripe</td>
<td>RoR</td>
<td><a href="http://silverstripe.com/">http://silverstripe.com/</a></td>
</tr>
<tr>
<td>Studs</td>
<td>Java Struts</td>
<td><a href="http://www.mojavelinux.com/projects/studs/">http://www.mojavelinux.com/projects/studs/</a></td>
</tr>
<tr>
<td>struts4php</td>
<td>Java Struts</td>
<td><a href="http://www.struts4php.org/">http://www.struts4php.org/</a></td>
</tr>
<tr>
<td>symfony</td>
<td>Ruby on Rails</td>
<td><a href="http://www.symfony-project.org/">http://www.symfony-project.org/</a></td>
</tr>
<tr>
<td>PHP Work</td>
<td>ASP.NET-like</td>
<td><a href="http://phpwork.org/">http://phpwork.org/</a></td>
</tr>
</tbody>
</table>

Table 4.1 Rails and Struts like implementation in PHP

4.1.2 **Ruby on Rails**

Rails is a Web application framework that uses Ruby as its programming language for developing web applications with a database component using the Model-View-Controller pattern[5]. Rails supports many web servers such as Apache HTTP server. Rails is configured with the MySQL database by default.
Rails also support other databases such as PostgreSQL, SQL Server, IBM’s DB2 UDB, and Oracle. Rails is supported on most operating systems.

Overview of Rails

Rails is a web application and persistence framework to develop database based web applications according to the Model-View-Controller pattern[66] see Figure 4.1. Views are the user interfaces of a web application. A view is rendered using Ruby embedded HTML (RHTML), and Ruby-generated XML (RXML). The controller sets instance variables required by a view and renders a view. A view contains links to methods (actions) defined in the controller with which controller actions are invoked. In Rails, models are typically based on the Active Record design pattern[29], which provides an object relational mapping (ORM) between business objects and a database. In an Active Record pattern a database table is represented by a class, and an object instance represents a row in the database table. The database table columns are represented by the attributes of the class, and the class provides accessors for each column in the database table. The controller is a class that extends the Application Controller class and consists of actions (methods). A controller integrates the model with the view using public methods (actions). The model provides the data, the controller provides business logic to process the data, and the view presents the data. A request is initiated from a view template in a browser. The web server forwards the request to a dispatcher. The dispatcher loads the controller. The controller provides the business logic and interacts with the Active Record persistence layer to return a response to the view template, which gets displayed in the browser.
Rails is built on two solid principles: convention over configuration, and Do not Repeat Yourself (DRY) [68].

**Convention over configuration**

In Rails, there is a standard set of conventions that must be followed. For example Rails expects that all database field names are lower cased, database table names are pluralized and every database table has primary key named id. Rails uses a standard naming convention for the relationship between classes and database tables (classes are
Related Work

singular, and tables are plural). So if we need to model cars in a rails application, we will name our class car and our database table cars.

Do not Repeat Yourself (DRY)

Another important Rails principle is to avoid duplicating information. A traditional program contains code describing database tables. The code tells the rest of the program about the structure of the tables. Only after this descriptive code is in place can the rest of the program read data from the database.

4.1.3 CakePHP

CakePHP [69] is a Web application framework for producing web applications. It is written in PHP, modelled after the concepts of Ruby on Rails. It enforces the Model-View-Controller (MVC) structure for Web applications. Basically, Cake separates typical operations into specific areas: models for all database interaction, views for all output and displays, and controllers for all commands/scripts for input and program flow see Figure 4.2. CakePHP accomplishes this goal by providing the resources to make MVC possible and also by using a consistent method for where to store operations in the application.

By naming the files a certain way allows CakePHP to piece together the various resources without using any code specifications.
1. The client sends a page request to the application, either by typing a URL or by clicking a link of some kind. By convention, a typical URL is usually structured like this: http://{Domain}.com/{Controller}/{Action}/{Parameter 1, etc.}

2. The dispatcher script parses the URL structure and determines which controller to execute. It also passes along any actions and parameters to the controller.

3. The function in the controller may need to handle more data than just the parameters forwarded by the dispatcher. It will send database requests to the model script.

4. The model script determines how to interact with the database using the requests submitted by the controller. It may run queries with the database and do all sorts of handy data-sorting instructions.

5. Once the model has pulled any data from or sent data to the database, it returns its output to the controller.
6. The controller processes the data and outputs to the view file.

7. The view adds any design or display data to the controller output and sends its output to the client’s browser.

CakePHP has built in CRUD (Create, Read, Update, and Delete) functions. Instead of writing each CRUD operation by hand, it has prebuilt classes that do it automatically. CakePHP includes the Bake script; a command-line tool that generates editable CRUD code based on the database schema and customized parameters.

4.1.4 Limitations of Application Centred Approach

The developed prototype system contrasted to the frameworks style with an example over the use of database keys. In frameworks following the Ruby on Rails approach primary keys must be identified by the use of a column whose name ends _id. This seems to be an overly restrictive approach since the fact that a column is a primary key can be obtained from the database metadata regardless of its name. The metadata approach allows the use of existing, or more natural, naming of database columns such as when an external real world entity such as a national insurance number provides a primary key value. This also makes working with compound primary keys which are likely to be found in databases of any reasonable complexity straightforward. Similar issues apply over foreign keys, where this research proposes obtaining the information from metadata rather than relying on naming conventions.

Frameworks similarly have problems dealing with legacy or pre existing databases.
4.2 External schema approach

There are many tools and utilities that can increase Web development speed, reduce debugging and testing time, and improve quality of the output. Fraternali [71] in his paper investigates the situation of Web development tools in research and commercial fields. The disadvantage of these tools is that they cannot deal with legacy and existing databases, and can not reflect any changes in database schema. In our approach we use database metadata rather than relying on an external model of the database to provide information about the database in contrast to these tools. The author has grouped existing tools into six categories, which exhibit the same features. The facilities these tools provide are different and have been put in order of increasing support to the structured development of Web applications. These are as follows:

1. Visual Editors and Site Managers: these contain productivity tools that developed directly from the WYSIWYG HTML editor, which established many concepts such as presentation styles and top-down site design. However, these editors do not provide support to the development of extensive database driven Web applications.

2. Hypermedia Authoring Tools: they share the same focus on authoring as visual HTML editors, but have different origin, offline hypermedia applications. Some of the known products of this group are: Aimtech Iconauthoer, and Allen Communication Quest, Macromedia’s Directore and Authoware, and many others.
They have limited support to structured development of large applications.

3. Web-DBPL (Database Programming Language): integrators address the integration of the Web technology and databases at the language level. They require a substantial development effort. Examples of HTML extensions include Cold Fusion’s Web database Construction Kit. Microsoft Active Server Pages (ASP) and other fourth generation languages.

4. Web Form Editors, Report Writers, and Database Publishing Wizards acquire a database centric approach to Web database integration, by focusing on the migration of client/server, form based applications. These tools aim at augmenting the implementer’s productivity in form editing, report writing, and event based programming; they offer a higher level of support with respect to Web-DBPL integrators, but still concentrate on the implementation phase. Some of the reviewed products are Visual Basic 6.0 and Microsoft Access97/2000, Oracle Developer 2000, and Microsoft’s Visual InterDev.

5. Multiparadigm Tools consists of a number of tools whose common feature is the integration of different development approaches and technologies from the previous four tools. The most typical configuration is in which visual HTML editing and site administration are extended with external components, which provide database connectivity, or with full fledged database publication wizards. Examples of such tools are FrontPage98, Elemental’s Drumbeat and NetObject’s Fusion (version 3.0).
6. Model Driven Web Generators provides a complete coverage of all the development activities, from analysis to the implementation stage of the pages. This category comprises a few commercial tools which demonstrate different conceptual models and code generation, some of these tools are Hyperwave Server 4.0, and Oracle’s Web Development Suit. Figure 4.3 summarizes the features of the above mentioned categories of Web development tools.

<table>
<thead>
<tr>
<th>Lifecycle coverage</th>
<th>Visual Editors</th>
<th>Hypermedia Tools</th>
<th>Web-DBPL Integrators</th>
<th>Form Editors</th>
<th>Multiparadigm Tools</th>
<th>Model-driven Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement, build, hierarchical design, link maintenance</td>
<td>Implement, design (authoring in-the-large)</td>
<td>Implement, maintenance (debugging)</td>
<td>Implement, hierarchical site design, link maintenance, debugging</td>
<td>Generation of HTML, database connection</td>
<td>Conceptual design, implementation, maintenance, reverse eng.</td>
<td>Generation of design schemas, navigation commands, interfaces</td>
</tr>
<tr>
<td>Automation</td>
<td>Generation of HTML, Java</td>
<td>Database connection, query shipment, result formatting</td>
<td>Generation of HTML, Java</td>
<td>Page, link, presentation style, form, table</td>
<td>Entity, relationship, module, table, column, collection, link</td>
<td>Modules, preferences, collections, links</td>
</tr>
<tr>
<td>Abstractions</td>
<td>Page, link, presentation style</td>
<td>Authoring metaphors</td>
<td>Form, report, client-side and server-side control</td>
<td>Page, link, presentation style, form, table</td>
<td>Entity, relationship, module, table, column, collection, link</td>
<td>Modules, preferences, collections, links</td>
</tr>
<tr>
<td>Reuse</td>
<td>Components, presentation styles</td>
<td>Libraries, components, styles</td>
<td>Page templates, DBPL units</td>
<td>Client-side and server-side components</td>
<td>Components, presentation styles, templates, 3-tiers, dynamic, Good graphic control and coherence (manual)</td>
<td>3-tiers, dynamic, predefined interfaces, low graphic control</td>
</tr>
<tr>
<td>Default architecture</td>
<td>2-tiers, static</td>
<td>2-tiers, static</td>
<td>3-tiers, dynamic</td>
<td>3-tiers, dynamic</td>
<td>3-tiers, dynamic, predefined interfaces, low graphic control</td>
<td></td>
</tr>
<tr>
<td>Support to usability</td>
<td>Good graphic control and navigation (manual)</td>
<td>Very good graphic control and navigation, synchronization control (manual)</td>
<td>Dynamic Interface through triggers</td>
<td>Dynamic Interfaces</td>
<td>Dynamic Interfaces, predefined interfaces, low graphic control</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 4.3 Summary of the Different Categories of Web Development Tools adopted from [61]*
4.3 DBMS Application Development Tools

There is a history of commercial DBMSs including application development tools as part of their overall software package. Examples of this go back as far as the 1980s where Ingres had a system covered under “The Ingres Application Development System” by C.J Date in his book on Ingres [72]. In this he covers the “data dictionary” the repository of metadata which as he notes “consists of some 20 or so system tables [72]p.103. This information is then used in areas such as the development of default forms where “the declared widths and names are taken from the definition of the specified table in the dictionary...[72]p.260. Further examples of the Ingres form of this type of tool can be found in Malamud [73]. This type of tool, often referred to as a 4GL was intended to allow rapid development of applications typically with terminal interfaces to the DBMS. In the next section we provide some detail of the Oracle tools in this area noting that as the environment has changed a wider range of these tools now exists some with the possibility of browser based interfaces.

4.3.1 Oracle Forms, Application Developer and Fusion

A stated aim of Oracle Forms is to create applications “from database definitions with minimal coding effort”[74]. Oracle Forms started as an interactive development environment (answering questions and the tool create a runnable application). Later it did develop to run in server-side environment in character mode and block mode terminals. Now it is used in other platforms, such as Windows, to function in a client-server environment. Later versions were ported to Java where it runs in a J2EE
container and can integrate with Java and web services. The previous versions of Oracle Forms prior to version 9i could run in client/server mode, but the newer versions can only run on the Web. Oracle Forms has an object navigator, property sheet and code editor that uses PL/SQL. The precise mechanism of using metadata is not made clear but they state that it uses a “data block wizard to automatically link your application to tables in your database” [74]. Oracle Forms access the Oracle database and generates a screen that displays the data. The source form (*.fmb) is compiled into an executable (*.fmx) that is interpreted by the forms runtime module. The form is used to view and edit data in applications.

Many Graphics User Interface elements, such as buttons, menus and scrollbars can be placed on the form. Oracle Forms consist of triggers that occur at certain steps in the processing of data, such as keyboard strokes and mouse movements. Different triggers are called after and during each step. Altering the default behaviour of the Oracle Forms is done by modifying the contents of these triggers.

More recently the range of these tools has expanded, a brief survey of the range and some information on the underlying philosophy is given in Ronald [75]. All the tools make some use of metadata but as a paper on moving from Forms to Application Express notes [76] the extent and style seems to vary with “Application Express renders applications using metadata stored in an Oracle database.” contrasting this with “Oracle Forms renders applications using metadata stored in an .fmx file.” as noted above. Ronald notes that the migration from one tool to another is not simple or automatic.
This approach by all these DBMS application tools is in principle is similar to our approach in using metadata. However as we have seen there is no general approach but rather the development of a tool that is tightly coupled to a specific DBMS (Oracle for example) and often targeted to a specific platform or style of application development, Ronald[75] questions whether the tools should be “just delivering an HTML version of the Forms UI” and notes that there may be deeper issues about the correct architecture required in different areas of applications. Additionally since these tools are tied to a specific DBMS they do not need to pay attention to issues of DB architecture (see section 3.5) but we presume follow the old Ingres model of using the (DBMS specific) system catalogs rather than a generic approach.

4.4 Automated development of Object Oriented user interfaces

4.4.1 Automatic generation of user interfaces from data structure specifications and object-oriented models

The authors in [77] present two approaches for automatically generating user interfaces from data structure information. The first uses information from object oriented mathematical models, together with a set of predefined elementary types and manually supplied layout and grouping information. In the second approach they have designed a tool called PDGen (Persistence and Display Generator) that automatically creates a graphical user interface and persistence routines from declarations of data structures used in applications. Structuring and grouping information is automatically extracted from the inheritance and part of relations in the object oriented model and transferred to PDGen which creates the user interface.
4.4.2 Considering User Interfaces to REA Systems

The approach suggested in Engleson et. al. has been adopted elsewhere as by Jesperson[78] where again class definitions are used to derive user interfaces but in this case from specifications rather than programming code. In particular here the focus is on inheritance as a feature of the object-oriented context but there are similarities in their approach of tying a particular class to a corresponding interface element to our approach of associating a type to a web form element.

4.4.3 The Development of OCPL, Object Conceptual Prototyping Language

James and Shipley in this paper [79] outlined the development of the object knowledge representation language OCPL (Object Conceptual Prototyping Language). They use class definitions metadata to generate the user interfaces (e.g. insert button, delete button, blank buildsheet etc.). Their approach represents a development from the two previous papers in that they use OCPL which is a prototyping or object-knowledge representation language rather than a programming language. The richer constraint model that comes with the use of a knowledge base approach means that OCPL has some greater similarities to our approach in areas in terms of deriving interface behaviour from more than just structure as in the cases on Engelson and Jespersen. OCPL can express rules for an application which can be compared to the rules we derive from the database metadata.
In general this approach is not applied to databases and focuses on object oriented software; it also makes use of data structure information in source code or formal specification as the source of information to build the interface which is in contrast to the database metadata approach where the metadata which is available at runtime is the source of information. In this respect the approach although not database focused is close to the database approaches that use external models to drive interface development, with source code or specification acting as the external model.

4.5 Database Centred Approach

4.5.1 Related Work to User Interface to Databases

The evolution of database technology both in hardware and software capability has provided users with significant memory and computational resources. Unfortunately, such resources exceed the training and experience possessed by prevailing users. As a consequence, database system designers have attempted to develop user interfaces which reduce the relative level of skill required to select information from relational databases. However, as the popularity of relational database use expands, and the level of training for a routine user decreases, the user friendly character of the human interface becomes a more important and significant contributor to the overall value of the database.

Thus there exists a need for an interface between a human user and relational database having an architecture which is both object-oriented and intelligent at a high level of
user interaction. The following section introduces a brief outline of some academic papers related to a Web user interface to databases.

4.5.1.1 Automatic generation of graphical user interfaces for interactive database applications

The author in [80] describes an approach aimed to novice developers with little design experience and implementation of graphical user interface (GIU), and with only a basic understanding of the conceptual database schema.

The system consists of an automatic screen layout generator and a visual language in which end user requirements are described through direct manipulation of an Entity-Relationship diagram of the database. The visual language provides a series of operators that enable the transformation of the schema diagram into another type of graph, termed application specification diagram which defines the entities and relationships of interest.

The interpreter translates these diagrams into textual descriptions of the application that are used as input to the screen layout generator. The author argues that the main difference between the use of the E-R diagram and that of the data modelling approach and query languages was in the way the diagrams were interpreted. The tool introduced in his approach transforms graphical representation into statements that describe the application components to the automatic screen layout generation facility. Whereas data modelling tools and query languages convert them into textual data definition and data manipulation language commends respectively.
Related Work

This approach is based on external (to the database) representation of the database, the Entity-Relationship diagram, which is in contrast to our approach which will be based on integrating the database itself.

4.5.1.2 Accessing Relational Databases from the World Wide Web

In [81] the authors proposed a mechanism to access database using SQL and HTML sections linked together via cross language variable substitution. This variable substitution mechanism is used in the design and implementation of a system called DB2 WWW Connection which enables construction of database Web applications, see Figure 4.4. The HTML section contains the format of the input and output reports, and the SQL section contains the database commands. The created HTML forms and the SQL commands are stored in macros at the Web server. These macros then get processed by the DB2 WWW Connection engine. The end user of this system only sees the requested forms and the results output. The disadvantage of this approach is that the information the users receive and the queries they make are tied, because the forms and reports are build in advance. This approach is not generic and it only works with DB2 database.
4.5.1.3 Automatically Generating World Wide Web Interfaces to Relational Databases

The authors in [82] argue that reformatting data held in databases into HTML pages appropriate for WWW requires tremendous effort in creating and maintaining these pages. They proposed an automating method of accessing relational databases from the WWW. They argue that their approach is capable of automatically generating a WWW interface to a database using metadata from the catalogue. The interface allows direct querying and browsing of the database through dynamic hyper text links which are derived from the referential integrity constraints defined in the metadata.

The proposed prototype according to [82] is a DBMS specific, it does not use a generic access method to the database; in addition to that the types of the metadata is limited.
On the contrary, our approach is more generic since the metadata tables are automatically generated from any given database using the ADOdb metadata functions. So the need for manual update of metadata tables or the web interface code will be eliminated.

4.5.2 Related work to User interface to metadata

Metadata is simply defined as "data about data." It is the background information which describes the content, quality, condition, and other appropriate characteristics of the data. Metadata can be organized into several levels ranging from a simple listing of basic information about available data to detailed documentation about an individual data set. Metadata insures that potential data users can make an informed decision about whether data are appropriate for the intended use.

4.5.2.1 Metadata tables to enable dynamic data modelling and web interface design: the SEER example

The authors in [3] proposed a model to develop a web based database interface whose structure and content is generated throughout interaction with a metadata table. They developed and implemented a metadata table that would represent a broad array of database models Figure 4.5. The metadata table contains table names, field names, field data type and the link between the tables see Table 4.2. The metadata table stores beside structural information, the descriptive information about the fields themselves. The web interface would display the descriptive information not the cryptic field name.
However, the described approach is error-prone since the metadata is built by hand, whereas, we suggest using database metadata that can be retrieved dynamically on the fly using PHP and abstraction layer ADOdb. Therefore, we conclude that the described approach is considered as using an external representation rather than directly using the database metadata.

Figure 4.5 A subset of the SEER data model captured from[3]
Related Work

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Field Name</th>
<th>Field Data Type</th>
<th>Linked Table</th>
<th>Linked Table Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Table</td>
<td>Unique ID</td>
<td>Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Central Table</td>
<td>SEER Registry</td>
<td>Foreign</td>
<td>SEER Registry</td>
<td>Register Code</td>
</tr>
<tr>
<td>Central Table</td>
<td>Birth Year</td>
<td>Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Central Table</td>
<td>Cancer Type</td>
<td>Foreign</td>
<td>Cancer Type</td>
<td>Cancer Code</td>
</tr>
<tr>
<td>Central Table</td>
<td>ICD-O Code</td>
<td>Foreign</td>
<td>ICD-O Code</td>
<td>ICD-O Code</td>
</tr>
<tr>
<td>Central Table</td>
<td>Site Code</td>
<td>Foreign</td>
<td>Site Code</td>
<td>Site Code</td>
</tr>
</tbody>
</table>

Continues for each field in the Central Table Table

| Registry Table | Registry Code | Number         |                    |
| Registry Table | Registry Name | String         |                    |
| Registry Table | Gender Code   | Number         |                    |
| Registry Table | Gender Name   | String         |                    |
| Registry Table | Cancer Code   | Number         |                    |
| Registry Table | Cancer Name   | String         |                    |

Continues for each field in each lookup table

Table 4.2 The metadata table that represents the SEER data model adopted from [3]

4.5.2.2 Developing Web Entry Forms Based on Metadata

The authors in [83] suggested an approach that can be used to develop and maintain HTML forms based on metadata extracted from system tables. They have used Java DataBase Connectivity (JDBC) [84], which has a general method for accessing different databases and also has metadata features. Dynamic HTML forms were generated and validated automatically. Since HTML is generated on the fly, changes made to the database are reflected the next time the data is accessed. They used Java and metadata to display help messages to the user and validate the input data.
In comparison with our approach, we are targeting the same goal and use the same information resource that is database metadata, obvious dissimilarities between them have been cited. We will be using the PHP language and the ADOdb library functions to get the information needed from the metadata to develop a general interface to a database.

4.5.2.3 Developing an Abstract Representation for User Interface Elements Based on Database Metadata

The proposed approach in this paper [85] aims to generating a Web user interface for Web applications can be achieved using metadata that is stored in system catalogue tables and integrating it with XML technology. The authors used Java Servlets for parsing the user entered data and passing it to database server as SQL statement using JDBC. Based on the extracted data from database, the Java servlet build up an XML document containing the necessary information to generate an XHTML Web form. A set of rules are applied to XML to generate dynamic XHTML form by calling XSLT style sheet. Small alterations to XSLT style sheets results in a new look Web user interface without rewriting or recompiling the source code. This approach is similar to ours in many ways in which it makes use of the extracted information from database metadata to build up dynamic Web user interface. In addition, the same concept of mapping every single attribute in database’s tables to a specific user interface control has been used. Our approach, in contrast, makes use of the free and open source technologies such as PHP and the abstraction library ADOdb.
Related Work

4.6 PHP

PHP like other open source software projects has been and continues to be developed by a wide community of users. Like other open source products it is also often seen as having a central originator or leader, in the case of Linux - Linus Torvalds, for Perl - Larry Wall and for PHP - Rasmus Lerdorf, the extent to which the process is broader and more democratic clearly varies from project to project. Many open source projects are effectively single person operations, over time some grow and involve larger numbers, some are independent and some exist as self contained add-ons to existing projects, this is particularly the case for specialized libraries.

Although this process has many benefits there also seem to be some drawbacks particularly in the area of a clear roadmap for the development process. Where there is evidence of a plan or specification it is often not in the form of a formal document or an academic paper but in a conference presentation or less formal document available on the internet e.g. in a mailing list archive. This process can be seen in the development of PHP, which has been rapid due to its wide adoption by users. Recent versions of PHP have ever increasing functionality due to an increasingly large number of libraries providing APIs for specific functions, our particular area of interest being database access.

However there seems to have been rather confused consideration of the structure and content of the libraries which in fact provide so much of PHP's functionality. There has been work on PEAR and PDO [56] [86] but even among developers there still remains a
feeling that it is rather ad-hoc and has been noted that "PHP is basically a collection of extensions which are all put together to form what we have now" [87] in relation to PHP 6 in the context of "most hosts are still in the PHP 4 era" the author also noted a developers meeting with Rasmus Lerdorf concluding [88] "PHP 5.1 introduces PDO, an extension that unifies Database APIs. With this we do not "need" older extensions to access databases anymore." but noting that "We cannot remove the "old" extensions, as at least OCI8 and MySQLi provide a very rich set of features, which are not all supported by PDO. Some ‘old’ extensions can probably be moved to PECL as they are either unmaintained, or superseded by PDO." and concluding that "We decide on moving DB extensions out of the core later".

This is one example of the confusion that exists around the Database libraries (and may be repeated in other areas). It is interesting to contrast the situation with that of both Perl and Java. Perl, also as an open source project with a large community of developers as well as users, had a history that was similar to that of PHP with a number of independent database libraries such as oraperl [89] which were then unified and superseded by the development of DBI and the set of DBD modules [53] [54]. In Perl's case that process seems to have been much smoother and the legacy approaches dried up quickly, despite following Perl in time and style PHP does not seem to have made the same transition and we are currently in a position where there are a number of competing and overlapping complimentary approaches. Even in the area of documentation and naming it is difficult to compare approaches when different libraries are described as "unifying database" [90] “query abstraction layer” [59] or "database
abstraction layer" [91] by different authors and some libraries are seen as working at different levels. Java, as a non open source project, although with a community process of sorts has a much more clearly defined Database API in JDBC [49] which is well and formally documented. Although this is not supported by documentation to motivate it [92] and the issues of compliance to the standard can be problematic [93].

JDBC's underlying structure can be viewed as architecturally similar to Perl's with a common interface that is specific DBMS independent as in DBI with DBMS specific operations contained with specific driver code as in DBD modules. This module, although not formally specified, provides us with a basis for describing the expected functionality of a database abstraction layer for a programming language and hence the level of abstraction we would like to implement in PHP.

As a consequence of this approach parallel and competing developments occur, this can be seen with frameworks and database access libraries.

4.6.1 Books and papers on PHP

In background research for this project a large number of books on PHP were consulted, in particular the range of books on PHP/DB which tended to be focused on a single DBMS, often MySQL. These books reflected what was also found in Web searches that while a lot of development is done in PHP there seems to be little research. The books found did not use any advanced techniques and tended to follow developments in other fields as was discussed above in relation to Rails style development.
Related Work

Similarly searches of ACM and IEEE digital libraries find a large number of hits for articles but these are predominantly using PHP and do not represent any study of PHP itself except for the few examples below.

1. On some drawbacks of the PHP platform

The author in [94] discusses some of PHP’s drawbacks. There are over 4000 functions in PHP, the naming conventions of these functions are not always consistent. Some of these functions have a unique purpose, such as phpinfo(), while some have a functionality which is overlapping with one or more of the others see Table 4.3.

<table>
<thead>
<tr>
<th>Language</th>
<th>Escaping</th>
<th>Sorting</th>
<th>List walk</th>
<th>Splitting</th>
<th>Matching</th>
<th>Regular expressions</th>
<th>Substituting</th>
<th>SQL connection</th>
<th>Reading / receiving</th>
<th>Printing / writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>14</td>
<td>16</td>
<td>10</td>
<td>8</td>
<td>11</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Perl</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.3 PHP and Perl functions per intention captured from [84]

Additionally functions in PHP must have prefixes to denote their source, because PHP does not support namespaces.

Another drawback of PHP is that lower level database access (pg_*type functions) are not unified regardless of the database type.

Apart from [94] there seems to have been little research into PHP as a language apart from [95] [96] which focus on issues of efficiency and security in lower level
implementation of the language core not on libraries which Nikolaj Cholakov highlights as significant for the software developer.

4.7 Conclusion

This chapter has discussed research related to our proposed approach. Most of the systems that were considered targeted at creating application domain specific interfaces rather than generic interfaces. While others did consider the notion of developing generic interfaces but they come with several limitations that we tried to solve in this study.

Comparing all attempts discussed in this chapter to our approach, our approach is not limited by domains, because we are not developing or maintaining an external data model or representation. Additionally our approach is more flexible since it can be used with a wider range of databases, not only those following the convention technique. This means that it is possible to work more easily with legacy systems or systems where the database structure includes column names and types reflecting other business logic demands. Moreover, combining database metadata with PHP allows our approach from generating Web entry forms without the need to recompile the source code or making any changes to access DBMS.
Chapter 5

Framework Design and Architecture

5.0 Introduction

This chapter presents our suggested approach, design and architecture. We aim to
design a prototype using metadata stored in system tables in databases to generate
automatic and dynamic Web entry forms. The aim is to enable a high level of software
reuse and produce brief code at a high level using trusted libraries which can allow
changes or upgrade of RDBMS, changes to database schema and database data without
recoding effort. The prototype is not specific to any DBMSs or platform, and it is
flexible and reusable with minimal effort.

It begins with an overview of the prototype features. The structure of the framework is
demonstrated and a system design diagram is illustrated.

5.1 Framework Overview

The main aim of our approach and its implementation was to study and investigate
to what extent we can use the potential information held in database metadata to
develop an abstract representation that can be used for producing Web entry forms that
is not bound to a specific platform, but not building a comprehensive Web application
targeting a particular domain.

Contrasted with the common approaches that we considered in chapter 4, our approach
is not bound to a specific domain. The internal information already present in relational
database metadata was considered and we avoided using external data models in this
Framework Design and Architecture

research. It is richer than the external data model. Our approach aims to get the most abstract, highest level, most reusable and general solution for extracting and using database metadata. Metadata can be retrieved from a logical model by several methods at certain levels. At the highest level, metadata can be extracted using ADOdb metadata functions at runtime. The second level of metadata resides in an information schema. At the bottom of the logical model, metadata can be accessed by querying system tables as shown Figure 5.1. Querying information schema and system catalogs could be restricted by the versions of DBMS. However, for reasons listed in section 3.5.3, ADOdb was chosen to implement our approach. The internal details of how ADOdb gets results at driver level by either query information schema or system catalogs; or how ADOdb encapsulates getting metadata is beyond the scope of this research. In our implemented framework, metadata will be retrieved using ADOdb metadata functions at runtime, and then sent back to a Web browser to transform it into a desired Web Entry Form based on applying a set of rules.
As far as the generation of Web Entry forms using a single prototype system is concerned, separation between the main components of a Web application (content, logic and presentation) is considered. Adopting CSS technology in our approach allows us to achieve this goal in a straightforward way.

5.2 Prototype Features

The prototype offers the following features:

- HTML Entry Forms are generated and validated automatically.
- Any changes made to the data or schema is reflected the next time the data is accessed.
- The code is non DBMS and non platform specific.
- Dynamic use of database metadata rather than the use of convention to derive information about the database.

### 5.3 Three-tier Connection

A Web application is commonly structured as a three-tiered application. In its most common form, a Web browser is the first tier. An engine using some dynamic Web content technology, we use PHP but an alternative such as ASP is equivalent, as the middle tier. A database is the third tier as shown in Figure 5.2. That it is three tier in general but with the particular technologies PHP, ADOdb and PostgreSQL that we use. The Web browser sends requests to the middle tier, which services them by making queries and updates against the database and generating a user interface.
5.3.1 **The Web Browser**

The first tier in the three-tier architecture model is usually a Web browser. A Web browser processes and displays HTML resources, issues HTTP requests for resources, and processes HTTP responses.

Other Web clients do exist such as Proxies, Search Engines etc, but they are beyond our concern in this thesis.

5.3.2 **The Web Server**

The second tier serves most of the roles that bring together the other tiers: it drives the structure and content of the data displayed to the user, and it processes input from the user as it is formed into queries on the database to read or write data. This tier integrates the Web with the database management system. The components of the second tier are a web server, a web scripting language, and the scripting language engine. A web server processes HTTP requests and formulates responses. In the case of web database applications, these requests are often for programs that interact with an underlying database management system. The web server we use in our prototype is *Apache HTTP server*. Apache is an open-source web server, fast and scalable. It can handle simultaneous requests from user agents and is designed to run under multitasking operating systems such as Unix and Windows.
We use the PHP scripting language and the abstraction library ADOdb for extracting the information from database metadata as our second-tier scripting language. PHP has a flexibility of embedding scripts in HTML pages which permits easy integration with the first tier.

5.3.3 **The Database Server**

The third-tier is the database tier which manages the data. The data management typically includes storage and retrieval of data, as well as managing updates, allowing simultaneous access by more than one second-tier process, providing security, ensuring the integrity of data, and providing support services such as data backup. In our approach, we use the PostgreSQL RDBMS to manage data as shown in the previous figure.

5.4 **Prototype Architecture**

The architecture of the proposed prototype consists of several processes as shown in Figure 5.3. These processes are listed as follow:

- A connection to a database management system is created using ADOdb. This database and its table’s metadata are extracted using ADOdb.
Figure 5.3 Architecture of the Framework
The retrieved information is transformed into an HTML document or into any desired format of Web entry forms. This process is performed in conjunction with a set of generic rules (see section 6.3). By using CSS this implemented approach makes a clear separation between content, logic and representation.

The generated Web form is returned back to the client for the user to fill in. The entered data will be validated against database metadata. This task can be performed on the client side, typically with JavaScript or on the server side via the functions of the scripting language, in our case PHP used as illustrated in Figure 5.4.
Figure 5.4 System components diagram
Chapter 6

Framework Implementation and Evaluation

6.0 Framework Mechanism

Based on the available Web technologies and the experimental work that we have done to investigate the capabilities of PHP and the abstraction library ADOdb for extracting the information from database metadata, the following section introduces an example of the implementation of this approach.

The concept is mapping each table’s columns to a specific Web entry form control based on a set of rules. The commonly used features of database metadata for this purpose are: the variety of data types, null or not null fields, primary/foreign keys.

6.1 Connecting to database and retrieving metadata

To illustrate our approach we create a database table which contains staff information as in Listing 6.1.
PHP and ADOdb is used to retrieve database metadata to get the information about each column in each database table such as column name, column type, column size...etc. A portion of the generic program code that is used to extract the database metadata is shown in listing 6.2. A sample of the extracted information when the code is run over the staff table is shown in Figure 6.1.

We then translate database native data type to a generic meta type as in Table 6.1.
Listing 6.2 Code snippet of ADOdb metadata methods

<?php
  include "adoDbConn.php"; // connect to the database server
  $dbname=$conn->MetadataNames(); // get database names
  $qry="select * from $table"; // get server information
  $recordSet = $conn->Execute($qry); // execute the query statement
  $host=$recordSet->connection->host;
  $汕k=$conn->MetaPrimaryKeys($stable); // get the primary key
  $汕k=$conn->MetaForeignKeys($stable); // get the foreign key

  function pkkey ($) {
    global $汕k;
    foreach ($汕k as $value) {
      return $value;
    }
  }

  function fkey ($) {
    global $汕k;
    foreach ($汕k as $value) {
      foreach ($value as $key) {
        $汕 = split (",", $key);
        return $汕[0];
      }
    }
  }

  if (!$recordSet)
    print $conn->ErrorMessage();
  else
    $汕m=$conn->MetaData($stable);
    $numcol=$recordSet->fieldCount();

    for ($汕=0; $汕<$numcol; $汕++) {
      // Looping through each column in the table
      $汕ld=$recordSet->FetchField($汕);
      $汕m=$汕ld->name;
      // get metadata information about column
      $汕name[] = $汕m[strtoupper($汕name)]->name;
      $汕type[] = $汕m[strtoupper($汕name)]->type;
      $汕size[] = $汕m[strttoupper($汕name)]->max_length;
      $汕null[] = $汕m[strttoupper($汕name)]->not_null;
      $汕default[] = $汕m[strttoupper($汕name)]->default_value;
      $汕max[] = $汕m[strttoupper($汕name)]->max;
      $汕number[] = $汕m[strttoupper($汕name)]->number;
  }
Figure 6.1 Part of the metadata retrieved from the database table in Listing 6.1.
<table>
<thead>
<tr>
<th>Defined Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Integer fields.</td>
</tr>
<tr>
<td>C</td>
<td>Character fields.</td>
</tr>
<tr>
<td>X</td>
<td>Large text fields.</td>
</tr>
<tr>
<td>N</td>
<td>Numeric fields.</td>
</tr>
<tr>
<td>R</td>
<td>Serial fields.</td>
</tr>
<tr>
<td>L</td>
<td>Logical field (Boolean or bit-field).</td>
</tr>
<tr>
<td>B</td>
<td>Binary Large Objects (Blobs).</td>
</tr>
<tr>
<td>D</td>
<td>Date fields.</td>
</tr>
<tr>
<td>T</td>
<td>Timestamp fields.</td>
</tr>
</tbody>
</table>

Table 6.1 Defined generic data types.

By using this information, it is possible to map every column in the database table to a specific Web Form control element (e.g. textbox, radio button, etc) as shown in Figure 6.7. This proposed approach is intended to be generic and abstract to the highest possible point. The framework produces a form which allows the user to enter data and that data will be stored in the database. In this approach the separation between content, logic and presentation is applied by putting the Web application's semantic content and structure in an HTML file, and the application’s visual layout is defined in an external style sheet file using CSS. As far as the generation of different types of Web Entry Forms using a single prototype system is concerned, separation between the main components of a Web application (content, logic and presentation) is considered.
Using CSS in our approach allows us to achieve this goal in a straightforward way. The difference is demonstrated in Figure 6.7 where all the components of the Web application are put in a single HTML file, but in Figure 6.8 we used an external style sheet file to contain the desired style to be presented to the user. The example demonstrates the use of (*) for the required fields and also the date of birth style is shown on the form as a guide to the user on how to fill in the form.

### 6.2 Generic metadata functionality

To illustrate the idea of generic metadata two tables with two columns have been created using PostgreSQL DBMS. The two columns in the first table 'first_name' and 'last_name' are of type character and size 30 as shown in Listing 6.3. This table’s metadata extracted using ADOdb methods is shown in Figure 6.2.

```sql
CREATE TABLE listname
{
  first_name varchar(30) NOT NULL,
  last_name varchar(30) NOT NULL
}
```

Listing 6.3 PostgreSQL table’s structure
Figure 6.2 Extracted metadata from table listname

The second table has two columns as the first table with the same names see Listing 6.5, the only difference is in the column type of the second column ‘last_name’, that is in this case is of type ‘lname’ created as a domain using PostgreSQL as in Listing 6.4.

```
CREATE DOMAIN lname
    AS varchar(30);
```

Listing 6.4 PostgreSQL domain structure
Listing 6.5 PostgreSQL table with domain type

The extracted columns metadata using ADOdb methods on the above table is shown in Figure 6.3.

<table>
<thead>
<tr>
<th>Column_Name</th>
<th>Column_Type</th>
<th>Column_Size</th>
<th>Nullable</th>
<th>Unique</th>
<th>Primary Key</th>
<th>Foreign Key</th>
<th>Default</th>
<th>Default_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar</td>
<td>80</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>first_name</td>
<td>varchar</td>
<td>30</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>last_name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.3 Extracted metadata from table namelist

By comparing the two extracted metadata from the two tables, the significant difference is in the column size for the column ‘last_name’, in the first table the column size is ‘30’ which is the real size of the column, but this is not the case in the second table where the column size is ‘-1’ and this does not reveal the true column size. In order to get the real column size in this situation, we would need to add DBMS specific code, calling system catalogs, to explain the domain definition.
6.3 Developing a set of heuristics

Any Web user interface is built from several user interface controls such as (text box, check box, drop down list, etc.), it will be useful to generate dynamically a user interface without the need to code the presentation of the user interface after any changes to each column data type in the database or to the database tables. By developing a set of rules, it is possible to automatically map each column in the database table to specific user interface control, and maintain database integrity.

Developing a set of rules is not done randomly. It is based on the information that exists in database metadata and data itself. It will be possible to automatically map each column in a database to a specific user interface control independently from how the browser will deliver the user interface controls.

The ADOdb library has a set of methods to get table column metadata, to use these methods a connection to a PostgreSQL database must be established first using the ADOdb connection function (NewADOConnection( )) as shown below:

```php
$dsn='pgsql://postgres:postdb@localhost/postgres';
$conn=&NewADOConnection($dsn);
```

Where in this case the user name is ‘postgres’, password is ‘postdb’, server name is ‘localhost’ and database name ‘postgres’.

Then using the connection handle $conn with these methods to get the required information.

```php
$conn->Meta.....
```
Examples of these metadata methods are described below:

**MetaColumns($table)**

This method returns an array of ADOFieldObjects, one field object for every column of $table. Each field object holds column information (i.e. name, type, size, nullable, unique etc.)

**MetaColumnNames($table)**

This method returns an array of column names for $table.

**MetaPrimaryKeys($table)**

This method returns an array containing column names that are the primary keys of $table.

**MetaForeignKeys($table)**

This method returns an associate array of foreign keys.

This range of metadata methods are used as shown below to implement a set of heuristics to map each column to a suitable user input.

**Examples of these rules:**

1. **Data Type Rules**

   The rules listed below are built based on the column’s data type:

   - Rule 1: if a column is a primary key or part of a primary key, then it should be mapped to an input box and its value must not be a null value.

     In order to get the column name(s) that make of the primary key, first the connection handle ($conn) is used to get the primary keys of the table using the ADOdb function (MetaPrimaryKeys(‘Name of table’)).
For example using the created PostgreSQL table ‘staff’ in Listing 6.1:

```php
$primarykey=$conn->MetaPrimaryKeys('staff);
```

Which returns $primarykey as an array containing the column name(s) that make
the primary key of the table ‘staff’, in this case it is the column ‘staff_no’. Then a
check is done to see whether a column is nullable or not by using the ADOdb
method MetaColumns($table) like this: $conn->MetaColumns('staff') which
contains the column metadata information. Some of this information is shown in
Figure 6.1.

Second in the PHP script we use this information to map this column to a user
input box like the first input box, and the (*) character is used to indicate that the
field is not null and the input box must not be left unfilled as shown in Figure
6.10.

The same procedure is done for all columns, primary key or not to check whether
null values are allowed.

JavaScript functions are invoked to validate the input data in a form before
sending off the content to a server. One of these functions, see example below,
checks if a required column has been left empty. If a required column is empty,
an alert box sends a message and the function returns false see Figure 6.13. If a
value is entered the function returns true.
Function field_required(field, alertmsg) {
    with (field) {

        if (value==null || value=="") {
            alert (alertmsg);

            return false;
        }
        else {
            return true;
        }
    }
}

- Rule 2: if a column is of type character/string and more than (for example) 30 characters, then it should be mapped to textarea control otherwise it might be mapped to input box.

By using the same connection handle ‘$conn’ and the ADOdb function ‘MetaColumns($table)’ to get the column’s metadata (e.g. name, size, type etc.). For example:

    $colinf = $conn->MetaColumns('staff');

This returns $colinf as an array of ADOFieldObject’s, one field object for every column of ‘staff’. A field object contains the data about the column.
In this rule the column’s type and size are used to map to appropriate user input currently set at 30 character i.e. input box. The limit could be varied so the user can choose the level of transition from input box to textarea according to the size of the column as shown in Figure 6.10 with both columns ‘first_name’ and ‘last_name’ are mapped to a user input box.

- Rule 3: if a column is a foreign key, then get the possible values from the referenced table and offer them as choices initiated as a pull down menu, if the number of values is under a specific limit. This rule shows the usage of database metadata to get the fact that it is a foreign key and makes use of the data itself for possible values.

```
$foreignkey=$conn->MetaForeignKeys('table');
```

This will return an associate array of foreign keys.

If table staff has a column called title that references a column named titles in a table called possible_titles where staff.title points to possible_titles.titles, then this method will return

```
array(
    possible_titles => array('title=titles')
)
```

An example to implement this rule using pseudocode will be like this:

```
if <its a data entry form>
    // check if the field is a foreign key using the above method
If <is field foreign key> and <number of values under (select count(distinct titles)
```
from possible_titles)

populate the form with pull down values from the table possible_titles

- Rule 4: if a column is a Boolean then it can be implemented as a group of radio buttons or drop down menu.

By using the same ADOdb method that is used in the second rule to get the type of the column, in the table ‘staff’ shown in Listing 6.1 the column ‘permanent’ is of type Boolean, and this is mapped to radio buttons see Figure 6.10.

If a column is of type enum (the data type consists of a predefined set of values with a specific order) then it can be implemented as a listbox but this would depend on both the enum type being available and DBMS specific system catalog information being needed. We could have a rule to deal with an enum type if a column is an enum, but we did not use this rule because this column type does not exist in all DBMS (for example Oracle) and also there is different syntax to create this type across different database systems, for example in PostgreSQL it is a separate type, but in MySQL it is used directly in table creation. In this case there is no generic approach to implementation and hence we did not implement this rule in our system.

In our approach each database column is mapped to a specific Web form element based on the database metadata and a set of rules. As shown in Figure 6.1 a
significant amount of information about each column in a database is retrieved. The major piece of information that can give an indication of how to perform the mapping task is a column data type.

2. **Column Name-Semantics Rules**

However, the data type portion of information on its own is not enough to generate an ideal Web entry form. For example, the only way to tell that a particular column is intended to be a password is its name. By using the database metadata semantics especially the semantics of columns’ name was the available solution because there exist no password data type in the database. To solve this problem, another rule is added to the rules list. For instance, the following assumption could be used to map a particular database column into a password text box in HTML.

- **Rule:** If a column data type is character and its length is between 8 to 12 characters and its name is either password or secret or subset of these words (e.g. passwd), then this column should be mapped into password text box.

Using the `MetaColumnNames($table)` method to get the column names of the table and regular expressions functions in PHP namely the function `ereg()` to compare a string to a certain pattern.

```php
for each column
    pattern = array('password', 'pass', 'pw', etc.);
    string = column name;
    if <ereg(pattern,string)>;
```
Framework Implementation and Evaluation

this column is mapped to password text box:

loop through all columns

endfor

It is possible to implement rules of this sort in a generic way since they only depend on the column’s name and are independent of any specific DBMS features.

3. Column Name-Labelling Rules

The default assumption is that the label is the actual value of the column’s name in the database. For improving the appearance of Web forms, a column’s name can be processed to produce a suitable label.

For example, a column’s name with more than one word separated with underscore could be represented by replacing underscores with space and the first character of all words in upper case. For instance, if a column’s name is last_name it could be presented as Last Name as shown in Figure 6.9 and Figure 6.10.

This set of rules is generic and can easily be extended because there is an obvious methodology to apply such rules.

4. Default Values

With most RDBMSs when you create a database table, you have the option to specify a DEFAULT value. Using default values on database columns helps to insulate database design issues from application code. Setting good default values not only helps ensure that you will get the data entered correctly, but it helps teach the user how the pieces of your application function.
In practice there are different ways to implement and inform default values with different DBMS. PostgreSQL allows the use of functions to define default values for columns (providing ultimate default value flexibility), in addition to the use of fixed values as defaults.

Note in the example in Listing 6.6 the column id does not have a default value in the create table statement but it has the behaviour of a column like timestamp where a default value is found via a function call on an insert statement where no value is given. In different systems a serial type is more or less clearly an alias for a function call to get the next value in a sequence.

In PostgreSQL although specified with the syntax shown in Listing 6.6 the table definition is saved in the system catalogs with a default statement of the form

```
not null default nextval('"numbered_id_seq"'::text)
```

We would also assume that the desired behaviour from the db designer's perspective is that the column should not be given a user input value.

Although there is no difference in SQL syntax we propose that defaults should be regarded as either: static or dynamic. Where a fixed default value is given we regard it as a static default, where a function call is made, either explicitly in the create table code or implicitly because of the type definition, as for example with the use of a serial, we regard it as a dynamic default.

For example, to demonstrate our point we created a table in PostgreSQL as shown in Listing 6.6.
CREATE TABLE books
{
    id serial NOT NULL,
    author_id int4 NOT NULL DEFAULT 50,
    name text NOT NULL DEFAULT 'PHP':text,
    available bool DEFAULT true,
    modtime timestamp DEFAULT now(),
    CONSTRAINT books_pkey PRIMARY KEY (id),
    CONSTRAINT books_author_id_fkey FOREIGN KEY (author_id)
        REFERENCES authors (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
}

Listing 6.6 PostgreSQL example to show a variety of defaults

The metadata about the database columns using a table specification similar to that shown in Listing 6.6 using PHP and its database metadata functions supported if needed by direct queries on the information schema or system catalogs is shown in Figure 6.4.

Host localhost
Server name: PostgreSQL 8.1.5 on i686-pc-linux-gnu, compiled by GCC g++ (GCC) 3.4.2 (mingw-special)
Database Name: postgres

Table name: books

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Not_Null</th>
<th>Unique</th>
<th>Primary</th>
<th>Foreign</th>
<th>Default</th>
<th>Default_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
<td>int4</td>
<td>4</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>nextval('books_id_seq'::regclass)</td>
<td>Dynamic</td>
</tr>
<tr>
<td>2</td>
<td>author_id</td>
<td>int4</td>
<td>4</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>50</td>
<td>Static</td>
</tr>
<tr>
<td>3</td>
<td>name</td>
<td>text</td>
<td>1</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>PHP\text</td>
<td>Static</td>
</tr>
<tr>
<td>4</td>
<td>available</td>
<td>bool</td>
<td>1</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>true</td>
<td>Static</td>
</tr>
<tr>
<td>5</td>
<td>modtime</td>
<td>timestamp</td>
<td>8</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>now()</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

Figure 6.4 Metadata information extracted from the table in Listing 5.2
From this distinction we propose a heuristic, which we then implement in the context of web data entry forms, that a static default for a column should appear to a user but be capable of being changed and that a dynamic default should mean that the column is not presented to the user and is not capable of being changed see Figure 6.5.

### 6.4 Automatically Generating the User Interface

The framework is capable of being used in a number of ways, from completely automatically where it has no separate interface; the code is simply called to create any Web form as required to a semi automatic style where the code is used to build a Web form without any knowledge of HTML or database semantics. The latter approach is illustrated in this section; the user interface design diagram is very simple see Figure 6.6. It is constructed using HTML Forms that are automatically
Framework Implementation and Evaluation

generated by invoking a PHP program. When the users access the URL of the system, they will see the Database name selection page as illustrated in Figure 6.7. The page displays a list of the available databases in the current DBMS, where a user can select the particular database of interest. Once the user selects the database, he then clicks the submit button, a new page gets displayed with a list of the chosen database’s tables as shown in Figure 6.8. So when a user selects a table of interest and clicks on the submit button, this invokes a PHP program and passes the selected table’s name as a requested parameter to obtain the table’s column metadata, then each column is mapped to an appropriate HTML user input element according to a set of general rules (as discussed in the previous section) and this is illustrated in Figure 6.10.
Database’s Name Form

Table’s Name Form

User Input Form

Figure 6.6 User Interface Design
Please select a database from the list below:

Select a database: postgres

Figure 6.7 Database name selection form
Figure 6.8 Table name selection form
Figure 6.9 HTML Web form generated on the fly from metadata.
The implemented methodology makes a clear separation between content, logic and representation. A small alteration to a style sheet file results in a new look Web user interface without the need to re-write or re-compile the source code on Web server see the difference between Figure 6.9 and Figure 6.10.

Figure 6.10 HTML Web entry form generated from database metadata on the fly, invoked in IE Web browser displayed using CSS
Figure 6.11 HTML Web entry form generated from database metadata on the fly, invoked in Fire Fox Web browser displayed using CSS.
6.5 Data Validation

Validation is essential to web database applications. There are three possible data environments in which validation can occur in a three-tiered web database application: in the DBMS, in server-side scripts, and on the client. DBMS validation can lead to poor/confusing messages getting to a user and since we are driven by better user experience we will test in this order, in the client side first because it’s quick and
friendly then we use server side test in case the client is turned off and finally the DBMS validation as a last resort. In our approach for the sake of simplicity we will focus on client side validation.

6.5.1 Client side validation

Client validation occurs at the client browser before a request is sent to the server and is usually validation of Web form data. The most common way to implement client validation is using JavaScript. Since an HTML Web form entry is generated automatically and dynamically on the fly, a generic method for data validation is needed to fit this approach. This generic method is a JavaScript function which is invoked to loop through the form elements to check for errors such as missing information or non-numeric characters in a numeric field etc. see Figure 6.13, Figure 6.14 and Figure 6.15.
Figure 6.13 Client side data validation
Figure 6.14 Required data validation
The drawback of validation at the client is that it depends on the user and his environment: the user can disable JavaScript, and can avoid the validation, and the client environment is not usually managed or standardized by the developer of the web database application. Client-side validation is not safe but may be implemented in addition to server-side validation.
6.5.2 Server side validation

Server-side validation is an essential validation tool. It is implemented in the middle-tier and uses PHP scripts to validate data and to make sure that all the Form Input data have a valid data values. PHP has many functions that are used to validate Form Input data. Some of these functions are `empty()` which is used to make sure that text boxes in a form are filled out. The second function is `isset()` which is used to validate non-text form elements such as radio buttons and checkboxes. The third function is `is_numeric()` which returns true if the submitted variable has a numerical value.

6.6 Framework Evaluation

Software evaluation and testing is an important part in the lifecycle process of software development, because the information it provides can be used to diagnose problems and to check the performance of a software product.

In order to evaluate the developed framework system and to show the generality of the metadata approach two different DBMS namely MySQL (5.0.26-community-nt) and Oracle (Database 10g Express Edition Release 10.2.0.1.0) were used to create the new databases.

Within the MySQL database, first the MySQL table with the same/similar structure to the example used with PostgreSQL development is created as in Listing 6.7 which has six columns of different types (e.g. int, varchar, date etc.).
CREATE TABLE staff
(
    staff_no int primary key,
    first_name varchar(30) not null,
    last_name varchar(30) not null,
    DoB date,
    Salary int,
    permanent Boolean,
);

Listing 6.7 MySQL database table’s structure

By using this table and changing only the connection string in the PHP script to connect to MySQL DBMS by using two methods, the first one is by using a standard connection string like the one shown below:

    $conn=&NewADOConnection('mysql');

    $conn->PConnect('host server', 'userid', 'password', 'database');

The second method is by passing DSN (Data Source Name) connection string to NewADOConnection( ) function as shown below:
Framework Implementation and Evaluation

dsn='mysql://root:mysqldb@localhost/mysql';

$conn=&NewADOConnection($dsn);

where ‘root’ is the user, ‘mysqldb’ is the password, ‘localhost’ is the host server and ‘mysql’ is the database name. This is the only change in the script that is required, the remaining code is the same as that used in the PostgreSQL example earlier, and the output is shown in Figure 6.16.

Figure 6.16 HTML Web entry form generated from MySQL database metadata on the fly
Secondly an Oracle database is created within which the same table is created in Oracle (10g Express Edition Release 10.2.0.1.0) with the exact number of columns as shown in Listing 6.8

```sql
CREATE TABLE "STAFF" (  "STAFF_NO" NUMBER,  "FIRST_NAME" CHAR(30) NOT NULL ENABLE,  "LAST_NAME" CHAR(30) NOT NULL ENABLE,  "DOB" DATE,  "SALARY" NUMBER,  "PERMANENT" CHAR(1),  CONSTRAINT "STAFF_PK" PRIMARY KEY ("STAFF_NO") ENABLE,  CONSTRAINT "STAFF_CON" CHECK ("PERMANENT" IN ( 'Y', 'N')) ENABLE )
```

**Listing 6.8 Oracle database table's structure**

Note the difference in create table syntax between the Oracle, PostgreSQL and MySQL versions.

There is no significant difference in the case of the specification of staff_no as a primary key, although the syntax to express this is different.

Because there is no direct column data type Bool or Boolean in Oracle, as an alternative to the Boolean data type a workaround would be creating a single character column, for example, a CHAR(1) data type (with Y (true) and N (false)) with a check constraint as
shown with column PERMANENT in listing 6.8. This can be mapped to the same type of Web Form element (radio buttons) as the Boolean of MySQL and PostgreSQL.

Figure 6.17 HTML Web entry form generated from Oracle database metadata on the fly

Comparing Figure 6.16 and Figure 6.17 to Figure 6.10 shows the similarity of the user interface.
The only noticed difference is in Figure 6.17 where the column names are all in capital letters and this is due to Oracle DBMS column definition.

Further development of the column naming rule (section 6.3) could be implemented to give first letter only capitalized output as occurs with other DBMSs whose columns are retrieved in lower case.

6.6.1 Arisen limitations

During the process of developing this framework some limitations have arisen and are considered in this section. The limitations focus around the nature of metadata available and its single table nature. That is, that metadata is essentially available on a “per table” basis and reflects the SQL used in a create table statement. The only connections between tables are those available via primary key /foreign key relationships and all the columns within a table have the same status, following relational theory, each is an independent attribute on an individual domain. Because of this we identify the following issues:

- Although connections can be made between tables in some cases, such as requiring a value to be present in table b before it is inserted in table a (as illustrated with selecting valid titles for a person), it is not possible to know that some tables which are connected should appear in a master-detail relationship as for example with order and order_line to use a classic scenario.
The mapping process is performed in a sequential manner. As a result, the web entry form controls will appear in the same order as the columns in every table (the ordering is that used in the create table statement). Therefore, if the database table is built up carelessly, the layout of web entry form controls will appear in a confused matter.

There is no clue to tell if there are some form controls which should be grouped together. As a result every form control will appear in a separate line even though there may be several form controls that could be represented in a single line such as `first_name` and `last_name`.

The issues of connections between tables could be approached by moving from a fully automated approach (as largely used here) to a more semi automatic approach as developed in Section 6.1 but extending that so that after selecting one table it could be specified as a master and a further table selected as the detail following the approach by some RDBMS 4GL tools (see Section 4.3 DBMS Application Development Tools) [73, 74].

The problem of ordering, grouping and exclusion of columns could also be tackled by extending the semi automatic production of Web Forms as developed in section 6.1. It would be possible after selecting a table to choose which columns to use and their order. Automated grouping would only seem to be possible if there were a greater semantic content to the metadata or if a more complex approach to column names were used.
DBMS's like PostgreSQL which offer complex types give the potential for grouping columns. For example defining an address type comprising street_address, town and post_code which would then be grouped as a composite web form control with three parts. But this approach would be limited to those RDBMSs that support those types and raises problems since those types are not supported well by metadata functions. Grouping based on column name semantics would be perhaps be possible if naming conventions were established, so the example above could become street_address, town_address and post_code_address using a convention that the same last word in a column name implied grouping.

An alternative approach to overcoming the limitations is via the use of views. Since views are virtual tables it is possible to access metadata for them in much the same way as has been done on base tables up to now; the metadata from a view is shown in Figure 6.18. This example uses selected columns from a single table but the same approach can be extended to reordering of columns and combining columns from more than one table.

The Form in Figure 6.19 is generated using the same PHP code as in the previous examples.
**Figure 6.18 Metadata from a view**

<table>
<thead>
<tr>
<th>Table_Name</th>
<th>staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column_Number</td>
<td>11</td>
</tr>
<tr>
<td>Column_Name</td>
<td>staff_no</td>
</tr>
<tr>
<td>Column_Type</td>
<td>int4</td>
</tr>
<tr>
<td>Column_Size</td>
<td>4</td>
</tr>
<tr>
<td>Nullable</td>
<td>YES</td>
</tr>
<tr>
<td>Unique</td>
<td>YES</td>
</tr>
<tr>
<td>Primary_Key</td>
<td>YES</td>
</tr>
<tr>
<td>Foreign_Key</td>
<td>NO</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Default_type</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table_Name</th>
<th>staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column_Number</td>
<td>12</td>
</tr>
<tr>
<td>Column_Name</td>
<td>first_name</td>
</tr>
<tr>
<td>Column_Type</td>
<td>varchar</td>
</tr>
<tr>
<td>Column_Size</td>
<td>NO</td>
</tr>
<tr>
<td>Nullable</td>
<td>YES</td>
</tr>
<tr>
<td>Unique</td>
<td>NO</td>
</tr>
<tr>
<td>Primary_Key</td>
<td>NO</td>
</tr>
<tr>
<td>Foreign_Key</td>
<td>NO</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Default_type</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table_Name</th>
<th>staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column_Number</td>
<td>13</td>
</tr>
<tr>
<td>Column_Name</td>
<td>salary</td>
</tr>
<tr>
<td>Column_Type</td>
<td>numeric</td>
</tr>
<tr>
<td>Column_Size</td>
<td>-1</td>
</tr>
<tr>
<td>Nullable</td>
<td>NO</td>
</tr>
<tr>
<td>Unique</td>
<td>NO</td>
</tr>
<tr>
<td>Primary_Key</td>
<td>NO</td>
</tr>
<tr>
<td>Foreign_Key</td>
<td>NO</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Default_type</td>
<td></td>
</tr>
</tbody>
</table>
Figure 6.19 Form generated from a view
6.7 Conclusion

The prototype system demonstrated the potential capabilities of relational database metadata and provides automatic generation of Web entry forms to ease the rapid deployment of interactive Web applications by developers with little Web development experience. Extracting database metadata is a simple task by using PHP and ADOdb. Converting this metadata type into generic data type simplifies transforming it into many Web form types and invoked in different Web browsers as shown in Figure 6.8, Figure 6.9 and Figure 6.10.

The implementation of the prototype has a number of advantages. First, the system is written in the open source and easy to use script language PHP and the abstract library ADOdb with metadata capabilities as mentioned in section 3.5.3. Second, the prototype system uses embedded system calls to communicate with the database. This does not restrict the portability of the system to different databases.

Knowing and retrieving a massive amount of information from database metadata does not make it a complete tool. Hard work on database metadata semantics is required in order to achieve a high level of accuracy.
Chapter 7

Conclusions and Future work

7.0 Conclusion

The Web is increasingly being used as the means to do everyday tasks. One common denominator for all these activities is the need to generate dynamic content. Frequent updates and searching/querying capabilities are the most common reasons behind dynamically generated Web sites.

The process of developing a web interface to a database is a complex task because of the variability in the data models that underlie each database. Adding to this diversity are differences in the number of tables in the model, in the names and the number of fields within each table, and the linkages and dependencies among the various tables.

By developing a heuristic that can be applied in automatic generation of web interfaces we can ensure that an additional feature of the database design, and its underlying meaning, can be included in applications. This should simplify the developers’ task and aid the user experience.

Previous work [1] and [97] showed that an automated metadata driven approach was possible using Java and JDBC but we have shown that there is a general concept and that it can be taken to other languages by using PHP, but with limitations mainly based on the metadata library capabilities. These limitations are of ease rather than fundamental principles.
Conclusions and Future Work

The metadata approach does have limits [98] but we have also found that the rule based approach is powerful and capable of extension within limits that depend on the level of metadata (which can vary, here contrasting the JDBC metadata classes with what we could get easily) and is capable of extension as we found with different forms of default. The decision was made to adapt ADOdb as the database abstraction library because of the set of facilities available in this library.

For this project any loss of performance due to the addition of an extra layer of library calls is not of significance where the issue of ease of transition across database servers is.

While these libraries are designed to allow database abstraction, you still have to be aware of the differences from one database to another and actually use the commands that help with that process.

Relational database was chosen as the database type (paradigm), because it is best in providing database metadata features, such as table and column definitions.

Our approach based on underlying database metadata, provided a number of advantages over other approaches e.g. framework approach. Since the output is in HTML format, there is no need for HTML page maintenance when the data changes. Any changes made to the database are reflected the next time the data is accessed. Customisation to the extracted names from the database catalogue to label the user interface is demonstrated.
Conclusions and Future Work

Although we have focused on contrasting our approach with that used in existing frameworks and focused on the lower level interface and programming issues it would be possible to integrate our approach into other frameworks replacing some of their reliance on conventions but keeping other advantages.

In conclusion, this approach aims to produce as generic code as possible hiding the DBMS specific functions inside our code so that we have a nice generic/abstract interface. Using PHP and the abstract library ADOdb will enable us to build interfaces that adapt to changes in database schema, data and are reusable across different databases.

7.1 Future Work

The purpose of this research was to explore to what extent we can use database metadata in conjunction with PHP to generate an abstract model for generation of Web entry forms on the fly. As a result of this exploration the following issues are raised and can be developed in further stages. We firstly consider developments that focus on the fully automated approach to metadata.

- Investigating the capabilities of relational database metadata in building interactive Web applications that use multimedia elements.

Increasingly databases are containing more multimedia content but as yet relational database metadata does not seem to be keeping pace with this. Many of the decisions about the correct web form element are based on type information from the metadata. At present the best support from metadata is
likely to be only that a multimedia element is stored as a large object (blob or clob), this is not enough information in itself to indicate how the data should be handled. Different “helpers” will be needed for video or audio content for example and even within one type such as video content various formats such as mpg, mpeg or flv files may have been used as the original source.

For input a standardized file browse and upload interaction which would work across formats could be developed but for playback without more specialized (and widely accepted) datatypes which are reflected in metadata systems are likely to have to rely on conventions like the use of file extensions to indicate types within column names, for example naming a column video_flv or text_docx to indicate a more specialized type and how it should be handled or displayed.

- Integrating our approach with AJAX in areas such as choosing unique passwords or usernames and as an alternative or addition to the simple JavaScript validation. Recent years have seen a growth in the attempts to produce Rich Internet Applications (RIA) with AJAX to the forefront in providing the desired interactivity. It would be possible to extend our automated production of JavaScript validation to automated AJAX. This would mean that current validation where a set of possible values is checked statically (in the web page) could be extended to more dynamic checking (on the server side) using AJAX. This can extend the range of data values that can be checked. For example values that should not be exposed to users for security reasons such as
other users' names can be checked on the server side. It also allows for a better user experience since relevant subsets of possible values can be displayed rather than a full range, in the fashion that current ecommerce sites use AJAX to “suggest” form completions as well as giving faster feedback when a unique value needs to be input.

- Since the set of rules developed throughout this research is very generic, it would be possible to develop domain specific rules to support the generic rules for manipulation of semantics of database metadata.

In the work presented an attempt has been made to make the rules used as generic as possible that is they should apply in any domain. Sets of rules that supplement the datatype information could be added for particular domains. For example any applications or databases in a banking domain that used a column called sort code could be assumed to follow the standard for that domain and require input of the form “dd-dd-dd” where d is a digit. Similarly all applications using book databases could have a rule that a column called “ISBN” which would be defined as char10 would in fact have to meet a stricter format of an ISBN where the first 9 characters must be digits and the 10th character position is a check digit using 0 – 9 and X as a substitute for 10.

The implementation of this type of approach would require the development of a method of applying generic rules followed by domain specific rules and ensuring that there was no possible rule conflict or that any conflict was solved by the application of the strongest i.e. most restrictive rule.
• Exploring the capabilities of the integration of XML technology with PHP in generating automatic Web entry forms using database metadata in particular investigate the possibility of using Xforms instead of HTML forms as a Web user interface.

There are a number of areas where web technologies are moving to use XML in addition or as a replacement for HTML. AJAX has been discussed above. Xforms is another area where developments in the facilities available on browsers could be exploited. Rather than the explicit coding of data validation in JavaScript Xforms would allow the data validation information to be specified in a more declarative way and then the Xform implementation determines how it is executed. This development is dependent on both the support of Xforms in browsers and the support for Xform development in libraries in PHP.

• Taking our approach into the Rails framework, since we know that we do not need to be so restrictive about for example primary key naming, since keys can always be found from metadata not simply a naming convention that it must be called something_id.

The Ruby on Rails framework and its PHP based clones strongly uses the notion on convention over configuration so at one level it would seem to operate in quite a different style from the approach in this work, however it could be argued that the automated approach used here to develop the application is in fact close in spirit to the Rails approach since it attempts to use standard approaches and produce standard outputs minimising programmer input.
The more significant contradiction may be that the Rails approach is essentially application centric, creating the database with its conventions for tables and columns from the application, and hence not being easily applied to legacy systems. If a Rails style approach can begin by addressing an existing database and extracting information from it using a metadata based approach as developed here to build an application the two apparently contrasting styles might be united.

There are also a number of areas where a semi automated approach, which focuses on less dynamic use of metadata or requires more user input.

- Using the same metadata based techniques to produce static Web Forms /Templates or Schema for other tools.

Although this work has focused on automatically producing the web forms whenever required and customising them with CSS it would be possible to use the metadata based techniques to produce what would be a first template for the web form which could then be modified by hand i.e. by a developer using tools such as Dreamweaver that are commonly used to build web pages. This approach has the advantage that metadata is consistently and automatically applied to web controls, overcoming any possible lack or error in documentation which might otherwise have been used, but its disadvantage is that without a “real time” connection to the database the interface may not be updated to reflect subsequent database changes.
Conclusions and Future Work

- Investigating the possibility of giving the user the choice of which table(s) and columns to use and how they will be presented.

A number of the issues in this area were considered in the discussion of limitations of our approach in the previous chapter. One way to implement user choice in which fields could be mapped to a user input element is by populating all the fields of a table as multi select list box or drop down menu where the user can choose the fields that can be shown on the user input form. A similar approach can be taken with multiple tables. The example of selecting a table from a list of all the tables could be extended to allow one table to be chosen as a master and then a detail table chosen from the other tables. If this approach was taken the connection between the tables could be automatic on the basis on primary key foreign references in the table definitions or with more user intervention by selecting columns from a metadata derived list. This approach although capable of some automation may require more user intervention, to be fully automated the generation of (possibly complex) SQL to address the multiple tables would be required. Ordering of columns into form elements can be tackled by associating a pull down number list with any column. The possible maximum number can itself be obtained from the table's metadata.

- Exploring the possibilities of the proposed approach to multiple databases.

A generalised approach was taken wherever possible leading to the use of ADOdb and a minimal set of metadata that could be found in any supported RDBMS. If this limitation were to be eased it should be possible to exploit the
more specialized or specific features of different DBMSs. We know that different systems have a range of features for example some supporting an explicit Boolean type and others using a single char instead also some systems have additional types like PostgreSQL's geometric types and network address types. If these types can be identified through the metadata functions for that particular DBMS then appropriate form controls could be deployed for them. This requires the correct DBMS, and possibly DBMS version, to be identified. This might be done automatically through metadata as can be seen in Fig 6.22 where the DBMS type and version are shown, or could be done with user intervention as in the examples where a particular table is selected, a DBMS and version could be chosen from a list by the user. The implications of extending the approach in this more specific way is that the code behind the system would get more complex as different DBMS specific branches were maintained, this might imply a different approach keeping the DBMS specific rules in some form of configuration file rather than embedding them too tightly in the application code. It could also mean that maintaining the system and making sure it worked with new versions became more complex.
References


http://www.w3.org/MarkUp/.
http://www.w3.org/TR/1998/REC-xml-19980210
us/library/d1wf56tt(VS.85).aspx.
us/library/yek4tbz0(VS.85).aspx.
[22] "ActiveX control",
[27] N. Foshay, A. Mukherjee, and A. Taylor, "Does data warehouse end-user
References


References


[43] "SQL Language Reference", 


[46] "The Information Schema", 
http://www.postgresql.org/docs/7.4/static/information-schema.html.


[48] "PostgreSQL: The world's most advanced open source database", 
http://www.postgresql.org/.


[50] G. Hamilton, R. Gattell, and M. Fisher, JDBC Database Access with Java: 
Addison-Wesley, 1997.

[51] Data Management: SQL Call Level Interface (CLI) (CAE Specification S.) 

[52] Microsoft, Microsoft ODBC 2.0 Programmer's Reference and SDK Guide: 


[55] "Four types of database abstraction layers", 
References


[57] "ADOdb Database Abstraction Library for PHP (and Python)",


[61] "Akelos PHP Framework: The pragmatic way to get Productive on PHP.",
http://www.akelos.org/.

[62] "PHP on TRAX - PHP Framework::Rapid Development Made Easy.",
http://www.phpontrax.com/.


[67] "Ruby on Rails Web Applications",


References


[84] "Understanding JDBC Metadata", 
http://members.aol.com/kgb1001001/Articles/JDBCMetadata/JDBC_Metadata.htm.

[85] M. M. Elsheh and M. J. Ridley, "Developing an Abstract Representation for User Interface Elements Based on Database Metadata" presented at IADIS
References


[92] R. Cattel, rick.cattel@sun.com.


References


[98] M. M. Elsheh, "Integration of Relational Database Metadata and XML Technology to Develop an Abstract Framework to Generate Automatic and Dynamic Web Entry Forms", in Department of Computing, PhD: University of Bradford, 2009.