Università degli studi di Milano

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE e NATURALI

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE DELLA COMUNICAZIONE MUSICALE

"INTERFACCE WEB PER L'INTERROGAZIONE DI REPOSITORY IEEE 1599"

Giovanni Sala

Tesi di Laurea

Relatore: *Prof. Luca Andrea Ludovico*

Correlatore: *Dott. Adriano Baratè*

Anno accademico 2008/2009

Indice

Indice delle figure

Introduzione

Obiettivi della tesi

Nel giugno del 2007 è stato approvato, come standard a livello mondiale, dalla IEEE-Standard Association, organo della IEEE, il nuovo formato IEEE 1599, basato sul linguaggio XML. Questo nuovo linguaggio consente di far interagire l'informazione musicale con tutti i livelli del testo, del suono, delle partiture, del timbro attraverso una gestione spazio-temporale degli elementi. Permette, quindi, di rappresentare la musica in tutti i modi con la quale la si può identificare (notazionale, simbolico, strutturale, audio...).

Durante il corso degli ultimi due anni, sono stati molti gli articoli a livello europeo e mondiale, e molte sono state le tesi presentate, atte a sviluppare l'utilizzo di questo nuovo linguaggio, sia a livello teorico, sia a livello pratico come la creazione di software per la conversione da altri linguaggi basati su XML a IEEE 1599, o software di analisi di partiture, file audio, attraverso la lettura di file IEEE 1599.

Lo scopo di questa tesi è stato quello di sviluppare un'interfaccia per la ricerca di file IEEE 1599-2008 da inserire all'interno del sito www.mx.dico.unimi.it, in modo da rendere possibile il recupero dei file in questo formato da parte di qualsiasi utente che non abbia a disposizione il framework MX.

Questa ricerca si rifà al database che è stato sviluppato da un altro stagista, sempre per conto del Laboratorio di Informatica Musicale (LIM).

Il database è stato sviluppato utilizzando l'ambiente PostgreSQL, questo ha dato delle direttive implicite per lo sviluppo dell'interfaccia web.

Per sviluppare questa interfaccia è stato svolto uno stage della durata di due mesi dove sono state approfondite le conoscenze di HTML, PHP e CSS in quanto sono state le tecnologie utilizzate per sviluppare l'interfaccia.

La loro scelta è stata dovuta anche per il fatto che il sito di partenza era già stato sviluppato con esse. Di conseguenza è stato deciso di mantenere l'uniformità delle tecnologie.

Inoltre sono state approfondite le conoscenze dello standard IEEE 1599, soprattutto per quanto riguarda la sua struttura, in quanto i campi di ricerca dell'interfaccia si riferiscono ai corrispettivi campi dello standard.

Qui di seguito viene spiegata come è stata strutturata questa tesi e quali argomenti vengono trattati al suo interno.

Dovendo trattare con file del formato IEEE 1599-2008 viene spiegata la loro struttura, in particolare i layer General, Audio e Notational in quanto sono quelli a cui si riferisce la ricerca. Ovviamente per comprendere meglio il funzionamento viene spiegato anche il linguaggio di markup XML.

Nel capitolo successivo vengono spiegate le tecnologie utilizzate per costruire l'interfaccia, cioè PHP, HTML e CSS in particolare come funzionano e da cosa sono regolate (W3C). Viene data una breve descrizione sul mondo dei database e in particolare sull'ambiente PostgreSQL.

Per finire nell'ultimo capitolo ci viene fatta l'analisi dell'interfaccia web sviluppata. Vengono analizzate tutte le parti del programma, la funzione di connessione, le form di ricerca e i metodi utilizzati per la stampa dei risultati. All'interno del capitolo vengono presentate ed analizzate parti di codice del programma.

Capitolo 1

XML e IEEE 1599

1.1 Cos’è un linguaggio di Markup

Il termine markup (o marcatura) deriva dall'ambiente tipografico dove si usava marcare con annotazioni le parti del testo che andavano evidenziate o corrette, allo scopo di segnalarle al compositore o al dattilografo.

Un linguaggio di marcatura è un linguaggio nato per segnalare, attraverso opportune istruzioni, le caratteristiche logiche di un documento e delle sue parti: ad esempio, la funzione di titolo svolta da una determinata porzione di testo. Le istruzioni di un linguaggio di marcatura sono interpretate dal browser, che decide come visualizzare la relativa informazione[8].

I diversi linguaggi di markup esistenti si distinguono fondamentalmente in:

- linguaggi di markup di tipo procedurale;
- linguaggi di markup di tipo descrittivo.

La differenza tra i due sta nel meccanismo usato per definire la rappresentazione del testo, sia per quanto riguarda la sua struttura, sia per quanto riguarda il significato degli elementi che lo compongono, sia per quanto riguarda la visualizzazione (o formattazione):

- i linguaggi di markup di tipo procedurale indicano le procedure di trattamento del testo aggiungendo le istruzioni che devono essere eseguite per visualizzare la porzione di testo referenziata;
- i linguaggi di markup di tipo descrittivo lasciano la scelta del tipo di rappresentazione da applicare al testo al software che di volta in volta lo riprodurrà.

I linguaggi del secondo tipo risultano più vantaggiosi perché si concentrano sui problemi strutturali di leggibilità e prescindono in fase di lettura dal software con cui sono stati generati. Sono, in altre parole, quelli che permettono di garantire una corretta separazione tra struttura e visualizzazione.

L'SGML (Standard Generalized Markup Language) è stato il primo metalinguaggio di markup descrittivo standardizzato a livello internazionale (ISO 8879 del 1986) che ha definito dei metodi di rappresentazione del testo in forma elettronica in modo indipendente dall'hardware e dal sistema utilizzato. L'SGML è basato sul concetto di definizione del tipo di documento o Document Type Definition (DTD) ovvero richiede per ogni documento la definizione del modo in cui i vari elementi del testo possono essere utilizzati. Ad esempio una lettera contiene degli elementi essenziali quali mittente, uno o più destinatari, data, oggetto, corpo, l'indicazione di colui che la firma, ... Tutti elementi che devono essere presenti, probabilmente anche con un certo ordine. La DTD deve prendersi carico di definire tutto questo, stabilendo ciò che è legale e cosa invece non lo è.

1.2 **XML**

XML significa eXtensible Markup Language ed è un metalinguaggio di markup, cioè un linguaggio che permette di definire altri linguaggi di markup.

In realtà, XML di per sé non è altro che un insieme standard di regole sintattiche per modellare la struttura di documenti e dati[8]. Questo insieme di regole, dette più propriamente specifiche, definiscono le modalità secondo cui è possibile crearsi un proprio linguaggio di markup. Le specifiche ufficiali sono state definite dal W3C (World Wide Web Consortium).

XML fondamentalmente è il risultato del tentativo di produrre una versione semplificata di SGML che appunto permettesse di definire nuovi semplici linguaggi di markup.

La parola eXtensible sta a indicare la possibilità di creare tag personalizzati

XML nasce per opera del W3C dalla necessità di un linguaggio di markup che dia maggiore libertà nella definizione dei tag, pur rimanendo in uno standard, in quanto all’inizio degli anni novanta, a causa delle guerre tra browser, risultava difficile riuscire a visualizzare correttamente una pagina web su browser diversi.

Questo linguaggio nato come soluzione di un problema di standard per il Web, in realtà risulta essere abbastanza generale per poter essere utilizzato in diversi contesti: definizione della struttura di documenti, scambio di informazioni tra sistemi diversi, rappresentazione di immagini, definizione di formati di dati, mezzo per l'esportazione di dati tra diversi DBMS.

Per di più XML contribuisce all'evoluzione dell'HTML, arrivando all'XHTML, questo però verrà descritto nel capitolo successivo.

1.2.1 *La struttura di XML*

Un documento XML è un file di testo che contiene una serie di tag, attributi e testo secondo regole sintattiche ben definite. Il file deve avere l'estensione *.xml*.

Per comprendere la sua struttura è bene analizzarlo dal punto di vista logico guardando cioè la sua struttura logica.

Un documento XML è intrinsecamente caratterizzato da una struttura gerarchica. Esso è composto da componenti denominati elementi. Ciascun elemento rappresenta un componente logico del documento e può contenere altri elementi (sottoelementi) o del testo.

Gli elementi possono avere associate altre informazioni che ne descrivono le proprietà. Queste informazioni sono chiamate attributi.

L'organizzazione degli elementi segue un ordine gerarchico o arboreo che prevede un elemento principale, chiamato root element o semplicemente root o radice.

La radice contiene l'insieme degli altri elementi del documento. Possiamo rappresentare graficamente la struttura di un documento XML tramite un albero, generalmente noto come document tree.

La struttura logica di un documento XML dipende dalle scelte progettuali. Sarà il progettista a decidere come organizzare gli elementi all'interno di un documento XML. Non esistono regole universali per l'organizzazione logica di un documento.

La struttura logica di un documento XML viene tradotta in una corrispondente struttura fisica composta di elementi sintattici chiamati tag. Essi vanno a rappresentare ogni singolo elemento. Questa struttura fisica viene implementata tramite un file di testo creato con un qualsiasi editor.

Per capire meglio il significato prendiamo un file XML come esempio:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "http://standards.ieee.org/downloads/1599/1599-
      2008/ieee1599.dtd">

<ieee1599 version="1.0" creator="Luca A. Ludovico">
      <general>
            <description>
                  <main_title>Gottes Macht und Vorsehung</main_title>
                  <author type="composer">Beethoven, Ludwig van</author>
                  <author type="poet">Christian Fürchtegott
                        Gellert</author>
                  <number>5</number>
                  <work_title>6 Geistliche Lieder Von Gellert</work_title>
                  <work_number>op. 48</work_number>
                  <genres>
                        <genre name="Vocal music" />
                        <genre name="Romanticism" />
                  </genres>
            </description>
      </general>
</ieee1599>
```

Come detto prima i tag vanno a rappresentare ogni elemento, guardando il codice si nota che in realtà l'elemento è delimitato all'interno di due etichette: il tag di apertura (start tag) e quello di chiusura (end tag). I due tag devono differenziarsi solo per il carattere "/" anteposto alla stringa all'interno del tag di chiusura.

La prima riga del documento lo identifica come un documento XML e ne specifica la versione (in questo caso la 1.0):

```
<?xml version="1.0" encoding="UTF-8"?>
```

In questo caso viene anche specificato lo schema di codifica utilizzato all'interno del documento, ma questo è un attributo, quindi non è obbligatorio.

Il corpo vero e proprio del documento segue questa prima riga, rappresentando gli elementi tramite i tag.

In XML i tag non sono predefiniti, esso ci lascia liberi di definire quelli che vogliamo. Per specificare un attributo per un elemento inseriamo il nome dell'attributo con il relativo valore all'interno del tag di apertura dell'elemento. L'organizzazione gerarchica degli elementi viene rappresentata in XML tramite il loro annidamento. In altre parole, ogni elemento potenzialmente contiene altri elementi, detti sottoelementi (sub-elements).

Alcuni elementi possono essere vuoti, cioè possono essere privi di contenuto testuale.

XML prevede che vengano sempre specificati i tag di apertura e chiusura.

Tuttavia XML prevede una sintassi abbreviata per gli elementi vuoti che evita di dover specificare il tag di chiusura. È infatti sufficiente terminare il tag di apertura con la sequenza di caratteri "/>", come nel caso del tag *genre:*

```
<genre name="Vocal music" />
```

Da questa linea di codice si nota la presenza dell'attributo *name*. Come già detto in precedenza gli attributi sono delle informazioni che servono per descrivere delle proprietà di un elemento.

Essi sono sempre associati allo *start tag* e ogni elemento può avere un numero qualsiasi di attributi.

Come si vede dal codice l'attributo è costituito da due parti:

- nome, che precede il segno "=";
- valore, che segue il segno "=" ed è racchiuso tra virgolette.

Nel caso che un elemento abbia più attributi questi sono separati tra loro con uno spazio.

All'interno dell'XML troviamo la possibilità di inserire commenti in qualsiasi punto del codice, essi dovranno essere racchiusi tra le seguenti sequenze di caratteri:

- `<!--`, apertura;
- `-->`, chiusura.

1.2.2 *Well formed e validity*

Tutti i documenti XML devono sottostare a un principio fondamentale: devono essere ben formati (well formed).

Un documento é detto, appunto, well-formed (ben-formato) se segue le regole grammaticali fornite dal W3C. Queste regole sono le seguenti:

- ogni documento XML deve contenere un unico root element che contenga tutti gli altri elementi del documento. Le sole parti di XML che possono stare all'esterno di questo elemento sono i commenti e le direttive di elaborazione (per esempio, la dichiarazione della versione di XML);
- ogni elemento deve avere un tag di chiusura o, se vuoti, possono prevedere la forma abbreviata (/>);
- gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'ordine inverso dei rispettivi tag di apertura;
- XML è case sensitive ergo fa distinzione tra maiuscole e minuscole, per cui i nomi dei tag e degli attributi devono coincidere nei tag di apertura e chiusura anche in relazione a questo aspetto;
- i valori degli attributi devono sempre essere racchiusi tra singoli o doppi apici.

Per quanto riguarda il contenuto, un documento XML può contenere potenzialmente qualsiasi carattere dell'alfabeto latino, cifre e punteggiatura. Normalmente vengono accettati come caratteri validi in un documento XML i primi 128 caratteri della codifica ASCII (lettere dell'alfabeto latino minuscole e maiuscole, cifre, segni di punteggiatura, ecc.).

Se un documento contiene caratteri che non rientrano tra questi (es.: lettere accentate, simboli di valuta, ecc.) è necessario specificare lo schema di codifica utilizzato (come visto nell’esempio precedente).

Potrebbe essere necessario inserire in un documento XML dei caratteri particolari che potrebbero renderlo non ben formato. Ad esempio, se dobbiamo inserire del testo che contiene il simbolo <, corriamo il rischio che possa venire interpretato come l'inizio di un nuovo tag.

Per evitare situazioni di questo tipo, XML prevede degli oggetti speciali detti entità che consentono di sostituire altri caratteri.

Oltre al concetto di *well formed* è importante considerare quello di *validità* del documento.

Un documento si dice *valido* per un particolare linguaggio se rispetta le regole definite da una grammatica.

Una grammatica è un insieme di regole che indica quali vocaboli (elementi) possono essere utilizzati e con che struttura è possibile comporre frasi (documenti).

La caratteristica di documento valido si affianca a quella di documento ben formato per costruire documenti XML adatti ad essere elaborati automaticamente.

C'è da sottolineare che un documento ben formato può non essere valido rispetto ad una grammatica, mentre un documento valido è necessariamente ben formato. tra l'altro, un documento valido per una grammatica può non essere valido per un'altra grammatica.

Per poter definire una grammatica ci sono due metodi differenti:

- DTD - Document Type Definition;
- XML Schema.

Un DTD è un documento che descrive i tag utilizzabili in un documento XML, la loro reciproca relazione nei confronti della struttura del documento e altre informazioni sugli attributi.

Al suo interno troviamo principalmente due dichiarazioni:

1. Element Type Definition, definisce gli elementi utilizzabili nel documento e la struttura del documento stesso;
2. Attribute List Declaration, definisce la lista di attributi per ciascun elemento.

Analogamente ad un DTD, un XML Schema è una descrizione formale di una grammatica per un linguaggio di markup basato su XML.

A differenza di un DTD, che utilizza una propria sintassi specifica, un XML Schema utilizza la stessa sintassi XML per definire la grammatica di un linguaggio di markup.

Quindi uno XML Schema è un documento XML che descrive la grammatica di un linguaggio XML utilizzando un linguaggio di markup specifico. In quanto documento XML, uno XML Schema ha un root element che contiene tutte le regole di definizione della grammatica.

1.3 **IEEE 1599**

Come detto in precedenza lo standard IEEE 1599 è stato pensato per rappresentare la musica in tutti i modi con la quale la si può identificare.

Gli autori di questo formato hanno identificato sei diversi strati (layers) per descrivere l'informazione musicale:

1. general;
2. music logic;
    - spine;
    - los (logically organized symbols);
    - layout.
3. structural;
4. notational;
5. performance;
6. audio.

Ogni layer rappresenta un elemento figlio del tag radice.

Ecco la stringa presente nel DTD:

```
<!ELEMENT mx (general, logic, structural?, notational?, performance?,
      audio?)
```

Guardando la cardinalità dei layer si nota che gli unici layer obbligatori sono general e music logic, tutti gli altri sono facoltativi.

Il nodo centrale di questo formato è rappresentato dal sotto elemento del Layer Music Logic spine.

Lo spine è una struttura astratta che rappresenta la relazione spazio temporale tra i diversi formati che rappresentano la musica[1]. Questo sotto elemento è il cuore di IEEE 1599, ogni layer, ad eccezione del layer general, fa riferimento ad esso. Esso integra i layer notational, performance e audio con il Music Logic, supera le differenze di codifica, si rende punto di riferimento per le istanze appartenenti a layer diversi, o anche allo stesso layer.

Ora vengono analizzati qui di seguito i layer general, notational e audio, in quanto il progetto sviluppato in questa tesi va ad utilizzare alcuni elementi di essi come campi per la ricerca.

1.3.1 *Layer General*

Questo layer è l'unico elemento a se stante e che quindi non è strettamente collegato con il resto.

Prende in considerazione il brano musicale nella sua interezza, dandone un inquadramento generale[1].

Questo layer ha diversi sottoelementi, alcuni dei quali non sono ancora stati implementati:

```
<!ELEMENT general (description, casting?, related_files?, analog_media?,
      notes?, rights?)>
```

Tra questi sottoelementi il più importante ai fini della descrizione di un brano è description, il quale a sua volta è composto da diversi elementi:

```
<!ELEMENT description (work_title?, work_number?, movement_title,
      movement_number?, genre?, author+)>
```

Ecco una breve descrizione di questi elementi:

- *work title:* è un elemento testuale che descrive il titolo dell'opera complessiva;
- *work number:* identifica il numero, se esiste, del catalogo dell'opera da cui è tratto il brano;
- *movement_title:* riguarda il brano, descrive il nome del movimento (brano);
- *movement_number:* riguarda il brano, descrive il numero del movimento (brano);
- *author:* descrive l'autore o gli autori;
- *author type:* specifica la tipologia dell'autore (revisore, librettista, compositore ecc…);
- *genre:* specifica il genere dell'elemento musicale in analisi.

Questi elementi, a parte *movement_title*, sono stati tutti rappresentati all'interno del database.

1.3.2 *Layer Notational*

Il layer Notational si riferisce alle istanze “visive” di un pezzo musicale.

Sono state identificate due modalità di scrittura e lettura della musica:

- notazionale (notational);
- grafica (graphical).

Un’istanza notazionale è spesso in formato binario, come i NIFF, e rappresenta informazione simbolica. Altre istanze di questo tipo in formati XML alternativi sono: MusicXML, MEI e MML.

Un’istanza grafica, invece, contiene immagini che rappresentano la partitura. Anch’essa solitamente si presenta in formato binario (ad esempio file JPEG, TIFF o PDF). L’informazione di questo layer si lega alla parte spaziale dello spine, il che ne consente la corretta localizzazione.

All’interno del database sono state collegate solamente le *graphic instance*, quindi di seguito viene descritto solo questo elemento.

```
<!ELEMENT graphic_instance_group (graphic_instance+)>
<!ATTLIST graphic_instance_group
   description CDATA #REQUIRED>
```

Come si vede dal codice *graphic_instance* è un sotto elemento di *graphic_instance_group*, perché il primo rappresenta una singola pagina della partitura, il secondo invece rappresenta la partitura intera, fa da “raccoglitore” delle varie *graphic_instance*.

Graphic_instance_group ha come unico attributo *description*, il quale contiene una descrizione testuale del gruppo. Esso è stato rappresentato all’interno del database.

Ecco ora l’elemento *graphic_instance*:

```
<!ELEMENT graphic_instance (graphic_event+, rights?)>
<!ATTLIST graphic_instance
   description CDATA #IMPLIED
   position_in_group CDATA #REQUIRED
   file_name CDATA #REQUIRED
   file_format %formats; #REQUIRED
   encoding_format %formats; #REQUIRED
   measurement_unit (centimeters | millimeters | inches |
```

```
                    decimal_inches | points | picas | pixels | twips)
                        #REQUIRED>
```

Come si nota dal codice *graphic_instance* contiene molti più attributi:

- *description:* contiene una descrizione testuale dell'elemento;
- *position_in_group:* indica la posizione della pagina all'interno della partitura (gruppo);
- *file_name:* indica l'indirizzo dell'elemento;
- *file_format:* deve fare parte dei formati presenti nel DTD di IEEE 1599;
- *enconding_format:* normalmente non c'è differenza tra *file_format* e *encoding_format,* alcuni formati però vengono considerati come contenitori di diversi formati. In questo caso *file_format* indica il contenitore, *encoding_format* indica il singolo formato;
- *measurement_unit:* fissa l'unità di misura utilizzata nel sotto elemento *graphic_event.*

Questi attributi, a parte *measurement_unit,* sono stati rappresentati all'interno del database.

1.3.3 *Layer Audio*

Il layer audio descrive le proprietà del materiale musicale audio. Si tratta del livello più basso nella visione multi-strato[1].

I formati per la rappresentazione dell'informazione audio si possono suddividere in due categorie: compressi e non compressi. A loro volta, i formati compressi possono avere perdita di informazioni (codifiche lossy) o non averne (codifiche lossless).

Per poter collegare un generico file audio allo spine, è necessario estrarre le caratteristiche degli eventi musicali relative alla loro localizzazione temporale. Si tratta di informazioni che non dipendono dal particolare formato di codifica o dall'eventuale compressione.

Capitolo 2

Tecnologie utilizzate

2.1 **HTML**

Letteralmente significa HyperText Markup Language ed è un linguaggio usato per descrivere la struttura dei documenti ipertestuali disponibili nel World Wide Web ossia su Internet[4]. Tutti i siti web sono scritti in HTML, codice che viene letto ed elaborato dal browser, il quale genera la pagina che viene visualizzata sullo schermo del computer.

L'HTML è un linguaggio di pubblico dominio la cui sintassi è stabilita dal World Wide Web Consortium (W3C).

L'ultima versione è stata rilasciata nel 1999 ed è la 4.01, si rimanda al paragrafo ***XHTML*** per capire perché non ci sono più state nuove versioni.

Come dice la sigla stessa, HTML non è un linguaggio di programmazione ma un linguaggio di Markup, nello specifico fa parte della famiglia dei linguaggi SGML.

2.1.1 *Struttura dell'HTML*

Ogni documento scritto in HTML per essere riconosciuto come tale deve essere contenuto in un file avente estensione *.html* o *.htm*

Come per XML, anche nel caso dell'HTML il componente principale della sintassi di questo linguaggio è l'elemento, inteso come struttura di base a cui è delegata la funzione di formattare i dati o indicare al browser delle informazioni.

Anche qui l'elemento è racchiuso tra i tag di apertura e di chiusura i quali hanno la stessa struttura di quelli dell'XML.

La differenza grossa rispetto all'XML è che qui i tag non vengono definiti dall'utente ma sono già stabiliti dal linguaggio.

Un documento HTML inizia sempre con il tag *<!doctype>* il quale ci consente di specificare di che tipo di documento si tratta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
http://www.w3.org/TR/html4/strict.dtd>
```

Questa indicazione segnala al browser l'URL delle specifiche HTML utilizzate per il documento, indicando quindi, implicitamente, quali elementi, attributi ed entità si possono utilizzare e a quale versione di HTML si fa riferimento. Di fatto, questa informazione serve al browser per identificare le regole di interpretazione e visualizzazione appropriate per lo specifico documento. Questa definizione deve pertanto precedere tutti i tag relativi al documento stesso.

In particolare è da notare l'elemento "`DTD HTML 4.01 Strict`", il quale sta a indicare che il documento fa riferimento a una DTD ("Document Type Definition"); la versione di HTML supportata è la 4.01 "strict".

Oltre alla *strict* ci sono altre due versioni: *transitional* e *frameset*.

Ecco il significato delle tre versioni:

- Transitional: nato per favorire la migrazione dalle vecchie versioni di HTML. Accetta tag definiti *deprecati* ed è tollerante riguardo ad alcune non conformità sintattiche;
- Strict: versione più restrittiva, non accetta tag *deprecati*, non è tollerante riguardo alle non conformità sintattiche ed esclude ogni elemento riguardante il layout (la cui formattazione è affidata ai CSS);
- Frameset: nato per motivi di compatibilità per suddividere la finestra visualizzata dal browser in diversi frame (sottofinestre), pratica resa *deprecata* dal w3c.

Al di sotto del tag *<!doctype>* troviamo il corpo vero e proprio del documento, il quale a sua volta è una struttura annidata dove ogni parte è delimitata da specifici tag.

Il corpo è delimitato innanzitutto dai tag <*html*> e </*html*>, al loro interno troviamo che il codice è contenuto in due sezioni fondamentali:

- la sezione di intestazione o *header*, delimitata tra i tag <*head*> e </*head*>, che contiene informazioni di controllo normalmente non visualizzate dal browser, con l'eccezione di alcuni elementi;
- la sezione del corpo o *body*, delimitata tra i tag <*body*> e </*body*>, che contiene la parte informativa vera e propria, ossia il testo, le immagini e i collegamenti che costituiscono la parte visualizzata dal browser.

2.1.2 *XHTML*

L'acronimo sta a indicare eXtensible HyperText Markup Language, è la versione più aggiornata dell'HTML.

Detto in parole semplici è una pagina HTML scritta in conformità con lo standard XML (eXtensible Markup Language).

Questo standard è stato rilasciato per la prima volta dal w3c nel 2000.

La scelta di rendere l'HTML conforme alle regole dell'XML è stata fatta per vari motivi, uno su tutti quello di fare in modo che le pagine possano mantenere conformità su diversi apparati, anche quelli con prestazioni basse che non permettono di interpretare l'HTML correttamente[4].

Passando a XHTML si è arrivati ad avere un linguaggio con regole più restrittive che aiutano a evitare soluzioni poco "belle" dal punto di vista del codice.

Nell'XHTML si punta a sfruttare i CSS nella maniera migliore così da tenere separate le parti di struttura e di presentazione.

Ovviamente è stata mantenuta la compatibilità con i software che supportano l'HTML 4.0.

Le versioni di XHTML rilasciate finora sono tre:

- XHTML 1.0;
- XHTML Basic;
- XHTML 1.1.

La versione 1.0 si distingue a sua volta in tre sotto categorie, le stesse dell'HTML (vedi paragrafo precedente).

La versione Basic è stata sviluppata appositamente per dispositivi mobili ed è una versione "ridotta" del linguaggio. Contiene solo gli elementi che si adattano a questi dispositivi.

Infine la versione 1.1 si basa sul DTD della versione 1.0 Strict, scelta fatta per escludere soluzioni definite *deprecate*. La novità sta nel fatto che è stata pensata come un insieme di moduli indipendenti contenenti ognuno gli elementi fondamentali. Questi moduli possono essere implementati o esclusi in base alle necessità.

2.2 **CSS**

Letteralmente Cascading Style Sheet (fogli di stile a cascata), questi CSS sono diventati fondamentali per lo sviluppo di internet in quanto servono a sopperire la maggior lacuna dell'HTML e cioè la parte di "presentazione".

In realtà non è giusto parlare di lacuna in quanto l'HTML non era nato per quello, a mano a mano che si è sentito sempre più il bisogno di migliorare il risultato grafico delle pagine sono stati inseriti nuovi tag per la formattazione che in realtà andavano a complicare il codice[4].

Da qui è nata l'idea e il bisogno di separare la parte strutturale del codice con tutto ciò che concerne la parte estetica, così si è arrivati proprio ai CSS.

Le regole per comporre i fogli di stile sono contenute in un insieme di direttive (Recommendations) emanate dal W3C, la cui prima specifica ufficiale (CSS1) è stata emanata nel 1996.

Nel 1998 si è passati alla versione 2 (CSS2) dove sono state aggiunte molte funzioni ma comunque la struttura di base è rimasta la stessa. Nel 2004 è uscita la versione 2.1, anche qui non troviamo stravolgimenti. La nuova specifica CSS3 è ancora in via di sviluppo.

Un problema che non si può tralasciare parlando di CSS è quello della compatibilità in quanto gli strumenti di navigazione che supportano le specifiche CSS ci sono ma bisogna considerare anche i vecchi browser e le diverse modalità di rendering di certe proprietà.

Il primo problema è facile da capire ma difficile da risolvere: non si può obbligare un utente a cambiare browser di conseguenza si rende necessario creare un CSS apposta per i vecchi browser, in quanto le specifiche sono differenti.

Ovviamente per poterlo fare e per fare in modo che venga utilizzato il CSS giusto in base al browser che sta facendo la richiesta ci sono varie tecniche e vari metodi che qui però non vengono spiegati.

Il secondo problema sta nel fatto di vedere come ogni singolo browser applica una determinata proprietà supportata. Ad esempio un browser supporta l'uso di parole chiave per definire la dimensione del font ma lo fa a modo suo interpretando la data parola in maniera errata.

Considerando questi problemi è necessario porre attenzione quando si va a sviluppare un foglio di stile. Una buona prassi è quella di provare il proprio codice sul maggior numero di browser così da potersi rendere conto dei vari problemi e poter cercare di porvi rimedio.

2.2.1 *A cosa servono i fogli di stile*

Grazie ai fogli di stile ora è possibile andare a modificare la pagina in modo che il testo abbia un aspetto da word-processor: oltre al font e al colore con i CSS è possibile modificare l'interlinea, la spaziatura delle lettere e delle parole, è possibile utilizzare stili diversi per titoli e paragrafi, sfruttare l'indentazione[8].

Per di più è possibile ordinare meglio gli elementi del foglio con un semplice sistema di margini senza dover più inserire immagini vuote o cose simili. Si possono anche inserire bordi a parti della pagina che non sono tabelle.

La caratteristica più importante però è la comodità di modifica. Se dobbiamo cambiare qualcosa dal punto di vista estetico, come ad esempio l'immagine di fondo, non è più necessario cambiare ogni pagina di codice ma basta semplicemente modificare il foglio CSS.

2.2.2 *Come inserire il foglio di stile all'interno del codice*

Per inserire il foglio di stile all'interno di un documento ci sono diverse modalità. Una nel caso di CSS esterno e due nel caso di CSS interno.

Per CSS esterno si intende che il foglio di stile è definito all'interno di un file diverso da quello che contiene il codice HTML, che avrà l'estensione *.css.*

Per CSS interno invece si intende quando il codice del foglio di stile è compreso in quello del documento.

Le tre modalità di inserimento dei fogli di stile sono:

- collegati (CSS esterno);
- incorporati (CSS interno);
- in linea (CSS interno).

Nel caso dei fogli collegati l'inserimento può essere fatto in due modi diversi.

Il primo, che è quello più compatibile, viene fatto inserendo il tag <*link*> all'interno del tag <*head*>.

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="foglio_di_stile.css">
</head>
```

Questo tag ha quattro attributi importanti, i primi tre obbligatori mentre il quarto opzionale:

1. *rel*: descrive il tipo di relazione tra il documento e il file collegato. Per i CSS due sono i valori possibili: stylesheet e alternate stylesheet;
2. *href*: serve a definire l'URL assoluto o relativo del foglio di stile;
3. *type*: identifica il tipo di dati da collegare. Per i CSS l'unico valore possibile è text/css;
4. *media*: con questo attributo si identifica il supporto (schermo, stampa, etc) cui applicare un particolare foglio di stile.

Il secondo modo è quello di utilizzare la direttiva *@import* all'interno del tag <*style*>:

```
<style>
@import url(stile.css);
</style>
```

In questo caso il CSS va collegato definendo un URL assoluto o relativo da racchiudere tra parentesi tonde e la dichiarazione deve chiudersi con un punto e virgola.

Nel caso di fogli incorporati essi vengono inseriti direttamente all'interno del documento HTML, in particolare viene inserito il tag <*style*> all'interno del tag <*head*>:

```
<html>
<head>
<title>Inserire i fogli di stile in un documento</title>
<style type="text/css">
body {
background: #FFFFCC;
}
</style>
</head>
```

Il tag <*style*> può avere due attributi:

1. *type*;
2. *media*.

Per questi due tag valgono le osservazioni fatte in precedenza, si comportano ugualmente a quelli visti all'interno del tag <*link*>, tant'è che anche qui l'attributo *type* è obbligatorio mentre l'attributo *media* è opzionale.

Infine nel caso dei fogli in linea viene utilizzato l'attributo *style*, il quale fa parte degli attributi *common* da HTML, cioè un attributo applicabile a tutti gli elementi. La dichiarazione avviene a livello dei singoli tag contenuti nella pagina e per questo si parla di fogli di stile in linea.

```
<span style="vertical-align:super;font-size:9px;">…</span>
```

Dal codice si nota che all'interno dell'attributo *style* è possibile dichiarare più regole di stile, basta separarle dal punto e virgola. I due punti si usano invece per introdurre il valore della proprietà da impostare.

Le tre modalità danno gli stessi risultati perciò la scelta tra quale utilizzare dipende esclusivamente dall'utente il quale deve valutare per ogni situazione qual è la soluzione migliore.

2.2.3 *Struttura del foglio di stile*

All’interno di un file CSS tutto quello che si trova fa parte di due categorie:

1. regole: sono il corpo vero e proprio del CSS;
2. commenti: delimitati dai caratteri “/*”(apertura) e “*/”(chiusura), servono per spiegare cosa indicano le varie parti del codice.

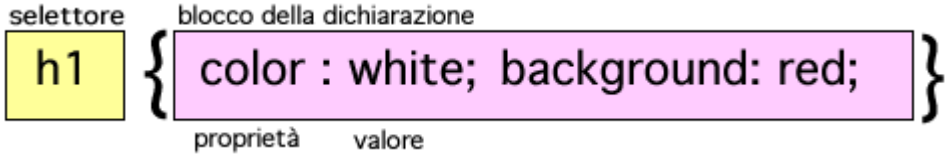

Figura 1: Struttura delle regole dei CSS

Come si vede dall’immagine ogni regola è suddivisa in due parti:

- il selettore;
- il blocco delle dichiarazioni.

Il selettore sta a indicare la parte di documento a cui applicare la regola. In questo caso la regola viene applicata al tag *<h1>* (ci sono diversi tipi di selettore, che qui però non verranno trattati).

Il blocco delle dichiarazioni è delimitato dalle parentesi graffe e al suo interno possono esserci più dichiarazioni.

Ogni dichiarazione è composta dalla coppia:

- proprietà;
- valore.

La proprietà definisce un aspetto dell'elemento da modificare (margini, colore di sfondo, etc) secondo il valore espresso. Proprietà e valore devono essere separati dai due punti.

Per ogni dichiarazione non è possibile indicare più di una proprietà, mentre in molti casi per ogni proprietà è possibile specificare più valori.

Quando vengono definite più dichiarazioni queste devono essere separate dal punto e virgola.

2.3 **PHP**

Conosciuto come PHP: Hypertext Preprocessor, è un linguaggio completo di scripting, con licenza open source e libera, che può girare praticamente su qualsiasi web server, è compatibile con la maggior parte dei sistemi operativi (Windows, Unix/Linux, Mac ecc.) e consente di interagire praticamente con qualsiasi tipo di database (SQLite, MySQL, PostgreSQL, SQL Server, Oracle, SyBase, Access e altri)[3].

Originariamente era stato concepito per la realizzazione di pagine web dinamiche in quanto in quel periodo il web era formato per la maggioranza da pagine statiche, cioè da documenti HTML che non potevano mutare finché qualcuno non interveniva manualmente a modificarle.

2.3.1 *Cenni storici*

Il PHP nasce nel 1994 ad opera del danese Rasmus Lerdorf, come una serie di macro la cui funzione era quella di facilitare ai programmatori l'amministrazione delle homepage personali: da qui trae origine il suo nome, che allora significava appunto Personal Home Page[7].

Essendo open source ben presto si formò una ricca comunità di sviluppatori, dalla quale nacque PHP 3, la versione del linguaggio che diede il via alla crescita esponenziale della sua popolarità.

Divenne molto popolare soprattutto per l’integrazione con il web server Apache che era il più diffuso allora e l’integrazione con mySQL che era uno dei DBMS più utilizzati[3].

Il linguaggio è continuato ad evolvere fino alla versione PHP 5, che è quella che viene utilizzata oggi. Ora quasi tutti i web server lo supportano e al suo interno sono stati integrati molti DBMS così da renderlo il più versatile possibile.

2.3.2 *Cos’è un linguaggio di scripting*

In informatica un linguaggio di scripting è un linguaggio di programmazione interpretato destinato in genere a compiti di automazione del sistema (batch) o delle applicazioni (macro), o ad essere usato all'interno delle pagine web[8].

I programmi sviluppati con questi linguaggi vengono detti script, essi si distinguono dai programmi con cui interagiscono che solitamente sono implementati in un linguaggio differente non interpretato. Inoltre, spesso gli script sono creati o modificati dall'utente finale.

2.3.3 *Caratteristiche di PHP*

Ogni documento scritto in PHP, per essere riconosciuto come tale, deve essere contenuto in un file avente estensione *.php*

PHP è un linguaggio che lavora principalmente a lato server. Questo significa che la pagina presente nel server sarà diversa da quella restituita al client. In poche parole quando viene richiesta una pagina il server va a guardare se all'interno della pagina c'è del codice PHP. Se c'è lo interpreta/esegue e restituisce al client una pagina contenente solo HTML, composta dalla parte fissa più il risultato dell'esecuzione del codice PHP.

La parte di codice PHP è delimitata dai tag di apertura:

```
<?php
```

e di chiusura:

```
?>
```

Essendo in grado di interagire con molti DBMS al suo interno è possibile trovare molte funzioni specifiche per ognuno di essi.

2.4 **World Wide Web Consortium**

Associazione fondata nel 1994 al MIT (Massachusetts Institute of Technology), in collaborazione con il CERN (European Organization for Nuclear Research) con lo scopo di migliorare gli esistenti protocolli e linguaggi per il World Wide Web e di aiutare il web a sviluppare tutte le sue potenzialità.

Questa associazione comprende circa 350 membri, tra di essi si trovano tra le più grandi aziende informatiche, compagnie telefoniche, istituzioni no-profit e istituti di ricerca.

Il w3c si occupa di aggiornare e sviluppare le specifiche necessarie ad ottenere un comune linguaggio di comunicazione per internet, così da poter renderne possibile l'accesso a diverse apparecchiature, in più essendo il web libero (chiunque può creare una pagina HTML e metterla online) il w3c ha il compito di fare in modo che non ci siano interessi che possano limitarne questa caratteristica.

Infine il w3c si occupa anche di rendere il più agevole possibile l'accesso al web da parte di portatori di handicap.

2.5 **Database e PostgreSQL**

In informatica, il termine database (o base di dati) indica un archivio strutturato in modo tale da consentire l'accesso e la gestione dei dati stessi (l'inserimento, la ricerca, la cancellazione ed il loro aggiornamento) da parte di particolari applicazioni software ad essi dedicate. Il database è un insieme di informazioni, di dati che vengono suddivisi per argomenti in ordine logico (tabelle) i quali vengono suddivisi a loro volta per categorie (campi).

Una base di dati deve contenere, oltre ai dati veri e propri, le informazioni sulle loro rappresentazioni e sulle relazioni che li legano.

La progettazione di un database non è mai semplice, bisogna capire come strutturarlo per fare in modo che sia veloce e semplice recuperare i dati al suo interno.

Molto importante evitare di duplicare i dati, perché questi vanno a rallentare le ricerche al suo interno.

Le basi di dati possono avere varie strutture, tipicamente:

1. *gerarchica,* rappresentabile tramite un albero;
2. *reticolare,* rappresentabile tramite un grafo;
3. *relazionale,* attualmente il più diffuso, rappresentabile mediante tabelle e relazioni tra esse;
4. *ad oggetti,* estensione alle basi di dati del paradigma "Object Oriented", tipico della programmazione a oggetti;
5. *semantica*, rappresentabile con un grafo relazionale.

Alcune di queste strutture possono essere combinate tra loro, ad esempio PostgreSQL è un database relazionale ad oggetti.

In un sistema informatico, una base di dati può essere manipolata direttamente dai programmi applicativi, interfacciandosi direttamente con il sistema operativo (file system). Tale strategia era quella adottata universalmente fino agli anni sessanta, ed è tuttora impiegata quando i dati hanno una struttura molto semplice, o quando sono elaborati da un solo programma[7].

A partire dalla fine degli anni Sessanta, tuttavia, per gestire basi di dati complesse condivise da più applicazioni si sono utilizzati appositi sistemi software, detti sistemi per la gestione di basi di dati (in inglese "Database Management System" o "DBMS"). Questi sistemi in poche parole consentono la creazione e la manipolazione (gestione) efficiente dei dati di un database.

Uno dei vantaggi è la possibilità di non agire direttamente sui dati, ma di vederne una rappresentazione concettuale.

Qui di seguito vengono descritti il modello relazionale (su cui si basa PostgreSQL) e PostgreSQL, DBMS utilizzato nel progetto di cui è argomento questa tesi.

2.5.1 *Il modello relazionale*

Il modello relazionale è un modello logico, sviluppato da Edgar F. Codd nel 1970, di rappresentazione dei dati implementato su sistemi di gestione di basi di dati (DBMS). Esso si basa sull'algebra relazionale e sulla teoria degli insiemi ed è strutturato attorno al concetto di relazione (tabella).

Questo modello ha come assunto fondamentale il fatto che ogni dato è rappresentato come una relazione ed essi sono manipolati con gli operatori dell’algebra relazionale[5].

Il modello relazionale consente di creare una rappresentazione consistente e logica dell'informazione. La consistenza viene ottenuta inserendo nel progetto del database appropriati vincoli, normalmente chiamati schema logico.

Durante la progettazione del database vengono sviluppati diversi schemi logici. Tramite il processo di normalizzazione viene selezionato lo schema maggiormente "desiderabile".

La struttura base del modello relazionale è il dominio o tipo di dato, definito come l'insieme dei valori che può assumere un determinato attributo. Una tupla è un insieme non ordinato di valori degli attributi. Un attributo è una coppia ordinata di "nome di attributo" e "nome di tipo", mentre un valore di attributo è un valore specifico valido per quel tipo di dato.

Una relazione consiste di una testata e di un corpo, dove la testata è un insieme di attributi e il corpo è un insieme di n tuple. La testata di una relazione è anche la testata di ciascuna delle sue tuple.

La tabella è la rappresentazione grafica normalmente accettata per rappresentare la relazione.

Il principio base del modello relazionale è che tutte le informazioni siano rappresentate da valori inseriti in relazioni (tabelle); dunque un database relazionale è un insieme di relazioni contenenti valori e il risultato di qualunque interrogazione (o manipolazione) dei dati può essere rappresentato anch'esso da relazioni (tabelle).

Questo modello risponde al requisito dell'indipendenza dei dati e prevede una distinzione tra il livello fisico e il livello logico.

2.5.2 *PostgreSQL*

Come detto in precedenza PostgreSQL è un database relazionale ad oggetti (ORDBMS) con licenza libera[9].

Per comprenderne la struttura è necessario capire cosa sia un database relazionale (RDBMS).

Letteralmente Relational database management system (RDBMS), indica un database management system basato sul modello relazionale.

Il tipo di dato come viene usato nei database relazionali può essere un insieme di numeri interi, un insieme di caratteri alfanumerici, l'insieme delle date, i valori booleani vero e falso, e così via. I corrispondenti "nomi di tipo" saranno le stringhe "int", "char", "date", "boolean", etc.

La teoria relazionale non definisce quali tipi devono essere supportati.

Molti sistemi garantiscono la possibilità di definire tipi di dati definiti dall'utente, in aggiunta a quelli "standard" forniti dal sistema.

Negli RDBMS con il termine attributo si va a indicare ciò che normalmente si definisce come colonna. Allo stesso modo tabella è normalmente usato al posto del termine teorico relazione. La struttura di una tabella è specificata come una lista di colonne, ciascuna delle quali ha un nome univoco e un dominio, un insieme cioè di valori accettati. Un valore di attributo è il valore di una cella identificata da una specifica coppia riga - colonna.

Una tupla è praticamente la stessa cosa di una riga.

Una relazione è la definizione di una tabella (cioè un insieme di colonne) insieme ai dati che vi compaiono. La definizione della tabella è la testata, i dati che vi appaiono sono il corpo (un insieme di righe).

Come suggerisce il nome, PostgreSQL usa il linguaggio SQL per eseguire delle query sui dati. Questi sono conservati come una serie di tabelle con chiavi esterne che servono a collegare i dati correlati.

SQL (Structured Query Language) nasce nel 1974 ad opera di Donald Chamberlin ed è un linguaggio di programmazione per database progettato per leggere, modificare e gestire dati memorizzati in un sistema basato sul modello relazionale, per creare e modificare schemi di database, per creare e gestire strumenti di controllo ed accesso ai dati.

SQL è un linguaggio per interrogare e gestire basi di dati mediante l'utilizzo di costrutti di programmazione denominati query.

Come detto in precedenza PostgreSQL è un database relazionale ad oggetti, si pone quindi il problema di convertire le informazioni dal mondo SQL a quello della programmazione orientata agli oggetti. Questo problema presenta difficoltà dovute principalmente al fatto che i due mondi utilizzano modelli di organizzazione dei dati molto differenti. Esistono un certo numero di soluzioni di mappatura, normalmente dette "object-relational mapping", che possono risolvere il problema, ma tendono ad essere costose e ad avere i loro problemi.

PostgreSQL può risolvere molti di questi problemi direttamente nel database. Esso permette agli utenti di definire nuovi tipi basati sui normali tipi di dato SQL, permettendo al database stesso di comprendere dati complessi[5].

PostgreSQL, inoltre, permette l'ereditarietà dei tipi, uno dei principali concetti della programmazione orientata agli oggetti[6].

La programmazione del database stesso può ottenere grandi vantaggi dall'uso delle funzioni. La maggior parte dei sistemi SQL permette agli utenti di scrivere una procedura, un blocco di codice SQL che le altre istruzioni SQL possono richiamare.

All’interno di PostgreSQL si trova un linguaggio nativo chiamato PL/pgSQL che offre particolari vantaggi nelle procedure che fanno un intensivo uso di query[6].

Il programmatore può inserire il codice sul server come funzioni in modo che il codice SQL possa richiamare funzioni scritte in altri linguaggi.

Capitolo 3

Lavoro Svolto

3.1 Login e mantenimento della connessione

scores.php è la prima pagina di codice ed è anche la struttura della prima pagina del sito.

```
<form action="ricerca.php" method="post">
      <p>Per effettuare la ricerca di documenti IEEE 1599-2008 fare la
      login.<br>Se non conosci i dati di accesso vai a richiederli al
      Laboratorio di Informatica Musicale!</p>
      <table border="0">
            <tr><td colspan="2"><h1>Login</h1></td></tr>
            <tr><td>Username:</td><td>
            <input type="text" name="user" maxlength="40">
            </td></tr>
            <tr><td>Password:</td><td>
            <input type="password" name="password" maxlength="50">
            </td></tr>
            <tr><td colspan="2" align="right">
            <input type="submit" name="submit" value="Login">
            </td></tr>
      </table>
</form>
```

All'interno troviamo il tag *form* HTML. Esso serve per inserire e inviare dati ad un'altra pagina.

In questo caso viene utilizzato per inserire il nome utente e la password da inviare alla pagina *ricerca.php*.

Come si vede dal codice nel tag *form* sono stati specificati due parametri: *action* e *method*.

```
<form action="ricerca.php" method="post">
```

Action è l'unico attributo obbligatorio del tag *form*, all'interno di esso viene specificato l'indirizzo della pagina a cui inviare i dati.

All'interno di *method* invece viene specificato quale metodo utilizzare per inviare i dati.

I metodi principali sono due: *GET* e *POST*.

Il metodo GET va ad accodare i dati da inviare all'indirizzo della pagina richiesta, facendo seguire il nome della pagina da un punto interrogativo e dalle coppie nome/valore dei dati che interessano. La stringa che segue il punto interrogativo viene chiamata query string[8].

I valori contenuti nella query string vengono memorizzati da PHP nell'array $_GET che è un array superglobale. Per poterli richiamare basta scrivere *$_GET[nome]* il quale indica il valore all'interno dell'array $_GET di posto "nome".

Con il metodo POST, invece, l'invio dei dati avviene in due fasi distinte: prima viene contattata la pagina sul server che deve processare i dati, poi vengono inviati i dati stessi. Per questo motivo i parametri non compaiono nella query string. Con questo metodo non ci sono limiti sulla lunghezza dei caratteri[8].

Come nel caso del metodo GET i parametri passati vengono memorizzati da PHP in un array superglobale, in questo caso chiamato *$_POST*. Anche qui per poterli richiamare il metodo è semplice, basterà scrivere, ad esempio, *$_POST[nome]*.

Per capire meglio il funzionamento di questi due metodi è necessario capire cos'è un array superglobale.

Di norma quando una variabile viene dichiarata all'interno di una funzione può essere richiamata esclusivamente all'interno della funzione stessa.

PHP per ovviare a questo problema ha creato al suo interno delle variabili dette globali, che possono essere cioè richiamate da qualsiasi funzione[3]. Un array superglobale è quindi una variabile di tipo array che può essere richiamata da parte di diverse funzioni.

Nel caso del progetto in questione è stato scelto di usare il metodo POST.

La scelta è stata fatta non tanto per l'assenza di limiti di lunghezza dei dati da inviare (dovendo passare solo il nome utente e la password) quanto per una questione di sicurezza.

Siccome in questo caso i dati da inviare sono dati d'accesso riservati, se venissero inviati con il metodo GET rimarrebbero visibili sulla URL della pagina. In questo modo se una persona qualsiasi, diversa da quella che ha eseguito l'accesso, si trovasse davanti a una delle pagine successive alla login vedrebbe i dati di accesso.

La connessione al database avviene attraverso l'utilizzo del comando *pg_connect*, questo comando è stato utilizzato nella pagina *pgconnect.php*.

```php
<?php
function connetti(){
	if ($_COOKIE["user"]&&$_COOKIE["password"]!=null){
		$user = $_COOKIE["user"];
		$pass = $_COOKIE["password"];
	}
	else{
		$user = $_POST["user"];
		$pass = $_POST["password"];
		setcookie("user",$user);
		setcookie("password",$pass);
	}
	$string = "dbname=1599_Repository user=" .$user. " password="
	.$pass;
	$connessione = @pg_connect($string);
	if(!$connessione) {
		setcookie("user","");
		setcookie("password","");
		die('Connessione fallita !<br /> Torna alla <a
	href="scores.php"> Login </a>');
	}
	return $connessione;
}
?>
```

In questa pagina viene creata la funzione “connetti” il cui nodo centrale è la funzione *pg_connect.*

Questa funzione è una funzione specifica di PHP per la connessione a database sviluppati nell’ambiente PostgreSQL.

Il suo funzionamento è semplice, basta specificare il nome del database a cui connettersi, il nome utente e la password. Se essi sono corretti la connessione sarà aperta.

In questa pagina viene anche gestito il problema del mantenimento della connessione, in quanto passando da una pagina ad un'altra la connessione con il database non resta aperta.

Qui per farlo è stato scelto di utilizzare i COOKIE, frammenti di testo che vengono inviati dal client al server e viceversa ogni volta che si mettono in comunicazione tra loro[8].

Ad essi è possibile assegnare una durata temporale, cosa che in questo caso non è stata specificata. Così facendo i COOKIE vengono cancellati nel momento in cui viene chiuso il browser.

Per creare un COOKIE viene utilizzato il comando *setcookie.*

```php
setcookie("user",$user);
```

Come si nota dal codice, all’interno del comando viene specificato il nome del COOKIE e il suo contenuto.

PHP, come nel caso dei metodi GET e POST, va a memorizzare i cookie all'interno dell'array superglobale *$_COOKIE*. Anche qui per poter richiamare il contenuto del cookie basta scrivere *$_COOKIE[nome]*.

Ora analizziamo in dettaglio come è strutturata la funzione "connetti" creata, come detto sopra, nella pagina *pgconnect.php.*

La funzione comincia con un costrutto IF che controlla l'esistenza delle variabili *$_COOKIE[user]* e *$_COOKIE[password]*. Nel caso che esistano va ad assegnare alla variabile temporanea *$user* il valore contenuto nella variabile *$_COOKIE[user]* e alla variabile temporanea *$pass* il valore contenuto nella variabile *$_COOKIE[password]*.

Se invece non esistono va ad assegnare a *$user* il valore contenuto nella varibile *$_POST[user]* e a *$pass* il valore contenuto nella variabile *$_POST[password]*, le quali contengono i dati passati dalla form.

```
if ($_COOKIE["user"]&&$_COOKIE["password"]!=null){
      $user = $_COOKIE["user"];
      $pass = $_COOKIE["password"];
}
else{
      $user = $_POST["user"];
      $pass = $_POST["password"];
      setcookie("user",$user);
      setcookie("password",$pass);
}
```

Subito dopo viene costruita la stringa assegnata alla variabile *$string* da "passare" alla funzione *pg_connect* che, come detto in precedenza, dovrà contenere il nome del database a cui connettersi, il nome utente e la password.

In questo caso la stringa ha il nome del database (*dbname*) fisso, visto che questo sistema di ricerca deve riferirsi sempre allo stesso database, mentre al posto dei valori dei campi *user* e *password* ci sono legate le due variabili *$user* e *$password*.

```
$string = "dbname=1599_Repository user=" .$user. " password=" .$pass;
```

Importante notare che in PHP per poter inserire all'interno di una stringa i valori contenuti all'interno delle variabili è necessario usare l'operatore di concatenamento ".". Questo perché se vengono messe le due variabili all'interno della stringa (che è contenuta tra gli apici) l'interprete PHP non le riconosce, quindi prende i due valori come semplice testo senza guardare all'interno delle variabili.

Per fare un esempio se viene scritto così:

```
$string = "dbname=1599_Repository user=$user password=$pass”;
```

Il risultato che si ottiene facendo stampare con una *echo* al browser il contenuto di *$string* è:

```
dbname=1599_Repository user=$user password=$pass
```

A questo punto viene assegnato alla variabile *$connessione* la funzione *pg_connect* con la stringa completa.

```
$connessione = @pg_connect($string);
```

Subito dopo viene messo un costrutto IF che controlla il buon esito della connessione; se la connessione non è riuscita viene lanciata la funzione di PHP *die()* - alias della funzione *exit()* -.

Questa funzione va a stampare il messaggio contenuto al suo interno e chiude lo script. In questo caso il messaggio che viene stampato è un messaggio di errore che avvisa della mancata connessione al database. Inoltre viene stampato un link che permette di tornare alla pagina *scores.php*.

Il link viene creato utilizzando il tag html *<a>*. In questo tag è molto importante l’attributo *href* : qui viene specificata la pagina di destinazione a cui si viene rimandati quando si va a cliccare il link associato.

Nel caso di connessione fallita ai due cookie *$_COOKIE[user]* e *$_COOKIE[password]* viene assegnato il valore “vuoto”.

Questa operazione viene fatta perché nel caso in cui un utente sbagli i dati di accesso questi vengono comunque assegnati ai COOKIE. Guardando il codice di controllo di connessione è facile notare che in questa situazione è impossibile connettersi al database perché anche reinserendo i dati il controllo viene fatto comunque tenendo conto del contenuto dei COOKIE.

In caso di connessione riuscita viene lanciata la funzione *return*. Essa se viene chiamata tramite una funzione va a chiuderne l’esecuzione e restituisce il suo argomento come valore della funzione chiamante.

Questa funzione va a interrompere lo script corrente. Se è stata inserita tramite *include* all’interno di un file, una volta interrotto lo script, ripassa il controllo al file che l’ha richiamata.

Da notare che davanti al nome della funzione *pg_connect* è stato inserito il carattere "@", questo fa sì che PHP non restituisca alcun errore in caso di problemi in modo da poter personalizzare la gestione degli errori.

Figura 2: Pagina di login

3.2 **Ricerca**

Come già detto i dati vengono passati alla pagina *ricerca.php*, pagina già esistente del sito www.mx.dico.unimi.it, nella quale sono state apportate delle modifiche.

La prima modifica è subito in testa al codice, infatti vi troviamo la funzione *include.*

Questa funzione permette di includere altre pagine di codice.

In questo caso viene utilizzata per includere la pagina *pgconnect.php,* per i motivi già visti in precedenza.

```php
<?php
	include("./pgconnect.php");
	$db=connetti();
	if(!$db) {
		die('Connessione fallita !<br />');
	}
	else {
?>
```

Al centro della pagina si trova la form di ricerca.

Figura 3: Pagina di ricerca

Al suo interno vi troviamo tutti i campi necessari a rendere la ricerca più specifica possibile.

Per comodità i campi di ricerca vengono suddivisi in tre aree che a loro volta si riferiscono ai corrispettivi tre layer dello standard IEEE 1599: general, audio e notational.

Premendo il pulsante *avvia la ricerca* si viene mandati alla pagina *ris_ricerca.php*, ovviamente vengono anche inviati ad essa i dati inseriti nei campi di ricerca.

Anche in questo caso il metodo utilizzato per inviarli è *POST*.

ris_ricerca.php è la pagina che si occupa di stampare a video i risultati, anche qui in testa alla pagina troviamo l'inclusione di *pgconnect.php*.

All'interno del body della pagina si trova il codice PHP che si occupa di prendere i dati passati dalla pagina *ricerca.php* e di utilizzarli per costruire la query di ricerca.

Per fare le query viene utilizzata la funzione *pg_query*.

Questa funzione è specifica di PHP e, come nel caso della funzione *pg_connect*, è costruita per eseguire delle query all'interno di ambienti sviluppati in PostgreSQL. La query da eseguire, ovviamente, è quella che gli viene assegnata. Per poter utilizzare la funzione deve essere aperta la connessione con il database.

La query è in forma di stringa e in quanto tale si possono usare degli accorgimenti per rendere la sua creazione dinamica.

Per prima cosa vengono assegnati tutti i parametri passati dalla pagina precedente a delle variabili temporanee.

```
$author = $_POST["author"];
```

Subito dopo viene creata la stringa della query generale di ricerca.

In questa query il contenuto della *SELECT* e quello della *FROM* restano fissi, mentre il contenuto della *WHERE* è dinamico.

Per quanto riguarda la *SELECT* al suo interno vengono specificati i campi che devono essere stampati all'interno della pagina dei risultati.

```
SELECT \"TFile_MX\".\"ID_file\",\"TFile_MX\".main_title,
      \"TAuthor\".author, \"TWork\".work_title
```

Come si nota dal codice in realtà viene selezionato anche il campo *ID_File* che non deve essere stampato, ma è necessario in quanto deve essere passato alle pagine dei dettagli.

Nella *FROM* invece vengono specificate le tabelle all'interno delle quali viene fatta la ricerca e le relazioni che le legano.

```
FROM \"TFile_MX\" natural left join (\"TAuthor_Type\" natural join
      \"TAuthor\") natural left join (\"TFile_Work\" natural join
      \"TWork\")  natural left join \"TDate\" natural left join
      \"TMIDI_Instance\" natural left join (\"TGraphic_Instance\"
      natural join \"TGraphic_Instance_Group\")
      natural left join (\"TGenre_File\" natural join \"TGenre\")
      natural left join (\"TTrack\" natural left join (\"TAlbum_Track\"
      natural join \"TAlbum\") natural left join (\"TLabels_Tracks\"
      natural join \"TLabels\") natural left join
      (\"TPerformers_Tracks\" natural join \"TPerformers\"))
```

Guardando il codice si nota l'utilizzo dell'operazione di *join*, in particolare *natural join* e *natural left join*.

L'operazione di *join* permette di correlare dati memorizzati in relazioni diverse, quindi di "attraversare" le relazioni.

Tradizionalmente il *join* è espresso in SQL tramite un prodotto Cartesiano a cui sono applicati uno o più *predicati di join*[5].

SQL prevede diverse operazioni di *join* esplicite, tra le quali si trova il *natural join*.

In questa operazione di *join* viene richiesta l'uguaglianza dei valori delle colonne con lo stesso nome.

Se ad esempio si vogliono cercare tutti i brani all'interno del database, selezionando anche l'autore di ogni brano bisognerà scrivere:

```
SELECT \"TFile_MX\".main_title, \"TAuthor\".author FROM \"TFile_MX\"
      natural join (\"TAuthor_Type\" natural join \"TAuthor\")
```

Questa *SELECT* come risultato va a restituire tutti i brani composti da ogni autore. Se però all'interno del database è presente un brano senza autore, questo brano non viene restituito.

Per capire meglio: in R NATURAL JOIN S non si ha traccia delle tuple di R che non corrispondono ad alcuna tupla di S.

Per poter fare ciò SQL mette a disposizione l'operazione *outer join*.

Questo operatore aggiunge al risultato le tuple di R e di S che non hanno partecipato al *join*.

Esistono tre varianti dell'*outer join:*

- *full*: sia le tuple di R che di S che non partecipano al *join* vengono inserite nel risultato;
- *left*: le tuple di R che non partecipano al *join* vengono inserite nel risultato;
- *right*: le tuple di S che non partecipano al *join* vengono inserite nel risultato.

Avendo capito il funzionamento delle *join* è semplice capire come è stata costruita la clausola *FROM* indicata sopra.

La clausola *WHERE*, come già detto in precedenza, viene generata dinamicamente ma ha comunque una prima parte fissa:

```
WHERE true
```

Per poter generare la seconda parte in maniera dinamica è stato utilizzato un metodo semplice: sono state costruite delle *IF*, una per ogni campo di ricerca, le quali vanno a controllare campo per campo se è stato inserito il corrispettivo parametro; se non è stato inserito la query non viene modificata, se invece è stato inserito viene aggiunta alla stringa della query una parte di codice specifica per ogni singolo campo.

Ad esempio se viene inserito un parametro all'interno del campo autore viene aggiunta la seguente parte di codice:

```
if ($author != "") {
      $query=$query."AND lower (author) like lower('%".$author."%') ";
}
```

La parte di codice è praticamente identica per ogni campo di ricerca eccezion fatta per le ricerche relative al titolo dell'opera e al genere, dove risulta più specifico:

```
if ($title != "") {
      $query=$query."AND lower (main_title) like lower ('%".$title."%')
      OR lower (other_title) like lower ('%".$title."%') ";
}

if ($genre != "") {
      $query=$query."AND lower (genre) like lower('%".$genre."%')
      OR lower (description) like lower('%".$genre."%') ";
}
```

Nel primo caso viene controllato se il titolo che si sta cercando si trova nel campo *main_title* oppure nel campo *other_title*. Questo perché non sempre l'utente può sapere se un determinato titolo di un'opera è quello principale.

Nel secondo caso viene controllato se il genere di un'opera si trova all'interno del campo *genre* o nel campo *description.*

Il campo *description* serve a contenere i generi secondari di un'opera, perché il campo genere del database può contenere un solo dato e ovviamente sarà quello ritenuto principale da parte del creatore del file.

Come si nota dal codice per collegare le varie parti viene inserito l'operatore di concatenamento ".".

Una volta controllati tutti i campi la query viene chiusa tramite l'aggiunta del punto e virgola:

```
$query=$query.";";
```

Guardando il codice si nota la presenza della funzione *lower(),* per poterne capire il funzionamento bisogna chiarire che le ricerche all'interno dei database di norma sono CASE SENSITIVE.

Questo termine letteralmente sta a indicare che l'applicazione fa distinzione tra le lettere maiuscole e quelle minuscole, quindi se si esegue una ricerca si otterranno risultati differenti a seconda di come verrà scritta la parola chiave di ricerca.

In un'applicazione CASE SENSITIVE, una ricerca per "BACH" dà risultati diversi da "bach".

Per ovviare a questo problema è stato deciso di forzare a caratteri minuscoli sia la parola di ricerca sia quella con cui viene confrontata all'interno del database. In questo modo la ricerca diventa CASE INSENSITIVE.

Questa forzatura avviene, a livello di query, tramite l'utilizzo della funzione *LOWER().* Essa rende minuscolo quello che viene inserito tra le due parentesi. Come detto in precedenza viene applicata sia alla parola inserita sia alla parola di confronto.

Un altro problema che sorge a livello di ricerca è quello dovuto al formato dei parametri inseriti nel database. Se si devono andare a cercare le opere di un dato autore non si può sapere a priori in che formato è stato inserito il nome dell'autore all'interno del database. Per questo bisogna fare in modo che se un utente va ad inserire anche solo una parte del nome di un autore deve poter avere dei riscontri.

Per fare ciò viene utilizzato il carattere “%” all’interno della query di ricerca.

Questo carattere se viene inserito prima della parola passata va a cercare tutte quelle parole che finiscono con essa, se viene inserito dopo va a cercare tutte quelle che iniziano con essa, infine se viene messo sia prima che dopo va a cercare tutte quelle che la contengono.

In questo caso viene messo in tutte le ricerche sia prima che dopo la parola passata.

Guardando la query si nota l’utilizzo del carattere “\” prima degli apici, questo carattere è quello che viene detto “carattere di escape”.

Per capire il motivo del suo utilizzo bisogna capire il significato degli apici in PHP.

In PHP gli apici indicano l’inizio e la fine di una stringa. Se però al suo interno è necessario inserirne altri, come nel caso delle query, bisogna trovare un metodo per dire a PHP che quelli non stanno a indicare l’apertura o la chiusura della stringa.

Il metodo per farlo è proprio quello del “carattere di escape”, il quale dice a PHP che il carattere successivo non deve essere interpretato ma è semplice testo.

Una volta che la query è completa è necessario controllarne la correttezza, per farlo viene inserito un costrutto *IF*. Questo costruttore nel caso in cui non viene eseguita la query, va a stampare gli eventuali errori. In questo modo è possibile avere un riscontro per poterli correggere.

Per poter recuperare gli errori dati da una query è necessario usare la funzione *pg_last_error*, la quale restituisce l’ultimo errore registrato sulla connessione.

```
$ris=pg_query($query);
      if  (!$ris) {
            echo "query did not execute";
            echo pg_last_error();
            exit();
      }
```

Per finire troviamo il costrutto *IF* che si occupa di stampare il risultato della query.

```
$num_row=pg_num_rows($ris);
      if ($num_row==0){
            echo "Non esistono file IEEE 1599-2008 che soddisfano questi
            parametri di ricerca";
      }
      else{
       $i = "1";
       echo "<table>";
            while ($row = pg_fetch_array($ris)) {
                  echo "<tr>";
                  echo "<td class=\"menu_chiaro\">";
                  echo "<table>";
                  echo "<tr><td>".$i.":</td><td>".$row[author]."</td>
                        <td style=\"font-size:22px;\">".$row[main_title]."
                        </td><td>".$row[work_title]."</td></tr>";
                  echo "<tr><td colspan=\"4\">
                        <a href=\"dettagli.php?id_file=".$row[ID_file]."\"
                         target=\"_blank\"> Dettagli </a>";
                  echo " | <a href=\"midi.php?id_file=".$row[ID_file]."\"
                        target=\"_blank\"> MIDI </a>";
                  echo " | <ahref=\"graphic.php?id_file=".$row[ID_file].
                        "\" target=\"_blank\"> Graphic </a>";
                  echo " | <a href=\"audio.php?id_file=".$row[ID_file]."\"
                        target=\"_blank\"> Audio </a>";
                  echo " | <a href=\"xml.php?id_file=".$row[ID_file]."\"
                        target=\"_blank\"> File XML </a></td></tr>";
                  echo "</table>";
                  echo "</td>";
                  echo "</tr>";
                  $i=$i+1;
            }
            echo "</table>";
      }
```

Questo ciclo innanzitutto controlla che la query dia risultati. Per farlo va a guardare se il numero delle righe restituite dal comando *pg_query* è uguale a 0. Se lo è viene stampato un messaggio che indica l'assenza di risultati. Se invece è diverso da 0 viene creata una tabella contenente tutti i risultati.

La funzione che permette di contare il numero delle righe restituite dalla funzione *pg_query* è la funzione *pg_num_rows*.

Per compilare le righe della tabella viene utilizzata la funzione *pg_fetch_array*. Questa funzione restituisce l'array numerico e l'array associativo che corrispondono alla singola tupla. Inserendo questo comando all'interno di un ciclo *WHILE* si riescono ad ottenere tutte le singole righe che, in questo caso, vengono messe in una tabella HTML.

Dal punto di vista estetico è stato deciso di fare in modo che questa pagina di risultati ricordi la form di ricerca.

Per poterlo fare si è inserito un header XHTML che permette di utilizzare gli stili contenuti nel foglio CSS specificato.

```
<link rel="stylesheet" href="stile.css" type="text/css">
```

Una volta inserito questo header nella pagina è possibile richiamare gli stili presenti all'interno del file *css.txt*. Per farlo basta inserire nell'attributo *style* il nome dello stile voluto. Ovviamente l'attributo *style* è specifico per quei tag che permettono la stampa a video dei risultati.

I risultati vengono stampati nel formato: *n°. author main_title work.* Per scelte esclusivamente di gusto il *main_title* viene stampato con una dimensione maggiore rispetto agli altri dati, per farlo è bastato indicare nell'attributo *style* della colonna in questione un *font-size* maggiore rispetto a quello di default.

Al di sotto di ogni riga della tabella vengono anche messi cinque link relativi al risultato trovato che danno informazioni aggiuntive a riguardo, utili per capire meglio se il risultato trovato è quello che si stava effettivamente cercando.

Figura 4: esempio di stampa dei risultati

3.3 **Recuperare dettagli specifici riguardanti i risultati trovati**

Nella pagina *ris_ricerca.php*, per ogni risultato trovato, vengono associati cinque diversi link:

- Dettagli;
- MIDI;
- Graphic;
- Audio;
- File XML.

Questi cinque link rimandano a cinque diverse pagine di codice:

- *dettagli.php;*
- *MIDI.php;*
- *graphic.php;*
- *audio.php;*
- *xml.php.*

Per poter recuperare le informazioni relative al dato selezionato tramite la ricerca è necessario passare un'informazione univoca alla pagina di destinazione, in questo caso l'informazione è rappresentata dall'ID del file.

Il metodo utilizzato per inviare questo dato è *GET*.

Le quattro pagine *dettagli.php, MIDI.php, graphic.php e audio.php* hanno una struttura simile tra loro che si differenzia solamente per la costruzione della query e la stampa dei risultati.

xml.php invece si differenzia ulteriormente in quanto si occupa di far salvare il file XML associato. Per questo di seguito viene descritta la pagina *dettagli.php* per intero, mentre delle altre vengono spiegate solamente le differenze.

3.3.1 Dettagli

All'inizio di questa pagina, così come nelle altre pagine, si trova il comando *include* che, come visto prima, viene utilizzato per mantenere la connessione.

Subito dopo, per comodità, viene associata ad una variabile temporanea il contenuto dell'array *$_GET[ID_File]*. Questo valore viene utilizzato nella clausola *WHERE* della query, in modo da recuperare i dettagli relativi all'elemento a cui si riferiscono.

```
"SELECT \"TFile_MX\".\"ID_file\", \"TFile_MX\".main_title,
      \"TFile_MX\".other_title, \"TFile_MX\".upload_date,
      \"TFile_MX\".file_xml, \"TGenre\".genre,
      \"TGenre_File\".description, \"TGenre_File\".weight, \"TDate\".date,
      \"TDate\".date_type, \"TFile_MX\".notes
FROM \"TFile_MX\" natural left join (\"TGenre_File\" natural join
      \"TGenre\") natural left join \"TDate\"
WHERE \"TFile_MX\".\"ID_file\" = ".$id.";";
```

Come si nota dal codice la query, oltre a selezionare i campi che devono essere stampati, va a selezionare anche il campo *ID.File*, questo è necessario per poter fare il confronto con l'*ID_File* che è stato inviato dalla pagina precedente.

Anche qui è stato inserito un costrutto *IF* identico a quello presente nella pagina *ris_ricerca.php,* che controlla la correttezza e l'esecuzione della query.

I risultati vengono stampati, tramite un ciclo *WHILE*, all'interno del tag HTML <*table*>. Qui è stato scelto di non stampare i risultati in una tabella vera e propria in quanto il campo *notes*, potendo contenere molte informazioni, non può essere contenuto in una colonna perchè sformerebbe la pagina.

Per questo motivo i dettagli vengono tutti stampati sulla stessa riga, a parte il contenuto di *notes* che viene stampato subito sotto.

In questa pagina e anche nelle altre (esclusa *xml.php)* viene aggiunto nel ciclo di stampa dei risultati il link che si occupa del download del file XML associato.

3.3.2 File XML

Questa pagina di codice ha due compiti: creare un file di formato xml nel quale viene scritto il contenuto del campo *file_xml* della tabella *TFile_MX* e far aprire una finestra che dia la possibilità all'utente di salvare il file appena creato sul proprio computer.

```
"SELECT \"TFile_MX\".file_name, CAST (\"TFile_MX\".file_xml as text)
FROM  \"TFile_MX\"
WHERE \"TFile_MX\".\"ID_file\" = ".$id.";";
```

Qui la query di ricerca va a selezionare il campo *file_name,* in modo da poterlo usare per dare un nome significativo al file che viene creato, e ovviamente il campo *file_xml*.

Per selezionare *file_xml* non basta fare una semplice SELECT ma in essa bisogna fare un CAST del campo in modo da renderlo di tipo text. Questo viene fatto perché il campo, all'interno del database, è di tipo xml e se venisse selezionato come tale postgreSQL lo interpreterebbe andando a eliminare la prima riga del file (che altro non è che la dichiarazione della versione dell'xml). Questa cosa è da evitare in quanto il file salvato risulterebbe incompleto e perderebbe di utilità.

Come detto prima, al contrario delle altre, questa pagina non deve stampare il risultato in una pagina del browser ma all'interno di un file.

Per fare ciò viene inserito questo header:

```
header('Content-disposition: attachment; filename="'.$row[file_name].'"');
```

La prima parte dell'header va a dichiarare che si sta generando un file allegato, la seconda parte specifica il nome che deve avere il suddetto allegato.

L'*echo* successiva scrive all'interno dell'allegato il contenuto del campo *file_xml.*

```
echo $row[file_xml];
```

3.3.3 Audio

Questa pagina di codice ha molti dati da stampare. Per questo è stato deciso di stamparli su più righe, così da non rischiare che l'utente si trovi a dover utilizzare la scroll bar orizzontale del browser per controllare tutti i dati.

3.3.4 Graphic e MIDI

In queste due pagine la stampa dei risultati è impostata nella stessa maniera, è stato scelto di stampare a video una tabella vera e propria mettendo anche le intestazioni delle colonne e i bordi di separazione tra le righe e le colonne.

Per inserirli basta semplicemente indicare nel tag <*table*> l'attributo *border* dichiarando un numero maggiore di 0.

```
echo "<table border=\"1\">";
```

Il numero dopo l'uguale di *border* indica la larghezza delle linee.

Per distinguere le intestazioni delle colonne è stato utilizzato il tag <*b*> che rende in grassetto quello che viene scritto al suo interno.

```
echo "<tr><td> </td><td> <b>Main Title</b></td><td> <b>File format</b>
      </td><td> <b>Encoding format</b></td><td><b>Description</b></td>
      <td> <b>Position</b></td><td><b>Download</b></td></tr>";
```

Conclusioni

Il risultato del progetto ha portato ad ottenere un'interfaccia web funzionante per la consultazione e ricerca dei documenti conservati all'interno del repository IEEE 1599.

Il suo sviluppo è stato portato a termine considerando gli standard W3C odierni, le esigenze degli utenti e basandosi sull'impostazione del già esistente sito ufficiale.

Essendo la ricerca basata su un repository, le sue eventuali modifiche possono portare altrettante modifiche a questa interfaccia, ad esempio se in futuro verranno aggiunte le *notation_istance* sarà necessaria una voce aggiuntiva tra i campi di ricerca.

In ogni caso, essendo una parte del sito www.mx.dico.unimi.it il quale è sotto la gestione del Laboratorio di Informatica Musicale, potrà essere sottoposta a modifiche, dovute magari al cambio d'impostazione del sito stesso.

Come detto prima l'interfaccia è stata sviluppata seguendo gli standard W3C odierni, nel caso in cui questi vengano cambiati il progetto deve essere aggiornato.

Questo progetto essendo così sensibile a diverse possibili modifiche potrà risultare utile come base per altri progetti futuri.

Bibliografia

1. Luca A. Ludovico, “Manuale di MX”, LIM – Laboratorio di Informatica Musicale / DICo – Dipartimento di Informatica e Comunicazione, 9 Febbraio 2005.

2. Luca A. Ludovico, “Key concepts of the IEEE 1599 Standard”, *Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects*, pp. 15-26, IEEE CS, Lugano, Switzerland, 2008.

3. Luca Welling, Laura Thomson, “PHP and MySQL web development” (4th edition), Addison-Wesley Pearson education, 2008

4. Michael Bowers, “Pro CSS and HTML design patterns”, Apress, 2007

5. The PostgreSQL Global Development Group, “The PostgreSQL Reference Manual Volume 1: SQL Language Reference”, Network Theory LTD, 2007

6. The PostgreSQL Global Development Group, “The PostgreSQL Reference Manual Volume 2: Programming Guide”, Network Theory LTD, 2007

Sitografia

7. http://php.net/
8. http://www.html.it/
9. http://www.postgresql.org/

Ringraziamenti

Per i ringraziamenti sarebbe necessario un elaborato a parte, non essendo possibile cercherò di riassumerli in questa pagina.

Desidero ringraziare innanzitutto i miei compagni di corso nonché ”colleghi” Matteo Radice e Luca Trinchinetti con i quali è stato deciso di intraprendere lo sviluppo del database – plugin – interfaccia web per lo standard IEEE 1599. Voglio ringraziare anche il prof. Luca A. Ludovico e il dott. Adriano Baratè per il supporto tecnico sia riguardo allo sviluppo del progetto che alla stesura dell’elaborato.

Un ringraziamento a tutte le persone incontrate in università, compagni di corso e non, grazie ai quali sono riuscito a mantenere la determinazione di finire gli studi.

Uno alla mia famiglia, per avermi sopportato e avermi permesso di intraprendere questo percorso universitario.

Un ringraziamento a tutti gli amici di sempre, dalla mia compagnia ai “soci del sabato e annessi” passando per i compagni di squadra e per i compagni dell’ITIS fino ad arrivare agli amici degli amici. Un ringraziamento particolare però va a Federica Magnano e a Marco Sangiorgio, per il supporto morale e la fiducia nei miei confronti.

Ringrazio anche tutti gli amici conosciuti in questi ultimi anni, in particolare Massimiliano Salina e i “laghesi”, perché le cose belle non si finiscono mai di scoprire.

Per finire un ringraziamento agli amici persi, è anche grazie a loro che sono riuscito ad arrivare fin qui.