UNIVERSITA' DEGLI STUDI DI PADOVA
FACOLTA' DI SCIENZE STATISTICHE

TESI DI LAUREA TRIENNALE
IN STATISTICA E TECNOLOGIE INFORMATICHE

GENERAZIONE AUTOMATICA DI DATI XML SU PALMARI PER VISITATORI DI MUSEI

Relatore: Prof. Massimo Melucci

Laureando: Andrea Visentin
Matricola: 491840STI

Anno Accademico: 2006/2007

Ad Angelica

Indice generale

1 - Introduzione

Questo mio lavoro si colloca nell'ambito di sviluppo del progetto *"Il museo si racconta"* del Dipartimento di Ingegneria Informatica dell'Università di Padova finalizzato alla realizzazione di un sistema informatico per la catalogazione di reperti museali.

A questo progetto aderiscono musei di 4 tipologie diverse: mineralogia, orto botanico, paleontologia, fisica. Ciascuna tipologia di reperto museale ha delle precise normative di catalogazione definite dall'Istituto Centrale per il Catalogo e la Documentazione (ICCD).

L'obiettivo di questo progetto è quello di realizzare una base dati completa e aggiornata dei reperti museali per facilitarne la divulgazione al pubblico in quanto patrimonio culturale.

Dal punto di vista tecnico il cuore del progetto *"Il museo si racconta"* è costituito da:

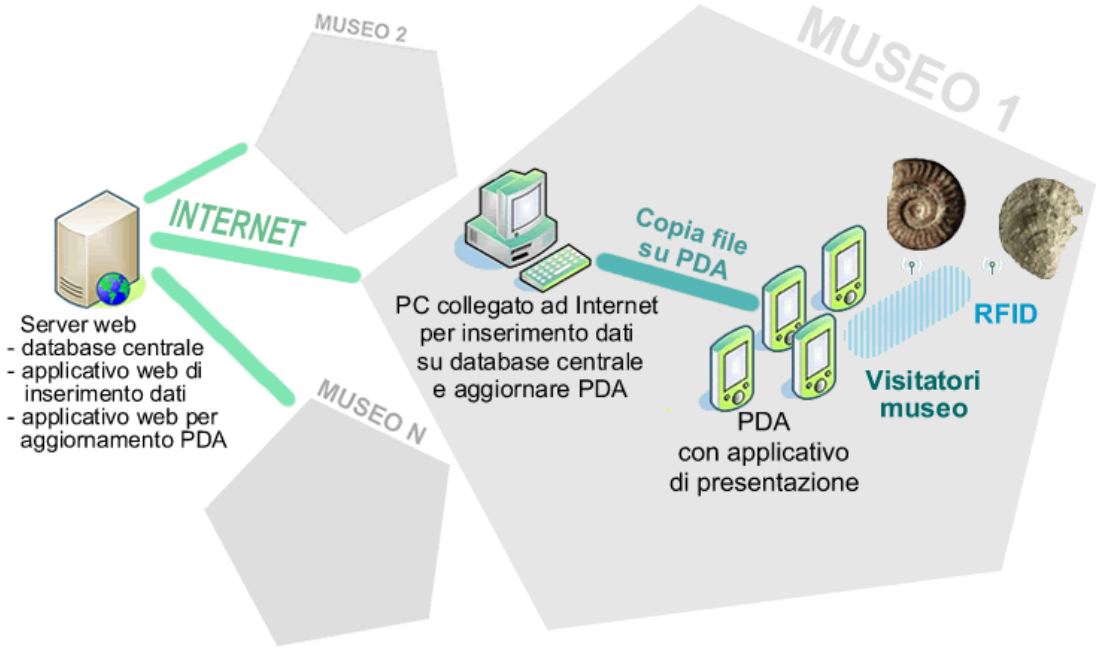
- un database centrale destinato ad accogliere le informazioni dei musei e dei reperti.
  E' un database relazionale implementato in PostgreSQL . La struttura del database è studiata per la catalogazione dei beni museali secondo le specifiche dell'ICCD ; offre il supporto multilingua e la possibilità di memorizzare documentazione multimediale (immagini, audio, video etc.).
- un'applicazione web di di inserimento e modifica dati per la compilazione delle catalogazioni dei beni museali. Tale applicazione permette al personale addetto dei rispettivi musei l'aggiornamento dei dati sul database accedendo semplicemente ad Internet.
- applicazioni di presentazione delle informazioni al pubblico.

Tra queste applicazioni è stato realizzato un sistema multimediale di presentazione dei reperti museali che sarà a disposizione dell'utente durante la visita al museo. Questo sistema è costituito da un Palmare (PDA) quindi un piccolo dispositivo portatile che, avvicinato ad un reperto museale, visualizza sullo schermo le relative informazioni e offre la possibilità di accedere ad ulteriori approfondimenti e ad eventuali file multimediali (immagini, audio, video). Il visitatore del museo può quindi usufruire di questo dispositivo portatile che gli garantirà un valido supporto informativo alla visita.
Il palmare utilizza la tecnologia RFID (Radio Frequency Identification) che permette appunto il riconoscimento dell'oggetto utilizzando un segnale in radiofrequenza.
Il software per la presentazione delle informazioni all'utente, realizzato in Macromedia Flash, riceve l'Identificativo dell'oggetto passato tramite l'interfaccia RFID e trova le informazioni corrispondenti analizzando una base dati XML (Extensible Markup Language) memoizzata sullo stesso PDA .

Premesso questo, a me è stato assegnato il compito di realizzare una procedura di esportazione dati dal database centrale al formato compatibile con le specifiche XML e di filesystem richieste dall'applicazione Flash di presentazione installata sui palmari.

Di seguito è riportato uno schema sintetico dell'ambito in cui verrà utilizzata la procedura di esportazione dati XML.

Figura 1



La mia procedura sarà quindi un'applicazione web alla quale i musei potranno collegarsi per scaricare i file di aggiornamento (XML e file multimediali) per i rispettivi palmari.

Di seguito sono riportate alcune schermate del programma di presentazione installato sui palmari.

Figura 2

Dopo questa parte introduttiva entriamo nel merito della procedura di esportazione che ho intitolato *"EsPDAmuseo"*.

2 - Breve descrizione di *EsPDAmuseo*

EsPDAmuseo è una applicazione web scritta in linguaggio PHP che esporta i dati presenti sul database centrale convertendoli in formato XML secondo le specifiche richieste dall'applicazione multimediale realizzata in *Macromedia Flash* installata sui palmari per i visitatori dei musei.

3 - Specifiche dei file richiesti dall'applicazione Flash

3.1 - Struttura filesystem

Di seguito sono riportate le specifiche richieste dal programma per i PDA.

Le informazioni generate per ogni museo a partire dalla corrispondente base di dati, risiederanno nelle posizioni specificate di seguito nel file system del palmare.
Detta cartella base la cartella a partire dalla quale saranno presenti i dati di interesse, si prevede la seguente struttura:

```
.. (cartella base)
    |
    |_____admin
    |           |_____ita
    |           |_____eng
    |           |_____deu
    |           |_____fra
    |           |_____esp
    |
    |_____audio
    |           |_____ita
    |           |_____eng
    |           |_____deu
    |           |_____fra
    |           |_____esp
    |
    |_____images
    |           |
    |           |_____gallery
    |                   |_____largeimages
    |                   |_____otherimages
    |                   |_____thumbs
    |
```

```
|_____video
            |_____thumbs
            |_____video
                    |_____ita
                    |_____eng
                    |_____deu
                    |_____fra
                    |_____esp
```

Nella **cartella base** risiederanno l'applicativo sviluppato con il programma Macromedia Flash Professional 8.0 ed eventuali altri file di supporto correlati all'applicativo ma senza attinenza con le informazioni memorizzate nella base di dati.

All'interno della cartella **admin** saranno presenti, in ciascuna sottocartella specifica per le lingue supportate, tutti i file XML generati automaticamente dal sistema a partire dalla base di dati; quindi, per ogni lingua prevista, vi saranno i file di descrizione globale delle risorse disponibili quali gallery_ita.xml, gallery_eng.xml, gallery_deu.xml, gallery_fra.xml e gallery_esp.xml. Saranno inoltre riportati in queste sottocartelle i relativi documenti XML di descrizione delle immagini da usare per le "slideshow", ovvero sequenze di immagini collegate ad un particolare reperto museale che si ripetono con cadenza predefinita sullo schermo del palmare.

La cartella **audio** conterrà invece le descrizioni parlate, suoni, melodie, eccetera caratterizzanti gli oggetti presenti nel museo (in formato MPEG-1 Audio Layer 3 o, più semplicemente, MP3); se si tratta di file audio senza parole (tipo suoni o musiche) questi potranno essere memorizzati direttamente nella cartella audio stessa per evitare inutili replicazioni, altrimenti per ciascuna delle cinque lingue previste si prevede il posizionamento nelle sottocartelle corrispondenti.

Nella cartella **images** troveranno posto le immagini delle bandiere che simboleggiano la scelta della lingua da effettuarsi all'entrata nel museo da parte dei visitatori; la sottocartella **gallery** conterrà inoltre le tre sottocartelle **largeimages** (immagini a tutto schermo dei singoli oggetti presenti in vetrina), **otherimages** (immagini visualizzate durante le "slideshow") e **thumbs** (immagini di anteprima degli oggetti nella visualizzazione "da vetrina"). Per quanto riguarda i **video**, vale lo stesso discorso fatto per l'audio: nel caso di una distinzione per l'audio in base alla lingua, le diverse versioni potranno essere memorizzate nelle relative sottocartelle; attualmente laimpostazione grafica predisposta non sfrutta le possibili anteprime da abbinare ai video da collocarsi nella sottocartella **thumbs** dei video.

3.2 - File principale XML

(ad esempio gallery_ita.xml residente in /admin/ita)

Il file sarà composto dal tag radice **photoalbum** avente come sottoelementi il tag **parameters** (che deve essere il primo e rappresenta l'impostazione della visualizzazione delle vetrine passata all'applicativo in Flash) e un numero di tag di tipo **album** quante sono le vetrine presenti nel museo, ciascuna caratterizzata dal valore dell'attributo **albumname** che rappresenterà l'UID (codice identificativo univoco) della etichetta RFID che marca la particolare vetrina. Si specifica che quando si parla di vetrina si intende un reperto museale avente una tag RFID che lo identifica: può quindi trattarsi di una "vera" vetrina contenente più oggetti distinguibili tra loro e dotati ciascuno di propria descrizione caratterizzante (situazione riscontrabile al museo di Mineralogia), di una stanza con all'interno strumenti di misura/apparecchi destinati a particolari impieghi (adesempio al museo di Fisica), oggetti/beni singoli (una pianta nell'Orto botanico), di una vetrina "fittizia" creata per scopi particolari richiesti dall'applicativo in Flash (ad esempio la pagina iniziale al museo di Fisica che segnala la collezione di Giovanni Poleni i cui oggetti sono marcati con tag aventi sfondo rosso).

Di seguito un estratto del file con la struttura prevista:

```
<?xml version ="1.0"?>
<photoalbum>
<!-- Impostazione della grafica per la visualizzazione delle vetrine -->
<parameters noofcolumn="2" noofrow="3" thumbxs="0" thumbys="0" thumbwidth="120" thumbheight="90"/>
<!-- Schermata iniziale di descrizione uso del sistema per Mineralogia, Fisica e Orto Botanico -->
<album albumname="UID_fittizio_che_identifica_uso_sistema">
<images>
</images>
</album>
<!-- Schermata iniziale di descrizione tag rossi per collezione di Poleni nel museo di Fisica -->
<album albumname="UID_fittizio_che_ identifica_collezione_Poleni">
<images>
<image imagename="cartellino_rosso.jpg" number="01" infotext="Poleni" infotext2="informazioni base
su Poleni" infotext3="approfondimenti su Poleni" audio="audiopoleni.mp3" video="videopoleni.flv"
img="slideshowpoleni.xml"/>
<image imagename="cartellino_blu.jpg" number="02" infotext="" infotext2="XXXX"
infotext3="approfondimenti su XXXXX" audio="XXXX.mp3" video="XXXX.flv" img="slideshowpoleni.xml"/>
</images>
</album>
<!-- Prima vetrina -->
<album albumname="EO401234749392">
<images>
……
</images>
</album>
……
<!-- Seconda vetrina -->
<album albumname="EO401234749394">
<images>
……
</images>
</album>
……
<!-- N-esima vetrina -->
<album albumname="EO401234749999">
<images>
……
```

```
</images>
</album>
</photoalbum>
```

All'interno di ogni vetrina (tag **album**), ci saranno come sottoelementi del tag **images** un numero variabile di oggetti singolarmente distinguibili evidenziati ciascuno dal tag **image**. Di questi oggetti vi sarà il nome del file contenente la relativa immagine (valore dell'attributo **imagename**), il numero progressivo (valore dell'attributo **number** – compreso tra 01 e 18) occupato dal reperto nella vetrina e che verrà sovrapposto alle immagini a tutto schermo e alle anteprime, il titolo (valore dell'attributo **infotext**), la descrizione testuale (valore dell'attributo **infotext2**), approfondimenti testuali (valore dell'attributo **infotext3**), file audio, video e file XML con le impostazioni della slideshow (valori degli attributi **audio**, **video** e **img** rispettivamente). I valori degli attributi *infotext*, *infotext2* e *infotext3* saranno forniti come file di testo senza specifica formattazione (nella decodifica, da Flash vengono mantenuti solo gli "a capo"). Le lettere accentate saranno specificate tramite Codifica ASCII Estesa (ad esempio, "à" diventerà nel file XML "&#xe0").

Occorre notare che nel caso sia presente una descrizione "globale" della vetrina, questa sarà disposta come primo oggetto della vetrina stessa (con descrizione testuale, approfondimenti, immagini audio e video); inoltre, l'applicativo in Flash sarà strutturato in modo che per una vetrina con un singolo oggetto la visualizzazione di "default", ovvero la prima visualizzazione dopo che il lettore RFID ha ricevuto il riconoscimento della tag associata a quella particolare vetrina, consisterà nell'immagine a tutto schermo del reperto stesso (evitando quindi il passaggio con la visualizzazione "da vetrina").

Si osserva che per ogni elemento di una vetrina i valori degli attributi *infotext*, *infotext2*, *infotext3*, *audio* e *video* saranno in generale, come spiegato prima, dipendenti dalla lingua (ovvero, ad esempio, i corrispondenti valori in gallery_ita.xml e gallery_fra.xml saranno usualmente differenti).

Il programma di estrazione informazioni dal database dovrà pertanto agire solamente sui valori degli attributi dell N vetrine, tenendo inalterati gli altri campi del file XML.

3.3 - File di impostazione per le "slideshow"

(ad es. slideshow_ita.xml, residente in /admin/ita)

Il file presenta il tag **gallery** come nodo radice; i valori degli attributi per gallery specificano le modalità di funzionamento della sequenza di immagini (si vedano i commenti inseriti nell'esempio riportato di seguito). Saranno inoltre presenti un numero di tag **image** quante sono le immagini che

fanno parte della slideshow; ciascuna di queste immagini è identificata dal valore dell'attributo **path** che specifica il file corrispondente da visualizzare. Si prevede inoltre di inserire anche un ulteriore attributo **desc** per una possibile descrizione di ogni immagine della sequenza (per slideshow_ita.xml la descrizione sarà in italiano) tramite testo sovrapposto all'immagine stessa. Valgono a tal riguardo le stesse indicazioni proposte in precedenza per gli attributi testuali.

Il programma di estrazione informazioni dal database dovrà pertanto agire solamente sui valori degli attributi delle immagini di cui è composta la sequenza, tenendo inalterati gli altri campi del file XML.Un esempio di file che rappresenta una slideshow con 8 immagini è riportato di seguito.

```
<!--
'timer' :: number of seconds between each image transition
'order' :: how you want your images displayed. choose either 'sequential' or 'random'
'looping' :: if the slide show is in sequential mode, this stops the show at the last image (use
'yes' for looping, 'no' for not)
'fadeTime' :: velocity of image crossfade. Increment for faster fades, decrement for slower.
Approximately equal to seconds.
'xpos' :: _x position of all loaded clips (0 is default)
'ypos' :: _y position of all loaded clips (0 is default)
-->
<gallery timer="5" order="sequential" fadetime="2" looping="yes" xpos="0" ypos="0">
<image path="images/gallery/otherimages/01.jpg" desc="Immagine 01" />
<image path="images/gallery/otherimages/02.jpg" desc="Immagine 02" />
<image path="images/gallery/otherimages/03.jpg" desc="Immagine 03" />
<image path="images/gallery/otherimages/04.jpg" desc="Immagine 04" />
<image path="images/gallery/otherimages/05.jpg" desc="Immagine 05" />
<image path="images/gallery/otherimages/06.jpg" desc="Immagine 06" />
<image path="images/gallery/otherimages/07.jpg" desc="Immagine 07" />
<image path="images/gallery/otherimages/08.jpg" desc="Immagine 08" />
</gallery>
```

4 - Il funzionamento di *EsPDAmuseo*

EsPDAmuseo si presenta all'utente tramite interfaccia web.
La procedura segue sommariamente questi passi:

1. La prima schermata web richiede all'utente la selezione del museo del quale si vogliono esportare i dati. Attualmente essendo la procedura in fase di prova c'è anche la selezione della lingua (italiano, inglese, francese, spagnolo, tedesco) assieme all'opzione di esportazione in tutte le lingue.

2. Dopo aver ricevuto i dati in input (museo e lingua) la procedura crea sul filesystem del server web in cui risiede la struttura di directory necessaria all'esportazione cioè quella compatibile alle specifiche descritte alla sezione "Specifiche dei file richiesti dall'applicazione Flash". Quindi viene creata una cartella nominata con il codice del museo selezionato (ad esempio "min" per il museo di mineralogia) e a partire da questa tutte le sottocartelle (admin, images, video...) che conterranno i file di esportazione xml e di supporto (file multimediali etc...). I file scritti da

precedenti esportazioni vengono cancellati all'inizio di una nuova esportazione.

3. Per ognuna delle lingue selezionate la procedura esporta le informazioni destinate ad essere visualizzate sul palmare, scrive il file gallery_*lingua*.xml , gli eventuali file xml delle slideshow e i documenti multimediali allegati, memorizzandoli nelle rispettive cartelle come specificato nella sezione “Specifiche dei file richiesti dall'applicazione Flash”.
   Al termine dell'esportazione in ciascuna lingua la procedura si ferma e visualizza una schermata di report delle operazioni svolte sul filesystem indicando un link per ogni file creato in modo che l'utente possa controllare in anteprima il risultato dell'esportazione.
   La procedura inoltre memorizza nel database il risultato dell'esportazione XML compresi i file multimediali.
   La parte di creazionedel contenuto dei file xml è la più laboriosa dell'esportazione in quanto si devono ricercare dati provenienti da varie tabelle del database in modo diversificato a seconda della tipologia di museo, del tipo di dati, ottenendo informazioni risultanti dall'aggregazione di più campi, abbinandoli perdipiù ad altrettante etichette e considerando allo stesso tempo la multicanalità linguistica. Per quanto riguarda l'esportazione dei file multimediali, questi ultimi sono memorizzati con tutto il loro contenuto nel database. Per l'esportazione di questi file *EsPDAmuseo* deve trovare la path di memorizzazione sul filesystem in modo da collocarli in conformità alle specifiche richieste dall'applicazione del palmare.
   Oltre a visualizzare il report delle operazioni eseguite sul filesystem e sul database la procedura visualizza il testo del file gallery_*lingua*.xml di volta in volta creato per ogni lingua.
4. Quando tutte le esportazioni nelle lingue selezionate sono concluse, *EsPDAmuseo* crea un archivio Zip contenente tutti i file risultanti dall'esportazione ed un link per il download. Scaricando il file Zip si potrà quindi aggiornare i Palmari estraendo i file nelle apposite cartelle.

Di seguito sono riportate alcune schermate web del programma.

Figura 3 “Pagina iniziale – selezione museo e lingue”

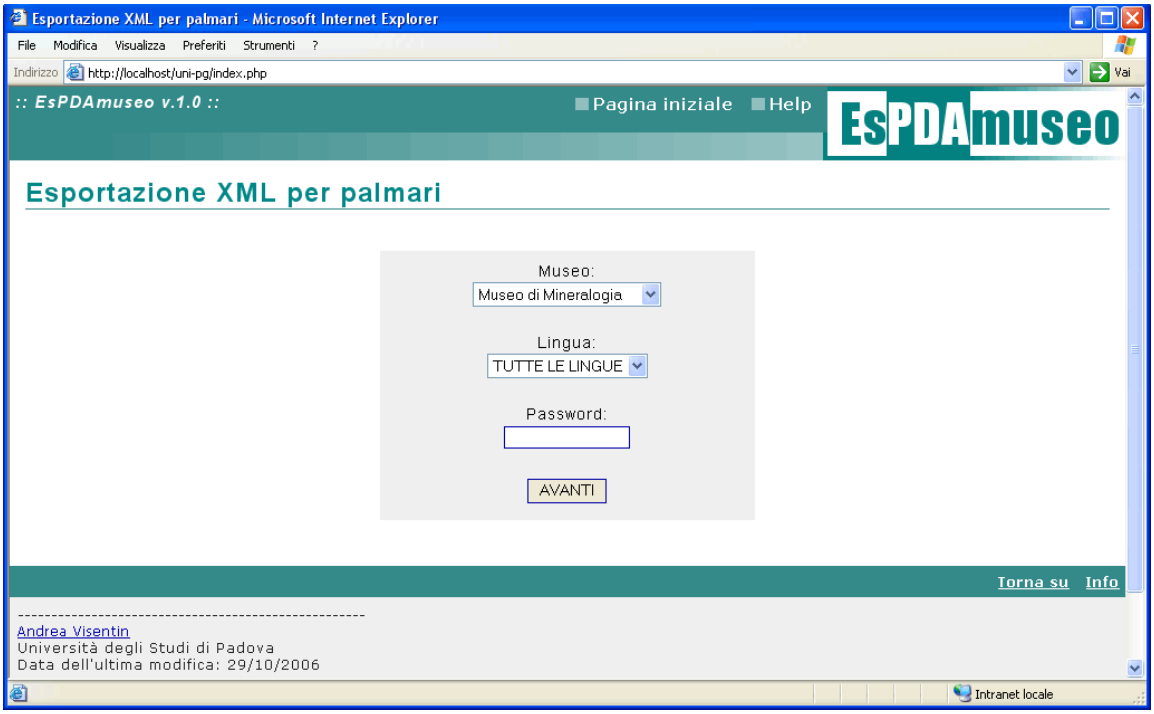


figura 4 “Un passo dell'esportazione”

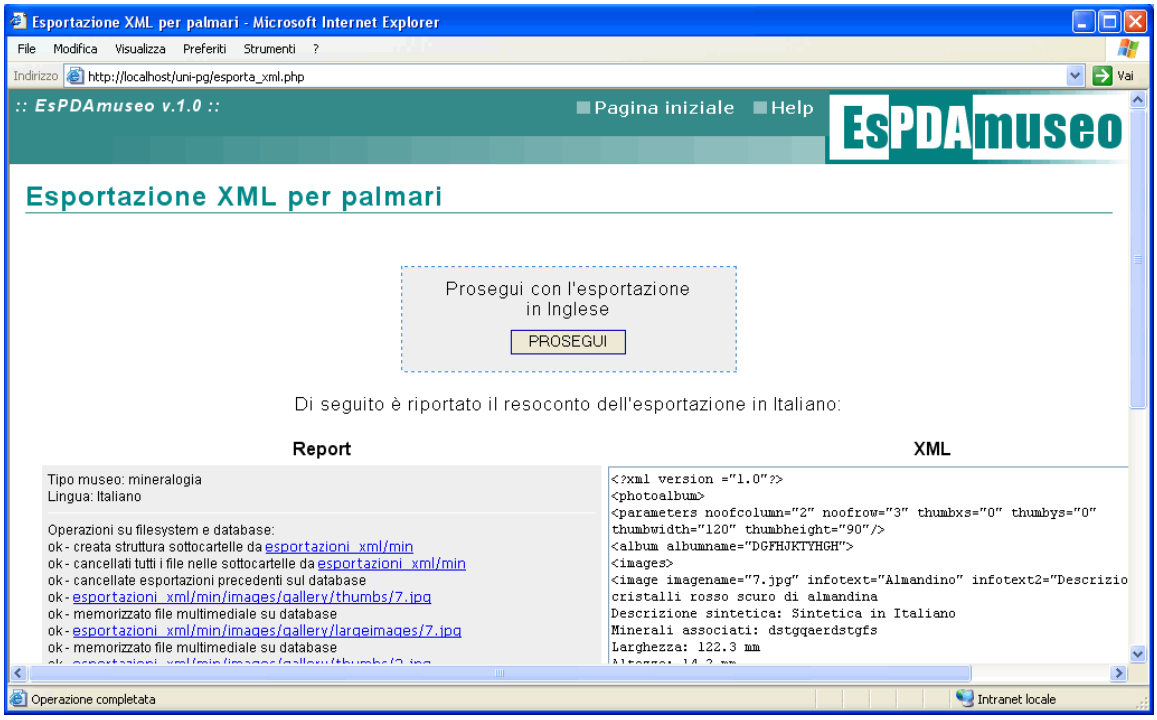


figura 5 “Possibilità di visualizzare i file esportati cliccando sui link del report”

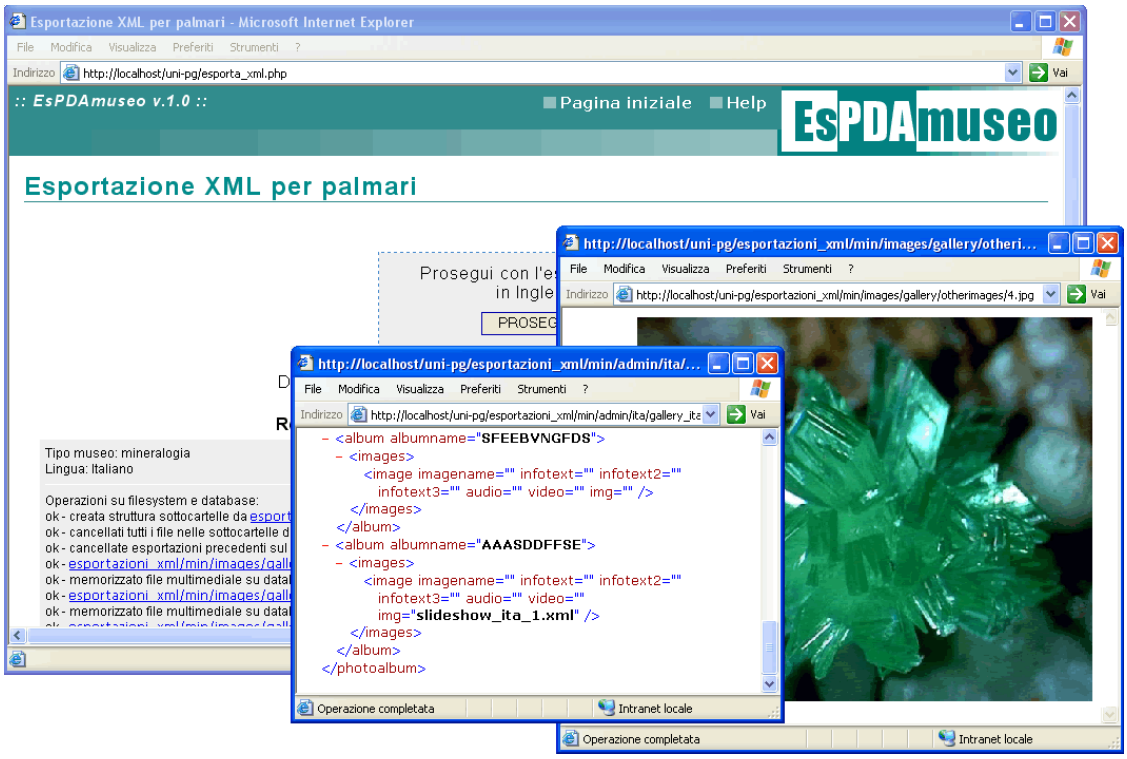
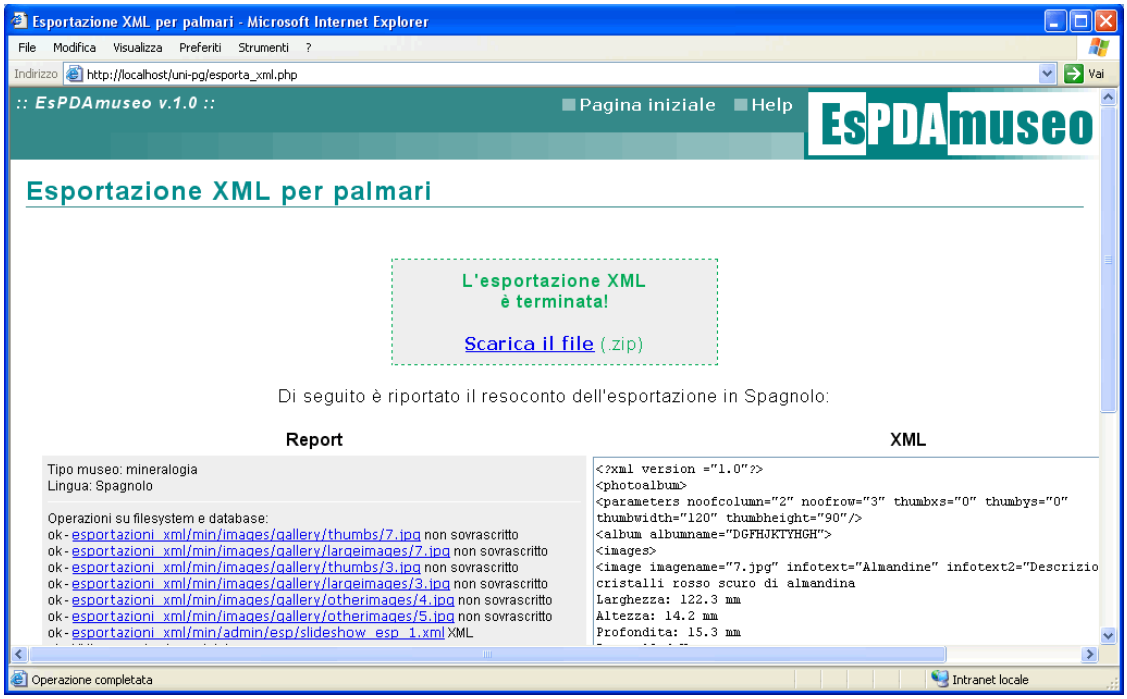


figura 5 “Schermata finale con download del file Zip”



5 - Breve descrizione del database centrale

Come introduzione all'approfondimento delle funzionalità della procedura *EsPDAmuseo* segue una breve descrizione delle parti del database coinvolte nell'esportazione dati per i PDA.

Una parte del database è dedicata alle informazioni generali dei musei (denominazione, infomazioni generali, contatti, etc...).

Un gruppo di tabelle del database è dedicato esclusivamente ai dati per il funzionamento della procedura di presentazione sul Palmare (palm_esemplare, palm_rfid, palm_aggregazione , palm_oggetto_mm , etc...). Queste tabelle contengono i codici RFID dei reperti, le informazioni descrittive, i raggruppamenti nelle vetrine, i file multimediali abbinati etc...

Vari gruppi di tabelle differenziati per seguire le specifiche necessità informative delle quattro tipologie di musei (mineralogia, orto botanico, paleontologia, fisica). Le tabelle sono strutturate rispettando le normative dell'ICCD.
Quindi ci sono tabelle dedicate a mineralogia (min_scheda_esemplare, min_ubicazione, min_nome_minerale, min_collezione_di_provenienza, etc...) , tabelle relative ad orto botanico (obt_scheda_esemplare, obt_al_nome_famiglia, obt_al_nome_pianta_lat, etc...) e così anche per paleontologia e fisica.

Per quanto riguarda il supporto multilingua la quasi totalità delle tabelle destinate a contenere informazioni descrittive prevedono l'abbinamento al codice lingua in modo da consentire la memorizzazione delle traduzioni sul database.

6 - L'implementazione di EsPDAmuseo

6.1 - Il supporto tecnologico utilizzato

EsPDAmuseo è stato implementato con linguaggio di scripting Php (www.php.net) su server web HTTP Apache (www.apache.org). Il database è realizzato in PostgreSQL 8.1 (www.postgresql.org). Per il supporto delle funzionalità che ho incluso nel programma è necessario disporre della versione PHP 4.3.0 o superiore.

6.2 - I file del programma

I file php che compongono EsPDAmuseo sono di 2 tipologie distinte:

1. File principali che corrispondono a rispettive pagine web.

2. File di inclusione che contengono programnmazione e funzioni che per chiarezza di leggibilità del codice sono stati separati dalle pagine principali.

I file principali sono i seguenti:

- index.php
  è la prima pagina web dell'applicazione in cui viene visualizzato il form di inserimento dati per la selezione del museo e delle lingue.

- esporta_xml.php
  è il centro dell'applicazione EsPDAmuseo. Contiene il codice che eseguire i vari passi dell'esportazione nelle lingue selezionate. Trova i reperti museali da visualizzare sul palmare, le relative descrizioni, i file multimediali corrispondenti e memorizza i risultati dell'esportazione sia su filesystem che su database visualizzando i rispettivi report fino ad arrivare al link per il download dell'archivio zip.

- help.php, info.php
  sono pagine web con contenuto statico a scopo di documentazione del programma.

I file di inclusione sono i seguenti:

- config_connessione.php
  file che inizializza la stringa dei parametri di connessione al database

- config_lingue.php
  contiene le configurazioni per i codici delle lingue da prendere in considerazione per l'esportazione. Nel file è predisposta la possibilità di prendere i codici delle lingue dal database oppure inizializzare le lingue scrivendo direttamente i codici manualmente.
  In più c'è la possibilità di configurare una variabile che definisce la lingua di default considerata ogni qualvolta la procedura non trovasse l'informazione nella lingua ricercata.

- crea_archivio_zip.php
  genera il file di archivio Zip. In particolare questa procedura crea lo zip contenente tutti i file e le sottocartelle a partire da una directory specificata. Ho realiazzato questa procedura attenendomi alle specifiche pubbliche del formato Zip (www.pkware.com).

- funzioni_fs_db.php
  contiene funzioni per le varie operazioni di scrittura sul filesystem e sul database necessarie per l'esportazione. In dettaglio ci sono le seguenti funzioni:

  - la funzione che inizializza l'esportazione creando la struttura di cartelle sul filesystem

(\admin, \images, \audio, \video etc...) e cancellando i file dell'eventuale esportazione precedente.

- la funzione di cancellazione della precedente esportazione sul database: elimina le registrazioni dell'eventuale esportazione già presente relativa al museo selezionato.
- la funzione che esporta il dato Binary Large Object cioè il contenuto del file multimediale memorizzato nel database e lo scrive sul filesystem.
- la funzione che scrive il file slideshow_*lingua*.xml attenendosi al formato definito dalle specifiche per l'esportazione e chiama la funzione per la memorizzazione di tale file sul database.
- La funzione che memorizza il file gallery_*lingua*.xml su filesystem e chiama la funzione di memorizzazione dell'xml sul database.
- La funzione che registra l'xml risultato dell'esportazione sul database. Utilizzata per memorizzare i file *gallery* e i file *slideshow*.
- La funzione che registra sul database le path dei file multimediali creati dall'esportazione abbinandoli all'id dell'oggetto binario già presente sul database.

- Trova_allegati_mm.php
  questo file contiene le query che permettono di trovare i file multimediali (immagini, audio, video) associati al reperto museale.
- Intestazione.php, pie_pagina.php
  file contenenti html e grafica per l'intestazione ed il piè di pagina uguale per tutte le pagine web di EsPDAmuseo.
- Fisica.php, mineralogia.php, orto_botanico.php, paleontologia.php
  contengono il codice e le query per estrarre i vari dati del reperto in base alla sua tipologia. Come già accennato prima, il database riserva tabelle diverse per ciascuna tipologia di museo.
- Scrivi_tag_image.php
  si occupa di creare un nodo dell'xml di esportazione con le informazioni di un esemplare ossia del reperto museale.
- Stile.css
  il foglio di stile dell'interfaccia web *EsPDAmuseo*.

6.3 - Dettagli tecnici

Inizia ora la parte che spiega approfonditamente il funzionamento di *EsPDAmuseo* analizzando la logica di realizzazione, il codice, le query sul database e le tabelle che è stato necessario aggiungere per il funzionamento dell'esportazione.

6.3.1 - La pagina iniziale: index.php

La pagina web iniziale della procedura è il file *index.php* in cui viene visualizzato il form di selezione del museo e delle lingue.

Il codice inizia con le inclusioni necessarie. Il funzionamento di ogni file di inclusione è stato spiegato sommariamente alla sezione “I file del programma”.

```
<?php
require 'intestazione.php'; //intestazione pagina html
require 'config_connessione.php'; //stringa di connessione al database
$connessione = pg_connect($stringa_di_connessione) or die ("non sono riuscito a connettermi"); //connessione DB
require 'config_lingue.php'; //inizializzazione lingue
?>
```

Scrive il form di input per la pagina *esporta_xml.php* ed il controllo per selezionare il museo che viene passato tramite il codice univoco *sigla_museo*:

```
<center>
<div style="text-alignment:center;width:35%;background-color:#eee;padding:10px;">
<form name="form1" action="esporta_xml.php" method="post">
Museo:
<br>
<select name="sigla_museo">
      <option value="0">--  seleziona  --</option>
<?php
//prende l'elenco dei musei
$query = "select * from palm_museo order by denominazione";
$risultato = pg_query($query);
while($row_museo = pg_fetch_object($risultato))
{
  echo "<option value=\"$row_museo->sigla_museo\">$row_museo->denominazione</option>";
}
?>
</select>
```

Scrive il controllo selezione lingue:

```
<p>

Lingua:
<br>
<select name="sel_lingua">
      <option value="0">--  seleziona  --</option>
<?php
//compila il select con le lingue aggiungendo anche TUTTE LE LINGUE
$opzione_tutte_le_lingue="";
$altre_opzioni="";
foreach ($arr_lingue as $k => $v)  // $arr_lingue è il vettore inizializzato in config_lingue.php
{
   if ($opzione_tutte_le_lingue != "") $opzione_tutte_le_lingue .= "|";
   $opzione_tutte_le_lingue .= $k;
   $altre_opzioni .= "<option value='". $k . "'>". $v . "</option>\n";
}
echo "<option value='$opzione_tutte_le_lingue'>TUTTE LE LINGUE</option>\n". $altre_opzioni;
?>
</select>
</p>
</p>
```

Da notare come viene creato il valore dell'opzione ”TUTTE LE LINGUE” : includendo i codici di tutte le lingue.

Di seguito c'è la definizione del campo nascosto *azione_specifica_passo*. Questo parametro serve ad indicare alla pagina *esporta_xml.php* eventuali azioni aggiuntive da eseguire nel relativo passo della procedura di esportazione. Come anticipato prima nel paragrafo dedicato alla logica di funzionamento di *EsPDAmuseo* la procedura si divide in passi, ciascuno corrispondente all'esecuzione dell'esportazione in una lingua. Così se l'utente seleziona “Tutte le lingue” la procedura sarà divisa in 5 passi (italiano, inglese, francese, tedesco, spagnolo). In base al valore del parametro *azione_specifica_passo* ricevuto in HTTP POST la pagina *esporta_xml.php* è predisposta ad eseguire specifiche operazioni prima di proseguire con l'esportazione nella lingua prevista. In questo caso *azione_specifica_passo*= “imposta_cartelle” comanda alla procedura di inizializzare la struttura a cartelle sul filesystem.

```
<input type="hidden" name="azione_specifica_passo" value="imposta_cartelle">
```

Scrive il campo password:

```
<input type="hidden" name="azione_specifica_passo" value="imposta_cartelle">
Password:<br><input name="pass" size="15" maxlength="20" type="password">
</p>
```

Attenzione! Il campo password non viene ancora utilizzato dalla procedura ma l'ho tenuto lo stesso per evidenziare la necessità che ha *EsPDAmuseo* di una politica di sicurezza . Infatti questa versione di *EsPDAmuseo* potenzialmente può accettare uno o più utenti “contemporaneamente” essendo appunto un'applicazione web. L'applicazione scrive sulle stesse cartelle per lo stesso museo selezionato. Quindi il problema è che se ci sono più utenti che eseguono “contemporaneamente” la procedura per lo stesso museo, non è garantita l'esatta esportazione dei dati visto che all'inizio vengono cancellate le esportazioni già esistenti (si pensi a questo punto a cosa succederebbe se un utente iniziasse la procedura di esportazione durante una fase avanzata di un'altra esportazione di un altro utente; quest'ultimo si troverebbe l'esportazione con dei file mancanti!). Comunque un'opportuna politica di sicurezza per questo problema non sarà difficile da implementare. Si po' pensare di bloccare l'inizio dell'esportazione ad altri utenti per lo stesso museo finchè l'eventuale fase di esportazione in corso non sia conclusa. Oppure si può pensare di dedicare cartelle diverse per ogni utente, magari impostando delle utenze abilitate all'utilizzao di *EsPDAmuseo*. Oppure basandosi sul codice univoco delle sessioni di accesso alla procedura. O semplicemente limitando gli accessi alla procedura!

```
<input type='submit' name="submit1" value="AVANTI" style='cursor:hand;'>
</form>
</div>
</center>
<br><br><br>
<?php
pg_close($connessione); //chiude la connessione al DB
require 'pie_pagina.php'; //piè di pagina
?>
```

6.3.2 - Il centro dell'applicazione: esporta_xml.php

Controlla che i parametri passati per *http post* siano valorizzati altrimenti rimanda alla pagina iniziale. Da notare che dall'opzione delle lingue selezionate viene creato un vettore.

```
<?php
// prende i valori passati dal form della pagina web precedente e
// rimanda il browser alla pagina iniziale se mancano informazioni in input
$dati_input = true;
if ($_POST["sigla_museo"] == "0")
  $dati_input = false;
else
  $sigla_museo = $_POST["sigla_museo"];
```

```
if ($_POST["sel_lingua"] == "0")
  $dati_input = false;
else
{
        $arr_lingue_selezionate = explode("|", $_POST["sel_lingua"]);
        $id_lingua=$arr_lingue_selezionate[0];
}

$parola_chiave = $_POST["pass"]; // inutilizzata
$azione_specifica_passo = $_POST["azione_specifica_passo"];
if ($dati_input == false)
{
  header("Location: index.php"); // rimanda il browser alla pagina iniziale
  exit;
}

require 'intestazione.php';
require 'config_connessione.php';
$connessione = pg_connect($stringa_di_connessione) or die ("non sono riuscito a connettermi");
require 'config_lingue.php';
require 'funzioni_fs_db.php';
?>
```

Tra i file inclusi c'è anche *funzioni_fs_db.php* contenente le varie funzioni per la scrittura dei file esportati e per la loro registrazione sul database.

Di seguito c'è la funzione dedicata a trovare le etichette dei dati. Per capire a cosa serva questa funzione è necessario sapere che per ogni reperto la procedura di esportazione estrapola dal database i vari dati informativi (ad esempio nome, genere, provenienza, dimensioni etc..) per aggregarli poi insieme a formare 3 frasi descrittive che sono poi gli attributi infotext, infotext2, infotext3 per ogni tag <image> dell'XML generato. Si veda la sezione “File principale XML”. Detto ciò ogni dato deve avere la sua etichetta, ad esempio per l'informazione relativa ai minerali associati memorizzato nel campo *minerali_associati* della tabella *min_ha_descrizione_sintetica* si deve anteporre la stringa “minerali associati: ” ciè l'etichetta del dato. Queste etichette non sono memorizzate sul database. Premesso che ognuna di esse deve avere la rispettiva traduzione nelle varie lingue e che non sarebbe comodo memorizzare le traduzioni direttamente nel codice del programma *EsPDAmuseo* vista la quantità (circa 60 etichette + le traduzioni) e la necessità di aggiornamenti, ho pensato di aggiungere due tabelle al database:

1. tabella ***xml_etichetta_campo***
   ogni record contiene il nome della tabella e del campo al quale si riferisce l'etichetta ed il valore di default dell'etichetta . Ecco l'SQL per la creazione di xml_etichetta_campo:

```
CREATE TABLE xml_etichetta_campo
(
 id_etichetta int4 NOT NULL,
 tabella_database varchar(100) NOT NULL,
 campo_tabella varchar(100) NOT NULL,
 etichetta_default varchar(255) NOT NULL,
 CONSTRAINT id_etichetta_pkey PRIMARY KEY (id_etichetta)
)
```

2. tabella ***xml_etichetta_campo_traduzione***

```
CREATE TABLE xml_etichetta_campo_traduzione
(
id_etichetta int4 NOT NULL,
id_lingua varchar(255) NOT NULL,
traduzione varchar(255) NOT NULL
)
```

Chiarita la questione delle etichette proseguiamo con la funzione dedicata a trovarle.
Tale funzione richiede in input il nome della tabella e il campo relativi al dato più il codice della lingua considerata. Da notare il vettore statico $arr_etichette che memorizza le etichette già trovate ed evita ulteriori accessi al database; si pensi che questa funzione verrà chiamata ad ogni esportazione relativa ad una lingua circa 20 volte per ogni reperto quindi $arr_etichette evita migliaia di accessi al database.

```
<center>
<?php
//trova l'etichetta
function trova_etichetta($tabella, $campo, $lingua)
{
  static $arr_etichette;
  $etichetta_presente = false;
  if (count($arr_etichette) > 0)
  {
     if (array_key_exists("$tabella-$campo", $arr_etichette)) $etichetta_presente = true;
  }
  if ($etichetta_presente != true)
{
        $query = "select * from xml_etichetta_campo where (tabella_database='$tabella') and
        (campo_tabella='$campo');";
        $risultato_etichetta = pg_query($query);
        $etichetta = "";
        $id_etichetta = 0;
        while($riga_etichetta = pg_fetch_object($risultato_etichetta))
```

```
        {
                $id_etichetta = $riga_etichetta->id_etichetta;
                $etichetta = $riga_etichetta->etichetta_default;
        }
        $query = "select * from xml_etichetta_campo_traduzione where (id_etichetta=$id_etichetta) and
        (id_lingua='$lingua');";

        $risultato_etichetta = pg_query($query);
        while($riga_etichetta = pg_fetch_object($risultato_etichetta))
        {
                $etichetta=$riga_etichetta->traduzione;
        }
        $arr_etichette["$tabella-$campo"]=$etichetta;
  }
  return $arr_etichette["$tabella-$campo"];
}
```

La seguente funzione serve a trovare le traduzioni dei dati descrittivi per le tabelle organizzate con *id_lingua*. In input vengono richiesti la query e il campo della tabella da tradurre. Il parametro $query_base accetta il testo di una qury il cui recordset risultante deve contenente fra gli altri anche il campo id_lingua.

```
//trova la traduzione nelle tabelle predisposte con id_lingua
function traduci($query_base, $campo)
{
        global $id_lingua, $traduzione_default;
        $arr_id_lingue = array($id_lingua, $traduzione_default);
        $traduzione=NULL;
        foreach ($arr_id_lingue as $lingua)
        {
                if ($traduzione==NULL)
                {
                        $query=$query_base . " and (id_lingua='$lingua');";
                        $risultato_traduzione = pg_query($query);
                        if ($riga_traduzione = pg_fetch_object($risultato_traduzione))
                                $traduzione=$riga_traduzione->$campo;
                }
        }
        return($traduzione);
}
```

La seguente funzione serve a convertire i caratteri speciali per creare l'xml valido. Il carattere “<” ad esempio è riservato al formato XML e deve essere convertito in “<”

```
//METTE I CARATTERI DI ESCAPE SU XML
function converti_caratteri_speciali($testo)
{
 return (htmlentities($testo,ENT_QUOTES));
}
```

la variabile $testo_report contiene l'html del report che viene visualizzato al termine di ogni passo dell'esportazione. Elenca tutte le operazioni di esportazione eseguite su filesystem e sul database.

```
//report delle azioni rilevanti eseguite dal programma
$testo_report = "";
```

Viene trovata la tipologia del museo selezionato cioè (mineralogia, orto botanico, paleontologia o fisica). Siccome per ogni tipo di museo i reperti contengono informazioni su tabelle distinte (ad esempio mineralogia usa la tabella min_scheda_esemplare, orto botanico usa la tabella obt_scheda_esemplare etc...) quindi il programma controlla la tipologia di museo controllando quali tabelle vengono utilizzate dai rispettivi reperti.

```
//trova tipo museo (mineralogia, orto botanico, paleontologia, fisica...), per fare poi le query sulle tabelle di
competenza
$tipo_museo = NULL;
$arr_tab_tipo = array('min_scheda_esemplare','obt_scheda_esemplare','bnp_ss','pst_og');
$arr_tipo = array('mineralogia','orto_botanico','paleontologia','fisica');
$cont = 0;
foreach ($arr_tab_tipo as $tabella)
{
        $query = "select count(dba_idinterno) from $tabella where (dba_idinterno in (select dba_idinterno from
                palm_esemplare where sigla_museo='$sigla_museo'));";
        $risultato_db = pg_query($query);
        $riga_db = pg_fetch_row($risultato_db);
        if ($riga_db[0]>0) $tipo_museo = $arr_tipo[$cont];
        $cont += 1;
}
$testo_report  .= "Tipo museo: $tipo_museo <br> Lingua: $arr_lingue[$id_lingua]<br>
                        <hr style='color:#fff;' >Operazioni su filesystem e database:<br>";
```

Controlla il valore di $azione_specifica_passo ed esegue le azioni programmate. In questa versione di *EsPDAmuseo* c'è una sola azione che può comandare il parametro $azione_specifica_passo ma se ne potranno aggiungere delle altre semplicemente mettendo altre condizioni *if ()*.

```
//imposta la struttura filesystem con cartella base "sigla_museo" se siamo al primo passo
if ($azione_specifica_passo == "imposta_cartelle") inizializza_directory_museo($sigla_museo);
```

Per ogni file slideshow si è pensato di creare un nome univoco aggiungendo semplicemente un numero sequenziale inizializzato a 0 ad ogni esportazione:

```
//inizializza variabili per museo e lingua
$contatore_elenco_slideshow = 0;   //variabile che serve a dare un nome univoco ai file slideshow
```

ed ecco che iniziamo a scrivere l'XML... ma solo il primo testo statico...

```
$testo_xml= "<?xml version =\"1.0\"?>\n<photoalbum>\n<parameters noofcolumn=\"2\" noofrow=\"3\"
thumbxs=\"0\" thumbys=\"0\" thumbwidth=\"120\" thumbheight=\"90\"/>\n";

$accapo=chr(13) . Chr(10); // per andare a capo riga nel testo xml
$numero_vetrina = 0;
```

Inizia la prima query sul database per trovare gli esemplari ossia i reperti del museo contrassegnati con RFID (Le "Specifiche dei file richiesti dall'applicazione Flash" indicano che questi oggetti devono essere registrati nell'xml come se fossero vetrine contenenti un solo oggetto, cioè se stessi).
Di seguito sono elencate le tabelle considerate nella query con i campi strettamente necessari al nostro contesto e tralasciando i vincoli di integrità referenziale:

1. tabella: **palm_esemplare**
   campi: **dba_idinterno** chiave primaria, **sigla_museo**
   descrizione: associa i codici dba_idinterno ai rispettivi musei. Ogni dba_idinterno corrisponde ad un reperto del museo;

2. tabella: **palm_esemplare_ha_rfid**
   campi: **dba_idinterno, id_rfid**
   descrizione: abbina i codici RFID ai reperti;

3. tabella: **palm_rfid**
   campi: **codice_rfid** chiave primaria, **mostrare_su_palmare**
   descrizione: contiene i codici RFID ed il campo che permette la visualizzazione sul palmare

La query sarà allora una *SELECT* che troverà tutti i codici dba_idinterno alle seguenti condizioni:

- ✔ sigla_museo=$sigla_museo
  $sigla_museo è la variabile del codice museo selezionato dall'utente
- ✔ palm_esemplare.dba_idinterno=palm_esemplare_ha_rfid.dba_idinterno
- ✔ palm_esemplare_ha_rfid.id_rfid=palm_rfid.id

- ✔ mostrare_su_palmare=true

```
//seleziona gli esemplari che hanno il codice RFID

$query = "select palm_esemplare.dba_idinterno, palm_rfid.codice_rfid from palm_esemplare,
palm_esemplare_ha_rfid, palm_rfid where (palm_esemplare.sigla_museo='$sigla_museo') and
(palm_esemplare.dba_idinterno=palm_esemplare_ha_rfid.dba_idinterno) and
(palm_esemplare_ha_rfid.id_rfid=palm_rfid.id) and (mostrare_su_palmare=true);";

$risultato_esemplari = pg_query($query);
```

A questo punto inizia il ciclo che considera ogni **dba_idinterno** trovato.

```
while($riga_esemplari = pg_fetch_object($risultato_esemplari))
{
  $dba_idinterno=$riga_esemplari->dba_idinterno;
```

Viene accodato al testo XML un nuovo tag vetrina **<album albumname="RFID">** (ogni reperto trovato in questa query viene considerato come una vetrina contenente un singolo oggetto, cioè se stesso) l'attributo albumname contiene il codice RFID. In più viene aggiunto anche il tag **<images>** che avrà poi i nodi figli **<image>,** ma in questo caso ci sarà un unico nodo **<image>** visto che la vetrina contiene un solo oggetto.

```
 //apro nodo album incrementando il numero consecutivo di vetrina
 $numero_vetrina += 1;
 $testo_xml .= "<album albumname=\"$riga_esemplari->codice_rfid\">\n";
 $testo_xml .= "<images>\n";  //apro nodo images
```

Includo il file *scrivi_tag_image.php* che si occupa di creare un tag xml con le informazioni dell'esemplare ossia del reperto museale identificato da **dba_idinterno**. Il nodo che sarà scritto ha la seguente struttura: <**image** imagename="" infotext="" infotext2="" infotext3="" audio="" video="" img="" /> .

```
 require("scrivi_esemplare.php"); //scrivo il tag image con tutti i suoi parametri
 $testo_xml .= "</images>\n"; //chiudo nodo images
 $testo_xml .= "</album>\n"; //chiudo nodo album
}
```

Inizia la seconda query cioè la selezione di tutte le vetrine che contengono più oggetti . Si vedano le "Specifiche dei file richiesti dall'applicazione Flash" per la spiegazione di cosa si intende per "vetrina". Il database nomina le "vetrine" che contengono oggetti con il termine "aggregazioni" e memorizza i dati ad esse relativi in opportune tabelle. Chiarisco subito questo concetto presentando le tabelle coinvolte nella prossima query; come fatto prima indico solo i campi

strettamente necessari al contesto e tralascio i vincoli di integrità:

1. tabella: **palm_aggregazione_tagged**
   campi: **id_aggregazione** chiave primaria, **sigla_museo, nome_aggregazione**
   descrizione: associa i codici id_aggregazione ai rispettivi musei.

2. tabella: **palm_aggregazione_ha_rfid**
   campi: **id_aggregazione, id_rfid**
   descrizione: associa un'aggregazione al proprio codice RFID;

3. tabella: **palm_rfid**
   campi: **codice_rfid**, **mostrare_su_palmare**
   già vista prima.

La query sarà allora una *SELECT* che troverà tutti i codici id_aggregazione e i corrispondenti RFID alle seguenti condizioni:

- ✔ sigla_museo=$sigla_museo
  $sigla_museo è la variabile del codice museo selezionato dall'utente
- ✔ palm_aggregazione_ha_rfid.id_aggregazione=palm_aggregazione_tagged.id_aggregazione
- ✔ palm_rfid.codice_rfid=palm_aggregazione_ha_rfid.id_rfid
- ✔ mostrare_su_palmare=true

```
//trova le aggregazioni ossia le vetrine che hanno il codice RFID

$query = "select palm_aggregazione_tagged.id_aggregazione, palm_rfid.codice_rfid from
palm_aggregazione_tagged, palm_aggregazione_ha_rfid, palm_rfid where (sigla_museo='$sigla_museo') and
(palm_aggregazione_ha_rfid.id_aggregazione=palm_aggregazione_tagged.id_aggregazione) and
(palm_rfid.codice_rfid=palm_aggregazione_ha_rfid.id_rfid) and (mostrare_su_palmare=TRUE);";

$risultato_aggregazioni = pg_query($query);

while($riga_aggregazioni = pg_fetch_object($risultato_aggregazioni))
{
 //apro nodo album incrementando il numero consecutivo di vetrina
 $numero_vetrina += 1;
 $testo_xml .= "<album albumname=\"$riga_aggregazioni->codice_rfid\">\n";
 $testo_xml .= "<images>\n"; //apro nodo images
```

A questo punto trovo gli esemplari che appartengono all'aggregazione. Sul database è memorizzata la seguente tabella:

- tabella: **palm_esemplare_ha_aggregazione**
  campi: **dba_idinterno**, **id_aggregazione**
  descrizione: associa gli esemplari alle aggregazioni

Facendo una query su questa tabella trovo gli esemplari corrispondenti all'aggregazione identificata da id_aggregazione. Per ogni dba_idinterno includo il file *scrivi_tag_image.php* come ho fatto per il ciclo precedente.

```
//trova gli esemplari che appartengono all'aggregazione

$query = "select dba_idinterno from palm_esemplare_ha_aggregazione where
(id_aggregazione=$riga_aggregazioni->id_aggregazione);";

$risultato_esemplari = pg_query($query);
while($riga_esemplari = pg_fetch_object($risultato_esemplari)
{
   $dba_idinterno=$riga_esemplari->dba_idinterno;
   require("scrivi_tag_image.php"); //scrivo il tag image con tutti i suoi parametri
}
$testo_xml .= "</images>\n"; //chiudo nodo images
$testo_xml .= "</album>\n"; //chiudo nodo album
}
```

A questo punto scrivo il tag di chiusura del testo xml e chiamo la funzione di memorizzazione di gallery.xml la quale scriverà il file sul filesystem e sul database. In seguito vedremo anche questa funzione.

```
//chiudo nodo photoalbum
$testo_xml .= "</photoalbum>\n";
scrivi_file_gallery("esportazioni_xml/$sigla_museo/admin/$id_lingua/gallery_$id_lingua.xml" , $testo_xml);
?>
```

Il passo di esportazione per la lingua considerata è concluso.

Ora, se il vettore $arr_lingue_selezionate contiene ancora codici lingue allora significa che si deve proseguire con un altro passo, altrimenti viene creato lo zip. Il seguente listato controlla se ci sono ancora altri passi da fare, in tal caso viene creato un form *http post* con il pulsante “PROSEGUI” in modo da poter proseguire con la lingua successiva. Questo form è indirizzato alla medesima pagina web *esporta_xml.php* che si troverà a fare le stesse operazioni che ha svolto per il passo appena compiuto ma per una lingua diversa. I campi del form (sigla_museo, sel_lingua, etc...) vengono compilati automaticamente dal programma: il campo *sel_lingua* questa volta conterrà una lingua in meno, mentre il campo *azione_specifica_passo* non avrà più il valore

“imposta_cartelle”. Se invece l'esportazione è conclusa allora il programma visualizza il messaggio “ESPORTAZIONE XML CONCLUSA” e crea il file archivio Zip con il link per il download.

```
//scrive il form con il pulsante per proseguire con il prossimo passo, altrimenti visualizza lo zip

if (count($arr_lingue_selezionate) > 1)
{
	//costruisce il form per il passo successivo
	$sel_lingua = implode("|", array_slice($arr_lingue_selezionate, 1));
	$prossima_lingua = $arr_lingue_selezionate[1];

	echo "<div style='color:#000;border:1px dotted #48C;text-
align:center;width:300px;margin:10px;padding:10px;background-color:#eee;'>" .
	"Prosegui con l'esportazione<br>in $arr_lingue[$prossima_lingua]<br>" .
	"<form name='form1' action='esporta_xml.php' method='post'>" .
	"<input type='hidden' name='sigla_museo' value='$sigla_museo'>" .
	"<input type='hidden' name='sel_lingua' value='$sel_lingua'>" .
	"<input type='hidden' name='pass' value='$parola_chiave'>" .
	"<input type='hidden' name='azione_specifica_passo' value=''>" .
	"<input type='submit' name='submit1' value='PROSEGUI' style='cursor:hand;margin-top:10px;'>" .
	"</form></div>";
}
else
{
	echo "<div style='color:#0a5;border:1px dotted #0a5;text-
align:center;width:300px;margin:10px;padding:10px;background-color:#eee;'><strong>L'esportazione XML<br>è
terminata!</strong>";
	require ("crea_archivio_zip.php");
	echo "</div>";
}
?>
```

Visualizza i report del passo di esportazione appena concluso e chiude la connessione al database.

```
<p>Di seguito è riportato il resoconto dell'esportazione in <?php echo "$arr_lingue[$id_lingua]" ?>:</p>
<table>
<tr><th>Report</th><th>XML</th></tr>
<tr>
<td><div style="font-size:12px;width:500px;padding:5px;background-color:#eee;"><?php echo $testo_report ?
></div></td>
<td><textarea rows="20" cols="80" style="font-size:12px;" readonly><?php echo $testo_xml ?
></textarea></td>
</tr>
</table>
</center>
<br><br><br>
```

```
<?php
pg_close($connessione);
require 'pie_pagina.php';
?>
```

6.3.3 - La compilazione dei tag image: scrivi_tag_image.php

Il file *scrivi_tag_image.php* contiene un breve listato che gestisce i passi per creare il tag **<image>** il quale contiene tutto ciò che riguarda un esemplare (reperto) cioè descrizioni testuali ed elementi multimediali allegati. Il file *scrivi_tag_image.php* viene incluso nel codice di *esporta_xml.php* sia nel ciclo che considera ciascun codice **dba_idinterno** per l'elenco delle vetrine senza oggetti, sia nel ciclo in cui per ogni aggregazione vengono trovati i **dba_idinterno** degli esemplari contenuti nell'aggregazione. Il tag <image> è l'elemento più importante dell'xml di esportazione in quanto contiene le informazioni dell'esemplare.

La compilazione degli attributi di <image> è stata delegata a vari file di include in modo da rendere chiara la struttura dell'applicazione. In particolare, per quanto riguarda la ricerca e l'esportazione dei file multimediali abbinati all'esemplare è stato creato il file *trova_allegati_mm.php*; per la compilazione dei campi testuali descrittivi viene utilizzata una inclusione dinamica che utilizza il file specifico dedicato al tipo di museo che si sta considerando, cioè rispettivamente per mineralogia, orto botanico, paleontologia e fisica sono stati creati i file *mineralogia.php, orto_botanico.php, paleontologia.php e fisica.php*.

Come si vede dal listato del file il programma include il file dedicato agli oggetti multimediali.

```
 <?php
 /*
     Trova le informazioni relative ad un esemplare cioè ad un reperto
     con codice identificativo dba_idinterno
     Scrive il nodo <image imagename="" infotext="" infotext2="" infotext3="" audio="" video="" img="" />
completo di tutti i parametri
 */
  //trova i file multimediali abbinati
  require("trova_allegati_mm.php");
  $infotext="";
  $infotext2="";
  $infotext3="";
```

Successivamente viene eseguita una query per trovare il testo della descrizione che andrà a collocarsi nell'attributo **infotext2** assieme ad altre descrizioni. Da notare che viene creata una query che poi viene passata alla funzione traduci() che ho descritto nella sezione dedicata al file

esporta_xml.php. La tabella interessata è **palm_esemplare_ha_descrizione_estesa** dedicata a contenere la descrizione estesa dell'esemplare. E' costituita dal campo **dba_idinterno** per la crrispondenza con il codice esemplare, dai campi **id_lingua** e **descrizione**. La funzione traduci restituirà la descrizione nella lingua indicata da id_lingua e, nel caso non ci fosse il record corrispondente alla lingua, troverà la descrizione nella lingua di default (ad esempio italiano) che è la variabile$lingua_default inizializzata in *config_lingue.php*. Viene poi utilizzata la funzione trova_etichetta() che in questo caso restituirà "Descrizione " o se la lingua è inglese "Description".

```
//trova la descrizione estesa per infotext2 (questo è l'unico campo descrittivo in comune per i vari tipi di museo)
  $query_base = "select * from palm_esemplare_ha_descrizione_estesa where (dba_idinterno=$dba_idinterno)";
  $descrizione=traduci($query_base,"descrizione");
  if ($descrizione!=NULL) $infotext2 .= trova_etichetta("palm_esemplare_ha_descrizione_estesa", "descrizione",
$id_lingua) . ": " . $descrizione . $accapo;
```

Successivamente c'è l'inclusione del file php specifico per il tipo di museo.

```
  //include il file che analizza le tabelle specifiche relative al tipo museo rilevato
  if ($tipo_museo != NULL) require($tipo_museo.".php");
```

Alla fine viene aggiunto il nodo <image> al testo XML.

```
  //scrivo nodo image con parametri
  $testo_xml .= "<image imagename=\"$imagename\" infotext=\"" . converti_caratteri_speciali($infotext) . "\"
infotext2=\"" . converti_caratteri_speciali($infotext2) . "\" infotext3=\"" . converti_caratteri_speciali($infotext3) .
"\" audio=\"$audio\" video=\"$video\" img=\"$img\"/>\n";

?>
```

6.3.4 - Le funzioni per le scritture su filesystem e sul database: funzioni_fs_db.php

Iniziamo con la prima funzione dedicata all'inizializzazione delle cartelle del filesystem dedicate a memorizzare i file dell'esportazione. La struttura di sottocartelle rispecchia le "Specifiche dei file richiesti dall'applicazione Flash". Da notare che le cartelle risiedono in una directory chiamata con il codice identificativo del museo *sigla_museo* a partire dalla directory "/esportazioni_xml/" dentro la cartella riservata al programma *EsPDAmuseo*.

Le esportazioni avranno quindi questa strutture a cartelle:

```
... path di EsPDAmuseo
                    |.. files del programma
                    |____esportazioni_xml
                                |____sigla museo 1
                                |              |____admin
                                |              |____images
```

```
|                   |____video
|                   |....
|____sigla museo 2
|                   |____admin
|                   |____images
|                   |____video
|....               |....
```

La seguente funzione richiede in input il codice museo ($sigla_museo) e viene chiamata all'inizio della procedura di esportazione dalla pagina *esporta_xml.php*:

```
<?php
//predispone le directory e cancella i file gia presenti per il museo selezionato
function inizializza_directory_museo($sigla_museo)
{
	global $testo_report, $arr_lingue;
	$arr_dir = NULL;
	$arr_dir[] = "esportazioni_xml";
	$arr_dir[] = "esportazioni_xml/$sigla_museo";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/admin";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/audio";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/images";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/images/gallery";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/images/gallery/largeimages";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/images/gallery/otherimages";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/images/gallery/thumbs";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/video";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/video/thumbs";
	$arr_dir[] = "esportazioni_xml/$sigla_museo/video/video";
	foreach ($arr_lingue as $k => $v)
	{
		$arr_dir[] = "esportazioni_xml/$sigla_museo/admin/" . $k;
		$arr_dir[] = "esportazioni_xml/$sigla_museo/audio/" . $k;
		$arr_dir[] = "esportazioni_xml/$sigla_museo/video/video/" . $k;
	}

	foreach ($arr_dir as $cartella)
	{	//se la cartella non esiste allora viene creata
		if (file_exists($cartella) != TRUE)
		{
			mkdir("$cartella", 0700);
		}
		else
		{
			//cancella i file gia presenti nella cartella
			//glob() trova i file in base ad un criterio di ricerca
			if (glob("$cartella/*.*") != false)
```

```
                        {
                                foreach (glob("$cartella/*.*") as $filename)
                                {
                                        unlink ("$filename");    //cancella il file
                                 }
                        }
                }
        }
        $testo_report  .= "ok - creata struttura sottocartelle da <a href='esportazioni_xml/$sigla_museo'
target='_blank'>esportazioni_xml/$sigla_museo</a><br>ok - cancellati tutti i file nelle sottocartelle da <a
href='esportazioni_xml/$sigla_museo' target='_blank'>esportazioni_xml/$sigla_museo</a><br>";

        cancella_esportazioni_su_db($sigla_museo);
}
```

La successiva funzione cancella dal database l'eventuale precedente esportazione xml relativa al museo selezionato.

Per quanto riguarda la memorizzazione delle esportazioni sul database devo premettere la spiegazione delle tabelle del database dedicate a questo scopo.
Tra le potenzialità di esportazione richieste all'applicazione *EsPDAmuseo* era stata richiesta anche la funzionalità di poter memorizzare l'esportazione xml oltre che sul filesystem, anche sul database.

A questo scopo ho creato altre due tabelle sul database:

1. tabella: **xml_esportazioni**
   questa tabella memorizza il testo dei file xml generati dall'esportazione cioè i file gallery.xml, i file slideshow.xml e altri eventuali file xml. Ogni record contiene il codice lingua, il codice del museo, percorso del file e il testo del contenuto xml. Il percorso del file parte dalla directory base in cui ci sono admin/, images/ etc...
   Di seguito riporto le istruzioni SQL:
   ```
   CREATE TABLE xml_esportazioni
   (
    id_lingua varchar(255),
    sigla_museo varchar(255) NOT NULL,
    percorso_file_xml varchar(255) NOT NULL,
    testo_xml text NOT NULL
   )
   ```
2. tabella: **xml_esportazioni_mm**
   memorizza la path dei file multimediali esportati e i codici id del loro contenuto che è già

presente sul database.

```
CREATE TABLE xml_esportazioni_mm
(
 sigla_museo varchar(255) NOT NULL,
 percorso_file_mm varchar(255) NOT NULL,
 id_oggetto_mm_allegato int8 NOT NULL
)
```

Avendo questi dati memorizzati sul database si può facilmente ricostruire il contenuto del filesystem.

Dopo questa introduzione procediamo con la seguente funzione dedicata invece a cancellare le esportazioni memorizzate sul database. Questa funzione richiede in input il codice del museo. La funzione cancella tutti irecord relativi al museo considerato dalle tabelle xml_esportazioni e xml_esportazioni_mm.

```
function cancella_esportazioni_su_db($sigla_museo)
{
	global $connessione, $testo_report;
	//pg_delete() vuole le condizioni di cancellazione memorizzate in un array
	$arr_condizioni["sigla_museo"] = $sigla_museo;
	$ris_cancella = pg_delete($connessione, "xml_esportazioni_mm", $arr_condizioni);
	if ($ris_cancella)
		$testo_report .= "";
	else
		$testo_report .= ":: errore di scrittura su database<br>";
	$ris_cancella = pg_delete($connessione, "xml_esportazioni", $arr_condizioni);
	if ($ris_cancella)
		$testo_report .= "ok - cancellate esportazioni precedenti sul database<br>";
	else
		$testo_report .= ":: errore di scrittura su database<br>";

}
```

La seguente funzione esporta il file multimediale che è memorizzato nel database come Binary Large Object. La funzione riceve in input l'id del record che contiene l'oggetto e prende il riferimento al contenuto binario. La tabella **palm_oggetto_mm_blob** contiene i campi id_oggetto_mm_allegato che è il codice univoco di un oggetto multimediale, e il campo **blobbyte** che contiene il riferimento all'oggetto binario memorizzato sul database. Il contenuto del file binario viene estrapolato con l'utilizzo della funzione pg_lo_export() della libreria dedicata a PostgreSQL di PHP. Questa funzione accetta in input il valore di riferimento all'oggetto e il percorso assoluto dove memorizzare il file estrapolato. Tutto ciò deve avvenire all'interno di una

transazione. La seguente funzione controlla anche se il file non sia già presente sul filesystem così evita tempi inutili.

```
 //esporta il campo Binary Large Object del database su file
function scrivi_file_blob($id_palm_oggetto_mm_blob, $percorsofile, &$connessione)
{
  global $testo_report;
  $query = "select * from palm_oggetto_mm_blob where id_oggetto_mm_allegato=$id_palm_oggetto_mm_blob;";
  $risultato_blob = pg_query($query);
  while($riga_blob = pg_fetch_object($risultato_blob))
  {
        //evita di riscrivere il file, le cartelle sono vuote all'inizio di ogni esportazione di un museo
        if (file_exists($percorsofile) != true)

        {
                $real_path = realpath("");
                pg_query ($connessione, "begin");  //inizio della transazione
                $numero=$riga_blob->blobbyte;
                pg_lo_export ($numero, $real_path."/".$percorsofile); //esportazione del Binary Large Object
                pg_query ($connessione, "commit");  //fine della transazione
                $testo_report  .= "ok - <a href='$percorsofile' target='_blank'>$percorsofile</a><br>";
                scrivi_percorso_mm_su_db($percorsofile);
        }
        else
        {
                $testo_report  .= "ok - <a href='$percorsofile' target='_blank'>$percorsofile</a> non
sovrascritto<br>";
        }
  }
}
```

La seguente funzione scrive i file di elenco, ad esempio slideshow.xml che contiene un elenco di iimagini. La funzione è predisposta ad accettare anche elenchi di file audio o video in quanto c'è il parametro in input $tipo che permette una eventuale distinzione delle intestazioni dei file di elenco. In input vengono passati i valori della path del file xml elenco da memorizzare, il tipo di elenco, e il vettore contenente le path dei file che sono contenuti nell'elenco. La funzione alla fine chiama un'altra procedura adibita al salvataggio del file di elenco nel database.

```
//scrive file xml slideshow.xml (è predisposto anche per eventuale audio.xml, video.xml)
function scrivi_file_elenco($percorsofile,$tipo,$elenco)
{
        global $testo_report;
        $handle = fopen  ( $percorsofile, "w");
```

```
	if ($handle != FALSE)
	{
		$testo = "";
		//scrive l'intestazione in base al tipo di file elenco
		if ($tipo == "slideshow") $testo="<!--  \n'timer' :: number of seconds between each image transition\n'order' :: how you want your images displayed. choose either 'sequential' or 'random'\n'looping' :: if the slide show is in sequential mode, this stops the show at the last image (use 'yes' for looping, 'no' for not) \n'fadeTime' :: velocity of image crossfade. Increment for faster fades, decrement for slower. Approximately equal to seconds.\n'xpos' :: _x position of all loaded clips (0 is default)\n'ypos' :: _y position of all loaded clips (0 is default)\n-->\n<gallery timer=\"5\" order=\"sequential\" fadetime=\"2\" looping=\"yes\" xpos=\"0\" ypos=\"0\">\n";

		for ($cont=0; $cont<count($elenco); $cont++)
		{
			$testo .= "<image path=\"$elenco[$cont]\" />\n";
		}
		$testo .= "</gallery>";
		$testo = str_replace("\n", chr(13).chr(10), $testo);
		if (!fwrite($handle, $testo))
		{
			echo "Non si riesce a scrivere nel file ($percorsofile)";
			exit;
		}
		fclose($handle);
		$testo_report  .= "ok - <a href='$percorsofile' target='_blank'>$percorsofile</a> XML<br>";
		scrivi_xml_su_db($percorsofile,$testo);
	}
}
```

La seguente funzione crea il file gallery.xml. In input accetta la path in cui deve essere memorizzato il file xml ed il testo effettivo dell'xml passato per riferimento visto che sarà destinato ad occupare un rilevante spazio di memoria. In più c'è la chiamata alla funzione dedicata a memorizzare l'XML sul database.

```
//scrive il file gallery.xml
function scrivi_file_gallery($percorsofile,&$testo)
{
	global $testo_report;
	$handle = fopen  ( $percorsofile, "w");
	if ($handle != FALSE)
	{
		if (!fwrite($handle, str_replace("\n", chr(13).chr(10), $testo)))
		{
			echo "Non si riesce a scrivere nel file ($percorsofile)";
```

```
                        exit;
                }
                fclose($handle);
                $testo_report  .= "ok - <a href='$percorsofile' target='_blank'>$percorsofile</a> XML<br>";
        }
        scrivi_xml_su_db($percorsofile, $testo);
}
```

Ed ecco le funzioni per la memorizzazione dell'esportazione sul database.

La seguente funzione memorizza sul database i file xml gioè gallery.xml e i file slideshow.xml e comunque è predisposta per memorizzare altri file xml. La funzione riceve in input la path del file e il contenuto passato per riferimento. Da notare che non viene utilizzata una query di INSERT ma la funzione pg_insert() della libreria dedicata a PostgreSQL di PHP. Tale funzione riceve in input la connessione, la tabella interessata e un'array di chiavi-valori corrispondente a campo-valore.

```
//memorizza i file xml sul database
function scrivi_xml_su_db($percorsofile,&$testo_xml)
{
        global $testo_report, $connessione, $sigla_museo, $id_lingua;
        //memorizza l'xml sul database
        $arr_record["sigla_museo"] = $sigla_museo;
        $arr_record["id_lingua"] = $id_lingua;
        $arr_record["percorso_file_xml"] = $percorsofile;
        $arr_record["testo_xml"] = $testo_xml;
        $ris_insert = pg_insert($connessione, "xml_esportazioni", $arr_record);
        if ($ris_insert)
                $testo_report .= "ok - XML memorizzato su database<br>";
        else
                $testo_report .= ":: errore di scrittura xml su database<br>";
}
```

La seguente funzione salva sul database i file multimediali o meglio il loro percorso e il riferimento al Binary Large Object memorizzato già sul database e riferito appunto al file multimediale.

```
//memorizza i percorsi dei file mm sul database
function scrivi_percorso_mm_su_db($percorsofile)
{
        global $testo_report, $connessione, $sigla_museo;
        $arr_record["sigla_museo"] = $sigla_museo;
        $arr_record["percorso_file_mm"] = $percorsofile;
        $ris_insert = pg_insert($connessione, "xml_esportazioni_mm", $arr_record);
        if ($ris_insert)
```

```
                $testo_report .= "ok - memorizzato file multimediale su database<br>";
        else
                $testo_report .= ":: errore di scrittura su database<br>";
}
```

6.3.5 - La ricerca degli allegati multimediali: trova_allegati_mm.php

Le tabelle coinvolte nell'abbinamento dei file multimediali agli esemplari sono le seguenti (tralascio i campi non necessari al nostro contesto e i vincoli di integrità):

1. tabella: **palm_esemplare_ha_oggetto_mm**
   campi: **id_oggetto_mm**, **id_tipo_informazione**, **dba_idinterno**
   descrizione: associa ai codici dba_idinterno i codici degli allegati multimediali (id_oggetto_mm), il campo id_tipo_informazione verrà spiegato dopo.
2. tabella: **palm_oggetto_mm_allegato**
   campi: **id_oggetto_mm, estensione, id_lingua, id_tipo_oggetto_mm**
   descrizione: per ogni file multimediale memorizza l'estensione, il nome la lingua di riferimento, il mime type.
3. tabella: **palm_oggetto_mm_blob**
   campi: **id_oggetto_mm_allegato, blobbyte**
   descrizione: associa ad ogni id_oggetto_mm_allegato il riferimento al Binary Large Object cioè al contenuto del file multimediale.

I campi **id_tipo_informazione** e **id_tipo_oggetto_mm** servono a definire il tipo di file multimediale cioè:

Tipo di file multimediale	id_tipo_informazione	id_tipo_oggetto_mm
Immagine miniatura		2
Immagine grande	1	1
Immagini slideshow	2	1
File audio		3
File video		4

Di seguito riporto il codice dedicato agli allegati multimediali.

```
<?php
//query_base_oggetto_mm è una stringa che si ripete per tutte le ricerche di allegato multimediale

$query_base_oggetto_mm="select palm_oggetto_mm_blob.id_oggetto_mm_allegato,
```

```
palm_oggetto_mm_allegato.estensione, palm_oggetto_mm_allegato.id_lingua from
palm_esemplare_ha_oggetto_mm, palm_oggetto_mm_allegato, palm_oggetto_mm_blob where
(palm_esemplare_ha_oggetto_mm.dba_idinterno=$dba_idinterno) and
(palm_esemplare_ha_oggetto_mm.da_visualizzare=TRUE) and
(palm_esemplare_ha_oggetto_mm.id_oggetto_mm=palm_oggetto_mm_allegato.id_oggetto_mm) and
(palm_oggetto_mm_allegato.id_oggetto_mm=palm_oggetto_mm_blob.id_oggetto_mm_allegato)";


//trova l'immagine miniatura (thumbnail)
$imagename="";
$query = $query_base_oggetto_mm . " and (id_tipo_oggetto_mm=2);";
$risultato_blob = pg_query($query);
while($riga_blob = pg_fetch_object($risultato_blob))
{
        $imagename .= "$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";
        scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/images/gallery/thumbs/$imagename", $connessione);
}
```

la funzione scrivi_file_blob() è stata vista prima nell'analisi del file *funzioni_fs_db.php* e praticamente esporta l'allegato multimediale sul filesystem e ne memorizza il percorso sul database.

```
//trova l'immagine grande
$query = $query_base_oggetto_mm . " and (id_tipo_informazione=1) and (id_tipo_oggetto_mm=1);";
$risultato_blob = pg_query($query);
while($riga_blob = pg_fetch_object($risultato_blob))
{
        if ($imagename == "")  $imagename .= "$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";
        scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/images/gallery/largeimages/$imagename", $connessione);
}
```

di seguito è riportato il codice per gestire l'esportazione degli eventiali *slideshow* in cui oltre ad esportare il file multimediale bisogna creare, in caso di più file trovati, un opportuno file xml con l'elenco degli allegati di tipo immagine. Vedi la sezione “Specifiche dei file richiesti dall'applicazione Flash” per chiarimenti sul formato dei file *slideshow*.

```
//trova immagini (se ci sono più oggetti_mm viene creato il file elenco slideshow.xml)
$img="";
$query = $query_base_oggetto_mm . " and (id_tipo_informazione=2) and (id_tipo_oggetto_mm=1);";
$risultato_blob = pg_query($query);
$arr_elenco = null;
while($riga_blob = pg_fetch_object($risultato_blob))
{
        $img = "$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";

        $arr_elenco[]="images/gallery/otherimages/$img";   //la path parte da images perchè il programma in
        Flash risiede in "esportazioni_xml/$sigla_museo/"
        scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/images/gallery/otherimages/$img",$connessione);
}
if (count($arr_elenco)>1)
{
        $contatore_elenco_slideshow += 1;
        $nome_file_elenco="slideshow_" . $id_lingua . "_" . $contatore_elenco_slideshow . ".xml";
        scrivi_file_elenco("esportazioni_xml/$sigla_museo/admin/$id_lingua/$nome_file_elenco", "slideshow",
        $arr_elenco);
        $img = $nome_file_elenco;
}
```

di seguito il codice per trovare audio e video:

```
//trova audio
$audio="";
$query = $query_base_oggetto_mm . "  and (id_tipo_oggetto_mm=3);";
$risultato_blob = pg_query($query);
while($riga_blob = pg_fetch_object($risultato_blob))
{
        //se la lingua è NULL vuol dire che l'oggetto è multilingua altrimenti si prende la lingua corrispondente
        if ($riga_blob->id_lingua == NULL)
        {
                $audio .= "audio$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";
                scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/audio/$audio", $connessione);
        }
        elseif ($riga_blob->id_lingua == $id_lingua)
        {
                $audio .= "audio$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";
                scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/audio/$id_lingua/$audio", $connessione);
        }
}
```

```
 //trova video

 $video="";
 $query = $query_base_oggetto_mm . "  and (id_tipo_oggetto_mm=4);";
 $risultato_blob = pg_query($query);
 while($riga_blob = pg_fetch_object($risultato_blob))
 {
        //se la lingua è NULL vuol dire che l'oggetto è multilingua altrimenti si prende la lingua corrispondente
        if ($riga_blob->id_lingua == NULL)
        {
                $video .= "video$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";
                scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/video/$video", $connessione);
        }
        elseif ($riga_blob->id_lingua == $id_lingua)
        {
                $video .= "video$riga_blob->id_oggetto_mm_allegato.$riga_blob->estensione";
                scrivi_file_blob($riga_blob->id_oggetto_mm_allegato,
        "esportazioni_xml/$sigla_museo/video/$id_lingua/$video", $connessione);
        }
}
?>
```

6.3.6 - La parte dedicata a mineralogia: mineralogia.php

In questa sezione verrà analizzato il procedimento per l'esportazione delle informazioni dedicate a mineralogia.

Di seguito sono riportate le tabelle coinvolte nell'esportazione (espongo solo le informazioni strettamente necessarie al nostro contesto):

1. tabella: **min_scheda_esemplare**
   campi: **dba_idinterno, id_nome_minerale, dimensione_x, dimensione_y, dimensione_z, id_unita_misura_dimensione, peso, id_unita_misura_peso, valore**
   descrizione: la tabella associa al codice dba_idinterno (l'esemplare) alcuni campi descrittivi e codici di riferimento per altre descrizioni (id_nome_minerale, id_unita_misura ...).
2. tabella: **min_al_minerale_ita_eng**
   campi: **id_minerale** chiave primaria**, nome_minerale_ita, nome_minerale_eng**
   descrizione: memorizza il nome del minerale in italiano e inglese.

La parte di programma che utilizza queste due tabelle:

```
<?php

if ($tipo_museo == "mineralogia")
{
	//compila infotext
	$query = "select * from min_scheda_esemplare, min_al_minerale_ita_eng where
		(dba_idinterno=$dba_idinterno) and
		(min_scheda_esemplare.id_nome_minerale=min_al_minerale_ita_eng.id_minerale);";
	$risultato_min = pg_query($query);
	while($riga_min = pg_fetch_object($risultato_min))
	{
		if (strtoupper($id_lingua)=="ITA") $infotext=$riga_min->nome_minerale_ita;
		else $infotext=$riga_min->nome_minerale_eng;
	}
```

Altra tabella:

3. tabella: **min_ha_descrizione_sintetica**
   campi: **dba_idinterno, id_lingua, minerali_associati, descrizione_esemplare_sintetica**
   descrizione: associa al codice dba_idinterno due campi descrittivi

```
	//compila infotext2
	$query = "select * from min_ha_descrizione_sintetica where (dba_idinterno=$dba_idinterno) and
		(id_lingua='$id_lingua');";
	$risultato_min = pg_query($query);
	while($riga_min = pg_fetch_object($risultato_min))
	{
		//Descrizione sintetica
		$infotext2 .= trova_etichetta("min_ha_descrizione_sintetica", "descrizione_esemplare_sintetica",
			$id_lingua) . ": " . $riga_min->descrizione_esemplare_sintetica . $accapo;
		//Minerali associati
		$infotext2 .= trova_etichetta("min_ha_descrizione_sintetica", "minerali_associati", $id_lingua) . ":
			" . $riga_min->minerali_associati . $accapo;
	}
```

4. tabella: **min_l_unita_misura_dim**
   campi: **id_unita_misura_dim** chiave primaria, **unita_misura_dim**
   descrizione: memorizza le unità di misura delle dimensioni per i valori presenti in min_scheda_esemplare

```
$query = "select * from min_scheda_esemplare, min_l_unita_misura_dim where (dba_idinterno=$dba_idinterno)
and (min_scheda_esemplare.id_unita_misura_dimensione=min_l_unita_misura_dim.id_unita_misura_dim);";
        $risultato_min = pg_query($query);

        while($riga_min = pg_fetch_object($risultato_min))
        {
                $unita_misura_dim = $riga_min->descrizione;
                //Larghezza
                $infotext2 .= trova_etichetta("min_scheda_esemplare", "dimensione_x", $id_lingua) . ": " .
                                $riga_min->dimensione_x . " " . $unita_misura_dim . $accapo;
                //Altezza
                $infotext2 .= trova_etichetta("min_scheda_esemplare", "dimensione_y", $id_lingua) . ": " .
                                $riga_min->dimensione_y . " " . $unita_misura_dim . $accapo;

                //Profondita
                $infotext2 .= trova_etichetta("min_scheda_esemplare", "dimensione_z", $id_lingua) . ": " .
                                $riga_min->dimensione_z . " " . $unita_misura_dim . $accapo;
        }
```

5. tabella: **min_l_unita_misura_peso**
   campi: **id_unita_misura_peso** chiave primaria, **descrizione**
   descrizione: memorizza le unità di misura peso del campo di min_scheda_esemplare.

```
        $query = "select * from min_scheda_esemplare, min_l_unita_misura_peso where
        (dba_idinterno=$dba_idinterno) and
        (min_scheda_esemplare.id_unita_misura_peso=min_l_unita_misura_peso.id_unita_misura_peso);";

        $risultato_min = pg_query($query);
        while($riga_min = pg_fetch_object($risultato_min))
        {
                //Peso
                $infotext2 .= trova_etichetta("min_scheda_esemplare", "peso", $id_lingua) . ": " .
                                $riga_min->peso . " " . $riga_min->descrizione . $accapo;
                //Valore
                $infotext2 .= trova_etichetta("min_scheda_esemplare", "valore", $id_lingua) . ": " .
                                $riga_min->valore . " Euro" . $accapo;
        }
```

6. tabella: **min_provenienza_esemplare**
   campi: **dba_idinterno, id_nazione, id_localita, id_provincia, id_regione, id_comune, id_tipologia_località**
   descrizione: associa all'esemplare i vari codici delle provenienze

7. tabella: **min_l_nazione**
   campi: **id_nazione** chiave primaria, **denominazione**

8. tabella: **min_l_regione**
   campi: **id_regione** chiave primaria, **regione**

9. tabella: **min_l_provincia**
   campi: **id_provincia** chiave primaria, **provincia**

10. tabella: **min_l_comune**
    campi: **id_comune** chiave primaria, **denominazione**

11. tabella: **min_l_localita**
    campi: **id_localita** chiave primaria, **denominazione**

12. tabella: **min_al_tipologia_localita**
    campi: **id_tipo_localita** chiave primaria, **denominazione**

Avendo le tabelle relative alla provenienza una struttura simile ho automatizzato la sequenza di query con 4 array.

```
//provenienza
$arr_prov_tabelle = array("min_l_localita", "min_l_comune", "min_l_provincia", "min_l_regione",
                          "min_l_nazione", "min_al_tipologia_localita");
$arr_prov_tabelle_id = array("id_localita", "id_comune", "id_provincia", "id_regione", "id_nazione",
                          "id_tipo_localita");
$arr_prov_tabelle_campi = array("denominazione", "denominazione", "provincia", "regione",
                          "denominazione", "denominazione");
$arr_prov_tabelle_etichette = array("Localita", "Comune", "Provincia", "Regione", "Nazione", "Tipologia
                          localita");
$arr_prov_id = array("id_localita", "id_comune", "id_provincia", "id_regione", "id_nazione",
                          "id_tipologia_localita");

for ($cont=0; $cont<count($arr_prov_tabelle); $cont++)
{
        $query = "select * from min_provenienza_esemplare, $arr_prov_tabelle[$cont] where
                (dba_idinterno=$dba_idinterno) and (min_provenienza_esemplare.$arr_prov_id[$cont]
                        =$arr_prov_tabelle[$cont].$arr_prov_tabelle_id[$cont]);";
        $risultato_min = pg_query($query);
        while($riga_min = pg_fetch_object($risultato_min))
        {
                $infotext2 .= trova_etichetta($arr_prov_tabelle[$cont], $arr_prov_tabelle_campi[$cont],
                        $id_lingua) . ": " . $riga_min->$arr_prov_tabelle_campi[$cont] . $accapo;
        }
}
```

13.tabella: **min_proviene_da_collezione**

campi: **dba_idinterno**, **id_collezione_provenienza**

descrizione: per l'esemplare memorizza il codice di collezione di provenienza

14.tabella: **min_collezione_provenienza**

campi: **id_collezione_provenienza** chiave primaria, **nome_provenienza**

```
        //collezione di provenienza
$query = "select * from min_proviene_da_collezione, min_collezione_provenienza where
(dba_idinterno=$dba_idinterno) and
(min_proviene_da_collezione.id_collezione_provenienza=min_collezione_provenienza.id_collezione_provenienza);";

        $risultato_min = pg_query($query);
        while($riga_min = pg_fetch_object($risultato_min))
        {
                //Collezione di provenienza
                $infotext2 .= trova_etichetta("min_collezione_provenienza", "nome_provenienza", $id_lingua) . ": "
                                . $riga_min->nome_provenienza . $accapo;
        }
```

15.tabella: **min_scheda_esemplare_ha_pubblicazione**

campi: **dba_idinterno, codice_pubblicazione**

descrizione: associa i codici pubblicazione agli esemplari

16.tabella: min_pubblicazione

campi: **codice** chiave primaria**, pubblicazione**

```
        //compila infotext3
        //bibliografia
$query = "select * from min_scheda_esemplare_ha_pubblicazione, min_pubblicazione where
(dba_idinterno=$dba_idinterno) and
(min_scheda_esemplare_ha_pubblicazione.codice_pubblicazione=min_pubblicazione.codice);";

        $risultato_min = pg_query($query);
        $bibliografia = "" ;
        while($riga_min = pg_fetch_object($risultato_min))
        {
                $bibliografia .= "(" . $riga_min->pubblicazione . ") ";
        }
        //Bibliografia
        if ($bibliografia != "") $infotext3 .= trova_etichetta("min_pubblicazione", "codice", $id_lingua). ": " .
                                        $bibliografia . $accapo;
}
?>
```

6.3.7 - La parte dedicata ad orto botanico: orto_botanico.php

In questa sezione verrà analizzato il procedimento per l'esportazione delle informazioni dedicate ad orto botanico.

Di seguito sono riportate le tabelle coinvolte nell'esportazione (espongo solo le informazioni strettamente necessarie al nostro contesto):

1. tabella: **obt_scheda_esemplare**
   campi: **dba_idinterno, id_nome_pianta**
   descrizione: associa all'esemplare il codice del nome della pianta

2. tabella: **obt_al_nome_pianta_lat**
   campi: **id** chiave primaria**, nome_pianta_lat**
   descrizione: memorizza il nome latino delle piante.

```
if ($tipo_museo == "orto_botanico")
{
        //compila infotext e infotext2
$query = "select * from obt_scheda_esemplare, obt_al_nome_pianta_lat where (dba_idinterno=$dba_idinterno)
and (obt_scheda_esemplare.id_nome_pianta=obt_al_nome_pianta_lat.id);";
        $risultato_obt = pg_query($query);
        $id_famiglia=NULL;
        $id_nome_pianta_latino=NULL;
        while($riga_obt = pg_fetch_object($risultato_obt))
        {
                 $infotext2 .= trova_etichetta("obt_al_nome_pianta_lat", "nome_pianta_lat", $id_lingua) . ": " .
                                  $riga_obt->nome_pianta_lat . $accapo;
                $id_famiglia=$riga_obt->id_famiglia;
                $id_nome_pianta_latino=$riga_obt->id;
        }
```

3. tabella: **obt_al_nome_pianta_ha_descrizione**
   campi: **id_lingua, id_nome_pianta_lat, nome_pianta**
   descrizione: associa al nome latino una descrizione multilingua

```
if ($id_nome_pianta_latino != NULL)
{
        $query_base = "select * from obt_al_nome_pianta_ha_descrizione where
                        (id_nome_pianta_lat=$id_nome_pianta_latino)";
        $nome_pianta=traduci($query_base,"nome_pianta");
        if ($nome_pianta!=NULL)
                $infotext .= trova_etichetta("obt_al_nome_pianta_ha_descrizione", "nome_pianta",
                        $id_lingua) . ": " . $nome_pianta;
}
```

4. tabella: **obt_al_nome_famiglia_ha_descrizione**
   campi: **id_lingua, id_nome_famiglia, id_nome_famiglia_lat, nome_famiglia**
   descrizione: associa al codice id_nome_famiglia della tabella obt_al_nome_pianta_lat il nome multilingua della famiglia dell'esemplare.

```
if ($id_famiglia != NULL)
{
        $query_base = "select * from obt_al_nome_famiglia_ha_descrizione where
                        (id_nome_famiglia=$id_famiglia)";
        $nome_famiglia=traduci($query_base,"nome_famiglia");
        if ($nome_famiglia!=NULL)
                $infotext2 .= trova_etichetta("obt_al_nome_famiglia_ha_descrizione", "nome_famiglia",
                                $id_lingua) . ": " . $nome_famiglia . $accapo;
}
```

5. tabella: **obt_appartiene_a_collezione**
   campi: **dba_id_interno, id_tipo_collezione**
   descrizine: associa tipo collezione all'esemplare

6. tabella: **obt_tipo_collezione_ha_descrizione**
   campi: **id_tipo_collezione** chiave primaria, **nome, descrizione, id_lingua**
   descrizione: contiene i tipi di collezione con la descrizione multilingua.

```
$query_base = "select * from obt_tipo_collezione_ha_descrizione, obt_appartiene_a_collezione where
(obt_appartiene_a_collezione.dba_idinterno=$dba_idinterno) and
(obt_tipo_collezione_ha_descrizione.id_tipo_collezione=obt_appartiene_a_collezione.id_tipo_collezione)";

$tipo_collezione=traduci($query_base,"nome");
if ($tipo_collezione!=NULL)
        $infotext2 .= trova_etichetta("obt_tipo_collezione_ha_descrizione", "nome", $id_lingua) . ": " .
                        $tipo_collezione . $accapo;
```

7. tabella: **obt_ha_descrizione**
   campi: **dba_idinterno, id_lingua, posizione_nel_orto, uogo_di_origine, habitat,**

caratteristiche_botaniche, utilizzo_usi, approfondimenti

descrizione: associa all'esemplare vari campi descrittivi

```
        //altre caratteristiche
        $arr_descrizione_campi = array("posizione_nel_orto", "luogo_di_origine", "habitat",
                                "caratteristiche_botaniche", "utilizzo_usi", "approfondimenti");
        foreach($arr_descrizione_campi as $campo)
        {
                $query_base = "select * from obt_ha_descrizione where (dba_idinterno=$dba_idinterno)";
                $desc=traduci($query_base,"$campo");
                if ($desc!=NULL) $infotext2 .= trova_etichetta("obt_ha_descrizione", "$campo", $id_lingua) . ": " .
                                $desc . $accapo;

        }
```

8. tabella: **obt_scheda_esemplare_ha_pubblicazione**

   campi: **dba_idinterno, codice_pubblicazione**

   descrizione: associa i codici delle pubblicazioni agli esemplari

9. tabella: **obt_pubblicazione**

   campi: **codice** chiave primaria, **pubblicazione, titolo**

   descrizione: contiene le descrizioni delle pubblicazioni.

```
        //compila infotext3
        //bibliografia
$query = "select * from obt_scheda_esemplare_ha_pubblicazione, obt_pubblicazione where
(dba_idinterno=$dba_idinterno) and
(obt_scheda_esemplare_ha_pubblicazione.codice_pubblicazione=obt_pubblicazione.codice);";

        $risultato_obt = pg_query($query);
        $bibliografia = "" ;
        while($riga_obt = pg_fetch_object($risultato_obt))
        {
                $bibliografia .= "(" . $riga_obt->pubblicazione . ") ";
        }
        //Bibliografia
        if ($bibliografia != "")
                $infotext3 .= trova_etichetta("min_pubblicazione", "codice", $id_lingua). ": " . $bibliografia;
}
?>
```

6.3.8 - La parte dedicata a paleontologia: paleontologia.php

In questa sezione verrà analizzato il procedimento per l'esportazione delle informazioni dedicate a paleontologia.

Di seguito sono riportate le tabelle coinvolte nell'esportazione (espongo solo le informazioni strettamente necessarie al nostro contesto):

1. tabella: **bnp_ss**
   campi: **dba_idinterno, ss_ssg, ss_sss**
   descrizione: associa all'esemplare , il Genere e la Specie.

```
if ($tipo_museo == "paleontologia")
{

        //compila infotext infotext2
        $query = "select * from bnp_ss where (dba_idinterno=$dba_idinterno);";
        $risultato_bnp = pg_query($query);
        while ($riga_bnp = pg_fetch_object($risultato_bnp))
        {
                //Genere
                $infotext2 .= trova_etichetta("bnp_ss", "ss_ssg", $id_lingua) . ": " .$riga_bnp->ss_ssg . $accapo;
                //Specie
                $infotext2 .= trova_etichetta("bnp_ss", "ss_sss", $id_lingua) . ": " .$riga_bnp->ss_sss . $accapo;
        }
```

2. tabella: **bnp_l_et_etce**
   campi: **codice** chiave primaria, **etce**
   descrizione: contiene nomi degli Eonotemi

3. tabella: **bnp_l_et_etct**
   campi: **codice** chiave primaria, **etct**
   descrizione: contiene i nomi degli Eratemi

4. tabella: **bnp_l_et_etcs**
   campi: **codice** chiave primaria, **etcs**
   descrizione: contiene i nomi dei Sistemi

5. tabella: **bnp_l_et_etcr**
   campi: **codice** chiave primaria, **etcr**
   descrizione: contiene i nomi delle Serie

6. tabella: **bnp_l_et_etcp**

campi: **codice** chiave primaria, **etcp**
descrizione: contiene i nomi dei Piani

Visto che la struttura delle suddette tabelle è simile, ho automatizzato le query con l'utilizzo di un vettore:

```
        //Eonotema, Eratema, Sistema, Serie, Piano
        $arr_campi = array("etce","etct","etcs","etcr","etcp");
        foreach ($arr_campi as $campo)
        {
                $query = "select * from bnp_l_et_$campo where codice in (select et_$campo from bnp_et where
                                dba_idinterno=$dba_idinterno);";
                $risultato_bnp = pg_query($query);
                if ($riga_bnp = pg_fetch_object($risultato_bnp))

                        $infotext2 .= trova_etichetta("bnp_l_et_$campo", "$campo", $id_lingua) . ": " .
                                        $riga_bnp->$campo . $accapo;
}
```

7. tabella: **bnp_et**
   campi: **dba_idinterno, et_etcc, et_etlg, et_etlf, et_etlm, et_etls, et_etn**
   descrizione: associa all'esemplare i campi descrittivi relativi a Cronozona, Gruppo, Formazione, Membro, Strato, Note

Anche in questo caso ho automatizzato con un array, visto che ogni campo richiede anche l'etichetta:

```
        $query = "select * from bnp_et where dba_idinterno=$dba_idinterno;";
        $risultato_bnp = pg_query($query);
        if ($riga_bnp = pg_fetch_object($risultato_bnp))
        {
             //Cronozona, Gruppo, Formazione, Membro, Strato, Note
             $arr_campi = array("et_etcc","et_etlg","et_etlf","et_etlm","et_etls","et_etn");
             foreach ($arr_campi as $campo)
                 $infotext2 .= trova_etichetta("bnp_et", $campo, $id_lingua) . ": " . $riga_bnp->$campo . $accapo;
        }
```

1. tabella: **bnp_et_etd**
   campi: **dba_idinterno, et_etc**
   descrizione: associa all'esemplare la datazione assoluta

```
        $query = "select * from bnp_et_etd where (dba_idinterno=$dba_idinterno);";
        $risultato_bnp = pg_query($query);
```

```
        while ($riga_bnp = pg_fetch_object($risultato_bnp))
        {
             //Datazione assoluta
            $infotext2 .= trova_etichetta("bnp_et_etd", "et_etd", $id_lingua) . ": " .$riga_bnp->et_etd . $accapo;
        }
}
?>
```

6.3.9 - La parte dedicata a fisica: fisica.php

In questa sezione verrà analizzato il procedimento per l'esportazione delle informazioni dedicate a fisica.

Di seguito sono riportate le tabelle coinvolte nell'esportazione (espongo solo le informazioni strettamente necessarie al nostro contesto):

1. tabella: **pst_og**
   campi: **dba_idinterno, og_ogtd** (codice definizione)**, og_ogtt** (codice tipologia)
   descrizione: associa agli esemplari i codici della definizione, e della tipologia

2. tabella: **pst_l_og_ogtd_ha_descrizioneml**
   campi: **codice**, **id_lingua, definizione**
   descrizione: contiene la definizione multilingua

```
if ($tipo_museo == "fisica")
{
        //compila infotext
        //trova la definizione
        $query_base = "select * from pst_l_og_ogtd_ha_descrizioneml, pst_og where
                        (dba_idinterno=$dba_idinterno) and (og_ogtd=codice)";
        $definizione=traduci($query_base,"definizione");
        if ($definizione!=NULL)
        $infotext .= trova_etichetta("pst_l_og_ogtd_ha_descrizioneml", "definizione", $id_lingua) . ": " .
                        $definizione;
```

3. tabella: **pst_l_og_ogtt_ha_descrizioneml**
   campi: **codice**, **id_lingua**, **ogtt**
   descrizione: contiene le tipologie

```
        //compila infotext2
        //trova la tipologia
        $query_base = "select * from pst_l_og_ogtt_ha_descrizioneml, pst_og where
                                (dba_idinterno=$dba_idinterno) and (og_ogtt=codice)";
        $tipologia=traduci($query_base,"ogtt");
        if ($tipologia!=NULL)
                $infotext2 .= trova_etichetta("pst_l_og_ogtt_ha_descrizioneml", "ogtt", $id_lingua) . ": " .
                                $tipologia .$accapo;
```

4. tabella: **pst_ct**
   campi: **dba_idinterno**, **ct_ctp**
   descrizione: associa all'esemplare la categoria principale

5. tabella: **pst_l_ct_ctp_ha_descrizioneml**
   campi: **ctp, id_lingua, codice**
   descrizione: contiene i nomi delle categorie principali (ctp) con i rispettivi codici (codice)

```
//trova la categoria principale
$query_base = "select * from pst_l_ct_ctp_ha_descrizioneml, pst_ct where (dba_idinterno=$dba_idinterno)
                and (ct_ctp=codice)";

$categoria_principale=traduci($query_base,"ctp");
if ($categoria_principale!=NULL)
        $infotext2 .= trova_etichetta("pst_ct", "ct_ctp", $id_lingua) . ": ". $categoria_principale .$accapo;
```

6. tabella: **pst_ct_cta**
   campi: **dba_idinterno, ct_cta**
   descrizione: memorizza i codici di altre categorie associate all'esemplare

In questo caso ad ogni dba_idinterno possono corrispondere più categorie:

```
//trova le altre categorie associate all'esemplare
$query = "select ct_cta from pst_ct_cta where dba_idinterno=$dba_idinterno;";
$risultato_pst = pg_query($query);
$altre_categorie = null;
while($riga_pst = pg_fetch_object($risultato_pst))
{
        $query_base = "select * from pst_l_ct_ctp_ha_descrizioneml where (codice=$riga_pst->ct_cta)";
        $altra_categoria = traduci($query_base,"ctp");
        if ($altra_categoria != null) $altre_categorie .= $altra_categoria . $accapo;
}
if ($altre_categorie != null)
        $infotext2 .= trova_etichetta("pst_ct_cta", "ct_cta", $id_lingua) .":" . $altre_categorie;
```

7. tabella: **pst_ct_ctc**
   campi: **dba_idinterno, ct_ctc**
   descrizione: memorizza i codici (ct_ctc) delle parole chiave associate all'esemplare

8. tabella: **pst_l_ct_ctc_ha_descrizioneml**
   campi: **codice, id_lingua, ct_ctc**
   descrizione: contiene le parole chiave con i risepttivi codici

Anche in questo caso ad ogni dba_idinterno possono corrispondere più parole chiave:

```
//trova le parole chiave
$query = "select ct_ctc from pst_ct_ctc where dba_idinterno=$dba_idinterno;";
$risultato_pst = pg_query($query);
```

```
$parole_chiave = null;
while($riga_pst = pg_fetch_object($risultato_pst))
{
        $query_base = "select * from pst_l_ct_ctc_ha_descrizioneml where (codice=$riga_pst->ct_ctc)";
        $parola_ch = traduci($query_base,"ct_ctc");

        if ($parola_ch != null) $parole_chiave .= $parola_ch . $accapo;
}
if ($parole_chiave != null)
        $infotext2 .= trova_etichetta("pst_ct_ctc", "ct_ctc", $id_lingua) .": ".$parole_chiave;
```

9. tabella: **pst_da_des**
   campi: **dba_idinterno, id_lingua, da_utf , da_utm , da_uts**
   descrizione: associa a dba_idinterno dei dati analitici: funzione (da_utf), modalità d'uso (da_utm) , cronologia d'uso (da_uts)

```
//funzione, modalità d'uso, cronologia d'uso
$arr_campi = array("da_utf","da_utm","da_uts");
foreach ($arr_campi as $campo)
{
        $query_base = "select * from pst_da_des where (dba_idinterno=$dba_idinterno)";
        $valore_campo=traduci($query_base,"$campo");
        if ($valore_campo!=NULL)
                $infotext2 .= trova_etichetta("pst_da_des", "$campo", $id_lingua) . ": ".
                        $valore_campo . $accapo;
}
```

10. tabella: **pst_da_isr**
    campi: **dba_idinterno, da_isrc, da_isrl, da_isrs, da_isrt, da_isrp, da_isra, da_isri**
    descrizione: memorizza i dati per le iscrizioni cioè, classe di appartenenza (da_isrc) , lingua (da_isrl), tecnica di scrittura (da_isrs), tipo di caratteri (da_isrt), posizione (da_isrp), utore (da_isra) , trascrizione (da_isri)

11. tabella : **pst_l_da_isrc**
    campi: **codice** chiave primaria, **isrc**
    descrizione: contiene i codici e i nomi delle classi di appartenenza

12. tabella : **pst_l_da_isrt**
    campi: **codice** chiave primaria, **isrt**
    descrizione: contiene i codici e i nomi dei tipi di caratteri

13. tabella : **pst_l_da_isrs**
    campi: **codice** chiave primaria, **isrs**

descrizione: contiene i codici e i nomi delle tecniche di scrittura

```
        // parte dedicata alle iscrizioni
        $arr_suffissi = array("isrc", "isrs", "isrt");

        foreach ($arr_suffissi as $suffisso)
        {
                $query = "select * from pst_da_isr,pst_l_da_$suffisso where (dba_idinterno=$dba_idinterno) and
                        (codice=da_$suffisso);";
                $risultato_pst = pg_query($query);
                if ($riga_pst = pg_fetch_object($risultato_pst))
                {
                        $valore_campo=$riga_pst->$suffisso;
                        if ($valore_campo!=NULL)
                                $infotext2 .= trova_etichetta("pst_da_isr", "da_$suffisso", $id_lingua) . ": " .
                                        $valore_campo . $accapo;
                }
        }

        //lingua
       $query = "select * from pst_da_isr, l_lingua where (dba_idinterno=$dba_idinterno)and(id_lingua=da_isrl);";
       $risultato_pst = pg_query($query);
       if ($riga_pst = pg_fetch_object($risultato_pst))
              $infotext2 .= trova_etichetta("pst_da_isr", "da_isrl", $id_lingua) . ": " . $riga_pst->lingua . $accapo;

        //posizione, autore, trascrizione
        $arr_campi = array("da_isrp", "da_isra", "da_isri");
        foreach ($arr_campi as $campo)
        {
                $query = "select * from pst_da_isr where (dba_idinterno=$dba_idinterno);";
                $risultato_pst = pg_query($query);
                if ($riga_pst = pg_fetch_object($risultato_pst))
                {
                        $valore_campo=$riga_pst->$campo;
                        if ($valore_campo!=NULL)
                                $infotext2 .= trova_etichetta("pst_da_isr", "$campo", $id_lingua) . ": " .
                                                $valore_campo . $accapo;
                }
        }
  }
?>
```

6.3.10 - Altri file del programma

Il file *config_lingue.php*:

```
<?php
//compila l'array di configurazione delle lingue
$arr_lingue["ita"]="Italiano";
$arr_lingue["eng"]="Inglese";
$arr_lingue["fra"]="Francese";
$arr_lingue["deu"]="Tedesco";
$arr_lingue["esp"]="Spagnolo";
/*
imposta la traduzione di default. Serve per le traduzioni in lingua diversa da quella di default, cioè nelle operazioni di ricerca dati ed etichette multilingua in caso di mancata traduzione sul db nella lingua cercata viene preso il dato nella lingua di default.
*/
$traduzione_default="ita";
?>
```

Il file *config_connessione.php*:

```
<?php
//stringa di connessione al database compatibile con le specifiche di pg_connect()
$stringa_di_connessione="host=xxxx port=xxxx dbname=xxxx user=xxxx password=xxxx";
?>
```

Il file *crea_archivio_zip.php*. Utilizza la funzione di compressione dati gzcompress() della libreria Zlib di PHP e si attiene alle specifiche pubbliche di formato file Zip per creare l'archivio compresso. Nel file *crea_archivio_zip.php* c'è la funzione per aggiungere un nome di directory nell'archivio *zip_crea_cartella()* , la funzione per aggiungere un file nell'archivio *zip_inserisci_file()* e la funzione di assemblamento dell'archivio *zip_assembla()*. Il codice trova tutti i file e le sottocartelle a partire dalla directory "*esportazioni_xml/$sigla_museo/"* e li include nel file archivio zip.
Alla fine della procedura viene registrato il file nominato *Dati_PDA_$sigla_museo.zip* nella directory "*esportazioni_xml/*" e viene aggiunto nella pagina HTML di EsPDAmuseo il link relativo per il download. Di seguito riporto solo alcune righe del codice:

```
.....
.............
$arr_dir_zip[]=null; //vettore che conterrà le sottocartelle da esaminare
array_push($arr_dir_zip, "esportazioni_xml/$sigla_museo/"); //cartella iniziale
$offset_percorso_file = strlen("esportazioni_xml/$sigla_museo/"); //questa parte di path non serve allo zip
//comincia l'esplorazione delle cartelle
```

```
while (count($arr_dir_zip)>0)
{
 $dir = array_pop($arr_dir_zip);
 if ($dh = opendir($dir))
 {
  while (($file = readdir($dh)) !== false ) //se la cartella non è vuota
  {
     if(($file != ".")&&($file != ".."))   //se è un file o una directory da includere
     {
        if(is_dir($dir.$file)) //se è una directory
        {
                //memorizza la directory nel vettore per poterne esaminare il contenuto ad una iterazione del ciclo
                array_push($arr_dir_zip, $dir.$file."/");
                //include la cartella nello zip
                zip_crea_cartella(substr($dir.$file."/", $offset_percorso_file));
        }
        else
        {
                //legge il file
                $contenuto_file = file_get_contents($dir.$file);
                //include il file nello zip
                zip_inserisci_file($contenuto_file, substr($dir.$file, $offset_percorso_file));
        }
    }
   }
   closedir($dh);
 }
}

$percorso_file_zip = "esportazioni_xml/Dati_PDA_$sigla_museo.zip"; //il nome del file zip
$fd = fopen ($percorso_file_zip, "wb"); //crea o sovrascrive il file; wb forza la scrittura in modalità binaria
$out = fwrite ($fd, zip_assembla()); //scrive lo zip
fclose ($fd);
echo "<p><strong><a href='$percorso_file_zip'>Scarica il file</a></strong> (.zip)</p>"; //mette il link HTML
unset ($zip_archivio);
?>
```

Il file *intestazione.php*. E' l'intestazione di ogni pagina web di *EsPDAmuseo*.

Ne riporto solo alcune righe:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
.....
.................
<body>
<h1>Esportazione XML per palmari</h1><br><div class='boxcontenuti'>
```

Il file *pie_di pagina.php*. E' il piè di pagina di ogni schermata web di E*sPDAmuseo*. Ne presento solo una piccola parte:

```
</div>
<div id="pie_pagina">
.......
.................
</body>
</html>
```

Il file *stile.css* è il foglio di stile. Essendo EsPDAmuseo un'applicazione web basata su HTML, necessita dell'utilizzo dei CSS per facilitare l'uniformità dello stile gafico delle pagine web e la separazione del contenuto informativo dalla parte di presentazione. Ne riporto solo alcune righe:

```
body { padding: 0; margin: 0; font-family: Arial, Helvetica, Sans-serif; font-size: 16px; letter-spacing: 1px; color:
#000; background-color: #fff;}
table { margin: 5px; font-size: 1em; }
td { border: 0; margin: 2px; padding: 2px; vertical-align:top;}
......
.....................
h1 { color: #088; font-size: 1.5em; font-weight: bold; margin: .6em; border-bottom:1px solid #388;}
```

6.3.11 - Tabelle per le etichette dati

Durante la realizzazione di *EsPDAmuseo*, man mano che venivano considerati i vari campi descrittivi relativi alla compilazione degli attributi infotext, infotext2, infotext3 venivano popolate le tabelle **xml_etichetta_campo** ed **xml_etichetta_campo_traduzione** in modo da poter includere nel risultato XML i dati con le rispettive etichette (vedi la sezione "Il centro dell'applicazione: esporta_xml.php").
Di seguito viene riportato lo stato attuale del contenuto delle tabelle dedicate alle etichette.

Tabella **xml_etichetta_campo**

id	tabella_database	campo_tabella	etichetta_default
1	min_ha_descrizione_sintetica	descrizione_esemplare_sintetica	Descrizione sintetica
2	min_ha_descrizione_sintetica	minerali_associati	Minerali associati
3	min_scheda_esemplare	dimensione_x	Larghezza
4	min_scheda_esemplare	dimensione_y	Altezza
5	min_scheda_esemplare	dimensione_z	Profondita
6	min_scheda_esemplare	peso	Peso
7	min_scheda_esemplare	valore	Valore
8	min_l_localita	denominazione	Localita di provenienza
9	min_l_comune	denominazione	Comune di provenienza
10	min_l_provincia	provincia	Provincia di provenienza
11	min_l_regione	regione	Regione di provenienza
12	min_l_nazione	denominazione	Nazione di provenienza
13	min_al_tipologia_localita	denominazione	Tipologia localita di provenienza
14	min_collezione_provenienza	nome_provenienza	Collezione di provenienza
15	min_pubblicazione	codice	Bibliografia
16	obt_al_nome_pianta_lat	nome_pianta_lat	Nome latino
17	obt_al_nome_pianta_ha_descrizione	nome_pianta	Nome
18	obt_al_nome_famiglia_ha_descrizione	nome_famiglia	Famiglia
19	obt_tipo_collezione_ha_descrizione	nome	Tipo collezione
20	palm_esemplare_ha_descrizione_estesa	descrizione	Descrizione
21	obt_ha_descrizione	note	Note
22	obt_ha_descrizione	posizione_nel_orto	Posizione nell'orto
23	obt_ha_descrizione	luogo_di_origine	Luogo di origine
24	obt_ha_descrizione	habitat	Habitat
25	obt_ha_descrizione	caratteristiche_botaniche	Caratteristiche botaniche
26	obt_ha_descrizione	utilizzo_usi	Utilizzo
27	obt_ha_descrizione	approfondimenti	Approfondimenti
28	bnp_ss	ss_ssg	Genere
29	bnp_ss	ss_sss	Specie
30	bnp_l_et_etce	etce	Eonotema
31	bnp_l_et_etct	etct	Eratema
32	bnp_l_et_etcs	etcs	Sistema
33	bnp_l_et_etcr	etcr	Serie
34	bnp_et	et_etcc	Cronozona
35	bnp_et	et_etlg	Gruppo
36	bnp_et	et_etlf	Formazione
37	bnp_et	et_etlm	Membro
38	bnp_et	et_etls	Strato
39	bnp_et	et_etn	Note
40	bnp_et_etd	et_etd	Datazione assoluta
41	bnp_l_et_etcp	etcp	Piano
42	pst_l_og_ogtd_ha_descrizioneml	definizione	Definizione
43	pst_l_og_ogtt_ha_descrizioneml	ogtt	Tipologia

id	tabella_database	campo_tabella	etichetta_default
44	pst_ct	ct_ctp	Categoria principale
45	pst_ct_cta	ct_cta	Altre categorie
46	pst_ct_ctc	ct_ctc	Parole chiave
47	pst_da_des	da_utf	Funzione
48	pst_da_des	da_utm	Modalita d'uso
49	pst_da_des	da_uts	Cronologia d'uso
50	pst_da_isr	da_isrc	Classe di appartenenza
51	pst_da_isr	da_isrl	Lingua
52	pst_da_isr	da_isrs	Tecnica di scrittura
53	pst_da_isr	da_isrt	Tipo di caratteri
54	pst_da_isr	da_isrp	Posizione
55	pst_da_isr	da_isra	Autore
56	pst_da_isr	da_isri	Trascrizione

Tabella **xml_etichetta_campo_traduzione** (solo alcune traduzioni per cominciare...)

id_etichetta	id_lingua	Traduzione
1	eng	Brief description
2	eng	Associate ores
3	eng	Width
4	eng	Height
5	eng	Depth
6	eng	Weight
7	eng	Value
8	eng	Place of provenance
9	eng	Town of provenance
10	eng	Province of provenance
11	eng	Region of provenance
12	eng	Country of provenance
13	eng	Typology of place of provenance
14	eng	Collection of provenance
15	eng	Bibliography
16	eng	Latin name
17	eng	Name
18	eng	Family
19	eng	Type of collection
20	eng	Descrption
21	eng	Notes
22	eng	Garden position
23	eng	Provenance
24	eng	Habitat
25	eng	Botanic features
26	eng	Use
27	eng	Closer examination
42	eng	Definition
43	eng	Typology
44	eng	Main category
45	eng	Other categories
46	eng	Keywords

7 - Conclusioni

Per la realizzazione di *EsPDAmuseo* ho cercato di impostare la struttura del programma (funzioni e file di include) con l'intento di facilitare l'implementazione dei futuri aggiornamenti. Cioè ho cercato di individuare quali fossero le parti più a rischio di recidive modifiche cercando di isolarle e renderle facilmente accessibili ai programmatori che le modificheranno.

In questa relazione ho presentato *EsPDAmuseo* cominciando da una descrizione generale per poi approfondire le caratteristiche dell'applicazione fino all'esposizione del codice sorgente delle procedure più specifiche.

Una volta che il database centrale avrà una strutturazione definitiva e stabile ci si potrà dedicare all'ottimizzazione delle query di *EsPDAmuseo* e all'ulteriore minimizzazione dei tempi di esecuzione. Alcune funzionalità come ad esempio la visualizzazione dei report delle esportazioni che ora sono molto utili in fase di test portanno essere tolte.

8 - Documentazione On-line

Documentazione PHP	Http://www.php.net
Documentazione PostgreSQL	Http://www.postgresql.org
Documentazione Apache	Http://www.apache.org
Documentazione specifiche Zip	Http://www.pkware.com
Documentazione HTML 4.0 e CSS	Http://www.w3.org