Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in

Ingegneria Informatica

Progettazione e implementazione di un'interfaccia web per la gestione e consultazione di un Conservatorio di Musica

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Matteo Valerio

Anno Accademico 2012/2013

a chi mi ha sostenuto

Indice

Elenco delle figure

Elenco dei listati codice

Capitolo 1

Introduzione

Lo scopo della presente tesi è quello di illustrare la progettazione e implementazione di un'applicazione web volta a gestire la struttura organizzativa di un Conservatorio di musica, quali le iscrizioni, le registrazioni degli esami e le organizzazioni dei corsi. Le informazioni, che saranno archiviate e organizzate in un database, saranno accessibili tramite l'utilizzo dell'interfaccia web implementata tramite un framework php. Si è voluto intraprendere questa scelta perché al giorno d'oggi all'interno di grandi istituti di alta cultura (quali Università e Conservatorio) è comune l'informatizzazione delle informazioni e l'interfacciamento con esse tramite sito web. Lo sviluppo del progetto è stato diviso nelle seguenti fasi:

- Capitolo 2: Analisi dei requisiti. Raccolta di informazioni sulla strutturazione del Conservatorio e sulle necessarie operazioni sui dati (Sezione 2.2)
- Capitolo 3: Progettazione e implementazione del database.
- Capitolo 4: Progettazione e implementazione di un'applicazione web basata sul framework php CodeIgniter [1].
- Capitolo 5: Conclusioni.

[1] http://ellislab.com/codeigniter/

Capitolo 2

Analisi dei requisiti

Lo scopo di questo progetto è la gestione dei dati relativi agli esami svolti in un Conservatorio di Musica: per questo motivo è stata analizzata la struttura interna della scuola (gli studenti iscritti, i docenti e gli insegnamenti erogati). Di seguito, in dettaglio, vengono descritti tutti i dati raccolti con delle considerazioni utili a facilitare la progettazione nella fase successiva.

2.1 Requisiti strutturati

- **Frasi per Studente**
  Ogni studente viene registrato qualora esso abbia effettuato almeno un'iscrizione al Conservatorio e può essere iscritto al più a una classe principale. Del suddetto vengono mantenuti dei recapiti (e-mail istituzionale, telefono), il nominativo (nome, cognome), il sesso, l'indirizzo, la città, il CAP, la data di nascita e gli viene assegnata una matricola univoca. Vengono mantenuti, inoltre, i dati riguardo allo storico degli esami di uno studente con relativo voto e data e la lista degli insegnamenti frequentati. Si vogliono, inoltre, conservare i dati sugli studenti ritirati.

- **Frasi per Classe Principale**
  Classe principale rappresenta in parte il concetto comune di "piano di studio", solo che è immutabile: gli insegnamenti associati ad esso sono fissati e distribuiti a seconda dell'anno di frequenza dello studente iscritto. Come attributo chiave è sufficiente utilizzare il suo nome in quanto non esistono duplicati. Viene inoltre mantenuta una piccola descrizione generica dei suoi contenuti.

- **Frasi per Insegnamento**
  Ogni insegnamento è una materia erogata dal Conservatorio. Dal momento che molte materie hanno durata pluriennale è stato scelto di differenziare

l'insegnamento anno per anno (i.e. Solfeggio1 è diverso da Solfeggio2): per questo motivo l'attributo chiave è l'ID e non il nome. Un insegnamento deve essere associato a un solo dipartimento e lo stesso insegnamento può essere tenuto da più docenti che possono variare anno per anno.

- **Frasi per Dipartimento**
  Ogni dipartimento è un insieme di insegnamenti affini (i.e. Dipartimento archi conterrà violino, violoncello, viola, viola da gamba e contrabbasso) e viene identificato da un ID (attributo chiave). Ogni dipartimento è diretto da un solo docente. Per ogni dipartimento si conserva anche il nome.

- **Frasi per Docente**
  Di ogni docente che insegna in Conservatorio si conservano dei recapiti (telefono, e-mail istituzionale), il nominativo (nome, cognome), il sesso, l'indirizzo, la città, il CAP, la data di nascita e il Codice Fiscale (attributo chiave). Si vogliono conservare gli insegnamenti erogati anno per anno dal singolo docente. Ogni docente può essere direttore di al più un dipartimento di cui i propri insegnamenti fanno parte.

2.2 Principali azioni sulla base di dati

Operazione	Tipo	Frequenza
Lista di tutti gli studenti iscritti	Interrogazione	1/anno
Lista di tutti i docenti	Interrogazione	1/mese
Lista di tutti gli insegnamenti	Interrogazione	1/mese
Aggiornamento studenti	Modifica	50/giorno all'inizio dell'Anno Accademico
Aggiornamento esami	Modifica	50/giorno durante le sessioni d'esame

2.3 Glossario

Termine	Descrizione	Sinonimi	Collegamenti
Studente	Chi studia al Conservatorio	Allievo	Classe Principale, Insegnamento
Classe Principale	Piano di studi	Strumento Principale	Insegnamento, Studente
Insegnamento	Materia di studio	Materia	Studente, Classe Principale, Docente, Dipartimento
Docente	Chi insegna al Conservatorio	Insegnante, Maestro	Insegnamento, Dipartimento
Dipartimento	Collezione di insegnamenti affini		Docente, Insegnamento

Capitolo 3

Progettazione del Database

Il lavoro di progettazione della base di dati utilizzata nel progetto è stato suddiviso sostanzialmente in tre fasi:

- Progettazione concettuale: schema E-R e sua ristrutturazione
- Progettazione logica: schema logico
- Progettazione fisica: implementazione vera e propria su una base di dati fisica

In questo capitolo verranno illustrate queste fasi.

3.1 Progettazione Concettuale

Lo scopo della progettazione concettuale è quello di creare uno schema di alto livello che rappresenti i dati e le relazioni tra di essi relativi al mini-mondo analizzato. Proprio perché si lavora ad alto livello, questa fase di progettazione garantisce l'assenza di dettagli implementativi che ne impedirebbero la comprensione a utenti non tecnici (quali il committente).
Attingendo alle informazioni raccolte nella preliminare analisi dei requisiti, i dati verranno suddivisi in due categorie: **entità** e **associazioni**.
Un'entità rappresenta un oggetto (fisico o astratto) del mini-mondo analizzato che viene descritto tramite attributi. Quando si costituisce un'entità si definiscono due concetti:

- un oggetto con vari attributi, ossia un tipo di entità che possiede le caratteristiche elencate (rappresentazione INTENSIONALE)
- una moltitudine di oggetti comuni, ossia una collezione di entità (rappresentazione ESTENSIONALE)

Ogni entità deve possedere almeno un **attributo chiave**, ossia un attributo che identifica univocamente l'entità. Graficamente un'entità viene disegnata con un

rettangolo all'interno del quale è scritto il suo nome rappresentativo.
Un'associazione, invece, rappresenta i legami logici e le relazioni che sussistono tra le varie entità trovate. Graficamente un'associazione viene disegnata con un rombo all'interno del quale è scritto il suo nome rappresentativo.

3.1.1 Modello Concettuale: Entità-Associazione (E-R)

In figura 3.1 è raffigurato lo schema concettuale per la rappresentazione della realtà di interesse:

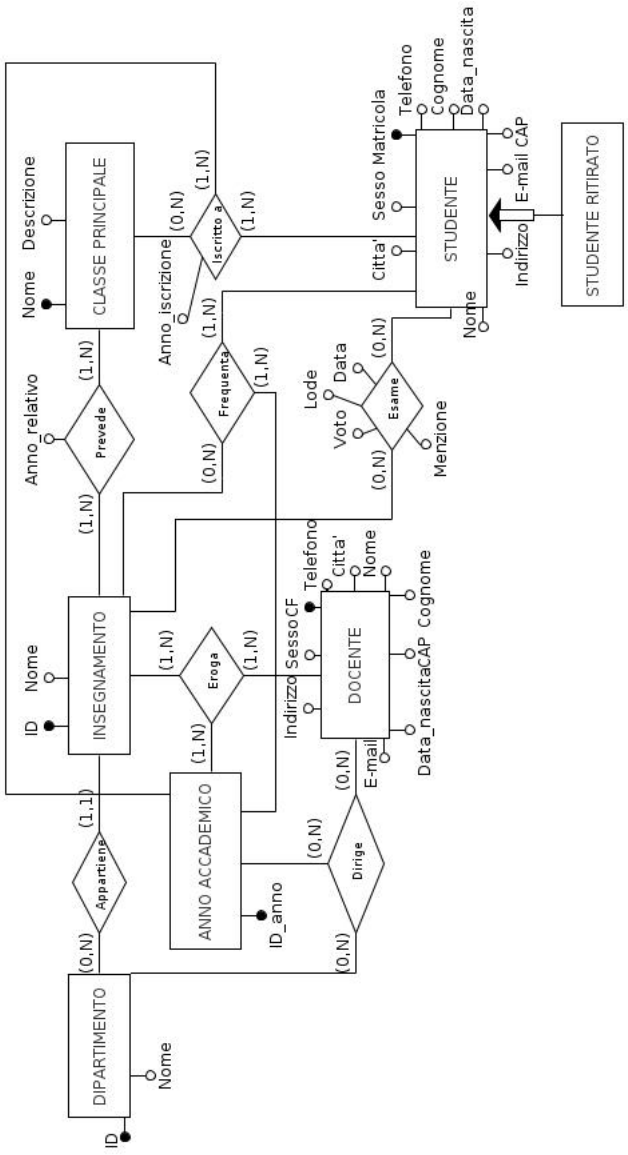


Figura 3.1: Modello Concettuale - Schema E-R

3.1.2 Ristrutturazione schema E-R

Durante la costruzione di uno schema E-R, è frequente l'uso di strutture più complesse e avanzate (i.e. le generalizzazioni e le specializzazioni) le quali, nonostante si avvicinino più alla logica del mini-mondo rappresentato, devono essere trasformate in sole entità o associazioni.
Questo procedimento prende il nome di ristrutturazione dello schema E-R. In figura 3.2 è rappresentato lo schema E-R concettuale ristrutturato. Dopo un'analisi formale qualitativa/quantitativa la generalizzazione "Studente Ritirato" è stata trasformata in entità debole (per evitare la ripetizione di moltissimi valori FALSE).

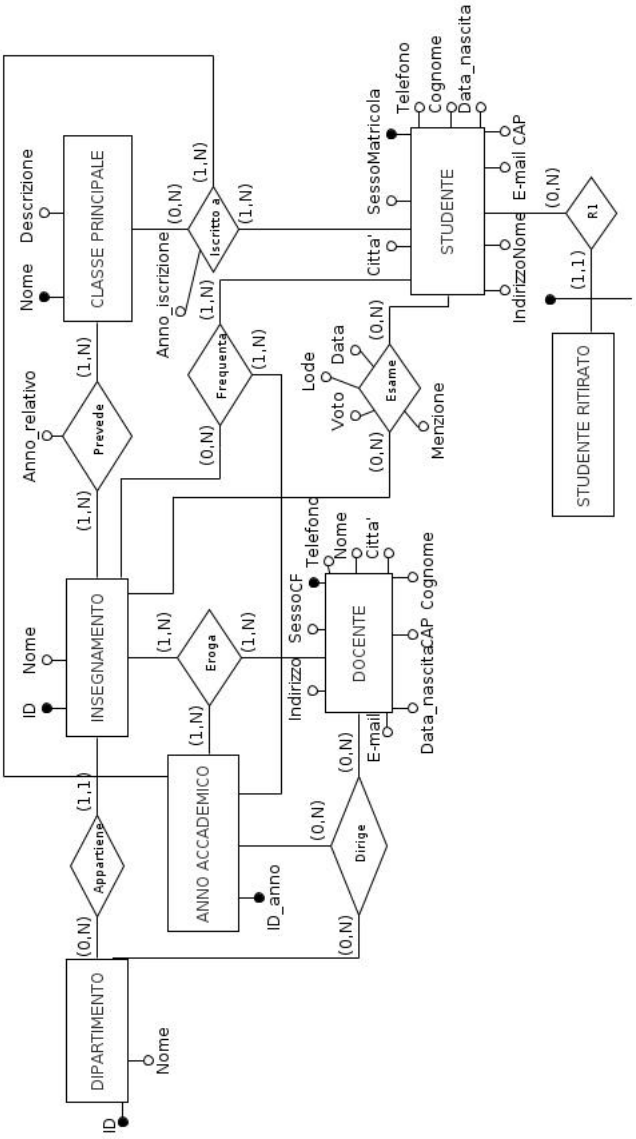


Figura 3.2: Modello Entità-Associazione (E-R) ristrutturato.

3.1.3 Dizionario dei dati

Entità

Entità	Descrizione	Attributi	Identificatore
Studente	Si iscrive a una determinata Classe Principale, frequenta dei corsi e sostiene degli esami	Matricola, Nome, Cognome, Sesso, Data_di_nascita, Indirizzo, Cittá, CAP, Telefono, E-mail	Matricola
Studente Ritirato	Studente che non ha finito il percorso di studi		Matricola
Classe Principale	Piano di Studi	Nome, Descrizione	Nome
Insegnamento	Corso annuale	ID, Nome	ID
Anno Accademico	Anno	ID_anno	ID_anno
Docente	Tiene uno o più insegnamenti	CF, Nome, Cognome, Sesso, Data_di_nascita, Indirizzo, Città, CAP, Telefono, E-mail	CF
Dipartimento	Contenitore di materie affini	ID, Nome	ID

Associazioni

Associazione	Attributi	Entità Collegate
R1		Studente (0,N), Studente Ritirato (1,1)
Iscritto_a	Anno_iscrizione	Classe Principale (0,N), Studente (1,N), Anno Accademico (1,N)
Prevede	Anno_relativo	Classe Principale (1,N), Insegnamento (1,N)
Frequenta		Insegnamento (0,N), Studente (1,N), Anno Accedemico (1,N)
Esame	Voto, Lode, Menzione, Data	Insegnamento (0,N), Studente (0,N)
Eroga		Anno Accademico (1,N), Insegnamento (1,N), Docente (1,N)
Appartiene		Insegnamento (1,1), Dipartimento (0,N)
Dirige		Docente (0,N), Dipartimento (0,N), Anno Accademico (0,N)

3.1.4 Schema Concettuale, Regole di vincolo

Per rispettare la rappresentazione fedele della realtà di interesse descritta in sezione 2.2 è necessario definire delle regole di vincolo per concetti altrimenti non esprimibili utilizzando il modello E-R.

- **Regola di Vincolo 1**
  Un insegnante può dirigere un determinato dipartimento solo se i suoi insegnamenti dell'anno in corso ne fanno parte.

- **Regola di Vincolo 2**
  Un dipartimento deve essere diretto da un solo docente per anno accademico: la cardinalità minima è stata posta a 0 in quanto nel momento in cui viene creato un nuovo dipartimento si può non essere ancora a conoscenza di chi lo dirigerà.

- **Regola di Vincolo 3**
  Uno studente può frequentare solo gli insegnamenti previsti dal suo piano di studi per il determinato anno a cui è iscritto.

- **Regola di Vincolo 4**
  Uno studente si può iscrivere ad una sola Classe principale per anno accademico.

- **Regola di Vincolo 5**
  Uno studente si può iscrivere all'anno n-esimo di una determinata classe solo se ha superato positivamente tutti gli esami previsti per l'anno (n-1)-esimo della medesima classe, fatta eccezione dell'iscrizione al primo anno.

- **Regola di Vincolo 6**
  Uno studente può sostenere esami solo per gli insegnamenti previsti dalla Classe principale che frequenta.

3.2 Progettazione Logica

La fase di progettazione successiva viene chiamata **progettazione logica** che ha lo scopo di creare un modello relazionale. Il modello relazionale è stato introdotto nel 1970 da Ted Codd e basa il suo fondamento teorico nella teoria degli insiemi e nella logica dei predicati del primo ordine.
Il modello logico rappresenta la base di dati come una collezione di relazioni. Una relazione costituisce una tabella di valori: ad ogni riga corrisponde un insieme di dati collegati (nel caso specifico corrisponde un'entità o un'associazione del mondo reale) mentre ad ogni colonna corrisponde un valore di uno specifico attributo.
Ogni schema di relazione è costituito da un nome rappresentativo **R** e da una lista di attributi (A_1 , A_2, ..., A_N).
Dato il suo fondamento teorico sulla teoria degli insiemi, sarà possibile operare sui dati utilizzando regole e operazioni insiemistiche (quali intersezioni, prodotto

cartesiano, unione ...).
A partire dallo schema E-R ristrutturato descritto nel capitolo precedente, si trasformano le varie entità ed associazioni in relazioni.

3.2.1 Modello Logico: Relazionale

In figura 3.3 lo schema logico prodotto per la rappresentazione della realtà di interesse:

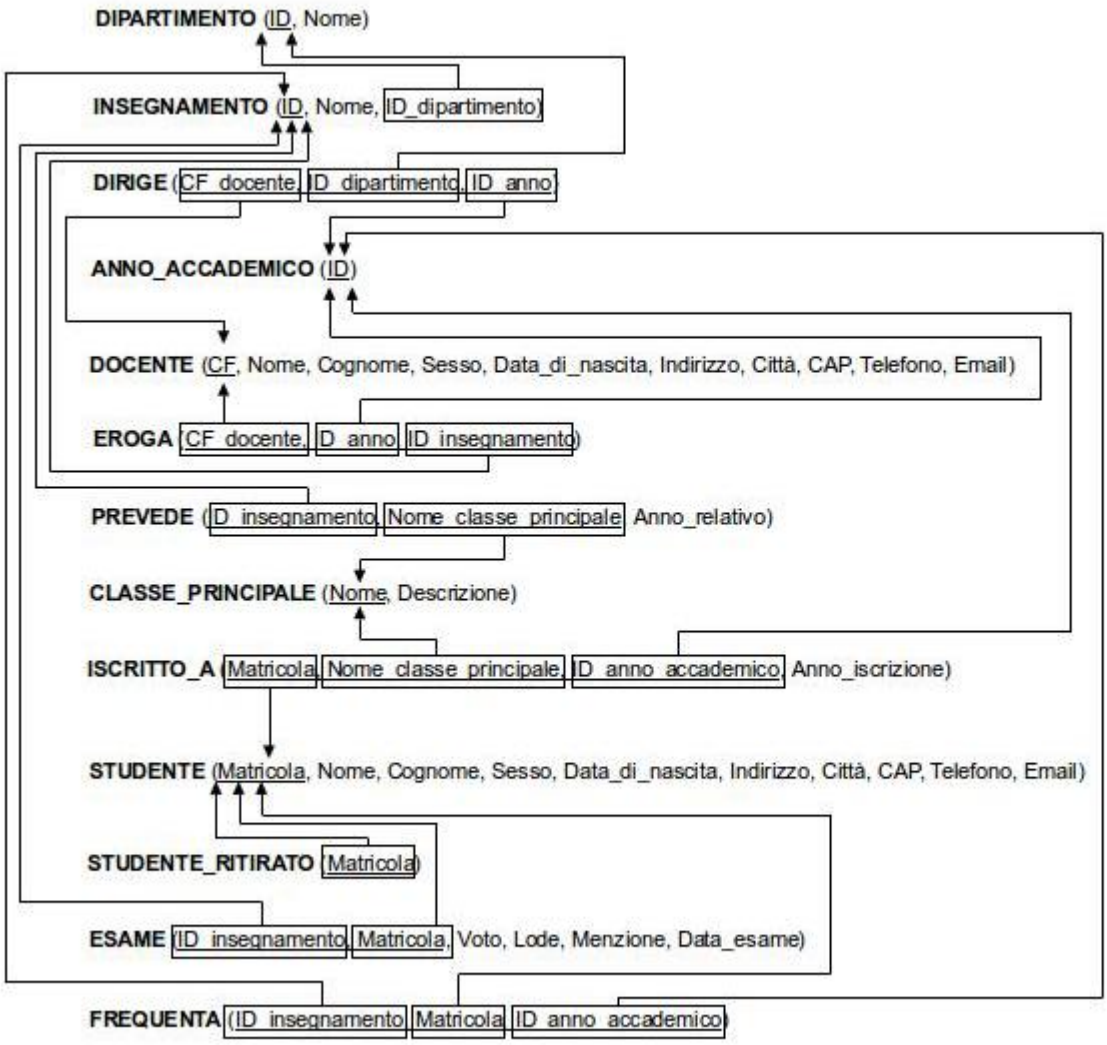


Figura 3.3: Modello Relazionale - Schema Logico

3.2.2 Schema Logico: Regole di Vincolo

Di seguito sono elencati i vincoli non esprimibili tramite gli strumenti forniti dalla progettazione concettuale.

- **RV1**: Gli attributi di Studente non devono essere nulli.
- **RV2**: Gli attributi di Esame non devono essere nulli.
- **RV3**: Gli attributi di Docente non devono essere nulli.
- **RV4**: Gli attributi di Iscritto_a non devono essere nulli.
- **RV5**: Gli attributi di Prevede non devono essere nulli.
- **RV6**: Una Classe Principale deve partecipare ad almeno un'istanza di Prevede.
- **RV7**: Un Insegnamento deve partecipare ad almeno un 'istanza di Prevede.
- **RV8**: Uno Studente deve partecipare ad almeno un'istanza di Frequenta.
- **RV9**: Un Anno Accademico deve partecipare ad almeno un'istanza di Frequenta.
- **RV10**: Un Anno Accademico deve partecipare ad almeno un'istanza di Eroga.
- **RV11**: Un Insegnamento deve partecipare ad almeno un'istanza di Eroga.
- **RV12**: Un Docente deve partecipare ad almeno un'istanza di Eroga.
- **RV13**: Uno studente deve partecipare ad almeno un'istanza di Iscritto_a.
- **RV14**: Un Anno Accademico deve partecipare ad almeno un'istanza di Iscritto_a.

Capitolo 4

Progettazione e implementazione dell'interfaccia web

Il classico supporto per un'interfaccia web a una base di dati è l'utilizzo di codice php: il php, infatti, permette la generazione di pagine dal contenuto dinamico operando dal lato server. Per questo progetto si è scelto di utilizzare un particolare framework php: CodeIgniter[1]. Questo framework fornisce un supporto notevole nella generazione di progetti web dinamici con l'approccio MVC, Model View Controller.

Nel capitolo verranno illustrati:

- Un'introduzione sul Php e sul suo funzionamento;
- Una descrizione del design pattern MVC utilizzato nel progetto;
- Una descrizione sul funzionamento del framework **CodeIgniter**;
- Un **esempio** dell'utilizzo fatto del pattern MVC in CodeIgniter.

4.1 PHP - Hypertext Preprocessor

Il Php[2] è un linguaggio di programmazione interpretato (non necessita quindi di essere compilato)open-source utilizzato prinicipalmente per la generazione di pagine web dinamiche. Un file Php può contenere testo semplice, codice HTML, codice Javascript oltre che a codice Php. Il linguaggio Php si accomuna molto ai più usati linguaggi di programmazione dal momento che è un linguaggio ad oggetti, fornendo prevalentemente dei costrutti che permettono di agire ad alto livello. Quando viene lanciato del codice Php (per esempio accedendo a una pagina web) questo viene eseguito sul web server contente il sorgente, che restituisce una pagina HTML visualizzabile tramite un browser web dall'utente finale. Il fondamentale vantaggio di utilizzare dei file Php al posto di semplici file HTML

[1] http://ellislab.com/codeigniter/
[2] http://php.net/

è la drastica riduzione di codice scritto: eseguendo lo stesso sorgente Php è possibile ottenere diversi risultati in situazioni differenti senza quindi dover scrivere un file HTML per ognuna di esse, in altre parole pagine web dinamiche.

4.2 MVC-Model View Controller

Un design pattern è una soluzione astratta a un problema reale ricorrente. Nonostante l'impiego di design patterns sia prevalentemente in campo informatico, essi possono essere validi anche in altri campi. Solitamente un design pattern è costituito da un nome, da una descrizione del problema affrontato e dalla sua soluzione. Il problema affrontato dal pattern MVC è quello di separare la logica del funzionamento del programma dalle considerazioni relative all'interfaccia utente[3]. Le classi dell'applicazione vengono così suddivise in tre sezioni:

- Model: contiene i dati relativi all'applicazione, con tutti i metodi di accesso, scrittura, modifica e cancellazione dei dati

- View: si occupa solo della realizzazione grafica, del modo in cui i dati appariranno all'utente finale

- Controller: gestisce il flusso di dati dal Model alla View, coordinando le interazioni tra le due sezioni.

Ogni richiesta di dati per un'interfaccia grafica giungerà sempre al controller che poi rielaborerà inoltrando poi la richiesta al Model. Viceversa i dati forniti dal Model verranno inviati al Controller che poi intoltrerà alla View.

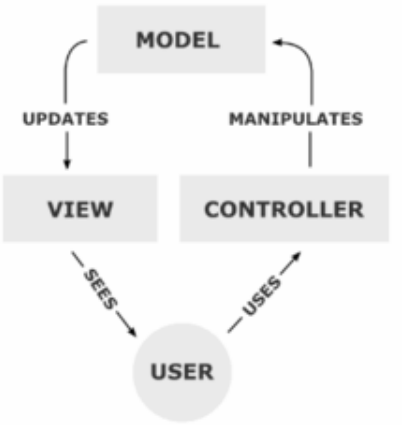


Figura 4.1: Schema di funzionamento del pattern MVC

[3]"Design Patterns: Elements of Reusable Object-Oriented Software"by E. Gamma et al. Addison-Wesley, 1994.

Il Model non conosce nessun dettaglio di come i dati verranno poi utilizzati, non ha alcun collegamento diretto con la View. Questo pattern pertanto fornisce una forte modularità e manutentibilità del codice, permettendo quindi agili modifiche, aggiornamenti.

4.3 CodeIgniter[4]

Code Igniter è un progetto open-source prodotto dall'azienda EllisLab che fornisce importanti agevolazioni per lo sviluppo di siti web in Php con approccio MVC. Lo scopo principale di questo framework è quello di velocizzare la produzione del codice fornendo molte librerie di utilità per i problemi e le necessità più comuni durante lo sviluppo di un progetto web. Questo permette di velocizzare la stesura del codice e agevola la creatività del programmatore. Tra i vari framework Php disponibili, la scelta è caduta su questo framework perchè:

- è open-source, con licenza Apache/BSD-style[5] , quindi liberamente utilizzabile;
- occupa poco spazio su disco e richiede pochissime librerie come prerequisiti, al contrario di altri framework che richiedono moltissime librerie preinstallate che complicano così notevolmente il lavoro iniziale;
- il codice è fortemente ottimizzato, viene eseguito molto velocemente;
- utilizza l'approccio MVC, favorendo molto progetti, come questo, che utilizzano template HTML;
- genera URL chiari e corti semplificando la lettura anche per l'utente finale;
- fornisce un vasto insieme di utility per il programmatore tra cui: facile accesso a database(i maggiori DBMS sono supportati), invio di e-mail, validazione delle form, utilizzo delle sessioni, manipolazione di immagini, test e molto altro;
- è facilmente estensibile nel caso in cui il programmatore avesse la necessità di utilizzare insieme ad esso altre librerie;

[4]http://ellislab.com/codeigniter/
[5]http://opensource.org/licenses/bsd-license.php

- non richiede l'uso dei template-engine che peggiorerebbero drasticamente la performance del codice;

- è documentato con estrema chiarezza;

- è presente una community di supporto molto attiva;

- è facile da utilizzare, il proprio codice va inserito all'interno di specifiche cartelle del package scaricato;

4.4 MVC e Code Igniter

Come già detto in precedenza Code Igniter fornisce supporto nell'utilizzo del pattern MVC durante lo sviluppo del progetto. Questo risulta evidente già all'apertura di un nuovo progetto: il programmatore troverà tre cartelle, tra le numerose presenti, chiamate Models, Controllers e Views all'interno delle quali inserirà il proprio codice relativo alla logica della corretta sezione del pattern. Un piccolo esempio relativo a una View sviluppata all'interno del mio progetto:

Codice 4.1: DocenteView.php

```
1
2  <?php
3  print '<table border=1, caption=docenti>';
4  echo '<tr><td><h3><b>Cognome</h3></td><td><h3><b>Nome</h3></td>
5      <td><h3><b>Insegnamento</h3></td><td><h3><b>email</h3></td><tr/>';
6  foreach($docente as $d):
7      echo '<tr><td>'.$d['cognome'].'</td>
8          <td>'.$d['nome'].'</td><td>'.$d['inome'].'</td><td>'.$d['email'].'</td><tr/>';
9
10 endforeach;
11 print '</table>';?>
```

All'interno di questo file è presente solo codice php che, ricevendo dati dal controller nella variabile "docente", stamperà a schermo una tabella contenente informazioni riguardo al nome, cognome del docente e il nome dell'insegnamento da lui erogato.

Il controller principale del mio sito è contenuto nel file “Conservatorio.php”, di seguito una piccola parte come continuazione del precedente esempio:

Codice 4.2: Conservatorio.php

```
1
2  <?php
3  class Conservatorio extends CI_Controller {
4
5    public function __construct(){
6      parent::__construct();
7      $this->load->helper('array');
8      $this->load->helper('url');
9      $this->load->model('visualizza_model');
10   }
11
12   public function __template($title, $page, $result)
13   {
14     $data['title'] = $title;
15     $this->load->view('templates/standard_header', $data);
16
17     /* Dopo aver caricato l'header controlla se i valori di $page e
18      * $result sono stringhe vuote
19      * in tal caso la pagina da caricare è la home!
20      */
21     if($page != '' || $result != '')
22     {
23       $this->load->view($page, $result);
24     }
25     /* Se $page e $result non sono stringhe vuote(quindi si tratterebbe della home)
26      * controlla se il file esiste.
27      * Se non esiste carica la pagina 404
28      */
29     else if($page != '' && $result != '' && ! file_exists('application/views/'.$page.'.php'))
30     {
31       show_404();
32     }
33
34     $this->load->view('templates/footer');
35   }
36
37   public function index()
38   {
39     $this->__template('BENVENUTO!', '', '');
40   }
41
42   public function docente()
43   {
44     $data['docente'] = $this->visualizza_model->get_docente();
45     $data['title'] = 'DOCENTI';
46     $this->__template($data['title'], 'docenteView', $data);
47   }
48   ...
49 }
```

Il metodo privato __construct() è obbligatorio e serve per caricare le librerie utilizzate nel file e i vari file model con cui si interagirá. Il metodo privato di utilità __template($title, $page, $result) carica dinamicamente l'header del template HTML usato in questo progetto, la pagina specifica richiesta dall'utente o una versione personalizzata della pagina di errore 404. Il metodo docente() ottiene i dati relativi ai vari docenti del conservatorio chiamando il metodo get_docente() della classe visualizza_model che interrogherà direttamente il database. I dati ottenuti vengono poi inoltrati alla view precedentemente illustrata.

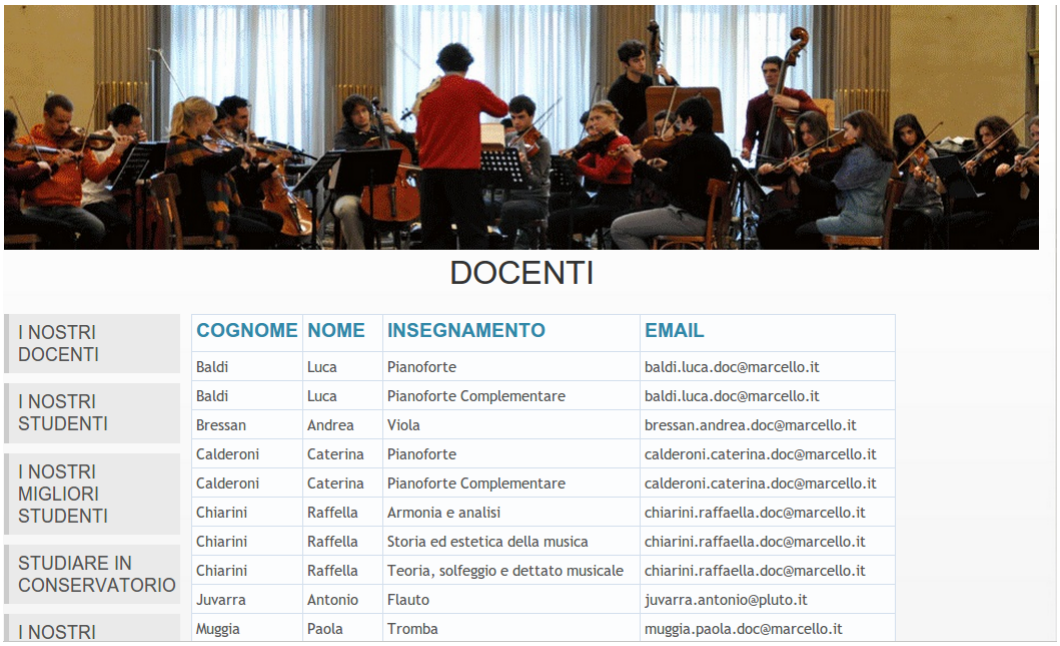
La classe visualizza_model.php si occupa del recupero di dati dal database e dell'invio degli stessi al controller che ne ha fatto richiesta. Di seguito viene citato una piccola parte di codice relativo al precedente esempio:

Codice 4.3: Visualizza_model.php'

```
1
2  <?php
3  class Visualizza_model extends CI_Model {
4
5    public function __construct()
6    {
7      $this->load->database();
8    }
9
10   public function get_elements($table)
11   {
12     $this->db->select('*');
13     $this->db->from(''.$table);
14     $query = $this->db->get();
15
16     return $query->result_array();
17   }
18
19   public function get_docente()
20   {
21
22     $this->db->select('docente.cognome, docente.nome, insegnamento.nome as inome, email');
23     $this->db->distinct();
24     $this->db->from('docente');
25     $this->db->join('eroga', 'docente.cf = eroga.cf_docente');
26     $this->db->join('insegnamento', 'eroga.id_insegnamento = insegnamento.id');
27     $this->db->order_by('docente.cognome, docente.nome, insegnamento.nome');
28
29     $query = $this->db->get();
30
31     return $query->result_array();
32   }
33
34 ...
35
36 }
```

Il metodo privato __construct() come prima è obbligatorio e serve per caricare le librerie utilizzate all'interno di questa classe. Il metodo get_elements($table) fornisce un supporto per il recupero di dati per una tabella generica, senza però fornire limitazioni, particolari ordini o dati combinati con altre tabelle. Il metodo get_docente() è specifico per quello che verrà poi visualizzato nella view dedicata: effettua una query al database recuperando così i dati relativi al nome, cognome del docente oltre che l'insegnamento da lui affrontato.

Il risultato finale che apparirà all'utente sarà una schermata simile a questa (i dati possono chiaramente variare):



DOCENTI

I NOSTRI DOCENTI

I NOSTRI STUDENTI

I NOSTRI MIGLIORI STUDENTI

STUDIARE IN CONSERVATORIO

I NOSTRI

COGNOME	NOME	INSEGNAMENTO	EMAIL
Baldi	Luca	Pianoforte	baldi.luca.doc@marcello.it
Baldi	Luca	Pianoforte Complementare	baldi.luca.doc@marcello.it
Bressan	Andrea	Viola	bressan.andrea.doc@marcello.it
Calderoni	Caterina	Pianoforte	calderoni.caterina.doc@marcello.it
Calderoni	Caterina	Pianoforte Complementare	calderoni.caterina.doc@marcello.it
Chiarini	Raffella	Armonia e analisi	chiarini.raffaella.doc@marcello.it
Chiarini	Raffella	Storia ed estetica della musica	chiarini.raffaella.doc@marcello.it
Chiarini	Raffella	Teoria, solfeggio e dettato musicale	chiarini.raffaella.doc@marcello.it
Juvarra	Antonio	Flauto	juvarra.antonio@pluto.it
Muggia	Paola	Tromba	muggia.paola.doc@marcello.it

Figura 4.2: Schermata applicazione Web

L'applicazione Web è stata caricata sul server del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova sfruttando lo spazio Web e l'utilizzo di Postgres concesso agli studenti del corso di Basi di Dati. Il sito è quindi consultabile presso l'indirizzo **wwwdb.dei.unipd.it/2012/valeriom/ci**.

Capitolo 5

Conclusioni

L'obiettivo del progetto è stato quello di progettare e realizzare un sito web per la gestione e la visualizzazione di dati archiviati in una base di dati relativa a un Conservatorio. Questo fatto permette di conoscere un framework php largamente utilizzato adottando il design pattern MVC: conoscenze, queste, quasi necessarie nel campo informatico dal momento che lo sviluppo di siti web volti a gestire i dati è di fondamentale importanza. La conoscenza pratica del pattern MVC è molto importante dato che è diffuso anche in molti altri campi dell'informatica, basti pensare a certe aziende che ne fanno uno dei loro paradigmi fissi[1]. L'implementazione fisica del progetto ha portato a un risultato buono anche se professionalmente incompleto e aperto a sviluppi futuri, infatti è possibile allargare le funzionalità per l'utente finale tra cui:

- possibilità di registrarsi nel sito;
- funzionalità di autenticazione;
- creazione degli appelli d'esame;
- possibilità per gli studenti di registrarsi ai vari appelli e per i docenti di crearli, chiuderli e modificarli;
- limitazione delle funzionalità in base al tipo di utente (admin, docente, studente o visitatore);
- aggiungere una più curata grafica con l'ausilio per esempio delle funzionalità offerte dal linguaggio Javascript

[1] https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html

Appendice A

Codice SQL

In informatica **SQL (Structured Query Language)** è un linguaggio standardizzato per database basati sul modello relazionale (RDBMS) che racchiude un insieme di linguaggi:

- **DDL (Data Definition Language)**
  Permette di creare, modificare o eliminare gli oggetti in un database ovvero agire sullo schema di database.
  i.e. i comandi CREATE TABLE.., DROP TABLE.. etc.

- **DML (Data Manipulation Language)**
  Consente di leggere, inserire, modificare o eliminare i dati in un database.
  i.e. i comandi INSERT INTO.., DELETE FROM.. etc.

- **DQL (Data Query Language)**
  Usato per creare query sui database e sui sistemi informativi da parte degli utenti. Serve per rendere possibile l'estrazione di informazioni dal database interrogando la base dei dati interfacciandosi dunque con l'utente e le sue richieste di servizio.
  i.e. i comandi SELECT.. FROM.. WHERE.. etc.

- **DCL (Data Control Language)**
  Utilizzato per fornire o revocare agli utenti i permessi necessari per poter utilizzare i comandi Data Manipulation Language (DML) e Data Definition Language (DDL), oltre agli stessi comandi DCL (che servono a loro volta a modificare i permessi su alcuni oggetti).
  i.e. i comandi GRANT.. ON.. TO.. etc.

Per il progetto in questione verrà utilizzato come DBMS PostgreSQL 8.3.8 [1] quindi il codice SQL che seguirà, necessario per l'implementazione fisica della base di dati, sarà ottimizzato per esso.

[1] http://www.postgresql.org/

A.1 Convenzioni sulla scelta degli ID

Di seguito vengono elencate le convenzioni utilizzate per gli identificatori:

- L'identificatore dell'anno accademico fa riferimento all"anno del primo semestre (i.e. 2012/2013 diventa 2012).
- L'identificatore di Insegnamento é il nome di questo piú un intero progressivo (che identifica l'anno) se ha durata pluriennale (i.e. Solfeggio viene suddiviso in Solfeggio1, Solfeggio2, etc..).
- L'identificatore di Dipartimento é il carattere "D"piú un numero progressivo intero (i.e. D1, D2, etc..).

A.2 Struttura

Codice A.1: Codice SQL struttura Database'

```
1  CREATE DOMAIN matricola AS integer CHECK (value>0);
2  CREATE DOMAIN sesso AS character(1) CHECK (value='M' OR value='F');
3  CREATE DOMAIN telefono AS integer CHECK (value>=0);
4
5  CREATE TABLE dipartimento
6  (
7    ID character varying(4) NOT NULL,
8    Nome character varying(60),
9    CONSTRAINT "pk_dipartimento" PRIMARY KEY (ID)
10 );
11
12
13 CREATE TABLE docente
14 (
15   CF character(16) NOT NULL,
16   Nome character varying(30) NOT NULL,
17   Cognome character varying(30) NOT NULL,
18   Sesso sesso NOT NULL,
19   Data_di_nascita date NOT NULL,
20   Indirizzo character varying(30) NOT NULL,
21   Citta character varying(25) NOT NULL,
22   CAP character(5) NOT NULL,
23   Telefono telefono NOT NULL,
24   Email character varying(50) NOT NULL,
25   CONSTRAINT "pk_docente_cf" PRIMARY KEY (CF)
26 );
27
28 CREATE TABLE anno_accademico
29 (
30   ID integer NOT NULL,
```

```
31    CONSTRAINT "pk_anno_accademico_id" PRIMARY KEY (ID),
32    CONSTRAINT "check_anno_accademico_id" CHECK (ID > 1940)
33  );
34
35  CREATE TABLE dirige
36  (
37    CF_docente character(16) NOT NULL,
38    ID_dipartimento character varying(4) NOT NULL,
39    ID_anno integer NOT NULL,
40    CONSTRAINT "pk_dirige"
41      PRIMARY KEY (CF_docente, ID_dipartimento, ID_anno),
42    CONSTRAINT "fk_dirige_cf_docente"
43      FOREIGN KEY (CF_docente)
44      REFERENCES docente (CF) MATCH SIMPLE
45      ON UPDATE CASCADE ON DELETE RESTRICT,
46    CONSTRAINT "fk_dirige_id_anno" FOREIGN KEY (ID_anno)
47      REFERENCES anno_accademico (ID) MATCH SIMPLE
48      ON UPDATE CASCADE ON DELETE RESTRICT,
49    CONSTRAINT "fk_dirige_id_dipartimento"
50      FOREIGN KEY (ID_dipartimento)
51      REFERENCES dipartimento (ID) MATCH SIMPLE
52      ON UPDATE CASCADE ON DELETE RESTRICT
53  );
54
55  CREATE TABLE insegnamento
56  (
57    ID character varying(50) NOT NULL,
58    Nome character varying(50),
59    ID_dipartimento character varying(4),
60    CONSTRAINT "pk_insegnamento_id" PRIMARY KEY (ID),
61    CONSTRAINT "fk_insegnamento_id_dipartimento"
62      FOREIGN KEY (ID_dipartimento)
63      REFERENCES dipartimento (ID) MATCH SIMPLE
64      ON UPDATE CASCADE ON DELETE RESTRICT
65  );
66
67  CREATE TABLE eroga
68  (
69    CF_docente character(16) NOT NULL,
70    ID_anno integer NOT NULL,
71    ID_insegnamento character varying(50) NOT NULL,
72    CONSTRAINT "pk_eroga"
73      PRIMARY KEY (CF_docente, ID_anno, ID_insegnamento),
74    CONSTRAINT "fk_eroga_cf_docente" FOREIGN KEY (CF_docente)
75      REFERENCES docente (CF) MATCH SIMPLE
76      ON UPDATE CASCADE ON DELETE RESTRICT,
77    CONSTRAINT "fk_eroga_id_anno" FOREIGN KEY (ID_anno)
78      REFERENCES anno_accademico (ID) MATCH SIMPLE
```

```
79      ON UPDATE CASCADE ON DELETE RESTRICT,
80    CONSTRAINT "fk_eroga_id_insegnamento"
81      FOREIGN KEY (ID_insegnamento)
82      REFERENCES insegnamento (ID) MATCH SIMPLE
83      ON UPDATE CASCADE ON DELETE RESTRICT
84  );
85
86  CREATE TABLE classe_principale
87  (
88    Nome character varying(50) NOT NULL,
89    Descrizione character varying(100),
90    CONSTRAINT "pk_classe_principale_nome" PRIMARY KEY (Nome)
91  );
92
93  CREATE TABLE prevede
94  (
95    ID_insegnamento character varying(50) NOT NULL,
96    Nome_classe_principale character varying(50) NOT NULL,
97    Anno_relativo smallint NOT NULL,
98    CONSTRAINT "pk_prevede"
99      PRIMARY KEY (ID_insegnamento, Nome_classe_principale),
100   CONSTRAINT "fk_prevede_id_insegnamento"
101     FOREIGN KEY (ID_insegnamento)
102     REFERENCES insegnamento (ID) MATCH SIMPLE
103     ON UPDATE CASCADE ON DELETE RESTRICT,
104   CONSTRAINT "fk_prevede_nome_classe_principale"
105     FOREIGN KEY (Nome_classe_principale)
106     REFERENCES classe_principale (Nome) MATCH SIMPLE
107     ON UPDATE CASCADE ON DELETE RESTRICT,
108   CONSTRAINT "check_anno_relativo"
109     CHECK (Anno_relativo >= 1 AND Anno_relativo <= 10)
110 );
111
112 CREATE TABLE studente
113 (
114   Matricola matricola NOT NULL,
115   Nome character varying(30) NOT NULL,
116   Cognome character varying(30) NOT NULL,
117   Sesso sesso NOT NULL,
118   Data_di_nascita date NOT NULL,
119   Indirizzo character varying(30) NOT NULL,
120   Citta character varying(30) NOT NULL,
121   CAP character(5) NOT NULL,
122   Telefono telefono NOT NULL,
123   Email character varying(50) NOT NULL,
124   CONSTRAINT "pk_studente_matricola" PRIMARY KEY (Matricola)
125 );
126
```

```
127 CREATE TABLE studente_ritirato
128 (
129   Matricola matricola NOT NULL,
130   CONSTRAINT "pk_studente_ritirato_id_studente"
131     PRIMARY KEY (Matricola),
132   CONSTRAINT "fk_studente_ritirato_id_studente"
133     FOREIGN KEY (Matricola)
134     REFERENCES studente (Matricola) MATCH SIMPLE
135     ON UPDATE CASCADE ON DELETE RESTRICT
136 );
137
138 CREATE TABLE frequenta
139 (
140   ID_insegnamento character varying(50) NOT NULL,
141   Matricola matricola NOT NULL,
142   ID_anno_accademico integer NOT NULL,
143   CONSTRAINT "pk_frequenta"
144     PRIMARY KEY (ID_insegnamento, Matricola,ID_anno_accademico),
145   CONSTRAINT "fk_frequenta_id_insegnamento"
146     FOREIGN KEY (ID_insegnamento)
147     REFERENCES insegnamento (ID) MATCH SIMPLE
148     ON UPDATE CASCADE ON DELETE RESTRICT,
149   CONSTRAINT "fk_frequenta_matricola" FOREIGN KEY (Matricola)
150     REFERENCES studente (Matricola) MATCH SIMPLE
151     ON UPDATE CASCADE ON DELETE RESTRICT,
152   CONSTRAINT "fk_frequenta_id_anno_accademico"
153     FOREIGN KEY (ID_anno_accademico)
154     REFERENCES anno_accademico (ID) MATCH SIMPLE
155     ON UPDATE CASCADE ON DELETE RESTRICT
156 );
157
158 CREATE TABLE esame
159 (
160   ID_insegnamento character varying(50) NOT NULL,
161   Matricola matricola NOT NULL,
162   Voto integer NOT NULL,
163   Lode boolean DEFAULT false,
164   Menzione boolean DEFAULT false,
165   Data_esame date NOT NULL,
166   CONSTRAINT "pk_esame"
167     PRIMARY KEY (ID_insegnamento, Matricola),
168   CONSTRAINT "fk_esame_id_insegnamento"
169     FOREIGN KEY (ID_insegnamento)
170     REFERENCES insegnamento (ID) MATCH SIMPLE
171     ON UPDATE CASCADE ON DELETE RESTRICT,
172   CONSTRAINT "fk_esame_matricola"
173     FOREIGN KEY (Matricola)
174     REFERENCES studente (Matricola) MATCH SIMPLE
```

```
175     ON UPDATE CASCADE ON DELETE RESTRICT,
176   CONSTRAINT "check_voto" CHECK (Voto >= 18 AND Voto <= 30)
177 );
178
179 CREATE TABLE iscritto_a
180 (
181   Matricola matricola NOT NULL,
182   Nome_classe_principale character varying(50) NOT NULL,
183   ID_anno_accademico integer NOT NULL,
184   Anno_iscrizione integer NOT NULL,
185   CONSTRAINT "pk_iscritto_a"
186     PRIMARY KEY (Matricola, Nome_classe_principale, ID_anno_accademico),
187   CONSTRAINT "fk_iscritto_a_id_anno_accademico"
188     FOREIGN KEY (ID_anno_accademico)
189     REFERENCES anno_accademico (ID) MATCH SIMPLE
190     ON UPDATE CASCADE ON DELETE RESTRICT,
191   CONSTRAINT "fk_iscritto_a_matricola"
192     FOREIGN KEY (Matricola)
193     REFERENCES studente (Matricola) MATCH SIMPLE
194     ON UPDATE CASCADE ON DELETE RESTRICT,
195   CONSTRAINT "fk_iscritto_a_nome_classe_principale"
196     FOREIGN KEY (Nome_classe_principale)
197     REFERENCES classe_principale (Nome) MATCH SIMPLE
198     ON UPDATE CASCADE ON DELETE RESTRICT,
199   CONSTRAINT "check_anno_iscrizione"
200     CHECK (Anno_iscrizione > 0 AND Anno_iscrizione < 11)
201 );
```

Bibliografia

[1] Ramez Elmasri, Shamkant B. Navathe : Sistemi di Basi di Dati Fondamenti, USA, (2011), (Pearson, Italia, 2011)

[2] Jennifer Niederst Robbins : HTML and XHTML, O'REILLY (2010)

[3] Ellislab : CodeIgniter User Guide Version 2.1.4, http://ellislab.com/codeigniter/user-guide/

[4] Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, Jakub Vrana and several others : PHP Manual, http://www.php.net/manual/en/

[5] Adam Griffiths : CodeIgniter 1.7 Professional Development, USA (2011)

[6] Carlo Fantozzi : Design Pattern, http://esp1213.wikispaces.com/file/view/Design Patterns.pdf/425372828/Design Patterns.pdf