Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in

Ingegneria Informatica

Progettazione e realizzazione di un'applicazione Web per la gestione degli eventi di un Conservatorio

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Davide Marchiori

Anno Accademico 2012/2013

Indice

Elenco delle figure

Elenco dei listati codice

Capitolo 1

Introduzione

La progettazione e la realizzazione di un'applicazione Web per la gestione di un database in generale si può suddividere in due fasi principali:

- La progettazione e la realizzazione della **base di dati** che andrà a rappresentare, e quindi anche a gestire, la realtà di interesse che vogliamo considerare.
- La progettazione e la realizzazione dell'**applicazione Web**, sotto forma di sito Web dinamico, che permetta l'accesso alle informazioni presenti nella nostra base di dati collocata in un Web Server mediante un client remoto.

La prima parte del progetto consisterà quindi nell'affrontare tutte le problematiche relative alla progettazione di una base di dati con un database di tipo relazionale. Essa passerà da una progettazione di tipo concettuale ad una di tipo logico e infine ad un'implementazione fisica attraverso l'uso del linguaggio SQL. La seconda parte consisterà nella realizzazione di un'applicazione Web che abbia il compito principale di mettere a disposizione alcune informazioni, che possono essere di dominio pubblico, prelevate dal database a tutti i visitatori e altri tipi di informazione rivolti esclusivamente a particolari categorie di utenza. L'applicazione dovrà garantire l'accesso ad informazioni diverse a seconda della categoria alla quale l'utente che le richiede appartiene. Prima di tutto dovrà garantire sicurezza contro l'accesso non autorizzato ad alcuni dati da parte di normali utenti, come ad esempio la visualizzazione di particolari dati personali relativi agli studenti iscritti al Conservatorio, e invece consentire alcune operazioni al database da parte di altri 'super-utenti' chiamati 'Amministratori', come ad esempio l'inserimento di tuple a particolari tabelle. Come seconda cosa dovrà garantire l'assoluta integrità di tutti i dati presenti nel database e quindi di conseguenza a quelli che verranno visualizzati ed a quelli inseriti dagli Amministratori con permessi adeguati. L'applicazione Web verrà implementata utilizzando un framework in PHP chiamato CodeIgniter [1].

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

Capitolo 2

Descrizione del progetto

Nell'ambito della progettazione delle basi di dati, nel corso degli anni, si è consolidata una metodologia di progettazione articolata in tre fasi principali da effettuare in cascata. Ogni fase si riferisce a un livello di astrazione nella rappresentazione dei dati e delle relazioni tra essi. Lo scopo è quello di separare le attività di risoluzione dei problemi e di garantire la possibilità di modificare delle soluzioni adottate ai livelli inferiori senza dover riprogettare quanto definito in precedenza nei livelli superiori. A ciascuna fase di progettazione corrispondono diversi modelli per la rappresentazione dei dati, ovvero tecniche per la rappresentazione degli aspetti rilevanti della realtà da modellare, definite da strumenti e vincoli specifici. La rappresentazione generata seguendo le regole del modello viene definita schema. Le fasi riconosciute fondamentali nella progettazione di un database sono le seguenti:

- Progettazione **Concettuale**
- Progettazione **Logica**
- Progettazione **Fisica**

In questo capitolo sarà analizzato lo stato preparatorio di **raccolta e analisi dei requisiti** che consiste nell'individuazione e nello studio delle proprietà e delle funzionalità che il sistema informativo dovrà avere. Questa prima analisi richiede un'interazione con gli utenti del sistema e produce una descrizione completa, ma generalmente informale, dei dati coinvolti.

Per il progetto si vuole realizzare un database relativo alla struttura organizzativa della gestione degli Eventi promossi dal Conservatorio e da Associazioni Esterne che collaborano con lo stesso, facendo distinzione tra Saggi Interni e Concerti Esterni. La base di dati dovrà consentire, inoltre, la gestione delle partecipazioni ad essi da parte degli studenti iscritti, la gestione degli studenti ritirati e permettere l'amministrazione dei vari dipartimenti musicali presenti.

Nel capitolo vedremo:

- **I requisiti strutturali** che andranno a descrivere, in un linguaggio informale, tutte le entità che faranno parte del database con le relative proprietà possedute.

- Un **glossario** che va a riassumere le principali entità con una breve descrizione, i sinonimi con cui ci si può riferire ad esse e i collegamenti diretti con le altre.

2.1 Requisiti Strutturali

Frasi per Studente
Ogni studente viene registrato qualora esso abbia effettuato almeno un'iscrizione al Conservatorio. 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. Viene mantenuta, inoltre, la lista degli eventi a cui ha ufficialmente partecipato. Si vogliono anche conservare i dati sugli studenti ritirati.

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 e può organizzare dei saggi interni al Conservatorio. 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). Ogni docente può essere direttore di al più un dipartimento all'anno di cui i propri insegnamenti fanno parte.

Frasi per Associazione
Ogni associazione può proporre al Conservatorio degli eventi musicali (concerti) per determinate occasioni. Di esse si mantiene il nome, alcuni recapiti (e-mail, telefono, fax), l'indirizzo della sede, la città, il CAP e la partita IVA (attributo chiave).

Frasi per Evento
Ogni evento a cui il Conservatorio ha partecipato può essere di due tipologie differenti: Saggio interno o Concerto esterno.

- Saggio interno: concerto proposto da un solo dipartimento del Conservatorio e che si svolge in ambienti interni alla struttura fisica del Conservatorio stesso.

- Concerto esterno: evento proposto da un'associazione esterna al Conservatorio e che si svolge in ambienti esterni alla struttura del Conservatorio stesso.

Per ogni evento si conservano le seguenti informazioni: breve descrizione (i.e. In occasione di qualche particolare ricorrenza), luogo di svolgimento, data e ora di svolgimento ed un identificatore (attributo chiave).

2.2 Glossario

Termine	Descrizione	Sinonimi	Collegamenti
Studente	Chi studia al Conservatorio	Allievo	Evento
Docente	Chi insegna al Conservatorio	Insegnante, Maestro	Dipartimento
Dipartimento	Collezione di insegnamenti affini		Docente, Evento
Evento	Evento musicale che coinvolge studenti del Conservatorio	Saggio, Concerto	Studente, Associazione, Dipartimento
Associazione	Ente esterno che promuove eventi musicali	Società	Evento

Capitolo 3

Progettazione e realizzazione del Database

3.1 Progettazione Concettuale

Un modello concettuale rappresenta concetti, ossia entità e relazioni tra queste, a differenza di un modello mentale che descrive idee di un certo dominio del problema. La modellazione o progettazione concettuale è una tecnica molto nota di progettazione dati, assieme alla progettazione logica e alla progettazione fisica.

Ha le seguenti qualità e caratteristiche:

- È in grado di rappresentare i dati della realtà d'interesse in termini di un modello (descrizione) formale, ad **alto livello**;
- È **indipendente** dal DBMS che verrà utilizzato.

Deve essere per definizione indipendente dai dettagli dell'implementazione, come la concorrenza o la memorizzazione dei dati. Ha lo scopo di esprimere il significato di termini e concetti usati dagli esperti del dominio per discutere il problema, e di trovare le giuste relazioni tra concetti differenti. Questo modello è anche chiamato modello semantico: cerca di chiarire il significato di vari termini spesso ambigui e assicura che non ci siano problemi con una differente interpretazione di termini e concetti. Questo perché tali interpretazioni possono portare a degli errori nel progetto software basato su di essi.

Vedremo quindi nella sezione 3.1:

- La rappresentazione del **modello Entità-Associazione** (E-R);
- Una sua **ristrutturazione** che serve a preparare lo schema per una conversione diretta ad uno modello logico secondo determinate regole;
- Un **Dizionario dei dati** dove si descrivono le varie Entità e Associazioni con i relativi attributi;

- Le **regole di Vincolo** che altrimenti non sarebbero direttamente intuibili guardando solamente lo Schema Concettuale.

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:

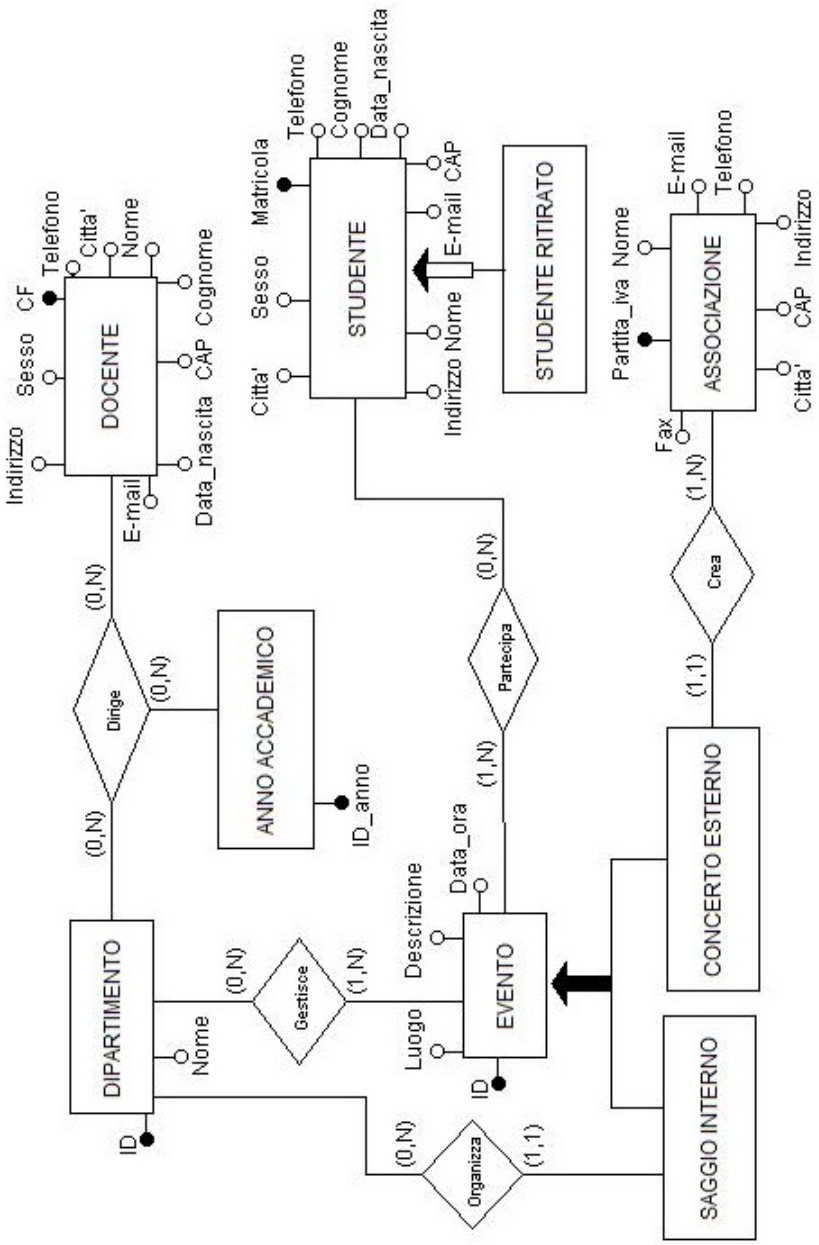

Figura 3.1: Modello Concettuale - Schema E-R

3.1.2 Ristrutturazione schema E-R

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). "Saggio interno" e "Concerto esterno" sono state inglobate nell'entità padre "Evento", aggiungendo l'attributo "Saggio_interno" di tipo boolean dal momento che la generalizzazione era totale:

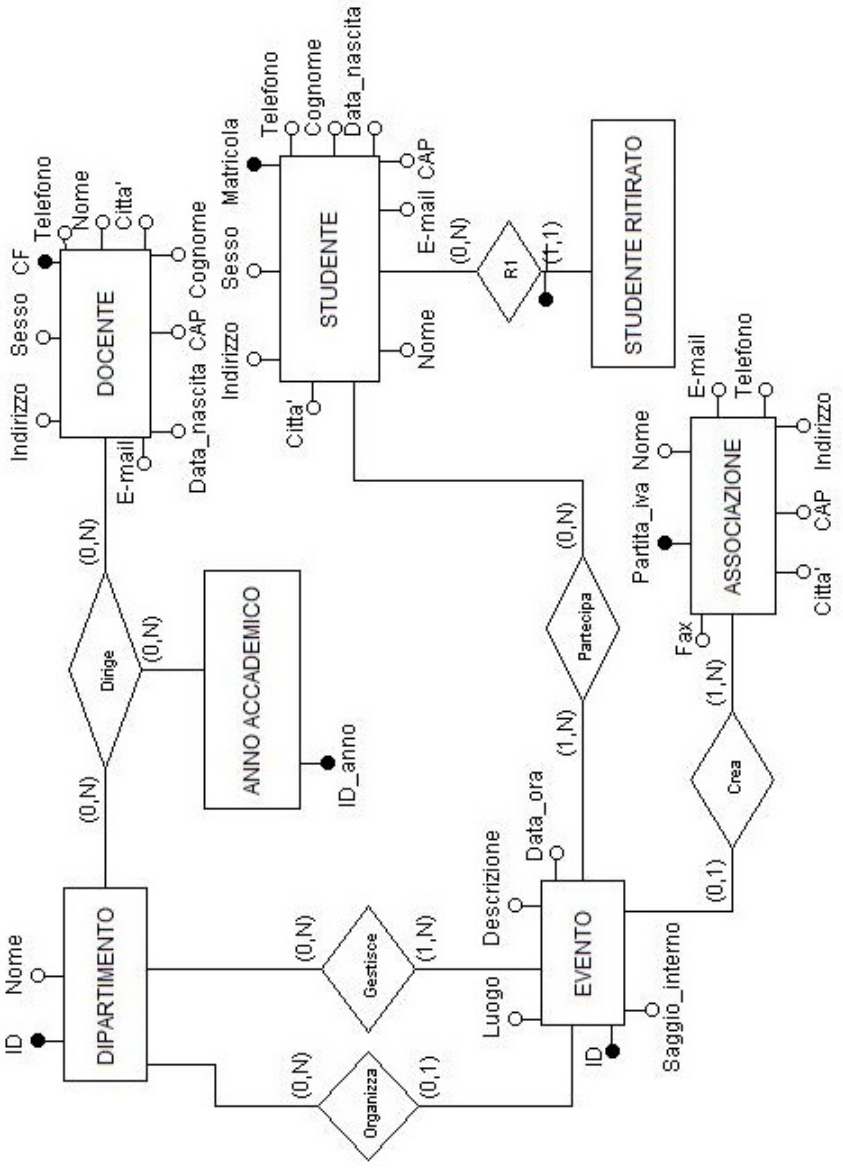

Figura 3.2: Modello Entità-Associazione (E-R) ristrutturato.

3.1.3 Dizionario dei dati

Entità

Entità	Descrizione	Attributi	Identificatore
Studente	Si iscrive al conservatorio, partecipa ad eventi	Matricola, Nome, Cognome, Sesso, Data_di_nascita, Indirizzo, Città, CAP, Telefono, E-mail	Matricola
Studente Ritirato	Studente che non ha finito il percorso di studi		Matricola
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	Gestisce e organizza concerti, contenitore di materie affini	ID, Nome	ID
Evento	Concerto o Saggio	ID, Descrizione, Luogo, Data_ora, Saggio_interno	ID
Associazione	Ente esterno che propone eventi	Partita_iva, Nome, Indirizzo, Città, CAP, Telefono, Fax, E-mail	Partita_iva

Associazioni

Associazione	Attributi	Entità Collegate
R1		Studente (0,N), Studente Ritirato (1,1)
Dirige		Docente (0,N), Dipartimento (0,N), Anno Accademico(0,N)
Gestisce		Dipartimento (0,N), Evento(1,N)
Organizza		Dipartimento (0,N), Evento (0,1)
Crea		Evento (0,1), Associazione (1,N)
Partecipa		Evento (1,N), Studente (0,N)

3.1.4 Schema Concettuale, Regole di vincolo

Per rispettare la rappresentazione fedele della realtà di interesse descritta in sezione 2.1 è necessario definire delle regole di vincolo per concetti altrimenti non esprimibili utilizzando il modello E-R.

- **Regola di Vincolo 1**
  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 2**
  Un'associazione non può creare saggi interni.
- **Regola di Vincolo 3**
  Un dipartimento non può organizzare concerti esterni.

3.2 Progettazione Logica

Il modello relazionale è un modello logico di rappresentazione o strutturazione dei dati di un database implementato su sistemi di gestione di basi di dati (DBMS), detti perciò sistemi di gestione di basi di dati relazionali (RDBMS). Si basa sulla teoria degli insiemi e sulla logica del primo ordine ed è strutturato intorno al concetto matematico di relazione (detta anche tabella). Per il suo trattamento ci si avvale di strumenti quali il calcolo relazionale e l'algebra relazionale.

Presenta le seguenti caratterisiche:

- Consiste nella traduzione dello schema concettuale in termini di un determinato modello logico (ad esempio il modello relazionale) di dati usato dal DBMS che si intende utilizzare.
- Include l'ottimizzazione della rappresentazione in funzione delle operazioni eseguite come ad esempio la **normalizzazione**.

Consiste quindi nel secondo grande passo della progettazione di una base di dati e utilizza il modello concettuale ristrutturato in precedenza per creare appunto uno schema relazionale. Esiste un insieme di regole che ne permette la traduzione in maniera quasi automatica ed efficiente e che evita i più comuni problemi che si verrebbero a creare in alcuni casi. Ad esempio i più frequenti:

- La **ridondanza** dei dati
- L'**inconsistenza** dei dati

Bisogna però notare che la **ridondanza dei dati**, che normalmente consiste in uno spreco di spazio, alcune volte può essere inserita apposta dal progettista per ottimizzare il tempo dell'esecuzione di particolari operazioni sulla base di dati (come ad esempio l'esecuzione frequente di particolari query che richiedono dati presenti in molte tabelle).

Nella sezione 3.2 vedremo:

- Il **modello logico** prodotto per il progetto;
- Un insieme di **regole di vincolo** dello schema relazionale che altrimenti non sarebbero intuibili solamente guardandolo.

3.2.1 Modello Logico: Relazionale

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

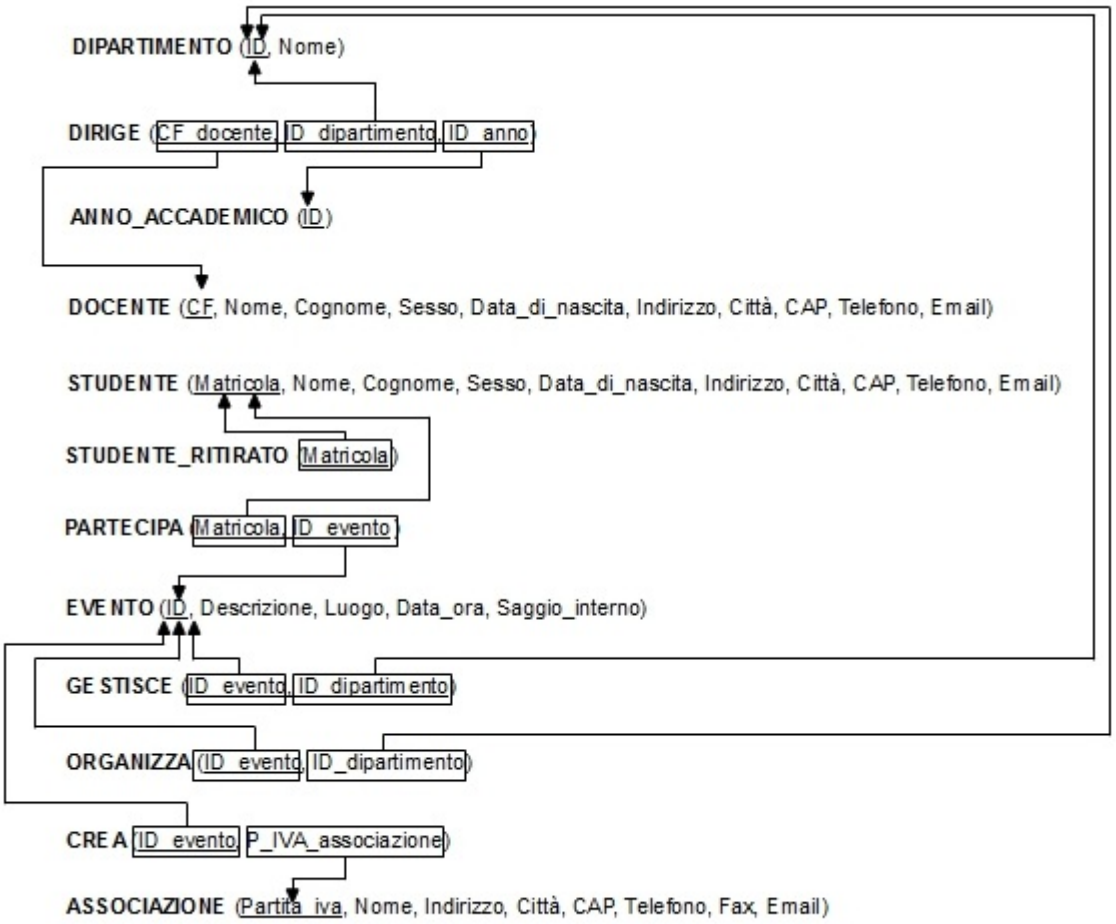

Figura 3.3: Modello Relazionale - Schema Logico

3.2.2 Schema Logico: Regole di Vincolo

- **RV1**: Gli attributi di Studente non devono essere nulli.
- **RV2**: Gli attributi di Docente non devono essere nulli.
- **RV3**: Gli attributi di Associazione non devono essere nulli, tranne gli attributi "e-mail" e "fax" che sono opzionali.
- **RV4**: Gli attributi di Evento non devono essere nulli.
- **RV5**: Gli attributi di Dipartimento non devono essere nulli.
- **RV6**: Un Evento deve partecipare ad almeno un'istanza di Partecipa.
- **RV7**: Un'Associazione deve partecipare ad almeno un'istanza di Crea.
- **RV8**: Un Evento deve partecipare ad almeno un'istanza di Gestisce.
- **RV9**: Un Evento, se è Saggio interno, deve partecipare a solo un'istanza di Organizza.
- **RV10**: Un Evento, se è Concerto esterno, deve partecipare a solo un'istanza di Crea.

Capitolo 4

Analisi del Framework e software utilizzato

Un framework è una struttura di supporto sulla quale un software può essere organizzato e progettato. Solitamente è composto da librerie di codice, scritte in vari linguaggi di programmazione, che permettono un supporto allo sviluppo del software. Consiste dunque in un insieme di strumenti volti all'aumentare la velocità di sviluppo del prodotto finito e al permettere al programmatore di concentrarsi sulle vere funzionalità dell'applicazione. Quindi possiamo dire che concretamente il suo scopo principale è quello di risparmiare allo sviluppatore la riscrittura di codice già steso per compiti simili. Nella pratica si presenta come un software che comprende in un unico pacchetto l'intero set di librerie e strumenti utili alla progettazione di un sito, un'applicazione Web o qualsiasi altro servizio online sia necessario creare con PHP o con un altro linguaggio di programmazione.

Nel capitolo vedremo:

- Un'introduzione sui **linguaggi** che sono stati utilizzati per lo sviluppo concreto dell'applicazione Web;
- Una descrizione sul funzionamento del framework **CodeIgniter**;
- Una spiegazione dell'**approccio MVC**;
- Un **esempio** di interazione MVC in CodeIgniter.

4.1 Applicazioni Web: linguaggi

Nell'ambito della realizzazione di applicazioni Web possono essere utilizzati più linguaggi a seconda delle funzionalità che si vogliono ottenere. Quelli che verranno utilizzati per la realizzazione della seconda parte del progetto saranno l'HTML e il PHP che permettereanno la creazione e la gestione di pagine web dinamiche.

L'HTML non è un linguaggio di programmazione (in quanto non prevede alcuna definizione di variabili, strutture dati, funzioni, strutture di controllo), ma

un linguaggio di formattazione che descrive le modalità di impaginazione o visualizzazione grafica (layout) del contenuto, testuale e non, di una pagina web mediante l'utilizzo di tag (parole chiave che descrivono un oggetto rendendone possibile la classificazione). In particolare, quindi, consiste nell'inserimento all'interno del testo di marcatori o etichette, chiamati appunto tag, che ne possono descrivere tra le altre cose la funzione, il colore, le dimensioni, la posizione relativa all'interno della pagina o altre caratteristiche. Inoltre, l'HTML supporta l'inserimento di script e oggetti esterni quali immagini o filmati. Il contenuto delle pagine web solitamente consiste di un documento HTML e dei file ad esso correlati che un web browser (i.e. Google Chrome[1], Mozilla Firefox[2], Safari[3], Internet Explorer[4], Opera[5]) scarica da uno o più web server per elaborarli, interpretando il codice sorgente, al fine di generare la visualizzazione, sullo schermo del computer, della pagina desiderata.

PHP (acronimo di PHP: Hypertext Preprocessor, originariamente acronimo di Personal Home Page) è un un linguaggio di programmazione interpretato, concepito per la programmazione Web, ovvero la realizzazione di pagine web dinamiche. Dalla sua nascita nel 1994, trattandosi di un progetto open source, programmatori di tutto il mondo contribuirono alla sua crescita fino ad arrivare alla versione PHP 3.0 che prevedeva un'ottima versabilità verso il database MySQL[6] e con il web server Apache[7]. Nello stesso anno il PHP 3.0 era presente su circa il 10% dei server web presenti su Internet. Ad oggi siamo arrivati ad un versione di PHP 5.0 che è quella che verrà utilizzata per la realizzazione di questo progetto. PHP è in grado di interfacciarsi a molti database tra cui MySQL, PostgreSQL [8], Oracle [9], Microsoft SQL Server[10], solo per citarne alcuni. Come detto in precedenza, quindi, il PHP è utilizzato per la creazione di pagine web dinamiche.

Una pagina web **dinamica** è una pagina web il cui contenuto, in tutto o in parte, è generato sul momento dal server, potendo dunque essere diversa ogni volta che viene richiamata, consentendo un'interattività con l'utente. Nel caso si abbia una pagina web **statica** (pagina scritta esclusivamente in HTML), nel momento in cui il browser di un client richiede al server una pagina HTML, il server la prende e la spedisce così com'è, insieme ad eventuali file allegati (ad esempio immagini), permettendo così all'utente di visualizzarla correttamente. Supponiamo ora che l'utente richieda invece una pagina PHP e questa, contrariamente a quella di prima, non contenga solo codice HTML, ma anche codice PHP. In questo caso il server, prima di spedire la pagina, esegue il codice PHP che in

[1]https://www.google.com/intl/it/chrome/browser/
[2]http://www.mozilla.org/it/firefox/new/
[3]http://www.apple.com/it/safari/
[4]http://windows.microsoft.com/it-it/internet-explorer/download-ie
[5]http://www.opera.com/it
[6]http://www.mysql.it/
[7]http://httpd.apache.org/
[8]http://www.postgresql.org/
[9]http://www.oracle.com/us/products/database/overview/index.html
[10]http://www.microsoft.com/en-us/sqlserver/default.aspx

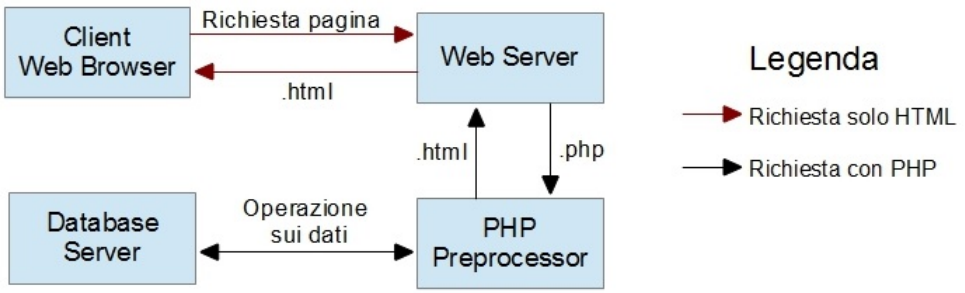

Figura 4.1: Schema di funzionamento del PHP

genere produce altro codice HTML. Dopo l'esecuzione, la pagina non conterrà più codice PHP, ma solo HTML. A questo punto è pronta per essere spedita al browser del client. (Ovviamente, il file che non contiene più codice PHP non è quello 'originale', ma la 'copia' che viene spedita al browser. L'originale rimane disponibile per le prossime richieste). Quindi l'utente vede solo il codice HTML, e non ha accesso al codice PHP che ha generato la pagina.

Questo aspetto è molto importante in quanto permette:

- Una **maggiore sicurezza** del codice non permettendo l'accesso diretto dell'utente alla pagina presente nel server;
- Un **dinamismo** in quando la stessa pagina web cambia anche a seconda dei momenti in cui viene richiesta;

  Ad esempio, nel nostro caso, la pagina web che si occupa di visualizzare l'elenco degli studenti varia a seconda dello stato del database al momento della richiesta al Web Server.
- L'esecuzione di parti di codice SQL alla base di dati direttamente dal server e quindi, se correttamente programmato, consente la **gestione del database** da un **client remoto**.

4.2 CodeIgniter

CodeIgniter è un framework che mette a disposizione un ricco insieme di librerie che aiutano il programmatore a risolvere i compiti più comuni che si presentano durante la scrittura di un codice sorgente in linguaggio PHP. L'obiettivo principale è quello di evitare di ripetere molte volte lo stesso codice per un'operazione che si ripresenta durante la programmazione, contribuendo così a velocizzare il lavoro e rendere più leggibile e pulito il codice finale. CodeIgniter permette di

focalizzare la propria creatività sul progetto, minimizzando il codice necessario per un determinato task.

Le caratteristiche più importanti che hanno portato alla scelta di questo framework rispetto ad altri sono:

- **Gratuito**
  CodeIgniter è licenziato sotto Apache/BSD-style open source license[11] così è possibile utilizzarlo comunque si desideri.

- **Leggero e Veloce**
  Il cuore del sistema richiede solo alcune librerie. Questo è in contrasto con molti framework che richiedono significative risorse. Librerie aggiuntive sono caricate solo a richiesta, in base alle necessità; questo permette di avere un sistema di base molto snello e veloce.

- **Approccio MVC**
  CodeIgniter usa l'approccio Model-View-Controller, il quale permette una forte separazione tra la parte logica dell'applicazione e quella dell'interfaccia. Questo è particolarmente utile in progetti che lavorano con template dove la presenza di codice è ridotta al minimo. L'approccio MVC verrà spiegato più in dettaglio nella sezione 4.3.

- **Genera URL Segment based**
  Gli URLs generati da CodeIgniter sono chiari e adatti per i motori di ricerca. Invece di usare l'approccio standard query string per URLs creati dinamicamente, CodeIgniter utiliza l'approccio segment-based:
  Ad esempio: http://www.example.com/notizie/articolo/345.

- **Ampia presenza di librerie**
  Mette a disposizione un'ampia serie di librerie che facilitano lo sviluppatore nella realizzazione dei task più comuni, come avere accesso ai database, spedire email, validare i dati di un form, gestire le sessioni, manipolare immagini e molto altro.

- **Estensibile**
  Il sistema può essere facilmente esteso attraverso l'uso di librerie plugin e helper, oppure attraverso l'estensione di classi o system hooks.

- **Non richiede un Template Engine**
  Un template engine è una libreria che consente di agevolare il lavoro di progettazione di un'applicazione Web permettendo l'uso di parole chiave al posto di particolari comandi in linguaggio PHP nativo sintatticamente più lunghi. Il problema dell'utilizzo di tale libreria è la possibile riduzione delle performance dell'applicazione in quanto il codice deve comunque essere riconvertito in PHP per funzionare. Proprio per questo motivo CodeIgniter, che si predisponde l'obiettivo di proporre un framework leggero e veloce, non obbliga il programmatore ad usarne alcuno.

[11] http://ellislab.com/codeigniter/user-guide/license.html

- **Completamente Documentato**
  E' presente una documentazione chiara e molto esaudiente con molti esempi pratici.

- **Importante Community di Utenti**
  C'è a disposizione una vasta comunità di utenti attivi partecipanti ai forum, sia italiano[12] che inglese[13].

4.3 L'approccio MVC

CodeIgniter utilizza l'approccio MVC (Model-View-Controller) che consente un ampio livello di separazione tra la logica dell'applicazione e la presentazione della stessa. L'utilizzo dell'MVC si rifletterà positivamente soprattutto nei progetti in cui sarà necessario che il lavoro dei Web designer non condizioni quello degli sviluppatori e viceversa.

L'approccio MVC è strutturato sulla base dei tre elementi fondamentali che ne compongono il nome:

- **Model**
  Gestione dei dati: mette a disposizione i metodi con cui accedere ai dati necessari per il funzionamento dell'applicazione.
  Un esempio molto importante delle funzioni del Model sono le interazioni con il database come: interrogazioni, inserimenti, aggiornamenti o cancellazioni.

- **View**
  Gestione dell'interfaccia: ha il compito di visualizzare i dati forniti dal Model e permette l'interazione tra utilizzatori e applicazione.
  Questo elemento dell'approccio MVC è l'unico che l'utente finale vede realmente in quanto è l'interfaccia che si interpone tra l'utente e il controller.

- **Controller**
  Gestione delle richieste provenienti dagli utenti: ad esso vengono inviate le istruzioni provenienti dall'utenza (mediati dalla view) e le esegue condizionando lo stato dei due componenti presentati in precedenza.
  Il Controller gestisce la parte delle richieste, le elabora e le indirizza verso l'oggetto destinatario. È il vero responsabile delle azioni degli utenti.
  Sulla base della richiesta di un URL proveniente dall'utente decide quale modello utilizzare nonché se e quale view mostrare.

Ecco come avvengono le interazioni tra i diversi componenti: il Controllore determina il tipo di richiesta che gli arriva, la quale viene usata dal Modello per manipolare i dati e quelli risultanti dall'elaborazione vengono poi passati

[12] http://www.codeigniteritalia.it/forum
[13] http://ellislab.com/forums/

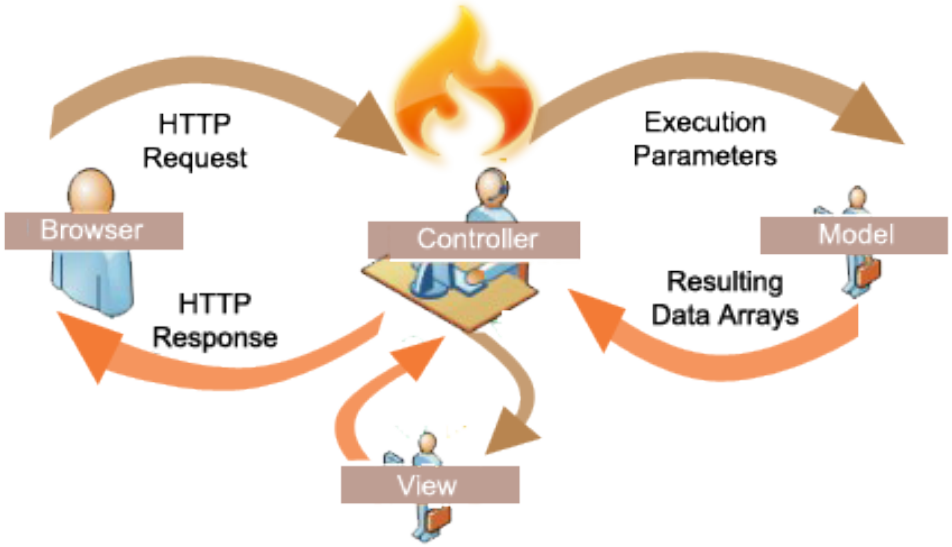

Figura 4.2: Approccio Model-View-Controller

alla Vista. Il Controllore non espone i dati nel Modello, comanda i suoi metodi cambiando i dati, e poi li passa alla Vista che li espone. In questo modo si divide il layout dal codice, che ha il compito di svolgere determinate azioni, rendendo il sito più flessibile e potente.

L'errore più comune che si compie quando si utilizza un framework come CodeIgniter è che i diversi ruoli che il sistema MVC affida ai suoi tre componenti non vengano rispettati.

Per tale ragione alcuni errori frequenti sono:

- **Inserire markup (html) nel Model o nel Controller:**
  l'interfaccia grafica deve essere affidata solo alle Viste.
- **Gestire i dati attraverso i Controller** (o peggio ancora nelle view):
  Il Controllore deve occuparsi esclusivamente di interpretare la richiesta effettuata dall'utente tramite l'URL andando a prelevare i dati necessari tramite gli opportuni Modelli e selezionare le Viste corrispondenti.

4.3.1 Vantaggi dell'MVC

I vantaggi che si possono ottenere dall'utilizzo dell'approccio MVC sono:

- Modifiche più rapide perchè si può cambiare facilmente una particolare richiesta al database agendo solo sul Modello appropriato, o si può cambiare la grafica di una pagina agendo solamente sulla Vista.
- Si seguono degli standard di programmazione che consentono ad altri sviluppatori di comprendere facilmente il codice.

- Possibilità di sviluppare un'applicazione modulare.
- Software più flessibile, manutenibile, aggiornabile nel tempo.
- Costi di sviluppo ridotti, grazie alla separazione dei compiti e alla possibilità di operare su un singolo componente in modo indipendente rispetto agli altri.

4.4 Interazione MVC in CodeIgniter

Per fare un esempio pratico di quanto detto finora, prendiamo in considerazione l'applicazione Web sviluppata per questo progetto. Viene descritto il codice sorgente degli elementi che svolgono un'operazione di lettura dal database e si vedrà come avvengono nella pratica le interazioni tra modelli, viste e controllori. L'operazione che viene analizzata è quella che visualizza un elenco degli studenti iscritti al Conservatorio. Tale operazione è concessa a tutti i tipi di utente.

Il codice PHP che segue è quello relativo al modello 'visualizzazione_model':

Codice 4.1: Modello 'visualizzazione_model'

```
1
2  <?php
3  class Visualizzazione_model extends CI_Model
4  {
5    public function  __construct()
6    {
7      parent::__construct();
8      $this->load->database();
9    }
10
11   public function studenti($limit, $offset, $ord, $cognome)
12   {
13     $this->db->select('matricola, nome, cognome, email');
14     $this->db->like('cognome', ucfirst(strtolower($cognome)), 'after');
15     $this->db->order_by($ord);
16     $this->db->limit($limit, $offset);
17     $query = $this->db->get('studente');
18     return $query->result_array();
19   }
20
21   ...
22 }
```

Il modello 'visualizzazione_model' è incaricato di interagire con il database eseguendo le interrogazioni di lettura dei dati. Ha più metodi differenziati tra loro, ognuno incaricato di eseguire una diversa interrogazione (query) della base di dati. Il metodo che si prende in considerazione in questo esempio è chiamato 'studenti'. Tale metodo esegue una selezione dei dati dalla tabella 'studente' presente nel database, prendendo solamente la matricola, il nome, il cognome e l'email e restituendo un array con tutti i dati estratti. Si può notare che l'interrogazione non è scritta in linguaggio SQL ma attraverso l'utilizzo di diverse funzioni messe a disposizione da CodeIgniter che semplificano la lettura del codice. Il framework, comunque, non obbliga assolutamente lo sviluppatore ad utilizzare

tali funzioni in quanto consente anche di definire una stringa in cui scrivere l'intera interrogazione in codice SQL, come se si stesse scrivendo direttamente da un terminale del DBMS, e di poi eseguirla. Questo è utile in quando alcune interrogazioni posso essere troppo complicate da scrivere mediante i metodi messi a disposizione da CodeIgniter. I parametri che vengono passati al metodo 'studenti' del modello sono necessari per ordinare i risultati o ricercare solamente particolari studenti in base al cognome, in modo tale da applicare un filtro a quelli visualizzati. I parametri 'limit' e 'offset' sono invece utilizzati per sfruttare la libreria di paginazione messa a disposizione dal framework. Tale libreria serve a limitare il numero di risultati visualizzati creando automaticamente delle pagine che permettono di visualizzarli evitando così di creare pagine con tabelle troppo lunghe.

La funzione del modello per essere eseguita deve essere necessariamente chiamata da un controllore. In questo caso il controllore incaricato è chiamato 'visualizzazione' e ne viene di seguito illustrato il codice:

Codice 4.2: Controllore 'visualizzazione'

```
1
2  <?php
3  class Visualizzazione extends CI_Controller
4  {
5    public function __construct()
6    {
7      parent::__construct();
8      $this->load->model('visualizzazione_model');
9    }
10
11   public function studenti($ord, $offset='')
12   {
13     ...
14     $data['names'] = $this->visualizzazione_model->studenti
15         ($limit, $offset, $ord, ucfirst(strtolower($object)));
16     $data['tipo_ord'] = $ord;
17     $page='login';
18     if ($this->session->userdata('logged'))
19       $page = 'logout';
20     $this->load->view('templates_cons/header');
21     $this->load->view('templates_cons/'.$page);
22     $this->load->view('templates_cons/menu');
23     if ($this->session->userdata('logged'))
24       $this->load->view('templates_cons/menu_admin');
25     $this->load->view('templates_cons/sub_menu');
26     $this->load->view('visualizzazione/v_studenti', $data);
27     $this->load->view('templates_cons/footer');
28   }
29
30   ...
31 }
```

Anche il controllore, come il modello, ha più metodi programmati per eseguire diverse operazioni. Si può notare che al momento della creazione viene esteso il costruttore permettendo il caricamento del modello 'visualizzazione_model', descritto in precedenza. È stata presa questa scelta di progettazone in modo tale da non caricare ogni volta lo stesso modello per ogni metodo chiamato. Viene caricato una volta sola (alla creazione) e poi viene utilizzato sicuramente da qualsiasi metodo venga richiamato del controllore 'visualizzazione'. Il metodo 'studenti'

ha il compito di gestire i dati che ottiene chiamando il modello, caricare le viste necessarie alla creazione della pagina Web e passare i risultati sottoforma di variabili. Inizialmente prende l'array di risultati passati dal modello, chiamandolo 'names', e poi carica tutte le viste necessarie passando a 'v_studenti' lo stesso array con tutti i dati degli studenti iscritti. Vengono caricate più viste una di seguito all'altra in modo da concatenare i vari codici contenuti ed ottenere la pagina completa. Nel codice del controllore 'studenti' vengono fatti anche alcuni controlli in quanto certi menù ed alcuni componenti grafici sono riservati esclusivamente ad utenti autenticati come amministratori. Sono state tralasciate alcune righe di codice presenti nella versione completa dell'applicazione Web come ad esempio quelli necessari per far funzionare correttamente la paginazione di CodeIgniter. La vista che va infine a visualizzare questi risultati è 'v_studenti' di cui viene mostrato di seguito il codice:

Codice 4.3: Vista 'v_studenti'

```
1
2  ...
3
4    <table width="750" id="sizeOptions">
5    <tr>
6      <td><b><?php echo anchor
7          ('visualizzazione/studenti/matricola', 'MATRICOLA','') ?></b></td>
8      <td><b><?php echo anchor
9          ('visualizzazione/studenti/nome', 'NOME','') ?></b></td>
10     <td><b><?php echo anchor
11         ('visualizzazione/studenti/cognome', 'COGNOME','') ?></b></td>
12     <td><b><?php echo anchor
13         ('visualizzazione/studenti/email', 'E-MAIL','') ?></b></td>
14   </tr>
15   <?php foreach ($names as $data_item): ?>
16       <tr>
17       <td><?php echo $data_item['matricola'] ?></td>
18       <td><?php echo $data_item['nome'] ?> </td>
19       <td><?php echo $data_item['cognome'] ?> </td>
20       <td><?php echo $data_item['email'] ?> </td>
21       </tr>
22   <?php endforeach ?>
23   </table><br />
24 </div>
25
26 ...
```

Analizzando solamente la porzione di codice che si occupa di gestire i dati passati dal controllore, si può vedere che CodeIgniter permette alle viste di trattare gli array passati come se fossero delle variabili in PHP. Con un ciclo foreach applicato all'array 'names', passato dal controllore 'visualizzazione' in precedenza, la vista estrae i dati uno alla volta e li imposta in una tabella permettendo così una visualizzazione finale pulita e ordinata dei dati. Si ricorda che per motivi di spazio sono stati omessi tutti i pezzi di codice html necessari per la corretta visualizzazione complessiva della pagina Web. L'utente finale durante il suo utilizzo vedrà solamente le viste che gli verranno fornite dai vari controllori e interagirà con essi mediante delle interfacce grafiche che lo guideranno nella navigazione, nell'inserimento, nella cancellazione, nella visualizzazione e nell'aggiornamento delle varie informazioni. In figura 4.3 si può vedere come appare la schermata di visualizzazione degli studenti iscritti al Conservatorio nell'applicazione Web.

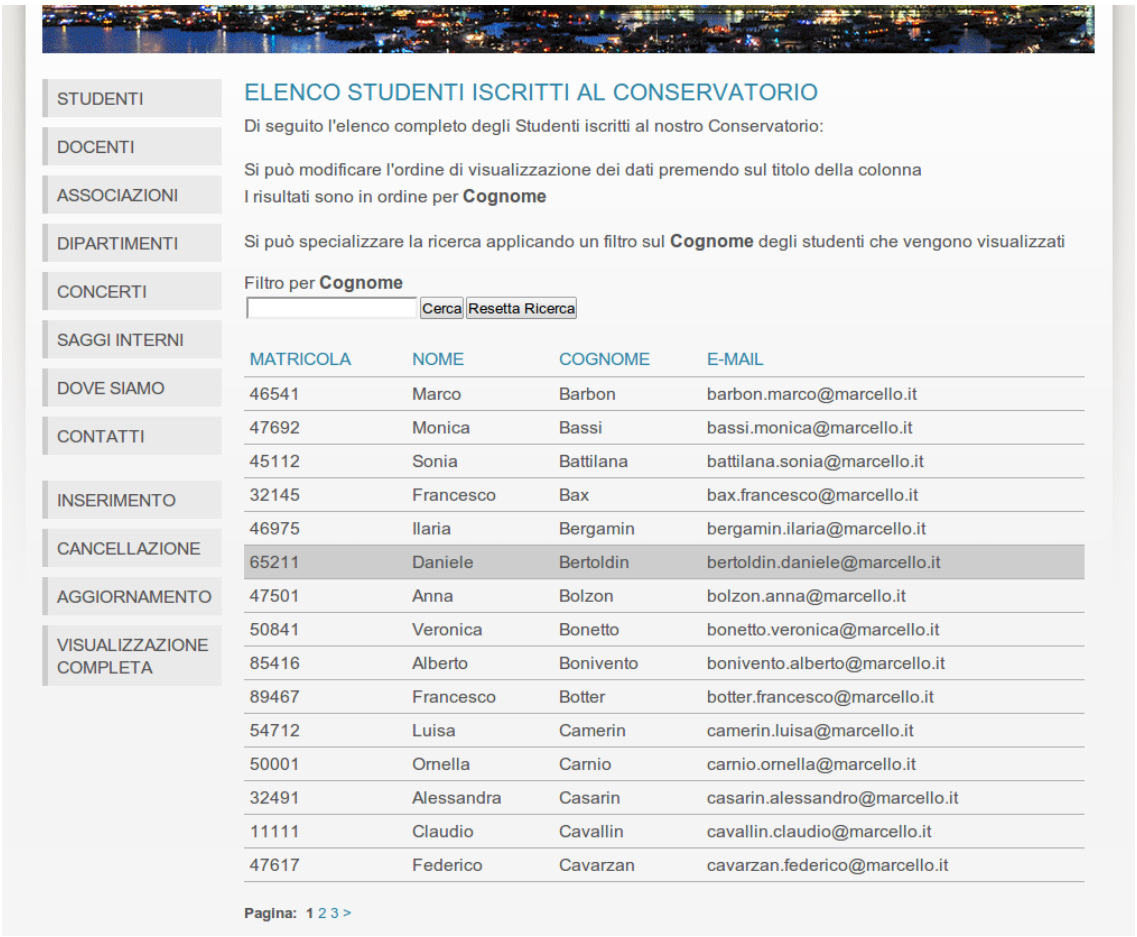

Figura 4.3: Schermata applicazione Web

L'applicazione Web funzionante, oltre ad essere stata progettata e sviluppata in locale, è stata caricata sul server del Dipartimento di Ingegneria dell'Informazione dell'Universita' di Padova sfruttando lo spazio Web personale e il database messo a disposizione di ogni studente del corso 'Basi di Dati'. Questo permette di visualizzarla e provarla direttamente online visitando l'indirizzo: **http://wwwdb.dei.unipd.it/2012/marchio2/**.

Capitolo 5

Conclusioni

In conclusione l'argomento principale della tesi ha riguardato la progettazione di un'applicazione Web tramite l'utilizzo del framework PHP CodeIgniter. Questo tipo di applicazioni Web, che sfruttano l'interazione con un database, sono ormai un aspetto fondamentale nel mondo della rete in quanto vengono utilizzate quasi quotidianamente nelle più comuni operazioni che si effettuano in internet come ad esempio un acquisto in un negozio on-line o anche un controllo del conto corrente bancario direttamente da casa. L'utilizzo di un framework PHP che sfrutti il paradigma MVC per lo sviluppo dell'applicazione Web si è ritenuto uno strumento molto valido. Ha permesso di ridurre molto sia il tempo necessario per lo sviluppo del progetto sia il ripetersi di pezzi di codice che altrimenti si sarebbero ripresentati in molte occasioni (come ad esempio il codice html necessario per definire il layout di una pagina che in un sito è solitamente sempre uguale o simile). Soprattutto ha permesso una progettazione dell'intera applicazione molto pulita ed ordinata perché, sfruttando il paradigma MVC, mette a disposizione dell'utente l'utilizzo di classi e funzioni come se si stesse utilizzando un linguaggio di programmazione ad oggetti.

Le possibilità di sviluppo futuro di questi strumenti sono molto vaste in quanto l'intero settore di progettazione che riguarda applicazioni che sfruttano la rete è in continua evoluzione. Un possibile sviluppo del progetto può essere l'ampliamento del database, in modo da permettere la gestione di altri aspetti che riguardano il Conservatorio, come ad esempio poter amministrare, oltre agli eventi, anche i corsi frequentati dai vari studenti. Per quando riguarda l'applicazione Web potrebbe essere la realizzazione di un'interfaccia grafica più completa che sfrutti altri linguaggi di programmazione in maniera più esaustiva e raffinata, come ad esempio Javascript.

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 Evento è un numero progressivo.
- L'identificatore di Dipartimento è il carattere "D" più 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)
3    CHECK (value='M' OR value='F');
4  CREATE DOMAIN telefono AS integer CHECK (value>=0);
5  CREATE DOMAIN fax AS integer CHECK (value>=0);
6
7  CREATE TABLE dipartimento
8  (
9    ID character varying(4) NOT NULL,
10   Nome character varying(60),
11   CONSTRAINT "pk_dipartimento" PRIMARY KEY (ID)
12 );
13
14 CREATE TABLE docente
15 (
16   CF character(16) NOT NULL,
17   Nome character varying(50) NOT NULL,
18   Cognome character varying(50) NOT NULL,
19   Sesso sesso NOT NULL,
20   Data_di_nascita date NOT NULL,
21   Indirizzo character varying(50) NOT NULL,
22   Citta character varying(50) NOT NULL,
23   CAP character(5) NOT NULL,
24   Telefono telefono NOT NULL,
25   Email character varying(50) NOT NULL,
26   CONSTRAINT "pk_docente_cf" PRIMARY KEY (CF)
27 );
28
29 CREATE TABLE anno_accademico
30 (
31   ID integer NOT NULL,
32   CONSTRAINT "pk_anno_accademico_id" PRIMARY KEY (ID),
```

```
33    CONSTRAINT "check_anno_accademico_id" CHECK (ID > 1940)
34  );
35  CREATE TABLE studente
36  (
37    Matricola matricola NOT NULL,
38    Nome character varying(50) NOT NULL,
39    Cognome character varying(50) NOT NULL,
40    Sesso sesso NOT NULL,
41    Data_di_nascita date NOT NULL,
42    Indirizzo character varying(50) NOT NULL,
43    Citta character varying(50) NOT NULL,
44    CAP character(5) NOT NULL,
45    Telefono telefono NOT NULL,
46    Email character varying(50) NOT NULL,
47    CONSTRAINT "pk_studente_matricola" PRIMARY KEY (Matricola)
48  );
49
50  CREATE TABLE studente_ritirato
51  (
52    Matricola matricola NOT NULL,
53    CONSTRAINT "pk_studente_ritirato_id_studente"
54      PRIMARY KEY (Matricola),
55    CONSTRAINT "fk_studente_ritirato_id_studente"
56      FOREIGN KEY (Matricola)
57      REFERENCES studente (Matricola) MATCH SIMPLE
58      ON UPDATE CASCADE ON DELETE RESTRICT
59  );
60
61  CREATE TABLE evento
62  (
63    ID integer NOT NULL,
64    Descrizione character varying(100) NOT NULL,
65    Luogo character varying(50) NOT NULL,
66    Data_ora timestamp(0) without time zone NOT NULL,
67    Saggio_interno boolean DEFAULT false,
68    CONSTRAINT "pk_evento_id" PRIMARY KEY (ID),
69    CONSTRAINT "check_evento_id" CHECK (ID > 0)
70  );
71
72  CREATE TABLE associazione
73  (
74    Partita_iva character(11) NOT NULL,
75    Nome character varying(50) NOT NULL,
76    Indirizzo character varying(50) NOT NULL,
77    Citta character varying(50) NOT NULL,
78    CAP character(5) NOT NULL,
79    Telefono telefono NOT NULL,
80    Fax fax NOT NULL,
```

```
81    Email character varying(50),
82    CONSTRAINT "pk_associazione_partita_iva"
83      PRIMARY KEY (Partita_iva)
84  );
85
86  CREATE TABLE partecipa
87  (
88    Matricola matricola NOT NULL,
89    ID_evento integer NOT NULL,
90    CONSTRAINT "pk_partecipa" PRIMARY KEY (Matricola, ID_evento),
91    CONSTRAINT "fk_partecipa_id_evento" FOREIGN KEY (ID_evento)
92      REFERENCES evento (ID) MATCH SIMPLE
93      ON UPDATE CASCADE ON DELETE RESTRICT,
94    CONSTRAINT "fk_partecipa_matricola" FOREIGN KEY (Matricola)
95      REFERENCES studente (Matricola) MATCH SIMPLE
96      ON UPDATE CASCADE ON DELETE RESTRICT
97  );
98
99  CREATE TABLE gestisce
100 (
101   ID_evento integer NOT NULL,
102   ID_dipartimento character varying(4) NOT NULL,
103   CONSTRAINT "pk_gestisce"
104     PRIMARY KEY (ID_evento, ID_dipartimento),
105   CONSTRAINT "fk_gestisce_id_dipartimento"
106     FOREIGN KEY (ID_dipartimento)
107     REFERENCES dipartimento (ID) MATCH SIMPLE
108     ON UPDATE CASCADE ON DELETE RESTRICT,
109   CONSTRAINT "fk_gestisce_id_evento" FOREIGN KEY (ID_evento)
110     REFERENCES evento (ID) MATCH SIMPLE
111     ON UPDATE CASCADE ON DELETE RESTRICT
112 );
113
114 CREATE TABLE organizza
115 (
116   ID_evento integer NOT NULL,
117   ID_dipartimento character varying(4),
118   CONSTRAINT "pk_organizza_id_evento"
119     PRIMARY KEY (ID_evento,ID_dipartimento),
120   CONSTRAINT "fk_organizza_id_dipartimento"
121     FOREIGN KEY (ID_dipartimento)
122     REFERENCES dipartimento (ID) MATCH SIMPLE
123     ON UPDATE CASCADE ON DELETE RESTRICT,
124   CONSTRAINT "fk_organizza_id_evento"
125     FOREIGN KEY (ID_evento)
126     REFERENCES evento (ID) MATCH SIMPLE
127     ON UPDATE CASCADE ON DELETE RESTRICT
128 );
```

```
129
130 CREATE TABLE dirige
131 (
132   CF_docente character(16) NOT NULL,
133   ID_dipartimento character varying(4) NOT NULL,
134   ID_anno integer NOT NULL,
135   CONSTRAINT "pk_dirige"
136     PRIMARY KEY (CF_docente, ID_dipartimento, ID_anno),
137   CONSTRAINT "fk_dirige_cf_docente" FOREIGN KEY (CF_docente)
138     REFERENCES docente (CF) MATCH SIMPLE
139     ON UPDATE CASCADE ON DELETE RESTRICT,
140   CONSTRAINT "fk_dirige_id_anno" FOREIGN KEY (ID_anno)
141     REFERENCES anno_accademico (ID) MATCH SIMPLE
142     ON UPDATE CASCADE ON DELETE RESTRICT,
143   CONSTRAINT "fk_dirige_id_dipartimento"
144     FOREIGN KEY (ID_dipartimento)
145     REFERENCES dipartimento (ID) MATCH SIMPLE
146     ON UPDATE CASCADE ON DELETE RESTRICT
147 );
148
149 CREATE TABLE crea
150 (
151   ID_evento integer NOT NULL,
152   P_iva_associazione character(11),
153   CONSTRAINT "pk_crea_id_evento"
154     PRIMARY KEY (ID_evento),
155   CONSTRAINT "fk_crea_id_evento"
156     FOREIGN KEY (ID_evento)
157     REFERENCES evento (ID) MATCH SIMPLE
158     ON UPDATE CASCADE ON DELETE RESTRICT,
159   CONSTRAINT "fk_crea_p_iva_associazione"
160     FOREIGN KEY (P_iva_associazione)
161     REFERENCES associazione (Partita_iva) MATCH SIMPLE
162     ON UPDATE CASCADE ON DELETE RESTRICT
163 );
```

Bibliografia

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

[2] Agostino Lorenzi, Enrico Cavalli : Basi di Dati e linguaggio SQL Teoria, Atlas (2009)

[3] Andrea Aulicino : La progettazione dei database, http://www.phpnews.it/corsi/la-progettazione-dei-database/, (2011)

[4] Gabliele Gigliotti : HTML 5 e CSS 3, Apogeo (2011)

[5] Massimiliano Bossi : Guida PHP, http://www.mrwebmaster.it/php/guide/guida-php/

[6] Andi Gutmans, Stig Bakken, Derick Rethans : PHP 5 Guida completa, Apogeo, (2005)

[7] Hugh E. Williams, David Lane : Applicazioni Web database con PHP e MySQL, Hops-Tecnologie (2005)

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

[9] Claudio Garau : Guida CodeIgniter, http://www.mrwebmaster.it/php/guide/guida-codeigniter/, (2012)

[10] Stefano Bianchini : Guida CodeIgniter a modo mio, http://stefanobianchini.blogspot.it/2012/11/guida-codeigniter-modo-mio-pdf-in.html, (2012)