INFORMATICA GENERALE

Andrea Pollini

PRIMA PARTE DEL TEMA

La prima parte del tema proposto consiste in una analisi delle caratteristiche fondamentali dell'evoluzione delle applicazioni web, partendo da quelle di prima generazione, per arrivare a quelle del web 2.0. Il web 2.0 è un termine coniato per descrivere tutto un ventaglio di tecnologie e di questioni, anche di natura sociologica e non prettamente informatica, per cui una trattazione sintetica risulta spesso semplificata ed assolutamente non banale.

Dal punto di vista sociologico il web 2.0 è il web della collaborazione. Il web dove il contenuto delle pagine web, prima generato *per* l'utente, viene generato dall'utente. È quello che viene definito *social web*, dove gli utenti utilizzano il web come strumento comunicativo ed espressivo. Questo web è formato da una pletora di applicazioni più o meno grandi e complesse che consentono di svolgere le azioni più disparate, dall'editare documenti come se si possedesse una suite di *office automation* (pensiamo al servizio *google docs*, ma anche ad altri), tenersi in contatto con amici e parenti (*facebook*), pubblicare le proprie foto, commentare le foto di altri (*flickr*), realizzare musica, video, creare reti di contatti di lavoro (*LinkedIn*), tutto direttamente online. Queste applicazioni con un grado di interattività elevata hanno avuto una diretta ripercussione sul lavoro di ricerca di strumenti che fossero sempre più espressivi con una complessità di utilizzo sempre minore per l'utente finale.

Dal punto di vista prettamente informatico queste esigenze di strumenti di comunicazione sempre più responsivi e dinamici ha reso necessaria la realizzazione e l'introduzione di tecnologie atte a rendere le applicazioni web sempre più simili alle applicazioni desktop, tanto che spesso oggigiorno le stesse applicazioni desktop funzionano come *web application* locali.
L'ostacolo che limitava lo sviluppo di *web application* simili ad applicazioni desktop era insito proprio nella struttura del web di prima generazione. L'interazione tra utente e sito era

LA TRACCIA MINISTERIALE

INFORMATICA GENERALE, APPLICAZIONI GESTIONALI, INFORMATICA GESTIONALE

Con il termine *Web 2.0* si indica una nuova fase dell'evoluzione di Internet (in modo particolare del World Wide Web) che vede un ampio insieme di applicazioni online permettere elevati livelli di interazione tra siti e utenti. Il candidato, dopo aver esposto le caratteristiche delle applicazioni per il web di prima generazione, consideri poi i principali esempi di applicazioni di seconda generazione, approfondendo quindi le implicazioni del fenomeno Web 2.0 ed esponendo considerazioni e riflessioni su di esso.

Il candidato consideri inoltre la seguente situazione.

Un'agenzia immobiliare intende potenziare la sua attività per offrire, nella città dove si trova, affitti di case per brevi periodi. In tale città è infatti forte la richiesta di tali servizi in ogni momento dell'anno ed anche in relazione a diversi eventi internazionali che richiamano un forte flusso turistico, che non trova accoglienza nelle strutture alberghiere. L'agenzia intende realizzare un sistema, anche accessibile dal suo sito web, che renda pubbliche le offerte di affitto di appartamenti di proprietari privati, consentendo al tempo stesso le prenotazioni e la conferma delle transazioni di affitto.
Degli appartamenti interessa registrare le caratteristiche generali e i dettagli rilevanti per le offerte, non escluse alcune foto. Dei proprietari degli appartamenti sono rilevati i dati anagrafici e di residenza, quelli di contatto e le coordinate bancarie per gli accrediti dei pagamenti. Per i potenziali clienti interessati all'affitto degli immobili, che devono registrarsi con nome utente e password occorrono dati anagrafici e di residenza, oltre a dati di contatto.
La disponibilita degli appartamenti è registrata per il mese corrente e per i sei mesi successivi. Le prenotazioni possono avvenire per i giorni che risultano disponibili e devono essere confermate entro tre giorni mediante il versamento di una quota pari al 40% del costo di affitto dovuto, altrimenti l'appartamento ritorna disponibile.
Dopo aver proceduto all'analisi concettuale e logica dei dati, il candidato indichi le istruzioni per ottenere dalla base di dati le seguenti informazioni:
– le caratteristiehe generali di un appartamento, dato il suo codice;
– l'elenco degli appartamenti che si trovano in un determinato quartiere;
– l'elenco degli appartamenti che offrano un numero di posti letto non inferiore ad un valore indicato;
– il numero totale di appartamenti offerti dall'agenzia, indipendentemente dalle loro caratteristiche;
– il costo totale di affitto per ogni prenotazione, in relazione al numero di giorni richiesti.
Il candidato esponga poi un esempio del codice necessario per la realizzazione della pagina web di presentazione di un generico appartamento, in HTML o in altro linguaggio di sua conoscenza.
Il candidato può formulare opportune ipotesi per completare quanto ritenga necessario specificare ulteriormente.

possibile unicamente mediante il click su dei link o la pressione del tasto di invio di un *web form*. L'esito di ogni tipo di interazione era quindi il caricamento di una pagina, con un peso notevole per ogni singola richiesta anche quando solo una semplice operazione di aggiornamento di una piccola porzione della pagina web era necessaria. Col web 2.0 si è superata questa limitazione, consentendo l'aggiornamento anche solo del contenuto di un singolo tag HTML della pagina o il ricaricamento di una tabella dati o la modifica dello stile di un elemento. Per aggiornare l'interfaccia utente in modo puntuale si è fatto ricorso a Javascript, una tecnologia da tempo utilizzabile ma che con l'avvento del web 2.0 è diventata uno strumento non più usato solo per creare animazioni, ma anche proprio come vero e proprio strumento di costruzione dell'interfaccia utente e della sua gestione sia a livello di interazione tra gli elementi di una stessa pagina, sia per aggiornarla con i dati che via via vengono comunicati dal server.
Inoltre è stata introdotta una particolare funzionalità che consente da una pagina di richiamarne in modo asincrono un'altra e di analizzarne, ed eventualmente eseguirne, il contenuto, se questo è codice Javascript. In questo modo abbiamo un canale di comunicazione bidirezionale tra client e server che consente il trasferimento di dati, mediante lo standard JSON o XML, e trasferimento dal server di codice che il client possa utilizzare per aggiornare la pagina che l'utente sta visualizzando, tramite codice Javascript.
Questo insieme di tecnologie, ma anche tante altre accessorie, formano quello che viene definito AJAX. Questo modo di comunicare è molto più economico in termini di risorse rispetto a quello basato su form e caricamento di pagine complete utilizzato nelle applicazioni di prima generazione. Questa ottimizzazione si traduce tuttavia in una maggiore complessità nella programmazione delle applicazioni stesse. Al programmatore è richiesto di essere in grado di ragionare in maniera asincrona per imparare a gestire correttamente il flusso di programmazione e per effettuare una programmazione efficace è necessario che esso padroneggi numerose tecnologie e librerie.

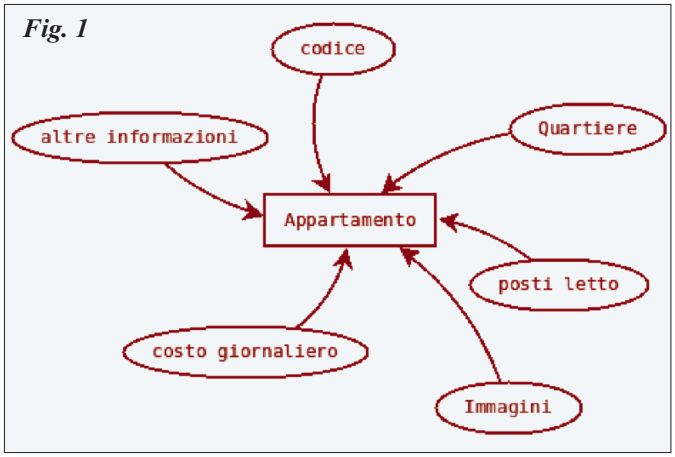
Seconda parte del tema

Nella seconda parte del tema viene proposto di modellizzare un sistema di prenotazione online di appartamenti in affitto.

Modello concettuale. Dal punto di vista concettuale la nostra applicazione gestirà quattro entità: l'*appartamento*, l'*affittuante*, l'*affitto* e il *cliente.*
L'appartamento dovrà contenere solo le informazioni necessarie per poter rispondere alle richieste del tema. Lasceremo inoltre un campo informazioni aggiuntive per consentire eventualmente di inserire anche ulteriori informazioni. Pur non essendo strutturate, potremo sempre effettuare una ricerca di tipo *full-text* e recuperare alcune informazioni utili.

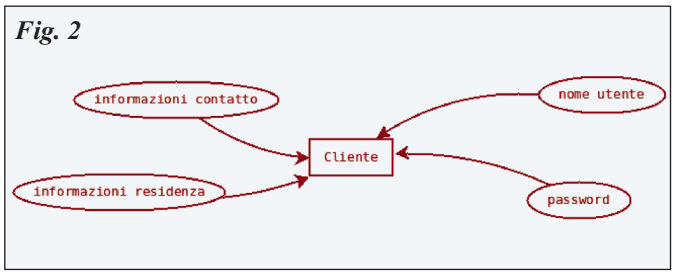
Inoltre sarà necessario consentire di gestire le prenotazioni e la conferma delle stesse mediante una entità *affitto*.
Analizziamo in dettagli le varie entità iniziando dall'entità *appartamento (Fig. 1)*.

Fig. 1



Il codice verrà generato automaticamente dall'applicazione al momento del salvataggio delle informazioni inserite da parte del proprietario dell'immobile e sarà univoco. Sono state poste in evidenza le informazioni che servono per individuare gli appartamenti secondo quanto richiesto dalle *query* proposte nel tema d'esame. Nel campo *altre informazioni* verranno introdotte tutte le informazioni che completano la descrizione dell'appartamento, ad esempio l'estensione, l'eventuale posto auto o la presenza di un collegamento Internet.
La seconda entità è l'entità *cliente.* Un cliente è chi si registra al sito web ed effettua le prenotazioni e il pagamento degli affitti *(Fig. 2)*.

Fig. 2



L'entità *affittuante* rappresenta colui che affitta l'appartamento e che riceverà i pagamenti. Assumeremo che le transazioni avvengano esternamente al sistema e l'unica preoccupazione dei gestori del sistema sarà quella di registrare l'avvenuto pagamento della prenotazione e dell'affitto (vedi *Fig. 3*).
L'entità *affitto* conterrà tutte le informazioni relative ad un affitto e la relativa prenotazione. Semplificando le nostre considerazioni iniziali possiamo pensare che ad un affitto saranno associati una data di inizio, una durata e l'indicazione se il pagamento della caparra sia stato o meno effettuato. L'entità avrà quindi la struttura indicata in *Fig. 4*.
Le relazioni da tener presente tra le entità che formano il

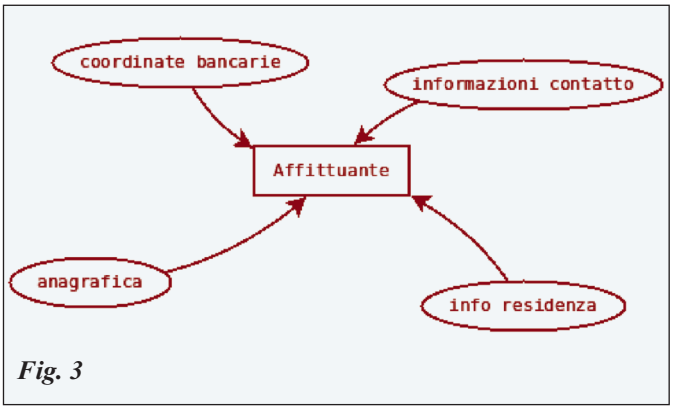


Fig. 3

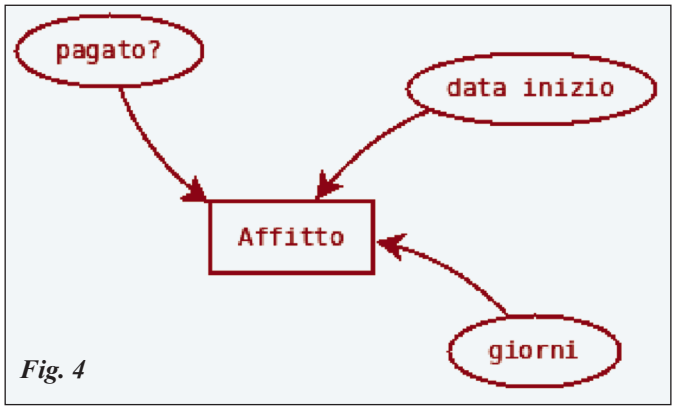


Fig. 4

modello concettuale sono ben descritte dal problema. Innanzitutto ogni appartamento sarà *posseduto* da un affittuante, e lo stesso appartamento sarà in relazione con più affitti diversi, relativi a periodi diversi. A sua volta ogni affitto farà riferimento al cliente che l'ha pagato. Il diagramma delle relazioni tra le entità del modello in esame possono essere sintetizzate nel diagramma rappresentato in *Fig. 5*.

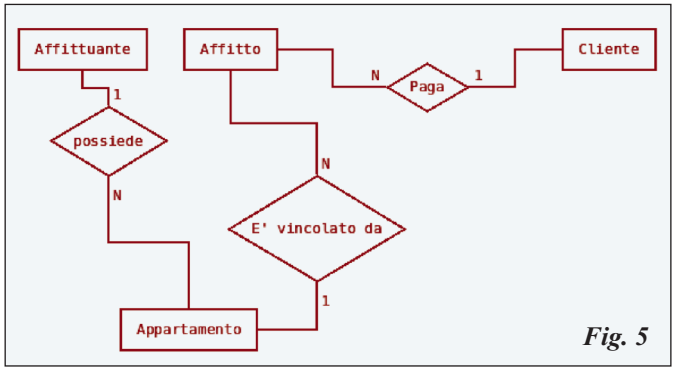


Fig. 5

Struttura logica della base dati

La struttura logica della base dati del problema proposto sarà composta da una tabella per ogni entità, più eventuali tabelle di supporto.
Inoltre in ogni tabella oltre agli attributi indicati nei diagrammi delle singole entità avremo campi aggiuntivi relativi alle relazioni ed alle necessità di definire in modo univoco ogni riga di ogni tabella, che avranno sempre il nome *id* e saranno di tipo autoincrementante.

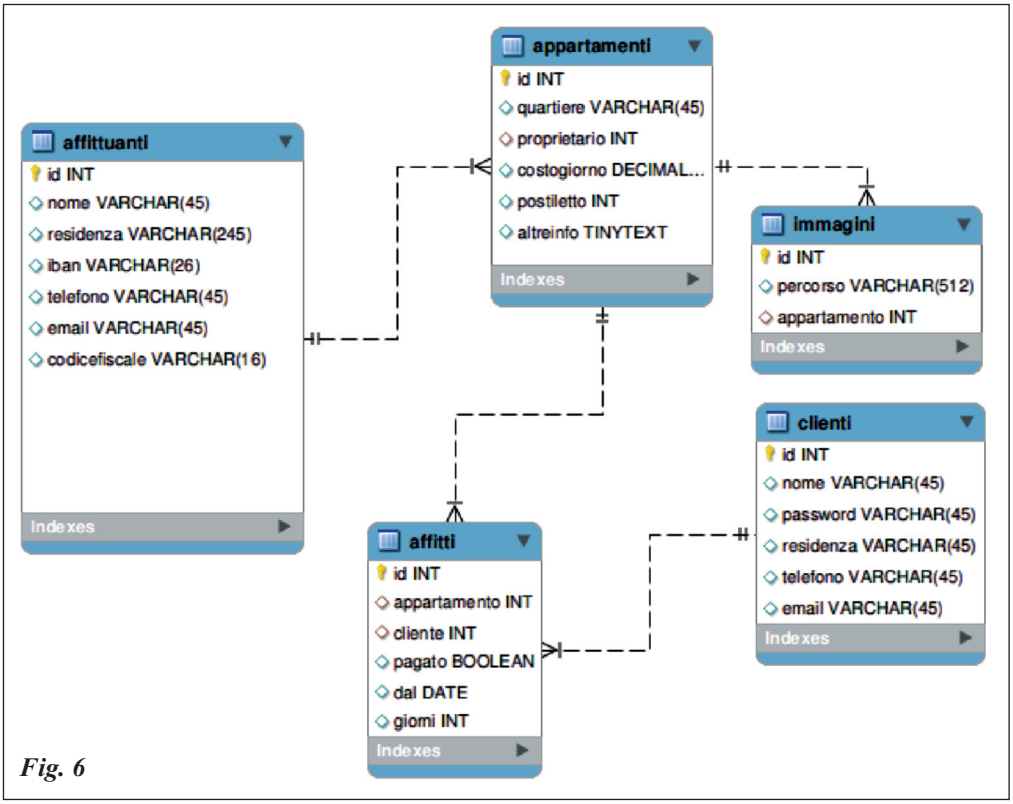


Fig. 6

Inoltre alcuni attributi, come dati anagrafici, di contatto e coordinate bancarie potrebbero essere o suddivisi in più campi, oppure addirittura essere posti in tabelle separate ed aggiuntive rispetto al modello concettuale. Per avere una gestione semplificata faremo invece l'assunzione che l'attributo *informazioni di contatto* sia formato da un numero di telefono, eventualmente mobile, e da un'indirizzo *e-mail*.
La *residenza* sarà modellizzata da un singolo campo testo, abbastanza grande da contenere tutte le informazioni richieste. Allo stesso modo le coordinate bancarie saranno la stringa del codice IBAN del conto corrente (*Fig. 6*).

Query Richieste

La prima *query* richiesta è quella che ritorni le caratteristiche generali di un appartamento, dato il suo codice. Questa *query* è molto facile. Utilizzando come codice il campo *id* della tabella *appartamenti* recuperare le informazioni di un singolo appartamento corrisponderà alla seguente query:

```
SELECT * FROM appartamenti WHERE id= :id
```

La notazione riferita ai parametri è la presenza dei due punti prima del parametro.

La seconda *query* richiesta è quella relativa all'elenco degli appartamenti di un singolo quartiere. Per recuperare questi dati sarà sufficiente ripetere la query precedente, ma non utilizzare più l'*id* come filtro, per cui la *query* che risolve questo quesito sarà la seguente:

```
SELECT * FROM appartamenti WHERE quartiere = :quartiere
```

La terza *query* da realizzare deve ritornare tutti gli appartamenti che posseggono un numero di posti letto maggiore di un valore assegnato. Dovremo far ricorso all'operatore di maggiore uguale, applicato al campo *postiletto* della tabella *appartamenti* come di seguito indicato:

```
SELECT * FROM appartamenti WHERE postiletto >= :postiletto
```

Da notare come serva utilizzare l'operatore di maggiore uguale e non di maggiore stretto in quanto il tema richiede che il numero di posti letto sia «non inferiore».

Per recuperare il numero totale di appartamenti offerti dall'agenzia è sufficiente contarli, utilizzando la funzione SQL COUNT(). Ovvero:

```
SELECT COUNT(*) FROM appartamenti
```

Queste *query* sono semplici; si tratta di filtrare una tabella o di applicare una funzione SQL.

L'ultima *query* proposta invece è un po' più articolata e complessa. Per prima cosa selezioniamo gli affitti ed i giorni di durata di ogni affitto:

```
SELECT affitti.id,affitti.giorni FROM affitti
```

A questo punto recuperiamo i dati di costo al giorno dall'appartamento relativo a quell'affitto:

```
SELECT appartamenti.costogiorno FROM appartamenti WHERE
  appartamenti.id  = :id
```

e colleghiamo le due *query*

```
SELECT

   affitti.id, affitti.giorni, appartamenti.costogiorno

FROM affitti, appartamenti

WHERE appartamenti.id = affitti.appartamento
```

Ora basta riscrivere questa *query* andando a moltiplicare i giorni per il costo al giorno:

```
SELECT
    affitti.id,affitti.giorni *  appartamenti.costogiorno
FROM affitti, appartamenti
WHERE appartamenti.id = affitti.appartamento
```

Il tema successivamente chiede di presentare il codice HTML necessario per visualizzare a video i dati relativi ad un singolo appartamento. L'HTML è l'HyperText Markup Language ed è il linguaggio che consente alle informazioni passate dal server web al browser di venire presentate all'utente. L'HTML verrà generato dinamicamente a partire da un *template*, ovvero da un file di testo che sarà l'HTML più alcuni costrutti particolari che indicheranno al *template engine* scelto, come e dove inserire i dati recuperati dal database.

Nella struttura della nostra applicazione utilizziamo quindi un approccio di tipo MVC modificato per il web, dove le viste vengono quindi generate dai *template* HTML, a cui il controller passa degli oggetti, rappresentazione informatica strutturata di righe del database e loro relazioni. Nel nostro caso utilizzeremo per il *template* il linguaggio di *template* del *framework* Django, che consente di create *template* leggibili anche da parte di chi non conosce i *framework* in modo semplice. Un esempio di *template* per la visualizzazione dei dati di un appartamento sarà quello proposto nel seguente spezzone di codice.

```
<html>
<head>
<title>{{appartamento.id}}</title>
</head>
<body>
<h1>Scheda appartamento : {{appartamento.id}}</h1>
<h2>Informazioni</h2>
   <p><b>Posti Letto</b>: {{appartamento.postiletto}}</p>
   <p><b>Costo giornaliero ( )</b> {{appartamento.costogiorno}}</p>
   <p><b>altre Informazioni:</b> {{appartamento.altreinfo}}
<h2>Immagini</h2>
   <ul>
   {% for immagine in appartamento.immagini %}
      <li><img src="{{immagine.percorso}}" />
   {% endfor %}
   </ul>
</body>
</html>
```

In prospettiva

Il tema proposto quest'anno non è un tema complesso. Infatti solo una delle *query* proposte presenta alcune difficoltà di realizzazione. Inoltre la domanda con cui si apre il tema è di carattere abbastanza generale e di certo non è complessa potendo essere affrontata a livelli di approfondimento diversi. Manca a mio avviso una richiesta sulla struttura dell'applicazione nel suo complesso, che avrebbe reso il tema più interessante e dove ogni studente avrebbe potuto esprimere al meglio la propria preparazione e la propria creatività.

Andrea Pollini
Università Cattolica del Sacro Cuore
Brescia