Capitolo 2

Il linguaggio Penelope

Penelope è un linguaggio dichiarativo che permette di generare siti web a partire da dati memorizzati su basi di dati relazionali. Tramite questo linguaggio è possibile definire delle istruzioni che descrivono la struttura delle pagine del sito che si vuole generare. Poiché queste istruzioni descrivono la struttura di pagine web, prendono il nome di page-scheme. L'insieme di più page-scheme prende invece il nome di site-scheme.

Sulla base di uno schema ADM, che descrive la struttura del sito, è possibile individuare gli attributi di ogni pagina. A partire da questo schema e utilizzando il linguaggio Penelope si ottiene una rappresentazione del sito in forma testuale.

Nei paragrafi che vanno dal 2.1 al 2.13 verrà presentata in maniera dettagliata la sintassi di tale linguaggio, mentre nel paragrafo 2.14 ne viene presentata la semantica.

Nell'ultimo paragrafo viene discusso come tramite il linguaggio Penelope sia possibile definire oltre la struttura anche il layout grafico del sito che si sta definendo.

2.1 Definizione di schema

La definizione dello schema è raccolta in un file[1] di testo e inizia con la seguente istruzione Penelope:

`SCHEME` *address* `ON` *database*

[1] Al file che definisce il site-scheme bisogna associare l'estensione .PL (Penelope Language)

address deve indicare il protocollo di connessione e la locazione logica o fisica del sito, il protocollo può assumere due forme diverse:

1. http://HostName/HostDir nel caso si tratti di un sito Web;
2. file:/FileSystemDir nel caso si intenda utilizzare il mirror di un sito memorizzato sul file system.

Database specifica invece il nome della sorgente[2] dati tramite la quale l'interprete del linguaggio Penelope dialoga con la base di dati che mantiene le informazioni da pubblicare nel sito.

Nei prossimi paragrafi verranno esposti i costrutti del linguaggio Penelope con l'ausilio di esempi che fanno riferimento alla base di dati relazionale `Department`, il cui schema logico è riportato di seguito:

```
Professor (Name, Email, Phone, Photo, Publications)
Student (Name, Email, Phone)
Course (CourseName, Description, Professor, Type)
PersonInGroup (Name, GroupName)
ResearchGroup (GroupName, Topic)
Lessons (CourseName,Instructor,Date)
TextBook (CourseName, Title, Author, Publisher)
```

2.2 Definizione di pagina

Consideriamo la seguente istruzione Penelope:

Esempio 2.1:

```
DEFINE PAGE GraduateCoursePage : <CourseName>
AS URL(<CourseName>);
   Name : TEXT = <CourseName>;
   Description : TEXT = <Description>;
```

[2] La sorgente dati è di solito un driver ODBC

```
Type : TEXT = <Type>;
USING Course
WHERE Type LIKE “Graduate”
ORDER BY <CourseName>
END
```

CourseName	Description	Professor	Type
Compilers	...	Torlone	graduate
DatabaseSystem	...	Atzeni	graduate
ArtificialIntelligence	...	Cialdea	graduate
...	...	...	...

Figura 2.1: Tabella relazionale `Course`

Questa istruzione, a partire dalla tabella relazionale `Course` in Figura 2.1 produce istanze di pagine Web descritte dallo schema ADM in Figura 2.2.

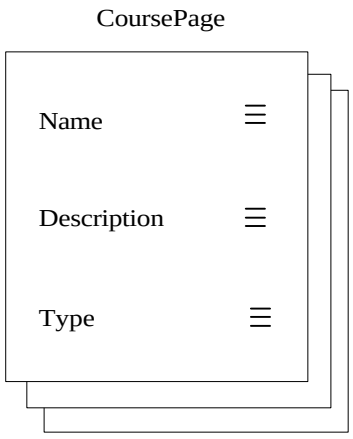

Figura 2.2: Schema ADM della pagina `CoursePage`

Come si può osservare dall’Esempio 2.1 un’istruzione Penelope può essere suddivisa fondamentalmente i cinque blocchi:

1. la clausola `DEFINE PAGE`;

2. la clausola AS;

3. la clausola USING;
4. la clausola WHERE;
5. la clausola ORDER BY.

Nei paragrafi che seguono analizzeremo nel dettaglio ognuna di queste clausole.

2.3 La clausola DEFINE PAGE

La clausola define page comincia con la parola chiave DEFINE PAGE.

In tale clausola viene specificato se il page-scheme che si vuole generare è dinamico(dynamic) o meno.

Nel caso in cui DEFINE PAGE sia seguita dalla parola DYNAMIC allora verrà generata o una JSP o una ASP, a seconda di cosa sceglie l'utente. Viceversa se la DEFINE PAGE è seguita da STATIC allora verrà generato un programma JAVA che una volta compilato ed eseguito genera una o più pagine nel formato selezionato dall'utente (HTML o XML). Per *default* se viene omesso sia DYNAMIC che STATIC si assume che si voglia generare un page-scheme statico. E' anche possibile generare direttamente pagine web, questo può essere scelto dall'utente attraverso dei comandi che sono descritti nel paragrafo 6.3 quando si parla della PenelopeGUI.

Dopo aver specificato il tipo di page-scheme che si vuole generare seguono il nome e il titolo della pagina. Nell'Esempio 2.1 il nome della pagina è CoursePage mentre il titolo viene costruito utilizzando i valori dell'attributo <CourseName> della tabella Course.

Un altro modo di costruire il titolo di una pagina è quello di associargli un valore costante racchiuso tra virgolette ad esempio :

```
DEFINE PAGE CoursePage : "Course"
```

Questo modo di costruire il titolo della pagina è utile per la creazione del titolo di pagine uniche.

2.4 La clausola AS

La clausola `AS` di un'istruzione Penelope specifica la struttura logica della pagina. Più in particolare in tale clausola vengono definiti l'`url` e gli attributi (testo, immagini, link etc...) della pagina. Facciamo ancora riferimento all'Esempio 2.1 e focalizziamo l'attenzione sulla definizione degli attributi poiché sulla definizione dell'url torneremo in dettaglio nel paragrafo successivo.

Notiamo la stretta corrispondenza tra gli attributi del nostro page-scheme e quelli dello schema ADM riportato in Figura 2.2. Il page-scheme `CoursePage` ha tre attributi, `Name,Description` e `Type`, di tipo `TEXT`[3]. Per costruire ciascun istanza di pagina i valori degli attributi sono estratti dalla tabella `Course` (vedi Figura 2.1), specificata nella clausola `USING` dell'istruzione Penelope. In particolare, notiamo che a ogni attributo del page-scheme viene associato un attributo della relazione `Course`, racchiuso tra parentesi acute ("<" e ">"); questo sta a significare che, per la creazione di ciascuna pagina, i valori di ogni attributo devono essere presi dai valori degli attributi della relazione `Course` in conformità con l'associazione specificata. Ad esempio, in ciascuna pagina istanza di `CoursePage`, il valore dell'attributo `Name` del page-scheme è estratto dall'attributo `CourseName` della tabella relazionale `Course`.

2.4.1 La funzione URL

La funzione `URL()` crea indirizzi il cui valore dipende dal valore dei parametri che le vengono passati. Essa non è altro che una funzione di *skolem* che associa a tuple di valori una stringa univoca rappresentante l'URL della pagina generata.

Affinché sia iniettiva è necessario che l'insieme degli attributi utilizzati per la costruzione dell'`URL` siano chiave o superchiave della relazione dalla quale vengono

[3] Nel linguaggio Penelope possono essere definiti i seguenti tipi di attributi: TEXT, IMAGE, LIST-OF,

estratti; questa condizione di iniettività è fondamentale poiché ogni pagina è individuata dal proprio `URL`.

Consideriamo ancora il page-scheme dell'Esempio 2.1 e riportiamo di seguito la funzione `URL()` in esso definita:

```
URL(<CourseName>)
```

Il parametro che le viene passato è un attributo della tabella `Course` (Figura 2.1) che ne rappresenta la chiave, in tal caso Penelope costruisce, per ogni tupla della tabella, una istanza di pagina con valore dell'`URL` ricavato a partire dal valore dell'attributo `<CourseName>`, ad esempio l'`url` della pagina del corso `Database Systems` sarà:

site-path\CoursePage\DatabaseSystems.html

nel caso si generi direttamente le pagine web statiche, mentre sarà:

site-path\CoursePage\CourseName.*ext*

con *ext* che può essere JSP, ASP o Java nel caso di generazione di programmi.

In realtà, come si può notare, l'indirizzo di una pagina è generato concatenando il site-path, il nome del page-scheme che la definisce e il valore ricavato dal parametro passato alla funzione `URL()`.

Al momento della generazione del sito, Penelope memorizza tutti i file nella directory di root del sito, cioè nella directory contenente il file (.PL) che lo definisce. Per ogni page-scheme *P* crea, nella directory di root del sito, una directory denominata *P* e in tale directory memorizza tutte le istanze di pagina oppure programmi

LINK-TO e MAP.

relativi a tale page-scheme. Il "site-path" coincide con il path della directory di root del sito.

Nel caso di pagine uniche ha senso definire una funzione URL() con un solo valore costante, ad esempio:

```
URL("index")
```

Quando la funzione `URL()` ha un solo parametro costante, essa genera un url a partire direttamente da questo valore. L'estensione viene determinata in base al formato scelto, HTML o XML, nel caso di generazione diretta di pagine web oppure JSP, ASP o Java nel caso di generazione di programmi.

E' necessario puntualizzare che alla funzione `URL()` possono essere passati più parametri[4] separati tra loro da virgole.

Nel caso alla funzione `URL`() vengano passati più parametri allora:

1. se si generano direttamente le pagine web, la funzione `URL()`costruisce un `url` per ogni valore dell'attributo `<CourseName>`. Ogni `url` viene costruito a partire dal valore di una stringa ottenuta dalla combinazione dei valori dei parametri;

2. se invece si generano programmi, si attua una gestione diversa per generare il nome del file fisico; gli attributi di tipo URL si presentano nella seguente forma :

   URL(item^{+}) ove item = {"valore costante", <valore variabile>}

   e la loro gestione, se si generano programmi, è stata ricondotta a 2 casi:

   1. URL("valore costante") o URL("valore costante"[, item]$^{+}$): viene tradotto con l'indirizzo di pagina **"valore costante".ext**

   2. In tutti gli altri casi l'indirizzo della pagina viene tradotto con **"nome del page-scheme". ext**

   dove **ext** può essere JSP, ASP oppure Java.

[4] Il numero di parametri che si possono passare alla funzione `URL()` è teoricamente illimitato,è comunque ragionevole che il numero di parametri non sia eccessivamente alto.

Di seguito è riportato un esempio di funzione `URL()` alla quale sono passati due parametri:

```
URL("index", <CourseName>)
```

Nelle due tabelle che seguono vengono riportate nella colonna di sinistra le coppie di valori della funzione `URL()` e nella colonna di destra il corrispondente `url` generato; nella tabella 2.1 si considera il caso in cui si stia generando direttamente pagine web statiche mentre nella tabella 2.2 si considera il caso in cui si stia generando programmi.

Coppie di valori di partenza	**Url per siti statici generati direttamente**
("index", Compilers)	SitePath\CoursePage\IndCompilers
("Index", Database Systems)	SitePath\CoursePage\IndDatabaseSys
.....	.....
("index", Operating Systems)	SitePath\CoursePage\indOperatingSys

Tabella 2.1

Coppie di valori di partenza	**Url nel caso di generazione di programmi**
("index")	SitePath\CoursePage\Index.ext
("Index", Database Systems)	SitePath\CoursePage\Index.ext
.....	.....
(<CourseName>, "index")	SitePath\CoursePage\CourseName.ext

Tabella 2.2

2.5 La clausola USING

Nella clausola `USING` di un'istruzione Penelope vengono specificate le tabelle, o viste relazionali, dalle quali vengono estratte le informazioni necessarie per istanziare la pagina definita in tale istruzione. Nella clausola `USING` dell'Esempio 2.1 viene specificata la sola tabella `Course`.

Consideriamo un esempio più complesso che utilizzi una vista relazionale:

Esempio 2.2:

```
DEFINE PAGE ResearchGroupListPage : "ResearchGroupListPage"
AS URL("researchGroupList.html");
    GroupList : LIST-OF (
            ToGroup : LINK-TO ResearchGroupPage(
                              URL (<GroupName>);
                              GName : TEXT = <GroupName>;
                                          );
                      );
  USING  Groups : ( Select GroupName From ResearchGroup )
         DISTINCT
  END
```

In questo esempio nella clausola `USING` viene definita la vista relazionale `Groups`. Tralasciamo per ora la direttiva `DISTINCT` della quale discuteremo alla fine del paragrafo.

Come si può notare, Penelope offre la possibilità di creare delle viste sul database nel caso in cui non si abbia la possibilità di ottenere le informazioni desiderate dalle tabelle relazionali in esso contenute.

GroupName	Topic
Database and Information Systems	Databases and the Web
Database and Information Systems	CASE technologies
Database and Information Systems	Database languages
Database and Information Systems	Requirement engineering
Artificial Intelligence	Logic for knowledge.
Artificial Intelligence	Modal and temporal logics
Graph Drawings	Geometric modeling
Graph Drawings	Computer graphics

Figura 2.3: Tabella relazionale `ResearchGroup`

GroupName
Database and Information Systems
Artificial Intelligence
Graph Drawings
Symbolic Computational

Figura 2.4: Vista relazionale `Groups`

Le viste relazionali possono essere generate utilizzando il linguaggio SQL. La query definita nell'Esempio 2.2 genera una tabella (vedi Figura 2.4) con un solo attributo, `GroupName`, a partire dalla tabella relazionale `ResearchGroup` riportata in Figura 2.3. Poiché nella clausola `USING` è stata specificata la direttiva `DISTINCT` la vista relazionale creata non conterrà duplicati.

La direttiva `DISTINCT` deve essere usata con attenzione; infatti per alcuni drivers ODBC (per esempio Microsoft Access), non è possibile eliminare i duplicati da tabelle con attributi `BLOB` o `MEMO`.

2.6 La clausola WHERE

La clausola `WHERE` ammette come argomento un'espressione booleana. Facciamo nuovamente riferimento all'Esempio 2.1 del quale riportiamo di seguito la parte relativa alla clausola `WHERE`:

```
WHERE Type LIKE "Graduate"
```

In questo caso Penelope esegue, per ogni tupla della tabella Course, un confronto tra il valore contenuto nell'attributo `Type` e la costante specificata dopo la parola chiave `LIKE` selezionando solo le tuple per le quali tale attributo assume un valore identico a quello di tale costante (nel nostro esempio `Graduate)`.

2.7 La clausola ORDER BY

La clausola `ORDER BY` è opzionale. Se modifichiamo leggermente l'istruzione Penelope introducendo la direttiva `ORDER BY`; l'istruzione Penelope diventa allora come segue:

Esempio 2.3:

```
DEFINE PAGE ResearchGroupListPage : "ResearchGroupListPage"
AS URL("researchGroupList.html");
    GroupList : LIST-OF (
            ToGroup : LINK-TO ResearchGroupPage(
                              URL (<GroupName>);
                              GName : TEXT = <GroupName>;
                                             );
                      );
USING Groups:(Select GroupName From ResearchGroup) DISTINCT
ORDER BY GroupName
```

```
END
```

In questo caso la vista `Groups` (vedi Figura 2.5) oltre a non contenere coppie di tuple con lo stesso valore viene ordinata rispetto al suo attributo `GroupName`.

GroupName
Artificial Intelligence
Database and Information Systems
Graph Drawings
Symbolic Computational

Figura 2.5: Vista relazionale `Groups`

Ovviamente questo ordinamento si ripercuote sull'ordinamento degli item della lista `GroupList`.

2.8 Attributi di tipo TEXT

Approfondiamo il discorso sugli attributi di tipo `TEXT` già introdotti nei precedenti paragrafi. Abbiamo visto che è possibile associare ad un attributo del page-scheme un attributo di una tabella della base di dati che contiene le informazioni relative al sito che si vuole generare. Proprio grazie a questa associazione Penelope è in grado di istanziare gli attributi definiti nel page-scheme. Un altro modo per assegnare un valore a un attributo di tipo `TEXT` è quello di associargli un valore costante racchiuso tra virgolette(“). Con riferimento all'Esempio 2.1 potremmo ad esempio voler aggiungere un attributo costante di nome `WebMaster` e di valore “`WebMaster: PaoloRossi`”, in tal caso otterremmo il seguente page-scheme:

Esempio 2.4:

```
DEFINE PAGE CoursePage : <CourseName>
```

```
AS URL(<CourseName>);
   Name : TEXT = <CourseName>;


   Description : TEXT = <Description>;
   Type : TEXT = <Type>;
   WebMaster : TEXT = “WebMaster: Paolo Rossi”;
USING Course
END
```

In questo modo in ogni istanza di pagina apparirà, come ultima informazione, la scritta `WebMaster: PaoloRossi`

2.9 Attributi di tipo IMAGE

Consideriamo la seguente istruzione Penelope:

Esempio 2.5:

```
DEFINE PAGE ProfessorPage : <Name>
AS URL(<Name>);
   Name : TEXT = <Name>;
   Email : TEXT = <email>;
   Phone : TEXT = <Phone>;
   Photo : IMAGE =<Photo>;
USING Professor
END
```

Questa istruzione, a partire dalla tabella relazionale `Professor` in Figura 2.7 produce istanze di pagine Web descritte dallo schema ADM in Figura 2.6

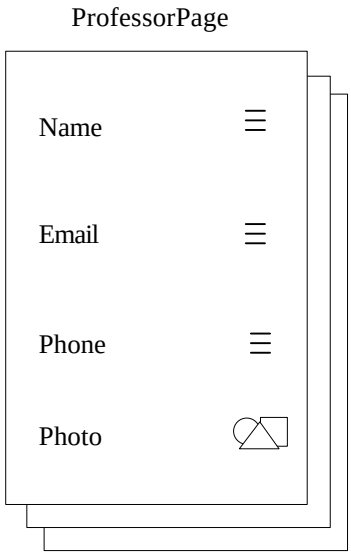

Figura 2.6: Schema ADM della pagina `ProfessorPage`

Name	Position	Email	Phone	Photo	Pubblications
Atzeni	FullProfessor	atzeni@...	3226	../icons/atz	http://www..
Di Battista	FullProfessor	gdb@...	3212	../icons/gdb	http://www..
Miola	Professor	miola@...	32..	../icons/m	http://www..
...	...	...	...	...	...

Figura 2.7: Tabella relazionale `Professor`

Nel page-scheme dell'Esempio 2.5 è definito un attributo di tipo `IMAGE.` La sintassi di questo tipo di attributo è molto simile a quella di un attributo di tipo `TEXT.` I valori che gli vengono assegnati sono estratti dalla tabella `Professor` in Figura 2.7, così ad esempio nella pagina relativa al professor Miola all'attributo `Photo` viene assegnato il valore `../icons/m...jpg`.

Focalizziamo l'attenzione sui valori che possono essere assegnati a un attributo di tipo `IMAGE`.

Il valore di un attributo di tipo `IMAGE` deve ovviamente specificare il nome di un file contenente l'immagine da visualizzare nella pagina da istanziare. Il path di questo file può essere specificato in due modi diversi:

1. può essere specificato il path relativo alla posizione della pagina nella quale deve essere visualizzata l'immagine;
2. può essere specificato il path relativo alla root del sito.

2.10 Attributi di tipo LIST-OF

Gli attributi di tipo `LIST-OF` sono attributi che realizzano la nidificazione poiché la loro definizione contiene a sua volta la definizione di altri attributi. Come si può infatti notare dall'esempio che segue la definizione dell'attributo `CourseList`, di tipo `LIST-OF`, contiene la definizione dell'attributo `Course`, di tipo `TEXT`.

Consideriamo la seguente istruzione Penelope:

Esempio 2.6:

```
DEFINE PAGE CourseListPage : "Courses Page"
AS URL("Courses.html");
   CourseList : LIST-OF (
                        Course : TEXT = <CourseName>;
                        );
USING Course
END
```

Questa istruzione Penelope genera, a partire dalla tabella relazionale `Course` (vedi Figura 2.1), una pagina Web avente la struttura descritta dallo schema ADM riportato in Figura 2.8.

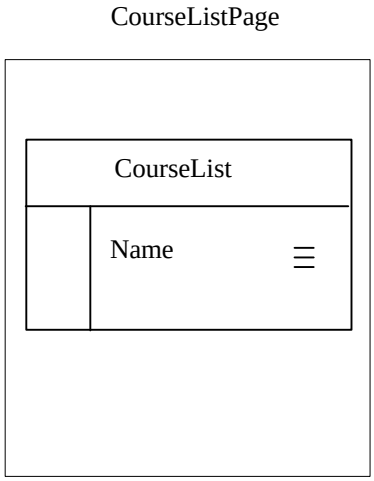

Figura 2.8: Schema ADM della pagina `CourseListPage`

In tale pagina Web viene visualizzata una lista di nomi di corsi, uno per ogni tupla contenuta nella tabella relazionale `Course`. I nomi dei corsi vengono estratti dall'attributo `CourseName`. Una definizione di pagina può avere diversi livelli di nidificazione. Per esempio, possiamo avere una lista di attributi contenente a sua volta un'altra lista di attributi:

```
DEFINE PAGE ResearchGroupList : "List of Research Groups"
AS URL ("index.html");
  GroupList: LIST-OF (
                Name: TEXT = <GroupName>;
                Topics: TEXT= <Topics>;
                MemberList: LIST-OF (
                      ToMember: LINK-TO StudentPage(
```

```
                                        URL(<Name>);
                                        Member: TEXT = <Name>;
                                                  );
                                    );
                        );
USING ResearchGroup,

   PersonGroup: (SELECT PersonInGroup.Name AS Name,
                        PersonInGroup.GroupName AS GroupName,

                        Student.Position AS Position
                 FROM PersonInGroup, Student
                 WHERE  PersonInGroup.Name = Student.Name)
ORDER BY PersonGroup.Name
END
```

In questo caso, stiamo definendo una pagina unica contenente la lista di tutti i gruppi di ricerca. Per ogni item nella lista, riportiamo la lista dei membri con un link alla pagina corrispondente (`StudentPage`). Possiamo notare che l'attributo `MemberList` è nidificato, Penelope permette infatti di definire attributi arbitrariamente nidificati, con qualche accortezza:

primo, dati appartenenti a diversi livelli di nidificazione possono essere estratti da tabelle differenti della clausola `USING`; nell'esempio appena visto i dati vengono estratti da due tabelle differenti. La prima, `ResearchGroup`, ha una tupla per ogni gruppo. La seconda è una vista. Penelope esegue un join naturale tra le due per correlare i dati ai differenti livelli. Nella pagina sopra definita prima di eseguire la nidificazione, viene eseguito un join naturale sull'attributo comune `GroupName` tra le due tabelle `ResearchGroup` e `PersonGroup`.

secondo, per ottimizzare l'esecuzione e l'uso del motore SQL, è necessario applicare una piccola restrizione agli attributi nidificati, diciamo che per ogni livello di nidificazione, tutti gli attributi non nidificati devono essere estratti da un'unica tabella della clausola `USING`; nell'esempio di sopra, i valori per tutti gli attributi non nidificati

(testi, immagini o links) in `Grouplist`, `Name` e `Topics`, vengono estratti dalla tabella `ResearchGroup`; similmente, nell'attributo `MemberList`, i valori di tutti gli attributi non nidificati, `ToMember` e `Member` vengono estratti dalla vista `PersonGroup`. Notiamo che, per essere correttamente unita in join con la tabella `ResearchGroup` di sopra, `PersonGroup` deve avere un attributo `GroupName`.

Il seguente esempio illustra meglio tali concetti.

```
L0 : LIST-OF (
        A01: …;
        A02: …;
        …: …;
        A0n: …;
 L1 : LIST-OF (
                A11: …;
                A12: …;
                …: …;
                A1m: …;
               );
L2 : LIST-OF ( … );
….
Lk : LIST-OF ( … );
);
```

Penelope consente di generare pagine con una struttura arbitrariamente nidificata, ad esempio l'attributo di tipo lista L0, conterrà in generale, una serie di attributi non nidificati (testo, immagini, link) A01, A02, …, A0n, ed una serie di attributi nidificati ovvero altre liste, L1, L2, …, Lk. In genere ciascuno degli attributi Aij corrisponderà ad un diverso attributo della base di dati; e siccome nella clausola `USING` ci possono

essere più tabelle (o viste) questi attributi, della base di dati, possono provenire da tabelle diverse.

C'è però una piccola restrizione: per ciascuna delle liste Li, gli attributi della base di dati devono appartenere tutti alla stessa tabella (o vista); questo vuol dire, per esempio, che nella clausola `USING` deve esisterci una tabella o vista T0 che contiene tutti gli attributi corrispondenti a A01, A02, …, A0n (nel gergo di Penelope T0 si chiama *tabella dominante* per la lista L0), ed un'altra T1 che contiene tutti gli attributi da mettere in corrispondenza con A11, A12, …, A1m; e così via. Viceversa, non è possibile mettere A0i (per i=0, …, n) in corrispondenza con un attributo di T1 e A1j (per j=0, …, m) con una attributo di T0.

Si è scelta tale strategia per motivi di efficienza, ovvero per evitare di effettuare nest complessi, che computazionalmente sono molto onerosi. Penelope effettua il nest nel seguente modo: se i dati per L0 vengono presi da T0 e i dati di L1 vengono presi da T1, la nidificazione viene fatta effettuando l'outer join tra T0 e T1, per cui le tuple di T1 vengono nidificate all'interno di T0. In questo modo se, per esempio, T0 contiene le informazioni riguardanti i professori e T1 i loro corsi, ed entrambi hanno un attributo nome_docente, il join verrà fatto su questo attributo, in modo da associare a ciascun professore i propri corsi. Tale scelta è stata fatta per semplificare la sintassi: in questo modo non è necessario specificare al sistema quali sono gli attributi su cui bisogna fare il join tra le varie tabelle dominanti. Per tale restrizione bisogna fare in modo che l'outer join dia il risultato corretto; per esempio occorre evitare che T0 e T1 abbiano entrambe un attributo con lo stesso nome, ma con significato diverso: infatti questo attributo verrebbe coinvolto nel join, creando quindi un risultato scorretto.

2.11 Attributi di tipo LINK-TO

Con il linguaggio Penelope è possibile definire degli attributi di tipo `LINK-TO` tramite i quali si possono collegare le pagine del sito. Un attributo di tipo `LINK-TO` possiede un `URL` e un'ancora.

Consideriamo la seguente istruzione Penelope:

Esempio 2.7:

```
DEFINE PAGE CoursePage : <CourseName>
AS URL(<CourseName>);
   Name : TEXT = <CourseName>;
   Description : TEXT = <Description>;
   Type : TEXT = <Type>;
   ToProfessor : LINK-TO ProfessorPage (
                        URL(<Professor>);
                  ProfessorName : TEXT = <Professor>;
                                       );

USING Course
END
```

Questa istruzione Penelope crea le istanze del page-scheme `CoursePage` descritto dallo schema `ADM` in Figura 2.10, con dati estratti dalla relazione `Course` in Figura 2.9.

CourseName	Description	Professor	Type
Compilers	...	Torlone	graduate
DatabaseSystem	...	Atzeni	graduate
ArtificialIntelligence	...	Cialdea	graduate
...	...	...	...

Figura 2.9: Tabella relazionale `Course`

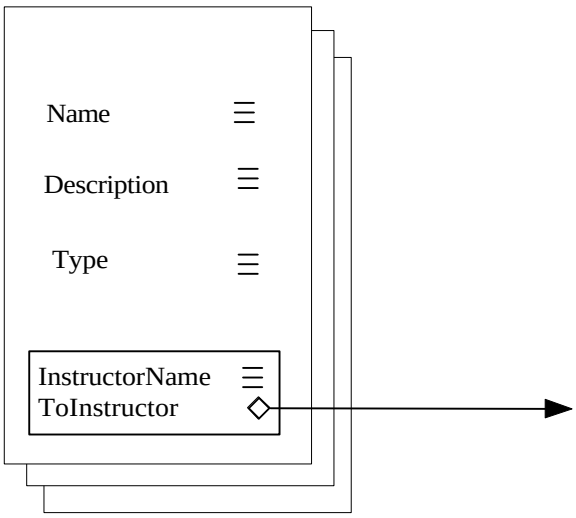

Figura 2.10: Schema ADM della pagina `CoursePage`

Con riferimento all'attributo `ToProfessor` analizziamone la sintassi a piccoli passi. Per prima cosa s'incontra il nome dell'attributo con la dichiarazione del tipo, seguito dal nome del page-scheme che definisce il tipo pagina alla quale il link deve

puntare, nel nostro caso `ProfessorPage` (vedi Esempio 2.7), quindi segue la funzione `URL()`. Se, come deve essere, tra la relazione `Course` e la relazione `Professor` si ha un vincolo di integrità referenziale sugli attributi `Name` e `Professor`, gli `URL` dell'attributo `ToProfessor` verranno creati dalla funzione `URL()` a partire dagli stessi valori con cui vengono creati gli `URL` delle pagine dei professori. Ad esempio sia l'`URL` della pagina del professor Atzeni, sia gli `URL` dell'attributo `ToProfessor` relativi ai corsi "Basi di Dati" e "Sistemi Informativi" sono creati a partire dallo stesso valore ("Paolo Atzeni") e daranno quindi origine allo stesso risultato (site-path/ProfessorPage/PaoloAtzeni.html).

Questo semplice meccanismo di generazione degli `URL` a partire da valori della base di dati, se usato correttamente, permette di mantenere consistenti tutti i link anche in siti di grandi dimensioni e con struttura complessa. L'ultimo elemento che s'incontra

nella definizione è l'ancora, questa può essere un attributo di tipo `TEXT` oppure un attributo di tipo `IMAGE`; nel nostro esempio l'ancora è rappresentata dall'attributo `ProfessorName.`

2.11.1 Link esterni

Con Penelope è possibile generare anche link a pagine non appartenenti al sito che si sta definendo e che quindi già esistono e hanno un proprio indirizzo (che necessariamente deve essere noto). La sintassi che definisce tali link è praticamente la stessa di quella appena descritta. Un esempio di link esterno è riportato nel page-scheme che segue:

Esempio 2.8:

```
DEFINE PAGE ProfessorPage : <Name>

AS URL(<Name>);
   Name : TEXT = <Name>;
   Email : TEXT = <email>;

   Phone : TEXT = <Phone>;
   Photo : IMAGE =<Photo>;
   ToPubblications : LINK-TO ExternalPage (
                  URL(<Pubblicatios>);
                  Pubblications : TEXT = “Publications;”
                                        );
USING Professor
END
```

Facendo riferimento all'attributo `ToPublications` cerchiamo di evidenziare ciò che differenzia un link esterno da un link come quello analizzato nel paragrafo precedente e che potremmo definire interno:

il nome della pagina a cui il link deve puntare, in questo caso `ExternalPage,` è assolutamente arbitrario, il parametro passato alla funzione URL deve coincidere con l'indirizzo (ad esempio http:\\www.uniroma3.it\DiBattista\Publications) della pagina alla quale si vuole puntare; come al solito il valore del parametro può essere contenuto nella base di dati oppure può essere direttamente specificato come valore costante. Quando Penelope si accorge che deve costruire un `URL` a partire da un valore che rappresenta un indirizzo capisce che tale `URL` si riferisce a un link esterno. In questo caso l'url generato da Penelope non sarà del tipo:

site-path\ExternalPage\http:\\ www.uniroma3.it \Atzeni\Publications

bensì avrà un valore proprio uguale a:

http:\\www.uniroma3.it\Atzeni\Pubilcations;

un simile meccanismo è utilizzato anche per indirizzi email (che cominciano con il prefisso mailto), riferimenti a file (che cominciano con il prefisso ../ oppure /), o indirizzi ftp (che cominciano con il prefisso ftp://).

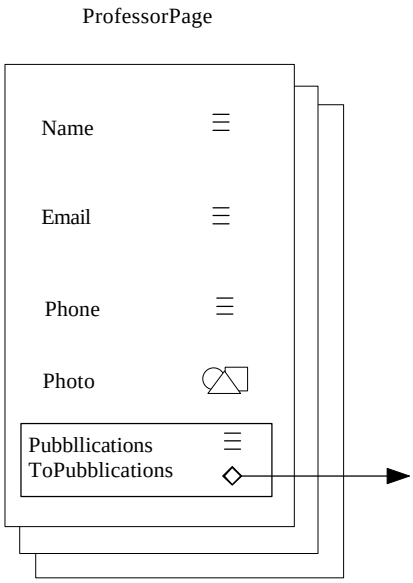

Figura 2.11: Schema ADM della pagina `ProfessorPage`

2.11.2 Link con offset

Un ultimo aspetto dei link da analizzare è l'indirizzamento con offset che permette di raggiungere uno specifico punto all'interno di una pagina web.

Consideriamo le due seguenti istruzioni Penelope:

Esempio 2.9:

```
DEFINE PAGE EducationPage : "EducationPage"
AS URL("education");
   CourseList : LIST-OF (
                        Name : TEXT = <CourseName>, OFFSET(<CourseName>);
                        Description : TEXT = <Description>;

                        ToProfessor : LINK-TO ProfessorPage(
                                              URL (<Professor>);
                                                ProfessorName : TEXT = <Professor>;
                                             );
```

```
                    );
USING Course
END
```

Esempio 2.10:

```
DEFINE PAGE ProfessorPage : <Name>
AS URL(<Name>);
   Name : TEXT = <Name>;
   Email : TEXT = <email>;
   Phone : TEXT = <Phone>;
   Photo : IMAGE =<Photo>;
   CourseList: LIST-OF(
       ToCourse : LINK-TO EducationPage (
                   URL("education.html" # <CourseName>);
                   Course : TEXT = <CourseName>;
                                   );
                       );
USING Professor,
      MyCourse: (SELECT CourseName, Professor AS Name
                  FROM Course )
END
```

Focalizziamo l'attenzione sull'attributo `ToCourse` di `ProfessorPage` e sull'attributo `Name` di `EducationPage`. Il link `ToCourse` punta alla pagina `EducationPage` con offset definito dal parametro `<CourseName>` che viene preceduto dal simbolo terminale '#'. Affinché tale indirizzamento sia efficace è necessario che nella pagina `EducationPage` si crei un riferimento. Per realizzare ciò, poiché nel nostro caso si vuole che `ToCourse` punti al nome del corso, la dichiarazione dell'attributo `Name` deve essere seguita dalla funzione `OFFSET` alla quale deve essere passato come parametro lo stesso parametro di `offset` passato alla funzione `URL` relativa all'attributo `ToCourse`.

2.11.3 LINK-TO UNION

Consideriamo la seguente istruzione Penelope:

Esempio 2.11:

```
DEFINE PAGE CuorseListPage : "CourseListPage"
AS URL("CourseList.html");
  CourseList :  LIST-OF (
   ToCourse : LINK-TO
           UnderGraduateCoursePage  UNION
           GraduateCoursePage IF Type LIKE 'graduate'
                           (
                             URL(<CourseName>);
                             Course : TEXT =<CourseName>;
                            );
                       );
USING Course
END
```

Essa genera la pagina descritta in Figura 2.12 a partire dalla tabella relazionale Course di Figura 2.13. Questa pagina, come è facile verificare, permette l'accesso alle singole pagine dei corsi mediante una lista di link. Una particolarità che possiamo evidenziare in questo esempio è l'uso della clausola `UNION` nella definizione del link `ToCourse`. Tale link punta a due diversi tipi di pagina , la `UnderGraduateCoursePage` e la `GraduateCoursePage`.

Per poter utilizzare la clausola `UNION` è necessario che la tabella della base di dati che viene utilizzata per creare il link abbia un attributo che permetta di distinguere le tuple relative ai corsi `graduate` da quelle relative ai corsi `undergraduate`. In tal caso la condizione `IF` permette a Penelope di capire se creare un link verso una `UnderGraduateCoursePage` oppure verso una `GraduateCoursePage`.

Per default (ovvero se nessuna condizione IF è soddisfatta) viene selezionato il page-scheme relativo alla prima condizione IF soddisfatta.

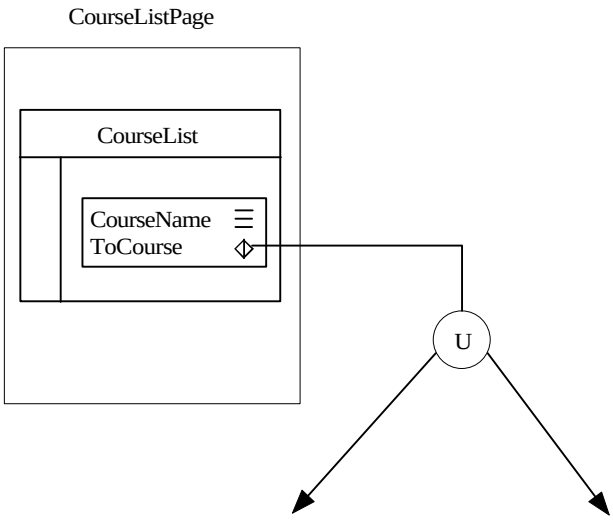

Figura 2.12: Schema ADM della pagina `CourseListPage`

CourseName	Description	Professor	Type
Compilers	...	Rossi	undergraduate
DatabaseSystem	...	Atzeni	graduate
ArtificialIntelligence	...	Cialdea	graduate
...	...	...	...

Figura 2.13: Tabella relazionale `Course`

2.12 Attributi di tipo MAP

Un attributo di tipo MAP rappresenta una mappa *cliccabile* ovvero una immagine suddivisa in più aree a ciascuna delle quali è associato un link. Gli elementi costituenti una mappa *cliccabile* sono tre:

1. l'immagine che è visualizzata dal browser;

2. le zone in cui è suddivisa l'immagine. Ogni zona è delimitata dalla posizione di 2 punti: l'angolo in alto a sinistra (S) e quello in basso a destra (D); per specificare la zona è quindi sufficiente utilizzare la lista delle coordinate dei 2 punti, lista che

3. ha il seguente formato Sx,Sy,Dx,Dy.
4. i link associati a ciascuna zona.

Di seguito riportiamo un esempio di definizione di attributo di tipo Map:

```
ClickMap : MAP(
        Immagine   : IMAGE = <Icon>;
        TargetLink : LINK-TO  PageSchema1 UNION
                              PageScheme2
                             (
                               URL (<Target>);
                               Coordinate : TEXT = <Coord>;
                             );
         [ALT :.....    ]*
         [AREASHAPE: (CIRCLE|POLY)]*
              );
```

Sulla base dell'esempio e delle considerazioni poc'anzi riportate, un attributo di tipo MAP è costituito da un attributo di tipo immagine (sempre specificato per primo) che indica l'immagine costituente la mappa e un attributo di tipo link che specifica i page-scheme di destinazione, le zone e i link associati. Possono inoltre essere presenti due attributi opzionali ALT e AREASHAPE. Quest'ultimo può assumere tre valori :

1. **poly**: ritaglia immagini frastagliate attraverso linee rette;
2. **rect**: ritaglia aree quadrate o rettagolari di 4 lati;
3. **circle**: ritaglia aree circolari;

Per default se non è presente l'attributo AREASHAPE si assume che il suo valore sia di tipo **rect**.

2.13 Attributi di tipo FORM

Gli attributi di tipo FORM sono introdotti al fine di modellare i comandi <FORM> del linguaggio HTML. Un attributo di tipo FORM è un attributo complesso, simile ad una lista, e può contenere all'interno attributi di tipo TEXT, IMAGE, MAP, LIST-OF ed alcuni particolari tipi che sono descritti di seguito. All'interno di un tipo FORM non può essere definito un altro attributo dello stesso tipo.

Vediamo di seguito un esempio:

```
NameForm : FORM (
action    : TEXT      = ”/cgi-bin/script”;
method    : TEXT      = “POST”;
enctype   : TEXT      = “data_type”;
inputText : TEXTLINE  = <valueText>, SIZE "20" "40";
inputPwd  : PASSSWORD = <valuePwd> , SIZE "20" "40";
inputHide : HIDDEN    = <valueHide>;
inputBox  : CHECKBOX  = <valueBox>   CHECKED ;
inputRadio: RADIO     = <valueRadio> CHECKED ;
inputRes  : RESET     = <valueRes> ;
inputSub  : SUBMIT    = <valueSub> ;
areaName  : TEXTAREA  = <areaValue><wrapValue>,SIZE "15" ”80";
inputButt : BUTTON    = <valueButt> <event><buttonFunction>;
selectName: SELECT (
             item1:OPTION="valueOpt1" "descOpt1" DISABLED;

             item2:OPTION="valueOpt2" "descOpt2" ;
             item3:OPTION="valueOpt3" "descOpt3" SELECTED;
                    )
```

```
MULTIPLE , SIZE="4";
  );
```

Nella figura 2.14 è raffigurato un page scheme con un attributo FORM.

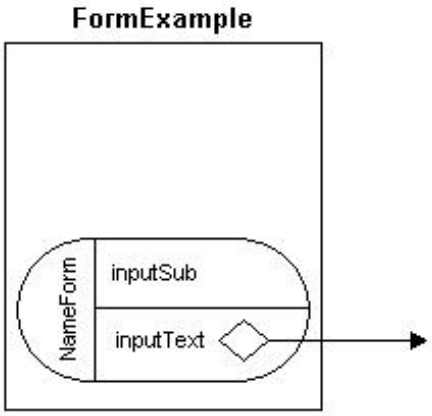

Figura 2.14: page-scheme contenente un attributo di tipo FORM

Action

È un parametro opzionale che indica l'indirizzo (URL) del programma o script associato alla form .

Method

È un parametro opzionale che indica il metodo utilizzato per inviare i parametri al server HTTP; gli unici valori significativi sono "GET" e "POST".

Enctype

È un parametro opzionale che indica il tipo di formato utilizzato per inviare i parametri al server HTTP (MIME-type).

TEXTLINE

È un tipo di attributo che modella gli elementi della form definiti dal comando <INPUT TYPE="TEXT">, ovvero campi di testo modificabile dall'utente. Per attributi di questo tipo è possibile specificare la dimensione con cui visualizzare il campo ed il numero massimo di caratteri che possono essere inseriti nel campo stesso (entambi i

parametri sono opzionali).

PASSWORD

È simile al tipo TEXTLINE descritto precedentemente; è utilizzato per modellare il comando HTML <INPUT TYPE="PASSWORD">, ovvero un campo di testo simile nel quale i caratteri digitati sono visualizzati con asterischi (*). Per attributi di questo tipo è possibile specificare la dimensione con cui visualizzare il campo ed il numero massimo di caratteri che possono essere inseriti nel campo stesso (entambi i parametri sono opzionali).

TEXTAREA

È un tipo di attributo che modella gli elementi della form definiti dal comando <TEXTAREA>, ovvero campi di testo modificabile dall'utente. Per attributi di questo tipo è possibile specificare la dimensione con cui visualizzare il campo espressa come numero di righe e di colonne (entrambi i parametri sono opzionali).

HIDDEN

È un tipo di attributo che modella gli elementi della form definiti dal comando <INPUT TYPE="HIDDEN">, si tratta di una stringa di testo che non viene visualizzata dal browser.

CHECKBOX

È un tipo di attributo che modella gli elementi della form definiti dal comando <INPUT TYPE="CHECKBOX">. Si tratta di interruttori ovvero elementi che possono

assumere valore on/off. Per attributi di questo tipo è possibile specificare o meno il valore di default (CHECKED corrisponde ad on).

SELECT

È un tipo di attributo che modella gli elementi della form definiti dal comando <SELECT>. Si tratta di un menu che consente di selezionare uno o più voci (in questo secondo caso è necessario specificare la parola chiave MULTIPLE). Le voci del menù sono costituite da attributi di tipo OPTION. Per attributi di questo tipo è possibile specificare il numero di voci che possono essere visualizzate contemporaneamente (anche in questo caso il parametro SIZE è opzionale).

OPTION

È un tipo di attributo che modella gli elementi della form definiti dal comando <OPTION>. Ad elemento, che corrisponde alla voce di un menù, è associato un valore ed una descrizione: la descrizione viene visualizzata dal browser, mentre il valore viene trasmesso al server HTTP. Per attributi di questo tipo è possibile specificare sono selezionati (SELECTED) o disabilitati (DISABLED) per default.

RADIO

È un tipo di attributo che modella gli elementi della form definiti dal comando <INPUT TYPE="RADIO">. Si tratta di interruttori ovvero elementi che possono assumere valore on/off. Per attributi di questo tipo è possibile specificare o meno il valore di default (CHECKED corrisponde ad on).

RESET

È un tipo di attributo che modella gli elementi della form definiti dal comando <INPUT TYPE="RESET">. Si tratta di un pulsante che ristabilisce i valori di default per tutti gli elementi della form.

SUBMIT

È un tipo di attributo che modella gli elementi della form definiti dal comando

<INPUT TYPE=”BUTTON”>. Si tratta di un pulsante per il quale sono definiti un insieme di eventi possibili (`event`) e di azioni che sono associate al verificarsi degli eventi stessi (`buttonFunction`).

Gli attributi visti nell’esempio precedente generano i seguenti comandi HTML.

```
<FORM name="NameForm" method="POST" action="cgi_url" enctype="data_type">
  <INPUT TYPE="text" name="inputText" value="valueText" SIZE=20 MAXLENGTH=40>
  <INPUT TYPE="password" name="inputPwd" value="valuePwd" SIZE=20
         MAXLENGTH=40>
  <INPUT TYPE="hidden" name="inputHide" value="valueHide">
  <INPUT TYPE="checkbox" name="inputBox" value="valueBox" CHECKED>
  <INPUT TYPE="radio" name="inputRadio" value="valueRadio" CHECKED>
  <INPUT TYPE="reset" name="inputRes" value="valueRes">
  <INPUT TYPE="submit" name="inputSub" value="valueSub">
  <INPUT TYPE="button" name="inputButt" value="valueButt" event="buttonFunction">
  <TEXTAREA name="areaName" rows=15cols=80 rap="wrapValue">areaValue
          </TEXTAREA>
  <SELECT name="selectName" MULTIPLE SIZE=4>
        <OPTION value="valueOpt1" SELECTED>descOpt1</OPTION>
        <OPTION value="valueOpt2" DISABLED>descOpt2</OPTION>
        <OPTION value="valueOpt3" SELECTED>descOpt3</OPTION>
  </SELECT>
</FORM>
```

2.14 Semantica di Penelope

In questo paragrafo sarà presentata la semantica del linguaggio Penelope. Per rendere più chiara l’esposizione della semantica di ogni passo del processo, Penelope è

stata astratta in un'algebra relazionale nidificata, i cui operatori sono brevemente descritti di seguito.

Al fine di comprendere per intero l'algebra Penelope è necessario conoscere le nozioni della classica algebra relazionale.

2.14.1 L'algebra Penelope

L'algebra Penelope è essenzialmente un'algebra relazionale nidificata estesa con un nuovo operatore URL, che fornisce un meccanismo di manipolazione degli identificatori necessari per la creazione di complesse strutture ipertestuali.

Gli operatori di questa algebra lavorano su relazioni e restituiscono oggetti nidificati che rappresentano pagine Web, nel seguente modo:
consideriamo una relazione R(A,B) in cui A e B siano due insiemi disgiunti di attributi della relazione *nest*, $\acute{\imath}_{A \rightarrow C}$ è un operatore che raggruppa insieme tutte le tuple di R che assumono gli stessi valori sull'insieme di attributi non contenuti nell'insieme A, tale insieme coincide ovviamente con l'insieme B . Questo operatore crea una singola tupla, per ognuno di tali gruppi, tale tupla ha un nuovo attributo chiamato C, in sostituzione dell'insieme di attributi A, il valore che la tupla creata assume sull'attributo C è l'insieme di valori che le tuple di R del gruppo assumevano
sull'insieme di attributi A; *generate URL*, $URL_{A \rightarrow I}$. Questo operatore aggiunge un nuovo attributo, I, alla relazione R; i valori di I sono identificatori. Essi possono essere URLs di pagine o links ad altre pagine. Ogni valore è il risultato dell'applicazione di una funzione di Skolem ai valori degli attributi dell'insieme A; altri operatori relazionali (*selezione*, *proiezione* e *join*) sono definiti come nell'algebra relazionale classica.

2.15 Gestione della presentazione

Finora abbiamo visto come tramite il linguaggio Penelope sia possibile definire lo schema di un sito Web. Più in particolare abbiamo visto come attraverso ogni singolo page-scheme sia possibile definire la struttura logica delle pagine.

Dall'uso del linguaggio Penelope per la creazione di siti di una certa complessità, è

emersa la necessità di scindere la progettazione delle pagine in due fasi:

1. nella prima viene definita la struttura logica,
2. nella seconda viene definita la presentazione.

Per realizzare ciò viene utilizzato Telemaco, uno strumento di supporto a Penelope, che permette una versatile gestione della presentazione dei page-scheme sia in fase di progettazione che in fase di manutenzione del sito.

2.15.1 Approccio di Telemaco

A partire da un'istruzione Penelope priva delle direttive di presentazione, Telemaco, per ogni page-scheme, produce un *template*[5]. Il template di un page-scheme può essere visto come una pagina HTML campione contenente delle variabili, ogni variabile corrisponde a un attributo definito nel page-scheme.

Consideriamo la seguente istruzione Penelope:

```
DEFINE PAGE PaginaRicercatore: "pagina ricercatore"
AS URL(<Cognome>,<Nome>);
   Cognome: TEXT = <Cognome>;
   Nome: TEXT = <Nome>;
   CampoRicerca: TEXT = <CampoRicerca>;
USING PersonaleDidattico
WHERE Titolo LIKE "Ricercatore"
END
```

Di seguito è riportato il sorgente del template associato al page-scheme:

[5] Il template è un semplice file di testo al quale è associata l'estensione .TPL (Template).

```
<HTML>
<TITLE>Template -- Generated by Telemaco</TITLE>
<BODY><P><!--END OF HEADER-->

<!--BEGIN OF BODY-->

$Cognome$<BR>
$Nome$<BR>
$CampoRicerca$<BR>
<!--END OF BODY-->
<!--BEGIN OF FOOTER--><P></BODY>
</HTML>
```

Come si può notare, il template generato contiene un insieme di stringhe racchiuse dai simboli terminali '$' e un insieme di tags HTML.

Le stringhe rappresentano gli attributi del page-scheme al quale il template afferisce, i tags HTML sono invece introdotti da Telemaco al fine di ottenere un template semplice ma che risulti comprensibile qualora venga visualizzato con un browser.

Il *template* visualizzato mediante un browser HTML, mostra quindi la pagina campione che riflette esattamente il risultato finale di ciascuna istanza di pagina.

Nel caso in cui si desideri che un template venga generato con uno stile più complesso è possibile associare al rispettivo page-scheme lo stile con il quale viene creato il corrispondente template. Tale stile deve essere definito in uno style sheet[6], che permette di associare a ogni tipo di attributo un insieme di direttive HTML. In questo modo i template ai quali è stato associato uno stile non vengono creati con lo stile standard usato da Telemaco, bensì con uno stile definito dall'utente e che si suppone definisca una presentazione molto vicina a quella definitiva. Ad esempio si può specificare che il template abbia uno sfondo verde, che gli attributi di tipo TEXT

[6] Il foglio di stile deve essere memorizzato in un file al quale deve essere associata l'estensione .STY (Style).

appaiano in grassetto e con font Arial, che gli items degli attributi di tipo lista siano organizzati come tabelle e così via.

Quello che segue è lo stile che permette di ottenere la formattazione appena descritta:

```
HEADER:[<HTML>

<TITLE>Template -- Generated by Telemaco</TITLE>
<BODY background="../icons/sfondo.gif" bgcolor="#FFFFFF"><P>]
TEXT:[<B><FONT FACE=Arial SIZE=-1 COLOR=blue>]
     [</FONT></B><BR>]
IMAGE:[] [<BR>] [ALT="immagine di prova"]
LINK:[] [<BR>] []
ANCHOR 1:[] []
ANCHOR 2:[] []

/* primo livello di lista */
LIST 1:[<TABLE>] [</TABLE>]
    ITEM LIST 1:[<TR>] [</TR>]
    ITEM ELEMENT LIST 1:[<TD>] [</TD>]
```

I template, una volta generati, possono essere manipolati con lo strumento che si preferisce: dall'editor di testi al più sofisticato WYSWYNG[7] HTML editor.

Ultimata la formattazione dei template, la presentazione di ognuno di essi viene salvata in una opportuna base di dati che poi sarà utilizzata da Penelope per costruire il sito. La base di dati consiste in un set di files[8], uno per ogni page-scheme. Un file ".ATS" memorizza le direttive HTML associate, tramite il template, a ogni attributo definito

[7] What You See Is What You Get cioè ciò che si vede sullo schermo è ciò che si ottiene in un browser web

[8] I files nei quali vengono salvate le direttive HTML dei page-scheme sono dei file di testo ai quali viene

nel page-scheme corrispondente.

associata l'estensione .ATS (Attribute Styles).

2.15.2 Gli style sheets

In Araneus, gli stili sono usati per associare delle direttive HTML ai tipi di attributi. In un sorgente Penelope, uno stile può essere associato all'intero site-scheme oppure a un singolo page-scheme utilizzando la clausola STYLE. Qualora in un sorgente Penelope venga definito uno stile per l'intero sito tale stile sarà associato a tutti i page-schemes esclusi quelli ai quali è associato un proprio specifico stile. Qualora invece non venga definito alcuno stile Telemaco usa un proprio stile di default.

L'argomento della clausola STYLE è il nome del file contenente lo style sheet. È necessario che tale nome contenga il path, assoluto o relativo, del file. Un esempio di path assoluto, per uno stile associato a un page-scheme, potrebbe essere:

```
DEFINE PAGE CoursePage
STYLE C:\programs\AraneusWBMS\styles\course.STY()
AS …
END
```

mentre un esempio di path assoluto, per uno stile associato al site-scheme, potrebbe essere:

```
SCHEME file:/afs/vn.uniroma3.it/group/db/araneus/department/
STYLE C:\programs\AraneusWBMS\styles\department.STY()
ON Department
```