

1

2

3

Idee per  
il tuo futuro

Fiorenzo Formichi  
Giorgio Meini

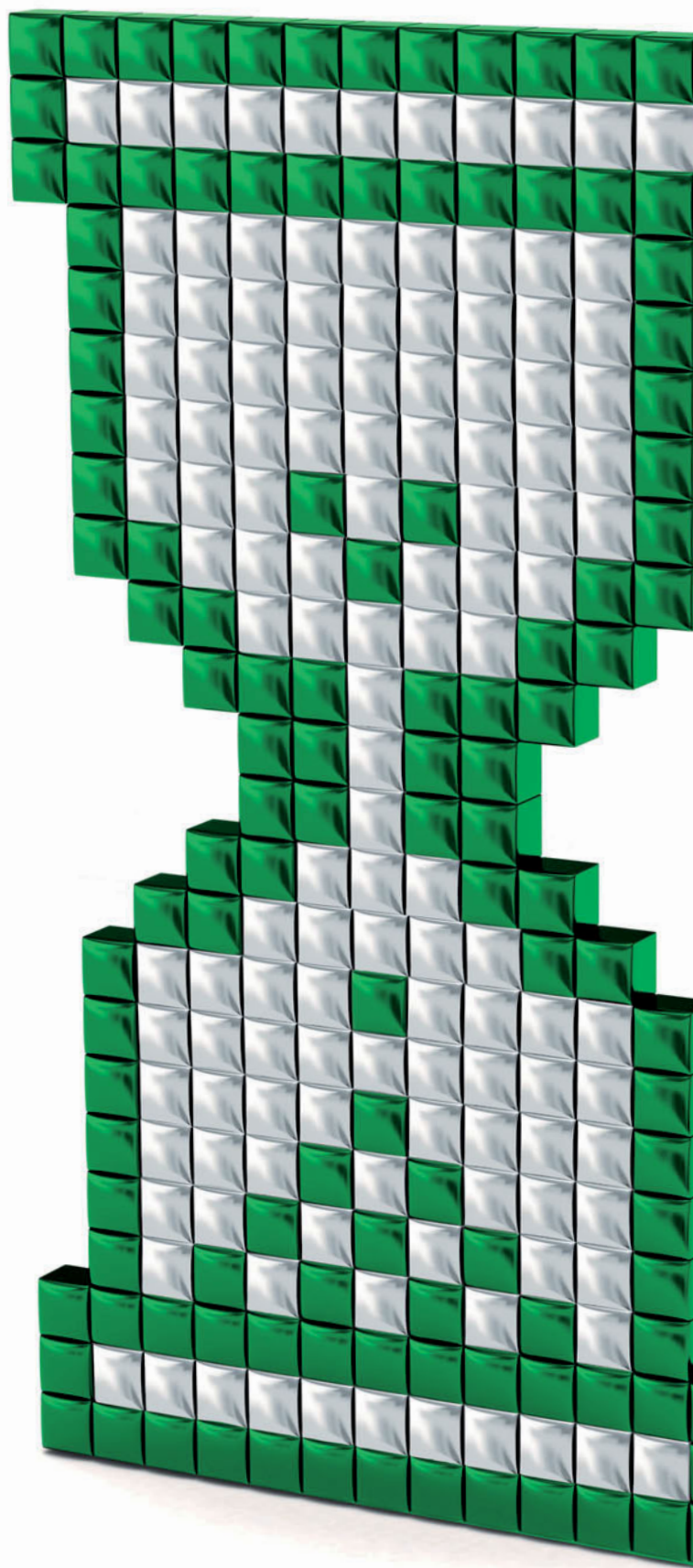
# Corso di informatica

per Informatica

Basi di dati relazionali  
e linguaggio SQL

Linguaggio XML

Pagine web dinamiche  
con linguaggio PHP



**TECNOLOGIA** **ZANICHELLI**

Fiorenzo Formichi

Giorgio Meini

# **Corso di informatica**

per Informatica

Basi di dati relazionali  
e linguaggio SQL

Linguaggio XML

Pagine web dinamiche  
con linguaggio PHP

I diritti di elaborazione in qualsiasi forma o opera, di memorizzazione anche digitale su supporti di qualsiasi tipo (inclusi magnetici e ottici), di riproduzione e di adattamento totale o parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche), i diritti di noleggio, di prestito e di traduzione sono riservati per tutti i paesi.  
L'acquisto della presente copia dell'opera non implica il trasferimento dei suddetti diritti né li esaurisce.

Per le riproduzioni ad uso non personale (ad esempio: professionale, economico, commerciale, strumenti di studio collettivi, come dispense e simili) l'editore potrà concedere a pagamento l'autorizzazione a riprodurre un numero di pagine non superiore al 15% delle pagine del presente volume. Le richieste per tale tipo di riproduzione vanno inoltrate a

Centro Licenze e Autorizzazioni per le Riproduzioni Editoriali (CLEARedi)  
Corso di Porta Romana, n. 108  
20122 Milano  
e-mail [autorizzazioni@clearedi.org](mailto:autorizzazioni@clearedi.org) e sito web [www.clearedi.org](http://www.clearedi.org)

L'editore, per quanto di propria spettanza, considera rare le opere fuori del proprio catalogo editoriale, consultabile al sito [www.zanichelli.it/f\\_catalog.html](http://www.zanichelli.it/f_catalog.html).

La fotocopia dei soli esemplari esistenti nelle biblioteche di tali opere è consentita, oltre il limite del 15%, non essendo concorrenziale all'opera. Non possono considerarsi rare le opere di cui esiste, nel catalogo dell'editore, una successiva edizione, le opere presenti in cataloghi di altri editori o le opere antologiche. Nei contratti di cessione è esclusa, per biblioteche, istituti di istruzione, musei ed archivi, la facoltà di cui all'art. 71 - ter legge diritto d'autore. Maggiori informazioni sul nostro sito: [www.zanichelli.it/fotocopie/](http://www.zanichelli.it/fotocopie/)

#### Realizzazione editoriale:

- Coordinamento redazionale: Matteo Fornesi
- Segreteria di redazione: Deborah Lorenzini
- Progetto grafico: Editta Gelsomini
- Collaborazione redazionale, impaginazione e disegni: Stilgraf, Bologna

#### Contributi:

- *Idee per il tuo futuro*: Laura Mancuso, Lorenzo Lancellotti (testi); Barbara Di Gennaro, Matteo Fornesi (redazione); Miguel Sal & C., Bologna (progetto grafico e impaginazione); Sara Colaone (disegni)

#### Copertina:

- Progetto grafico: Miguel Sal & C., Bologna
- Realizzazione: Roberto Marchetti
- Immagine di copertina: valdis torms/Shutterstock; Artwork Miguel Sal & C., Bologna

Prima edizione: marzo 2013

L'impegno a mantenere invariato il contenuto di questo volume per un quinquennio (art. 5 legge n. 169/2008) è comunicato nel catalogo Zanichelli, disponibile anche online sul sito [www.zanichelli.it](http://www.zanichelli.it), ai sensi del DM 41 dell'8 aprile 2009, All. 1/B.



Zanichelli garantisce che le risorse digitali di questo volume sotto il suo controllo saranno accessibili, a partire dall'acquisto dell'esemplare nuovo, per tutta la durata della normale utilizzazione didattica dell'opera. Passato questo periodo, alcune o tutte le risorse potrebbero non essere più accessibili o disponibili: per maggiori informazioni, leggi [my.zanichelli.it/fuoricatalogo](http://my.zanichelli.it/fuoricatalogo)



#### File per diversamente abili

L'editore mette a disposizione degli studenti non vedenti, ipovedenti, disabili motori o con disturbi specifici di apprendimento i file pdf in cui sono memorizzate le pagine di questo libro. Il formato del file permette l'ingrandimento dei caratteri del testo e la lettura mediante software screen reader. Le informazioni su come ottenere i file sono sul sito [www.zanichelli.it/diversamenteabili](http://www.zanichelli.it/diversamenteabili)

#### Suggerimenti e segnalazione degli errori

Realizzare un libro è un'operazione complessa, che richiede numerosi controlli: sul testo, sulle immagini e sulle relazioni che si stabiliscono tra essi. L'esperienza suggerisce che è praticamente impossibile pubblicare un libro privo di errori. Saremo quindi grati ai lettori che vorranno segnalarceli.

Per segnalazioni o suggerimenti relativi a questo libro scrivere al seguente indirizzo:

[lineazeta@zanichelli.it](mailto:lineazeta@zanichelli.it)

Le correzioni di eventuali errori presenti nel testo sono pubblicate nel sito [www.zanichelli.it/aggiornamenti](http://www.zanichelli.it/aggiornamenti)

Zanichelli editore S.p.A. opera con sistema qualità  
certificato CertiCarGraf n.477  
secondo la norma UNI EN ISO 9001:2008

# Indice

SEZIONE

**A**

## Basi di dati relazionali e linguaggio SQL. Linguaggio XML

### **A1** Sistemi informativi e sistemi informatici

<b>1</b>	Dati e informazione	3
<b>2</b>	Sistemi informativi e sistemi informatici	5
<b>3</b>	Ciclo di vita di un sistema informatico	6
<b>4</b>	Aspetti intensionale ed estensionale dei dati	12
<b>5</b>	File di dati	14
<b>6</b>	Basi di dati e sistemi di gestione delle basi di dati	18
<b>7</b>	Architettura logica di un sistema di gestione delle basi di dati	22
	<b>SINTESI</b>	23
	<b>QUESITI</b>	25
	<b>ORIGINAL DOCUMENT</b>	28

### **A2** Le basi di dati relazionali

<b>1</b>	Diagrammi Entità/Relazioni	33
<b>2</b>	Il modello dei dati relazionale	41
<b>3</b>	Progettazione e normalizzazione di una base di dati relazionale	48
<b>4</b>	Esempi di progettazione di basi di dati relazionali	58
<b>5</b>	Linguaggi per operare su basi di dati relazionali	64
<b>6</b>	Transazioni	67
<b>7</b>	Algebra e operatori relazionali	69
	<b>SINTESI</b>	78
	<b>QUESITI • ESERCIZI</b>	81
	<b>ORIGINAL DOCUMENT</b>	89

## A3 Il linguaggio SQL

1	Il comando SELECT e l'algebra relazionale	94
2	La chiusura del linguaggio SQL e le query nidificate; <i>join</i> e <i>self-join</i>	100
3	Le funzioni di aggregazione e la clausola di raggruppamento	107
4	Operatori di unione, intersezione e differenza	112
5	I comandi DDL del linguaggio SQL: CREATE, ALTER e DROP	114
6	I comandi DML del linguaggio SQL: INSERT, UPDATE, DELETE	124
7	I <i>trigger</i>	126
8	Accesso concorrente ai dati	128
	<b>SINTESI</b>	131
	<b>QUESITI • ESERCIZI • LABORATORIO</b>	133
	<b>ORIGINAL DOCUMENT</b>	142

## A4 Accesso a una base di dati in linguaggio Java con JDBC

1	Architettura client/server e API <i>Java DataBase Connectivity</i>	146
2	Connessione a un DBMS ed elaborazione di comandi e <i>query</i> SQL in linguaggio Java	149
3	Classi CRUD in linguaggio Java; corrispondenza tra tipi SQL e tipi Java	157
4	Uso di oggetti <i>RowSet</i>	167
5	Gestione delle transazioni	176
	<b>SINTESI</b>	177
	<b>QUESITI • ESERCIZI</b>	180

## A5 Il linguaggio XML per la rappresentazione dei dati

1	La sintassi del linguaggio XML e la struttura ad albero dei documenti	185
2	La definizione di linguaggi XML mediante schemi XSD	190
3	Riferimento ai nodi di un albero XML con <i>XPath</i>	208
	<b>SINTESI</b>	215
	<b>QUESITI • ESERCIZI</b>	216
	<b>ORIGINAL DOCUMENT</b>	220

## A6 Gli strumenti per la gestione dei dati rappresentati in linguaggio XML

1	L'interrogazione di basi di dati XML con il linguaggio <i>XQuery</i>	226
2	API per la gestione di documenti XML con il linguaggio Java	236
	<b>SINTESI</b>	253
	<b>QUESITI • ESERCIZI • LABORATORIO</b>	254
	<b>ORIGINAL DOCUMENT</b>	259



## A7 I sistemi di gestione delle basi di dati Microsoft Access e Oracle My-SQL

1	Creazione e gestione di basi di dati in ambiente Access	
2	Definizione di query in ambiente Access	
3	Creazione, gestione e interrogazione di basi di dati in ambiente My-SQL	
4	Amministrazione dei privilegi di accesso degli utenti del DBMS My-SQL	

### SEZIONE

## B

## Pagine web dinamiche con linguaggio PHP

### B1 Il linguaggio PHP e le *form* HTML

1	Architetture software client-server	265
2	La sintassi del linguaggio PHP	269
3	Le variabili del linguaggio PHP	272
4	Gli array del linguaggio PHP	275
5	Le funzioni del linguaggio PHP	279
6	La gestione di <i>form</i> HTML con il linguaggio PHP; validazione dell'input e passaggio di dati tra pagine web	286
7	Gestione dei <i>cookies</i> e delle sessioni in linguaggio PHP	306
	<b>SINTESI</b>	310
	<b>QUESITI • ESERCIZI • LABORATORIO</b>	312
	<b>ORIGINAL DOCUMENT</b>	319

## B2 La programmazione a oggetti nel linguaggio PHP

1	Classi e oggetti nel linguaggio PHP	327
2	Ereditarietà e classi astratte	342
3	Gestione delle eccezioni	348
	<b>SINTESI</b>	351
	<b>QUESITI • ESERCIZI</b>	353

## B3 Accesso a una base di dati in linguaggio PHP

1	L'interfaccia del linguaggio PHP con il DBMS My-SQL	358
2	Gestione degli utenti e delle password con DBMS My-SQL e linguaggio PHP	379
3	PDO (PHP Data Objects) per l'accesso a DBMS	388
	<b>SINTESI</b>	392
	<b>QUESITI • ESERCIZI • LABORATORIO</b>	393
	<b>ORIGINAL DOCUMENT</b>	397



## B4 Sviluppo di pagine web dinamiche con IDE NetBeans

1	Uso dell'IDE NetBeans per la programmazione in linguaggio PHP	
2	Debug di codice PHP in ambiente NetBeans	
3	Gestione del DBMS My-SQL in ambiente NetBeans	

<b>Indice analitico</b>	399
-------------------------	-----



## Esempi di risoluzione delle prove scritte dell'esame di Stato



I capitoli affiancati da questa icona sono disponibili, con chiave di attivazione, all'indirizzo:

[www.online.zanichelli.it/formichimeinicorsoinformatica](http://www.online.zanichelli.it/formichimeinicorsoinformatica)

# Idee per il tuo futuro



## CHE COSA FARÒ DA GRANDE

[www.ideeperiltuofuturo.it](http://www.ideeperiltuofuturo.it)

Sei alla fine del tuo percorso scolastico. Che cosa fare adesso? Iscriverti a un corso universitario? Fare uno *stage* o un corso professionalizzante? Cercare di entrare subito nel mondo del lavoro? Studiare e al contempo lavorare?

Per aiutarti nella scelta ti proponiamo alcuni dati relativi al 2009-2011. È impossibile dire come saranno le cose tra qualche anno, i tempi recenti ci hanno abituati a cambiamenti anche repentini.

**La laurea “paga”.** Una recente ricerca Isfol<sup>1</sup> ha mostrato che chi è laureato ha più possibilità di trovare un’occupazione e in media riceve uno stipendio più alto rispetto a chi possiede soltanto un diploma.

Dal momento che i diplomati entrano nel mondo del lavoro prima dei laureati, inizialmente il tasso di occupazione per i primi è superiore rispetto a quello dei secondi, ma già prima del compimento dei 30 anni chi possiede una laurea ha più possibilità di trovare lavoro, per arrivare nella fascia 34-44 anni, dove il tasso di occupazione dei laureati supera del 7% quello dei diplomati.

In media tra 25 e 64 anni è occupato il 73,1% dei diplomati e il 79,2% dei laureati.

Secondo uno studio OCSE del 2011 i giovani laureati subiscono di più gli effetti della recente crisi economica rispetto ai loro coetanei con istruzione secondaria inferiore<sup>2</sup>.

**Quali lauree valgono un lavoro? Le lauree “brevi” servono?** Le lauree triennali si rivelano molto utili ai fini dell’occupazione: a un anno dal termine degli studi il 42,1% dei laureati triennali lavora, con picchi dell’81,7% per le professioni sanitarie. Tirocini e *stages* sono determinanti per formare e inserire questi laureati nel mondo del lavoro. I tassi di occupazione più alti si hanno tra i medici, seguiti dai laureati in chimica farmaceutica e ingegneria. In generale sono le discipline di tipo scientifico – sia a livello di diploma sia a livello di laurea – le più spendibili nel mondo del lavoro, mentre le discipline umanistiche condannano a una difficile collocazione sul mercato, anche a fronte di un eccesso di offerta di laureati in questi ambiti.

**A Nord c’è più lavoro, ma...** A livello nazionale il tasso di disoccupazione è 7,8%, che sale a 27,4% se si considerano solo i giovani (15-24 anni): più alto al Sud (39,2%), meno al Centro (25,3%), più basso al Nord (19,0%). La situazione per le ragazze è più critica: il tasso della disoccupazione femminile, nella fascia 15-24 anni, supera di circa 8 punti percentuali quello maschile (32,3% per le donne, 23,9% per gli uomini), forbice che si mantiene simile nelle diverse zone geografiche: al Nord il tasso è 22,7% per le donne e 16,4% per gli uomini; al Centro è 34,8% per le donne e 18,7% per gli uomini e a Sud è di 44,0% per le donne e 36,0% per gli uomini.

Tuttavia i dati della disoccupazione giovanile non devono scoraggiare chi cerca lavoro: se la disoccupazione giovanile è del 27,4%, vuol dire che una parte non piccola dei giovani che hanno cercato lavoro (il 72,6%) lo ha trovato<sup>3</sup>. Inoltre i dati variano molto da luogo a luogo e anche all’interno di una stessa regione può esservi una grande varietà di situazioni. L’Emilia-Romagna è tra le regioni in cui la disoccupazione giovanile incide meno, ma con grandi differenze tra le province: se Bologna nel 2010 raggiunge un tasso di disoccupazione di 29,2%, a Piacenza il valore è più che dimezzato (13,6%)<sup>4</sup>.

<sup>1</sup> Tutti i dati sono tratti da una ricerca Isfol con dati relativi al 2010, (l’Isfol, Istituto per lo Sviluppo della Formazione Professionale dei Lavoratori è un ente pubblico di ricerca), e ISTAT del II Trimestre 2011.

<sup>2</sup> Rapporto OCSE *Education at a Glance* 2011.

<sup>3</sup> Dati ISTAT del II Trimestre 2011.

<sup>4</sup> Dati Confartigianato Imprese Emilia-Romagna, 2010.



# COME FUNZIONA L'UNIVERSITÀ

## POSSO ISCRIVERMI ALL'UNIVERSITÀ?

Per iscriversi all'Università è necessario il diploma di maturità quinquennale oppure quello quadriennale con un anno integrativo o, in alternativa, un obbligo formativo aggiuntivo da assolvere durante il primo anno di corso.

## L'Università italiana offre corsi di studio organizzati in tre cicli:

- **laurea**, di durata triennale (180 crediti formativi in un massimo di 20 esami), al termine della quale si consegue il titolo di Dottore; ad esempio laurea in *Tecniche di radiologia medica* o in *Scienze del comportamento e delle relazioni sociali*.
- **Laurea magistrale**, di durata biennale (120 crediti in un massimo di 12 esami), al termine della quale si consegue il titolo di Dottore magistrale; ad esempio laurea in *Biotecnologie mediche* o in *Psicologia clinica*.
- **Dottorato di ricerca e Scuola di specializzazione**.

Esistono anche corsi di laurea magistrali a ciclo unico, della durata di 5 (300 crediti in un massimo di 30 esami) o 6 anni (360 crediti in un massimo di 36 esami); ad esempio *Medicina e Chirurgia*.

Per approfondire gli studi si può accedere a master di 1° e di 2° livello e ai corsi di alta formazione.

I **crediti formativi universitari (CFU)** misurano il carico di lavoro dello studente (1 CFU = 25 ore di impegno; 60 CFU = 1 anno di impegno universitario), compresi lo studio individuale ed eventuali esperienze di apprendistato<sup>5</sup>. Sono stati introdotti per facilitare il confronto tra i sistemi e i programmi di differenti corsi e Atenei italiani ed europei, e quindi il passaggio da un corso di studio a un altro, oppure da un'Università a un'altra, anche straniera: i CFU sono trasferibili in ECTS (*European Credit Transfer and Accumulation System*) e quindi riconosciuti nelle Università di tutta Europa.

Tramite i CFU è possibile valutare ai fini della laurea anche esperienze quali *stages* e tirocini. Infine i CFU permettono di semplificare la determinazione dei **piani di studio individuali (PSI)** che ciascuno studente può modulare su se stesso. In alcuni casi è possibile personalizzare il proprio percorso di studi, inserendo nel piano degli esami da sostenere alcuni corsi non previsti dal piano di studi istituzionale.

Quando si presenta il PSI bisogna rispettare il minimo di crediti obbligatori per ciascun ambito disciplinare previsti dal proprio corso di laurea.

**Vorrei studiare in Europa.** I cittadini dell'Unione europea (UE) possono studiare, dalla scuola primaria al dottorato di ricerca, in uno dei paesi UE.

Per facilitare questi scambi è stato creato Ploteus, il portale delle opportunità di apprendimento ([www.europa.eu/ploteus](http://www.europa.eu/ploteus)): programmi di scambio, borse di studio, descrizioni dei sistemi di istruzione e apprendimento dei vari paesi europei, nonché indicazioni dei siti web degli istituti di istruzione superiore, i database dei corsi di formazione, le scuole... Attraverso Ploteus è possibile anche avere notizie pratiche, ad esempio su come raggiungere la località e dove alloggiare, sul costo della vita, le tasse, i servizi cui si può accedere.



Quanto costa l'Università?

[www.ideeperiluoofuturo.it](http://www.ideeperiluoofuturo.it)



Il mio diploma è riconosciuto in Europa?

<http://www.enic-naric.net/>



Vorrei studiare negli USA

[www.ideeperiluoofuturo.it](http://www.ideeperiluoofuturo.it)

<sup>5</sup> Regolamento recante norme concernenti l'autonomia didattica degli atenei, Decreto Ministeriale 3 novembre 1999, n.509



# DOVE SI STUDIA INFORMATICA

La classe dei corsi di laurea dove l'informatica è il principale oggetto di studio è quella di **scienze e tecnologie informatiche**, che comprende:

- ▶ **Informatica:** fornisce conoscenze di base di algebra, del calcolo differenziale e di quello integrale, affiancate da conoscenze sui linguaggi di programmazione, gli algoritmi e i sistemi operativi; inoltre prevede attività di laboratorio mirate a sviluppare metodi di programmazione e di progettazione dei sistemi informatici. Dopo la laurea triennale esistono diversi corsi magistrali, come quelli in: **sistemi informatici, tecnologie informatiche, modelli computazionali**.
- ▶ **Informatica per il management:** fornisce le conoscenze di base nel campo dell'informatica e dell'utilizzo di internet in ambito sociale ed economico. Il laureato acquisisce le capacità per applicare sistemi informatici nell'organizzazione e comunicazione di informazioni e nei relativi processi di elaborazione in ambito aziendale.
- ▶ **Comunicazione digitale:** fornisce le conoscenze teoriche e pratiche per progettare, realizzare, organizzare e gestire la comunicazione attraverso sistemi informatici, telematici e multimediali.
- ▶ **Informatica musicale:** mira a formare professionisti qualificati nell'area della comunicazione musicale che trovano la loro collocazione nei campi dell'editoria, dei nuovi media, del commercio, della televisione e della pubblicità.



Per saperne di più

[www.progettolauree.scientifiche.eu](http://www.progettolauree.scientifiche.eu)

Corsi di laurea particolari che si occupano di aspetti legati all'informatica sono:

- ▶ **Tecnologie web e multimediali:** fornisce competenze per operare nel campo della progettazione, dello sviluppo, della gestione e della manutenzione di applicazioni internet e multimediali. Il laureato trova impiego sia in ditte specializzate nel settore, sia nei gruppi editoriali, nelle agenzie di marketing o di pubblicità.
- ▶ **Comunicazione multimediale e tecnologie dell'informazione:** affianca le competenze tecnologiche con quelle umanistiche al fine di sviluppare la capacità di progettare e gestire nuovi media digitali.

Esistono numerosi indirizzi del corso di laurea in ingegneria nei quali si affronta l'informatica in modo approfondito.

- ▶ Ingegneria informatica
- ▶ Ingegneria elettronica
- ▶ Ingegneria delle telecomunicazioni
- ▶ Ingegneria elettronica e delle telecomunicazioni
- ▶ Ingegneria dell'automazione
- ▶ Ingegneria biomedica

Infine esistono corsi di laurea, come quello di **matematica** e di **fisica**, in cui occorre sostenere un esame obbligatorio di informatica, mentre nella maggior parte degli altri, come **economia, scienze politiche e lingue straniere**, gli studenti devono superare la prova di idoneità informatica. Questa prova obbligatoria è volta ad accertare le conoscenze informatiche di base e, pur prevedono una valutazione positiva o negativa, non dà luogo ad alcuna votazione.



Qui trovi tante informazioni in più e le prove assegnate negli ultimi anni

<http://accessoprogrammato.miur.it>



Qui trovi tante informazioni in più e degli esempi di test

[www.cisiaonline.it](http://www.cisiaonline.it)

# I TEST DI AMMISSIONE

L'accesso ad alcuni corsi di laurea è filtrato da una prova di ammissione, per iscriversi alla quale occorre versare un importo (attorno ai 60 euro): sono medicina e chirurgia, odontoiatria e protesi dentaria, medicina veterinaria, le lauree a ciclo unico finalizzate alla formazione in altre professioni sanitarie e in architettura.

## Informatica

Per gli studenti che desiderano iscriversi ad uno dei corsi di laurea di informatica esiste un test di accesso obbligatorio ai fini dell'immatricolazione. La prova d'ingresso ha carattere nazionale e i quesiti sono a cura della Conferenza dei Presidi delle Facoltà di Scienze MM.FF.NN. Il modulo di logica e matematica di base è costituito da 25 domande a risposta multipla, con una soglia di sufficienza di 13 risposte corrette. L'esito negativo non preclude l'iscrizione degli studenti al corso di laurea, ma assegna loro un obbligo formativo aggiuntivo (OFA), che può prevedere il superamento di ulteriori prove entro certi limiti di tempo e che può condizionare il proseguimento degli studi.

Per informazioni dettagliate ed esempi di test con le relative soluzioni si consiglia di consultare il sito <http://www.testingressoscienze.org/>.

Di seguito sono riportati alcuni quesiti delle varie sezioni della prova di ammissione alle facoltà di Scienze MM.FF.NN. assegnata dall'Università di Bologna nel 2009.

**01** Siano dati due numeri reali qualsiasi  $a, b$ . Quale delle seguenti identità è falsa?

- ☐ a  $(a \times b)^2 = a^2 \times b^2$
- ☐ b  $a : a = 0$
- ☐ c  $a + b = b + a$
- ☐ d  $(a - b) \times (a + b) = a^2 - b^2$

**02** Si consideri l'affermazione "Condizione sufficiente affinché una persona abbia la patologia P, è che esibisca il sintomo S". Questo significa che:

- ☐ a Tutte le persone che hanno la patologia P esibiscono il sintomo S
- ☐ b Tutte le persone che esibiscono il sintomo S hanno la patologia P
- ☐ c Esiste almeno una persona con il sintomo S che non ha la patologia P
- ☐ d Se una persona ha la patologia P, allora esibisce il sintomo S

**03** Se modifico i lati  $a$  e  $b$  di un rettangolo allungandoli del 10%, l'area del rettangolo aumenta

- ☐ a del 10%
- ☐ b del 20%
- ☐ c dell'11%
- ☐ d del 21%

## Ingegneria

Gli studenti che desiderano immatricolarsi ai corsi di laurea di ingegneria devono sostenere una prova di ammissione, che si svolge con modalità identiche per tutti i differenti corsi di laurea. Questo test iniziale ha uno scopo orientativo per alcuni corsi, mentre ha una vera e propria funzione selettiva per i corsi a numero chiuso (come quello di Ingegneria Edile-Architettura) dove la disponibilità è inferiore al numero degli aspiranti. La prova consiste nel rispondere ad una serie di 80 quesiti a risposta multipla, suddivisi in cinque aree tematiche. Ogni quesito prevede cinque risposte, delle quali solamente una è corretta. La risposta giusta attribuisce 1 punto, l'errore penalizza di -0,25 punti e l'assenza di risposta vale 0 punti. La prova è divisa in 5 sezioni così strutturate: 15 quesiti di logica, 15 domande di comprensione del testo, 20 quesiti di matematica, 20 quesiti di scienze fisiche e chimiche e ulteriori 10 quesiti di matematica. Il tempo fornito per svolgere ogni sezione della prova è prestabilito e viene indicato dai singoli atenei.

Di seguito sono riportati alcuni quesiti delle varie sezioni della prova di ammissione alle facoltà di ingegneria del 2005.

**01** Un circuito è costituito da una batteria da 36 V, un gruppo di due resistenze in parallelo da 6  $\Omega$  e da 3  $\Omega$  rispettivamente, una resistenza in serie di valore  $R$  sconosciuto. In queste condizioni la corrente circolante è 3 A. Assumendo che la resistenza interna della batteria sia trascurabile, il valore della resistenza  $R$  è:

- ☐ a 10  $\Omega$
- ☐ b 2  $\Omega$
- ☐ c 12  $\Omega$
- ☐ d 18  $\Omega$
- ☐ e 4  $\Omega$

**02** Una squadra di operai deve asfaltare una piazzola circolare. Arrivata sul posto, scopre che la piazza ha diametro doppio del previsto. Quanto asfalto serve, rispetto a quello preventivato?

- ☐ a Non si può rispondere se non si conosce o il raggio previsto o quello effettivo
- ☐ b Una quantità  $\pi^2$  volte quella prevista
- ☐ c Il doppio
- ☐ d Il quadruplo
- ☐ e Una quantità  $2\pi$  volte quella prevista

**03** Aldo, Bea, Carlo, Dario, Ebe, Franco vanno in treno e trovano uno scompartimento a sei posti libero. Considerando che Aldo e Bea devono stare vicino al finestrino, quanti modi diversi hanno i sei amici di disporsi nello scompartimento?

- ☐ a 48
- ☐ b 4
- ☐ c 240
- ☐ d 8
- ☐ e 10

# VERSO IL LAVORO

Vorresti trovare lavoro? Nelle pagine che seguono trovi informazioni su come e dove cercare lavoro, cos'è lo stage, come scrivere un curriculum e una lettera di accompagnamento, come sostenere un colloquio. Sul sito [www.ideeperiltuofuturo.it](http://www.ideeperiltuofuturo.it) trovi tante informazioni utili e dettagliate in più per aiutarti nella tua ricerca in Italia e all'estero: i centri per l'impiego e i Career days, siti internazionali, una panoramica dei contratti di lavoro e altro ancora.



Vuoi cercare lavoro all'estero?

[www.  
ideeperiltuofuturo.it](http://www.ideeperiltuofuturo.it)

**La ricerca di lavoro in Italia.** Per mettere in contatto domanda e offerta di lavoro esistono in Italia numerosi soggetti, sia pubblici sia privati, autorizzati dallo Stato a svolgere servizi di intermediazione e collocamento. Sono i **Centri per l'impiego (CIP)**, le **Agenzie per il lavoro**, la **Borsa continua nazionale del lavoro (BCNL)** e il portale «**Cliclavoro**». Anche le scuole secondarie di secondo grado, le Università, i comuni, le associazioni dei datori di lavoro e dei lavoratori, i patronati, i gestori di siti internet possono svolgere attività di intermediazione, purché non abbiano fini di lucro.

**Cercare lavoro tra le pagine dei giornali.** Un canale tradizionale ma sempre valido per chi cerca annunci di lavoro è rappresentato da supplementi e inserti delle maggiori testate a diffusione nazionale e dai giornali specializzati; ne segnaliamo alcuni fra i principali:

- il supplemento «Tutto Lavoro» del lunedì della «Stampa»;
- le pagine dedicate al lavoro il giovedì dalla «Repubblica»;
- il supplemento «Corriere lavoro», con la sezione «Trovo Lavoro», del «Corriere della Sera» del venerdì;
- il supplemento «Carriere&Lavoro» del «Sole 24Ore» del venerdì tocca temi che relative al nuovo mercato del lavoro attraverso inchieste e dossier, e fornisce strumenti e notizie utili per cambiare mestiere e migliorare la propria carriera.

Fra i giornali specializzati:

- il settimanale «Trova Lavoro» con annunci dall'Italia e dall'estero e una selezione dei concorsi tratti dalla Gazzetta Ufficiale;
- «Walk on Job», un bimestrale distribuito gratuitamente in 41 città italiane, che dà spazio al mondo del lavoro e della formazione, con inchieste, interviste, notizie e opportunità prima e dopo la laurea;
- il mensile «Bollettino del Lavoro».

**Cercare lavoro online.** Accanto alla versione cartacea dei supplementi dei giornali, si trova anche la versione online, col vantaggio di consentire un aggiornamento continuo degli annunci, l'inserimento immediato del proprio curriculum in apposite banche dati, di inviare direttamente la propria candidatura in risposta alle offerte di lavoro, di ricevere gli annunci sulla propria e-mail.

Tra le versioni online segnaliamo «Job24» del «Sole 24Ore» e «MioJob» della «Repubblica». Tra i più importanti (e seri) siti per la ricerca di lavoro indichiamo **Monster** ([www.monster.it](http://www.monster.it)) e **Infojobs** ([www.infojobs.it](http://www.infojobs.it)). Da consultare è anche il sito [www.concorsi.it](http://www.concorsi.it), che informa sui concorsi pubblici banditi in Italia. Per quanto riguarda i *social network* professionali si segnalano **LinkedIn** ([www.linkedin.com](http://www.linkedin.com)) e **Xing** ([www.xing.com](http://www.xing.com)) che, oltre a funzionalità come «find job» offrono la possibilità di entrare a far parte di gruppi di discussione utili alla crescita professionale.

## LA TOP TEN DEI LAVORI IN ITALIA

Non hai un'idea precisa di cosa vorresti fare? Alcune figure professionali sono molto ricercate in Italia, ecco la top ten dei profili lavorativi più ricercati in Italia nel 2011, secondo il quotidiano «Il Sole 24 Ore».

- 1) Farmacista
- 2) Progettista settore metalmeccanico
- 3) Infermiere
- 4) Addetto consulenza fiscale
- 5) Sviluppatore software
- 6) Progettista meccanico
- 7) Educatore professionale
- 8) Addetto logistica
- 9) Disegnatore tecnico Cad-Cam
- 10) Fisioterapista

(Fonte: Union Camere-Excelsior 2011)

## CURRICULUM VITAE E LETTERA DI ACCOMPAGNAMENTO

**Il Curriculum Vitae.** Quando si è alla ricerca di un lavoro, prima o poi arriva il momento di inviare (per posta ordinaria o per e-mail) il proprio *Curriculum Vitae* (CV) e una lettera di accompagnamento alle aziende per le quali si desidera lavorare, sperando di essere chiamati per un colloquio.

Il CV è la carta di identità professionale del candidato e deve indicare l'iter formativo, le conoscenze e le competenze di chi si propone per ottenere un impiego.

Si comincia sempre dai dati anagrafici, per un'inquadratura iniziale, e dai contatti (indirizzo, numero di telefono, cellulare, e-mail...), per poi passare in rassegna le precedenti esperienze lavorative e le varie tappe della propria istruzione/formazione, dalla più recente alla più lontana nel tempo.

Altre informazioni indispensabili riguardano la padronanza di una o più lingue straniere e le competenze tecniche; conviene anche mettere in rilievo le capacità relazionali e organizzative, se si posseggono.

Per quanto riguarda altre informazioni personali, è meglio inserire solo quelle che possono essere apprezzate dalla specifica azienda cui è indirizzato il CV.

Infine, non bisogna mai dimenticare di autorizzare il trattamento dei dati personali, facendo riferimento al d. lg. 196/2003.

Un CV efficace sarà completo, chiaro e soprattutto breve (due pagine di solito sono sufficienti): bisogna tenere conto che chi lo legge è abituato a valutarne decine tutti i giorni e apprezzerà il fatto di trovare subito le informazioni che gli interessano.

Meglio selezionare solo le aziende che più si avvicinano al proprio profilo professionale e scrivere per ciascuna una lettera di accompagnamento mirata.

I portali che si occupano di selezione del personale solitamente danno la possibilità di compilare CV online, secondo modelli prestabiliti; oppure si può preparare da soli il CV e poi caricarlo sul sito su cui ci si vuole proporre.

**La lettera di accompagnamento (o *cover letter*)** va preparata con molta attenzione perché serve a convincere il selezionatore a prendere in considerazione l'offerta di lavoro e quindi a esaminare il CV.

La forma deve essere curata e corretta, per dimostrare un buon livello di istruzione.

La lettera di accompagnamento è una e-mail (o una lettera) dalla quale devono emergere in maniera sintetica (dieci righe al massimo) le motivazioni del candidato, le competenze, i titoli, le esperienze che rendono la persona adatta per quel posto di lavoro.

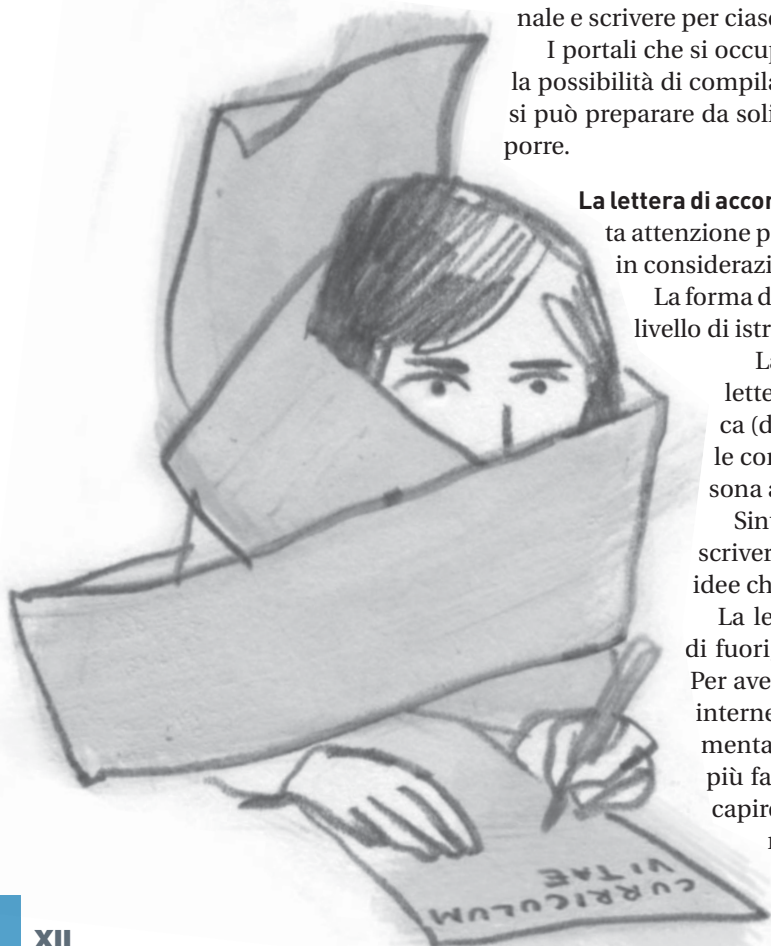
Sintetici sì, ma non vaghi o generici: l'impegno nello scrivere la lettera sta proprio nel risultare sinceri, con le idee chiare ma anche aperti a varie possibilità.

La lettera deve far capire che si conosce, anche se dal di fuori, l'azienda e che se ne comprendono le necessità. Per avere queste informazioni è necessario visitarne il sito internet ma anche, ad esempio, cercare e, se si può, sperimentare i prodotti di quell'azienda. In questo modo sarà più facile mettersi dal punto di vista dell'azienda stessa, capire quali competenze potrebbero essere utili e puntare su quelle.



Scarica il CV  
Europass

[www.europassitalia.it](http://www.europassitalia.it)





## CURRICULUM VITAE E LETTERA DI ACCOMPAGNAMENTO

Le possibilità di essere valutati crescono se la busta che contiene lettera e CV, o l'e-mail, è indirizzata al direttore del settore nel quale vorremmo lavorare e non genericamente all'impresa o, ad esempio, all'ufficio delle risorse umane. In questo caso bisogna fare accurati controlli per essere certi di scrivere correttamente il nome, il titolo di studio, la posizione che ricopre la persona a cui indirizziamo la lettera ed essere sicuri che effettivamente lavori ancora lì.

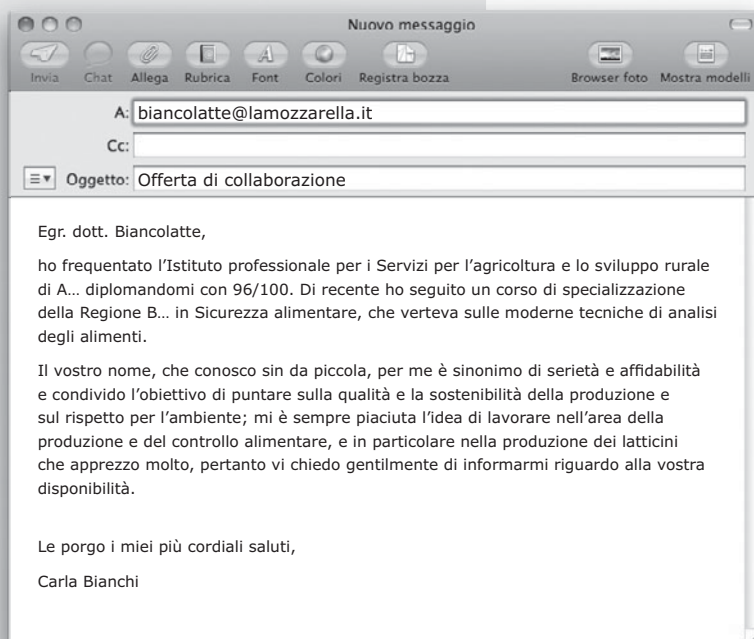
**Una lettera di accompagnamento.** Carla è diplomata in Servizi per l'agricoltura e lo sviluppo rurale. Ha sfruttato un periodo di lavoro *part-time* in un *call center* per avere il tempo di cercare un corso di formazione che faccia al caso suo. Dopo ha frequentato un corso della Regione di 180 ore in Sicurezza alimentare.

Nel frattempo visita i siti di varie aziende della zona in cui abita e ne individua alcune cui decide di inviare il CV.

La ditta dove vorrebbe lavorare è "La Mozzarella", che produce latte e derivati. Nel sito si insiste sulla qualità dei prodotti unita al rispetto dell'ambiente.

A chi vuole lavorare per "La Mozzarella" è richiesta personalità, grinta e condivisione dei valori dell'azienda. Con una telefonata Carla verifica che il responsabile della sicurezza alimentare è il dott. Biancolatte.

Ecco la lettera di accompagnamento scritta da Carla.



## IL COLLOQUIO E LO STAGE

**Il colloquio.** La strategia per la buona riuscita di un colloquio di lavoro comincia nel momento in cui si viene contattati. Innanzitutto è importante **rispondere subito** e con gentilezza alla convocazione (che sia arrivata per telefono, lettera o e-mail) e presentarsi puntuali all'appuntamento.

Per evitare ritardi, conviene informarsi bene su come raggiungere la sede del colloquio e partire con largo anticipo, così da non arrivare trafelati all'incontro.

Il successo di un colloquio dipende anche da una serie di informazioni che sarà stato possibile raccogliere sull'azienda e utilizzare a proprio vantaggio. Ad esempio, per decidere quale sia l'**abbigliamento più adatto**, uno sguardo allo stile dell'azienda è consigliato. Basterà poi adattare questo stile al proprio e alla posizione alla quale si aspira. Se, ad esempio, cerchiamo lavoro in banca potrebbe essere una buona idea non mettere i jeans, se si tratta di un'azienda di grafica che ha uno stile giovane e casual i jeans andranno benissimo. **Conoscere l'azienda per la quale si desidera lavorare** è importante anche per mostrare in maniera mirata le competenze di cui si dispone, nonché interesse e sintonia con quella specifica linea imprenditoriale.

Quando ci si trova di fronte alla persona incaricata della selezione bisogna mostrarsi **sicuri e determinati** senza essere spavaldi o sbruffoni. Non conviene mentire a proposito delle esperienze lavorative precedenti o essere disonesti riguardo alle proprie capacità: prima o poi si verrà scoperti, magari nel momento meno opportuno... È invece importante mostrarsi positivi, **disponibili a imparare e a risolvere problemi**.

I reclutatori rivolgono al candidato una serie di domande, a volte prevedibili, che possono riguardare la sfera personale (ad esempio "Da quanto tempo cerca lavoro?"...) o la sfera professionale: sia sulle esperienze passate (ad esempio: "Mi parli del suo curriculum", "Perché ha scelto proprio quel corso di studi?"...), sia sul lavoro per cui si è a colloquio (ad esempio "Cosa sa della nostra azienda?", o anche "Perché dovremmo assumerla?").

Alcune aziende preparano un **colloquio di gruppo**, per osservare in che modo i candidati interagiscono tra loro, collaborano, affrontano alcune situazioni critiche che simulano quelle reali. In questi casi il consiglio è di non essere eccessivi: la cosa migliore è mostrare senso pratico e capacità di mediare e partecipare o guidare il gruppo verso la soluzione del problema.

**Lo stage (tirocinio formativo o *internship*).** Si tratta di un'esperienza professionale utile per chi si avvicina al mondo del lavoro per la prima volta, per accrescere le proprie competenze e arricchire il *Curriculum Vitae*, anche perché è difficile trovare un impiego senza avere precedenti esperienze.

Lo stage non rientra nelle tipologie di lavoro subordinato poiché è obbligatoria per il tirocinante solo un'assicurazione in caso di infortunio (e non lo stipendio).

Per quantificare l'utilità dello stage è stato creato il sistema dei **crediti formativi**, ossia un punteggio che il giovane studente guadagna nel corso del suo tirocinio e che può spendere ai fini formativi: di **diploma**, per gli studenti del quinto anno di scuola media superiore; di **esame** o di **laurea**, per gli universitari.

Un'esperienza di stage può anche arrivare a sostituire un esame universitario: è sufficiente certificare che l'esperienza svolta durante lo stage va a integrare le conoscenze acquisite nell'arco degli studi, completandole e arricchendole.



E se mi fanno una domanda assurda?

www.  
ideeperiltuofuturo.it

# A

## **Basi di dati relazionali e linguaggio SQL. Linguaggio XML**

**A1**

**Sistemi informativi e sistemi informatici**

**A2**

**Le basi di dati relazionali**

**A3**

**Il linguaggio SQL**

**A4**

**Accesso a una base di dati in linguaggio  
Java con JDBC**

**A5**

**Il linguaggio XML per la rappresentazione  
dei dati**

**A6**

**Gli strumenti per la gestione dei dati  
rappresentati in linguaggio XML**



**A7**

**I sistemi di gestione delle basi di dati  
Microsoft Access e Oracle My-SQL**



# Sistemi informativi e sistemi informatici

L'idea di una scuola pubblica viene generalmente identificata con la sua funzione primaria, ovvero quella di favorire lo sviluppo delle competenze dei suoi studenti, che in ultima analisi rientra nelle attività che mirano a fornire servizi di qualità sempre migliore ai cittadini. Perché questo possa avere luogo un'organizzazione scolastica deve gestire numerosi dati, come quelli necessari alle attività didattiche:

- studenti iscritti;
- docenti assegnati;
- corsi attivati;
- orario annuale delle lezioni;
- spazi disponibili (aule, laboratori, palestre, ...);
- gestione dei risultati intermedi e finali; ...

Ma, affinché tali attività possano essere svolte adeguatamente, una scuola deve gestire anche altri dati a cui non è immediato pensare, per esempio quelli relativi:

- al personale di servizio (bidelli, ...);
- al personale amministrativo che opera nei vari uffici (contabilità, didattica, amministrazione, ...);
- al magazzino dei beni di consumo (carta, registri, toner, ...); ...

Un'adeguata gestione di questi processi impone che l'organizzazione scolastica sia strutturata in modo articolato, in un contesto in cui personale responsabile, organi consultivi e decisionali, uffici che rispondono alla dirigenza (Preside) devono scambiarsi continuamente informazioni di supporto ai processi operativi e di governo della scuola stessa.

La FIGURA 1 rappresenta in forma schematica i processi funzionali di una scuola.

Le attività di raccolta, organizzazione e conservazione dei dati costituiscono uno dei principali settori di applicazione dei sistemi informatici.

Le caratteristiche di un'istituzione scolastica, a cui abbiamo fatto riferimento, così come la quotazione di azioni nei mercati telematici internazionali, la gestione dei conti correnti bancari o delle carte di credito, gli elenchi delle utenze telefoniche, il rilevamento e l'elaborazione di dati censuari, la catalogazione dei libri di una biblioteca e la gestione dei loro prestiti sono solo alcuni esempi di settori dove il trattamento di quantità più o meno grandi di dati risulta inevitabile. Le tecnologie informatiche permettono di memoriz-

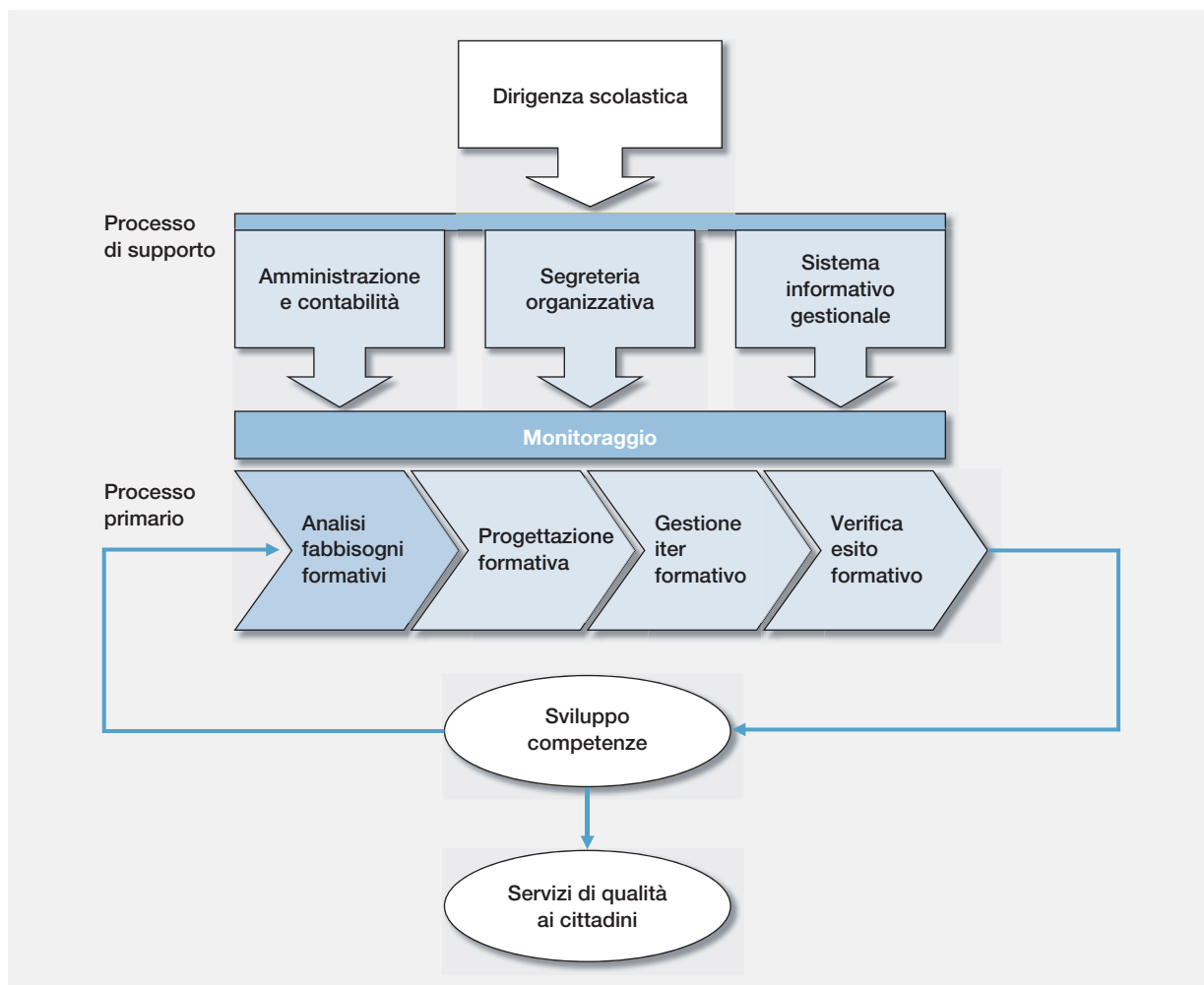


FIGURA 1

zare i dati in modo permanente su specifici dispositivi e di gestirli in modo da riflettere rapidamente le variazioni rispetto alla loro accessibilità da parte di varie categorie di utenti che li consultano direttamente, o in forma elaborata.

## 1 Dati e informazione

Nella pratica quotidiana spesso si usano in modo interscambiabile i termini «dato» e «informazione»: in effetti non è immediato distinguerli. Nel seguito proviamo a formularne una definizione illustrata da alcuni esempi.

► Un **dato** (dal latino *datum*, «fatto») è la misura di un fenomeno che siamo interessati a osservare.

### ESEMPIO

Si può utilizzare un metro per misurare l'altezza di una persona, oppure utilizzare un termometro per misurare la sua temperatura corporea. Supponendo di aver misurato nel primo caso 180 cm e nel secondo caso 39 °C, abbiamo ottenuto due dati, ciascuno relativo a uno specifico fenomeno osservato.

► L'**informazione** è ciò che si ottiene dall'elaborazione di un insieme di dati e che accresce lo stato di conoscenza relativo a un fenomeno.

**ESEMPIO**

Nel caso della temperatura corporea a cui si è fatto riferimento nell'esempio precedente, l'elaborazione del dato permette di dedurre l'informazione che nella salute del soggetto osservato è presente un'alterazione: è noto infatti che una temperatura corporea superiore a 37 °C è indice di uno stato febbrile in corso (nel caso specifico l'elaborazione è limitata al confronto del dato rilevato con un valore di soglia predefinito).

Da un punto di vista puramente intuitivo è possibile affermare che maggiore è la quantità di dati di cui si dispone rispetto a un fenomeno da analizzare, migliore sarà la qualità dell'informazione che, a partire da una corretta elaborazione dei dati, si consegue per approfondire la sua conoscenza.

**ESEMPIO**

Così come un medico incrocia tra loro i sintomi e i risultati delle analisi (dati) per formulare una corretta diagnosi dei problemi di salute del paziente (informazione), allo stesso modo un analista aziendale esamina i dati di un'impresa per determinarne l'andamento economico ed eventualmente individuare i correttivi idonei affinché la sua redditività si mantenga soddisfacente.

Non sempre, osservando lo scenario di un fenomeno, tutti i dati analizzabili risultano utili alla sintesi dell'informazione: esistono, per esempio, situazioni in cui il paziente presenta sintomi che non solo non sono utili alla formulazione di una corretta diagnosi, ma che rischiano di portare il medico a commettere un errore di valutazione.

**OSSERVAZIONE**

Il conseguimento di valide informazioni discende da aspetti sia quantitativi sia qualitativi dei dati: non solo è necessario ottenere più dati possibile relativi al fenomeno oggetto dell'analisi, ma anche scartare quelli che non sono necessari allo scopo dell'analisi. Una corretta elaborazione di tali dati produrrà informazioni utili sia alla comprensione sia all'interazione con il fenomeno studiato.

Esaminando il lavoro di un medico (o di un analista) è possibile osservare che è in grado di raccogliere molti dati relativi a un paziente (o a un'impresa), di scartare con precisione i dati inutili allo scopo dell'analisi, di correlare in modo corretto i dati e di elaborarli per formulare una corretta diagnosi e individuare la cura appropriata, ma... talvolta il malato muore (o l'azienda fallisce): in che cosa ha sbagliato? Probabilmente non ha tenuto conto di un ulteriore elemento critico, il fattore tempo: ha impiegato troppo tempo nel processo di analisi dei dati e di formulazione della diagnosi.

**OSSERVAZIONE** Frequentemente le informazioni reali hanno una validità limitata nel tempo; è sufficiente pensare all'andamento dei titoli in borsa: nel giro di pochi minuti la loro valutazione può infatti cambiare anche notevolmente.

## 2 Sistemi informativi e sistemi informatici

Raccogliere, archiviare ed elaborare dati per gestire e comunicare informazioni è una necessità che da sempre ha caratterizzato le attività umane; a questo scopo gli strumenti che sono stati utilizzati nel tempo vanno dalle incisioni su pietra ai moderni elaboratori elettronici.

Il concetto di sistema informativo è indipendente dagli strumenti utilizzati per la gestione delle informazioni che esso gestisce.

Un **sistema informativo** viene utilizzato da un'organizzazione pubblica o privata per il conseguimento di specifici obiettivi; si può affermare che gli scopi per i quali è necessario disporre di adeguate informazione sono sostanzialmente due: operativo e decisionale.

- **Scopo operativo.** Tutte le organizzazioni hanno la necessità di gestire dati relativi funzionali alle loro attività operative (infrastrutture, dipendenti, materiali, strumenti, ...) e da cui derivano informazioni che pertanto possiamo definire «di servizio».

### ESEMPI

Un'amministrazione comunale gestisce l'anagrafe dei cittadini per fornire servizi come l'emissione di certificati; un'azienda di produzione deve provvedere alla fatturazione delle merci e alle paghe dei propri dipendenti; uno studio commerciale deve compilare le dichiarazioni dei redditi dei propri clienti, ....

- **Scopo decisionale.** Per poter prendere decisioni relative alle attività di programmazione, controllo e valutazione, un'organizzazione deve basarsi su informazioni comunemente denominate «di governo».

### ESEMPI

Un'amministrazione comunale decide di localizzare un asilo nido o un centro di assistenza per gli anziani in funzione delle fasce di età e della zona di residenza dei propri cittadini; un'azienda commerciale decide di rivedere il prezzo di un prodotto in base al costo di produzione, all'andamento delle vendite e alla situazione del mercato su cui opera, .... In tutti questi casi una corretta informazione costituisce una risorsa essenziale per il conseguimento degli obiettivi dell'organizzazione.

**OSSERVAZIONE** Esistono informazioni che sono al tempo stesso sia di servizio sia di governo. I costi e i ricavi di un'azienda sono informazioni di servizio per la contabilità fiscale e di governo per gli organi interni preposti all'assunzione di decisioni sulle future strategie aziendali.

I servizi di trattamento delle informazioni necessarie a un'organizzazione vengono gestiti mediante un insieme di procedure e di risorse sia umane sia materiali: questo insieme è denominato «sistema informativo».

## La «crisi del software»

Circa un quarto di tutti i grandi progetti software viene interrotto prima della conclusione e più della metà viene terminato in notevole ritardo o con costi molto maggiori di quelli inizialmente previsti.

Queste statistiche – unite al fatto che frequentemente il software realizzato non soddisfa i requisiti del committente – non sono accettabili per la moderna industria del software: la situazione è nota almeno dal 1968, anno in cui il Comitato scientifico della NATO organizzò un convegno a Garmisch in Germania sulla «crisi del software», ed è solo con l'avvento del nuovo millennio che l'adozione sistematica delle metodologie dell'ingegneria del software ha introdotto miglioramenti significativi nella capacità di progettare e realizzare software di qualità.

Uno dei più noti esempi di progetti software dall'esito disastroso è dato dallo sviluppo del sistema di controllo dello smistamento dei bagagli dell'aeroporto di Denver, in Colorado ►

► Un **sistema informativo** è un insieme strutturato di procedure e di risorse umane e materiali finalizzati alla raccolta, archiviazione, elaborazione e comunicazione di dati, allo scopo di ottenere le informazioni necessarie a un'organizzazione per gestire sia le attività operative sia quelle di governo.

**OSSERVAZIONE** Le risorse materiali di un sistema informativo non necessariamente sono costituite da strumentazione hardware, così come le procedure non sempre sono sistemi software. I sistemi informativi esistevano già molto tempo prima dell'avvento degli elaboratori elettronici: gli archivi delle banche o dei servizi anagrafici esistono da secoli e per molto tempo si sono basati solo su schedari e registri cartacei ad accesso manuale.

Gli indubbi vantaggi derivanti dall'utilizzo delle tecnologie informatiche per la gestione delle informazioni – parallelamente alla loro evoluzione e diffusione favorita dal progressivo abbassamento dei costi – ne hanno reso sempre più conveniente l'impiego in questo contesto, come in molti altri.

► Un **sistema informatico** è il sottoinsieme di un sistema informativo dedicato al trattamento «automatico» di informazioni derivanti dalla gestione di dati archiviati in formato digitale.

**OSSERVAZIONE** La presenza di tecnologie informatiche non significa necessariamente una completa automatizzazione del sistema informativo: esistono aspetti di un sistema informativo che può non valere la pena trattare informaticamente, per questioni sia pratiche sia economiche (come, per esempio, le comunicazioni verbali tra impiegati di uno stesso ufficio).

## 3 Ciclo di vita di un sistema informatico

La progettazione di un sistema informatico generalmente passa attraverso un processo piuttosto complesso che deve essere condotto da personale professionalmente qualificato. Le conseguenze di un progetto gestito male possono portare a inefficienza, perdita di dati, alti costi di manutenzione, eventuali costi di riprogettazione, blocco delle attività operative.

**OSSERVAZIONE** Un errore tipico nella progettazione di un sistema informatico consiste nel concepirlo come una banale automazione delle procedure eseguite manualmente. L'informatizzazione di un sistema informativo deve essere un'occasione per razionalizzarne le attività, garantendone comunque la coerenza, al fine di eliminare inefficienze preesistenti e migliorarne l'efficienza e l'efficacia.

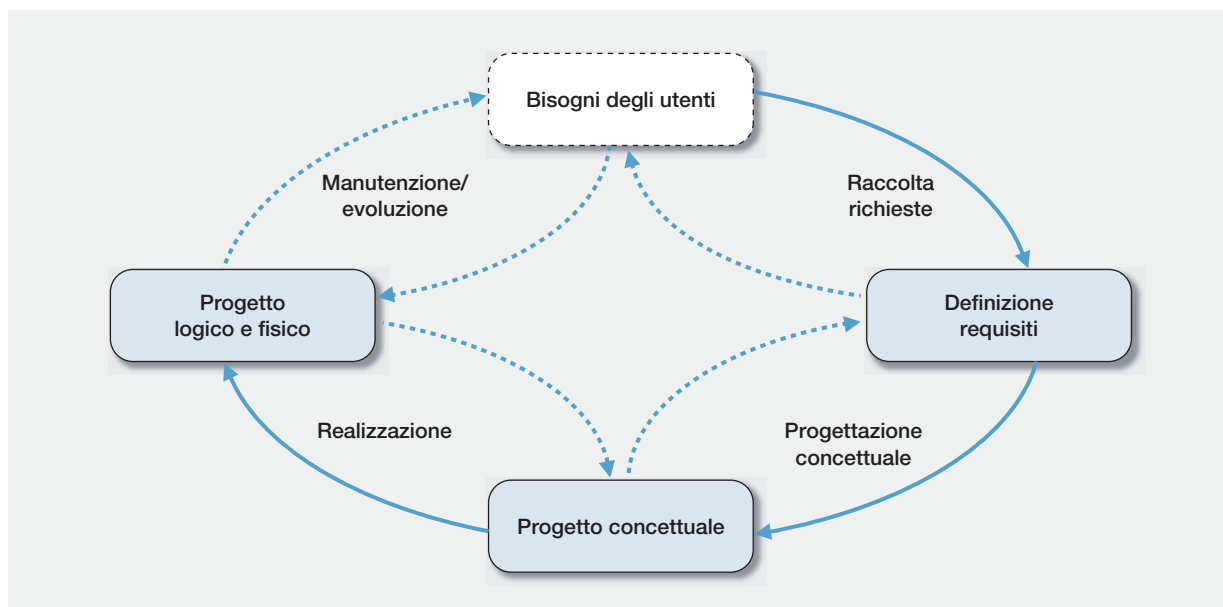


FIGURA 2

Nella pratica non esiste una metodologia di progettazione standard, ma esistono diverse tecniche, con livelli di formalizzazione differenziati, impiegate dalle aziende produttrici di software e dai professionisti del settore.

Tutti però concordano nel sostenere che la progettazione di un sistema informatico sia un processo ciclico permanente durante tutto il tempo di vita del sistema stesso. Tale processo si articola su un insieme di attività raggruppabili in almeno tre macrofasi concettualmente distinte:

- **raccolta delle richieste degli utenti;**
- **progettazione concettuale;**
- **realizzazione (o progettazione logica e fisica).**

Nella FIGURA 2 è rappresentato graficamente il «ciclo di vita» di un sistema informatico (gli archi tratteggiati indicano attività di verifica – *feedback* – che devono essere eseguite prima di passare alla fase successiva).

Le tre fasi indicate producono rispettivamente come output:

- la definizione dei requisiti a cui il sistema informatico dovrà essere conforme;
- il progetto concettuale;
- il progetto logico/fisico che rappresenta l'insieme delle componenti software che implementano il sistema informatico.

**OSSERVAZIONE** Il processo di progettazione è ciclico perché con l'uso del sistema gli utenti e i committenti avanzano richieste correttive o evolutive del software. Inoltre si possono creare nuove esigenze per la dinamica propria di ogni organizzazione: la realtà operativa evolve e il sistema deve adeguarsi di conseguenza per supportarla.

► (USA), uno dei più grandi del mondo: la complessità di realizzazione del software distribuito su centinaia di computer che doveva gestire lo spostamento di migliaia di convogli guidati automaticamente su una rete di binari sotterranei estesa decine di chilometri ha ritardato di quasi due anni – dal 1993 al 1995 – l'inaugurazione dell'aeroporto, comportando perdite economiche che hanno superato il milione di dollari al giorno. La conclusione della commessa ha in ogni caso richiesto una riprogettazione sostanziale del sistema di smistamento dei bagagli in cui è stato abbandonato il controllo completamente automatico in favore di una più tradizionale movimentazione gestita da operatori. Tutte le analisi condotte su questo disastro dell'industria del software concordano sul fatto che i responsabili dell'aeroporto e della ditta fornitrice del sistema ne hanno – in mancanza di una rigorosa disciplina progettuale – sottovalutato l'enorme complessità realizzativa.

Le metodologie di progettazione più diffuse hanno come riferimento la struttura dei dati da archiviare e la loro modellazione: in pratica, come meglio vedremo nel seguito, si costruisce un modello della realtà indipendente dalle procedure, in modo da garantirne la futura introduzione di nuove senza dover rivedere il lavoro già svolto sui dati.

### 3.1 Raccolta delle richieste degli utenti

Nel corso della prima fase è necessario raccogliere tutti quegli elementi che servono a definire le caratteristiche che il sistema informatico dovrà avere per supportare adeguatamente i bisogni degli utenti: il tempo e il costo di tutto il progetto dipende dalla qualità del lavoro svolto in questa fase. Essa, in generale, prevede almeno le seguenti attività:

■ **Indagine preliminare relativa all'introduzione del sistema informatico nell'organizzazione.** Tale attività dovrebbe essere svolta a cura del personale dell'organizzazione stessa al fine di:

- individuare i settori dell'organizzazione potenzialmente interessati all'introduzione delle tecnologie informatiche e, per ognuno di essi, valutarne l'effettiva convenienza di informatizzazione, in relazione ai piani di sviluppo aziendali («obiettivi perseguibili»);
- valutare gli impatti dovuti all'introduzione delle tecnologie informatiche sull'organizzazione del lavoro (riassetto e riqualificazione del personale, ecc.) e sul sistema informativo esistente («vincoli organizzativi»);
- valutare le risorse disponibili (budget economico) per l'introduzione delle tecnologie informatiche.

■ **Analisi del sistema informativo esistente.** Questa attività viene normalmente svolta da una specifica figura professionale denominata **analista**: egli ha lo scopo di raccogliere, catalogare e sistematizzare le conoscenze relative al sistema informativo esistente nei settori da informatizzare. Gli analisti interagiscono con gli utenti del sistema informativo a vari livelli, dai dirigenti al personale operativo, per acquisire un'adeguata conoscenza dei loro bisogni: informazioni necessarie, procedure in essere, tempistica delle elaborazioni, privacy delle informazioni, possibili esigenze future, ecc. Come risultato di questa attività si ottengono indicazioni relative alle possibilità e ai limiti del futuro sistema informatico, in modo da catalogare correttamente i dati e le procedure da prendere in considerazione. Si tratta di un'attività delicata, perché condiziona il resto della progettazione. Caratteristica essenziale di un analista è quella di avere ottime capacità di osservazione e comprensione della realtà: competenze, queste, che sono raggiungibili solo con l'esperienza.

■ **Definizione dei requisiti del nuovo sistema.** In base a osservazioni e valutazioni, l'analista produce una documentazione dettagliata descrivendo elementi quali:

- la classificazione dei dati utilizzati (nome, tipo, uso, ecc.);



- i vincoli di integrità dei dati, cioè le condizioni che essi devono rispettare per essere significativi e validi;

#### ESEMPIO

Un vincolo semplice può essere quello relativo ai componenti di una data (giorno, mese, anno) per essere considerata valida; una situazione più complessa può essere quella di una compagnia aerea che non accetta prenotazioni di passeggeri minorenni se non accompagnati da almeno un passeggero maggiorenne.

- descrizione delle procedure da automatizzare che metta in evidenza per ciascuna di esse i dati coinvolti, la relazione tra dati di ingresso e dati in uscita, la modalità d'interazione con l'utente, i vincoli sui tempi di risposta del sistema, la frequenza d'uso, ecc.;
- volume iniziale e previsione di crescita dei dati nel tempo;
- grado di privacy dei dati differenziato a seconda degli utenti e del tipo di utilizzazione.

**OSSERVAZIONE** Rispetto alla privacy dei dati devono essere valutati due aspetti: uno sotto il profilo della praticità e uno sotto quello della sicurezza. Circa l'aspetto pratico è opportuno che in un'organizzazione complessa i vari utenti del sistema informatico possano accedere solo ai dati di propria competenza per evitare che si possano creare situazioni confuse da «sovraccarico informativo» (*information overloading*). Per quanto riguarda l'aspetto relativo alla sicurezza, la regola generale è «non tutti devono conoscere tutto»: normalmente, per esempio, solo pochi utenti necessitano di conoscere dettagli relativi alle future strategie aziendali, o progetti realizzativi di nuovi prodotti, per evitare potenziali situazioni di spionaggio industriale volte a favorire la concorrenza.

## 3.2 Progettazione concettuale

La documentazione prodotta nella fase di raccolta delle richieste costituirà l'input della successiva fase di progettazione concettuale. Questa consiste nell'utilizzare gli elementi raccolti per la definizione di un modello astratto del sistema informatico (progetto concettuale), destinato a definire le linee guida di riferimento e di lavoro per i progettisti della successiva fase di realizzazione. Molto spesso il progetto concettuale costituisce anche il tramite di verifica fra il committente e i progettisti.

In relazione a quest'ultimo aspetto le vignette di FIGURA 3 illustrano in tono umoristico i rischi insiti nella realizzazione di un sistema informatico.

I termini chiave «astratto» e «concettuale» usati in precedenza sono dovuti al fatto che in questa fase non si affrontano i futuri dettagli realizzativi, in particolare la progettazione si astrae dalle tecnologie hardware e software che saranno impiegate per l'effettiva realizzazione del sistema informatico; si tende invece a definire una opportuna organizzazione dei dati.

### Linguaggi di modellizzazione

La progettazione concettuale può essere effettuata ricorrendo a un linguaggio di modellizzazione che consenta la descrizione della realtà analizzata e al tempo stesso la definizione del sistema informatico da realizzare per implementarla.

I diagrammi E/R (*Entity-Relationship*) sono utilizzati da molto tempo come formalismo grafico per la documentazione di strutture di dati organizzate secondo il modello «relazionale», affermatosi nel corso degli anni '70 del secolo scorso e tuttora dominante.

Nello stesso periodo sono emersi in ambito militare i linguaggi grafici della famiglia IDEF (*Integration Definition*) di cui il più noto – IDEF1X – è un formalismo di documentazione dei dati.

Recentemente si è imposto come strumento utilizzato da moltissimi analisti UML (*Unified Modeling Language*): si tratta di un insieme di diagrammi standard e di formalismi grafici per descrivere vari aspetti di un sistema informatico.

Da UML deriva Sys-ML (*System Modeling Language*), un linguaggio di modellizzazione specifico per la descrizione dei sistemi complessi.

Infine BPMN (*Business Process Model and Notation*) è un linguaggio grafico per la descrizione e definizione dei processi aziendali.



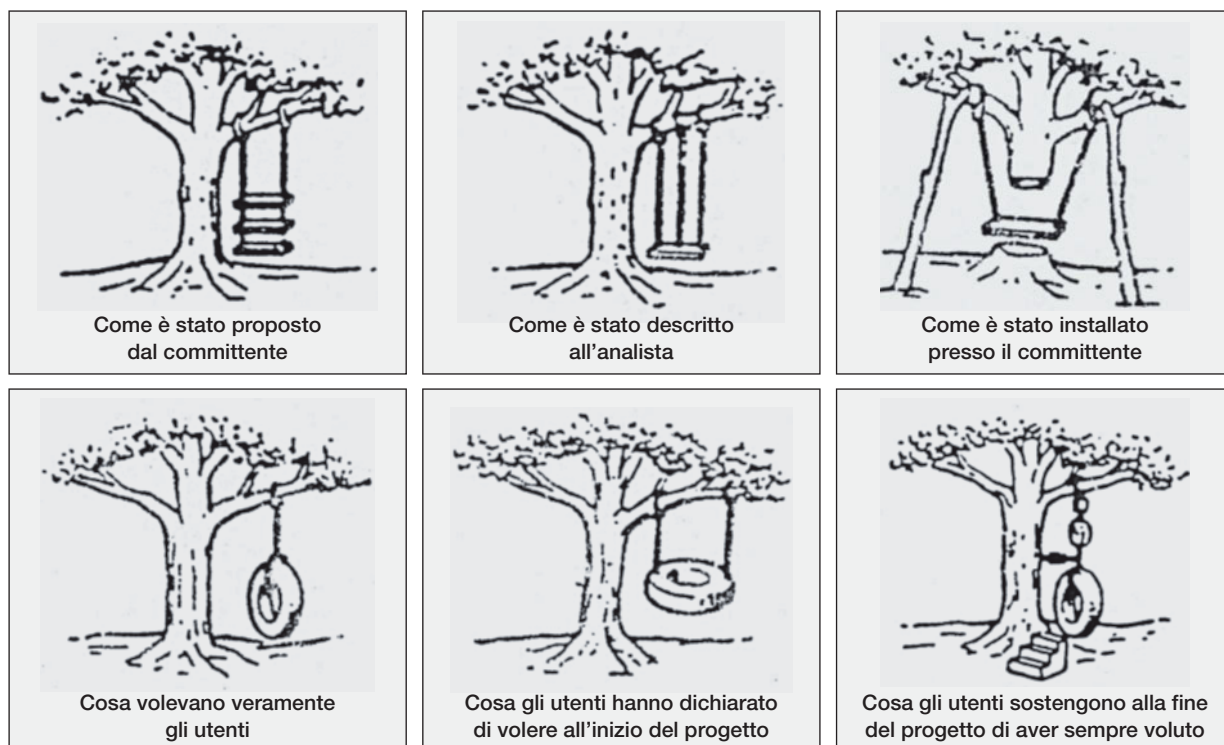


FIGURA 3

A parte questa idea generale, c'è da dire che nella realtà delle cose non esiste un accordo preciso su cosa debba essere un progetto concettuale e sugli strumenti necessari per definirlo. La maggior parte degli attuali approcci considera il progetto concettuale come un insieme di documenti, schemi e diagrammi che descrivono in modo organico e coerente:

- la struttura dei dati in termini di insiemi e relazioni fra insiemi;
- i vincoli di ammissibilità dei dati;
- le procedure di elaborazione dei dati;
- i vincoli sui tempi di risposta;
- l'integrità e la riservatezza delle informazioni;
- i costi in termini di risorse informatiche richieste per la realizzazione del progetto (per esempio, la configurazione hardware, le componenti software, ecc.).

### 3.3 Realizzazione (progettazione logica e fisica)

La progettazione logica e fisica consiste nella realizzazione effettiva del sistema informatico nelle varie componenti:

- **infrastrutturali**: individuazione e acquisizione delle piattaforme hardware, software e di comunicazione;
- **applicative**: acquisizione di prodotti software già disponibili sul mercato, eventualmente da personalizzare e integrare, e sviluppo di software specifico.

Questo testo si occupa prevalentemente dello studio delle caratteristiche delle componenti software – sia infrastrutturali sia applicative – relative alla memorizzazione e alla elaborazione dei dati.

La centralità dei dati ha sempre caratterizzato le applicazioni informatiche per l'automazione dei sistemi informativi, ma solo a partire dalla fine degli anni '60 del secolo scorso sono stati sviluppati ambienti software specificamente dedicati alla loro gestione. In assenza di tali ambienti, ancora oggi alcuni sistemi informatici vengono realizzati affidandosi a linguaggi di programmazione tradizionali o orientati agli oggetti (esistono ancora numerose applicazioni scritte in COBOL, un linguaggio di programmazione specifico per applicazioni aziendali molto utilizzato negli anni '60 e '70 del secolo scorso). Questo tipo di approccio viene generalmente indicato come approccio basato su *file system*, in considerazione del fatto che l'archiviazione dei dati avviene mediante file memorizzati nella memoria di massa di un elaboratore gestiti mediante le funzionalità rese disponibili dal sistema operativo.

**OSSERVAZIONE** Un file consente la memorizzazione e la ricerca di dati, ma fornisce solo semplici meccanismi di accesso e condivisione. Seguendo questo approccio le procedure implementate mediante un linguaggio di programmazione sono autonome: ognuna di esse, infatti, definisce e utilizza uno o più file privati, ed eventuali dati di interesse per procedure distinte sono spesso replicati pur comportando una inaccettabile ridondanza che può causare situazioni di incoerenza.

#### ESEMPIO

In un'azienda l'archivio dei prodotti è gestito dall'ufficio fatturazione, che ha il compito di fatturare le merci vendute, dal magazzino, che deve controllare il carico e lo scarico delle merci, e dall'ufficio vendite, che intende utilizzare i dati per proporre azioni di vendita promozionale. Se questi diversi soggetti gestissero separatamente i dati dei prodotti si avrebbero numerose duplicazioni dei dati e, a lungo andare, si potrebbero avere dati duplicati non aggiornati in modo coerente.

Fin dagli anni '80 del secolo scorso l'approccio basato su *file system* è stato gradualmente sostituito dall'approccio basato sui **Sistemi di gestione delle basi di dati** (DBMS, *Data Base Management System*), sistemi software in grado di gestire grandi collezioni di dati integrate, condivise e persistenti.

Il confronto fra l'approccio basato su *file system* e DBMS sarà analizzato nel seguito; anticipiamo che tali sistemi eliminano la duplicazione dei dati e che in termini di facilità di programmazione, flessibilità e affidabilità presentano indubbi vantaggi: la presenza di un DBMS introduce un livello di astrazione più elevato di quello del *file system* e fornisce automaticamente una serie di servizi che l'altro approccio non prevede.

Inizialmente la larga diffusione di cui oggi godono i DBMS era stata ostacolata dal costo relativamente alto e dalle risorse hardware richieste per il

funzionamento, ma la crescente disponibilità di software e hardware potenti ed economici ha reso conveniente il loro utilizzo anche nella realizzazione di piccoli sistemi informatici.

## 4 Aspetti intensionale ed estensionale dei dati

Lo scopo dei dati in un sistema informativo è quello di rappresentare i fatti che caratterizzano un aspetto della realtà dell'organizzazione; i dati sono normalmente registrati su specifici supporti (carta, dischi magnetici, dischi ottici, ecc.) per consentirne la conservazione e la trasmissione. Come abbiamo già notato, un'informazione è l'incremento di conoscenza acquisita o dedotta dai dati mediante la loro elaborazione: da ciò deriva il fatto che i dati sono utili solo quando si dispone di una chiave interpretativa che consenta di comprenderne il significato («semantica»), cioè i fatti che essi rappresentano. Per esempio, i dati dell'insieme schematizzato nella FIGURA 4

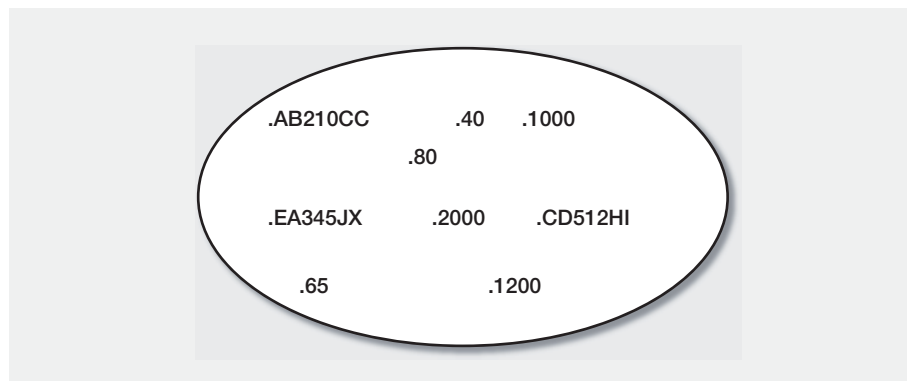


FIGURA 4

non sono significativi fino a quando non venga specificato che cosa effettivamente rappresentano e le eventuali relazioni che li legano. In altre parole, il contenuto informativo dei dati, cioè il loro significato, non può nascere solo dai valori specifici dei dati (**aspetto estensionale dei dati**) ma anche e necessariamente dalla loro interpretazione (**aspetto intensionale dei dati**). Questo è evidente se rappresentiamo l'insieme visto in precedenza nella seguente forma:

Targa auto	Cilindrata	kW
CD512HI	1000	40
EA345JX	2000	80
AB210CC	1200	65

Questa deve essere interpretata come una delle possibili **istanze** (o estensioni) del seguente **schema** (o intensione):

Targa auto	Cilindrata	kW
------------	------------	----

La distinzione fra **intensione** ed **estensione** è molto importante nella teoria della gestione degli insiemi di dati e a essa si farà spesso riferimento nel seguito.

**OSSERVAZIONE** Nei linguaggi di programmazione il significato intensionale di un dato è il tipo del dato. Non a caso sono considerati «insicuri» i linguaggi in cui è possibile modificare dinamicamente il tipo di un dato attribuendo interpretazioni diverse allo stesso dato. Il caso più significativo è quello del linguaggio macchina in cui è possibile interpretare il contenuto di una locazione di memoria come dato numerico, come un singolo carattere di una sequenza, come istruzione eseguibile, come riferimento a un dato diverso, ecc.

Nel linguaggio naturale il significato intensionale dei dati che possono comparire in una frase viene generalmente descritto nella frase stessa.

**ESEMPIO**

Nella frase «la fattura numero 254 ha un'importo di 125,00 euro»:

- «numero di fattura» «254»;
- «importo in euro» rappresenta il significato intensionale del dato «125,00»;
- il verbo «ha» stabilisce inoltre un'associazione fra i due dati «254» e «125».

Quando si è in presenza di gruppi consistenti di dati aventi una stessa interpretazione, è conveniente fornire l'interpretazione a livello di gruppo, piuttosto che per ogni singolo dato.

**ESEMPIO**

Dovendo elencare gli importi di diverse fatture viene immediato organizzare una tabella di coppie del tipo  $(X, Y)$  il cui significato intensionale è «la fattura numero  $X$  ha un importo di  $Y$  euro». Essendo tale significato comune a tutte le coppie, esso può essere specificato una sola volta.

**OSSERVAZIONE** Questa distinzione fra i dati e la loro interpretazione si ritrova in molti ambiti della vita quotidiana: gli orari degli autobus e dei treni o la classifica di una gara sportiva, così come tutte le rappresentazioni tabellari in cui viene fornita insieme ai dati stessi anche la descrizione che consente di interpretarli.

Un insieme di molti dati aventi la stessa interpretazione è uno degli elementi più ricorrenti in ambito informatico: definendo «categoria» una collezione di dati aventi la stessa interpretazione, possiamo affermare che i sistemi informatici sono caratterizzati dall'avere un numero di categorie relativamente basso e costante rispetto ai dati contenuti in ciascuna categoria. In altri termini, la dimensione delle «tabelle» è prevalente rispetto al numero delle tabelle stesse. Caratteristica, questa, che consente di progettare soluzioni semplici ed efficienti. La situazione complementare in cui

si abbia un significativo numero variabile di categorie, ognuna delle quali ha pochi dati, ricade nell'ambito dei linguaggi naturali, dove per ogni dato viene fornita una specifica interpretazione.

**OSSERVAZIONE** La realizzazione di sistemi informatici che gestiscano situazioni con un numero elevato di categorie richiede tecniche e capacità elaborative tipiche dell'area dell'intelligenza artificiale (interpretazione e traduzione dei linguaggi naturali, sistemi esperti, ecc.).

## 5 File di dati

► Un **archivio di dati**, o **file**, è un insieme di dati correlati identificato da un nome, memorizzato permanentemente su un supporto di memoria di massa di un elaboratore e avente vita indipendente dal/dai programma/i utilizzato/i per la sua creazione e/o modifica.

**OSSERVAZIONE** Il fatto che un file abbia vita indipendente dal programma utilizzato per la sua creazione e/o modifica significa che un file creato con un programma può essere successivamente elaborato sia mediante lo stesso programma, sia utilizzandone altri.

Il modulo software di base che consente la gestione dei file è denominato *file system*: è un componente fondamentale del sistema operativo e permette di utilizzare gli archivi memorizzati sulle memorie di massa dell'elaboratore riferendoli mediante nomi simbolici.

Il *file system* del sistema operativo svolge le seguenti funzioni di base:

- mantiene traccia dei file, del loro stato e della loro posizione sul supporto di memorizzazione utilizzando tabelle denominate *directory*;
- controlla, in base alle richieste effettuate dai programmi in esecuzione, le protezioni e i diritti di accesso (lettura, scrittura, ecc.);
- rende o meno disponibili i dati contenuti nei singoli file ai programmi che ne fanno richiesta.

In un file i dati sono normalmente raggruppati in unità logiche denominate **registrazioni** o **record**.

### ESEMPIO

Nell'elenco telefonico i dati dei vari abbonati sono costituiti da un insieme di elementi (cognome, nome, numero telefonico, indirizzo, ...) che costituisce la struttura del record (aspetto intensionale del dato). Ogni singolo elemento all'interno di questa struttura è noto come **campo** o **field**. Prendendo in considerazione uno specifico abbonato, l'insieme dei dati che lo rappresentano in accordo con la struttura predefinita («Bianchi», «Giovanni», «0586584421», «Via Roma 1», ...) costituiscono uno specifico record. L'insieme di tutti i record di un file costituisce l'aspetto estensionale dei dati.

## 5.1 Operazioni sui file

Le operazioni di base che possono essere eseguite su un file sono le seguenti.

- **Creazione.** Inizializza lo spazio destinato a contenere i dati del file sulla memoria di massa dell'elaboratore.
- **Apertura.** Permette l'accesso a un file esistente per poterne elaborare il contenuto; con questa operazione viene assegnato un nome simbolico di riferimento al file fisico presente sulla memoria di massa e stabilita un'associazione tra quest'ultimo e la memoria centrale dell'elaboratore dove i record devono transitare (in lettura e/o scrittura) per poter essere elaborati.
- **Chiusura.** Indica al *file system* del sistema operativo che l'uso del file è terminato e che eventuali dati ancora presenti nella memoria centrale dell'elaboratore devono essere scritti sulla memoria di massa.
- **Inserimento.** Aggiunge nuovi record al file.
- **Modifica.** Aggiorna i dati presenti in uno o più record del file.
- **Cancellazione.** Elimina uno o più record del file.
- **Ricerca.** Individua all'interno del file il record ricercato per effettuarne l'elaborazione; si tratta di un'operazione fondamentale: dal tempo di esecuzione che la caratterizza dipende la velocità delle elaborazioni dei dati del file.

**OSSERVAZIONE** L'operazione di ricerca è necessaria anche in caso di modifica o cancellazione dei record del file: per eseguire tali operazioni è infatti necessario prima ricercare i record a cui applicarle. Spesso l'operazione di ricerca è necessaria anche in caso di inserimento se, per esempio, prima di inserire un nuovo record si intende verificare se esso non sia già presente nel file.

## 5.2 Organizzazione dei file

I file sono strutture informative che realizzano sistemi organizzati per la memorizzazione e l'elaborazione dei dati: queste strutture sono caratterizzate da una organizzazione interna. Per organizzazione o implementazione di un file si intende sia il modo in cui esso è memorizzato sul supporto fisico della memoria di massa, sia il modo in cui viene elaborato.

L'organizzazione interna di un file è distinta in due livelli:

- **fisico**, relativo al supporto fisico di memorizzazione dei dati del file;
- **logico**, relativo alle modalità di gestione dei dati del file.

Il supporto su cui sono memorizzati i dati individua l'organizzazione fisica e condiziona l'organizzazione logica: i programmi applicativi interagiscono sempre con il livello logico, ma per il programmatore è importante sapere che questo si basa su un livello fisico con caratteristiche determinate.

L'organizzazione logica fa riferimento alla gestione del file, cioè al modo in cui i record sono disposti all'interno del file stesso e, di conseguenza, al modo in cui possono essere ricercati e individuati («modalità di accesso»).

I tipi fondamentali di organizzazione logica dei file sono tre: sequenziale, ad accesso diretto e indicizzata.

■ **Sequenziale.** I singoli record sono registrati uno successivamente all'altro, nell'ordine in cui sono stati inseriti; per accedere a uno specifico record è necessario «scorrere» tutti quelli che lo precedono, per cui l'accesso è strettamente sequenziale e generalmente unidirezionale, a partire dal primo record fino all'ultimo. Questo tipo di organizzazione è adatto solo quando si effettua lo stesso tipo di elaborazione per tutti i record del file, per cui file con questa modalità di accesso sono generalmente utilizzati esclusivamente come file storici di conservazione dei dati.

**OSSERVAZIONE** Con file ad accesso sequenziale l'operazione di ricerca può essere facilitata se i record risultano essere ordinati in funzione del campo in base al quale deve essere effettuata la ricerca: la ricerca sequenziale può infatti essere interrotta se, per esempio in caso di ordinamento crescente, viene elaborato un record in cui il valore del campo in oggetto è maggiore di quello ricercato. Altrimenti l'unica possibilità è la ricerca completa nella quale l'elaborazione termina solo se si trova il record avente come valore del campo il valore ricercato, oppure al raggiungimento della fine del file.

■ **Accesso diretto (o casuale, *random*).** Ogni singolo record è individuato da un numero che ne rappresenta la posizione all'interno del file: quando i record sono inseriti sequenzialmente, il numero è assegnato incrementando il numero dell'ultimo record memorizzato. Per accedere direttamente a uno specifico record per effettuare un'operazione di lettura o scrittura dei dati è necessario specificare il numero che ne indica la posizione (l'accesso sequenziale può essere realizzato anche per un sottoinsieme dei record del file a partire da una posizione alla quale si accede direttamente simulando una sequenza di accessi diretti che differiscono tra loro di una singola posizione; in questo caso il file può essere scorso sia in «avanti» che «indietro» usando rispettivamente un incremento positivo o negativo).

Nella FIGURA 5 è schematizzato un archivio random dove i singoli record occupano le posizioni che vanno da 1 a  $n$ .

**OSSERVAZIONE** L'organizzazione ad accesso diretto, pur rappresentando un miglioramento notevole rispetto a quella ad accesso sequenziale, è ancora carente per quanto riguarda la ricerca: è infatti difficile, se si trascurano i casi banali, che esista una correlazione diretta tra il numero di posizione del record e il suo contenuto informativo. Per quanto riguarda l'operazione di ricerca valgono quindi le considerazioni relative all'organizzazione sequenziale. In questo caso, però, è possibile – ordinando



Numero record	Nominativo	Telefono	Indirizzo	...
1	Rossi Mario	0586578182	Via del mare 12	
2	Bianchi Giovanni	0634162567	Via Roma, 14	
3	Verdi Carla	0265134257	Piazza Garibaldi, 21	
4	Rossini Giuseppe	0554567123	Viale Marconi, 1	
5	Neri Marta	0567821453	Corso Mazzini, 56	
6	Bianchi Andrea	0587223311	Viale Italia, 144	
7	Neri Daniele	0586441234	Via Leopardi, 5	
...	...	...		
n	Rossi Maria	0552432185	Scali Manzoni, 68	

FIGURA 5

i record del file rispetto al valore del campo su cui si applica la ricerca – utilizzare la tecnica della ricerca binaria, pagando il costo di un’inefficiente operazione di riordinamento dei record per ogni nuovo inserimento e perdendo l’ordine cronologico di inserimento dei record. Questo tipo di organizzazione – analoga a un *array* – permette di realizzare a livello del software applicativo modalità di accesso come l’indirizzamento *hash* e le liste concatenate di record.

■ **Indicizzata (o *indexed*)**. Nella struttura del record del file viene individuato un campo (o un insieme di campi) denominato «chiave», i cui valori identificano univocamente i singoli record. In questo tipo di organizzazione, oltre a un file primario ad accesso diretto in cui sono memorizzati i record in ordine di inserimento, viene gestito un ulteriore «file indice» contenente una tabella delle chiavi: questa viene utilizzata dal codice che effettua l’accesso al file che la gestisce normalmente come un albero binario di ricerca mantenendola costantemente ordinata secondo un criterio lessicografico. La ricerca di uno specifico record avviene, a partire da un valore fornito in input, consultando la tabella delle chiavi sulla quale per la sua organizzazione la ricerca risulta essere molto rapida e utilizzando la posizione del record per accedere al file primario dei dati. In questo modo è possibile accedere «direttamente» a un record specificandone la chiave e, a partire da questa, elaborare sequenzialmente i record precedenti e successivi seguendo l’ordinamento dei valori chiave.

**OSSERVAZIONE** L’organizzazione *indexed* offre la possibilità di definire più indici in funzione di chiavi diverse (relativamente a un elenco telefonico potrebbe essere necessario definire un indice per il numero di telefono e uno, distinto, sul cognome e nome dell’abbonato) mantenendo unico il file primario dei dati. In questo modo è possibile «vedere» il file ordinato in modi diversi a seconda delle necessità di elaborazione: questa caratteristica determina quella che generalmente viene chiamata multidimensionalità di un archivio.



	Nominativo	Telefono	Indirizzo
1	Rossi Mario	0586578182	Via del mare 12
2	Bianchi Giovanni	0634162567	Via Roma, 14
3	Verdi Carla	0265134257	Piazza Garibaldi, 21
4	Rossini Giuseppe	0554567123	Viale Marconi, 1
5	Neri Marta	0567821453	Corso Mazzini, 56
6	Bianchi Andrea	0587223311	Viale Italia, 144
7	Neri Daniele	0586441234	Via Leopardi, 5
...	...	...	...
n	Rossi Maria	0552432185	Scali Manzoni, 68

Indice nominativo	
Bianchi Andrea	6
Bianchi Giovanni	2
Neri Daniele	7
Neri Marta	5
Rossi Maria	n
Rossi Mario	1
Rossini Giuseppe	4
...	
Verdi Carla	3

Indice telefono	
0265134257	3
0552432185	n
0554567123	4
0567821453	5
0586441234	7
0586578182	1
0587223311	6
...	
0634162567	2

FIGURA 6

Nella FIGURA 6 è schematizzato un archivio organizzato in maniera *indexed* su due distinti indici definiti rispettivamente sul nominativo e sul numero di telefono. L'archivio centrale (primario) è gestito in maniera random, mentre i due indici hanno le rispettive chiavi mantenute costantemente ordinate e accanto a ognuna di esse viene gestito il riferimento al relativo record nell'archivio primario. Accedendo all'archivio attraverso un indice o l'altro è possibile «vedere» i record ordinati secondo i due diversi criteri.

## 6 Basi di dati e sistemi di gestione delle basi di dati

### DBMS

I DBMS commerciali più diffusi sono: Oracle DB (il discendente del primo DBMS relazionale rilasciato nel 1979), Microsoft SQL-server e IBM DB2. A questi si affiancano due noti prodotti *open source*: PostgreSQL e MySQL. Quest'ultimo è attualmente distribuito da Oracle Corporation con licenza sia commerciale sia libera.

Alla fine degli anni '60 del secolo scorso furono introdotti i primi DBMS: da allora questo settore dell'informatica ha conosciuto uno sviluppo costante, sia dal punto di vista della diffusione, interessando aree applicative sempre più vaste, sia teorico, dando luogo a proposte di sistemi anche molto diversi tra loro per gli aspetti logici e le capacità operative.

► Un **DBMS** (*Data Base Management System* o sistema di gestione delle basi di dati) è un sistema software in grado di gestire grandi collezioni di dati integrate, condivise e persistenti assicurando loro affidabilità e privacy. Come ogni prodotto informatico un DBMS deve essere efficiente ed efficace.

► Una **base di dati** (o database) è una collezione di dati gestita da un DBMS (un DBMS può gestire diverse basi di dati distinte).

Schematicamente si può affermare che gli scopi che hanno portato allo sviluppo di questi sistemi siano fondamentalmente i seguenti:

- rendere possibile una ricca definizione degli aspetti strutturali dei dati;
- rendere possibili convenienti interazioni tra gli aspetti dinamici relativi all'elaborazione dei dati mediante procedure e gli aspetti statici relativi alla struttura dei dati.

Questi due aspetti, che sono alla base delle differenze tra l'approccio basato su *file system* e quello fondato su DBMS, sono illustrati nelle FIGURE 7 e 8.

Nel superato approccio basato su *file system* ogni programma dispone dei propri file di dati anche se alcuni di essi sono condivisi allo scopo di ridurre

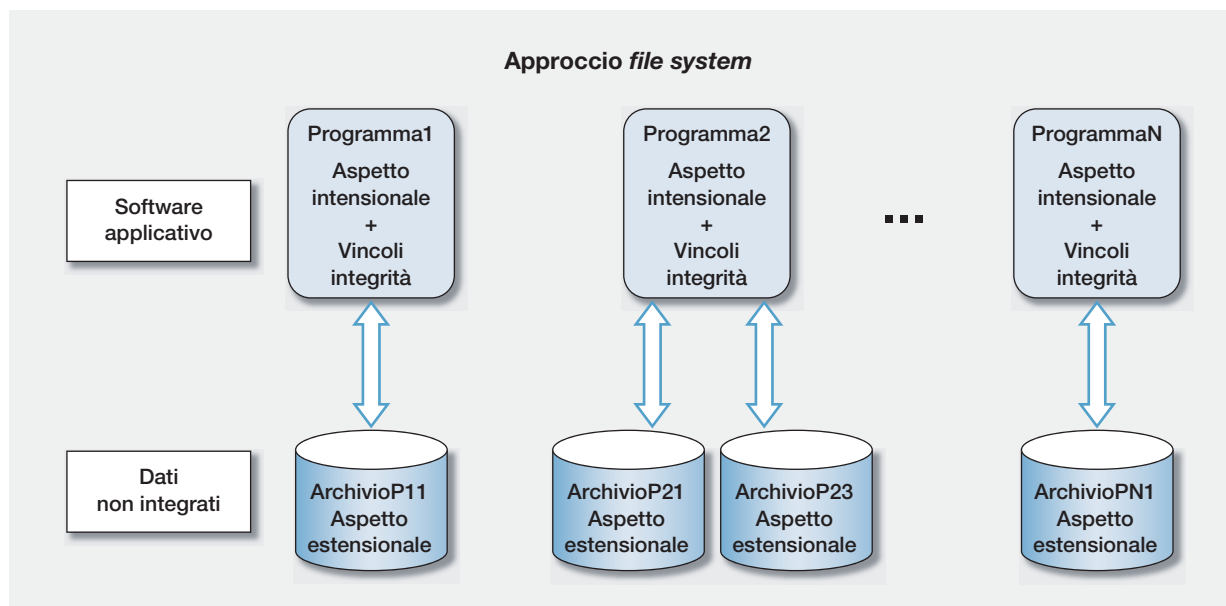


FIGURA 7

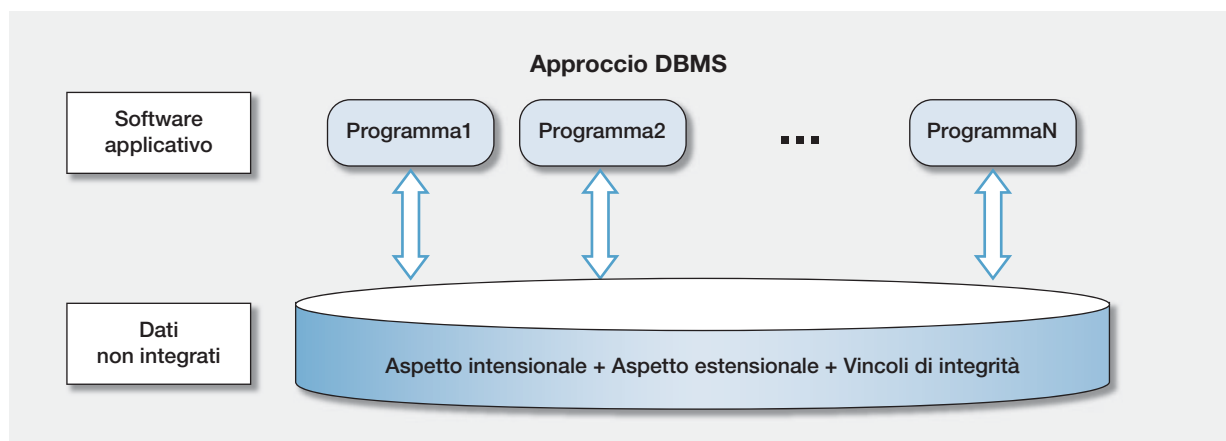


FIGURA 8

## Approccio fondato su DBMS con file system

La diffusione universale dell'approccio fondato su DBMS alla gestione dei dati da parte di applicazioni software ha portato a soluzioni che ne fanno uso anche in assenza di un DBMS vero e proprio.

«SQL lite», per esempio, è una libreria software che può essere inclusa in programmi scritti in linguaggio C/C++ o PHP e che consente di «vedere» i dati contenuti in un unico file come se fossero gestiti da un DBMS vero e proprio.

«Apache Derby» è invece un DBMS compatto, inizialmente sviluppato da IBM, che può essere incorporato in un programma Java mantenendo i dati nella memoria RAM del computer: è noto anche come «Java DB».

la ridondanza dei dati: il problema di questo approccio risiede nel fatto che l'aspetto intensionale dei dati e le regole di integrità a cui devono soddisfare sono implementati dal codice dei programmi che gestiscono i file.

**OSSERVAZIONE** Se si ha la necessità di modificare la struttura dei dati (per esempio semplicemente per aggiungere nuovi campi) o le regole di integrità, è necessario modificare tutti i programmi che sono coinvolti nella loro elaborazione.

Nell'approccio fondato su DBMS i dati sono memorizzati una sola volta in modo integrato: il database contiene infatti sia l'aspetto estensionale sia l'aspetto intensionale dei dati, compresi i vincoli di integrità. I programmi non prevedono al loro interno la definizione della struttura dei dati, ma fanno riferimento ai soli nomi dei campi che devono elaborare: in questo modo è possibile modificare la struttura dei dati senza che questo implichi modifiche ai programmi applicativi (o, eventualmente, solo a un numero limitato di questi).

**OSSERVAZIONE** Nell'approccio basato su *file system* il significato dei dati e delle relazioni che «legano» i vari file tra di loro (per esempio i clienti e le relative fatture, oppure i docenti e i loro studenti) sono immersi – e quindi «nascosti» – nel codice delle procedure di gestione, rendendo difficile l'interpretazione della struttura informativa dello scenario a cui si riferiscono.

Nell'approccio fondato su DBMS questi aspetti sono invece espliciti a priori come parte integrante della definizione dei dati che realizzano una chiara descrizione della realtà, in modo indipendente dalle procedure di elaborazione.

I principali punti che qualificano il ricorso all'approccio fondato su DBMS sono i seguenti.

- **Integrazione.** Una base di dati è un insieme integrato di dati strutturati e permanenti memorizzati senza ridondanze superflue e organizzati in modo tale da poter essere usati da applicazioni diverse senza dipendere da alcuna di esse.
- **Indipendenza logica.** I dati sono definiti indipendentemente dalle procedure che li gestiscono: in questo modo la struttura logica della base di dati può essere ampliata senza la necessità di modificare i programmi applicativi.
- **Indipendenza fisica.** Descrive la struttura dei dati astraendo da quella che è la loro implementazione fisica (organizzazione della memorizzazione, modalità di accesso, ecc.) in modo tale che si possa modificare quest'ultima senza modificare la struttura logica dei dati e di conseguenza i programmi applicativi.
- **Integrità.** È il sistema di gestione delle basi di dati e non le procedure di gestione a dover prevedere meccanismi per controllare che i dati inseriti o modificati soddisfino ai vincoli di integrità specificati.

È possibile classificare l'utenza di un sistema di gestione della base di dati nel seguente modo.

- **Programmatori di applicazioni.** Utenti professionali che hanno il compito di sviluppare applicazioni software che si interfacciano con una base di dati mediando l'accesso ai dati per gli utenti finali.
- **Utenti finali.** Utenti non professionali che interagiscono con la base di dati esclusivamente mediante programmi applicativi realizzati per svolgere determinate attività operative di gestione ed elaborazione dei dati nell'ambito di una organizzazione (per esempio: addetti a sportelli postali o bancari, magazzinieri, utenti web, ecc.) senza conoscere la struttura dei dati con cui interagiscono.
- **Utenti avanzati.** Utenti che conoscono la struttura dei dati e sono in grado di operare attività di indagine utilizzando un linguaggio generico di interrogazione della base di dati senza alterarla.
- **Utenti amministratori (DBA, Data Base Administrator).** Utenti professionali a cui è demandata la manutenzione della base di dati nel tempo: l'amministratore, utilizzando gli strumenti di amministrazione resi disponibili dal sistema DBMS, è l'unico soggetto che può, in funzione delle necessità, modificare la struttura della base di dati (aspetto intenzionale), definire o modificare i diritti di accesso ai dati per ogni singolo utente del DBMS, ecc.

L'utente interagisce con il sistema di gestione della base di dati tramite specifici linguaggi:

- **DDL (Data Definition Language):** linguaggi per la definizione della struttura dei dati; sono usati principalmente dall'amministratore del sistema e sono costituiti da comandi che consentono di definire e modificare la struttura della base di dati e dei vincoli di integrità;
- **DML (Data Manipulation Language):** linguaggi per la gestione e l'utilizzazione dei dati contenuti in una base di dati; sono costituiti da comandi che permettono di accedere ai dati per effettuare operazioni di interrogazione o manipolazione (inserimento, eliminazione, modifica). I linguaggi DML si distinguono in:
  - **ospitati:** linguaggi «immersi» in un linguaggio di programmazione e utilizzati dagli sviluppatori software per codificare le applicazioni che interagiscono con le basi di dati di un DBMS;
  - **autonomi:** linguaggi indipendenti, solitamente privi dei costrutti che ne consentirebbero l'uso per l'accesso ai dati da parte di utenti finali non avanzati.

L'approccio fondato su DBMS prevede che la fase di realizzazione possa essere scomposta in due sottofasce distinte:

- **progettazione logica:** consiste nella conversione del progetto concettuale in un «progetto logico», cioè nella strutturazione dei dati e nella formalizzazione delle procedure applicative utilizzando il linguaggio DDL e gli operatori previsti a tale scopo dal DBMS;
- **progettazione fisica:** consiste nella descrizione dell'organizzazione fisica dei file che realizzano la base di dati del sistema informatico.

## Modello dei dati

Si può affermare che la teoria delle basi di dati come disciplina informatica è nata con la nozione di modello dei dati.

Intuitivamente un modello dei dati è uno strumento concettuale non immediato da definire e che assume sfumature diverse a seconda degli autori.

Il termine **modello dei dati** fu introdotto alla fine degli anni '60 del secolo scorso quando fu evidente che esistevano diversi modelli cui ispirarsi per esprimere la semantica dei dati: più precisamente il termine fu proposto nel 1970 da Edgar Codd, al tempo ricercatore dell'IBM, in coincidenza della presentazione del modello relazionale dei dati (a cui è dedicata un'ampia parte in questo libro) che è divenuto in seguito dominante e che consente al progettista di attribuire un certo significato (o interpretazione) ai dati.

Il significato viene attribuito principalmente assegnando una struttura ai dati mediante specifici meccanismi di strutturazione previsti dal modello stesso: infatti, come abbiamo già osservato, i dati di per sé non producono informazione in assenza di una interpretazione. Possiamo quindi dire che un modello dei dati è uno strumento concettuale mediante il quale si può acquisire conoscenza da un insieme di dati altrimenti insignificanti.

**OSSERVAZIONE** La definizione dei nomi dei campi è un tipico aspetto della progettazione logica, invece la definizione dei tipi (numerico, testuale, data, valuta, ecc.) dei campi è un tipico aspetto della progettazione fisica.

Questi due aspetti sono oggetto dei capitoli che seguono: in ogni caso la progettazione di un sistema informatico è un processo normalmente complesso che, anche se comporta un'attività di tipo creativo, deve essere affrontato impiegando una precisa metodologia e utilizzando adeguati strumenti software.

## 7 Architettura logica di un sistema di gestione delle base di dati

L'architettura di un sistema di gestione di database può essere schematizzata come in FIGURA 9.

Seguendo un modello ormai classico essa è strutturata su tre livelli.

- **Livello logico utente.** Ogni utente ha un proprio spazio di lavoro (cioè un'area di input/output che permette di ricevere/trasmettere i dati della base di dati nelle operazioni di lettura/scrittura) che si interfaccia con

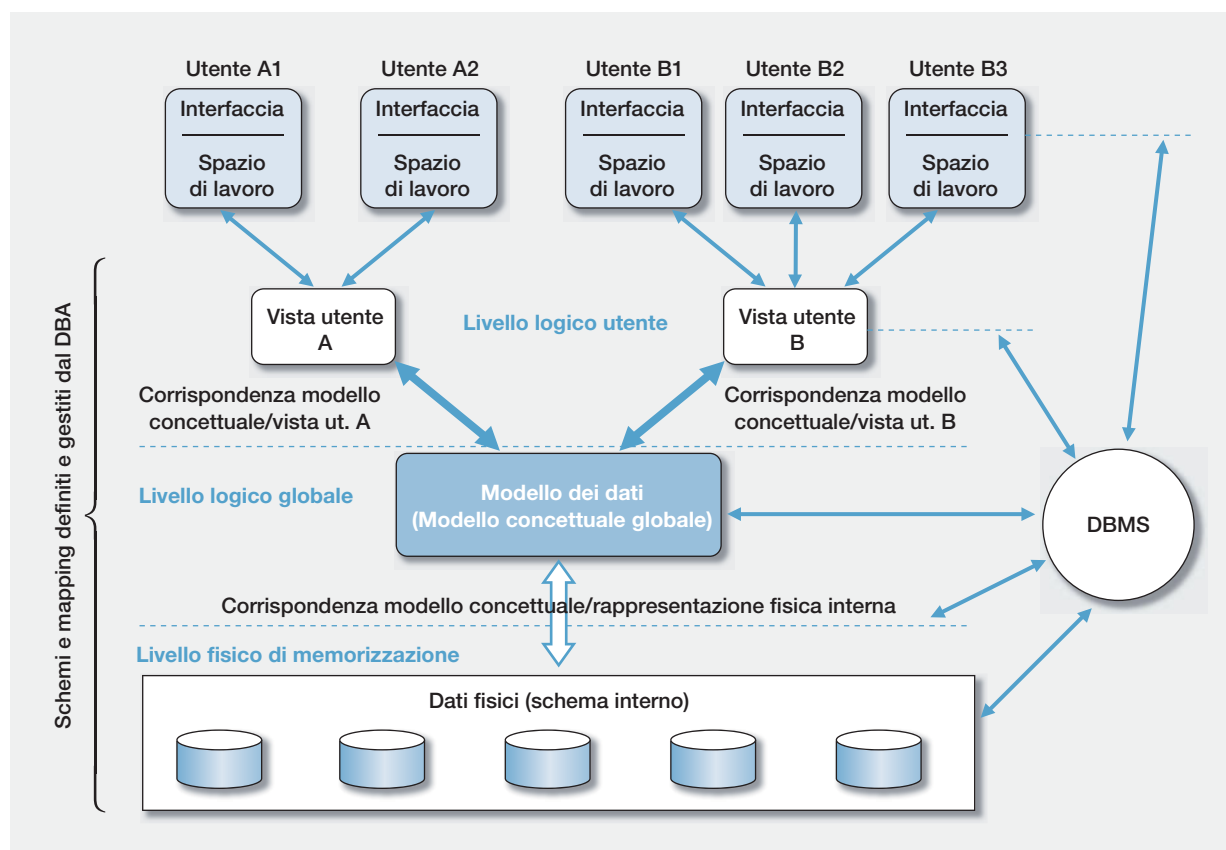


FIGURA 9

la base di dati mediante un'applicazione, o direttamente tramite il linguaggio DML. A questo livello sono definite dall'amministratore utilizzando il linguaggio DDL le «viste utente», ovvero dei sottoinsiemi del modello logico globale dell'intera base di dati: le viste utente realizzano i meccanismi fondamentali della privacy dei dati.

- **Livello logico globale.** È relativo alla definizione della struttura logica (aspetto intensionale dei dati, vincoli di integrità, ecc.) generale della base di dati: viene definito dall'amministratore utilizzando il linguaggio DDL.
- **Livello fisico di memorizzazione.** È il livello più basso dove si trovano i dati fisici e pertanto relativo ai dispositivi di memorizzazione, all'organizzazione fisica dei dati e alle relative modalità di accesso.

**OSSERVAZIONE** Nello schema i collegamenti tra i livelli sono realizzati dal DBMS che gestisce l'intera struttura mediante tecniche di corrispondenza (*mapping*), cioè procedure software che associano gli elementi di un livello ai corrispondenti di quello contiguo.

## Sintesi

■ **Dato.** Misura primaria del fenomeno osservato.

■ **Informazione.** Si ottiene dall'elaborazione di un insieme di dati; è in grado di accrescere lo stato di conoscenza di un fenomeno. La qualità dell'informazione è legata agli aspetti qualitativo e quantitativo dei dati elaborati e ha una validità limitata nel tempo.

■ **Sistema informativo.** Insieme strutturato di procedure e di risorse umane e materiali finalizzate alla raccolta, archiviazione, elaborazione e comunicazione di dati al fine di produrre le informazioni necessarie a un'organizzazione per gestire sia le attività operative sia quelle di governo.

■ **Sistema informatico.** È il sottoinsieme di un sistema informativo dedicato alla gestione automatica di informazioni derivanti dalla gestione di dati in formato digitale.

■ **Ciclo di vita di un sistema informatico.** Processo ciclico permanentemente in vita relativo alla progettazione di un sistema informatico; normalmente prevede una serie di attività raggruppabili in tre fasi concettualmente distinte: fase di **raccolta delle richieste degli utenti**, fase di **proget-**

**tazione concettuale** e fase di **realizzazione (o progettazione logica e fisica)**. Le tre fasi producono in output rispettivamente: la definizione dei requisiti a cui il sistema informatico deve essere conforme, il progetto concettuale e il progetto logico e fisico che rappresenta l'insieme delle componenti software che implementano il sistema informatico stesso.

■ **Approccio basato su file system.** È l'approccio obsoleto alla gestione dei dati in un sistema informatico: prevede il ricorso a linguaggi di programmazione per la gestione di file di dati basate sulle organizzazioni rese disponibili dal *file system* del sistema operativo.

■ **Approccio fondato su DBMS.** È l'approccio attualmente più utilizzato per la gestione dei dati in un sistema informatico: prevede l'impiego di DBMS (*Data Base Management System*), sistemi software in grado di gestire grandi collezioni di dati integrate, condivise e persistenti allo scopo di eliminare la duplicazione dei dati e di garantirne l'integrità e la privacy.

■ **Aspetti intensionale ed estensionale dei dati.** L'aspetto intensionale dei dati è relativo alle informazioni necessarie per la loro corretta in-



interpretazione. Nel caso di un file l'aspetto intensionale è dato dalla struttura del record. L'aspetto estensionale dei dati è invece relativo al valore dei dati stessi: in un file fa quindi riferimento al suo contenuto.

■ **File (archivio).** Un file di dati (o archivio) è un insieme di dati correlati identificato da un nome, memorizzato permanentemente su un supporto di memoria di massa e avente vita indipendente dal/dai programma/i utilizzato/i per la sua creazione e/o modifica. Le operazioni fondamentali che si possono applicare a un file sono: **creazione, apertura/chiusura, inserimento di dati, modifica di dati, cancellazione di dati, ricerca di dati.**

■ **Record (registrazione).** Unità logiche in cui i dati sono raggruppati in un file.

■ **Organizzazione di un file.** Questa espressione indica sia il modo in cui il file è memorizzato sul supporto fisico della memoria di massa (organizzazione fisica), sia il modo in cui il contenuto del file viene elaborato (organizzazione logica). Le organizzazioni classiche sono: **sequenziale**, in cui i record sono inseriti e letti uno di seguito all'altro, dal primo all'ultimo; **diretta (o casuale, random)**, in cui i record sono inseriti e letti specificando il numero della loro posizione all'interno del file; **indicizzata (indexed)**, in base alla quale nella struttura del record del file si individua un campo chiave per mantenere l'ordinamento dei record in funzione dei dati contenuti; parallelamente al file primario dove sono memorizzati i record in ordine di immissione, viene gestito un file indice che contiene una tabella delle chiavi.

■ **Modalità di accesso a un file.** Questa espressione indica la modalità con cui l'organizzazione del file consente l'accesso ai singoli record di dati. L'organizzazione sequenziale consente esclusivamente l'accesso sequenziale; l'organizzazione casuale (*random*) consente sia l'accesso diretto, specificando il numero di posizione del record nel file, sia l'accesso sequenziale; l'organizzazione indicizzata (*indexed*) consente sia l'accesso diretto per valore chiave, sia l'accesso sequenziale in base all'ordinamento predefinito di valori del campo chiave.

■ **DBMS (Database Management System).** Sistema software in grado di gestire grandi collezioni di dati integrate, condivise e persistenti as-

sicurando la loro affidabilità e privacy e perseguendo obiettivi di efficienza ed efficacia.

■ **Base di dati (database).** Una base di dati è una collezione di dati gestita da un DBMS (un DBMS può gestire diverse basi di dati).

■ **Integrazione dei dati.** Una base di dati è vista come un insieme integrato di dati strutturati e permanentemente memorizzati senza ridondanze superflue e organizzati in modo tale da poter essere usati da applicazioni diverse senza dipendere da alcuna di esse.

■ **Indipendenza logica e fisica dei dati.** In un database il fatto che i dati siano descritti indipendentemente dalle procedure di gestione comporta che la struttura logica della base di dati può essere ampliata senza dover modificare i programmi applicativi (indipendenza logica). Inoltre la possibilità di descrivere la struttura dei dati astraendo da quella che è la loro rappresentazione fisica (organizzazione di memorizzazione, modalità di accesso, ecc.) permette la loro indipendenza fisica, ovvero la possibilità di modificare i supporti di memorizzazione senza dover modificare la struttura dei dati.

■ **Integrità dei dati.** In un DBMS è il sistema stesso e non le procedure gestionali a prevedere strumenti per controllare che i dati inseriti o modificati soddisfino ai vincoli di integrità definiti per uno specifico database.

■ **Amministratore (DBA, Data Base Administrator).** È la figura professionale a cui è demandata la manutenzione della base di dati nel tempo. L'amministratore, utilizzando specifici strumenti di amministrazione resi disponibili dal DBMS, è l'unico che può, in funzione delle necessità, modificare la struttura (aspetto intensionale) della base di dati, definire o modificare i diritti di accesso ai dati per ogni utente, ecc.

■ **DDL (Data Definition Language).** Linguaggio per la definizione dei dati usato dal DBA; è costituito da comandi che permettono di definire e modificare la struttura della base di dati e i vincoli di integrità a cui essi debbono soddisfare per essere validi.

■ **DML (Data Manipulation Language).** Linguaggio per l'interrogazione e l'uso dei dati; è costituito da comandi che permettono di accedere ai dati per effettuare operazioni di interrogazione o

manipolazione. Un linguaggio DML è solitamente distinti in: «ospitato», cioè immerso in un linguaggio di programmazione utilizzato per codificare le applicazioni che consentono l'accesso ai dati agli utenti non professionali; oppure «autonomo».

■ **Architettura logica di un sistema di gestione delle basi di dati.** Seguendo un modello classico essa è strutturata su tre livelli: livello logico utente, livello logico globale e livello fisico di memorizzazione.

■ **Livello logico utente.** Ogni utente ha un proprio spazio di lavoro (cioè un'area di input/output che permette di ricevere/trasmettere i dati della base di dati nelle operazioni di lettura/scrittura) che si interfaccia con la base di dati mediante un'applicazione, o direttamente tramite il linguaggio

DML. A questo livello sono definite dal DBA utilizzando il linguaggio DDL le «viste utente», ovvero dei sottoinsiemi del modello logico globale dell'intera base di dati: le viste utente realizzano i meccanismi fondamentali della privacy dei dati.

■ **Livello logico globale.** È relativo alla definizione della struttura logica (aspetto intensionale dei dati, vincoli di integrità, ecc.) generale della base di dati: viene definito dal DBA utilizzando il linguaggio DDL.

■ **Livello fisico di memorizzazione.** È il livello più basso dove si trovano i dati fisici e pertanto relativo ai dispositivi di memorizzazione, all'organizzazione fisica dei dati e alle relative modalità di accesso.

## QUESITI

### 1 Un dato è ...

- A ... l'interpretazione di un fenomeno che siamo interessati a osservare.
- B ... la misura di un fenomeno che siamo interessati a osservare.
- C ... è la stessa cosa di un'informazione.
- D Nessuna delle risposte precedenti.

### 2 Un'informazione è ...

- A ... la stessa cosa di un dato.
- B ... la misura di un fenomeno che siamo interessati a osservare.
- C ... ciò che si ottiene dall'elaborazione di un insieme di dati e che accresce lo stato di conoscenza relativo a un fenomeno che siamo interessati a osservare.
- D Nessuna delle risposte precedenti.

### 3 Un sistema informativo è ...

- A ... un sottoinsieme di un sistema informatico.
- B ... un insieme strutturato di procedure e di risorse umane e materiali finalizzate, tramite

uso di tecnologia informatica, alla raccolta, archiviazione, elaborazione e comunicazione di dati allo scopo di ottenere le informazioni necessarie per gestire sia le attività operative sia quelle di governo di un'azienda.

- C ... un insieme strutturato di procedure e di risorse umane e materiali finalizzate alla raccolta, archiviazione, elaborazione e comunicazione di dati allo scopo di ottenere le informazioni necessarie per gestire sia le attività operative sia quelle di governo di un'azienda.
- D Nessuna delle risposte precedenti.

### 4 Quali delle seguenti sono macrofasi del ciclo di vita di un sistema informatico?

- A Raccolta delle richieste degli utenti.
- B Acquisto delle componenti hardware.
- C Progettazione concettuale.
- D Acquisto delle componenti software.

### 5 Che cos'è un analista?

- A È colui che effettua la manutenzione di un sistema informatico nel tempo.
- B È il sistemista che deve supervisionare al funzionamento di un sistema informatico.



C È un programmatore che deve sviluppare specifici moduli software di un sistema informatico.

D È colui che viene incaricato di studiare le caratteristiche di un sistema informativo in modo da poter definire le caratteristiche di un sistema informatico di supporto.

**6** Quali dei seguenti sono elementi che caratterizzano il progetto concettuale di un sistema informatico?

A Linguaggio/i da utilizzare per sviluppare il software.

B La struttura dei dati in termini di insiemi e relazioni fra insiemi.

C Caratteristiche dell'hardware da utilizzare.

D I vincoli di ammissibilità dei dati.

**7** Quali delle seguenti affermazioni sono vere relativamente all'approccio *file system*?

A La struttura dei dati viene definita internamente ai programmi.

B È possibile ampliare la struttura dei dati senza dover modificare i programmi preesistenti.

C La ridondanza dei dati è ridotta al minimo indispensabile.

D I vincoli di validità dei dati sono memorizzati insieme alla struttura dei dati.

**8** L'aspetto intensionale dei dati è relativo ...

A ... alle caratteristiche dei programmi che li gestiscono.

B ... a elementi che servono ad attribuire una corretta interpretazione del significato dei dati.

C ... al valore stesso dei dati.

D Nessuna delle risposte precedenti.

**9** Dovendo elaborare sistematicamente tutti i record di un archivio, quali delle seguenti organizzazioni è la più conveniente?

A Sequenziale.

B *Random*.

C *Indexed*.

D Nessuna delle risposte precedenti.

**10** Dovendo elaborare pochi record di un archivio sulla base di valori chiave forniti dall'utente, quali delle seguenti organizzazioni è la più conveniente?

A Sequenziale.

B *Random*.

C *Indexed*.

D Nessuna delle risposte precedenti.

**11** Dovendo elaborare i record di un archivio come se questo fosse un *array*, quali delle seguenti organizzazioni è la più conveniente?

A Sequenziale.

B *Random*.

C *Indexed*.

D Nessuna delle risposte precedenti.

**12** Una base di dati è ...

A ... una base su cui definire una struttura dati.

B ... una grande collezione di dati integrati, condivisi e persistenti.

C ... una grande collezione di dati non necessariamente integrati, condivisi o persistenti.

D Nessuna delle risposte precedenti.

**13** Un DBMS è ...

A ... l'equivalente di una base di dati.

B ... un ambiente software che permette di definire e gestire basi di dati.

C ... uno qualsiasi dei programmi applicativi che elaborano una base di dati.

D Nessuna delle risposte precedenti.

**14** I due termini *integrità* e *integrazione* riferiti a una base di dati ...

A ... sono equivalenti.

B ... il primo si riferisce al fatto che i dati sono memorizzati senza ridondanze superflue, mentre il secondo si riferisce ai vincoli cui i dati devono soddisfare per essere validi.

C ... il primo si riferisce ai vincoli cui i dati debbono soddisfare per essere validi, mentre il secondo si riferisce al fatto che i dati sono memorizzati senza ridondanze superflue.

D Nessuna delle risposte precedenti.

**15** I due termini DDL e DML riferiti a un DBMS ...

- A ... sono equivalenti e si riferiscono al personale addetto alla manutenzione del sistema in oggetto.
- B ... il primo indica un linguaggio per la definizione della struttura dei dati di un database, mentre il secondo si riferisce a un linguaggio per l'uso dei dati in esso contenuti.
- C ... il primo indica un linguaggio per l'uso dei dati contenuti in un database, mentre il secondo si riferisce a un linguaggio per la definizione della struttura sua dei dati.
- D Nessuna delle risposte precedenti.

**16** Indicare quale delle sequenze riportate di seguito indica l'ordine corretto dei livelli dell'architettura logica di una base di dati a partire dal livello utente.

- A Livello logico utente, modello concettuale globale, livello fisico di memorizzazione.
- B Livello fisico di memorizzazione, livello logico utente, modello concettuale globale.

- C Livello fisico di memorizzazione, modello concettuale globale, livello logico utente.
- D Nessuna delle risposte precedenti.

**17** Relativamente a un database, col termine *in-dipendenza fisica dei dati* si intende la possibilità ...

- A ... di descrivere la struttura dei dati astraendo da quella che è la loro implementazione fisica per definire vari livelli di privatezza dei dati.
- B ... di ampliare la struttura logica della base di dati senza la necessità di modificare i programmi applicativi.
- C ... di descrivere la struttura dei dati astraendo da quella che è la loro implementazione fisica (organizzazione della memorizzazione, modalità di accesso, ecc.) in modo tale che si possa modificare quest'ultima senza modificare la struttura logica dei dati e di conseguenza i programmi applicativi.
- D Nessuna delle risposte precedenti.

**asset**

bene, attività

**the foregoing**

la cosa precedente

**to accrue**

aumentare

**corollary**

corollario

**to arise**

presentarsi/sorgere

**to carry out**

effettuare

**to point out**

evidenziare

**elsewhere**

altrove

**fairly**

abbastanza

**to devote**

dedicare

## 1.2 Why Database?

Why should an enterprise choose to store its operational data in an integrated database? There are many answers to this question. One general answer is that it provides the enterprise with *centralized control* of its operational data [...] **is** its most valuable asset. This is in sharp contrast to the situation [...], where typically each application has its own private files [...] **too**—so that the operational data is widely dispersed, and there is little or no attempt to control it in a systematic way.

The foregoing implies that in an enterprise with a database system there will be some one identifiable person – the *database administrator*, or DBA – who has this central responsibility for the operational data. In fact, we may consider the DBA as part of the database system (that is, the system is more than just data plus software plus hardware – it includes people, too). We shall be discussing the role of the DBA in more detail later; for the time being, it is sufficient to note that the job will involve both a high degree of technical expertise and the ability to understand and interpret management requirements at a senior level. (In practice the DBA may consist of a team of people instead of just one person.) It is important to realize that the position of the DBA within the enterprise is a very senior one.

Let us now consider the advantages that accrue from having centralized control of the data, as discussed above.

- **The amount of redundancy in the stored data can be reduced.**

In most current systems each application has its own private files. This can often lead to considerable redundancy in stored data, with resultant waste in storage space. For example, a personnel application and an education-records application may each own a file containing name, number, and department for every employee. With central control, however, the DBA can identify the fact that the two applications require essentially the same data, and hence can integrate the two files; that is, the data can be stored once only and can be shared by the two applications.

- **Problems of inconsistency in the stored data can be avoided (to a certain extent).**

This is really a corollary of the previous point. If the “same” fact about the real world – say, the fact that employee E3 works in department D8 – is represented by two distinct entries in the database, then at some time the two entries will not agree (i.e., when one and only one has been updated). The database is then inconsistent. If, on the other hand, the information is represented by a single entry – if the redundancy has been removed – such an inconsistency cannot arise.

- **The stored data can be shared.**

This point has already been made in passing but is worth stating here as an important advantage in its own right. It means not only that all the files of existing applications are integrated, but also that new applications may be developed to operate against the existing database.

- **Standards can be enforced.**

With central control of the database, the DBA can ensure that installation and industry standards are followed in the representation of the data. This simplifies problems of maintenance and data interchange between installations.

- **Security restrictions can be applied.**

Having complete jurisdiction over the operational data, the DBA (a) can ensure that the only means of access to the database is through the proper channels, and hence (b) can define authorization checks to be carried out whenever access to sensitive data is attempted. Different procedures can be established for each type of access (retrieve, update, delete, etc.) to each type of data field in the database. [Perhaps it should also be pointed out that without such procedures the security of the data may actually be more at risk in a database system than in a traditional (dispersed) filing system.]

- **Data integrity can be maintained.**

The problem of integrity is the problem of ensuring that the database contains only accurate data. Inconsistency between two entries representing the same “fact” is an example of lack of integrity (which of course can only occur if redundancy exists in the stored data). Even if redundancy is eliminated, however, the database may still contain incorrect data. For example, an employee may be shown as having worked 200 hours in the week, or a list of employee numbers for a given department may include the number of a nonexistent employee. Centralized control of the database helps in avoiding these situations (in so far as they can be avoided) by permitting the DBA to define validation procedures to be carried out whenever any storage operation is attempted. (Here, as elsewhere, we use the term “storage operation” to cover all the operations of updating, inserting, and deleting.)

- **Conflicting requirements can be balanced.**

Knowing the overall requirements of the enterprise – as opposed to the requirements of any individual user – the DBA can structure the database system to provide an overall service that is “best for the enterprise.” For example, a representation can be chosen for the data in storage that gives fast access for the most important applications at the cost of poor performance in some other applications.

Most of the advantages listed above are fairly obvious. However, one other point, which is not so obvious – although it is implied by several of the foregoing – must be added to the list, namely, the provision of data independence. (Strictly speaking, this is an objective rather than an advantage.) This concept is so important that we devote a separate section to it.

[C.J. Date, *An Introduction to Database Systems*, Addison Wesley Publishing Company, 1979]

## QUESTIONS

- a What does the acronym DBA stand for?
- b Briefly describe the problem of data integrity.
- c What are security restrictions?
- d Briefly describe the problem of data inconsistency.
- e Describe the relationship between data inconsistency and data redundancy.

# Le basi di dati relazionali

Si deve realizzare un sistema informatico per gestire i dati di una piccola biblioteca; in questo scenario si possono individuare alcune classi di dati fondamentali insieme ai loro attributi, cioè le proprietà caratteristiche che li descrivono:

- per i libri: il codice ISBN, il titolo, l'anno di pubblicazione, la lingua, il prezzo;
- per gli autori: il cognome e il nome, la nazionalità, la data di nascita, il sesso;
- per gli editori: la ragione sociale (cioè la denominazione), l'indirizzo, la città;
- per i generi, cioè le categorie con cui si possono classificare i libri (romanzi, gialli, saggi, ecc.): la descrizione del genere stesso.

Inoltre si osservano i seguenti fatti:

- un autore può avere scritto più di un libro, anche se ogni libro è stato scritto da un solo autore (ignoriamo per ora situazioni in cui un libro può essere scritto da più autori, come è spesso il caso per i libri di genere tecnico-scientifico);
- un editore ha pubblicato numerosi libri, mentre normalmente un libro è pubblicato da un determinato editore;
- ogni genere comprende un intero insieme di libri.

Un modo per rappresentare questo scenario è quello di fornirne una rappresentazione grafica mediante un diagramma Entità/Associazioni (o Entità/Relazioni, nel seguito E/R) il cui formalismo è stato definito da Peter Chen nel 1976 (FIGURA 1).

Un tipo di formalismo grafico molto utilizzato dai progettisti di database e dagli sviluppatori software è invece quello illustrato in FIGURA 2<sup>1</sup>.

**OSSERVAZIONE** In quest'ultimo tipo di formalismo, in aggiunta agli attributi identificati per le classi di dati, sono previsti elementi identificativi (contrassegnati dal simbolo «PK», acronimo di *Primary Key*) e associativi (contrassegnati dal simbolo «FK», acronimo di *Foreign Key*) che, come vedremo più avanti, sono necessari per l'effettiva implementazione di un database.

In entrambi i diagrammi è immediato individuare gli insiemi di dati (*Libri*, *Autori*, *Editori*, *Genere*) e i relativi attributi che caratterizzano lo scenario della realtà da informatizzare e le associazioni che esistono tra essi.

1. È il formalismo reso disponibile dal software Microsoft Visio.

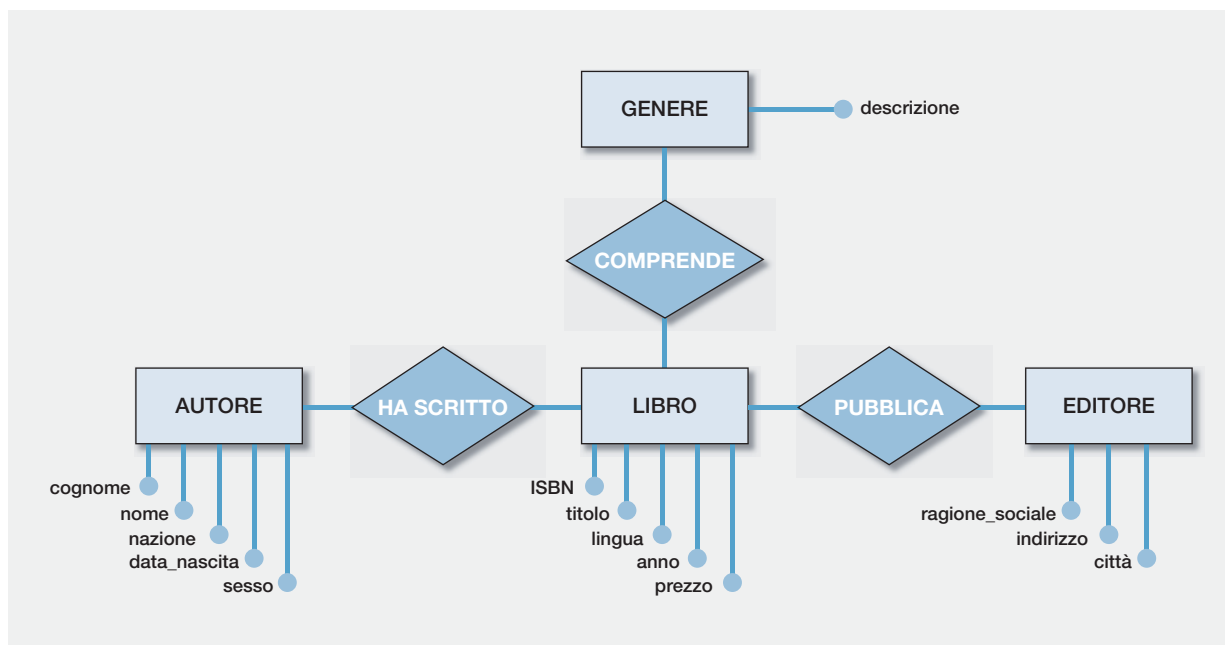


FIGURA 1

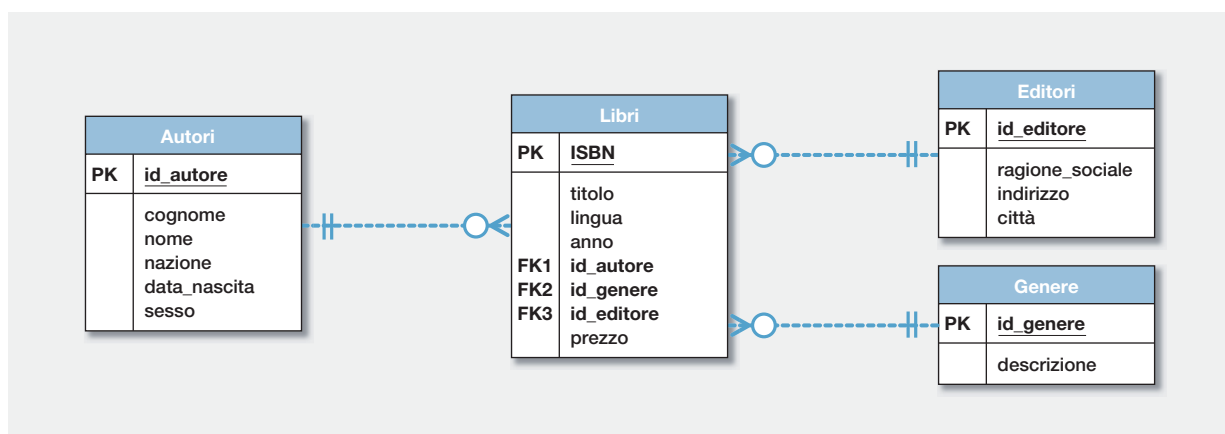


FIGURA 2

In particolare il formalismo grafico E/R viene utilizzato per la realizzazione di modelli logici di scenari reali che prescindono da come i dati saranno effettivamente memorizzati e organizzati nei computer di un sistema informatico.

**OSSERVAZIONE** Il modello E/R è ancora oggi uno degli strumenti concettuali più usati per la modellazione di dati in ambito informatico; viene impiegato non solo per la progettazione di database, ma anche come modello di riferimento per altri tipi di rappresentazione finalizzate alla modellazione: ad esempio il linguaggio grafico UML ha ereditato i concetti introdotti con il modello E/R.

Le varie fasi del processo di progettazione di una base di dati sono rappresentate nello schema di flusso di FIGURA 3.

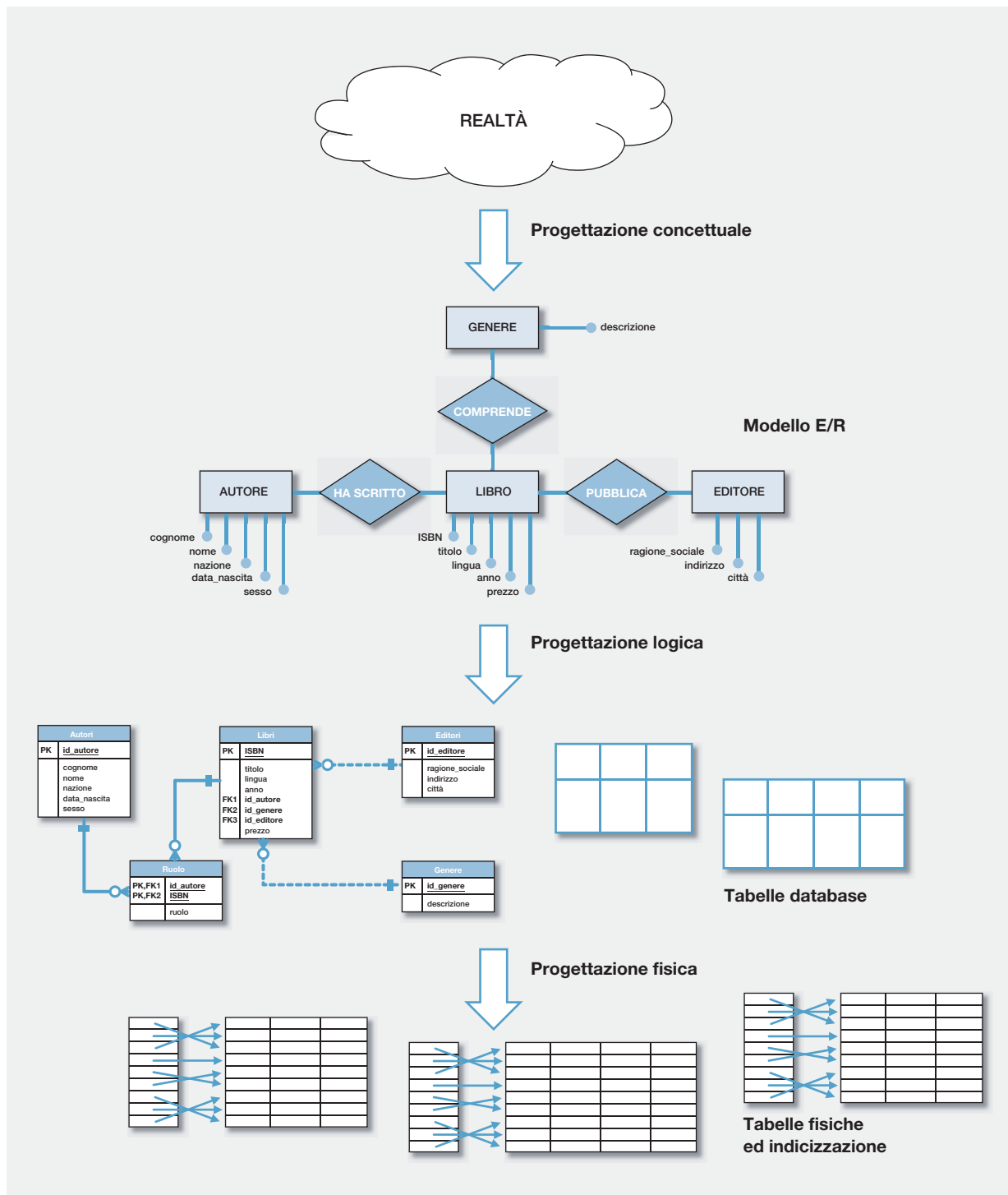


FIGURA 3

A partire dai risultati dell'analisi viene realizzato un modello della realtà da informatizzare mediante uno schema E/R che descrive a livello concettuale le caratteristiche della base di dati da implementare. Tale modello viene in seguito trasformato in uno schema relazionale costituito da una collezione di tabelle. Infine viene definita la struttura fisica dei dati (tipo e dimensioni dei campi) e vengono specificate le strutture ausiliarie (come gli indici di ricerca) per renderne efficiente la gestione dei dati.



# 1 Diagrammi Entità/Relazioni

Qualunque sia lo scenario della realtà da modellare, i concetti fondamentali su cui il modello E/R si basa sono i seguenti.

## 1. Entità

Rappresentano classi di oggetti (cose, persone, fatti, ecc.) che hanno proprietà comuni ed esistenza autonoma nel contesto dello scenario analizzato. Un'istanza (o occorrenza) di un'entità è un oggetto (o istanza) della classe che l'entità rappresenta: la persona X, il corso di laurea Y, l'automobile Z, sono esempi di istanze delle entità Persone, Corsi di laurea, Veicoli.

In un diagramma E/R ogni entità ha un nome che la identifica univocamente ed è rappresentata graficamente con un rettangolo etichettato con il nome dell'entità e l'elenco delle sue proprietà (o attributi).

ES.

Nello scenario dell'esempio considerato in apertura del capitolo sono state individuate le entità: *Autori*, *Libri*, *Editori* e *Genere*.

## 2. Associazioni

Le associazioni (o relazioni) costituiscono il mezzo tramite il quale sono modellate le corrispondenze tra le istanze di due o più entità. Il grado dell'associazione indica il numero di entità coinvolte.

**OSSERVAZIONE** Per progettare un valido schema E/R è in genere consigliato ricondursi a situazioni in cui vi sia una prevalenza di associazioni di grado 2 (cioè tra due sole entità).

Nel caso particolare di associazioni di grado 2 o binarie, cioè tra due entità A e B, si distinguono corrispondenze di cardinalità 1:1, 1:N e M:N. Se A rappresenta il dominio dell'associazione e B il suo codominio, si hanno i seguenti casi.

- **Associazioni 1:1.** Sono corrispondenze biunivoche: a ogni istanza della classe A ne corrisponde una della classe B e viceversa (FIGURA 4).

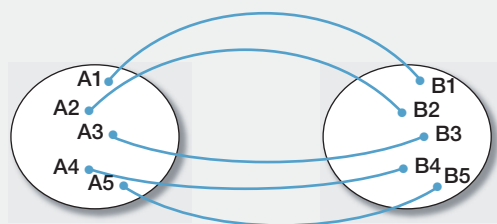


FIGURA 4

## Peter Chen

Il professor Peter Pin-Shan Chen è uno studioso americano di informatica di origine cinese, membro di rilievo della Carnegie Mellon University. Noto per la sua produzione scientifica ed è a lui che si deve lo sviluppo del modello E/R pubblicato nel 1976 nel famoso articolo «The entity-relationship model: toward a unified view of data». Il suo lavoro ha ispirato un nuovo campo di ricerca definito **Modellazione concettuale** (*Conceptual Modeling*): a partire dal 1979 si tiene annualmente una conferenza internazionale su questa tematica. Per ricordare Chen come fondatore di questa disciplina è stato istituito nel 2008 il premio annuale *Peter Chen Award*, da assegnare a un ricercatore che abbia prodotto un contributo significativo in questo campo.

- **Associazioni 1:N.** La corrispondenza diretta è multipla e in generale parziale, cioè a ogni istanza della classe *A* ne corrispondono zero, una o più della classe *B*, mentre l'associazione inversa è univoca e totale, cioè a ogni istanza della classe *B* ne corrisponde una e una sola della classe *A* (FIGURA 5).

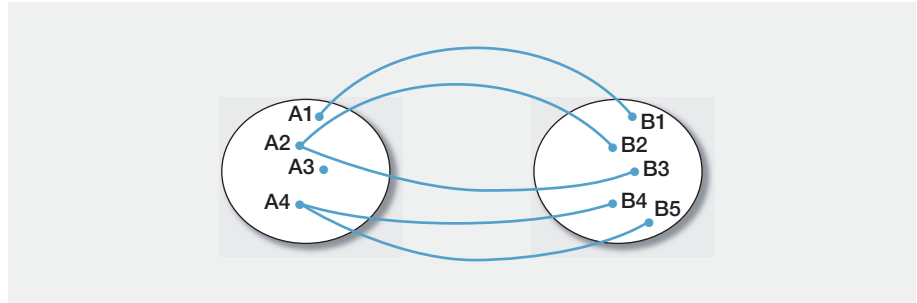


FIGURA 5

- **Associazioni M:N.** La corrispondenza è multipla e parziale in entrambe le direzioni, cioè a ogni istanza della classe *A* ne corrispondono zero, una o più della classe *B* e viceversa (FIGURA 6).

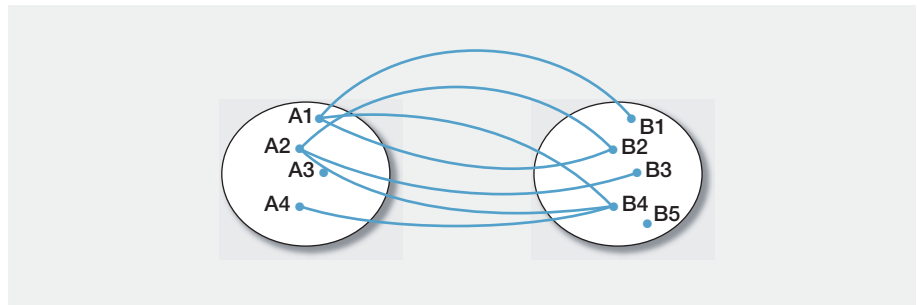


FIGURA 6

**OSSERVAZIONE** Un'associazione dalla classe *A* alla classe *B* (o viceversa) è **totale** se è definita per ogni istanza della classe *A* (o *B*), altrimenti è **parziale**.

Nel formalismo grafico originale di Chen la cardinalità delle associazioni viene indicata direttamente nel diagramma E/R.

## ESEMPIO

Per l'esempio di apertura, in cui sono state individuate le associazioni

- *Autori, Libri*: 1:N (un libro ha un solo autore, ma un autore può avere scritto più libri),
- *Editori, Libri*: 1:N (un libro ha un solo editore, ma un editore pubblica molti libri),
- *Genere, Libri*: 1:N (ogni libro viene classificato per uno specifico genere, ma per ogni genere esistono molti libri),

vale quindi il diagramma di FIGURA 7.

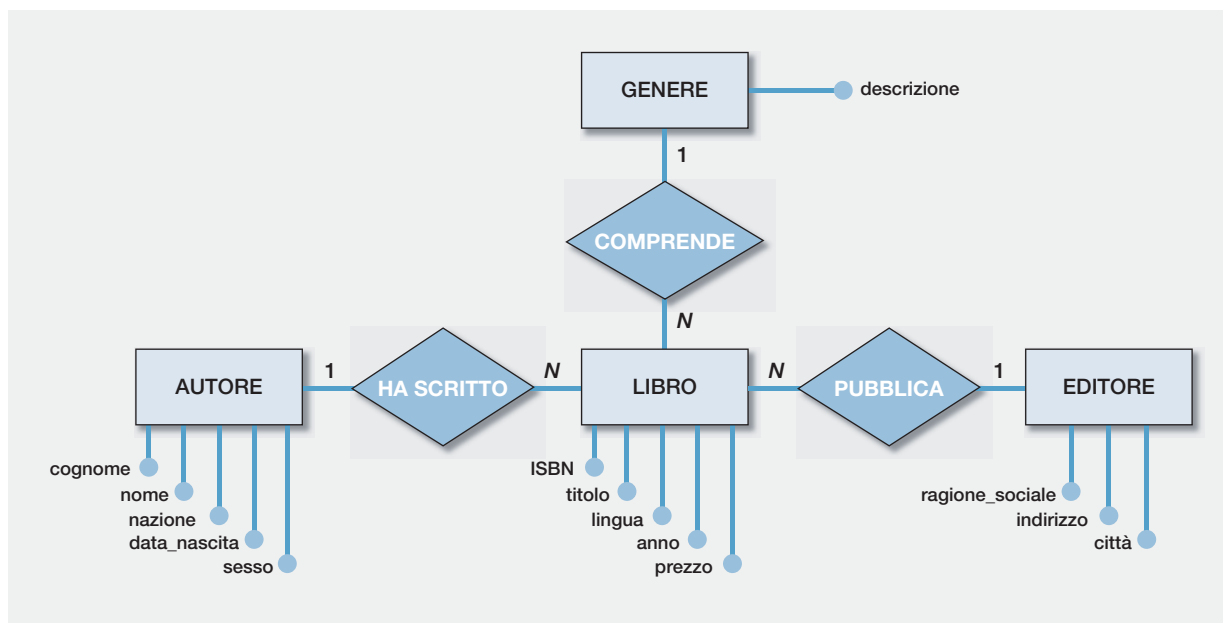


FIGURA 7

Nei diagrammi utilizzati dai progettisti di database e dagli sviluppatori software, un'associazione di tipo 1:N viene rappresentata mediante la seguente simbologia grafica



dove il connettore unico a sinistra insiste sulla classe dominio e quello multiplo a destra sulla classe codominio: i simboli esprimono graficamente la cardinalità della corrispondenza tra le istanze della classe dominio e quelle della classe codominio.

La casistica dei simboli utilizzati per rappresentare le associazioni è riportata nella TABELLA 1.

TABELLA 1

Simbolo	Descrizione
	Associazione 1:1 parziale
	Associazione 1:1 totale
	Associazione 1:N parziale
	Associazione 1:N totale

**OSSERVAZIONE** La simbologia illustrata non permette di rappresentare un'associazione di cardinalità N:M in quanto essa non è direttamente implementabile mediante un database di tipo relazionale; come sarà illustrato nel seguito questo tipo di associazione è sempre trasformabile in due associazioni di tipo 1:N.

### 3. Attributi

Le entità – e, in alcuni casi, anche le associazioni – possono essere descritte usando una serie di attributi (o proprietà): tutte le istanze della stessa entità (o associazione) hanno gli stessi attributi.

L'individuazione degli attributi riflette il livello di dettaglio con il quale si intendono rappresentare le informazioni relative alle entità (o alle associazioni).

#### ESEMPIO

Per l'entità *Autori* dell'esempio introduttivo sono stati individuati gli attributi:

- *cognome*: cognome dell'autore;
- *nome*: nome dell'autore;
- *nazione*: nazionalità dell'autore;
- *data\_nascita*: data di nascita dell'autore;
- *sex*: sesso dell'autore.

Inoltre, allo scopo di discriminare due autori eventualmente omonimi, aventi lo stesso sesso e nazionalità e nati lo stesso giorno, è stato introdotto l'attributo *id\_autore* come codice alfanumerico univoco per ogni specifico autore.

## 1.1 Chiavi primarie ed esterne

Per ciascuna entità (o, eventualmente, associazione) si definisce chiave primaria un insieme minimale di attributi (costituito cioè dal numero minimo di attributi) che identificano univocamente ciascuna istanza dell'entità (o dell'associazione).

#### ESEMPIO

Per l'entità *Libri* dell'esempio di apertura il codice *ISBN* – che è diverso per ogni libro – è una chiave primaria in quanto valori diversi individuano istanze diverse dell'entità. Sempre nell'esempio relativo alla biblioteca non esiste per l'entità *Autori*, tra gli attributi individuati, una chiave primaria (infatti due istanze diverse dell'entità, cioè due autori distinti, possono al limite condividere tutti i valori di tutti gli attributi: *cognome*, *nome*, *sex*, *nazione*, *data\_nascita*). In questo caso si aggiunge un attributo avente lo scopo di identificare univocamente la singola istanza mediante un codice: nello specifico, *id\_autore* per l'entità *Autore* e *id\_editore* per l'entità *Editore*. Il diagramma E/R diventa quindi quello di FIGURA 8, nel quale le chiavi primarie delle entità sono in colore diverso da quello delle altre proprietà.

**OSSERVAZIONE** Le necessità di gestione dei sistemi informativi in generale e dei sistemi informatici in particolare ha portato all'introduzione nella pratica quotidiana di molti «codici» che rappresentano chiavi primarie naturali in fase di progettazione di un database; solo a titolo di esempio ricordiamo: il codice fiscale per le persone fisiche, la partita IVA per le aziende, la targa per gli autoveicoli, il codice a barre per i prodotti.

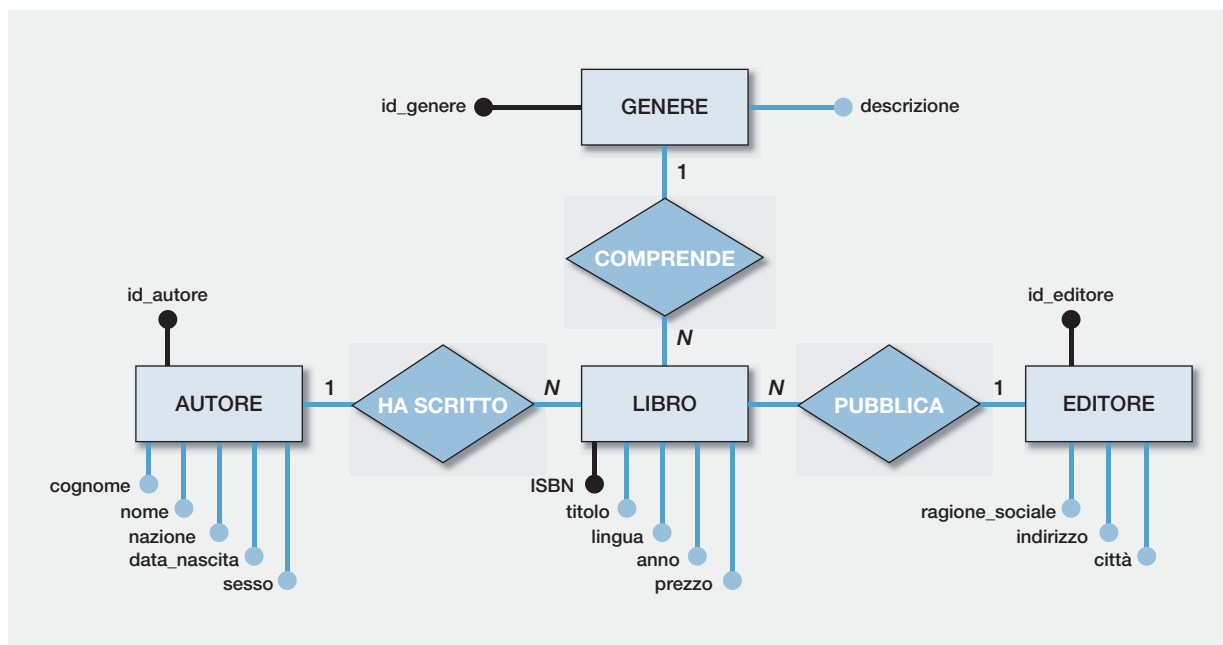


FIGURA 8

Nei diagrammi utilizzati dai progettisti software il nome dell'attributo (o i nomi degli attributi) che costituisce (o che costituiscono) la chiave primaria viene sottolineato e contrassegnato dall'etichetta «PK» (*Primary Key*).

Per modellare le associazioni di cardinalità 1:N si ricorre alle cosiddette chiavi esterne (FK, *Foreign Key*): l'associazione viene realizzata aggiungendo l'attributo o gli attributi che forma/no la chiave primaria dell'entità dominio all'entità codominio.

#### ESEMPIO

Per l'entità *Libri* dell'esempio di apertura il riferimento all'editore avviene inserendo tra gli attributi *id\_autore* per registrare il codice dell'editore (istanza dell'entità *Editori*) del libro (istanza dell'entità *Libri*).

La rappresentazione grafica del modello della biblioteca diventa quindi quello di FIGURA 9.

Come è stato osservato nel formalismo grafico a cui facciamo riferimento non è prevista la rappresentazione di associazioni di cardinalità M:N, che sono invece previste nel modello originale di Peter Chen.

#### ESEMPIO

Generalizzando il diagramma E/R che rappresenta lo scenario della biblioteca è più corretto considerare l'associazione tra *Libri* e *Autori* di cardinalità M:N, in quanto un libro può essere stato scritto da più autori (FIGURA 10).

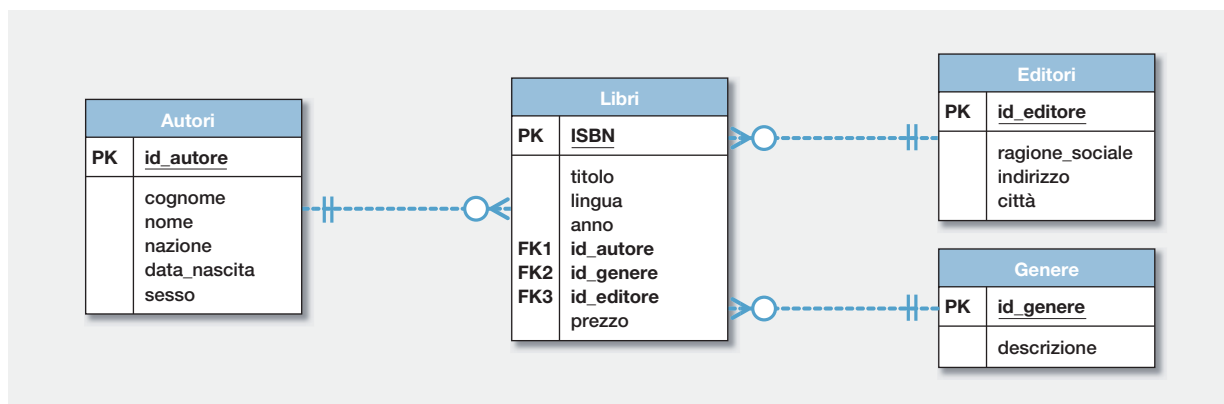


FIGURA 9

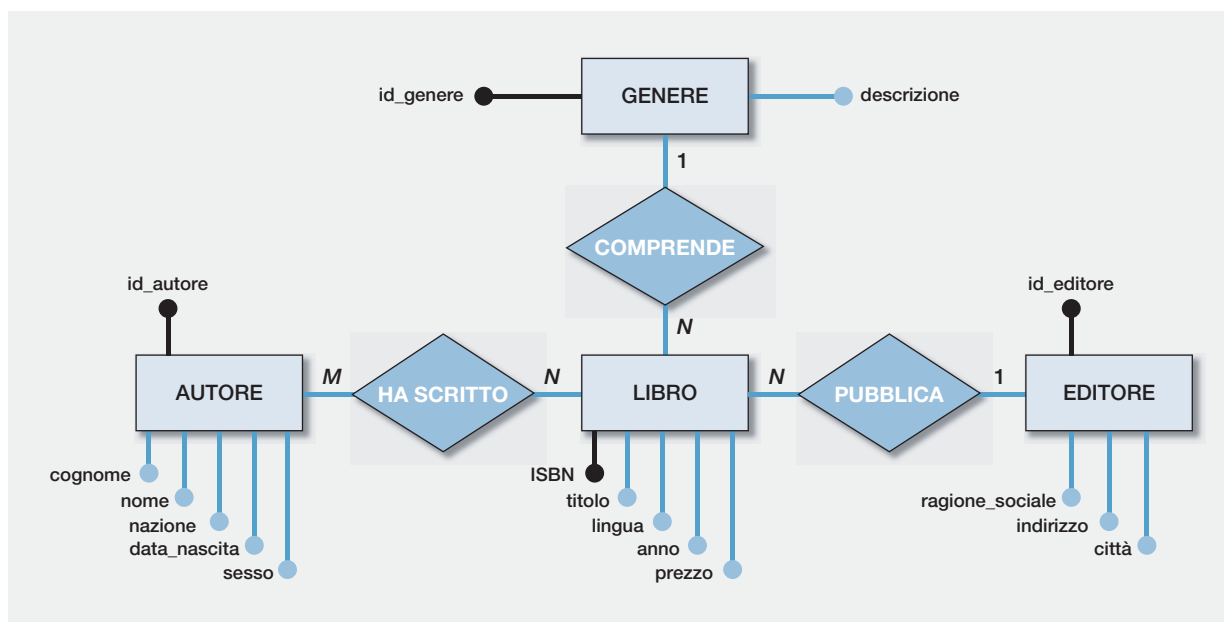


FIGURA 10

In modo propedeutico all'implementazione di basi di dati di tipo relazionale, una volta effettuata la progettazione concettuale utilizzando i diagrammi E/R, in cui possono essere presenti associazioni di cardinalità  $M:N$ , utilizziamo la rappresentazione grafica normalmente impiegata da progettisti e sviluppatori ricorrendo alla soluzione che prevede la scomposizione di tali associazioni in due associazioni di tipo  $1:N$  introducendo un'entità intermedia i cui attributi sono chiavi esterne che riferiscono le chiavi primarie delle entità originalmente associate.

#### ESEMPIO

Il diagramma E/R illustrato nell'esempio precedente presenta un'associazione  $N:M$ , dato che un autore può avere scritto più libri, ma anche uno stesso libro può essere stato scritto da più autori; la relativa rappresentazione grafica diventa quindi quella di FIGURA 11.

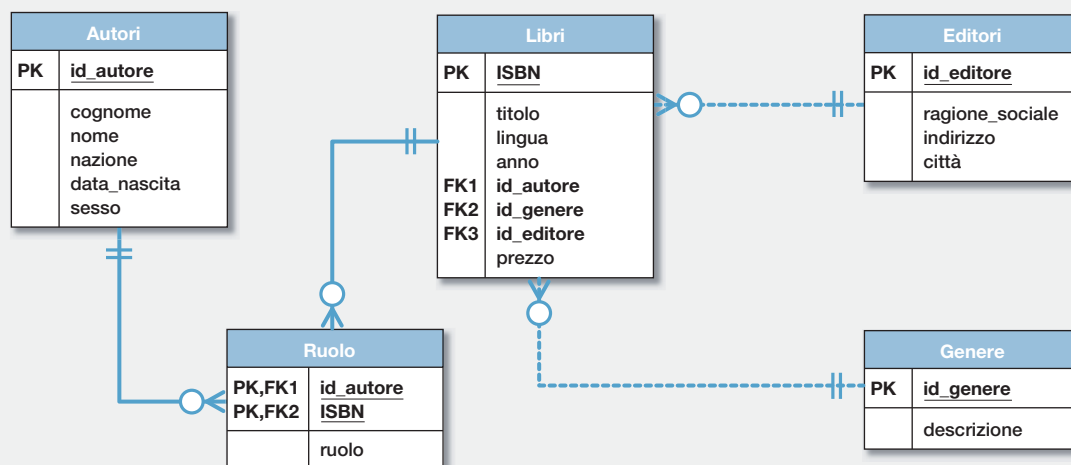


FIGURA 11

**OSSERVAZIONE** Allo scopo di trasformare l'associazione  $M:N$  tra le entità *Libri* e *Autori*, viene introdotta la nuova entità *Ruolo*, che rappresenta – oltre all'associazione tra le entità originali – anche il ruolo che uno specifico autore ha avuto nella realizzazione del libro (autore, coautore, curatore, revisore, ecc.): a questo scopo l'attributo *ruolo* è stato inserito nell'entità come proprietà caratterizzante l'associazione tra uno specifico autore e uno specifico libro.

Le chiavi primarie delle entità *Autori* (*id\_autore*) e *Libri* (*ISBN*) sono ciascuna chiave esterna nell'entità *Ruolo*, riferendo rispettivamente una singola istanza dell'entità *Autori* e una singola istanza dell'entità *Libri*: la loro combinazione forma la chiave primaria dell'entità *Ruolo* (aspetto graficamente evidenziato dal fatto che i connettori sono rappresentati da linee continue e non tratteggiate). In questo modo possono esistere più istanze che, pur riferendo lo stesso libro, riferiscono autori diversi, realizzando l'associazione desiderata (non è invece possibile avere più istanze che associano lo stesso libro allo stesso autore, situazione che violerebbe il vincolo di univocità della chiave primaria).

Anche se è buona norma ricondursi, quando possibile, a situazioni che comprendono esclusivamente associazioni binarie, esistono casi complessi in cui le associazioni sono multiple: un caso non troppo raro è quello relativo alle relazioni ternarie.

#### ESEMPIO

Il diagramma di FIGURA 12 deriva dal diagramma E/R di FIGURA 13, in cui:

- i vari uffici di un'azienda utilizzano prodotti acquistati da fornitori esterni;
- un fornitore può fornire diversi prodotti, ma uno stesso prodotto può essere fornito da fornitori diversi (per acquistarlo deve essere fatta una gara per decidere da chi acquistare la merce al prezzo più basso).



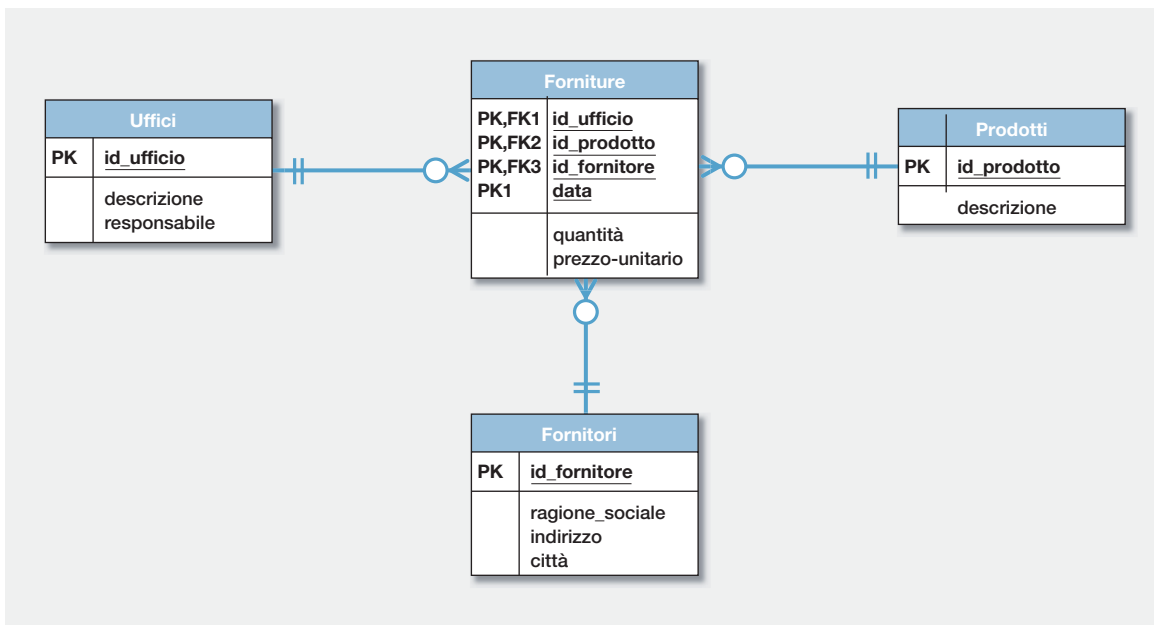


FIGURA 12

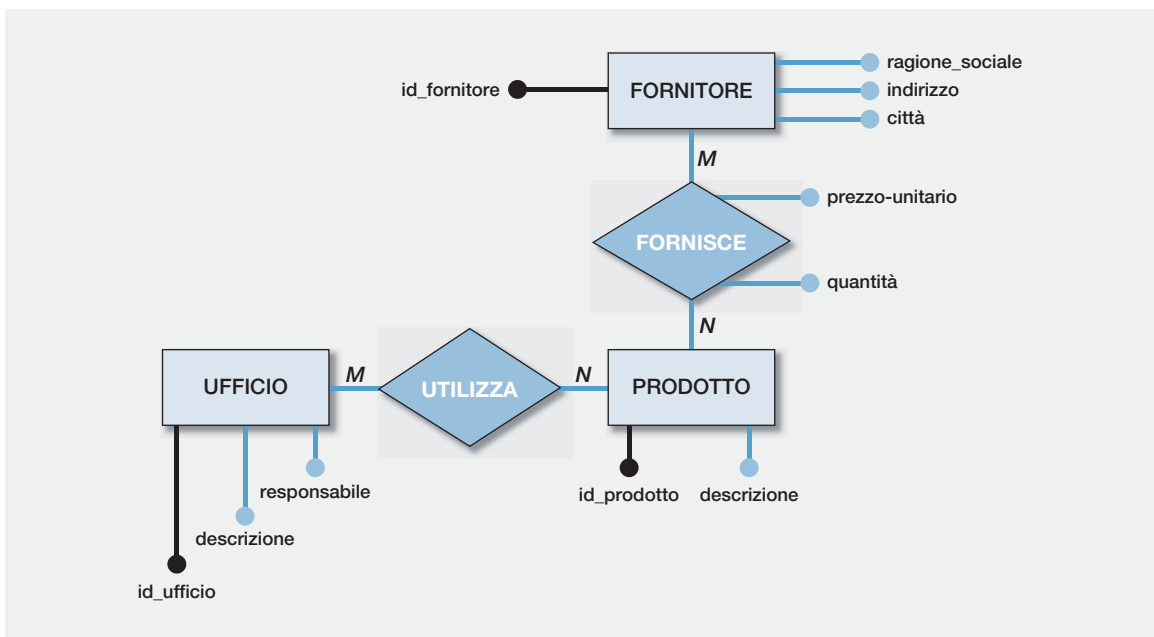


FIGURA 13

Dato che tra le entità *Uffici* e *Prodotti*, così come tra le entità *Prodotti* e *Fornitori*, esiste un'associazione di cardinalità  $M:N$ , l'entità fittizia *Forniture* realizza l'associazione ternaria tra *Uffici*, *Prodotti* e *Fornitori*. Si noti che la chiave primaria dell'entità *Forniture* comprende anche la data della specifica fornitura: è infatti possibile che uno stesso ufficio acquisti lo stesso prodotto dal medesimo fornitore in momenti diversi, in questo caso le date diverse differenziano le chiavi primarie delle due forniture.

È possibile associare un'entità con sé stessa (associazione ricorsiva), o associare tra loro le stesse entità con più corrispondenze distinte.

Il diagramma E/R di FIGURA 14 rappresenta la relazione esistente tra i vari impiegati di una certa azienda, che possono, o meno, essere colleghi, cioè lavorare nella stessa sede.

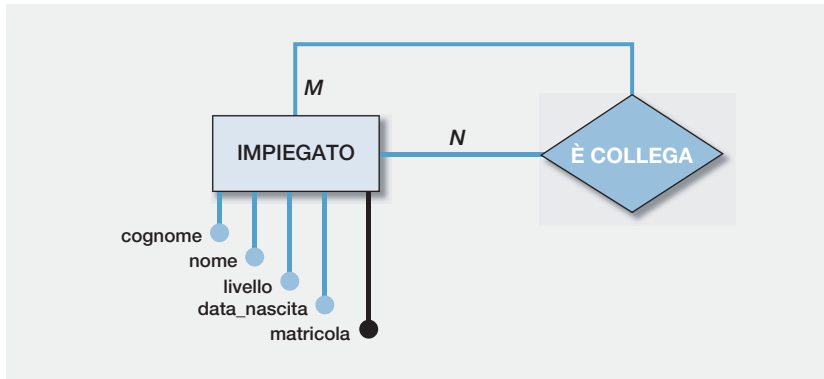


FIGURA 14

Il passaggio alla rappresentazione grafica della base di dati prevede l'inserimento di una nuova entità che implementi l'associazione di cardinalità  $M:N$  tra impiegati (ogni istanza dell'entità *Colleghi* rappresenta una coppia di impiegati colleghi tra loro) (FIGURA 15).

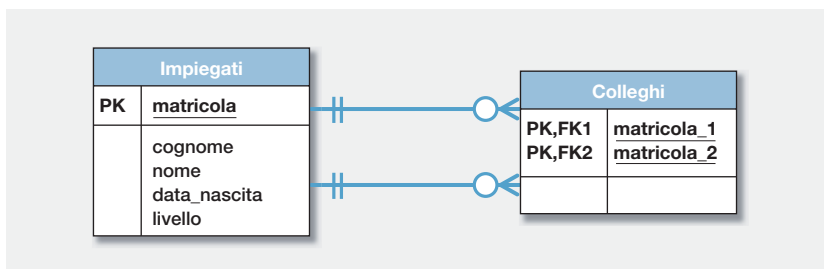


FIGURA 15

**OSSERVAZIONE** In particolare nei casi di realtà complesse da modellare, è importante allegare al diagramma E/R e/o alla rappresentazione grafica della base di dati un'appropriata documentazione avente lo scopo in generale di facilitarne l'interpretazione e, in particolare, di descrivere proprietà caratteristiche dei dati rappresentati che non possono essere espresse con il formalismo adottato.

## 2 Il modello dei dati relazionale

Il modello dei dati relazionale fu proposto da Edgar Codd nel 1970: con esso la gestione dei dati viene ricondotta all'unico concetto di «relazione», intesa in senso algebrico.

Molti DBMS sono basati sul modello dei dati relazionale; tra i più diffusi ricordiamo: Oracle DB, IBM DB2, Microsoft SQL-server e My-SQL recentemente acquisito da Oracle Corporation.

## Edgar Codd

Edgar Codd (1948-2003) è stato una personalità di primo piano dell'informatica teorica.

Alle fine degli anni '60 del secolo scorso, lavorando come ricercatore per IBM, ideò il modello relazionale per la gestione dei database le cui caratteristiche furono pubblicate nel 1970 in un articolo dal titolo «A relational model of data for large shared data banks».

IBM non sfruttò le idee di Codd per realizzare un sistema di gestione dei database almeno fino a quando la concorrenza cominciò a implementarle. È sulla base delle idee di Codd relative ai database relazionali, infatti, che Larry Ellison nel 1977 fondò la Oracle Corporation e iniziò la commercializzazione dell'omonimo DBMS.

Edgar Codd fu autore di molti articoli rilevanti nel campo dell'informatica teorica, ma il modello relazionale e la teoria generale della gestione dei dati hanno riscosso un tale successo nell'informatica applicata da rappresentare sicuramente il suo contributo più importante a questa scienza.

Il modello dei dati relazionale è un modello formale per la rappresentazione della struttura logica di una base di dati finalizzato al conseguimento di una buona indipendenza dei dati dalle procedure di elaborazione e alla semplificazione dei linguaggi impiegati per la loro definizione e manipolazione, basato su un approccio matematico rigoroso che ha come fondamenti teorici il concetto di relazione matematica, la teoria elementare degli insiemi e la logica dei predicati del primo ordine.

► Dati gli insiemi  $D_1, D_2, \dots, D_N$  (non necessariamente distinti) una **relazione**  $R$  su questi  $N$  insiemi è costituita da un insieme di ennuple  $(d_1, d_2, \dots, d_N)$  tali che l'elemento  $d_i$  appartiene all'insieme  $D_i$  per ogni indice  $i = 1, 2, \dots, N$ ; in altre parole  $R$  è un sottoinsieme del prodotto cartesiano  $D_1 \times D_2 \times \dots \times D_N$ .  
Gli insiemi  $D_1, D_2, \dots, D_N$ , il numero  $N$  e il numero delle ennuple che compongono la relazione  $R$  costituiscono rispettivamente i **domini**, il **grado** e la **cardinalità** di  $R$ .

### ESEMPIO

Quella che segue è una relazione relativa ad alcuni risultati di partite di calcio in cui si vede chiaramente che  $D_1$  e  $D_2$  sono domini definiti sulle stringhe di caratteri mentre  $D_3$  e  $D_4$  sono domini definiti su numeri interi. Ogni ennupla stabilisce un legame tra due squadre di calcio e il relativo numero di reti messe a segno in una partita:

Roma	Milan	2	1
Inter	Sampdoria	1	0
Cagliari	Udinese	0	2
Juventus	Torino	1	1
Napoli	Lazio	2	2

**OSSERVAZIONE** Una relazione di fatto è un insieme di ennuple e quindi:

- non è definito alcun ordinamento fra le sue ennuple (nella relazione precedente l'ordine delle ennuple non è significativo ai fini informativi degli incontri e delle reti segnate);
- le ennuple di una relazione sono tutte distinte una dall'altra perché in un insieme non possono esistere elementi uguali;
- ciascuna ennupla al proprio interno è ordinata in quanto l' $i$ -esimo valore appartiene all' $i$ -esimo dominio (se scambiassimo i domini della relazione vista in precedenza modificheremmo il significato della relazione stessa).

Relativamente all'ultimo punto dell'osservazione appena vista possiamo dire che l'ordinamento dei domini di una relazione è una caratteristica matematica della stessa, ma non è soddisfacente rispetto alla possibilità di

organizzare e gestire dei dati. Per questa ragione, generalmente, si tende a privilegiare notazioni che non siano posizionali (come nel caso degli array) ma che permettano di fare riferimento ai dati tramite nomi simbolici (come nel caso dei record).

**ESEMPIO**

Nel seguito

Squadra_Casa	Squadra_Ospite	Reti_Casa	Reti_Ospiti
Roma	Milan	2	1
Inter	Sampdoria	1	0
Cagliari	Udinese	0	2
Juventus	Torino	1	1
Napoli	Lazio	2	2

le intestazioni delle colonne (o attributi della relazione) rendono di fatto ininfluenti anche l'ordine delle colonne, essendo le stesse «etichettate» in maniera univoca.

In generale possiamo parlare di **relazione con attributi** facendo riferimento a una relazione  $R$  rappresentata come una **tabella** bidimensionale avente le seguenti proprietà:

- le colonne sono identificate da attributi distinti e ognuna di esse corrisponde a un dominio della relazione  $R$  (l'insieme degli attributi di una tale relazione  $R$  è indicato con il termine **schema** e rappresenta l'aspetto intensionale dei dati);
- ogni riga corrispondente a una ennupla: non vi possono essere due o più righe uguali (il contenuto delle righe della tabella rappresenta l'aspetto estensionale dei dati);
- l'ordine delle righe e delle colonne non è significativo.

**OSSERVAZIONE** Essendo una relazione con attributi rappresentata come una tabella, in letteratura viene spesso usato il termine «t-upla» indicando con esso una ennupla di tale tabella.

Una ennupla deve essere individuata univocamente da una **chiave**, cioè dal valore assunto da un insieme di attributi dello schema della relazione a cui essa appartiene.

► In ogni tabella esiste almeno un sottoinsieme  $K$  degli attributi, definito **superchiave**, tale che per ogni possibile coppia di righe distinte il valore assunto dalla superchiave è diverso.

**OSSERVAZIONE** Per ogni tabella non sono ammessi valori nulli per attributi che concorrono alla definizione di una chiave.

- In ogni tabella esiste almeno un sottoinsieme  $K$  di attributi, definito **chiave candidata** tale che:
  - $K$  è una superchiave;
  - nessun elemento di  $K$  può essere eliminato senza perdere questa proprietà.
- Un attributo di una tabella è definito **attributo primo** se esso è membro di almeno una chiave candidata. Un attributo è invece definito **attributo non primo** se non è membro di alcuna chiave candidata.
- Tra tutte le possibili chiavi candidate di una tabella ne viene scelta una definita **chiave primaria** (normalmente ne viene scelta una formata da un numero minimale di attributi).

**OSSERVAZIONE** Dalla definizione di relazione segue che non vi possano essere due o più ennuple uguali, quindi in teoria per ogni tabella esiste la chiave primaria che, come caso limite, è formata da tutti i suoi attributi.

Normalmente lo schema di una relazione  $R$  con attributi  $(a_1, a_2, \dots, a_N)$  viene rappresentato con la notazione  $R(a_1, a_2, \dots, a_N)$ , sottolineando gli attributi che ne costituiscono la chiave primaria.

#### ESEMPIO

Lo schema della relazione *Autori*, corrispondente all'omonima entità dell'esempio introduttivo è il seguente:

$\text{Autori}(\text{id\_autore}, \text{cognome}, \text{nome}, \text{nazione}, \text{data\_nascita}, \text{sex})$

La seguente tabella ne rappresenta una possibile configurazione (istanza):

id_autore	cognome	nome	nazione	data_nascita	sex
A001	Calvino	Italo	Italia	15/10/1923	M
A012	Woolf	Virginia	Inghilterra	25/01/1882	F
A115	Marquez	Garcia	Colombia	06/03/1927	M

- Definiamo **chiave esterna** un insieme di attributi di una tabella  $T_1$  che assumono valori nel dominio di una chiave primaria di una tabella  $T_2$  e che generalmente non costituiscono una chiave primaria per  $T_1$ .

**OSSERVAZIONE** La chiave primaria e la chiave esterna esprimono due importanti vincoli di integrità:

- **chiave primaria (univocità)**: data una tabella **non** è possibile inserire in essa due righe uguali;
- **chiave esterna (integrità referenziale)**: date due tabelle  $T_1$  e  $T_2$  dove la chiave primaria  $K_1$  di  $T_1$  corrisponde alla chiave esterna  $K_2$  di  $T_2$ , è possibile inserire in  $T_2$  una riga **solo se** il valore assunto dagli attributi di  $K_2$  è uguale a un valore assunto dagli attributi di  $K_1$  in  $T_1$  (che, se esiste, è unico).

Lo schema della relazione *Libri*, corrispondente all'omonima entità dell'esempio di apertura del capitolo è il seguente:

**Libri**(ISBN, titolo, lingua, anno, prezzo, id\_genere, id\_editore)

Nello schema l'attributo *id\_editore* è una chiave esterna che riferisce la chiave primaria della relazione *Editori*:

**Editori**(id\_editore, ragione\_sociale, indirizzo, città)

Alcune possibili configurazioni delle tabelle che estendono questi schemi di relazione sono le seguenti:

ISBN	titolo	lingua	anno	prezzo	id_genere	id_editore
123-456-789	Pinocchio	Italiano	2001	15,00€	G01	E100
987-654-321	The Hobbit	Inglese	2010	20,00€	G01	E101

id_editore	ragione_sociale	indirizzo	città
E099	Zanichelli	Via Irnerio, 34	Bologna
E100	Mondadori	Via Bianca di Savoia, 12	Milano
E101	RCS	Via Rizzoli, 8	Milano

Il valore della chiave esterna del libro «Pinocchio» («E100») corrisponde al valore della chiave primaria dell'editore «Mondadori» che ha pubblicato il libro, mentre il valore della chiave esterna del libro «The Hobbit» («E101») corrisponde al valore della chiave primaria dell'editore «Feltrinelli» che ha pubblicato il libro. Dovendo aggiungere un nuovo libro alla tabella *Libri*, questo potrà essere pubblicato esclusivamente da uno dei tre editori presenti nella tabella *Editori*, perché il contenuto della chiave esterna (*id\_editore*) deve coincidere con uno dei contenuti della chiave primaria della tabella *Editori* (*id\_editore*); in caso contrario è necessario aggiungere prima il nuovo editore nella tabella *Editori*.

**OSSERVAZIONE** Come chiaramente illustrato nell'esempio precedente, il ruolo pratico di una chiave esterna è quello di contenere, per ogni riga della tabella, un riferimento ai dati relativi contenuti in una diversa tabella. Le motivazioni che portano i progettisti dei database a dividere i dati in più tabelle correlate sono approfonditi nel seguito di questo capitolo.

- Una **base di dati relazionale** è costituita da un insieme finito di tabelle, le cui righe variano nel tempo, tante quanti sono gli schemi delle relazioni che ne definiscono la struttura logica.

Nel modello relazionale, quindi, i concetti di entità, proprietà e associazioni sono sostituiti dall'unico concetto di tabella, cioè un insieme di valori degli attributi che rappresentano sia le proprietà delle entità sia le associazioni.

**OSSERVAZIONE** Alle entità individuate nella costruzione del modello concettuale E/R di uno scenario corrispondono nel modello logico specifiche tabelle.

Per rappresentare un'associazione binaria di cardinalità 1:N tra due entità è sufficiente estendere lo schema di relazione della tabella corrispondente al codominio con una chiave esterna che assume i valori della chiave primaria nel dominio dell'associazione stessa.

Un'associazione binaria di cardinalità M:N può essere invece rappresentata introducendo una nuova tabella intermedia la cui chiave primaria è ottenuta combinando le chiavi primarie del dominio e del codominio dell'associazione stessa.

Lo schema logico completo della base di dati relativa alla biblioteca dell'esempio di apertura del capitolo è costituito dai seguenti schemi di relazione:

**Libri**(ISBN, titolo, lingua, anno, prezzo, id\_genere, id\_editore)

**Autori**(id\_autore, cognome, nome, nazione, data\_nascita, sesso)

**Editore**(id\_editore, ragione\_sociale, indirizzo, citta)

**Genere**(id\_genere, descrizione)

**Ruolo**(id\_autore, ISBN, ruolo)

Le tabelle che seguono esemplificano un possibile stato in un determinato istante temporale della base di dati:

ISBN	titolo	lingua	anno	prezzo	id_genere	id_editore
123-456-789	Pinocchio	Italiano	1990	10,00€	G01	E100
987-654-321	The Hobbit	Inglese	1999	10,00€	G01	E101
012-345-678	The Lord of Rings	Inglese	2001	25,00€	G01	E101
876-543-210	Informatica	Italiano	2012	20,00€	G99	E099

id_autore	cognome	nome	nazione	data_nascita	sesso
A001	Tolkien	John	Inghilterra	03/01/1892	M
A002	Collodi	Carlo	Italia	24/11/1826	M
A003	Meini	Giorgio	Italia	23/03/1965	M
A004	Formichi	Fiorenzo	Italia	06/10/1954	M

id_editore	ragione_sociale	indirizzo	citta
E099	Zanichelli	Via Irnerio, 34	Bologna
E100	Mondadori	Via Bianca di Savoia, 12	Milano
E101	RCS	Via Rizzoli, 8	Milano

id_genere	descrizione
G01	Fantasy
G99	Tecnologia
G90	Scienza

id_autore	ISBN	ruolo
A002	123-456-789	autore
A001	987-654-321	autore
A001	012-345-678	autore
A004	876-543-210	autore
A003	876-543-210	coautore



L'analisi del contenuto della tabella relativa alla tabella *Ruolo* – introdotta per realizzare nel modello logico l'associazione di cardinalità *M:N* del modello concettuale – consente di verificare che:

- il libro «Pinocchio» (ISBN 123-456-789) ha come autore «Carlo Collodi» (*id\_autore* «A002»);
- il libro «The Hobbit» (ISBN 987-654-321) ha come autore «John Tolkien» (*id\_autore* «A001»);
- il libro «The Lord of Rings» (ISBN 012-345-678) ha come autore «John Tolkien» (*id\_autore* «A001»);
- il libro «Informatica» (ISBN 876-543-210) ha come autore «Fiorenzo Formichi» (*id\_autore* «A004») e come co-autore «Giorgio Meini» (*id\_autore* «A003»).

**OSSERVAZIONE** L'elenco degli schemi delle relazioni rappresenta il modello logico di una base di dati relazionale. Nella fase successiva dell'implementazione fisica sarà necessario definire per ogni attributo il tipo di dato a cui esso fa riferimento: stringa di caratteri, valore numerico (intero, decimale, ...), data, orario, ... con eventuale indicazione del numero di caratteri o cifre impiegate per la rappresentazione e vincoli sull'ammissibilità dei valori. Ulteriori specificazioni saranno relative alla definizione di eventuali indici per gli attributi in base ai quali si intendono velocizzare le operazioni di ricerca.

## 2.1 Lo standard relazionale

Dopo la definizione del modello relazionale, Edgar Codd fissò in un famoso articolo del 1985 («Is your DBMS really relational?») tredici regole (numerate da 0 a 12) per stabilire quando un DBMS corrisponda alla sua proposta (i DBMS di tipo relazionale sono spesso identificati con l'acronimo **RDBMS**, *Relational Data Base Management System*).

- 0 Un DBMS è di tipo relazionale se esso impiega esclusivamente le proprie funzionalità relazionali per la gestione delle basi di dati.
- 1 In una base di dati relazionale tutte le informazioni sono rappresentate a livello logico esclusivamente mediante valori inseriti nelle tabelle.
- 2 Ogni valore della base di dati deve essere logicamente accessibile attraverso una combinazione del nome di una tabella, del valore della chiave primaria che identifica la riga e del nome della colonna.
- 3 Deve essere previsto, indipendentemente dal tipo di dato, un indicatore nullo per rappresentare a livello logico il concetto di «informazione mancante» (distinto dal carattere vuoto, da una stringa di caratteri bianchi, dal valore numerico zero e da ogni altro numero); il DBMS deve permettere di gestire in modo sistematico l'indicatore nullo.
- 4 La descrizione della struttura della base di dati deve essere rappresentata a livello logico mediante tabelle così come i dati ordinari, in modo tale che gli utenti possano applicare lo stesso linguaggio relazionale per la sua manipolazione.
- 5 Deve essere previsto almeno un linguaggio relazionale che abbia le seguenti proprietà:
  - sintassi lineare (comandi composti in via di principio da una sola riga di testo);

- utilizzabile sia in modalità autonoma sia ospitato in un linguaggio di programmazione;
  - supporto per la definizione della struttura dei dati (compresa la definizione di viste dei dati specifiche per singoli utenti), per la manipolazione dei dati (modifica e recupero), per la gestione della sicurezza e dell'integrità dei dati e per l'esecuzione di transazioni.
- 6 Deve essere sempre possibile modificare le viste di dati che risultano teoricamente modificabili.
  - 7 Deve essere possibile manipolare una tabella base o risultato di un'operazione su altre tabelle come un singolo operando utilizzabile in operazioni di inserimento, aggiornamento e cancellazione dei dati che coinvolgano più colonne e più tabelle contemporaneamente.
  - 8 I programmi applicativi e le operazioni effettuate dall'utente non devono essere alterate in caso di modifica della struttura fisica di memorizzazione dei dati o dei metodi di accesso (indipendenza fisica dei dati).
  - 9 I programmi applicativi non devono essere alterati in caso di modifiche al livello logico della base di dati (indipendenza logica dei dati).
  - 10 Deve essere possibile definire i vincoli di integrità per una base di dati al livello della sua struttura logica, indipendentemente dai programmi applicativi.
  - 11 La ridistribuzione fisica dei dati di una base di dati su elaboratori diversi non deve influenzarne la struttura logica e deve essere trasparente per i programmi applicativi.
  - 12 I vincoli di consistenza e integrità dei dati non devono poter essere aggirati da un eventuale linguaggio a basso livello operante su una singola riga alla volta.

**OSSERVAZIONE** Codd scrisse questi «comandamenti» agli albori dell'industria dei DBMS relazionali per indirizzare le scelte dei progettisti che realizzavano nuovi sistemi DBMS: oggi – in particolare con l'adozione generalizzata del linguaggio SQL – tali regole sono quasi universalmente rispettate e implementate.

### 3 Progettazione e normalizzazione di una base di dati relazionale

La limitatezza di concetti alla base del modello relazionale per descrivere la realtà può creare dei problemi in fase di progettazione di una base di dati di tipo relazionale. Allo scopo di eliminare eventuali anomalie sono stati proposti dei procedimenti di «normalizzazione».

La relazione *Incarichi*

*Incarichi*(impiegato, livello, progetto, budget, anno, ruolo, costo\_orario)

di cui la seguente tabella è un possibile esempio



impiegato	livello	progetto	budget	anno	ruolo	costo_orario
Rossi	B1	Alfa	4	2011	tecnico	10
Bianchi	C2	Beta	6	2012	progettista	25
Bianchi	C2	Delta	6	2011	progettista	25
Neri	D2	Delta	6	2011	responsabile	40
Neri	D2	Beta	6	2012	consulente	30
Neri	D2	Alfa	4	2011	consulente	30
Verdi	D1	Alfa	4	2011	responsabile	40
Verdi	D1	Delta	6	2011	consulente	30
Grigi	C4	Delta	6	2011	tecnico	10
Grigi	C4	Beta	6	2012	responsabile	40

non risulta ben progettata in quanto, operando con operazioni di inserimento, cancellazione e aggiornamento, si possono incontrare i seguenti problemi.

- **Inserimento:** non è possibile inserire i dati di un impiegato (*impiegato, livello*) se non è coinvolto in almeno un progetto; non è possibile inserire un nuovo progetto con i relativi attributi (*progetto, budget, anno*) se non coinvolge almeno un impiegato; questi vincoli derivano dal fatto che gli attributi impiegato e progetto sono parte della chiave primaria della relazione/tabella.
- **Cancellazione:** la cancellazione della riga identificata dal valore della chiave primaria (Neri, Alfa) è diversa dalla cancellazione della riga identificata dal valore della chiave primaria (Rossi, Alfa): nel primo caso si conservano sia i dati del progetto Alfa sia quelli dell'impiegato Neri, mentre nel secondo caso i dati dell'impiegato Rossi vanno perduti.
- **Aggiornamento:** l'aggiornamento del livello di inquadramento dell'impiegato Rossi implica la modifica di una sola riga, mentre quello dell'impiegato Neri tre righe.

Le anomalie evidenziate nell'esempio precedente sono una diretta conseguenza della ridondanza dei dati, cioè del fatto che lo stesso dato è presente nella tabella replicato più volte.

La metodologia per eliminare questo tipo di anomalie si articola in tre fasi successive:

- riduzione in **prima forma normale** (*1<sup>st</sup> Normal Form, 1NF*);
- riduzione in **seconda forma normale** (*2<sup>nd</sup> Normal Form, 2NF*);
- riduzione in **terza forma normale** (*3<sup>rd</sup> Normal Form, 3NF*).

### 3.1 Normalizzazione

La normalizzazione è un procedimento che, allo scopo di evitare le anomalie descritte, porta a decomporre una tabella in più tabelle. Questo processo di decomposizione deve essere applicato in modo che:

- non comporti perdita informativa in modo da garantire la ricostruzione delle informazioni originarie;
- conservi le dipendenze, ovvero il mantenimento dei vincoli di integrità originari.

Con l'espressione **decomposizione senza perdita** si intende che, suddividendo una tabella  $T$  in due tabelle  $T_1$  e  $T_2$ , oltre a conservare le dipendenze funzionali tra i dati, si verifica anche la condizione  $T = T_1 \text{ join } T_2$ ; il significato puntuale di questa espressione sarà chiarito nel seguito, per ora è sufficiente osservare che deve esistere un operatore (*join*) che, a partire dalle due tabelle  $T_1$  e  $T_2$ , consente di ottenere la tabella  $T$  originaria.

**OSSERVAZIONE** A prima vista risulta strano il fatto di applicare un procedimento di decomposizione delle tabelle per normalizzarle, quando in fase di ricerca dei dati è spesso necessario utilizzare operatori che applichino il processo inverso di ricomposizione.

La decomposizione ha come scopo la limitazione della ridondanza dei dati al fine di garantire una corretta esecuzione delle operazioni di modifica dei dati di una tabella (inserimento, aggiornamento, cancellazione): il prezzo da pagare per garantire l'integrità dei dati è quello di dover ricomporre le tabelle mediante specifici operatori al momento in cui si effettuano operazioni di ricerca dei dati.

**TEOREMA** Sia data una tabella  $T(X)$ , con  $X$  insieme degli attributi di  $T$ , e due sottoinsiemi  $A$  e  $B$  di  $X$  tali che  $A \cup B = X$ ; siano date, inoltre, due tabelle  $T_1$  e  $T_2$  rispettivamente con insiemi di attributi  $A$  e  $B$ . Condizione sufficiente affinché la decomposizione di  $X$  su  $A$  e  $B$  sia «senza perdita» è che l'insieme  $C = A \cap B$  sia una superchiave per  $T_1(A)$  o  $T_2(B)$ .

#### ESEMPIO

A partire dallo schema di relazione dell'esempio precedente è possibile operarne la suddivisione nei due seguenti schemi di relazione:

$\text{Incarichi\_1}(\text{impiegato}, \text{livello}, \text{progetto}, \text{costo\_orario})$ ;

$\text{Incarichi\_2}(\text{impiegato}, \text{progetto}, \text{budget}, \text{anno}, \text{ruolo})$ .

La relativa tabella viene quindi scomposta nelle due seguenti tabelle:

impiegato	livello	progetto	costo_orario
Rossi	B1	Alfa	10
Bianchi	C2	Beta	25
Bianchi	C2	Delta	25
Neri	D2	Delta	40
Neri	D2	Beta	30
Neri	D2	Alfa	30
Verdi	D1	Alfa	40
Verdi	D1	Delta	30
Grigi	C4	Delta	10
Grigi	C4	Beta	40

<b>impiegato</b>	<b>progetto</b>	<b>budget</b>	<b>anno</b>	<b>ruolo</b>
Rossi	Alfa	4	2011	tecnico
Bianchi	Beta	6	2012	progettista
Bianchi	Delta	6	2011	progettista
Neri	Delta	6	2011	responsabile
Neri	Beta	6	2012	consulente
Neri	Alfa	4	2011	consulente
Verdi	Alfa	4	2011	responsabile
Verdi	Delta	6	2011	consulente
Grigi	Delta	6	2011	tecnico
Grigi	Beta	6	2012	responsabile

L'intersezione tra gli attributi dello schema della relazione *Incarichi\_1* e quelli dello schema della relazione *Incarichi\_2* è costituito dagli attributi (impiegato, progetto) che sono in questo caso chiave per entrambe le relazioni. Non è in effetti difficile immaginare di ricomporre la tabella originale a partire dalle due tabelle scomposte facendo corrispondere i valori degli attributi *impiegato* e *progetto*.

Sempre a partire dallo schema di relazione dell'esempio precedente è anche possibile operarne la suddivisione nei due schemi seguenti:

*Incarichi\_1*(impiegato, livello, progetto, costo\_orario);

*Incarichi\_2*(progetto, budget, anno, ruolo).

In questo caso la relativa tabella viene quindi scomposta nelle due seguenti tabelle:

<b>impiegato</b>	<b>livello</b>	<b>progetto</b>	<b>costo_orario</b>
Rossi	B1	Alfa	10
Bianchi	C2	Beta	25
Bianchi	C2	Delta	25
Neri	D2	Delta	40
Neri	D2	Beta	30
Neri	D2	Alfa	30
Verdi	D1	Alfa	40
Verdi	D1	Delta	30
Grigi	C4	Delta	10
Grigi	C4	Beta	40

<b>progetto</b>	<b>budget</b>	<b>anno</b>	<b>ruolo</b>
Alfa	4	2011	tecnico
Beta	6	2012	progettista
Delta	6	2011	progettista
Delta	6	2011	responsabile
Beta	6	2012	consulente
Alfa	4	2011	consulente
Alfa	4	2011	responsabile
Delta	6	2011	consulente
Delta	6	2011	tecnico
Beta	6	2012	responsabile



L'intersezione tra gli attributi dello schema della relazione *Incarichi\_1* e quelli dello schema della relazione *Incarichi\_2* è in questo caso costituito dal solo attributo *progetto*, che non rappresenta una chiave in nessuna delle due tabelle. In effetti è impossibile ricomporre la tabella originale in quanto l'associazione tra i valori dell'attributo nelle due tabelle non è univoca.

- Una tabella è in **prima forma normale (1NF)** se tutti i suoi attributi hanno domini atomici (cioè sono valori elementari non ulteriormente scomponibili) ed esiste per essa una chiave primaria.

#### ESEMPIO

La tabella che estende il seguente schema di relazione

**Persona**(nominativo, indirizzo, telefono)

non è in prima forma normale, in quanto non è stata individuata una chiave primaria e gli attributi *nominativo* e *indirizzo* non sono atomici; infatti il primo può essere ulteriormente scomposto in (*cognome*, *nome*), il secondo in (*indirizzo*, *CAP*, *città*, *nazione*). In base a queste osservazioni è possibile definire il seguente schema di relazione la cui tabella è in prima forma normale:

**Persona**(codice\_fiscale, cognome, nome, indirizzo, CAP, città, nazione, telefono)

Per l'analisi della seconda e della terza forma normale è fondamentale il concetto di **dipendenza funzionale** tra insiemi di attributi di una tabella.

- Sia  $X = \{x_1, x_2, \dots, x_N\}$  un insieme di attributi di una tabella  $T$  e  $Y$  un attributo di  $T$ . Si dice che  $Y$  **dipende funzionalmente** da  $X$  (o che  $X$  determina  $Y$ :  $X \rightarrow Y$ ) se e solo se, per ogni possibile configurazione del contenuto delle righe di  $T$ , i valori degli attributi di  $X$  determinano univocamente il valore di  $Y$ .

**OSSERVAZIONE** La definizione precedente afferma che, se due o più righe hanno gli stessi valori per gli attributi  $x_1, x_2, \dots, x_N$ , allora devono necessariamente avere anche lo stesso valore dell'attributo  $Y$ ; in altre parole l'insieme di attributi  $X$  rappresenta una chiave per  $Y$ .

- Una dipendenza funzionale è detta **parziale** se  $Y$  dipende da un sottoinsieme  $X'$  di  $X$ , mentre si ha una dipendenza funzionale **completa** se  $Y$  dipende da tutti gli elementi di  $X$ .

- Data una tabella  $T$  e due insiemi di attributi  $X$  e  $Y$  non vuoti di  $T$ , per cui si ha  $Y \subseteq X$ , si ha che ogni possibile configurazione del contenuto delle righe di  $T$  soddisfa la dipendenza funzionale  $X \rightarrow Y$ : in questo caso si parla di **dipendenza funzionale banale**.

Le dipendenze funzionali non sono sempre immediatamente individuabili analizzando i dati contenuti in una tabella: per determinare le dipendenze funzionali occorre analizzare attentamente la realtà descritta dalla base di dati.

#### ESEMPIO

La tabella *Incarichi* degli esempi precedenti presenta le seguenti dipendenze funzionali:

- *impiegato* → *livello* (ogni impiegato ha un proprio livello di inquadramento);
- *progetto* → *budget* e *progetto* → *anno* (ogni progetto ha un unico budget e un unico anno di riferimento);
- (*impiegato*, *progetto*) → *ruolo* (a ogni impiegato per un certo progetto è assegnato uno specifico ruolo);
- *ruolo* → *costo\_orario* (per ogni ruolo è previsto un determinato costo orario).

► Una tabella *T* è in **seconda forma normale (2NF)** se è in 1NF e non esistono, tra i possibili insiemi di attributi, dipendenze funzionali parziali, ma si hanno solo dipendenze funzionali complete: ogni attributo *A* non primo di *T* dipende funzionalmente in modo completo dalla chiave primaria di *T*.

#### ESEMPIO

Procediamo ora con la normalizzazione della tabella *Incarichi* degli esempi precedenti. Essa è già 1NF perché ha una chiave primaria definita e i domini dei valori di tutti gli attributi sono atomici e quindi non ulteriormente scomponibili, ma non è in 2NF perché gli attributi *anno* e *budget* dipendono esclusivamente dall'attributo *progetto* e non dall'intera chiave primaria. Per effettuare la normalizzazione è necessario scomporre la tabella *Incarichi* in questo modo:

*Incarichi*(*impiegato*, *progetto*, *ruolo*, *costo\_orario*)

*Progetti*(*progetto*, *budget*, *anno*)

*Impiegati*(*impiegato*, *livello*)

A partire dalla tabella riportata negli esempi precedenti la scomposizione produce le seguenti tabelle:

<i>impiegato</i>	<i>progetto</i>	<i>ruolo</i>	<i>costo_orario</i>
Rossi	Alfa	tecnico	10
Bianchi	Beta	progettista	25
Bianchi	Delta	progettista	25
Neri	Delta	responsabile	40
Neri	Beta	consulente	30
Neri	Alfa	consulente	30
Verdi	Alfa	responsabile	40
Verdi	Delta	consulente	30
Grigi	Delta	tecnico	10
Grigi	Beta	responsabile	40

<i>progetto</i>	<i>budget</i>	<i>anno</i>
Alfa	4	2011
Beta	6	2012
Delta	6	2011

<i>impiegato</i>	<i>livello</i>
Rossi	B1
Bianchi	C2
Neri	D2
Verdi	D1
Grigi	C4



## 4NF e 5NF

Tra le tante proposte che hanno caratterizzato gli studi sulla normalizzazione delle basi di dati relazionali, sono emerse anche la quarta e la quinta forma normale. Si tratta di due ulteriori forme normali che raramente vengono utilizzate, perché a un loro incremento di rigore nell'eliminazione delle ridondanze corrisponde un forte degrado delle prestazioni. In queste forme, infatti, le operazioni per il recupero o la modifica dei dati richiedono molto tempo per la loro esecuzione.

**OSSERVAZIONE** Le tabelle in seconda forma normale possono ancora presentare delle ridondanze e, di conseguenza, delle anomalie. Questo accade quando esistono attributi non chiave che dipendono da altri attributi non chiave; nel nostro caso abbiamo:

- $ruolo \rightarrow costo\_orario$ ;
- $(progetto, impiegato) \rightarrow ruolo$ .

L'attributo *costo\_orario* dipende dalla chiave solo in modo transitivo tramite l'attributo *ruolo*.

- Una tabella *T* è in **terza forma normale (3NF)** quando è in 2NF (cioè non si hanno dipendenze funzionali parziali) e non esistono attributi non primi dipendenti transitivamente dalla chiave primaria.

**OSSERVAZIONE** La definizione precedente può essere così riformulata:

Una tabella *T* è in terza forma normale se, per ogni dipendenza funzionale non banale  $X \rightarrow Y$  che sussiste in *T*, si ha che *X* è una superchiave di *T*, oppure *Y* è un attributo primo di *T*.

### ESEMPIO

Volendo portare la tabella *Incarichi* in 3NF si ha la necessità di operare un'ulteriore scomposizione:

*Incarichi*(impiegato, progetto, ruolo)

*Progetti*(progetto, budget, anno)

*Impiegati*(impiegato, livello)

*Ruoli*(ruolo, costo\_orario)

A partire dalla tabella riportata negli esempi precedenti la scomposizione produce le seguenti tabelle:

impiegato	progetto	ruolo
Rossi	Alfa	tecnico
Bianchi	Beta	progettista
Bianchi	Delta	progettista
Neri	Delta	responsabile
Neri	Beta	consulente
Neri	Alfa	consulente
Verdi	Alfa	responsabile
Verdi	Delta	consulente
Grigi	Delta	tecnico
Grigi	Beta	responsabile

impiegato	livello
Rossi	B1
Bianchi	C2
Neri	D2
Verdi	D1
Grigi	C4

progetto	budget	anno
Alfa	4	2011
Beta	6	2012
Delta	6	2011

ruolo	costo_orario
tecnico	10
progettista	25
consulente	30
responsabile	40

## 3.2 Forma normale di Boyce-Codd

- Una tabella  $T$  è in **forma normale di Boyce-Codd** (*Boyce-Codd Normal Form*, **BCNF**) se e solo se è in 3NF e, per ogni dipendenza funzionale non banale  $X \rightarrow Y$  (dove  $X$  è un sottoinsieme degli attributi  $T$  e  $Y$  è un attributo di  $T$ )  $X$  è una superchiave per  $T$ .

**OSSERVAZIONE** La definizione di BCNF è simile a quella di 3NF: la sola differenza è che la condizione della 3NF che consente all'attributo  $Y$  di essere primo è in questo caso assente.

La forma normale di Boyce-Codd è stata proposta come forma normale più semplice della 3NF, ma in realtà è più restrittiva: ogni tabella in BCNF è anche in 3NF, ma – anche se in pratica la maggior parte delle tabelle in 3NF è anche in BCNF – questo non è necessariamente sempre vero.

**OSSERVAZIONE** Dato un insieme di tabelle, non è possibile garantirne la trasformazione in BCNF; in particolare il mancato raggiungimento di questo obiettivo è indice che la base dati è affetta da un'anomalia di cancellazione (è quindi possibile perdere dati a seguito di un'operazione di cancellazione).

### ESEMPIO

Data la seguente tabella *Insegna*

docente	materia	studente
Turing	Informatica	Rossi
Codd	Informatica	Neri
Madnick	Sistemi	Neri
Donovan	Elettronica	Neri
Turing	Informatica	Bianchi
Knuth	Sistemi	Bianchi
Wirth	Informatica	Verdi
Codd	Informatica	Grigi
Madnick	Sistemi	Rossi

sussistono le seguenti dipendenze:

- $(studente, materia) \rightarrow docente$
- $docente \rightarrow materia$

e la coppia di attributi  $(studente, materia)$  è una chiave candidata per la tabella.

La tabella *Insegna* è in 3NF perché:

- è in 2NF in quanto esistono solo dipendenze funzionali complete;
- non vi sono dipendenze funzionali transitive.

Essa però non è in BCNF, perché nella dipendenza funzionale  $b$ ,  $docente$  non è superchiave per la tabella *Insegna*.

La decomposizione di questa tabella non è immediata perché si possono ottenere le seguenti tre coppie di tabelle alternative:

1. (docente, studente) e (studente, materia):

docente	studente	studente	materia
Turing	Rossi	Rossi	Informatica
Codd	Neri	Neri	Informatica
Madnick	Neri	Neri	Sistemi
Donovan	Neri	Neri	Elettronica
Turing	Bianchi	Bianchi	Informatica
Knuth	Bianchi	Bianchi	Sistemi
Wirth	Verdi	Verdi	Informatica
Codd	Grigi	Grigi	Informatica
Madnick	Rossi	Rossi	Sistemi

2. (docente, materia) e (studente, materia):

docente	materia	studente	materia
Turing	Informatica	Rossi	Informatica
Codd	Informatica	Neri	Informatica
Madnick	Sistemi	Neri	Sistemi
Donovan	Elettronica	Neri	Elettronica
Knuth	Sistemi	Bianchi	Informatica
Wirth	Informatica	Bianchi	Sistemi
		Verdi	Informatica
		Grigi	Informatica
		Rossi	Sistemi

3. (docente, materia) e (docente, studente):

docente	materia	docente	studente
Turing	Informatica	Turing	Rossi
Codd	Informatica	Codd	Neri
Madnick	Sistemi	Madnick	Neri
Donovan	Elettronica	Donovan	Neri
Knuth	Sistemi	Turing	Bianchi
Wirth	Informatica	Knuth	Bianchi
		Wirth	Verdi
		Codd	Grigi
		Madnick	Rossi

In tutti e tre i casi si perde la dipendenza funzionale  $a$ , ma l'unica scomposizione corretta è la terza perché applicando un'operazione di composizione – cioè effettuando il prodotto cartesiano tra le righe delle coppie di tabelle sulla base di un elemento in comune (*join*) – non sono generate righe spurie, ma viene ricostruita correttamente la tabella originale:



1. la composizione avviene sulla base del valore dell'attributo *studente* (sono evidenziate le righe spurie):

docente	materia	studente
Madnick	Informatica	Rossi
Turing	Informatica	Rossi
Codd	Informatica	Neri
Donovan	Informatica	Neri
Madnick	Informatica	Neri
Codd	Sistemi	Neri
Donovan	Sistemi	Neri
Madnick	Sistemi	Neri
Codd	Elettronica	Neri
Donovan	Elettronica	Neri
Madnick	Elettronica	Neri
Knuth	Informatica	Bianchi
Turing	Informatica	Bianchi
Knuth	Sistemi	Bianchi
Turing	Sistemi	Bianchi
Wirth	Informatica	Verdi
Codd	Informatica	Grigi
Madnick	Sistemi	Rossi
Turing	Sistemi	Rossi

2. la composizione avviene sulla base del valore dell'attributo *materia* (sono evidenziate le righe spurie):

docente	materia	studente
Turing	Informatica	Grigi
Turing	Informatica	Verdi
Turing	Informatica	Bianchi
Turing	Informatica	Neri
Turing	Informatica	Rossi
Codd	Informatica	Grigi
Codd	Informatica	Verdi
Codd	Informatica	Bianchi
Codd	Informatica	Neri
Codd	Informatica	Rossi
Madnick	Sistemi	Rossi
Madnick	Sistemi	Bianchi
Madnick	Sistemi	Neri
Donovan	Elettronica	Neri
Knuth	Sistemi	Rossi
Knuth	Sistemi	Bianchi
Knuth	Sistemi	Neri
Wirth	Informatica	Grigi
Wirth	Informatica	Verdi
Wirth	Informatica	Bianchi
Wirth	Informatica	Neri
Wirth	Informatica	Rossi



3. la composizione avviene sulla base del valore dell'attributo *docente* (non ci sono righe spurie):

docente	materia	studente
Turing	Informatica	Rossi
Codd	Informatica	Neri
Madnick	Sistemi	Neri
Donovan	Elettronica	Neri
Turing	Informatica	Bianchi
Knuth	Sistemi	Bianchi
Wirth	Informatica	Verdi
Codd	Informatica	Grigi
Madnick	Sistemi	Rossi

**OSSERVAZIONE** Che l'unico risultato corretto nell'esempio precedente sia quello generato dalla scomposizione 3 è dato dal fatto che è l'unica scomposizione in cui l'attributo su cui opera la successiva composizione (*docente*) è chiave primaria per la prima tabella e chiave esterna per la seconda.

## 4 Esempi di progettazione di basi di dati relazionali

In questo paragrafo sono illustrati alcuni esempi di progetto concettuale (mediante diagramma E/R) e logico (utilizzando il formalismo grafico introdotto cui fanno riferimento i professionisti del settore software) di database relazionali che rappresentano scenari di situazioni realistiche già analizzate.

### 4.1 Registro informatico di navigazione

#### Analisi dello scenario

Una recente disposizione normativa prevede che ogni armatore mercantile mantenga un registro informatizzato delle proprie navi con le loro caratteristiche (codice, nome, stazza, lunghezza, anno di costruzione, potenza dei motori, ecc.) e di tutti i loro viaggi da un porto all'altro (ogni porto del mondo ha un proprio codice identificativo): di ogni viaggio devono essere registrati il porto e la data/ora di partenza e il porto e la data/ora di arrivo, oltre al peso totale del carico trasportato e all'elenco del personale imbarcato (di cui deve essere riportato il cognome, il nome, il luogo e la data di nascita e il ruolo a bordo; il personale può imbarcarsi su navi diverse in viaggi diversi).

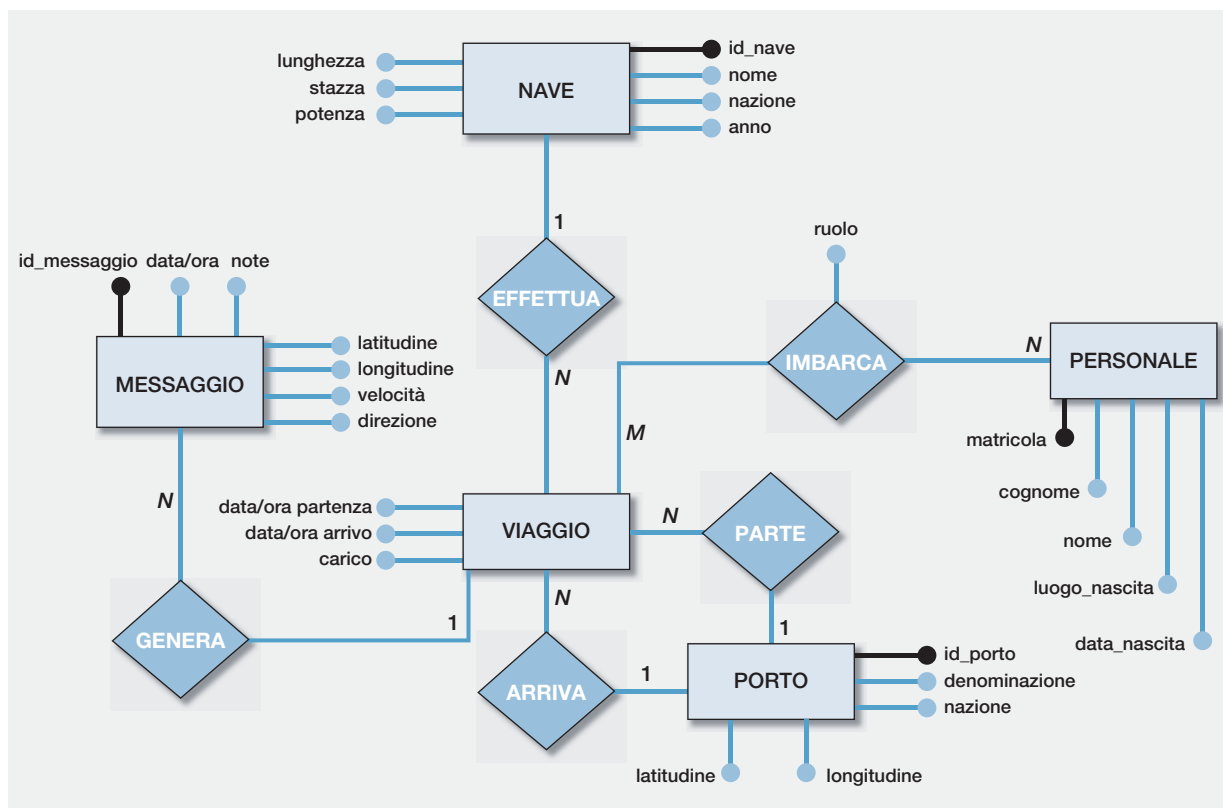


FIGURA 16

Nel corso del viaggio ogni nave invia al centro informatico dell'armatore messaggi aventi un codice univoco generato dal sistema di ricezione che riportano la posizione (latitudine e longitudine), la data, l'ora, la direzione e la velocità di navigazione più eventuali note stilate dal capitano.

## Progetto concettuale

Individuando le seguenti entità come rappresentative dello scenario

- Navi
- Porti
- Viaggi
- Personale
- Messaggi

il diagramma E/R di FIGURA 16 ne rappresenta gli attributi principali e le associazioni.

La cardinalità delle associazioni indicate nel diagramma E/R è motivata dalle seguenti considerazioni:

- *Nave/Viaggio* 1:N, in quanto una nave effettua numerosi viaggi nella sua vita operativa, ma uno specifico viaggio è effettuato da una sola nave;
- *Viaggio/Porto* 1:N, perché, anche se da un porto arrivano e partono molti viaggi, un viaggio ha un solo porto di partenza e un solo porto di arrivo;
- *Viaggio/Personale* M:N, in base al fatto che il singolo viaggio di una nave imbarca varie unità di personale, ma ogni unità di personale nella propria vita professionale effettua più viaggi;

- *Viaggio/Messaggio* 1:N, considerando il fatto che nel corso di un viaggio sono generati numerosi messaggi, ma che uno specifico messaggio si riferisce a un solo viaggio.

**OSSERVAZIONE** Per l'entità *Viaggio* non è stata indicata nel diagramma E/R nessuna chiave primaria in quanto non emerge una scelta suggerita in fase di analisi. In questi casi è sempre possibile aggiungere un attributo identificativo: per esempio, in questo caso, un codice univoco per ogni viaggio. In alternativa è possibile prendere in considerazione un insieme di attributi dell'entità i cui valori non possono ripetersi, cosa che è consigliato fare in fase di progettazione logica, dopo avere definito le necessarie chiavi esterne per ogni tabella.

## Progetto logico

Dato che ogni entità individuata nel progetto concettuale diviene una tabella, lo schema di FIGURA 17 rappresenta graficamente il database che deriva dal precedente diagramma E/R.

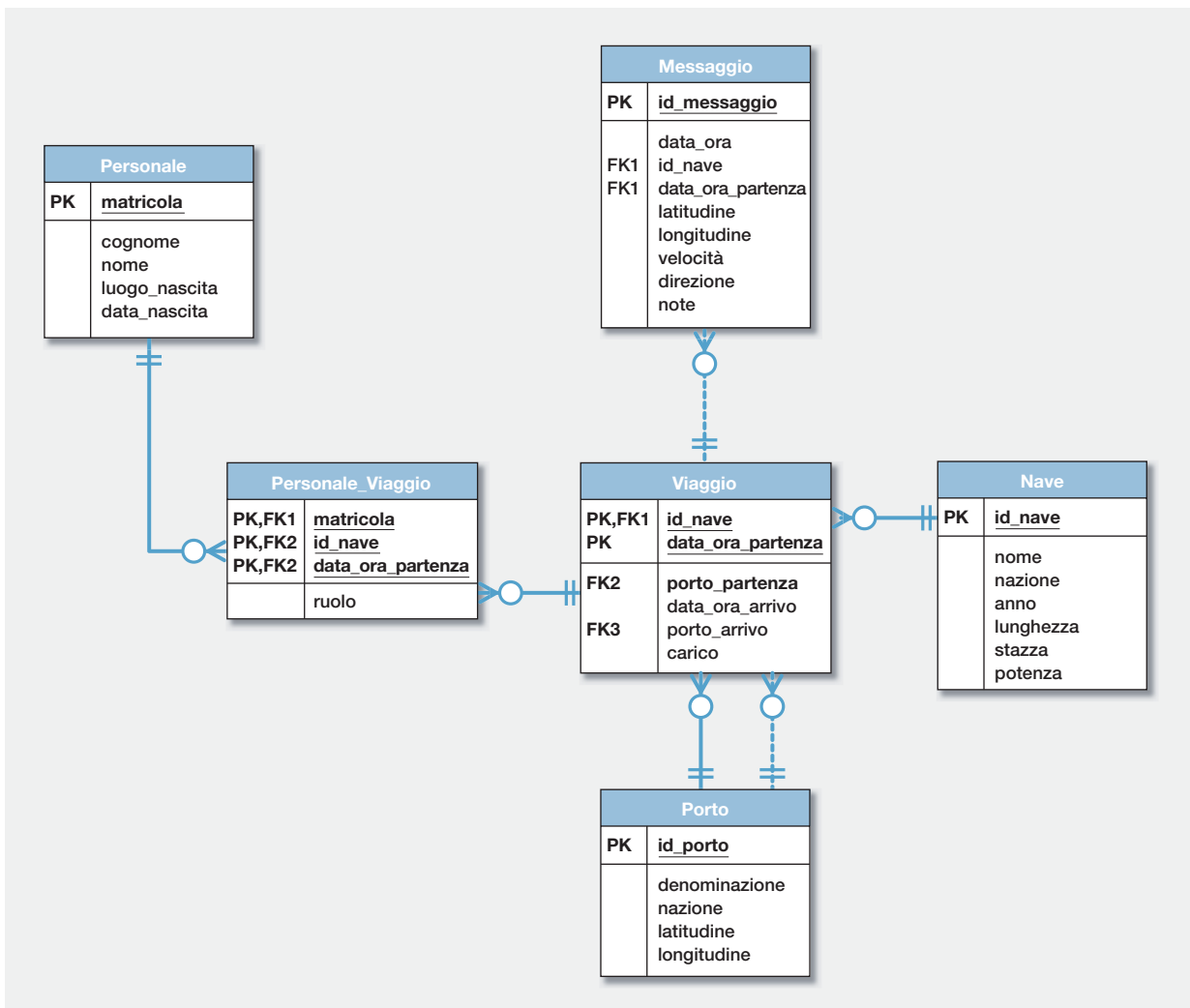


FIGURA 17



**OSSERVAZIONE** La chiave primaria della tabella *Viaggio* è costituita dagli attributi *id\_nave* (chiave esterna che riferisce il codice della nave che effettua il viaggio) e *data\_ora\_partenza*. Non è infatti possibile che i valori di questa coppia di attributi possano ripetersi, in quanto anche se una nave effettuerà più viaggi in partenza non potrà farlo partendo lo stesso giorno alla stessa ora! Le tabelle che riferiscono un viaggio (una specifica riga della tabella *Viaggio*) per costruire la chiave esterna necessitano di tutti e due questi attributi, come è possibile verificare, per esempio, nella struttura della tabella *Messaggio*.

Nella rappresentazione grafica del database le relazioni di cardinalità 1:N del diagramma E/R divengono chiavi esterne che riferiscono le chiavi primarie della tabella associata. Per realizzare l'associazione M:N tra le tabelle *Viaggio* e *Personale* viene introdotta la tabella *Viaggio\_Personale* che ha come chiave primaria gli attributi *matricola*, *id\_nave* e *data\_ora\_partenza*, che consentono di riferire le chiavi primarie delle tabelle *Personale* e *Viaggio*.

## 4.2 Prenotazioni per treni ad alta velocità

### Analisi dello scenario

La società TVI (Treni Veloci Italiani) deve gestire l'esercizio di collegamenti ferroviari ad alta velocità tra alcune delle maggiori città italiane. Un treno della società TVI è caratterizzato da un numero di treno (il numero è lo stesso per le corse che si ripetono in date di partenza diverse<sup>2</sup>), da una stazione di partenza e dal relativo orario di partenza, da una stazione di arrivo e dal relativo orario di arrivo; non sono effettuate fermate intermedie.

Ogni treno ha sempre la stessa composizione in termini di carrozze di prima e seconda classe, ciascuna avente un proprio codice e uno specifico numero di posti (per semplicità si suppone che per un treno vengano utilizzate sempre le stesse carrozze fisiche, una carrozza però può essere utilizzata per comporre, in momenti diversi, treni diversi).

Il costo del biglietto dipende esclusivamente dalla classe (prima o seconda) e dalla tratta percorsa, cioè dalla stazione di partenza e da quella di arrivo; una prenotazione è identificata da un codice univoco ed è sempre relativa a un particolare posto di una specifica carrozza di una corsa di un treno per la tratta percorsa e può essere effettuata da un'agenzia per un cliente occasionale o da un cliente registrato (agenzie e clienti hanno un proprio codice identificativo) che la effettua direttamente mediante un servizio accessibile tramite un sito web cui accede mediante username e password; in questa modalità la prenotazione viene confermata mediante una mail inviata all'indirizzo del cliente.

Dovendo fornire alle proprie agenzie concessionarie e agli utenti registrati la possibilità di acquistare biglietti – identificati da un numero progressivo e sempre relativi a una prenotazione specifica, eventualmente modificabi-

2. Per esempio il treno 777 con partenza da Roma alle ore 8.00 e arrivo a Milano alle ore 12.00 ha corse che si ripetono il 14.08.2016, il 16.08.2016, il 17.08.2016, il 19.08.2016, ...

le – il responsabile dei sistemi informativi di TVI commissiona la progettazione del database di supporto al servizio di prenotazione dei posti e di emissione dei biglietti.

### Progetto concettuale

Individuando le seguenti entità come rappresentative dello scenario

- Treni
- Carrozze
- Corse
- Stazioni
- Tratte
- Prenotazioni
- Biglietti
- Agenzie
- Clienti

il diagramma E/R dei FIGURA 18 ne rappresenta gli attributi principali e le associazioni.

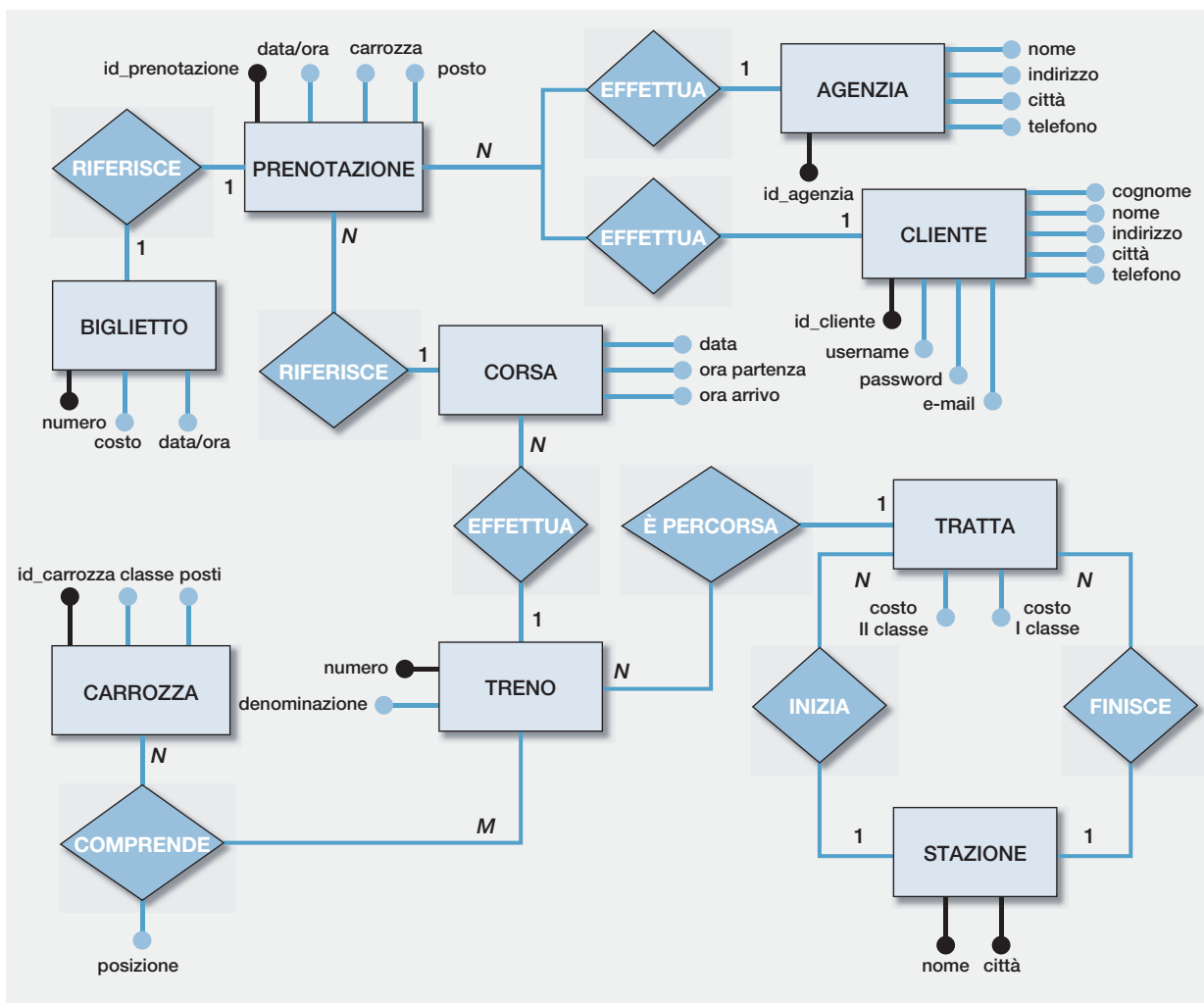


FIGURA 18

**OSSERVAZIONE** Le entità *Tratta* e *Corsa* non hanno una chiave primaria esplicita: sarà individuata in fase di progettazione logica della base di dati.

La cardinalità delle associazioni indicate nel diagramma E/R è motivata dalle seguenti considerazioni:

- *Treno/Corsa* 1:N, lo stesso treno effettua in giorni diversi corse diverse, ma una corsa si riferisce sempre allo stesso treno;
- *Tratta/Treno* 1:N, su una certa tratta viaggiano treni diversi mentre uno specifico treno percorre sempre la stessa tratta;
- *Stazione/Tratta* 1:N, una tratta ha una sola stazione di partenza e una sola stazione di arrivo, ma da una stazione possono partire e possono arrivare più tratte;
- *Treno/Carrozza* M:N, perché un treno comprende più carrozze nelle varie posizioni (1, 2, 3, ...), ma una carrozza può essere usata in momenti diversi per comporre treni diversi;
- *Corsa/Prenotazione* 1:N, una prenotazione si riferisce sempre a una corsa specifica identificata da treno e data, ma le prenotazioni per una corsa possono essere tante quanti sono i posti disponibili;
- *Prenotazione/Biglietto* 1:1, per ogni prenotazione esiste uno e un solo biglietto che può eventualmente essere emesso in una data diversa;
- *Prenotazione/Agenzia* 1:N, ogni prenotazione è effettuata da una sola agenzia (o, in alternativa da un cliente), ma un'agenzia effettua molte prenotazioni;
- *Prenotazione/Cliente* 1:N, ogni prenotazione è effettuata da un solo cliente (o, in alternativa da un'agenzia), ma un cliente può effettuare varie prenotazioni.

**OSSERVAZIONE** Una prenotazione viene effettuata da un'agenzia o, in modo mutuamente esclusivo, da un cliente registrato: per rappresentare questa condizione si prevede di lasciare nullo il valore dell'attributo della tabella che deriva dall'entità *Prenotazione* che non rappresenta la modalità di prenotazione effettuata: *agenzia* per le prenotazioni effettuate dai clienti web, oppure *cliente* per le prenotazioni effettuate dalle agenzie concessionarie.

## Progetto logico

Lo schema grafico di FIGURA 19 rappresenta il database che deriva dal precedente diagramma E/R.

Per ogni entità del diagramma E/R è stata derivata una tabella; la tabella aggiuntiva *Treni\_Carrozze* consente di realizzare l'associazione di cardinalità N:M tra le rispettive entità.

**OSSERVAZIONE** Il fatto che la chiave primaria della tabella *Stazione* sia la coppia di attributi *città* e *nome\_stazione* (nella stessa città possono infatti esserci più stazioni) obbliga a formare chiavi esterne composte da

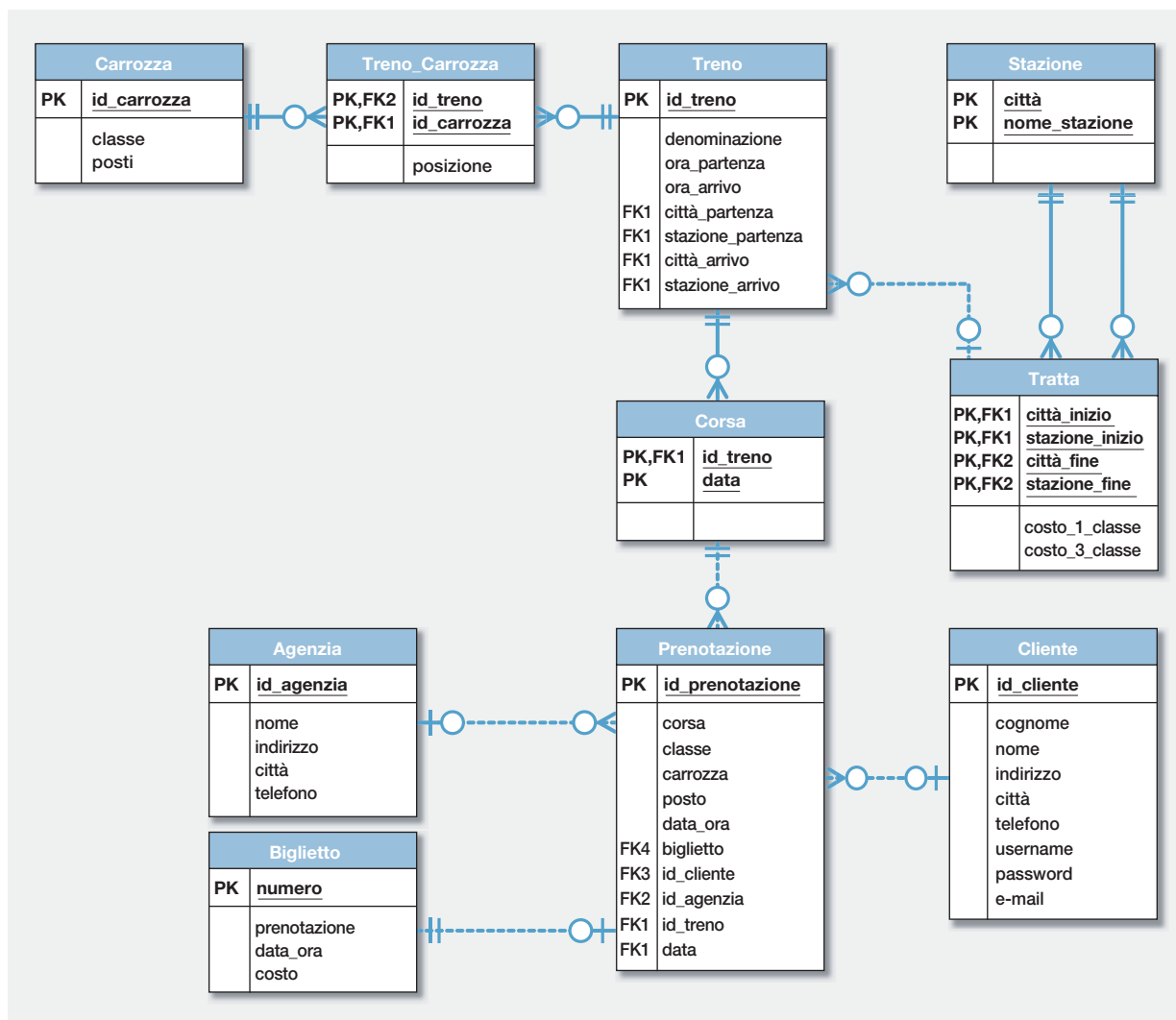


FIGURA 19

due attributi per la tabella *Tratta* che riferisce *Stazione*. Analogamente la tabella *Tratta* ha una chiave primaria composta da quattro attributi (*città\_inizio*, *stazione\_inizio*, *città\_fine*, *stazione\_fine*). Di conseguenza la chiave esterna della tabella *Treno* che riferisce *Tratta* è composta da quattro attributi (*città\_partenza*, *stazione\_partenza*, *città\_arrivo*, *stazione\_arrivo*).

## 5 Linguaggi per operare su basi di dati relazionali

L'utente interagisce con un sistema di gestione della basi di dati utilizzando specifici linguaggi che dal punto di vista funzionale possono essere classificati in:

- **DDL** (*Data Definition Language*), linguaggi che permettono all'amministratore del DBMS di definire la struttura dei dati di una base di dati;

- **DML** (*Data Manipulation Language*), linguaggi per la gestione dei dati contenuti in una base di dati.

L'area dei linguaggi DML è stata nel tempo oggetto di studi che hanno generato una diversità di proposte che è possibile classificare nelle seguenti categorie.

**Algebra relazionale.** Linguaggi basati su un insieme di operatori applicabili alle tabelle per produrre altre tabelle come risultato. Un linguaggio basato sull'algebra relazionale fu proposto da Edgar Codd quando introdusse il modello relazionale dei dati.

**ESEMPIO**

Quella che segue è una possibile formulazione di interrogazione in algebra relazionale per estrarre dal database della biblioteca l'elenco dei libri pubblicati dopo l'anno 2000:

$$T \leftarrow \sigma_{(\text{anno} > 2000)} \text{Libri}$$

La tabella  $T$  risultato dell'interrogazione contiene le righe estratte mediante l'operatore di selezione  $\sigma$  dalla tabella *Libri* sulla base della condizione per cui il valore dell'attributo *anno* deve essere maggiore di 2000.

**Calcolo relazionale.** Linguaggi basati sulla teoria del calcolo dei predicati del primo ordine; anche in questo caso si opera su tabelle per produrre altre tabelle. La differenza rispetto all'algebra relazionale consiste nel fatto che le proprietà cui deve soddisfare una tabella risultato sono descritte con una formula del calcolo dei predicati con variabili quantificate sulle tabelle, cioè con valori che sono righe di una tabella.

**ESEMPIO**

L'espressione

$$\{r \mid \text{Libri}(r) \text{ AND } r.\text{anno} > 2000\}$$

significa: «insieme delle righe  $r$  che appartengono alla tabella *Libri* e il cui il valore dell'attributo *anno* è maggiore di 2000».

**Linguaggi relazionali.** Linguaggi simili a quelli basati sul calcolo relazionale, ma di più facile comprensione dato che i connettivi logici e i quantificatori sono sostituiti da una sintassi fondata su costrutti derivata dalla lingua inglese; le variabili assumono valori sui domini delle colonne delle tabelle. Il principale linguaggio di questa categoria è **SQL** (*Structured Query Language*), che rappresenta da tempo il linguaggio standard per i sistemi di gestione delle basi di dati relazionali.

**ESEMPIO**

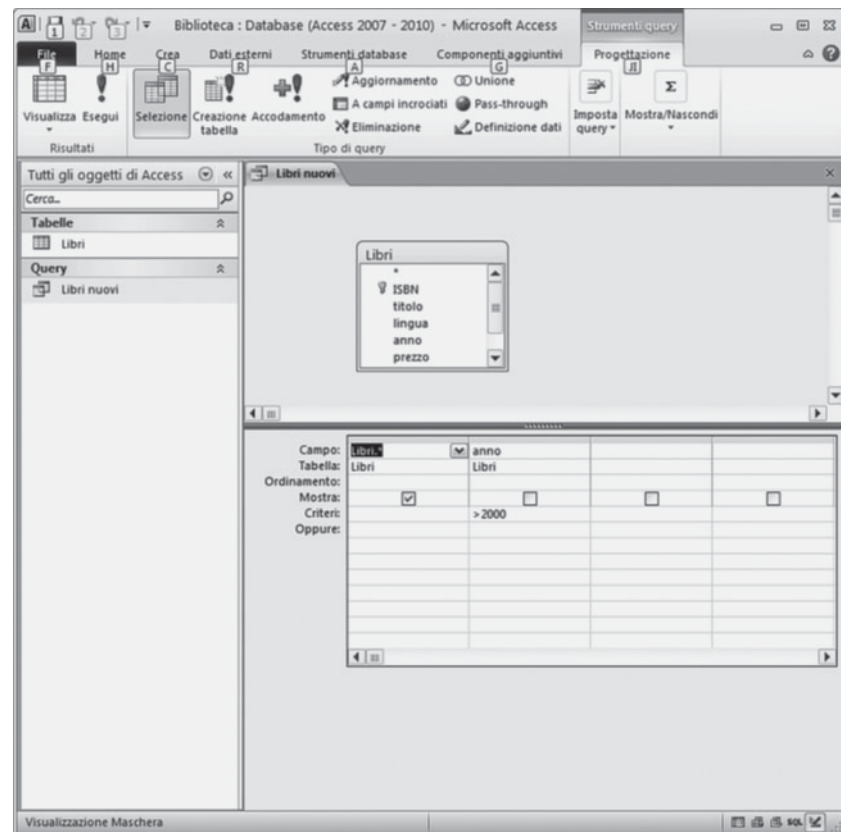
In SQL l'interrogazione per ottenere le righe della tabella *Libri* aventi l'attributo *anno* con valore maggiore di 2000 è:

```
SELECT * FROM Libri WHERE anno > 2000;
```

**Linguaggi grafici.** Caratteristica comune di questa categoria di linguaggi è che le operazioni sui database non sono effettuate utilizzando un linguaggio testuale dotato di una sintassi definita, ma interagendo con il DBMS mediante un'interfaccia grafica; linguaggi di questo tipo sono noti come linguaggi **QBE** (*Query By Example*).

**ESEMPIO**

In Microsoft Access l'interrogazione per ottenere le righe della tabella *Libri* aventi l'attributo *anno* con valore maggiore di 2000 può essere così impostata:



**Linguaggi naturali.** Sono linguaggi per utenti non esperti che intendono interagire con il sistema di gestione delle basi di dati in modalità simile a quella del linguaggio naturale. Questi linguaggi si basano su tecnologie che afferiscono all'area dell'intelligenza artificiale, dovendo effettuare operazioni quali:

- analisi lessicale (scomposizione di una frase in parole);
- analisi grammaticale (associazione delle parti della frase a ciascuna parola);
- analisi sintattica (ordinamento delle parole in una struttura sintattica, generalmente ad albero);
- analisi semantica (assegnazione di un significato alla struttura sintattica e quindi alla frase).

## 6 Transazioni

Uno degli aspetti sulla base del quale sono classificati i sistemi per la gestione di basi di dati è dato dal numero di utenti che possono interagire simultaneamente con essi. Un DBMS è **monoutente** se il sistema può essere utilizzato da un solo utente per volta, è invece **multiutente** se può essere utilizzato da più utenti contemporaneamente, comportando di fatto accessi concorrenti al database. La maggior parte dei DBMS attuali è di tipo multiutente.

**OSSERVAZIONE** Con il termine generico **utente** si intende un'applicazione software che manipola e/o elabora i dati di un database.

### ESEMPIO

Il sito web di una compagnia aerea viene utilizzato contemporaneamente da operatori addetti alle prenotazioni di più agenzie di viaggio, così come avviene nel caso di sistemi informatici bancari o finanziari.

In tutti questi casi il sistema deve essere in grado di supportare l'esecuzione di molteplici operazioni richieste in contemporanea da più utenti o applicazioni.

Relativamente alle operazioni di interrogazione dei dati la problematica è esclusivamente di tipo prestazionale e viene risolta da una scelta adeguata dell'hardware che esegue il DBMS<sup>3</sup> e dal fatto che questo implementi sofisticate tecniche di ottimizzazione delle interrogazioni.

Le operazioni che trasformano la base di dati (inserimento, modifica e cancellazione dei dati) comportano invece problematiche di tipo funzionale. Due situazioni tipiche sono le seguenti:

- due (o più) utenti accedono contemporaneamente allo stesso dato con l'intenzione di aggiornarlo: uno di questi lo farà per primo e, quando l'altro (gli altri) tenta a sua volta l'aggiornamento, trova una situazione variata rispetto al momento in cui aveva letto i dati, con il rischio potenziale di creare uno stato dei dati incoerente;
- un'applicazione software deve effettuare modifiche multiple, ma logicamente correlate fra loro, tanto che tutte devono essere annullate se anche una sola di esse non ha esito positivo.

### ESEMPIO

Due utenti di un sito di e-commerce si apprestano ad acquistare un prodotto di cui al momento è disponibile solo un esemplare. Se l'acquisto viene effettuato in contemporanea, l'applicazione software del sito deve verificare che l'operazione abbia esito positivo per uno solo dei due utenti, evitando che l'altro effettui un pagamento in mancanza del prodotto già venduto.

Il concetto di transazione si riferisce a unità logiche di elaborazione nel corso delle quali una base di dati viene trasformata da uno stato iniziale  $S_i$  consistente a un nuovo stato  $S_n$  consistente.

3. I DBMS professionali possono essere eseguiti da più server (anche centinaia) in modo trasparente per l'utente e le applicazioni.



## Gestione delle transazioni nei DBMS

Nei DBMS le transazioni sono gestite da un componente software denominato *Transaction Processing*, normalmente suddiviso in due moduli: il *Concurrency Control Manager* (CCM), che garantisce l'atomicità e l'isolamento, e il *Logging/Recovery Manager* (LRM), che è invece preposto alla gestione della persistenza e della coerenza.

La transazione effettua le modifiche su una copia del database: se essa non termina con successo la copia viene distrutta, altrimenti le modifiche effettuate sulla copia sono rese permanenti con l'operazione di *commit*, garantendo l'atomicità delle operazioni.

Le varie transazioni sono eseguite in isolamento le une dalle altre, anche se molte di esse operano in maniera concorrente: è il CCM a garantire che le singole azioni delle varie transazioni siano eseguite nell'ordine corretto per non interferire le une con le altre. ►

Un'unica transazione può comprendere una o più operazioni di modifica dei dati e deve automaticamente ripristinare lo stato iniziale  $S_i$  se anche una sola di tali operazioni non ha successo. L'esecuzione di un'applicazione può essere vista come una sequenza di transazioni, intervallate dall'esecuzione di operazioni non relative ai dati.

In una transazione si individuano le seguenti fasi operative:

- inizio della transazione (**BOT**, *Begin Of Transaction*);
- fine della transazione (**EOT**, *End of Transaction*);
- consolidamento del nuovo stato della base di dati dopo le modifiche effettuate con successo (l'operazione che rende definitivi i cambiamenti è nota come **commit**);
- ripristino dello stato iniziale precedente alla transazione in caso di insuccesso (l'operazione che annulla le modifiche applicate alla base di dati è nota come **rollback**).

Lo schema di FIGURA 20 rappresenta invece gli stati fondamentali in cui può trovarsi una transazione nel corso della sua esecuzione:

- **ACTIVE**: transazione in corso;
- **ABORTED**: transazione terminata con insuccesso o interrotta, in attesa dell'invocazione esplicita del comando *rollback* per il ripristino dello stato iniziale;
- **COMMITTED**: transazione terminata con consolidamento del nuovo stato;
- **PARTIALLY COMMITTED**: transazione terminata con successo, prima dell'invocazione esplicita del comando *commit* che consolida il nuovo stato del database;
- **FAILED**: transazione interrotta o terminata esplicitamente, da questo stato si passa automaticamente allo stato **ABORTED**.

**OSSERVAZIONE** Gli stati intermedi di una transazione sono invisibili ad altre eventuali transazioni eseguite in contemporanea: l'interazione con i dati da parte di transazioni diverse avviene in modo «atomico», esclusivamente nello stato che precede l'inizio di una transazione, o nello stato che segue la fine di una transazione.

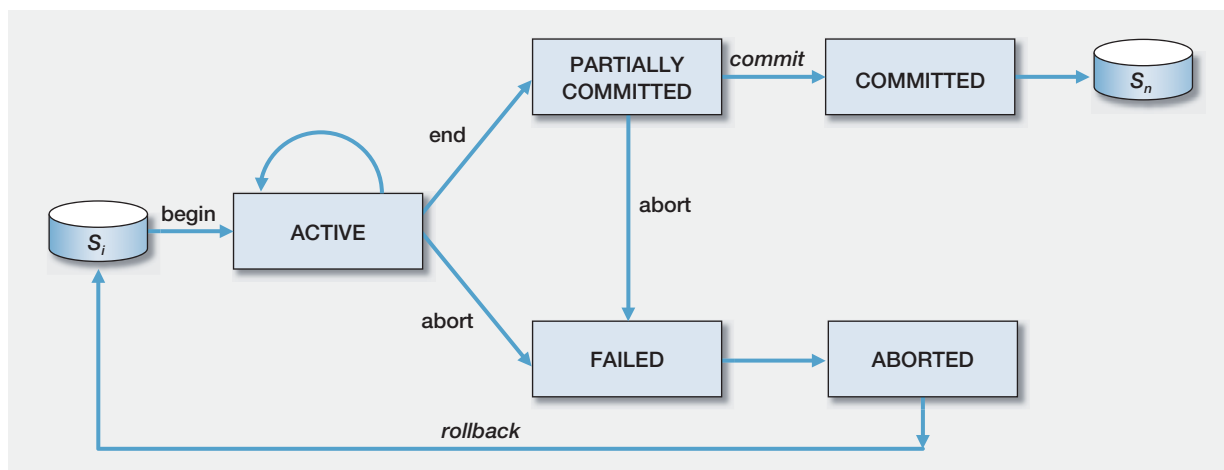


FIGURA 20

Nelle applicazioni reali è spesso necessario impostare transazioni che comprendono più comandi di trasformazione della base di dati: in questi casi l'operazione di *commit* deve essere esplicitamente gestita dal software applicativo. L'acronimo **ACID** (*Atomicity, Consistency, Isolation, Durability*) indica le quattro proprietà fondamentali che devono soddisfare i DBMS che implementano le transazioni perché queste operino in modo corretto sui dati:

- **Atomicity** (atomicità). Garantisce che le operazioni di una transazione siano eseguite in modo «atomico»: tutte o nessuna. Questa proprietà è cruciale per l'integrità dei dati: se un'operazione ha esito negativo, il sistema deve essere in grado di annullare tutti i cambiamenti effettuati a partire dall'inizio della transazione.
- **Consistency** (consistenza). Assicura che l'esecuzione della transazione trasformi la base di dati da uno stato iniziale coerente, ovvero senza violazioni di eventuali vincoli di integrità, a un altro stato coerente quando la transazione ha termine.
- **Isolation** (isolamento). Garantisce che l'esecuzione di una transazione sia indipendente dalla esecuzione contemporanea di altre transazioni in modo che transazioni concorrenti non si influenzino l'una con l'altra (anche l'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione).
- **Durability** (persistenza). Assicura che, se la transazione ha successo, dopo l'esecuzione del comando di *commit*, l'effetto della transazione viene registrato in modo permanente nella base di dati.

**OSSERVAZIONE** Per evitare che nel periodo di tempo che intercorre tra il momento in cui la base di dati si impegna a scrivere le modifiche nel database e quello in cui le scrive effettivamente si verifichino perdite di dati dovuti a malfunzionamenti, sono mantenuti dei registri di *log* in cui sono annotate tutte le operazioni sul database che possono eventualmente essere ripristinate o eliminate in seguito al malfunzionamento.

Le proprietà ACID impongono che un'unica transazione comprenda una o più operazioni di modifica della base di dati che devono essere eseguite in modo corretto e atomico, indipendentemente da altre transazioni, e che si concluda con una eventuale modifica permanente del database.

► Tipicamente questo risultato viene ottenuto attraverso la gestione di *lock* (blocchi di controllo per l'accesso alle risorse del database) memorizzati in una sezione specifica del DBMS e gestita da un particolare modulo detto CCM (*Concurrency Control Manager*).

Il CCM ha anche il compito di risolvere i *deadlock*, cioè le situazioni di stallo dovute alla mutua interazione tra le varie transazioni quando i *lock* non permettono il corretto completamento di alcune di esse, facendo abortire una o più transazioni.

Per assicurare la persistenza e la coerenza dei dati anche in caso di *crash* (dovuto, per esempio, a situazioni di stallo nell'accesso ai dati da parte delle transazioni concorrenti), ogni singola modifica apportata al database viene registrata separatamente in un file di *log*. Il modulo LRM (*Logging/Recovery Manager*) registra queste modifiche per consentire in qualsiasi momento (anche in seguito a un *crash*) il ripristino del database in uno stato consistente.

## 7 Algebra e operatori relazionali

L'algebra relazionale fu introdotta da Edgar Codd come esempio di linguaggio minimale per il calcolo su relazioni, cioè su tabelle; lo stesso Codd introdusse inoltre il concetto di linguaggio relazionale **completo**, inteso come linguaggio i cui operatori devono essere in grado di fornire gli stessi risultati forniti da una qualsiasi espressione dell'algebra relazionale.

Una caratteristica sostanziale dell'algebra relazionale è la proprietà di chiusura: gli operatori si applicano a tabelle per produrre come risultato nuove tabelle. Questa proprietà è fondamentale in quanto permette di applicare un qualsiasi operatore al risultato da una qualsiasi operazione.

I linguaggi basati sull'algebra relazionale forniscono una sintassi specifica per rappresentare i tradizionali operatori insiemistici (unione, intersezione, differenza, prodotto cartesiano) e altri tipici del calcolo con le relazioni (restrizione, proiezione, congiunzione, divisione).

L'algebra relazionale ha sei operatori primitivi, nessuno dei quali può essere omesso senza perdere in espressività, e alcuni operatori derivati che in linea di principio possono essere definiti come combinazione di operatori primitivi.

Gli operatori possono essere di tipo unario, se operano su una sola tabella, o binario, se operano su due tabelle.

### Operatori primitivi:

- **Unione** (unario)
- **Differenza** (binario)
- **Prodotto cartesiano** (binario)
- **Selezione o restrizione** (unario)
- **Proiezione** (unario)
- **Ridenominazione** (unario)

### Operatori derivati:

- **Intersezione** (binario)
- **Congiunzione o join** (binario)
- **Divisione** (binario)

La notazione dell'algebra relazionale prevede simboli speciali per i diversi operatori: nel seguito della trattazione saranno utilizzati sia questi simboli sia una diversa sintassi basata sui nomi degli operatori. Nella **TABELLA 2** che segue sono riportati sia i simboli che i nomi degli operatori corrispondenti:

**TABELLA 2**

Operazione	Operatore	Simbolo
Proiezione	PROJECT	$\pi$
Selezione	RESTRICT	$\sigma$
Ridenominazione	RENAME	$\rho$
Unione	UNION	$\cup$
Prodotto cartesiano	TIMES	$\times$
Congiunzione ( <i>join</i> )	JOIN	$\bowtie$
<i>Left outer join</i>	LEFT OUTER JOIN	$\bowtie$
<i>Right outer join</i>	RIGHT OUTER JOIN	$\bowtie$
<i>Full outer join</i>	FULL OUTER JOIN	$\bowtie$
Divisione	DIVIDE	$\div$
Intersezione	INTERSECT	$\cap$
Differenza	MINUS	$-$

► Con l'espressione **tabelle compatibili** si intendono due tabelle in cui gli attributi sono in corrispondenza biunivoca, in modo che gli attributi corrispondenti risultano definiti sullo stesso dominio.

**OSSERVAZIONE** Gli operatori base che si applicano agli insiemi (unione, intersezione e differenza) possono essere applicati solo a tabelle che abbiano lo stesso insieme di attributi (uguali per numero, nome e tipo). Questa è una limitazione di tipo matematico, ma anche logico: non si devono considerare omogenei valori di natura diversa; il risultato di queste operazioni è una tabella il cui schema è uguale a quello delle due tabelle di partenza.

■ **Unione.** L'unione di due tabelle compatibili  $R$  e  $S$

$R \text{ UNION } S$

produce una tabella risultato costituita dall'insieme delle righe che appartengono a  $R$  e/o a  $S$ . Nell'esempio che segue la tabella  $T$  è il risultato dell'unione tra  $R$  e  $S$ :

R			S			T		
x	y	z	x	y	z	x	y	z
p	1	2	s	2	4	p	1	2
p	2	1	s	3	3	p	2	1
q	1	2				q	1	2
						s	2	4
						s	3	3

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$T \leftarrow R \cup S$

■ **Differenza.** La differenza di due tabelle compatibili  $R$  e  $S$

$R \text{ MINUS } S$

è costituita dalle righe di  $R$  che non appartengono a  $S$ . Nell'esempio che segue la tabella  $T$  è il risultato della differenza tra  $R$  e  $S$ :

R			S			T		
x	y	z	x	y	z	x	y	z
p	1	2	s	2	4	p	1	2
s	2	4	s	3	3	p	2	1
q	1	2				q	1	2
p	2	1						
s	3	3						

Con la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow R - S$$

■ **Prodotto cartesiano.** Il prodotto cartesiano di due tabelle  $R$  e  $S$  non necessariamente compatibili

$$R \text{ TIMES } S$$

produce, come tabella risultato, un insieme le cui righe sono costituite dalla concatenazione di **ogni** riga di  $R$  **con tutte** le righe di  $S$ . Nell'esempio che segue la tabella  $T$  è il risultato del prodotto cartesiano tra  $R$  e  $S$ :

R			S		T				
x	y	z	k	l	x	y	z	k	l
p	1	2	2	t	p	1	2	2	t
p	2	1	3	u	p	1	2	3	u
q	1	2	4	v	p	1	2	4	v
r	2	5			p	2	1	2	t
r	3	3			p	2	1	3	u
					p	2	1	4	v
					q	1	2	2	t
					q	1	2	3	u
					q	1	2	4	v
					r	2	5	2	t
					r	2	5	3	u
					r	2	5	4	v
					r	3	3	2	t
					r	3	3	3	u
					r	3	3	4	v

Con la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow R \times S$$

■ **Selezione.** L'operatore **RESTRICT** permette la costruzione di un sottoinsieme «orizzontale» (per righe) di una tabella selezionando tutte le righe di una tabella che soddisfino a una specificata condizione logica. Per esempio il comando

$$\text{RESTRICT } R \text{ WHERE } z=2$$

restituisce una nuova tabella  $T$  in cui sono presenti solo le righe della tabella  $R$  in cui l'attributo  $z$  assume valore 2:

R			T		
x	y	z	x	y	z
p	1	2	p	1	2
p	2	1			
q	1	2	q	1	2
r	2	5			
r	3	3			

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow \sigma_{(z=2)} S$$

■ **Proiezione.** L'operatore PROJECT viene usato per costruire un sottoinsieme «verticale» (per colonne) di una tabella selezionando solo le colonne degli attributi specificati ed eliminando eventuali righe replicate, condizione che può verificarsi nel caso in cui non siano selezionati gli attributi che formano la chiave primaria. Per esempio il comando

PROJECT R OVER k, l

restituisce una nuova tabella  $T$  in cui sono presenti solo le colonne della tabella  $R$  identificate dagli attributi  $k$  e  $l$ :

R					T	
x	y	z	k	l	k	l
p	1	2	2	t	2	t
q	1	2	2	t	5	t
r	2	5	5	t	3	u
r	3	3	3	u		

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow \pi_{(k,l)} R$$

■ **Ridenominazione.** L'operatore RENAME non altera il contenuto di una tabella, ma ne modifica lo schema cambiando il nome di alcuni attributi. Esso permette di trasformare una tabella che ha domini dei dati coerenti con quelli di un'altra tabella, ma attributi con nomi diversi in una tabella con lo stesso schema spesso allo scopo di applicare a esse un operatore binario dell'algebra relazionale che richiede la coincidenza dei nomi degli attributi. La ridenominazione è corretta solo se il nuovo schema ha attributi con nomi tutti distinti. Utilizzando questo operatore è anche possibile modificare il nome di una tabella.

Per esempio il comando

RENAME R (x, y), (t, v)

applicato alla tabella  $R$  modifica i nomi degli attributi  $x$  e  $y$  rispettivamente in  $t$  e  $v$ , lasciando inalterato il contenuto della tabella  $R$  (i valori della colonna associata a  $x$  sono associati a  $t$  e quelli associati a  $y$  a  $v$ ).

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$\rho_{(t,v) \leftarrow (x,y)} R$$

■ **Intersezione.** L'intersezione di due tabelle compatibili  $R$  e  $S$

R INTERSECT S

produce una tabella risultato formata dall'insieme delle righe che sono comuni a  $R$  e a  $S$ . Per esempio l'intersezione tra le due seguenti tabelle  $R$  e  $S$  produce la tabella  $T$ :

R			S			T		
x	y	z	x	y	z	x	y	z
p	1	2	s	2	4	s	2	4
s	2	4	s	3	1			
q	1	2						
p	2	1						
s	3	3						

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow R \cap S$$

■ **Congiunzione.** L'operatore `JOIN` può essere applicato a una coppia di tabelle  $R$  e  $S$  per produrre una tabella le cui righe sono costituite dal prodotto cartesiano delle righe di  $R$  e  $S$  che soddisfano a una specificata condizione (a differenza dell'operatore prodotto cartesiano che produce come risultato tutte le possibili combinazioni tra le righe di due tabelle). Il formato generale del comando è

```
JOIN R WITH S WHERE (R.<attributo> <operatore> S.<attributo>)
```

dove i due  $R.<attributo>$  e  $S.<attributo>$  devono essere definiti sullo stesso dominio di dati e  $<operatore>$  può essere uno dei seguenti:  $>$ ,  $<$ ,  $=$ ,  $<>$ ,  $>=$ ,  $<=$ .

**OSSERVAZIONE** L'operatore `JOIN` è derivato: i risultati che produce sono infatti i risultati che si ottengono applicando l'operatore `RESTRICT` (selezione) al risultato dell'operatore `TIMES` (prodotto cartesiano).

Per esempio il comando

```
JOIN R WITH S WHERE (z=k)
```

produce la seguente tabella  $T$  a partire dalle tabelle  $R$  e  $S$  selezionando dal prodotto cartesiano di  $R$  e  $S$  le sole righe per le quali il valore dell'attributo  $z$  è uguale al valore dell'attributo  $k$ :

R			S		T				
x	y	z	k	l	x	y	z	k	l
p	1	2	2	t	p	1	2	2	t
p	2	1	3	u	q	1	2	2	t
q	1	2	4	v	r	2	5	5	t
r	2	5	5	t	r	3	3	3	u
r	3	3	4	t					



Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow R \bowtie_{(z=k)} S$$

**OSSERVAZIONE** Il caso dell'esempio precedente è quello più frequente e rappresenta una cosiddetta *equi-join*, un tipo di *join* che consente di correlare due tabelle sulla base di valori *uguali* in attributi che hanno come dominio lo stesso tipo di dati. Dato che spesso i due attributi su cui opera l'uguaglianza hanno lo stesso nome, è in questo caso possibile esprimere un'operazione di *equi-join* con la seguente sintassi (*natural-join*):

JOIN R WITH S ON <attributo>

Date due tabelle *R* e *S* l'applicazione dell'operatore JOIN non mantiene nel risultato le righe di *R* che non hanno corrispondenze in *S* e viceversa, secondo il criterio di congiunzione specificato. Per ovviare a questo problema esiste l'operatore OUTERJOIN, nelle forme LEFT, RIGHT e FULL, che rende possibile l'inserimento nel risultato delle righe che sarebbero altrimenti ignorate dall'operatore JOIN. L'applicazione dell'operatore OUTERJOIN completa con valori nulli le righe che sarebbero eliminate con l'applicazione dell'operatore JOIN.

Per esempio il comando

```
LEFT OUTERJOIN R WITH S WHERE (z=k)
```

applicato alle seguenti tabelle *R* e *S* produce come risultato la tabella *T*:

R			S		T				
x	y	z	k	l	x	y	z	k	l
p	1	2	2	t	p	1	2	2	t
p	2	1	3	u	p	2	1	null	null
q	1	2	4	v	q	1	2	2	t
r	2	5	5	t	r	2	5	5	t
r	3	3	4	t	r	3	3	3	u

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow R \bowtie_{(z=k)} S$$

■ **Divisione.** Nella sua forma più semplice la divisione tra tabelle è definita come un'operazione tra una tabella di grado 2*R* (dividendo) e una tabella di grado 1*S* (divisore), che produce una tabella di grado 1 (quoziente). Il formato generale del comando di divisione è

```
DIVIDE R BY S OVER (R.<attributo>;S.<attributo>)
```

dove *R.<attributo>* e *S.<attributo>* devono essere definiti sullo stesso dominio e la tabella che si ottiene come risultato è costituita dal minimo sottoinsieme di valori della colonna identificata da *R.<attributo>* tale che il suo prodotto cartesiano con i valori di *S.<attributo>* è incluso in *R*.

Anche in questo caso, se i due attributi coinvolti hanno lo stesso nome, questo può essere specificato una sola volta.  
Per esempio il comando

DIVIDE R BY S OVER k

applicato alle seguenti tabelle R e S produce come risultato la tabella T:

R		S	T
k	1	k	1
2	t	2	t
3	u	4	
4	v		
5	t		
4	t		

Utilizzando la sintassi dell'algebra relazionale l'operazione ha la seguente scrittura:

$$T \leftarrow R \div S$$

## ESEMPIO

Vediamo alcuni esempi di interrogazioni formulate mediante gli operatori dell'algebra relazionale sulla base di dati relativa alla biblioteca di cui si riporta lo schema grafico (FIGURA 21) e una possibile configurazione delle tabelle che la costituiscono:

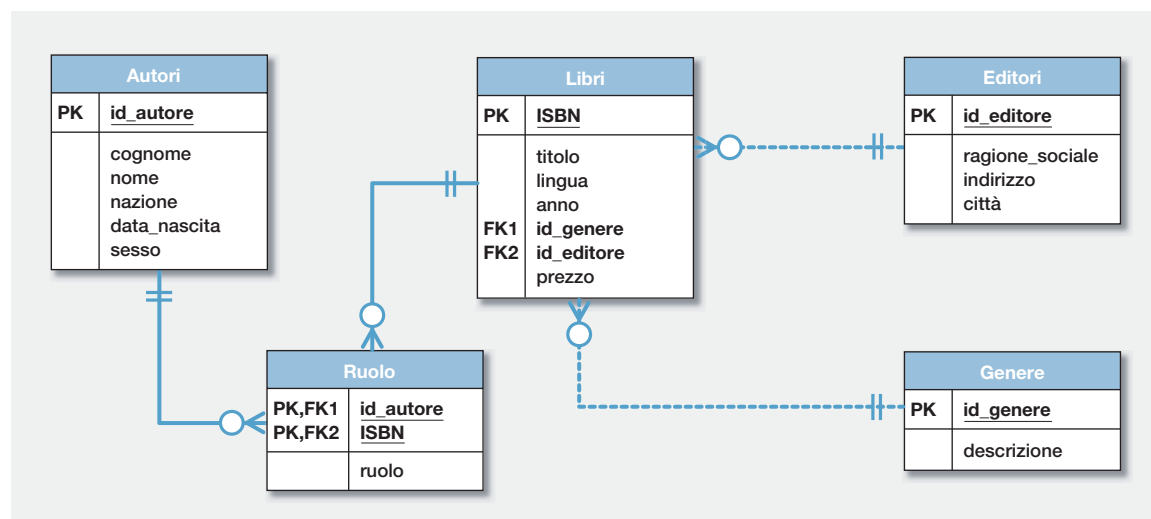


FIGURA 21

Libri						
ISBN	titolo	lingua	anno	prezzo	id_genere	id_editore
123-456-789	Pinocchio	Italiano	1990	10,00€	G01	E100
987-654-321	The Hobbit	Inglese	1999	10,00€	G01	E101
012-345-678	The Lord of Rings	Inglese	2001	25,00€	G01	E101
876-543-210	Informatica	Italiano	2012	20,00€	G99	E099

Autori					
id_autore	cognome	nome	nazione	data_nascita	sexso
A001	Tolkien	John	Inghilterra	03/01/1892	M
A002	Collodi	Carlo	Italia	24/11/1826	M
A003	Meini	Giorgio	Italia	23/03/1965	M
A004	Formichi	Fiorenzo	Italia	06/10/1954	M

Editori			
id_editore	ragione_sociale	indirizzo	citta
E099	Zanichelli	Via Irnerio, 34	Bologna
E100	Mondadori	Via Bianca di Savoia, 12	Milano
E101	RCS	Via Rizzoli, 8	Milano

Genere	
id_genere	descrizione
G01	Fantasy
G99	Tecnologia
G90	Scienza

Ruolo		
id_autore	ISBN	ruolo
A002	123-456-789	autore
A001	987-654-321	autore
A001	012-345-678	autore
A004	876-543-210	autore
A003	876-543-210	coautore

- Elenco degli autori di nazionalità italiana:

```
RESTRICT Autori WHERE nazione = 'Italia'
```

id_autore	cognome	nome	nazione	data_nascita	sexso
A002	Collodi	Carlo	Italia	24/11/1826	M
A003	Meini	Giorgio	Italia	23/03/1965	M
A004	Formichi	Fiorenzo	Italia	06/10/1954	M

- Elenco dei libri con il solo titolo e anno di pubblicazione:

```
PROJECT Libri OVER titolo, anno
```

titolo	anno
Pinocchio	1990
The Hobbit	1999
The Lord of Rings	2001
Informatica	2012

- Elenco dei titoli dei libri il cui prezzo è minore di 25 euro:

```
PROJECT (
  RESTRICT Libri WHERE prezzo < 25
) OVER titolo
```

L'effetto di questo comando è quello di selezionare prima le righe della tabella *Libri* in cui l'attributo *prezzo* assume valori minori di 25, quindi sulle tabella ottenute come risultato viene effettuata una proiezione per isolare i valori assunti dal solo attributo *titolo*.

titolo
Pinocchio
The Hobbit
Informatica

- Elenco degli autori, limitato al cognome e al nome, che hanno partecipato alla redazione di libri in qualità di coautori:

```
PROJECT (
    JOIN (
        RESTRICT Ruolo WHERE ruolo = 'coautore'
    ) WITH Autori ON id_autore
) OVER cognome, nome
```

L'effetto di questo comando si sviluppa in tre fasi:

- selezione delle righe della tabella *Ruolo* in cui l'attributo *ruolo* assume il valore «coautore»;
- prodotto cartesiano tra le righe selezionate nella fase 1 e quelle della tabella *Autori* in cui vi sia coincidenza di valori per gli attributi *id\_autore*;
- proiezione dei valori degli attributi *cognome* e *nome* delle righe ottenute nella fase 2.

cognome	nome
Meini	Giorgio

## Sintesi

■ **Diagrammi E/R.** Rappresentazioni grafiche che permettono di costruire un modello della realtà in termini di entità e associazioni.

■ **Entità.** Rappresentano classi di oggetti (cose, persone, eventi, ecc.) che hanno proprietà comuni. Un'istanza (o occorrenza) di un'entità è un oggetto della classe che l'entità rappresenta: la persona X, il corso di laurea Y, l'auto Z, sono esempi di istanze di entità (*Persone*, *Corsi di laurea*, *Veicoli*). In un diagramma E/R, ogni entità ha un nome che la identifica univocamente e viene rappresentata graficamente mediante un rettangolo etichettato con il nome dell'entità e l'elenco delle sue proprietà (o attributi).

■ **Associazioni.** Le associazioni (o relazioni) costituiscono il mezzo tramite il quale vengono

modellate le corrispondenze tra le istanze di due o più entità: il grado dell'associazione indica il numero di entità coinvolte. Restrignendo la casistica ad associazioni binarie tra due entità *A* e *B*, dove *A* è il dominio e *B* il codominio dell'associazione, si possono distinguere associazioni di cardinalità 1:1 (a ogni istanza di *A* corrisponde una e una sola istanza di *B* e viceversa), 1:*N* (a ogni istanza di *A* possono corrispondere 0, 1 o più istanze di *B*, ma a ogni istanza di *B* corrisponde una e una sola istanza di *A*), *M*:*N* (a ogni istanza di *A* possono corrispondere 0, 1 o più istanze di *B* e viceversa).

■ **Attributi.** Le entità (e in alcuni casi le associazioni) sono descritte tramite un elenco di attributi (o proprietà). Tutti gli oggetti istanza della stessa entità (o eventualmente associazione) hanno

gli stessi attributi. La scelta degli attributi riflette il livello di dettaglio con il quale sono rappresentate le informazioni sulle entità (o sulle associazioni).

■ **Chiavi.** Per ciascuna entità si definisce una chiave, ovvero un insieme minimale di attributi che identifica univocamente ogni istanza dell'entità.

■ **Modello relazionale.** Il modello relazionale è un modello formale per la rappresentazione della struttura logica di una base di dati finalizzato al conseguimento dell'indipendenza dei dati dalle applicazioni e alla semplificazione dei linguaggi impiegati per la loro definizione e manipolazione. Il modello relazionale dei dati è basato su un approccio matematico rigoroso, avendo come fondamenti teorici il concetto di relazione matematica, la teoria elementare degli insiemi e la logica dei predicati del primo ordine.

■ **Relazione.** Dati gli insiemi  $D_1, D_2, \dots, D_N$  (non necessariamente distinti) una relazione  $R$  su questi  $N$  è costituita da un insieme di ennuple  $(d_1, d_2, \dots, d_N)$  tali che l'elemento  $d_i$  appartiene all'insieme  $D_i$  per ogni indice  $i = 1, 2, \dots, N$ ; in altre parole  $R$  è un sottoinsieme del prodotto cartesiano  $D_1 \times D_2 \times \dots \times D_N$ . Gli insiemi  $D_1, D_2, \dots, D_N$  il numero  $N$  e il numero delle ennuple che compongono la relazione  $R$  costituiscono rispettivamente i domini, il grado e la cardinalità di  $R$ . Per rendere ininfluente l'ordine delle colonne si usa «etichettare» le medesime in modo che ogni etichetta (attributo) identifichi univocamente una singola colonna ottenendo così una «relazione con attributi». In generale un tale tipo di relazione  $R$  viene rappresentata come una tabella bidimensionale in cui: le colonne sono identificate da attributi distinti e ognuna di esse corrisponde a un dominio della relazione  $R$  (l'insieme degli attributi è rappresentato con il termine «schema»); ogni riga corrisponde a una ennupla e non vi possono essere due o più righe uguali; l'ordine delle righe e delle colonne non è significativo. Nel modello relazionale, i concetti di entità, proprietà e associazione sono sostituiti dall'unico concetto di tabella, cioè un insieme di valori degli attributi che rappresentano sia le proprietà delle entità sia le associazioni.

■ **Superchiave.** In ogni tabella esiste almeno un sottoinsieme  $K$  degli attributi definito «superchiave» tale che, per ogni possibile coppia di righe distinte, il valore assunto dalla superchiave è diverso.

■ **Chiave candidata.** In ogni tabella esiste almeno un sottoinsieme  $K$  di attributi definito «chiave candidata» tale che:  $K$  è una superchiave e nessun elemento di  $K$  può essere eliminato senza perdere questa proprietà.

■ **Chiave primaria.** Tra tutte le possibili chiavi candidate di una tabella ne viene scelta una definita «chiave primaria» (normalmente ne viene scelta una formata da un numero minimale di attributi).

■ **Chiave esterna.** È un insieme di attributi di una tabella  $T_1$  che assumono valori nel dominio di una chiave primaria di una tabella  $T_2$  e che generalmente non costituiscono una chiave primaria per  $T_1$ . Le chiavi esterne sono impiegate per implementare nel modello relazionale le associazioni di cardinalità 1:N.

■ **Base di dati relazionale.** Una base di dati relazionale è costituita da un insieme finito di tabelle, le cui righe variano nel tempo, tante quanti sono gli schemi delle relazioni che ne definiscono la struttura logica.

■ **Dipendenza funzionale.** Sia  $X = \{x_1, x_2, \dots, x_N\}$  un insieme di attributi di una tabella  $T$  e  $Y$  un attributo di  $T$ . Si dice che  $Y$  dipende funzionalmente da  $X$  (o che  $X$  determina  $Y$ :  $X \rightarrow Y$ ) se e solo se, per ogni possibile configurazione del contenuto delle righe di  $T$ , i valori degli attributi di  $X$  determinano univocamente il valore di  $Y$ . In altri termini, si afferma che, se due o più righe hanno gli stessi valori per gli attributi di  $x_1, x_2, \dots, x_N$ , allora hanno necessariamente anche lo stesso valore dell'attributo  $Y$ , cioè l'insieme di attributi  $X$  rappresenta una chiave per  $Y$ . Data una tabella  $T$  e due insiemi di attributi  $X$  e  $Y$  non vuoti di  $T$  per cui si ha  $Y \subseteq X$ , allora ogni possibile configurazione del contenuto delle righe di  $T$  soddisfa la dipendenza funzionale  $X \rightarrow Y$ : in questo caso si parla di dipendenza funzionale banale. Una dipendenza funzionale è parziale se  $Y$  dipende da un sottoinsieme  $X'$  di  $X$ , mentre è completa se  $Y$  dipende da tutti gli attributi di  $X$ .

■ **Normalizzazione di una base di dati relazionale.** È un procedimento che ha lo scopo di evitare le anomalie che possono verificarsi in una base di dati relazionale a fronte di operazioni di inserimento, aggiornamento e cancellazione di dati.

È basato sulla decomposizione di una tabella in più tabelle e si articola principalmente su tre «forme normali» (1NF, 2NF e 3NF).

■ **Prima forma normale (1NF).** Una tabella è in 1NF se tutti i suoi attributi hanno domini atomici (cioè sono valori elementari non ulteriormente scomponibili) ed esiste per essa una chiave primaria.

■ **Seconda forma normale (2NF).** Una tabella è in 2NF se è in 1NF e non esistono tra i possibili insiemi di attributi dipendenze funzionali parziali, ma si hanno solo dipendenze funzionali complete: ogni attributo  $A$  che non fa parte della chiave primaria dipende funzionalmente dalla chiave primaria e non da un sottoinsieme di essa.

■ **Terza forma normale (3NF).** Una tabella è in 3NF se è in 2NF e non esistono attributi non appartenenti alla chiave primaria dipendenti trasversalmente dalla chiave primaria.

■ **Forma normale di Boyce-Codd (BCNF).** Una tabella  $T$  è in BCNF se e solo se è in 3NF e, per ogni dipendenza funzionale non banale  $X \rightarrow Y$ ,  $X$  è una superchiave per  $T$ . Ogni tabella in BCNF è anche in 3NF, ma non è sempre vero il contrario.

■ **Linguaggi per DBMS.** Tutti i linguaggi per DBMS integrano una componente DDL (*Data Definition Language*) per la definizione della struttura delle basi di dati e una componente DML (*Data Manipulation Language*) per l'interrogazione e la modifica dei dati. Tra i linguaggi per DBMS si hanno: l'algebra e il calcolo relazionale, i linguaggi relazionali come SQL (*Structured Query Language*), i linguaggi grafici come QBE (*Query By Example*) e i linguaggi naturali interpretati con tecniche di AI (*Artificial Intelligence*).

■ **Transazioni.** Il concetto di transazione si riferisce a unità logiche di elaborazione nel corso delle quali una base di dati viene trasformata da uno stato iniziale  $S_i$  consistente a un nuovo stato  $S_n$  consistente. Un'unica transazione può comprendere una o più operazioni di modifica dei dati e deve automaticamente ripristinare lo stato iniziale  $S_i$  se anche una sola di tali operazioni non ha successo. L'esecuzione di un'applicazione può essere vista come una serie di transazioni, intervallate dall'esecuzione di operazioni non relative ai dati.

■ **Commit.** Comando di consolidamento delle trasformazioni operate da una transazione su una base di dati che conferma il passaggio dallo stato  $S_i$  al nuovo stato  $S_n$ .

■ **Rollback.** Comando di ripristino dello stato  $S_i$  di una base di dati dopo il fallimento di una transazione.

■ **ACID.** Acronimo dei termini *Atomicity*, *Consistency*, *Isolation* e *Durability*, con cui si indicano le quattro proprietà fondamentali che devono soddisfare i DBMS che implementano le transazioni perché queste operino in modo corretto sui dati.

■ **Atomicità.** Le operazioni di una transazione sono eseguite in modo «atomico»: tutte o nessuna. Se un'operazione ha esito negativo, il sistema deve essere in grado di annullare tutti i cambiamenti effettuati a partire dall'inizio della transazione.

■ **Consistenza.** L'esecuzione della transazione deve trasformare la base di dati da uno stato iniziale coerente (senza violazioni dei vincoli di integrità), a un altro stato coerente quando la transazione ha termine.

■ **Isolamento.** L'esecuzione di una transazione deve essere indipendente dalla esecuzione contemporanea di altre transazioni in modo che transazioni concorrenti non si influenzino l'una con l'altra.

■ **Durabilità (o persistenza).** Se una transazione ha successo, dopo l'esecuzione del comando di *commit*, l'effetto della transazione deve essere registrato in modo permanente nella base di dati.

■ **Algebra relazionale.** L'algebra relazionale fu introdotta da Codd come esempio di linguaggio minimale per il calcolo su relazioni, cioè su tabelle. Egli introdusse anche il concetto di linguaggio relazionale «completo», inteso come linguaggio i cui operatori devono essere in grado di fornire gli stessi risultati forniti da una qualsiasi espressione dell'algebra relazionale. Una caratteristica sostanziale dell'algebra relazionale è la proprietà di chiusura: gli operatori si applicano a tabelle per produrre come risultato nuove tabelle; questa proprietà è fondamentale in quanto permette di applicare un qualsiasi operatore al risultato di una qualsiasi operazione.

■ **Operatori primitivi dell'algebra relazionale.** **Unione:** effettua l'unione delle righe di due tabelle di uguale grado e con attributi aventi lo stesso nome. **Differenza:** la differenza tra due tabelle compatibili  $R$  e  $S$  produce come risultato una tabella costituita dalle righe di  $R$  che non appartengono a  $S$ . **Prodotto cartesiano:** date due tabelle  $R$  e  $S$ , produce come risultato una tabella le cui righe sono costituite dalla concatenazione di ogni riga di  $R$  con tutte le righe di  $S$ . **Selezione (o restrizione):** permette la costruzione di un sottoinsieme «orizzontale» di una tabella dato dalle righe che soddisfano una condizione specificata. **Proiezione:** permette la costruzione di un sottoinsieme «verticale» di una tabella che comprende solo le colonne relative ad attributi specificati, eliminando eventuali righe replicate. **Ridenominazione:** consente di modificare lo schema di una tabella cambiando

il nome di uno o più attributi, ma lasciando inalterato il contenuto.

■ **Operatori derivati dell'algebra relazionale.** **Intersezione:** date due tabelle compatibili  $R$  e  $S$ , produce una tabella risultato formata dall'insieme delle righe che sono comuni a  $R$  e a  $S$ . **Congiunzione (o join):** date due tabelle  $R$  e  $S$ , produce una tabella le cui righe sono costituite dal prodotto cartesiano delle righe di  $R$  e di  $S$  che soddisfano a una condizione specificata. **Divisione:** nella sua forma più semplice è definita come un'operazione tra una tabella di grado  $2R$  (dividendo) e una tabella di grado  $1S$  (divisore); produce come risultato una tabella di grado 1 (quoziente) che comprende il minimo sottoinsieme di valori della colonna di  $R$  specificata tale che il suo prodotto cartesiano con i valori della colonna di  $S$  specificata è incluso in  $R$ .

## QUESITI

### 1 I diagrammi E/R servono a ...

- A ... rappresentare algoritmi.
- B ... sviluppare procedure software.
- C ... modellare una realtà.
- D Nessuna delle risposte precedenti.

### 2 Quali dei seguenti sono elementi su cui si basano i diagrammi E/R?

- A Associazioni.
- B Tabelle.
- C Entità.
- D Comandi condizionali.

### 3 Un'associazione è ...

- A ... un mezzo per modellare le caratteristiche di un'entità.
- B ... una corrispondenza tra tabelle di una base di dati relazionale.
- C ... una tabella del modello dei dati relazionale.
- D Nessuna delle risposte precedenti.

### 4 Un'entità è ...

- A ... una classe di oggetti aventi proprietà comuni.
- B ... una classe di associazioni aventi proprietà comuni.
- C ... una tabella del modello dei dati relazionale.
- D Nessuna delle risposte precedenti.

### 5 Quali delle seguenti affermazioni circa le associazioni di cardinalità $M:N$ sono vere?

- A Possono essere rappresentate direttamente con due tabelle nel modello relazionale.
- B La corrispondenza è multipla e parziale in entrambe le direzioni, cioè a ogni istanza dell'entità dominio ne corrispondono 0, 1 o più dell'entità codominio e viceversa.
- C Non possono essere rappresentate direttamente con due tabelle nel modello relazionale.
- D Nessuna delle risposte precedenti.

### 6 Un attributo è ...

- A ... una proprietà di un'entità.
- B ... un mezzo per modellare una corrispondenza tra istanze di entità distinte.



- C ... un mezzo per modellare una corrispondenza ricorsiva tra istanze della stessa entità.
- D Nessuna delle risposte precedenti.
- 7** Una chiave è ...
- A ... una tecnica per definire proprietà comuni a entità diverse.
- B ... un insieme minimale di attributi che identifica univocamente un'istanza di un'entità.
- C ... un mezzo per modellare una corrispondenza tra istanze di entità distinte.
- D Nessuna delle risposte precedenti.
- 8** Una relazione nel modello relazionale viene implementata tramite ...
- A ... una tabella bidimensionale.
- B ... una tabella a più dimensioni.
- C ... almeno due tabelle bidimensionali distinte.
- D Nessuna delle risposte precedenti.
- 9** Una chiave candidata nel modello relazionale è ...
- A ... un insieme di attributi (tutti indispensabili) i cui valori identificano univocamente le righe di una tabella.
- B ... un insieme di attributi (non tutti indispensabili) i cui valori identificano univocamente le righe di una tabella.
- C ... è la stessa cosa di una superchiave.
- D Nessuna delle risposte precedenti.
- 10** Una chiave primaria nel modello relazionale è ...
- A ... una chiave scelta tra le possibili chiavi candidate.
- B ... una chiave scelta tra le possibili superchiavi.
- C ... una chiave qualsiasi.
- D Nessuna delle risposte precedenti.
- 11** Un attributo di una tabella viene detto primo ...
- A ... se è atomico, ovvero non ulteriormente scomponibile.
- B ... se è membro di una qualsiasi superchiave.
- C ... se è membro di una qualsiasi chiave candidata
- D Nessuna delle risposte precedenti.

- 12** Quali delle seguenti affermazioni circa una chiave esterna nel modello relazionale sono vere?
- A Serve a modellare un'associazione di cardinalità 1:N tra le righe di due tabelle.
- B È sicuramente chiave primaria per due tabelle tra cui si vuole modellare un'associazione di cardinalità 1:N tra le relative righe.
- C È chiave primaria per una sola tra due relazioni di cui si vuole modellare un'associazione di cardinalità M:N tra le relative righe.
- D Nessuna delle risposte precedenti.
- 13** Tramite una chiave esterna si realizza un vincolo di integrità referenziale?
- A Sì, sempre.
- B No, mai.
- C Sì, ma solo se la chiave primaria a cui fa riferimento è composta da un solo attributo.
- D Nessuna delle risposte precedenti.
- 14** Il valore `NULL` assunto da un attributo di una tabella in una sua riga indica ...
- A ... valore non presente.
- B ... valore 0 se il campo è numerico.
- C ... stringa vuota se il campo è di tipo testuale.
- D Nessuna delle risposte precedenti.
- 15** In quali delle seguenti operazioni una relazione non normalizzata può presentare problemi?
- A Interrogazione.
- B Inserimento.
- C Cancellazione.
- D Aggiornamento.
- 16** Quali delle seguenti affermazioni relative al procedimento di normalizzazione di una base di dati relazionale sono vere?
- A Tende ad aumentare il numero di tabelle tramite un processo di scomposizione.
- B Tende a diminuire il numero di tabelle tramite un processo di composizione.
- C Lascia inalterato il numero di tabelle della base di dati, ma ne modifica lo schema.
- D Nessuna delle risposte precedenti.

**17** Il concetto di dipendenza funzionale nel modello relazionale è ...

- A ... una dipendenza che esiste tra la chiave di una tabella e gli altri suoi attributi.
- B ... una dipendenza che esiste tra le possibili chiavi di una stessa tabella.
- C ... una dipendenza che esiste tra le chiavi di tabelle diverse.
- D Nessuna delle risposte precedenti.

**18** Una tabella viene detta in 3NF se ...

- A ... è in 1FN e vi sono solo dipendenze funzionali complete.
- B ... è in 1FN e vi sono solo dipendenze funzionali transitive.
- C ... è in 2FN e vi sono solo dipendenze funzionali transitive.
- D Nessuna delle risposte precedenti.

**19** Quali delle seguenti affermazioni relative alla BCNF sono vere?

- A Una relazione in 3NF è sicuramente anche in BCNF.
- B Una relazione in BCNF è sicuramente anche in 3NF.
- C Una relazione in BCNF è sicuramente anche in 2NF.
- D Nessuna delle risposte precedenti.

**20** Quali delle seguenti affermazioni sono vere relativamente a una transazione?

- A È una sequenza di operazioni che, se ha esito positivo, non trasforma la base di dati.
- B È una sequenza di operazioni viste come un'unica unità.
- C L'operazione di *rollback* viene applicata se essa fallisce.
- D L'operazione di *commit* viene applicata se essa fallisce.

**21** Relativamente a una transazione, quale delle seguenti è la proprietà che deve garantire la coerenza dei dati dopo che essa è terminata?

- A Atomicità.
- B Isolamento.

- C Persistenza.
- D Consistenza.

**22** Quali dei seguenti sono operatori derivati dell'algebra relazionale?

- A Prodotto cartesiano.
- B Intersezione.
- C Unione.
- D Congiunzione (*join*).

**23** Che cosa indica la proprietà di chiusura dell'algebra relazionale?

- A Che i suoi operatori si applicano a tabelle per produrre tabelle come risultato.
- B Che è un linguaggio non interfacciabile con altri linguaggi.
- C Che può essere utilizzato solo nel contesto di una base di dati relazionale.
- D Nessuna delle risposte precedenti.

**24** È ragionevole affermare che l'operazione di *join* è in qualche modo l'operazione inversa del processo di normalizzazione di una tabella?

- A Sì, perché è finalizzata a ricomporre tabelle risultato della scomposizione operata dalla normalizzazione.
- B No, perché l'operazione di *join* non è un'operazione di composizione di tabelle.
- C No, perché l'operazione di *join* effettua un prodotto cartesiano completo tra due tabelle.
- D Nessuna delle risposte precedenti.

**25** Volendo selezionare solo alcune colonne di una tabella, quale operatore dell'algebra relazionale è possibile utilizzare?

- A Selezione (RESTRICT).
- B Proiezione (PROJECT).
- C Congiunzione (JOIN).
- D Intersezione (INTERSECT).

**26** Quale operatore dell'algebra relazionale opera le seguenti trasformazioni?

A

C1	C2	C3	C4	C5		C1	C3	C5
a	b	c	d	e	→	a	c	e
f	g	h	i	j		f	h	j
k	l	m	n	o		k	m	o
p	q	r	s	t		p	r	t

B

C1	C2	C3	C4	C5		C1	C2	C3	C4	C5
a	b	c	d	e	→	a	b	c	d	e
a	b	x	y	z		a	b	x	y	z
o	p	q	r	s		v	w	x	y	z
a	i	j	k	l						
v	w	x	y	z						

C

A1	A2	B1	B2	B3		A1	A2	B1	B2	B3
a	b	a	x	y	→	a	b	a	x	y
c	d	a	b	c		a	b	a	b	c
e	f	i	j	k						

**27** L'operatore relazionale JOIN è dato dall'applicazione degli operatori ...

A ... RESTRICT e TIMES.

B ... RESTRICT e PROJECT.

C ... PROJECT e TIMES.

D Nessuna delle risposte precedenti.

## ESERCIZI

Gli esercizi dal numero 1 al numero 15 documentano l'analisi svolta su scenari che si intendono informatizzare; per ogni esercizio è richiesta la realizzazione di un diagramma E/R che modelli la realtà descritta a livello concettuale, lo schema delle relazioni che implementano il livello logico e la rappresentazione grafica delle relative tabelle con indicazione degli attributi, delle chiavi primarie ed esterne e delle associazioni.

- 1** Una banca deve gestire i dati relativi alle filiali. Per ogni filiale si devono registrare i seguenti

dati: codice, nome, città e patrimonio totale in euro. Ogni filiale gestisce un certo insieme di conti correnti. Ogni conto corrente è descritto dal numero del conto e dal suo saldo in euro (positivo o negativo); ogni conto corrente può avere uno o più intestatari (clienti), ognuno dei quali può essere intestatario di più di un conto, anche in filiali diverse. Per ogni cliente si registrano i seguenti dati: codice fiscale, nominativo, indirizzo, città e numero di telefono. Ogni filiale, inoltre, concede dei prestiti ai clienti (un prestito, come un conto corrente, può essere intestato a più di un cliente): un prestito è descritto da un codice identificativo, dal codice del cliente a cui è stato concesso, dal suo ammontare in euro, dal codice dell'ufficio che lo ha concesso, dalla matricola dell'impiegato che lo ha stipulato, dalla data di apertura e dalla data entro la quale esso dovrà essere estinto.


- 2** Un piccolo comune deve realizzare una base di dati relativa alle multe per violazioni al Codice della strada. Si intendono gestire dati relativi alle seguenti categorie:

- agenti, descritti da: matricola e nominativo;
- infrazioni, descritte da: codice infrazione, data, agente che l'ha elevata, denominazione dell'infrazione («divieto di sosta», «eccesso di velocità», ecc.), importo e targa dell'auto multata;
- auto, descritte da: targa, marca («Fiat», «Citroen», ecc.), modello («Cinquecento», «Picasso», ecc.), automobilista proprietario;
- automobilisti, descritti da: codice fiscale, nominativo, indirizzo, città, CAP.

- 3** È richiesto un database per organizzare alcune informazioni relative al campionato di calcio di serie A: dati anagrafici dei calciatori, dati delle squadre (nome, colori sociali, città), dati degli scudetti annuali con relativa squadra vincente e punteggio conseguito. Per ogni calciatore è necessario conoscere in quale squadra ha giocato e in quale ruolo («portiere», «difensore», «centrocampista», «attaccante», ecc.), in quale anno e quante reti ha realizzato (un calciatore, pur potendo aver ricoperto ruoli diversi in squadre diverse nel corso della propria carriera, nell'arco di un campionato ha ricoperto sempre un solo ruolo militando in un'unica squadra).


**4** Allo scopo di realizzare il database relativo agli esami ECDL di una scuola, i dati da gestire sono i seguenti:

- esami: data, ora e sede in cui sono stati effettuati o sono pianificati gli esami;
- tipologia di esame: tipo («elaborazione testi», «foglio elettronico», ecc.) e percentuale minima di risposte corrette necessarie per superare la prova;
- esaminandi: dati identificativi degli studenti che si sono iscritti almeno una volta a uno o più esami ECDL (codice della *skill-card*, nominativo, sesso, data di nascita, luogo di nascita, studente o meno della scuola);
- risultati: esiti per ogni esame, tipo di esame ed esaminando con la percentuale riportata.

 **5** Un albergo di una grande città intende gestire in modo automatizzato le prenotazioni e realizzare una base di dati. Ogni cliente viene individuato, tra l'altro, con i dati anagrafici, il numero di telefono e l'eventuale e-mail. Per quanto riguarda le prenotazioni occorre indicare il periodo, i dati relativi alle persone che soggiogneranno, il numero di camera assegnato, l'eventuale disdetta, il tipo di trattamento: mezza pensione (*Half Board*, HB), pensione completa (*Full Board*, FB), pernottamento e prima colazione (*Bed & Breakfast*, B&B).

**6** Un'autofficina intende informatizzare la gestione del proprio lavoro; a questo scopo necessita di un database che consenta di registrare le seguenti informazioni:


- dati relativi ai clienti e alle auto;
- dati relativi agli interventi prenotati ed effettuati con indicazione dell'auto, della durata in ore, della data e dei pezzi di ricambio utilizzati;
- dati relativi ai pezzi di ricambio con indicazione della quantità presente in officina;
- dati relativi ai fornitori dei pezzi di ricambio.

 **7** Un dipartimento di ricerca e sviluppo dispone di alcune decine di persone tra ricercatori dipendenti e consulenti esterni per lo svolgimento di vari progetti. A ogni progetto partecipano vari ricercatori/consulenti, ma una singola persona può essere comunque coinvolta anche contemporaneamente in più progetti. È richiesta la progettazione di un database relazionale che con-

senta la memorizzazione e l'interrogazione dei dati relativi ai ricercatori/consulenti (cognome/nome, sesso, data di nascita, retribuzione annua) e dei progetti (denominazione, data di inizio/fine, percentuale di avanzamento).

**8** Nel corso della manifestazione «World of coffee» la *Speciality Coffee Association of Europe* (SCAE) organizza una competizione internazionale di assaggio del caffè espresso. Alla competizione verranno presentati più di 100 campioni di caffè selezionati e prenderanno parte non meno di 25 assaggiatori qualificati: ogni singolo campione di caffè deve essere valutato, anche in momenti distinti, almeno da 10 assaggiatori diversi, ciascuno dei quali lo classifica con un punteggio centesimale. È richiesta la progettazione di un database relazionale che consenta la memorizzazione e l'interrogazione dei dati relativi ai campioni di caffè (denominazione e nazionalità, produttore e paese di produzione, torrefattore), agli assaggiatori (nome e cognome, nazionalità, matricola SCAE) e ai test dei singoli campioni effettuati dal singolo assaggiatore (data/ora e punteggio attribuito).

**9** L'ufficio centrale della Motorizzazione civile mantiene un registro di tutte le automobili italiane con i relativi proprietari (una persona può possedere più automobili, ma una singola automobile può anche appartenere a più persone) e un registro storico delle patenti rilasciate nel quale sono comprese anche le patenti ormai scadute ed eventualmente rinnovate con altro codice. È richiesta la progettazione di un database relazionale che consenta la memorizzazione e l'interrogazione dei dati relativi alle automobili e alle patenti.

 **10** È richiesto un database per la gestione del «registro di classe» dove, per ogni studente della scuola, di cui si conoscono i dati anagrafici e la classe che frequenta, sono memorizzate le seguenti informazioni:

- assenze (data, giustificate/ingiustificate);
- ingressi fuori orario (data, ora, motivazione);
- uscite fuori orario (data, ora, motivazione).

**11** Una facoltà universitaria, che attiva ogni anno accademico corsi didattici ai quali gli studenti si iscrivono per la frequenza, deve informatizzarne la gestione. Ogni corso è tenuto da un singolo docente della facoltà, ma un docente può tene-

re anche contemporaneamente più corsi. I corsi si concludono con il superamento di uno o più esami («presentazione progetto», «verifica scritta», «colloquio orale», ecc.) valutati mediante una votazione espressa in trentesimi: lo studente che fallisce una prova di esame può ripeterla in una diversa data.

**12** Deve essere realizzato un sistema informatico per gestire le informazioni necessarie a un amministratore di condomini. Ogni condominio è costituito da un insieme di appartamenti: ogni appartamento ha un inquilino e un proprietario che possono coincidere. L'amministratore effettua delle spese, ciascuna relativa a un condominio: le spese possono essere ordinarie, e come tali addebitabili agli inquilini, o straordinarie, e come tali addebitabili ai proprietari. L'addebito delle spese avviene come segue: ogni inquilino/proprietario ha un saldo che viene decrementato per le spese (ripartite tra gli inquilini e/o i proprietari) e decrementato a fronte di pagamenti, che devono essere singolarmente registrati nel sistema.

**13** Una grande libreria con più punti di vendita decide di offrire, esclusivamente ai propri clienti registrati, un servizio di acquisto elettronico online con inoltro a mezzo corriere presso l'indirizzo del cliente di una selezione dei propri prodotti costituiti da libri, CD musicali e video DVD. Ogni cliente dispone di una somma a credito per gli acquisti elettronici versata preventivamente presso le casse della libreria: gli acquisti possono essere effettuati esclusivamente nei limiti di tale somma. Il singolo acquisto di uno specifico cliente è identificato da un numero d'ordine ed è costituito da uno o più prodotti tra quelli disponibili, ciascuno eventualmente in più copie: il costo del singolo acquisto viene calcolato come somma dei costi dei singoli prodotti più un costo di spedizione proporzionale al peso del pacco da inviare. Lo stato dell'ordine (ricevuto, impacchettato, inoltrato, consegnato) deve essere consultabile online dal cliente che lo ha effettuato: deve essere registrata la data-ora di ricezione, di inoltro al corriere e di consegna al cliente per ogni singolo ordine. Il sito Internet del servizio riporta per ogni singolo articolo in vendita (libro, CD o DVD) il nome dell'autore/compositore/esecutore, il titolo del libro/disco/film e il nome dell'editore/produttore/distributore,

l'immagine fotografica del prodotto e alcune sintetiche informazioni di consultazione.

**14** Un'agenzia espressi deve organizzare il proprio servizio in una grande città aprendo più sedi da dove sarà possibile spedire plichi o pacchi urgenti da recapitare a un qualsiasi indirizzo della città. Alla consegna per la spedizione viene individuata la sede più vicina all'indirizzo di destinazione: il plico o pacco verrà prima consegnato a questa sede, tramite un servizio di trasporto che avviene più volte in un giorno, e poi al destinatario finale da parte di un fattorino.

Per ogni plico o pacco devono essere memorizzati:

- codice a barre applicato;
- ragione sociale e indirizzo del mittente;
- sede dell'agenzia che riceve la spedizione;
- data e ora di ricezione della spedizione;
- nome dell'addetto che riceve la spedizione;
- ragione sociale e indirizzo del destinatario;
- sede dell'agenzia di destinazione;
- data e ora di inoltro all'agenzia di destinazione;
- identificativo del mezzo di inoltro a destinazione;
- data e ora di arrivo all'agenzia di destinazione;
- data e ora di consegna al destinatario;
- nome del fattorino che effettua la consegna;
- peso del plico/pacco;
- importo della spedizione.

**15** Si vuole realizzare il CUP (Centro Unico di Prenotazione) di una ASL in base alle informazioni che lo caratterizzano e che sono descritte di seguito. Esiste una anagrafica dei medici convenzionati: per ogni medico, vogliamo gestire il suo numero di codice, il nome e il cognome, l'indirizzo, il telefono e il reparto ospedaliero in cui opera; ogni medico è uno specialista di cui si vuole considerare la/le specialità. Delle varie specialità si vuole gestire un elenco dove queste ultime sono inventariate: a fronte di ogni specialità si deve gestire un prontuario di possibili prestazioni mediche, per ognuna delle quali si prevede un certo ticket di spesa. Per ogni medico si conosce un calendario annuo dove in specifici giorni della settimana egli presta la propria opera per un certo numero di ore (il calendario viene preparato preventivamente e si prevede un numero massimo di pazienti al giorno). Dei pazienti della ASL si conoscono

le informazioni seguenti: numero di codice sanitario, nome e cognome, sesso, data di nascita, indirizzo ed eventuale numero di telefono. Un paziente può richiedere una prestazione specialistica con uno specifico medico per un determinato giorno: il CUP deve essere in grado di verificare la disponibilità di quel medico in quel dato giorno, oppure di proporre un altro medico nel giorno indicato, o ancora il primo giorno disponibile per il medico desiderato. A fronte di ogni prenotazione si vuole mantenere traccia delle prestazioni effettivamente prestate: si provvederà a registrare i singoli eventi indicandone data, ora di inizio e ora di fine, la patologia riscontrata (deve essere prevista una tabella specifica per i vari tipi di patologia), la prestazione effettuata, le eventuali prescrizioni e la data indicativa per un eventuale controllo, il pagamento del ticket o meno nel caso il paziente fosse esentato.

**16** Con riferimento alla base di dati progettata nell'esercizio 2, scrivere utilizzando gli operatori dell'algebra relazionale le seguenti interrogazioni:

- elenco delle infrazioni elevate a partire dal 1° gennaio 2000;
- elenco degli agenti che hanno elevato almeno una infrazione;
- elenco degli automobilisti con i dati dell'auto con la quale hanno subito infrazioni;
- elenco che riporta data dell'infrazione, nome dell'agente che l'ha elevata, targa, modello e marca dell'auto multata.

**17** Con riferimento alla base di dati progettata nell'esercizio 5, scrivere utilizzando gli operatori dell'algebra relazionale le seguenti interrogazioni:

- lista delle camere prenotate in un determinato giorno;
- elenco dei clienti che hanno prenotato in un certo giorno dell'anno;

- elenco dei clienti che hanno dato disdetta in un certo giorno dell'anno.

**18** Con riferimento alla base di dati progettata nell'esercizio 7, scrivere utilizzando gli operatori dell'algebra relazionale le seguenti interrogazioni:

- elenco in ordine alfabetico dei soli ricercatori;
- elenco delle persone assegnate a uno specifico progetto con indicazione del sesso.

**19** Con riferimento alla base di dati progettata nell'esercizio 8, scrivere utilizzando gli operatori dell'algebra relazionale le seguenti interrogazioni:

- elenco in ordine alfabetico dei caffè di origine brasiliana iscritti alla competizione;
- elenco dei torrefattori di caffè iscritti alla competizione con indicazione della nazionalità;
- elenco dei caffè testati da uno specifico assaggiatore.

**20** Con riferimento alla tabella «Voli», riportata in basso e relativa ai voli nazionali giornalieri di una certa compagnia aerea:

**Voli** (ora\_volo, tratta, partenza, arrivo, aereo, posti)

- individuare in essa le dipendenze funzionali;
- proporre una riduzione di 3NF;
- verificare se la 3NF individuata è anche in BCNF.

**21** Con riferimento alla tabella «Ordini», riportata alla pagina successiva in alto e relativa a ordini di prodotti effettuati da clienti a una certa azienda:

**Ordini** (cliente, indirizzo, articolo, descrizione, prezzo, n\_ordine, quantità)

sapendo che gli ordini sono numerati progressivamente per cliente e che in uno stesso ordine possono essere ordinati solo articoli diversi in quantità specificata,

- individuare in essa le dipendenze funzionali;

ESERCIZIO 20 Tabella «Voli»

ora_volo	tratta	partenza	arrivo	aereo	posti
07.00	T1	Milano	Napoli	A01	90
07.00	T2	Roma	Milano	A05	80
12.30	T1	Milano	Napoli	A02	70
10.00	T2	Roma	Milano	A01	90
10.00	T3	Roma	Napoli	A07	85
11.30	T1	Milano	Napoli	A05	80



ESERCIZIO 21 Tabella «Ordini»

cliente	indirizzo	articolo	descrizione	prezzo	n_ordine	quantita
C1	I1	A1	D1	15	1	50
C1	I1	A2	D2	10	1	50
C1	I1	A1	D1	15	2	100
C1	I1	A3	D3	5	3	50
C2	I2	A2	D2	10	1	60
C2	I2	A4	D4	20	2	100
C3	I3	A1	D1	15	1	60

- proporre una riduzione di 3NF;
- verificare se la 3NF individuata è anche in BCNF.

**22** Una palestra ospita diversi corsi appartenenti a diverse tipologie di arti marziali (karate, judo, ju-jitsu, ecc.). Ogni corso ha una sigla, che lo identifica, un insegnante e alcuni allievi. Un insegnante offre in generale più corsi, anche di diverse tipologie, e anche un allievo può essere iscritto a più corsi. Di ogni insegnante interessano il nome (che lo identifica) e l'indirizzo. Di ogni allievo interessano il nome (che lo identifica) e il numero di telefono. Per ogni allievo interessa sapere, per ogni corso che frequenta, quanto ha già versato finora. La palestra gestisce attualmente i dati con un'unica tabella, strutturata come quella riportata in basso:

- individuare in essa le dipendenze funzionali;
- proporre una riduzione di 3NF;
- verificare se la 3NF individuata è anche in BCNF.

**23** Si considerino i seguenti schemi relazionali:



**Vendite** (Commesso, Negozio, Città, Data, CodiceArticolo, Taglia, Colore)

**Articoli** (CodiceArticolo, Taglia, Colore, Prezzo)

Supponendo che

- un certo commesso lavori in un solo negozio;

- uno specifico negozio si trovi in una sola città;
- un dato prodotto abbia sempre lo stesso prezzo;
- ogni prodotto sia disponibile in più taglie e colori;

- individuare le dipendenze funzionali;
- determinare le chiavi delle due relazioni;
- verificare se lo schema è in FNBC, altrimenti lo si decomponga.

**24** Sono dati i seguenti schemi relazionali che rappresentano la gestione dei laboratori scolastici di informatica:

**Computer** (idComputer, marca, modello, fornitore, telefono\_fornitore)

**Installazione** (idComputer, idSoftware, descrizione\_software, data\_installazione)

Supponendo che

- computer di un certo modello siano identici e identificati dal loro codice di inventario,
- i computer di una stessa marca abbiano tutti lo stesso fornitore,
- uno stesso software sia in genere installato su più computer;

- individuare le dipendenze funzionali,
- verificare se lo schema è in FNBC, altrimenti lo si decomponga.

ESERCIZIO 22 Tabella «Gestione dati»

Allievo	Telefono	Corso	TipoCorso	Saldo	Insegnante	Indirizzo
Matteo	303130	C1	Ju-jitsu	250.00	Giovanni	Via Roma
Pietro	304140	C1	Ju-jitsu	200.00	Giovanni	Via Roma
Matteo	303130	C4	Karate	300.00	Marco	Via Bini
Andrea	303130	C3	Judo	300.00	Luca	Via Roma
Pietro	304140	C7	Judo	250.00	Giovanni	Via Roma

### 1.3 A Relational View of Data

The term *relation* is used here in its accepted mathematical sense. Given sets  $S_1, S_2, \dots, S_n$  (not necessarily distinct),  $R$  is a relation on these  $n$  sets if it is a set of  $n$ -tuples each of which has its first element from  $S_1$ , its second element from  $S_2$ , and so on. We shall refer to  $S_j$  as the  $j$ th *domain* of  $R$ . As defined above,  $R$  is said to have *degree*  $n$ . Relations of degree 1 are often called *unary*, degree 2 *binary*, degree 3 *ternary*, and degree  $n$  *n-ary*.

For expository reasons, we shall frequently make use of an array representation of relations, but it must be remembered that this particular representation is not an essential part of the relational view being expounded. An array which represents an  $n$ -ary relation  $R$  has the following properties:

- (1) Each row represents an  $n$ -tuple of  $R$ .
- (2) The ordering of rows is immaterial.
- (3) All rows are distinct.
- (4) The ordering of columns is significant – it corresponds to the ordering  $S_1, S_2, \dots, S_n$  of the domains on which  $R$  is defined (see, however, remarks below on domain-ordered and domain-unordered relations).
- (5) The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

The example in Figure 1 illustrates a relation of degree 4, called *supply*, which reflects the shipments-in-progress of parts from specified suppliers to specified projects in specified quantities.

<i>supply</i>	<i>(supplier</i>	<i>part</i>	<i>project</i>	<i>quantity)</i>
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

FIG. 1. A relation of degree 4.

One might ask: if the columns are labeled by the name of corresponding domains, why should the ordering of columns matter? As the example in Figure 2 shows, two columns may have identical headings (indicating identical domains) but possess distinct meanings with respect to the relation. The relation depicted is called *component*. It is a ternary relation, whose first two domains are called *part* and third domain is called *quantity*. The meaning of *component*  $(x, y, z)$  is that part  $x$  is an immediate component (or subassembly) of part  $y$ , and  $z$  units of part  $x$  are needed to assemble one unit of part  $y$ . It is a relation which plays a critical role in the parts explosion problem.

<i>component</i>	<i>(part</i>	<i>part</i>	<i>quantity)</i>
	1	5	9
	2	5	7
	3	5	2
	2	6	12
	3	6	3
	1	7	1
	6	7	1

FIG. 2. A relation with two identical domains.

It is a remarkable fact that several existing information systems (chiefly those based on tree-structured files) fail to provide data representations for relations which have two or more identical domains. [...]

The totality of data in a data bank may be viewed as a collection of time-varying relations. These relations are of assorted degrees. As time progresses, each  $n$ -ary relation may be subject to insertion of additional  $n$ -tuples, deletion of existing ones, and alteration of components of any of its existing  $n$ -tuples. [...]

Users should not normally be burdened with remembering the domain ordering of any relation (for example, the ordering *supplier*, then *part*, then *project*, then *quantity* in the rela-

**conveyed**  
comunicato, espresso

**to matter**  
importare

**depicted**  
raffigurata

**subassembly**  
sottogruppo

**remarkable**  
notevole

**shipment in progress**  
spedizione in corso

**time-varying**  
variante nel tempo

**burdened**  
gravato

**to sum up**  
riassumere

**counterpart**  
omologo

**to cross-reference**  
riferirsi a, consultare

**cross-reference**  
legame, rimando



tion *supply*). Accordingly, we propose that users deal, not with relations which are domain-ordered, but with *relationships* which are their domain-unordered counterparts. To accomplish this, domains must be uniquely identifiable at least within any given relation, without using position. Thus, where there are two or more identical domains, we require in each case that the domain name be qualified by a distinctive *role name*, which serves to identify the role played by that domain in the given relation. For example, in the relation *component* of Figure 2, the first domain *part* might be qualified by the role name *sub*, and the second by *super*, so that users could deal with the relationship *component* and its domains – *sub.part* *super.part*, *quantity* – without regard to any ordering between these domains.

To sum up, it is proposed that most users should interact with a relational model of the data consisting of a collection of time-varying relationships (rather than relations). Each user need not know more about any relationship than its name together with the names of its domains (role qualified whenever necessary). [...]

There are usually many alternative ways in which a relational model may be established for a data bank. In order to discuss a preferred way (or normal form), we must first introduce a few additional concepts (active domain, primary key, foreign key, [...] nonsimple domain) and establish some links with terminology currently in use in information systems programming. [...]

Consider an example of a data bank which includes relations concerning parts, projects, and suppliers. One relation called *part* is defined on the following domains:

- (1) part number
- (2) part name
- (3) part color
- (4) part weight
- (5) quantity on hand
- (6) quantity on order

and possibly other domains as well. Each of these domains is, in effect, a pool of values, some or all of which may be represented in the data bank at any instant. While it is conceivable that, at some instant, all part colors are present, it is unlikely that all possible part weights, part names, and part numbers are. We shall call the set of values represented at some instant the *active domain* at that instant.

Normally, one domain (or combination of domains) of a given relation has values which uniquely identify each element (*n*-tuple) of that relation. Such a domain (or combination) is called a *primary key*. In the example above, part number would be a primary key, while part color would not be. A primary key is *nonredundant* if it is either a simple domain (not a combination) or a combination such that none of the participating simple domains is superfluous in uniquely identifying each element. A relation may possess more than one nonredundant primary key. This would be the case in the example if different parts were always given distinct names. Whenever a relation has two or more nonredundant primary keys, one of them is arbitrarily selected and called *the* primary key of that relation.

A common requirement is for elements of a relation to cross-reference other elements of the same relation or elements of a different relation. Keys provide a user-oriented means (but not the only means) of expressing such cross-references. We shall call a domain (or domain combination) of relation *R* a *foreign key* if it is not the primary key of *R* but its elements are values of the primary key of some relation *S* (the possibility that *S* and *R* are identical is not excluded). In the relation *supply* of Figure 1, the combination of *supplier*, *part*, *project* is the primary key, while each of these three domains taken separately is a foreign key. [...]

[E.F. Codd, *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, vol. 13, number 6, June, 1970]

## QUESTIONS

- a What is a *relation* in the relational data model?
- b What is a *primary key* in the relational data model?
- c What is a *foreign key* in the relational data model?
- d What is the *degree* of a relation?

# Il linguaggio SQL

# A3

Dato il seguente schema di una base di dati relazionale relativa a voli aerei:

Voli(id\_volo, partenza, destinazione, orario, data\_volo)

Passeggeri(id\_volo, n\_posto, nominativo, sesso, data\_nascita)

il cui diagramma delle tabelle è schematizzato in FIGURA 1

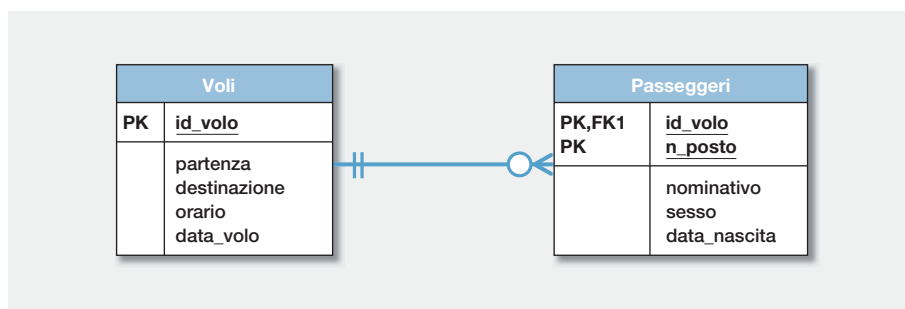


FIGURA 1

la query

```
PROJECT (
  RESTRICT (
    JOIN Voli WITH Passeggeri ON id_volo
  ) WHERE id_volo='AZ215' AND sesso='M'
) ON data_volo,nominativo,n_posto
```

espressa in algebra relazionale seleziona per ogni passeggero maschio del database il nominativo, il posto occupato e la data del volo di tutti i voli con la sigla 'AZ215'.

Essa può essere così formulata in linguaggio SQL:

```
SELECT data_volo,nominativo,n_posto
FROM Voli,Passeggeri
WHERE id_volo='AZ215' AND sesso='M'
AND Voli.id_volo = Passeggeri.id_volo
```

Il linguaggio SQL permette di ricercare, inserire, modificare e cancellare dati e di operare con funzioni gestionali e amministrative su database.

## SQL

SQL deriva dal linguaggio SEQUEL (*Structured English QUery Language*) implementato nel 1974 da Donald Chamberlin nell'ambito del DBMS SYSTEM-R, prototipo di DBMS relazionale sviluppato da IBM.

Nelle intenzioni del progettista, esso doveva essere un linguaggio più semplice e compatto dell'algebra relazionale per formulare query su database relazionali.

Attualmente SQL è lo strumento più diffuso per interrogare e gestire basi di dati, esso è stato standardizzato da ANSI nel 1986 (SQL-86); nel corso degli anni sono state rilasciate versioni successive, riconosciute anche da ISO.

La maggior parte dei sistemi per la gestione di database implementano questi standard aggiungendo proprie funzionalità. Pensato in origine come linguaggio dichiarativo, si è evoluto con l'introduzione di costrutti procedurali, istruzioni per il controllo di flusso, tipi di dati definiti dall'utente e varie altre estensioni.

A partire dalla definizione dello standard SQL:1999 (conosciuto anche come SQL3) molte di queste estensioni sono state formalmente adottate come parte integrante di SQL.

## Uso di SQL

La maggior parte delle implementazioni dei DBMS permettono l'esecuzione diretta dei comandi SQL da riga di comando in alternativa alle interfacce di tipo grafico.

Il linguaggio SQL può essere usato sia come linguaggio autonomo, che ospitato in altri linguaggi di programmazione (Java, C++, ecc.).

Quest'ultima caratteristica permette lo sviluppo di applicazioni complesse dove il linguaggio ospitante mette a disposizione le sue caratteristiche per l'implementazione dell'interfaccia utente e delle procedure di elaborazione, mentre a SQL sono demandate le funzionalità di accesso e gestione dei dati su cui tali applicazioni operano.

In generale si classificano i comandi SQL in due gruppi fondamentali: DML e DDL. Nella trattazione che segue prenderemo in considerazione tre gruppi:

- comandi per l'interrogazione dei dati (come gruppo separato anche se in effetti fa parte del DML);
- comandi per la modifica dei dati (DML);
- comandi per la definizione della struttura dei dati e l'amministrazione del database (DDL).

Il linguaggio SQL come tutti i *relational query languages*, ha la proprietà di «chiusura», cioè opera su tabelle per produrre risultati che sono ancora tabelle.

**OSSERVAZIONE** Alcune delle critiche più frequenti rivolte a SQL riguardano la scarsa portabilità del codice fra DBMS diversi a causa dei differenti modi di trattare i dati mancanti (**NULL**), e la semantica a volte inutilmente complicata.

Presentiamo qui un semplice scenario a cui faremo riferimento più volte in questo capitolo.

La situazione è quella di un'azienda divisa in dipartimenti aventi ciascuno un certo numero di impiegati (personale). A ogni dipartimento è affidata la realizzazione di determinati prodotti ottenuti assemblando alcuni componenti forniti dall'esterno:

- **un dipartimento** è descritto da codice (`id_dip`), dal suo nome (`nome_dipartimento`), dalla località in cui si trova (`localita`) e dalla provincia (`provincia`);
- **un impiegato** è descritto da un numero di matricola (`matricola`), dal cognome e nome (`nominativo`), dalla data di nascita (`data_nascita`), da una categoria lavorativa (`qualifica`) e dallo stipendio percepito (`stipendio`);
- **un prodotto** è descritto da un codice (`id_prod`), un nome (`nome_prodotto`) e dal suo prezzo di vendita (`prezzo`);
- **un componente** è descritto da un codice (`id_comp`), un nome (`nome_componente`), dal suo costo unitario (`costo_unitario`) e dal nome del suo fornitore (`fornitore`).

Tra queste entità si osservano le seguenti corrispondenze:

- **dipartimenti : personale** (1:N);
- **dipartimenti : prodotti** (1:N);
- **prodotti : componenti** (M:N).

Inoltre è di interesse l'informazione che descrive il numero di unità di un certo componente che sono impiegate per la realizzazione di un particolare prodotto.

Quello che segue è uno schema di database che rappresenta la situazione descritta:

**OSSERVAZIONE** La tabella *Composizione* è stata inserita per spezzare l'associazione  $M:N$  tra *Prodotti* e *Componenti* nelle due associazioni di tipo  $1:N$  tra *Prodotti* e *Composizione* e *Componenti* e *Composizione*. L'informazione relativa che descrive il numero di unità di un certo componente che sono impiegate per la realizzazione di un particolare prodotto è stata necessariamente inserita nella tabella *Composizione*: la chiave primaria di questa tabella è formata dalle chiavi primarie delle tabelle con cui essa è in associazione.

**OSSERVAZIONE** Nel corso del presente capitolo, per la trattazione dei comandi del linguaggio SQL si farà riferimento prevalentemente alle caratteristiche del DBMS MySQL e per illustrarne la sintassi sarà utilizzata la seguente simbologia:

Diagramma ER di un database per la gestione di prodotti e componenti. Le tabelle sono:

- Personale**: attributi matricola (PK), id\_dip, nominativo, data\_nascita, qualifica, stipendio.
- Dipartimenti**: attributi id\_dip (PK), nome\_dipartimento, localita, provincia.
- Prodotti**: attributi id\_prod (PK), id\_dip (FK1), nome\_prodotto, prezzo.
- Composizione**: attributi id\_prod (PK, FK1), id\_comp (PK, FK2), unita\_comp.
- Componenti**: attributi id\_comp (PK), nome\_componente, costo\_unitario, fornitore.

Le relazioni sono:

- Personale a Dipartimenti**: relazione 1:M. La chiave primaria di Personale (matricola) è collegata alla chiave primaria di Dipartimenti (id\_dip).
- Dipartimenti a Prodotti**: relazione 1:M. La chiave primaria di Dipartimenti (id\_dip) è collegata alla chiave primaria di Prodotti (id\_prod).
- Composizione a Prodotti**: relazione 1:M. La chiave primaria di Composizione (id\_prod) è collegata alla chiave primaria di Prodotti (id\_prod).
- Composizione a Componenti**: relazione 1:M. La chiave primaria di Composizione (id\_comp) è collegata alla chiave primaria di Componenti (id\_comp).

93

## MySQL

Il DBMS MySQL nasce negli anni '80 del secolo scorso, anche se il linguaggio SQL verrà effettivamente supportato a partire dal 1996, come prodotto della società svedese MySQL AB che ne prevede da subito la distribuzione sia con licenza GNU GPL sia con licenza commerciale.

Nel gennaio 2008 Sun Microsystems ha acquistato la società per un miliardo di dollari, stimando il mercato del DBMS in 15 miliardi di dollari.

Nell'aprile 2009 alla stessa Sun Microsystems è stata proposta l'acquisizione da parte di Oracle Corporation per 7,4 miliardi di dollari.

L'accordo, approvato dall'antitrust USA, è poi passato al vaglio degli organi corrispondenti dell'Unione Europea, preoccupati dal conflitto di interessi costituito dai DBMS commerciali Oracle rispetto a MySQL.

Il padre di MySQL, Michael Widenius, ha lanciato una petizione online per opporsi alla fusione. Nonostante ciò l'Unione Europea ha dato parere favorevole, e l'acquisizione è stata perfezionata nel gennaio del 2010.

Oracle Corporation fino a oggi ha mantenuto lo sviluppo del DBMS con doppia licenza, open-source e commerciale.

Per esempio, nella forma generale del comando "**SELECT**"

```
SELECT <lista_campi>
FROM <lista_tabelle>
[WHERE <condizione>];
```

<lista\_campi> indica un elenco di campi di una tabella,

<lista\_tabelle> indica un elenco di tabelle del database,

[**WHERE** <condizione>] indica una parte opzionale del comando che nel caso specifico prevede, se presente, la formulazione di una condizione esprimibile sui campi delle tabelle coinvolte.

Un'espressione del tipo

```
...
{ INSERT | UPDATE | DELETE }
...
```

indica l'alternativa tra l'uso dei tre comandi separati dal simbolo «|».

Una istruzione SQL termina con il carattere «;» anche se molti DBMS ne accettano l'omissione.

Anche se su molte piattaforme MySQL non è *case-sensitive* rispetto ai nomi di database e tabelle, è necessario, nel contesto di uno stesso comando, riferirsi ai nomi, se questi occorrono più volte, usando sempre la stessa modalità di capitalizzazione. Per esempio il seguente comando:

```
SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

non è corretto perché riferisce la stessa tabella usando due espressioni diverse: `my_table` e `MY_TABLE`.

## 1 Il comando **SELECT** e l'algebra relazionale

L'operazione fondamentale del linguaggio SQL è la *mapping*, rappresentato sintatticamente dal costrutto **SELECT-FROM-WHERE** la cui forma base generale è la seguente:

```
SELECT <lista_campi>
FROM <lista_tabelle>
[WHERE <condizione>];
```

Il comando **SELECT** del linguaggio SQL realizza un'interrogazione (o query) della base di dati restituendo come risultato una tabella virtuale avente come colonne i campi specificati dopo la parola chiave **SELECT**.

Con il comando:

```
SELECT nome_prodotto, prezzo
FROM Prodotti
WHERE id_dip = 'D2';
```

viene selezionato il nome e il prezzo di tutti i prodotti realizzati dal dipartimento D2.

Da questo esempio si comprende come il *mapping* consista, in termini di algebra relazionale, in un'operazione di restrizione (la selezione delle righe di *Prodotti* in cui il campo *id\_dip* assume il valore D2) seguita da una proiezione (estrazione dei valori assunti da *nome\_prodotto* e *prezzo* dalle righe trovate in precedenza).

La proiezione può essere estesa all'intera tabella per estrarne righe complete, usando la clausola **SELECT** *\**.

Riprendiamo adesso l'esempio di apertura del capitolo per analizzare nel dettaglio le analogie tra algebra relazionale e linguaggio SQL:

```
PROJECT (
  RESTRICT (
    JOIN Volo WITH Passeggeri
          ON id_volo
  ) WHERE id_volo='AZ215'
    AND sesso='M'
  )
  ON data_volo, nominativo, n_posto
```

```
SELECT data_volo, nominativo, n_posto
FROM Voli, Passeggeri
WHERE id_volo='AZ215'
  AND sesso='M'
  AND Volo.id_volo = Passeggeri.id_volo
```

Si osservi che:

- la <lista\_campi> (*Data\_Volo*, *Nominativo*, *NPosto*) che segue la parola chiave **SELECT** realizza l'operatore **PROJECT** dell'algebra relazionale;
- la <lista\_tabelle> (*Volo*, *Passeggero*) che segue la parola chiave **FROM** indica le tabelle coinvolte nella query e, se queste sono più di una, realizza di fatto l'operatore **JOIN** dell'algebra relazionale;
- le singole componenti della condizione espressa nella clausola **WHERE** rappresentano il criterio di selezione espresso dall'operatore **RESTRICT** dell'algebra relazionale (*Id\_volo*='AZ215' **AND** *Sesso*='M') e se, come in questo caso, le tabelle coinvolte sono più di una, il criterio di *join* sulla base del quale debbono essere legate le righe di tali tabelle (*Volo*.*Id\_volo* = *Passeggero*.*Id\_volo*).

Nella formulazione delle condizioni possono essere utilizzati i classici operatori di relazione =, >, <, >=, <=, <> e gli operatori booleani **AND**, **OR** e **NOT**. L'uso dei simboli «(» e «)» consente di stabilire la precedenza nella valutazione delle singole condizioni elementari nel contesto della condizione generale.

Le costanti di tipo stringa debbono essere racchiuse tra simboli «'» (alcuni DBMS accettano anche il simbolo «"»).

**OSSERVAZIONE** Una singola condizione di una clausola **WHERE** può specificare un insieme di valori a cui deve soddisfare uno stesso campo.

**ESEMPIO**

Volendo conoscere il nome e la qualifica di tutti i dipendenti dei dipartimenti D2, D4 e D5, si può formulare la seguente interrogazione:

```
SELECT nominativo, qualifica
FROM Personale
WHERE id_dip IN ('D2','D4','D5');
```

che è equivalente a:

```
SELECT nominativo, qualifica
FROM Personale
WHERE id_dip = 'D2' OR id_dip = 'D4' OR id_dip = 'D5';
```

Si noti che premettendo alla clausola **IN** l'operatore booleano **NOT** è possibile fare riferimento al complementare dell'insieme specificato. La seguente interrogazione:

```
SELECT nominativo, qualifica
FROM Personale
WHERE id_dip NOT IN ('D2','D4','D5');
```

recupera il nome e la qualifica di tutti i dipendenti che non appartengono ai dipartimenti D2, D4 e D5.

La costante **NULL** può essere usata nella formulazione di condizioni per selezionare righe in cui alcuni campi non sono stati avvalorati.

**ESEMPIO**

La query che segue

```
SELECT *
FROM Personale
WHERE qualifica IS NULL;
```

equivale a

```
SELECT *
FROM Personale
WHERE ISNULL(qualifica);
```

e permette la selezione delle righe dalla tabella *Personale* relative agli impiegati a cui non sia stata assegnata una qualifica specifica.

Le righe che risultano da una interrogazione vengono restituite secondo un ordinamento determinato automaticamente dal sistema a meno che non si specifichi esplicitamente un criterio di ordinamento.

Per esempio per ottenere la matricola, il nominativo e lo stipendio degli impiegati del dipartimento D4 in ordine di matricola, si deve formulare la seguente query:

```
SELECT matricola,nominativo,stipendio
FROM Personale
WHERE id_dip = 'D4'
ORDER BY matricola;
```

Il criterio di ordinamento può essere espresso su più campi con la possibilità di dichiarare se l'ordinamento debba essere ascendente («ASC») o discendente («DESC»); se quest'ultima opzione non viene esplicitata come ordinamento predefinito viene assunto quello ascendente.

#### ESEMPIO

Per ottenere la matricola, il nominativo e lo stipendio degli impiegati del dipartimento D1, in ordine ascendente di provincia e, nell'ambito della stessa provincia, discendente di stipendio, si ricorre alla seguente interrogazione:

```
SELECT provincia,matricola,nominativo,stipendio
FROM Personale
WHERE id_dip = 'D1'
ORDER BY provincia, stipendio DESC;
```

**OSSERVAZIONE** Il risultato di una query potrebbe fornire righe duplicate, la duplicazione può essere evitata premettendo alla lista dei campi la parola chiave **DISTINCT**.

#### ESEMPIO

La seguente query seleziona i codici dei prodotti presenti nella tabella *Composizione* riportando ogni codice di prodotto una sola volta:

```
SELECT DISTINCT id_prod
FROM Composizione;
```

Senza la clausola **DISTINCT** ogni codice prodotto sarebbe stato ripetuto tante volte quanti sono i suoi componenti.

Per quanto riguarda le condizioni espresse su campi di tipo stringa il linguaggio SQL mette a disposizione l'operatore **LIKE** che usato insieme al carattere *jolly* «%» permette di effettuare ricerche relative al contenuto di tali campi (*pattern-matching*).

#### ESEMPIO

Volendo estrarre tutti i dati di tutti i dipendenti il cui cognome inizia per «Ros» («Rossi», «Rossini», «Rossetti», ecc.) si usa la seguente query:

```
SELECT *
FROM Personale
WHERE Nominativo LIKE 'Ros%';
```





Allo stesso modo, la query

```
SELECT *  
FROM Personale  
WHERE Nominativo LIKE '%aria';
```

permette di estrarre i dati di coloro in cui il contenuto del campo *Nominativo* termina per «aria» (*Maria*, *Ilaria* ecc.).

Infine la query

```
SELECT *  
FROM Personale  
WHERE Nominativo LIKE '%erto%';
```

permette di estrarre i dati di coloro in cui il campo *Nominativo* contenga in un punto qualsiasi la sequenza «erto» (*Roberto*, *Bertolini* ecc.).

Un altro operatore reso disponibile dal linguaggio SQL per la formulazione delle condizioni è **BETWEEN**; esso permette di confrontare il valore di un campo con un intervallo di valori nella forma:

<campo> **BETWEEN** <valore\_iniziale> **AND** <valore\_finale>

equivalente a

<campo> >= <valore\_iniziale> **AND** <campo> <= <valore\_finale>

#### ESEMPIO

Volendo estrarre tutti i dati dei dipendenti nati tra il 1960 e il 1969 si può operare come segue:

```
SELECT *  
FROM Personale  
WHERE data_nascita BETWEEN '1960-01-01' AND '1969-12-31';
```

Oppure utilizzando la funzione **YEAR** che estrae l'anno da un campo di tipo data:

```
SELECT *  
FROM Personale  
WHERE YEAR(data_nascita) BETWEEN 1960 AND 1969;
```

**OSSERVAZIONE** Le costanti data usate per delimitare l'intervallo temporale sono state espresse come stringhe nel formato AAAA-MM-GG (quattro cifre per l'anno, due per il mese e due per il giorno separate dal simbolo «-»), mentre nel caso dell'anno i valori di confronto sono stati espressi come numeri interi.

La lista di campi può comprendere anche campi definiti in maniera estemporanea sulla base di elaborazioni effettuate sui campi effettivamente contenuti nelle tabelle.

Per ottenere un elenco relativo alla matricola, al nominativo e all'età dei dipendenti del dipartimento D1, ordinato dal più anziano al più giovane, si può realizzare la seguente query:

```
SELECT matricola,nominativo,
       YEAR(CURRENT_DATE())-YEAR(data_nascita) AS eta
FROM Personale
WHERE id_dip='D1'
ORDER BY eta DESC;
```

La seguente query permette di visualizzare la matricola, il nominativo, lo stipendio e una eventuale gratifica del 10% da corrispondere ai dipendenti del dipartimento D1 con qualifica Q6:

```
SELECT matricola,nominativo,stipendio,stipendio*0.1 AS gratifica
FROM Personale
WHERE id_dip='D1' AND qualifica='Q6';
```

Ovviamente il tempo di vita dei campi virtuali `eta` e `gratifica` si esaurisce con l'esecuzione della query stessa in quanto essi non sono memorizzati nel database. Si noti come la parola chiave **AS** permetta di assegnare un nome a questi campi (in assenza di tale specifica il DBMS assegna un nome fittizio a tali campi).

La parola chiave **AS** può essere utilizzata anche per cambiare, nel contesto di una query, il nome di un campo con un nome diverso; ad esempio:

```
SELECT nominativo AS cognome_nome, stipendio
FROM Personale;
```

Il risultato di una query è sempre una tabella virtuale che non viene memorizzata nel database. È comunque possibile (se si hanno i necessari privilegi come utente del DBMS) memorizzare la tabella ottenuta nel database utilizzando la clausola **INTO**.

La seguente query permette di memorizzare nel database i dati relativi ai dirigenti intesi come dipendenti aventi la qualifica Q7:

```
SELECT * INTO Dirigenti
FROM Personale
WHERE qualifica='Q7';
```

Questo tipo di operazione, che crea una nuova tabella, è in genere riservato all'amministratore della base di dati (DBA).

Il DBMS MySQL non supporta la clausola **INTO** per cui per ottenere lo stesso risultato è necessario utilizzare il seguente comando equivalente:

```
CREATE TABLE Dirigenti
SELECT *
FROM Personale
WHERE qualifica='Q7';
```

## 2 La chiusura del linguaggio SQL e le query nidificate; *join* e *self-join*

Con la proprietà di chiusura del linguaggio SQL è possibile usare il risultato di una query come argomento della clausola **WHERE** di un'altra query (subquery) realizzando di fatto delle query nidificate: una subquery può fare uso a sua volta di altre subquery e il risultato lo si ottiene risolvendo le query a partire dal blocco più interno.

### ESEMPIO

Volendo conoscere il nome dei prodotti che usano 3 unità del componente C005, si utilizza la seguente query:

```
SELECT nome_prodotto
FROM Prodotti
WHERE id_prod IN
      (SELECT id_prod
       FROM Composizione
       WHERE unita_comp = 3 AND id_comp = 'C005') AS T;
```

L'operatore **IN** può operare non solo su singoli campi, ma anche su liste di essi. Nella query che segue si è utilizzata tale possibilità per selezionare i dati degli impiegati che hanno lo stesso codice dipartimento e la stessa qualifica di tutti coloro che hanno uno stipendio maggiore di 3000.00.

```
SELECT *
FROM Personale
WHERE (id_dip,qualifica) IN
      (SELECT id_dip,qualifica
       FROM Personale
       WHERE stipendio>3000.00) AS T;
```

**OSSERVAZIONE** La prima query dell'esempio avrebbe potuto essere formulata utilizzando un'operazione di *join*:

```
SELECT nome_prodotto
FROM Prodotti,Composizione
WHERE unita_comp = 3 AND id_comp = 'C005'
      AND Prodotti.id_prod=Composizione.id_prod;
```

dove l'ultima parte della condizione esprime il criterio di *join* applicato.

**OSSERVAZIONE** Nella formulazione originale, la subquery eseguita per prima fornisce l'insieme dei codici dei prodotti che utilizzano 3 unità del componente C005. La query esterna realizza una restrizione sulla tabella prodotto per trovare il nome di quei prodotti il cui codice è tra quelli forniti dalla query interna.

Nella seconda formulazione si è invece realizzata un'operazione di *join* tra la tabella *Prodotti* e la tabella *Composizione* in funzione dell'associazione della chiave primaria/esterna `id_prod`. Sulle righe fornite dall'operazione di *join* è stato applicato il criterio di selezione `unita_comp = 3 AND id_comp = 'C005'`.

Non è mai immediato dire se sia preferibile usare una query nidificata, o un'operazione di *join* equivalente. In effetti molto dipende da come opera il motore del DBMS utilizzato per ottimizzare la strategia di esecuzione della query. In generale è ragionevole pensare di utilizzare la soluzione più semplice a scriversi.

**OSSERVAZIONE** Nelle query la cui la clausola **FROM** coinvolge più tabelle è necessario, nel caso in cui più campi di diverse tabelle abbiano lo stesso nome, fare riferimento a ognuno di essi con la notazione:

`<nome_tabella>.<nome_campo>`

in caso contrario viene rilevata un'ambiguità e la query non viene eseguita.

La notazione in oggetto può comunque essere utilizzata indipendentemente dalla situazione di ambiguità.

L'operazione di *join* realizza il prodotto cartesiano tra le righe di più tabelle. In generale se le tabelle coinvolte nell'operazione di *join* sono  $N$  è necessario specificare  $N - 1$  criteri di *join*.

#### ESEMPIO

Per ottenere il nome dei dipartimenti che utilizzano il componente C003 possiamo formulare la seguente query:

```
SELECT DISTINCT NomeDipartimento
FROM Dipartimenti,Prodotti,Composizione
WHERE id_comp = 'C003'
      AND Dipartimenti.id_dip=Prodotti.id_dip
      AND Prodotti.id_prod=Composizione.id_prod;
```

**OSSERVAZIONE** In generale l'operazione di *join* tra due tabelle prevede la specifica di un criterio che si basa sull'uguaglianza tra la chiave primaria della tabella dominio dell'associazione e la chiave esterna corrispondente della tabella codominio (*equijoin*).

Come caso limite è possibile non specificare alcun criterio di *join*: in questo caso si otterrà come risultato il prodotto cartesiano completo tra le due tabelle. Se le due tabelle hanno rispettivamente  $N$  ed  $M$  righe si ottiene una tabella di  $N \times M$  righe, ovvero una tabella in cui ogni riga della prima tabella viene combinata con tutte le  $M$  righe della seconda.

## OSSERVAZIONE

In un database ben progettato e normalizzato le operazioni di *join* tra tabelle sono effettuate dove esiste una relazione di integrità referenziale tra queste. Si noti però che è possibile eseguire operazioni di *join* tra coppie di tabelle che non hanno questa relazione. L'unico vincolo richiesto è quello per cui i campi su cui viene effettuata l'operazione di *join* siano compatibili nelle due tabelle.

Per esprimere condizioni tra righe diverse della stessa tabella («*self-join*») si possono usare *alias* per rappresentare viste multiple della tabella stessa: gli *alias* sono specificati tramite la parola chiave **AS**.

## ESEMPIO

Per selezionare i codici di tutti i prodotti che impiegano i componenti C003 e C005 è possibile operare con una *self-join* come segue:

```
SELECT X.id_prod
FROM Composizione AS X, Composizione AS Y
WHERE Y.id_comp = 'C003'
AND X.id_comp = 'C005'
AND X.id_prod = Y.id_prod;
```

La stessa query avrebbe potuto essere eseguita anche tramite una subquery nidificata:

```
SELECT id_prod
FROM Composizione
WHERE id_comp = 'C003'
AND id_prod IN (
    SELECT id_prod
    FROM Composizione
    WHERE id_comp = 'C005'
) AS T;
```

oppure ancora con due subquery e una *join*:

```
SELECT id_prod
FROM (SELECT * FROM Composizione WHERE id_comp = 'C003') AS T1,
     (SELECT * FROM Composizione WHERE id_comp = 'C005') AS T2
WHERE T1.id_comp = T2.id_comp;
```

**OSSERVAZIONE** In teoria non esiste un limite alla nidificazione delle query, ma le subquery possono essere espresse solo nell'ambito della clausola **FROM**, oppure nell'ambito della clausola **WHERE**. In effetti è possibile specificare una subquery anche nell'elenco dei campi che segue **SELECT** a patto che tale query restituisca un singolo valore scalare, cioè una tabella di una sola riga e una sola colonna.

L'operazione di *join* può in genere essere espletata con una sintassi diversa da quella appena vista utilizzando l'operatore **INNER JOIN** nella clausola

**FROM:** in questo modo si ricorre alla clausola **WHERE** solo per la specifica della condizione di selezione.

**ESEMPIO**

Riformuliamo due query viste in precedenza utilizzando l'operatore **INNER JOIN**.

- nome dei prodotti che usano 3 unità del componente C005:

```
SELECT nome_prodotto
FROM Prodotti INNER JOIN
      Composizione ON Prodotti.id_prod=Composizione.id_prod
WHERE unita_comp = 3 AND id_comp = 'C005';
```

- nome dei dipartimenti che utilizzano il componente C003:

```
SELECT DISTINCT nome_dipartimento
FROM Dipartimenti INNER JOIN (
      Prodotti INNER JOIN
      Composizione ON Prodotti.id_prod=Composizione.id_prod)
ON Dipartimenti.id_dip=Prodotti.id_dip
WHERE id_comp = 'C003';
```

Una query che effettua un'operazione di *join* tra due tabelle A e B applica il seguente criterio: effettua la *join* tra le righe della tabella A e quelle della tabella B scartando le righe della tabella A che non hanno almeno una corrispondenza con quelle della tabella B in funzione del criterio di *join* specificato.

Questo criterio può nascondere all'utente informazioni che in alcuni casi è necessario conoscere.

Per esempio, supponendo di avere le due seguenti tabelle rispettivamente per le entità *Dipartimenti* e *Personale* (i campi sono riportati in modo parziale):

id_dip	nome_dipartimento	...
D1	Primo dipartimento	
D2	Secondo dipartimento	
D3	Terzo dipartimento	
D4	Quarto dipartimento	
D5	Quinto dipartimento	

matricola	id_dip	nominativo	...
00013	D1	ROSSI PIERO	
00034	D2	NERI GIOVANNI	
00075	D4	ARNETTI ENNIO	
04346	D3	BELLI DANIELA	
04434	D2	VERDI MARCO	
04450	D3	SANDRI DONATA	
04532	D3	GIANNINI PIETRO	
04541	D3	TESINI MARIO	
04551	D1	BIANCHI MAURO	
04717	D2	PIERINI MARIO	
04794	D2	CARLETTI PAOLO	
05019	D4	SOLDANI GIULIO	
05462	D3	LAPINI PAOLO	
05477	D4	BRESCHI CARLA	

eseguendo la query

```
SELECT *  
FROM Dipartimenti INNER JOIN Personale  
ON Dipartimenti.id_dip = Personale.id_dip
```

si ottiene il seguente risultato:

Dipartimento. id_dip	nome_dipartimento	matricola	Personale. id_dip	nominativo
D1	Primo dipartimento	00013	D1	ROSSI PIERO
D1	Primo dipartimento	04551	D1	BIANCHI MAURO
D2	Secondo dipartimento	00034	D2	NERI GIOVANNI
D2	Secondo dipartimento	04434	D2	VERDI MARCO
D2	Secondo dipartimento	04717	D2	PIERINI MARIO
D2	Secondo dipartimento	04794	D2	CARLETTI PAOLO
D3	Terzo dipartimento	04346	D3	BELLI DANIELA
D3	Terzo dipartimento	04450	D3	SANDRI DONATA
D3	Terzo dipartimento	04532	D3	GIANNINI PIETRO
D3	Terzo dipartimento	04541	D3	TESINI MARIO
D3	Terzo dipartimento	05462	D3	LAPINI PAOLO
D4	Quarto dipartimento	00075	D4	ARNETTI ENNIO
D4	Quarto dipartimento	05019	D4	SOLDANI GIULIO
D4	Quarto dipartimento	05477	D4	BRESCHI CARLA

Non avendo il dipartimento D5 alcun dipendente esso non viene riportato nella query.

**OSSERVAZIONE** La query precedente avrebbe potuto essere formulata in modo analogo come segue:

```
SELECT *  
FROM Dipartimenti, Personale  
WHERE Dipartimenti.id_dip = Personale.id_dip;
```

Dal momento che i due campi su cui viene formulato il criterio di *join* hanno lo stesso nome, il linguaggio SQL permette di scrivere in maniera più compatta l'operatore **INNER JOIN**:

```
SELECT *  
FROM Dipartimenti INNER JOIN Personale  
USING (id_dip)  
WHERE Dipartimenti.id_dip = Personale.id_dip;
```

Per visualizzare i dati di due tabelle anche nel caso di mancate corrispondenze tra le loro righe, è necessario ricorrere all'operatore *outer join* nella forma **LEFT OUTER JOIN** (o **LEFT JOIN**) o **RIGHT OUTER JOIN** (o **RIGHT JOIN**).

Per visualizzare i dati dei dipartimenti indipendentemente dal fatto che abbiano o meno impiegati registrati è possibile usare la query:

```
SELECT *
FROM Dipartimenti LEFT JOIN Personale
ON Dipartimenti.id_dip = Personale.id_dip
```

che produce il seguente risultato:

Dipartimento.id_dip	nome_dipartimento	matricola	Personale.id_dip	nominativo
D1	Primo dipartimento	00013	D1	ROSSI PIERO
D1	Primo dipartimento	04551	D1	BIANCHI MAURO
D2	Secondo dipartimento	00034	D2	NERI GIOVANNI
D2	Secondo dipartimento	04434	D2	VERDI MARCO
D2	Secondo dipartimento	04717	D2	PIERINI MARIO
D2	Secondo dipartimento	04794	D2	CARLETTI PAOLO
D3	Terzo dipartimento	04346	D3	BELLI DANIELA
D3	Terzo dipartimento	04450	D3	SANDRI DONATA
D3	Terzo dipartimento	04532	D3	GIANNINI PIETRO
D3	Terzo dipartimento	04541	D3	TESINI MARIO
D3	Terzo dipartimento	05462	D3	LAPINI PAOLO
D4	Quarto dipartimento	00075	D4	ARNETTI ENNIO
D4	Quarto dipartimento	05019	D4	SOLDANI GIULIO
D4	Quarto dipartimento	05477	D4	BRESCHI CARLA
D5	Quinto dipartimento			

in cui sono presenti, nell'ultima riga, i dati del dipartimento D5 senza alcun dipendente associato.

In modo simile, supponendo che tra le due tabelle non sia stato definito alcun vincolo di integrità referenziale (altrimenti quanto andiamo a ipotizzare non sarebbe possibile perché impedito dal DBMS), inserendo nella tabella del personale un impiegato con codice di dipartimento non avvalorato (**NULL**), la query

```
SELECT *
FROM Dipartimenti RIGHT JOIN
Personale ON Dipartimenti.id_dip = Personale.id_dip;
```

fornisce il seguente risultato:

Dipartimento.id_dip	nome_dipartimento	matricola	Personale.id_dip	nominativo
D1	Primo dipartimento	00013	D1	ROSSI PIERO
D1	Primo dipartimento	04551	D1	BIANCHI MAURO
D2	Secondo dipartimento	00034	D2	NERI GIOVANNI
D2	Secondo dipartimento	04434	D2	VERDI MARCO
D2	Secondo dipartimento	04717	D2	PIERINI MARIO
D2	Secondo dipartimento	04794	D2	CARLETTI PAOLO
D3	Terzo dipartimento	04346	D3	BELLI DANIELA
D3	Terzo dipartimento	04450	D3	SANDRI DONATA
D3	Terzo dipartimento	04532	D3	GIANNINI PIETRO
D3	Terzo dipartimento	04541	D3	TESINI MARIO
D3	Terzo dipartimento	05462	D3	LAPINI PAOLO
D4	Quarto dipartimento	00075	D4	ARNETTI ENNIO
D4	Quarto dipartimento	05019	D4	SOLDANI GIULIO
D4	Quarto dipartimento	05477	D4	BRESCHI CARLA
		05486		MARINI MARIA

in cui è possibile verificare che l'impiegato con matricola 05486 non è legato ad alcun dipartimento.



Il linguaggio SQL prevede, nella formulazione delle subquery che possono restituire più valori, gli operatori **ALL** e **ANY** che permettono di verificare rispettivamente se una certa relazione vale per tutti i valori, o per almeno un valore.

#### ESEMPIO

Per conoscere matricola e nominativo dei dipendenti con lo stipendio più basso, si può formulare la seguente query:

```
SELECT matricola, id_dip, nominativo
FROM Personale
WHERE Stipendio <= ALL (SELECT Stipendio
                        FROM Personale) AS T;
```

Per conoscere codice e nome di tutti i dipartimenti in cui esiste almeno un dipendente con stipendio maggiore di 1500.00, si può formulare la seguente query:

```
SELECT id_dip, nome_dipartimento
FROM Dipartimenti
WHERE id_dip = ANY (SELECT id_dip
                   FROM Personale
                   WHERE Stipendio > 1500.00) AS T;
```

Si noti che la forma «= **ANY**» equivale all'operatore **IN**.

Inoltre il linguaggio SQL prevede nella formulazione delle subquery l'operatore **EXIST** che permette di verificare se il risultato di una subquery restituisce almeno una riga.

#### ESEMPIO

Volendo conoscere codice e nome di tutti i dipartimenti in cui esiste almeno un dipendente con qualifica Q5, si può formulare la seguente query:

```
SELECT id_dip, nome_dipartimento
FROM Dipartimenti AS T
WHERE EXISTS (SELECT * FROM Personale
             WHERE Qualifica='Q5' AND id_dip = T.id_dip) AS T1;
```

In questo caso la subquery dipende da un dipartimento specifico e il significato diviene: «per ogni riga del blocco esterno, considera il valore di T.id\_dip e risolvi la subquery».

Quando una subquery fa riferimento ad *alias* definiti in un blocco esterno, si definisce **correlata**.

Utilizzando query correlate è anche possibile operare la cosiddetta «divisione relazionale».

L'affermazione «dipartimenti in cui sono presenti tutte le qualifiche» equivale a «dipartimenti in cui non esiste una qualifica non presente»; la seguente query individua tali dipartimenti:

```
SELECT nome_dipartimento
FROM Dipartimenti AST
WHERE NOT EXISTS
  (SELECT *
   FROM Personale AST1
   WHERE NOT EXISTS
     (SELECT *
      FROM Personale AST2
      WHERE T.id_dip = T2.id_dip
      AND T1.qualifica = T2.qualifica) AS T3
   ) AS T4;
```

In questo caso il blocco più interno viene valutato per ogni combinazione di T e T1 e il blocco centrale è il «divisore» (T1.qualifica). Dato un dipartimento T, se in esso manca una qualifica, la subquery più interna non restituisce **NULL**, quindi la subquery intermedia restituisce almeno una riga e la condizione espressa nella clausola **WHERE** di T non è soddisfatta.

### 3 Le funzioni di aggregazione e la clausola di raggruppamento

Il linguaggio SQL fornisce alcune funzioni che possono essere usate nel comando **SELECT** per determinare la somma e la media aritmetica di un insieme di valori (**SUM** e **AVG**), la cardinalità di un insieme di righe (**COUNT**), il massimo e il minimo di un insieme di valori (**MAX** e **MIN**).

Per conoscere il prezzo minimo, massimo e medio di vendita dei prodotti del dipartimento D2, è possibile utilizzare la seguente query:

```
SELECT MIN(prezzo) AS prezzo_min, MAX(prezzo) AS prezzo_max,
       AVG(prezzo) AS prezzo_medio
FROM Prodotti
WHERE id_dip = 'D2';
```

Per calcolare l'importo totale degli stipendi dei dipendenti con qualifica Q7 è possibile usare la seguente query:

```
SELECT SUM(stipendio) AS totale_stipendi_dirigenti
FROM Personale
WHERE qualifica = 'Q7';
```

Per determinare il numero dei dipendenti del dipartimento D3 è possibile usare la query:

```
SELECT COUNT(*) AS n_dipendenti_D3
FROM Personale
WHERE id_dip = 'D3';
```

**OSSERVAZIONE** Nell'esempio precedente è stata usata la clausola `COUNT (*)` per contare tutte le righe della tabella che soddisfano alla condizione espressa. Invece del carattere *jolly* «\*» è possibile utilizzare il nome di un campo: in questo caso il significato è quello di contare tutte le righe che soddisfano la condizione formulata e in cui il campo specifico abbia un valore diverso da **NULL**.

Nel caso in cui un campo sia specificato come argomento di una funzione `COUNT`, è anche possibile premettere a esso la parola chiave **DISTINCT** per evitare che uno stesso valore venga contato più volte.

Il linguaggio SQL permette di partizionare in «gruppi» le righe di una tabella utilizzando la clausola **GROUP BY**. A tali gruppi può essere applicata una qualsiasi delle funzioni di aggregazione viste in precedenza.

**ESEMPIO**

La seguente query calcola lo stipendio medio per ogni dipartimento:

```
SELECT id_dip, AVG(stipendio) AS stipendio_medio
FROM Personale
GROUP BY id_dip;
```

Nell'eseguire questa query viene partizionata la tabella *Personale* nei seguenti quattro gruppi:

matricola	id_dip	nominativo	data_nascita	...	stipendio
00013	D1	ROSSI PIERO	1965-01-15		1640.00
04551	D1	BIANCHI MAURO	1968-07-09		1180.00
00034	D2	NERI GIOVANNI	1974-08-05		2930.00
04434	D2	VERDI MARCO	1977-05-09		2280.00
04717	D2	PIERINI MARIO	1969-06-20		2520.00
04794	D2	CARLETTI PAOLO	1981-07-02		2270.00
04346	D3	BELLI DANIELA	1977-02-25		3740.00
04450	D3	SANDRI DONATA	1979-05-01		3070.00
04532	D3	GIANNINI PIETRO	1978-06-04		3200.00
04541	D3	TESINI MARIO	1978-12-28		3740.00
05462	D3	LAPINI PAOLO	1977-01-11		3960.00
00075	D4	ARNETTI ENNIO	1967-01-15		4750.00
05019	D4	SOLDANI GIULIO	1983-03-27		1180.00
05477	D4	BRESCHI CARLA	1976-05-02		4200.00

quindi applicato il calcolo della media ad ogni singolo gruppo fornendo il risultato che segue:

id_dip	stipendio_medio
D1	1410.00
D2	2500.00
D3	3542.00
D4	3376.66

Per ottenere il codice e il nome di ogni dipartimento e il numero dei componenti utilizzati (ogni componente deve essere contato una sola volta) possiamo usare la seguente query:

```
SELECT Dipartimenti.id_dip,nome_dipartimento,
       COUNT(DISTINCT Composizione.id_comp) AS N_Componenti
FROM Dipartimenti,Prodotti,Composizione
WHERE Prodotti.id_prod = Composizione.id_prod
      AND Dipartimenti.id_dip = Prodotti.id_dip
GROUP BY Dipartimenti.id_dip,nome_dipartimento;
```

**OSSERVAZIONE** Quando la lista dei campi della clausola **SELECT** comprende sia campi semplici che funzioni di aggregazione, è obbligatorio specificare una clausola **GROUP BY** seguita dalla lista di tutti i campi semplici.

Se invece la lista dei campi comprende solo campi semplici, l'utilizzo della clausola **GROUP BY** è consentita a patto che questa preveda la specifica della lista completa di tali campi e in tal caso la clausola **GROUP BY** produce lo stesso risultato della parola chiave **DISTINCT**.

Volendo calcolare il costo unitario di produzione dei vari prodotti come somma del costo unitario di ogni componente moltiplicato la quantità utilizzata nel singolo prodotto, è necessario ricorrere alla seguente query:

```
SELECT Prodotti.id_prod,id_dip,nome_prodotto,
       SUM(costo_unitario*unita_comp) AS costo_produzione
FROM Prodotti, Componenti, Composizione
WHERE Componenti.id_comp=Composizione.id_comp
      AND Prodotti.id_prod=Composizione.id_prod
GROUP BY Prodotti.id_prod, id_dip, nome_prodotto;
```

Una tabella può essere partizionata in gruppi a cui applicare una condizione per selezionarne solo alcuni. Questa selezione viene realizzata impiegando la clausola **HAVING**.

La clausola **HAVING** può essere utilizzata insieme a una qualsiasi funzione di aggregazione, ma sempre associata a una clausola **GROUP BY**.

Per elencare i codici dei prodotti che impiegano almeno cinque componenti distinti, si ricorre alla seguente query:

```
SELECT id_prod
FROM Composizione
GROUP BY id_prod
HAVING COUNT(*) >= 5;
```

**OSSERVAZIONE** Si deve fare attenzione alla differenza tra il modo di operare delle due clausole **WHERE** e **HAVING**.

La prima permette la formulazione di condizioni sulla base dei valori dei singoli campi delle diverse righe di una tabella, mentre le condizioni formulabili con la seconda riguardano interi gruppi di righe formati dalla clausola **GROUP BY**.

Per esempio la condizione **HAVING COUNT(\*)** esprime una condizione sulla cardinalità dei gruppi (numero di righe) costruiti dalla clausola **GROUP BY**. Ovviamente i diversi tipi di condizioni possono agire in sinergia all'interno di una singola query: la clausola **WHERE** seleziona un insieme di righe, la clausola **GROUP BY** ne provvede al partizionamento in gruppi dei quali solo quelli che soddisfano il criterio specificato dalla condizione **HAVING** saranno quelli selezionati.

Vediamo ora alcuni esempi più articolati.

## ESEMPIO

Per determinare il codice del/dei prodotto/i che utilizzano la massima quantità del componente C005, la metodologia da utilizzare è la seguente:

prima di tutto si deve determinare la quantità massima di componente C005 utilizzata dai singoli prodotti:

```
(SELECT MAX(unita_comp)
FROM Composizione
WHERE id_comp='C005') AS T;
```

quindi è necessario trovare il codice di prodotto che utilizza il componente C005 in quantità uguale a quella calcolata in *T*

```
SELECT id_prod
FROM Composizione
WHERE id_comp='C005' AND unita_comp = T
```

sostituendo a *T* la subquery relativa, si ottiene:

```
SELECT id_prod
FROM Composizione
WHERE id_comp='C005'
      AND unita_comp = (SELECT MAX(unita_comp)
                        FROM Composizione
                        WHERE id_comp='C005') AS T;
```

Si noti come la condizione «*id\_comp='C005'*» sia necessaria sia nella query interna che in quella esterna: infatti se non fosse ripetuta anche in quella esterna sarebbero selezionati anche tutti quei prodotti che utilizzano un qualsiasi componente in quantità uguale a quella calcolata dalla query interna indipendentemente dal fatto che tale quantità si riferisca o meno al componente C005.

## OSSERVAZIONE

L'uso dell'operatore di uguaglianza nella formulazione della condizione «*unita\_comp = (SELECT MAX(unita\_comp) ...)*» è reso possibile dal fatto che la query che seleziona la massima quantità di componente è una **query scalare**, ovvero restituisce un solo valore (altrimenti sarebbe stato necessario utilizzare al posto di «*=*» l'operatore **IN**).

Per determinare il dipartimento con lo stipendio medio maggiore e il relativo importo medio, la metodologia da utilizzare è la seguente:

la prima cosa da fare è determinare la media degli stipendi per ogni dipartimento:

```
(SELECT id_dip, AVG(Stipendio) AS stipendio_medio
FROM Personale
GROUP BY id_dip) AS T;
```

quindi è necessario trovare il massimo degli stipendi medi:

```
SELECT MAX(stipendio_medio)
FROM T;
```

Ora è possibile comporre la query finale:

```
SELECT id_dip, stipendio_medio
FROM T
WHERE stipendio_medio = (SELECT MAX(stipendio_medio)
                        FROM T);
```

sostituendo a *T* la subquery relativa, si ottiene:

```
SELECT id_dip, stipendio_medio
FROM (SELECT id_dip, AVG(Stipendio) AS stipendio_medio
      FROM Personale
      GROUP BY id_dip) AS T1
WHERE stipendio_medio =
      (SELECT MAX(stipendio_medio)
       FROM (SELECT id_dip, AVG(Stipendio) AS stipendio_medio
             FROM Personale
             GROUP BY id_dip)
          AS T2
      );
```

Per determinare il/i dipartimento/i col massimo numero di dipendenti la metodologia da utilizzare è la seguente:

la prima cosa da fare è determinare il numero dei dipendenti per ogni dipartimento:

```
(SELECT id_dip, COUNT(*) AS N_Dipendenti
FROM Personale
GROUP BY id_dip) AS T;
```

quindi è necessario trovare il massimo tra i dipendenti calcolati in *T*

```
SELECT MAX(N_Dipendenti)
FROM T;
```

la query definitiva è:

```
SELECT id_dip, N_Dipendenti
FROM T
WHERE N_Dipendenti = (SELECT MAX(N_Dipendenti)
                     FROM T
                     );
```

sostituendo a *T* la subquery relativa, si ottiene:



```

SELECT id_dip, N_Dipendenti
FROM (SELECT id_dip, COUNT(*) AS N_Dipendenti
      FROM Personale
      GROUP BY id_dip
    ) AS T1
WHERE N_Dipendenti = (SELECT MAX(N_Dipendenti)
                      FROM (SELECT id_dip, COUNT(*) AS N_Dipendenti
                            FROM Personale
                            GROUP BY id_dip
                          ) AS T2
                      );

```

## 4 Operatori di unione, intersezione e differenza

L'operatore **UNION** effettua l'unione di due tabelle generate da comandi **SELECT**, secondo la seguente sintassi:

```

SELECT <lista_campi1>
FROM <lista_tabelle1>
[WHERE <condizione1>]

```

**UNION**

```

SELECT <lista_campi2>
FROM <lista_tabelle2>
[WHERE <condizione2>]

```

1. Questa caratteristica può essere ottenuta implementando un *trigger*.

dove <lista\_campi1> e <lista\_campi2> devono essere le stesse.

### ESEMPIO

Si dispone di una tabella *Cronologia\_Costi\_Componenti* in cui, ogni volta che si modifica il costo unitario di un componente viene memorizzato il valore precedente e la data dell'avvenuta modifica<sup>1</sup>.

Se le tabelle *Componenti* e *Cronologia\_Costi\_Componenti* sono le seguenti:

id_comp	nome_componente	costo_unitario
C001	Componente01	0.30
C002	Componente02	0.50
C003	Componente03	0.30
C004	Componente04	0.25
C005	Componente05	0.10
C006	Componente06	0.05
C007	Componente07	0.05
C008	Componente08	0.10
C009	Componente09	0.50
C010	Componente10	0.40
C011	Componente11	0.20
C012	Componente12	0.05
C013	Componente13	0.05
C014	Componente15	0.20
C015	Componente15	0.15
C020	Componente20	0.15

id_comp	nome_componente	costo_unitario	data_validita
C001	Componente01	0.10	2008-03-01
C001	Componente01	0.15	2010-05-15
C001	Componente01	0.25	2011-01-05
C004	Componente04	0.05	2010-06-20
C004	Componente05	0.10	2011-01-05
C009	Componente09	0.35	2011-01-20
C010	Componente10	0.25	2011-01-20
C011	Componente11	0.15	2010-11-05
C012	Componente12	0.02	2011-09-20
C020	Componente20	0.10	2010-02-10

la seguente query ricostruisce l'evoluzione del costo di alcuni componenti nel tempo

```
SELECT id_comp,nome_componente,costo_unitario,
       CURRENT_DATE() AS data_validita
FROM Componenti
WHERE nome_componente LIKE 'Componente0%'

UNION

SELECT id_comp,nome_componente,costo_unitario,data_validita
FROM Cronologia_Costi_Componenti
WHERE nome_componente LIKE 'Componente0%'

ORDER BY id_comp, data_validita
```

e produce il seguente risultato:

id_comp	nome_componente	costo_unitario	data_validita
C001	Componente01	0.10	2008-03-01
C001	Componente01	0.15	2010-05-15
C001	Componente01	0.25	2011-03-01
C001	Componente01	0.30	2012-01-05
C002	Componente02	0.50	2012-01-05
C003	Componente03	0.30	2012-01-05
C004	Componente04	0.05	2010-06-20
C004	Componente04	0.10	2011-01-05
C004	Componente04	0.25	2012-01-05
C005	Componente05	0.10	2012-01-05
C006	Componente06	0.05	2012-01-05
C007	Componente07	0.05	2012-01-05
C009	Componente09	0.35	2011-01-20
C009	Componente09	0.50	2012-01-05

**OSSERVAZIONE** Nella prima subquery è stata inserita l'espressione `CURRENT_DATE() AS data_validita` che crea un campo calcolato per uniformare i campi di *Componenti* e *Cronologia\_Costi\_Componenti*.

Gli operatori **INTERSECT** ed **EXCEPT** del linguaggio SQL eseguono rispettivamente l'intersezione e la differenza in termini insiemistici di due tabelle generate da comandi **SELECT**, secondo la seguente sintassi:

```
SELECT <lista_campi1>
FROM <lista_tabelle1>
[WHERE <condizione1>]

{ INTERSECT | EXCEPT }

SELECT <lista_campi2>
FROM <lista_tabelle2>
[WHERE <condizione2>]
```

dove <lista\_campi1> e <lista\_campi2> devono essere le stesse.



Il DBMS MySQL non supporta questi operatori per cui una query del tipo

```
SELECT id_prod
FROM Composizione
WHERE id_comp='C03'
INTERSECT
SELECT id_prod
FROM Composizione
WHERE id_comp='C05'
```

deve essere riscritta in una forma come la seguente che utilizza una *self-join*:

```
SELECT X.id_prod
FROM Composizione AS X INNER JOIN
      Composizione AS Y ON X.id_prod=Y.id_prod
WHERE X.id_comp='C03' AND Y.id_comp='C05'
```

**OSSERVAZIONE** L'uso dell'operatore **EXCEPT** può essere banalmente evitato:

```
SELECT id_prod
FROM Composizione
WHERE id_comp='C03'
      AND id_prod NOT IN (SELECT id_prod
                          FROM Composizione
                          WHERE id_comp='C05')
```

## 5 I comandi DDL del linguaggio SQL: CREATE, ALTER e DROP

Il linguaggio DDL di SQL prevede comandi da usare per creare e modificare la struttura delle tabelle e dagli altri oggetti di un database. L'effetto dell'esecuzione di un comando di tipo DDL è immediato.

Il comando **CREATE** permette di creare database, tabelle, indici, vincoli di integrità su tabelle e tra tabelle, ecc.

Per la creazione di una tabella questo comando nella sua forma base ha la seguente sintassi:

```
CREATE TABLE <nome_tabella> (
  <nome_campo_1> <tipo_1>[<vincolo_1>],
  ...
  <nome_campo_n> <tipo_n> [<vincolo_n>]);
```

Il seguente comando permette la creazione della tabella *Personale*:

```
CREATE TABLE Personale (
    matricola VARCHAR(5) NOT NULL DEFAULT '',
    id_dip VARCHAR(3) DEFAULT NULL,
    nominativo VARCHAR(50) DEFAULT NULL,
    data_nascita DATE NOT NULL,
    qualifica VARCHAR(2) DEFAULT NULL,
    stipendio DOUBLE DEFAULT 0
);
```

**OSSERVAZIONE** Il vincolo **NOT NULL** indica che il campo dovrà essere sempre avvalorato, mentre la parola chiave **DEFAULT** indica quale valore dovrà inserire il DBMS se tale campo all'atto dell'inserimento di una nuova riga non dovesse essere avvalorato.

Il comando **CREATE TABLE** permette di definire contestualmente anche i vincoli di integrità referenziale relativi alla chiave primaria e alle chiavi esterne.

Il seguente comando permette la creazione della tabella *Composizione*:

```
CREATE TABLE Composizione(
    id_prod VARCHAR(4),
    id_comp VARCHAR(4),
    unita_comp REAL,
    CONSTRAINT ChiavePrimaria PRIMARY KEY(id_prod,id_comp),
    CONSTRAINT ComponentiComposizione FOREIGN KEY(id_comp)
        REFERENCES Componente(id_comp),
    CONSTRAINT ProdottiComposizione FOREIGN KEY(id_prod)
        REFERENCES Prodotto(id_prod)
);
```

Sono di facile individuazione nell'esempio precedente i punti in cui vengono definiti i vincoli di integrità referenziale. In corrispondenza alla parola chiave **CONSTRAINT** la clausola **PRIMARY KEY** permette di definire i campi della chiave primaria, la clausola **FOREIGN KEY** definisce i campi di una chiave esterna, mentre **REFERENCES** individua la chiave primaria della tabella a cui una chiave esterna si riferisce.

Il nome che segue la parola chiave **CONSTRAINT** è il nome con cui il vincolo specificato viene memorizzato nel database e può essere arbitrario.

## 5.1 Tipi di dato

Oltre ai tipi di dato definiti dallo standard SQL, ogni DBMS ne definisce di propri. Il DBMS MySQL usa tre categorie fondamentali di tipi di dato:

- numerici;
- date e orari;
- stringhe.

## Dati numerici

Nella **TABELLA 1** sono riassunti i tipi numerici previsti dal DBMS MySQL. Le indicazioni comprese fra parentesi quadre sono opzionali (M indica il numero totale di cifre rappresentate e D il numero di cifre decimali). Tutti i tipi numerici escluso **BIT** possono avere le opzioni **UNSIGNED** e **ZEROFILL**. Con la prima si specifica che il numero è senza segno, per cui non sono consentiti valori negativi; con la seconda si indica al DBMS di memorizzare i numeri con zeri davanti nel caso in cui la lunghezza sia inferiore a quella massima prevista (in questo caso MySQL aggiunge automaticamente **UNSIGNED**).

**TABELLA 1**

Tipo di dato	Descrizione
<b>BIT</b> [ (M) ]	È rappresentato con il numero di bit specificato da M (da 1 a 64; default 1)
<b>TINYINT</b> [ (M) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato su un byte e può contenere 256 valori, da -128 a +127, oppure da 0 a 255 se <b>UNSIGNED</b>
<b>SMALLINT</b> [ (M) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato su due byte e può contenere 65536 valori da -32768 a 32767, oppure da 0 a 65535 se <b>UNSIGNED</b>
<b>MEDIUMINT</b> [ (M) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato su tre byte e può contenere 16.777.216 valori da -8388608 a 8388607, oppure da 0 a 16777215 se <b>UNSIGNED</b>
<b>INT</b> [ (M) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato su quattro byte e può contenere oltre 4 miliardi di valori da -2147483648 a 2147483647, oppure da 0 a 4294967295 se <b>UNSIGNED</b>
<b>BIGINT</b> [ (M) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato su otto byte e può contenere circa 18 miliardi di miliardi di valori da -9223372036854775808 a 9223372036854775807 oppure da 0 a 18446744073709551615 se <b>UNSIGNED</b>
<b>FLOAT</b> [ (M,D) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato in virgola mobile a «precisione singola»: i suoi limiti teorici vanno da -3.402823466E+38 a -1.175494351E-38 e da 1.175494351E-38 a 3.402823466E+38, oltre allo zero
<b>DOUBLE</b> [ (M,D) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	È rappresentato in virgola mobile a «precisione doppia»: i limiti teorici vanno da -1.7976931348623157E+308 a -2.2250738585072014E-308 e da 2.2250738585072014E-308 a 1.7976931348623157E+308, oltre allo zero
<b>DECIMAL</b> [ (M[,D]) ] [ <b>UNSIGNED</b> ] [ <b>ZEROFILL</b> ]	Rappresentano numeri «esatti», con M cifre totali di cui D decimali. I valori di default sono 10 per M e 0 per D. I valori limite per questi dati sono gli stessi di <b>DOUBLE</b> . Il massimo di cifre consentite è 65 per M e 30 per D

**OSSERVAZIONE** Per i tipi **DOUBLE** e **FLOAT** i limiti reali dipendono dall'hardware e dal sistema operativo. Se M e D non sono indicati i valori possono essere memorizzati fino ai limiti effettivi. Per questi dati l'uso di **UNSIGNED** disabilita i valori negativi, ma non ha effetto sui valori massimi positivi memorizzabili. La precisione dei numeri in virgola mobile è affidabile fino alla settima cifra decimale (circa) per **FLOAT** e alla quindicesima (circa) per **DOUBLE**.

Esistono numerosi sinonimi per i dati numerici riportati nella **TABELLA 2**.

TABELLA 2

Tipo di dato	Sinonimo
BOOL BOOLEAN	TINYINT (1)
INTEGER	INT
DOUBLE PRECISION REAL	DOUBLE
DEC NUMERIC FIXED	DECIMAL

## Date e orari

Nella TABELLA 3 sono riassunti i tipi data e ora previsti dal DBMS MySQL.

TABELLA 3

Tipo di dato	Descrizione
DATE	Rappresenta date comprese tra '1000-01-01' (1 gennaio 1000) e '9999-12-31' (31 dicembre 9999). MySQL visualizza le date nel formato mostrato, ma consente di inserirle sotto forma di stringhe o numeri
DATETIME	Rappresenta una data e un'ora, con lo stesso <i>range</i> visto per DATE. La visualizzazione è nel formato 'AAAA-MM-GG HH:MM:SS', ma anche in questo caso possono essere usati formati diversi per l'inserimento
TIMESTAMP [ (M) ]	Può essere usato per memorizzare i valori corrispondenti al <i>timestamp</i> Unix, corrispondente al numero di secondi trascorsi dalla mezzanotte del 1 gennaio 1970
TIME	Rappresenta un valore di orario (ore, minuti e secondi) che va da '-838:59:59' a '838:59:59'. Anche qui la visualizzazione avviene nel formato indicato, ma è possibile usare formati diversi per l'inserimento
YEAR [ (2   4) ]	Rappresenta, su quattro cifre, un anno compreso fra 1901 e 2155, oppure 0000. Su due cifre invece i valori vanno da 70 (1970) a 69 (2069)

**OSSERVAZIONE** I valori relativi al tempo possono essere inseriti sia come stringhe che come sequenze di numeri. MySQL consente di utilizzare, nel caso delle stringhe, diversi tipi di caratteri come separatori. L'importante però è che l'ordine dei valori sia sempre anno-mese-giorno-ore-minuti-secondi. Quando si usano i separatori nelle stringhe è anche possibile omettere gli zeri non significativi (ad esempio è consentito '2005-9-21', altrimenti è necessario usare '20050921').

L'operazione più comune lavorando con date e orari è quella di estrarne i valori dal database e presentarli in un formato facilmente interpretabile dall'utente. Come abbiamo detto la rappresentazione che MySQL adotta per valori **DATE** (AAAA-MM-GG), **DATETIME** (AAAA-MM-GG HH:MM:SS) non risulta in generale di immediata lettura. Si ha quindi la necessità di elaborare il dato in modo da modificarne la visualizzazione. La funzione `DATE_FORMAT()` può risolvere il problema in maniera semplice ed elegante. Essa riceve due argomenti: il primo è la data da formattare, il secondo una

stringa composta da alcuni caratteri speciali, preceduti dal simbolo «%», che indicano come tale data debba essere formattata. La TABELLA 4 riporta i principali caratteri di formato.

TABELLA 4

Carattere	Descrizione
%d	giorno del mese numerico 01...31
%M	nome del mese January...December
%m	mese numerico 01...12
%H	ora 00...23
%i	minuti 00...59
%s	secondi 00...59
%Y	anno di quattro cifre
%y	anno di due cifre

ESEMPIO

Le seguenti query:

```
SELECT DATE_FORMAT('2005-04-12 15:23:04','%d-%m-%Y %H:%i:%s')
AS data_ formattata;
```

```
SELECT DATE_FORMAT('2005-04-12 15:23:04','%d %M %y, ore: %H')
AS data_ formattata;
```

```
SELECT DATE_FORMAT('20050412152304','giorno:%d,mese:%m,anno:%Y,
ore:%H e %i')
AS data_ formattata;
```

producono rispettivamente i seguenti risultati:

```
12-04-2005 15:23:04
12 April 05, ore: 15
giorno:12,mese:04,anno:2005,ore:15 e 23
```

Negli esempi visti è stata passata direttamente alla funzione la data da formattare, ma nulla cambia se si tratta di un valore estratto da una tabella; una possibile query applicata al campo `data_nascita` della tabella *Personale*, potrebbe essere così strutturata:

```
SELECT DATE_FORMAT(data_nascita,'%d-%m-%Y') AS Data_Nascita
FROM Personale;
```

**OSSERVAZIONE** Per formattare informazioni relative al tempo si può ricorrere alla funzione `TIME_FORMAT()`, del tutto analoga a `DATE_FORMAT`, ma che utilizza solo gli specificatori di formato per gestire ore, minuti e secondi.

La funzione `STR_TO_DATE()` del DBMS MySQL è l'inversa di `DATE_FORMAT`, essa riceve come argomenti la stringa contenente la data e la corrispondente stringa di formattazione e restituisce un valore **DATETIME**, **DATE** o **TIME** a seconda del contenuto.

#### ESEMPIO

Le seguenti query:

```
SELECT STR_TO_DATE('03/10/2009', '%d/%m/%Y');
SELECT STR_TO_DATE('03.10.2009 12.33', '%d.%m.%Y %H.%i');
```

producono rispettivamente i seguenti risultati:

```
2009-10-03
2009-10-03 12:33:00
```

### Geo-database per sistemi Gis

Oltre ai tipi di dato classici (numeri, stringhe, date, orari), molti DBMS prevedono la gestione di dati geometrici (punti, linee, poligoni, ecc.) cioè tipi di dati che si basano sulle specifiche dell'Open GIS Consortium e che permettono l'esecuzione di query di tipo spaziale per estrarre informazione da dati geografici.

Opportune estensioni del linguaggio SQL supportano la gestione di questi tipi di dati e la loro analisi geometrica.

MySQL fornisce alcune funzioni di uso molto comune che ritornano valori di tipo data o orario (TABELLA 5).

TABELLA 5

Funzione	Descrizione
<code>NOW()</code> <code>CURRENT_TIMESTAMP()</code> <code>SYSDATE()</code>	Forniscono la data e l'ora corrente nel formato 'AAAA-MM-GG HH:MM:SS' o AAAAMMGGHHMMSS a seconda che si tratti di un contesto stringa o numerico
<code>CURRDATE()</code> <code>CURRENT_DATE()</code>	Forniscono la data corrente nel formato 'AAAA-MM-GG' o AAAAMMGG a seconda che si tratti di un contesto stringa o numerico
<code>CURRTIME()</code> <code>CURRENT_TIME()</code>	Forniscono l'ora corrente nel formato 'HH:MM:SS' o HHMMSS a seconda che si tratti di un contesto stringa o numerico
<code>UNIX_TIMESTAMP()</code>	Se non viene specificato alcun argomento <code>UNIX_TIMESTAMP()</code> fornisce lo Unix <i>timestamp</i> corrente, se invece si specifica un valore di tipo <b>DATE</b> , <b>DATETIME</b> , <b>TIMESTAMP</b> fornisce lo Unix <i>timestamp</i> corrispondente alla particolare data

Per la gestione di date e orari il DBMS MySQL offre varie funzioni di utilità di cui le principali sono riepilogate nella TABELLA 6.

In relazione alle operazioni con date e orari valgono le due regole seguenti:

- la somma di un intero  $n$  a una data produce come risultato la data che si ottiene da quella di partenza aggiungendo  $n$  giorni;
- la differenza tra due date produce un intero che rappresenta il numero di giorni che intercorrono tra le due date.

TABELLA 6

Funzione	Descrizione
YEAR() MONTH() DAY() WEEK()	Estraggono da un valore o campo data rispettivamente l'anno, il mese, il giorno, la settimana in formato numerico
HOURL() MINUTE() SECOND()	Estraggono da un valore orario rispettivamente l'ora, i minuti, i secondi in formato numerico
ADDDATE()	Aggiunge un intervallo temporale espresso in giorni a una data
SUBDATE()	Sottrae un intervallo temporale espresso in giorni a una data
DATEDIFF()	Effettua la differenza tra due date espressa in giorni

**ESEMPIO**

Le seguenti query

```
SELECT ADDDATE('2010-07-19',5);
SELECT SUBDATE('2010-07-19',5);
SELECT ADDDATE('2011-07-19',-5);
```

producono rispettivamente i seguenti risultati:

```
2010-07-24
2011-07-14
2011-07-14
```

mentre la query

```
SELECT DATEDIFF(CURRDATE(),data_nascita)
FROM Personale
WHERE matricola='00034';
```

fornisce come risultato il numero di giorni che intercorrono tra la data odierna e la data di nascita del dipendente con matricola 00034.

## Stringhe di caratteri

La TABELLA 7 riepiloga i tipi per le stringhe del DBMS MySQL.

TABELLA 7

CHAR (M)	È una stringa di lunghezza fissa M completata con spazi a destra al momento della memorizzazione, che sono eliminati in fase di lettura; la lunghezza prevista va da 0 a 255 caratteri
VARCHAR (M)	È una stringa a lunghezza variabile con lunghezza massima dichiarabile di 65535 caratteri
BINARY (M)	Corrispondono a CHAR e VARCHAR, ma memorizzano stringhe di byte invece che di caratteri; i valori BINARY sono completati a destra con valori 0
VARBINARY (M)	
TINYTEXT TEXT [ (M) ] MEDIUMTEXT LONGTEXT	Memorizzano campi di testo: la lunghezza massima è 255 caratteri per TINYTEXT, 65535 per TEXT, 16777215 per MEDIUMTEXT, 4294967296 per LONGTEXT
ENUM ('valore1', 'valore2',...)	Può contenere uno dei valori elencati nella definizione, oppure NULL o una stringa vuota, che viene assegnata quando si cerca di inserire un valore non valido; i valori possibili possono essere fino a 65535

## 5.2 Altri usi del comando CREATE; DB-schema e integrità referenziale

Il comando **CREATE**, oltre alla creazione delle tabelle, può servire anche a creare oggetti diversi, tra i più comuni:

- database:

```
CREATE DATABASE <nome_database>;
```

- una tabella a partire dal risultato di una query:

```
CREATE TABLE <nome_tabella>  
  SELECT <lista_campi>  
  FROM <lista_tabelle>  
  [WHERE <condizione>;]
```

- una nuova vista a partire dal risultato di una query:

```
CREATE VIEW <nome_vista> AS  
  (SELECT <lista_campi>  
   FROM <lista_tabelle>  
   [WHERE <condizione>]);
```

In SQL una **vista** è una tabella virtuale utilizzabile a tutti gli effetti come se fosse una tabella reale ma che non fa parte della base di dati se non come query memorizzata. Tali viste vengono calcolate solo quando richieste. È possibile definire il nome della vista e dei suoi campi.

### ESEMPIO

Il seguente comando SQL crea una vista che, per ciascun dipartimento, fornisce il totale degli stipendi mensili:

```
CREATE VIEW Totale_Stipendi(id_dip,importo_totale_stipendi) AS  
  (SELECT id_dip, SUM(stipendio)  
   FROM Personale  
   GROUP BY id_dip);
```

- un indice per velocizzare le ricerche su uno specifico campo di una tabella:

```
CREATE INDEX <nome_indice>  
  [UNIQUE] ON <nome_tabella>(<nome_campo>;
```

### ES.

```
CREATE INDEX indice_nominativo  
  ON Personale(nominativo);
```

**OSSERVAZIONE** La clausola **UNIQUE** utilizzabile in un comando di creazione di un indice fa sì che non siano accettati valori duplicati nel campo su cui esso viene definito.



Un DB-*schema* è l'insieme delle istruzioni DDL che permettono la creazione della struttura (aspetto intensionale) della base di dati. Esso viene spesso usato anche per effettuare un *porting* del database da un DBMS a un altro, anche di tipo diverso.



## ESEMPIO

Una DB-*schema* relativo alla base di dati che abbiamo usato per gli esempi del presente capitolo potrebbe essere il seguente:

```
CREATE TABLE Componenti(
    id_comp VARCHAR(4) NOT NULL,
    nome_componente VARCHAR(25) NOT NULL,
    fornitore VARCHAR(25) NOT NULL,
    costo_unitario DOUBLE NOT NULL,
    CONSTRAINT ChiavePrimaria PRIMARY KEY(id_comp)
);

CREATE TABLE Dipartimenti(
    id_dip VARCHAR(3) NOT NULL,
    nome_dipartimento VARCHAR(30) NOT NULL,
    localita VARCHAR(20) NOT NULL,
    provincia VARCHAR(2) NOT NULL,
    CONSTRAINT ChiavePrimaria PRIMARY KEY(id_dip)
);

CREATE TABLE Personale(
    matricola VARCHAR(5) NOT NULL,
    id_dip VARCHAR(3) NOT NULL,
    nominativo VARCHAR(50) NOT NULL,
    data_nascita DATE NOT NULL,
    qualifica VARCHAR(2) NOT NULL,
    stipendio DOUBLE NOT NULL,
    CONSTRAINT ChiavePrimaria PRIMARY KEY(matricola),
    CONSTRAINT DipartimentiPersonale FOREIGN KEY(id_dip)
        REFERENCES Dipartimenti(id_dip)
);

CREATE TABLE Prodotti(
    id_prod VARCHAR(4) NOT NULL,
    id_dip VARCHAR(3) NOT NULL,
    nome_prodotto VARCHAR(50) NOT NULL,
    prezzo DOUBLE NOT NULL,
    CONSTRAINT ChiavePrimaria PRIMARY KEY(id_prod),
    CONSTRAINT DipartimentiProdotti FOREIGN KEY(id_dip)
        REFERENCES Dipartimenti(id_dip)
);

CREATE TABLE Composizione(
    id_prod VARCHAR(4) NOT NULL,
    id_comp VARCHAR(4) NOT NULL,
    unita_comp REAL NOT NULL,
    CONSTRAINT ChiavePrimaria PRIMARY KEY(id_prod, id_comp),
    CONSTRAINT ComponentiComposizione FOREIGN KEY(id_comp)
        REFERENCES Componenti(id_comp),
    CONSTRAINT ProdottiComposizione FOREIGN KEY(id_prod)
        REFERENCES Prodotti(id_prod)
);
```

Si noti che l'ordine in cui sono state specificate le tabelle è importante. Se, ad esempio, fosse stata definita *Composizione* prima di *Prodotti* o di *Componenti*, il DBMS non sarebbe stato in grado di creare i vincoli relativi alle chiavi esterne che riferiscono le chiavi primarie di tali tabelle. In alternativa è possibile definire solo la struttura delle tabelle in un ordine qualsiasi e solo dopo aggiungere tali vincoli mediante istruzioni **ALTER TABLE**.

**OSSERVAZIONE** Alla definizione di una **FOREIGN KEY** è possibile specificare (congiuntamente o singolarmente) le clausole legate all'integrità referenziale **ON DELETE CASCADE** e **ON UPDATE CASCADE** per indicare rispettivamente che:

- cancellando una riga nella tabella che contiene la chiave primaria corrispondente si richiede la cancellazione automatica di tutte le righe in cui la chiave esterna assume lo stesso valore;
- aggiornando il valore della chiave primaria nella tabella dominio di una relazione si richiede l'aggiornamento automatico di tutte le righe in cui la chiave esterna assume uguale valore.

## Gestione dei privilegi in un DBMS

In generale i DBMS dispongono di una gestione dei permessi che consentono di definire quali tipi di operazioni (esecuzione di query, inserimento, modifica e cancellazione dei dati, creazione di tabelle e indici, ecc.) i vari utenti sono abilitati o meno ad effettuare.

I due comandi base per la gestione dei privilegi sono **GRANT** e **REVOKE**: il primo per concedere privilegi a un utente, il secondo per eliminarli.

Questi comandi sono a disposizione esclusiva del DBA (*Database Administrator*).

## 5.3 I comandi ALTER, RENAME e DROP

Il comando **ALTER** può essere usato per modificare una tabella aggiungendo, togliendo o modificando un campo, una chiave, un indice, ecc.:

```
ALTER TABLE <nome_tabella> <operazione_di_modifica>;
```

### ESEMPI

```
ALTER TABLE Personale ADD INDEX indice_data_nascita(data_nascita);
```

crea un indice sul campo relativo alla data di nascita nella tabella *Personale*;

```
ALTER TABLE CronologiaPersonale  
ADD PRIMARY KEY(matricola,data_operazione);
```

crea la chiave primaria per la tabella *CronologiaPersonale* sulla combinazione dei campi relativi alla matricola e alla data di operazione;

```
ALTER TABLE Personale  
CHANGE matricola numero_matricola(6) NOT NULL;
```

cambia nome e dimensione del campo *matricola* della tabella *Personale*;

```
ALTER TABLE Prodotti MODIFY prezzo FLOAT NOT NULL;
```

modifica il tipo della colonna *prezzo* nella tabella *Prodotti*;

```
ALTER TABLE prodotti ADD costo_produzione DOUBLE NOT NULL DEFAULT 0;
```

aggiunge il campo relativo al costo di produzione nella tabella *Prodotti*;

```
ALTER TABLE prodotti DROP costo_produzione;
```

rimuove il campo relativo al costo di produzione dalla tabella *Prodotti*;

```
ALTER TABLE composizione  
ADD CONSTRAINT ProdottiComposizione FOREIGN KEY(id_prod)  
REFERENCES Prodotto(id_prod);
```

aggiunge la chiave esterna nella tabella *Composizione* relativa all'integrità referenziale con la tabella *Prodotti*.

Il comando **RENAME** permette la ridenominazione di una tabella esistente:

```
RENAME TABLE <vecchio_nome> TO <nuovo_nome>;
```

Il comando **DROP** permette l'eliminazione di un qualsiasi oggetto di database:

```
DROP <nome_oggetto>;
```

**ESEMPIO**

```
DROP DATABASE test;
```

elimina il database «test»;

```
DROP TABLE CronologiaPersonale;
```

elimina la tabella *CronologiaPersonale*;

```
DROP INDEX indice_data_nascita ON Personale
```

elimina l'indice *indice\_data\_nascita* dalla tabella *Personale*.

**6**

## I comandi DML del linguaggio SQL: INSERT, UPDATE, DELETE

I comandi fondamentali del linguaggio SQL per inserire, cancellare e modificare i dati delle tabelle di un database sono rispettivamente **INSERT**, **DELETE** e **UPDATE**.

Le forme generali del comando **INSERT** sono le seguenti:

```
INSERT INTO <tabella> [(<lista_campi>)] VALUES (<lista_valori>);
```

oppure

```
INSERT INTO <tabella> SET <campo 1> = <valore 1>, ...,  
                        <campo n> = <valore n>;
```

**ESEMPIO**

Per inserire i dati relativi al nuovo dipartimento di Firenze, si può usare il seguente comando:

```
INSERT INTO Dipartimenti  
          (id_dip,nome_dipartimento,localita,provincia)  
VALUES ('D7','Dipartimento 7','Firenze','FI');
```

oppure

```
INSERT INTO Dipartimenti  
SET id_dip='D7',nome_dipartimento = 'Dipartimento 7',  
    localita='Firenze', provincia='FI';
```

**OSSERVAZIONE** La prima versione del comando **INSERT** consente di omettere l'elenco dei campi, in questo caso però debbono essere specificati i valori per ogni campo della tabella nell'ordine in cui questi sono stati definiti nella creazione della tabella stessa.

Viceversa la forma in cui viene specificato l'elenco dei campi consente di utilizzare un ordine diverso da quello stabilito al momento della creazione della tabella ed eventualmente di omettere qualche campo che verrà impostato a **NULL**. A questo proposito di quest'ultima eventualità è bene precisare che il comando verrà rifiutato dal DBMS nel caso in cui non siano specificati campi che fanno parte della chiave primaria della tabella o campi per i quali in fase di creazione della medesima sia stato specificato il vincolo **NOT NULL**. Queste ultime osservazioni valgono anche per la seconda forma del comando **INSERT** nel caso in cui non tutti i campi della tabella siano stati specificati nella clausola **SET**.

Il linguaggio SQL permette anche di inserire dati in una tabella estrapolandoli dal risultato di una query.

#### ESEMPIO

La seguente query permette di inserire nella tabella *Componenti* un insieme di nuovi componenti estratti dalla tabella *NuoviComponenti*.

```
INSERT INTO Componenti (id_comp,nome_componente,fornitore)
SELECT id_comp,nome_componente,fornitore
FROM NuoviComponenti;
```

Per cancellare i dati di una tabella SQL usa il comando **DELETE** la cui forma generale è:

```
DELETE FROM <tabella> [WHERE <condizione>]
```

#### ESEMPIO

Per eliminare i dati dei componenti acquistati dal fornitore Rossi si può usare il seguente comando:

```
DELETE
FROM Componenti
WHERE fornitore='Rossi';
```

Se è stata attivata la clausola di cancellazione automatica dei record correlati, questa cancellazione comporta l'eliminazione dalla tabella *Composizione* dei record relativi ai componenti eliminati.

**OSSERVAZIONE** Se nel comando **DELETE** non viene specificata la clausola **WHERE**, l'operazione si applica a tutte le righe della tabella ed eventualmente, se è attiva l'opzione di cancellazione in cascata, a tutti i dati interessati nelle tabelle a essa correlate.

## Stored procedure

Le *Stored procedures* sono programmi scritti generalmente in linguaggio SQL, memorizzati nel database. Essi consentono di creare procedure di gestione dei dati in modo che siano sempre disponibili per il DBMS. Il linguaggio SQL è per sua natura dichiarativo e le *Stored procedure* rappresentano una sua estensione procedurale.

Per la loro creazione alcuni DBMS prevedono dei veri e propri linguaggi di programmazione come PL/SQL per i DBMS Oracle.

Il comando **UPDATE** permette di aggiornare i dati memorizzati in una tabella. La sua forma generale è la seguente:

```
UPDATE <tabella> SET <campo 1> = <valore 1>, ...,
                    <campo n> = <valore n>
[WHERE <condizione>];
```

### ESEMPIO

Il seguente comando aggiorna lo stipendio (+10%) del dipendente con matricola 0078:

```
UPDATE Personale
SET stipendio = stipendio*1.1
WHERE matricola='0078';
```

**OSSERVAZIONE** Se nel comando **UPDATE** non viene specificata la clausola **WHERE**, l'operazione di aggiornamento si applica a tutte le righe della tabella.

Vediamo un esempio di comando **UPDATE** più articolato.

### ESEMPIO

Supponiamo di avere aggiunto alla tabella *Dipartimenti* un nuovo campo *N\_dipendenti* per contenere il numero di dipendenti di ogni singolo dipartimento e di voler aggiornare in modo automatico tale valore.

Il comando che segue risolve il problema mediante una sola **UPDATE** che utilizza una subquery e una *join*:

```
UPDATE Dipartimenti, (SELECT id_dip, COUNT(*) AS N_dipendenti
                     FROM Personale
                     GROUP BY id_dip) AS T
SET Dipartimenti.N_dipendenti = T.N_dipendenti
WHERE Dipartimenti.id_dip = T.id_dip;
```

## 7 I trigger

In SQL un *trigger* è un meccanismo per intercettare gli eventi in cui si verificano specifiche condizioni sul valore dei dati indicando le azioni da intraprendere di conseguenza. Gli eventi per i quali si attiva un trigger corrispondono all'esecuzione di una istruzione **INSERT**, **UPDATE** o **DELETE** su una tabella.

I *trigger* sono utilizzati per diversi scopi nella progettazione e implementazione di un database:

- per mantenere l'integrità referenziale tra le tabelle;
- per mantenere l'integrità dei dati di una singola tabella;
- per verificare il contenuto dei campi di una tabella;
- per creare tabelle di registrazione delle modifiche o delle eliminazioni apportate al database.

La sintassi di un *trigger* è la seguente:

```
CREATE TRIGGER <nome_trigger>
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON <nome_tabella>
FOR EACH ROW
<codice_SQL>
```

#### ESEMPIO

Il seguente *trigger* attiva, quando viene aggiornato il costo di un componente, l'inserimento dei dati di tale componente in una tabella chiamata *CronologiaCostiComponenti* per registrare l'avvenuto aggiornamento:

```
CREATE TRIGGER Aggiorna_Costo_Componente
AFTER UPDATE
ON Componenti
FOR EACH ROW
BEGIN
    IF NEW.costo_unitario <> OLD.costo_unitario
        INSERT INTO CronologiaCostiComponenti
        VALUES (OLD.id_comp, OLD.nome_componente, OLD.costo_unitario,
            CURRENT_DATE());
    END IF;
END;
```

Allo stesso modo, il *trigger* che segue attiva, quando viene cancellata una riga della tabella *Personale*, l'inserimento dei dati dell'impiegato cancellato in una opportuna tabella chiamata *CronologiaPersonale* che prevede la registrazione della matricola, codice di dipartimento, nominativo, tipo di operazione e data dell'operazione:

```
CREATE TRIGGER Elimina_Impiegato
BEFORE DELETE
ON Personale
FOR EACH ROW
INSERT INTO CronologiaPersonale
VALUES (OLD.matricola, OLD.id_dip, OLD.nominativo, 'Eliminato', CURRENT_DATE());
```

Infine il seguente *trigger* attiva, quando viene inserito un nuovo prodotto, il controllo sul prezzo di vendita ponendo il prezzo uguale a 0 se è stato inserito un valore negativo oppure ponendo lo stesso uguale a 250 se è stato inserito un valore maggiore:

```
CREATE TRIGGER Nuovo_Prodotto
BEFORE INSERT ON Prodotti
FOR EACH ROW
BEGIN
    IF NEW.prezzo < 0 THEN
        SET NEW.prezzo = 0;
    END IF;
    IF NEW.prezzo > 250.00 THEN
        SET NEW.prezzo = 250.00;
    END IF;
END;
```

**OSSERVAZIONE** Nella sintassi dei *trigger* sono utilizzate parole chiave che permettono di fare riferimento rispettivamente alle righe della tabella interessate all'evento (**FOR EACH ROW**), al nuovo valore dei campi di tali righe (**NEW**) e al vecchio valore dei campi di tali righe (**OLD**).

## 8 Accesso concorrente ai dati

Una delle problematiche classiche che un DBMS deve gestire è quella relativa all'accesso concorrente ai dati, sia in lettura che scrittura, da parte di più utenti, e in generale da parte di più applicazioni software eseguite contemporaneamente.

Una situazione tipica di questo scenario è il caso in cui due utenti accedono contemporaneamente allo stesso dato con l'intenzione di aggiornarlo: uno dei due lo farà per primo e quando il secondo effettuerà a sua volta l'aggiornamento troverà una situazione variata rispetto al momento in cui aveva letto i dati e rischierà di salvare dati incoerenti.

Un'altra situazione ricorrente è quella in cui un'applicazione debba effettuare più aggiornamenti logicamente correlati fra loro, tanto da richiedere che tutti gli aggiornamenti siano annullati se anche uno solo di essi dovesse non avere esito positivo.

Le soluzioni a questi problemi sono rappresentate da due tecniche standard:

- il *locking* sulle tabelle nei casi più semplici;
- le transazioni nei casi più complessi.

**OSSERVAZIONE** È preferibile utilizzare le transazioni ogni volta che è possibile.

### Locking

I *lock* sono vincoli di «uso esclusivo» che un utente può ottenere su specifiche tabelle per il tempo necessario a svolgere le operazioni necessarie.

Un *lock* può essere richiesto in lettura o in scrittura: nel primo caso l'utente ha la sicurezza che nessuno effettuerà aggiornamenti sulla tabella bloccata fino a quando non sarà rilasciato il *lock*, ma gli altri utenti hanno la possibilità di leggere dalla stessa tabella (anche l'utente che ha ottenuto il *lock* non può fare aggiornamenti).

Il *lock* in scrittura invece impedisce agli altri utenti qualsiasi tipo di accesso alla tabella e consente all'utente che l'ha ottenuto di effettuare sia operazioni di lettura che di scrittura.

In MySQL la sintassi delle operazioni di *lock* è la seguente:

#### LOCK TABLES

```
<tabella> [AS <alias>] {READ [LOCAL] | [LOW_PRIORITY] WRITE}  
[, <tabella> [AS <alias>] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
```

**OSSERVAZIONE** È possibile effettuare il *lock* su più tabelle con un'unica istruzione **LOCK TABLES**. Dal momento che ogni istruzione **LOCK TABLES** causa il rilascio dei *lock* ottenuti in precedenza se è necessario effettuare un *lock* su più tabelle, questo deve essere effettuato con un'unica istruzione.

**OSSERVAZIONE** A ogni tabella su cui si opera un *lock* è possibile associare un *alias*, esattamente come nelle query. Dopo avere effettuato un *lock*, le query possono utilizzare solo le tabelle bloccate: non è possibile quindi, in presenza di *lock* attivi, accedere ad altre tabelle; inoltre, alle tabelle si deve accedere utilizzando gli *alias* definiti nel *lock*. Se una tabella è presente più volte in una query, sono necessari più *alias*: di conseguenza si deve ottenere un *lock* per ogni *alias*, anche se la tabella è la stessa.

La parola chiave **LOCAL** associata a un **READ lock** consente ad altri utenti di effettuare inserimenti che non vadano in conflitto con le operazioni di lettura. La parola chiave **LOW\_PRIORITY** associata a un **WRITE lock** lascia la precedenza alle richieste di *lock* in lettura (altrimenti un *lock* in scrittura ha priorità più alta).

■ I lock ottenuti vengono rilasciati con l'istruzione **UNLOCK TABLES**.

**OSSERVAZIONE** Anche una nuova richiesta di *lock* causa il rilascio dei precedenti; inoltre i *lock* sono rilasciati automaticamente alla disconnessione dal DBMS.

## 8.1 Le transazioni

L'uso delle transazioni permette di rendere effettive le modifiche apportate alla base dati in un momento determinato: dall'istante in cui ha inizio una transazione, gli aggiornamenti rimangono sospesi (e invisibili ad altri utenti) fino a quando non sono confermati (*commit*) o, in alternativa, annullati (*rollback*).

Per iniziare una transazione deve essere usato il comando **START TRANSACTION**; da istante tutti gli aggiornamenti restano sospesi. La transazione può essere chiusa con il comando **COMMIT**, che consolida le modifiche, oppure con **ROLLBACK**, che annulla tutti gli aggiornamenti effettuati e sospesi nel corso della transazione.

All'interno di una transazione è possibile stabilire dei *savepoint*, cioè degli stati intermedi ai quali è possibile ritornare con una **ROLLBACK**, in alternativa ad annullare interamente la transazione.

**OSSERVAZIONE** MySQL, come altri DBMS, opera normalmente in *autocommit mode*: questo significa che tutti gli aggiornamenti sono automaticamente consolidati nel momento stesso in cui sono eseguiti. È possibile disattivare l'*autocommit mode* con il comando **SET AUTOCOMMIT=0**; in questo caso non è necessario avviare le transazioni con **START TRANSACTION** perché tutti gli aggiornamenti restano sospesi fino all'uso di **COMMIT** o **ROLLBACK**.



```
START TRANSACTION;
<istruzioni di aggiornamento 1>
SAVEPOINT s1;
<istruzioni di aggiornamento 2>
ROLLBACK TO SAVEPOINT s1;
<istruzioni di aggiornamento 3>
COMMIT;
```

In questo caso, dopo avere avviato la transazione è stato eseguito un primo blocco di aggiornamenti, seguito dalla creazione del *savepoint* *s1*; in seguito è stato eseguito un secondo blocco di aggiornamenti; il comando **ROLLBACK TO SAVEPOINT** *s1* ripristina la situazione esistente al momento della creazione del *savepoint*: in pratica solo il secondo blocco di aggiornamenti viene annullato e la transazione rimane aperta; un comando **ROLLBACK** invece avrebbe annullato tutto e chiuso la transazione.

Il comando **COMMIT** effettuato dopo il terzo blocco di aggiornamenti consolida gli aggiornamenti effettuati nel primo e nel terzo blocco e chiude la transazione.

**OSSERVAZIONE** Alcuni tipi di operazioni non sono annullabili: in generale tutte quelle che alterano la struttura di database e tabelle. È bene quindi evitare di includere in una transazione tali operazioni che nella maggior parte dei casi, causano una **COMMIT** implicita.

In alcune situazioni è utile specificare un *lock* quando si esegue un comando **SELECT** impiegando la seguente sintassi:

```
SELECT ... FOR UPDATE;
SELECT ... LOCK IN SHARE MODE;
```

La parola chiave **FOR UPDATE** stabilisce un *lock* su tutte le righe lette che impedisce ad altri utenti di leggere le righe coinvolte fino al termine della transazione: si utilizza per leggere un dato con l'intenzione di aggiornarlo. La parola chiave **LOCK IN SHARE MODE** stabilisce un *lock* che impedisce gli aggiornamenti, garantendo che il contenuto della riga resta invariato per la durata della transazione.

## Livello di isolamento

Un aspetto importante legato alle transazioni è il livello di isolamento (isolation level) nel quale esse vengono effettuate. I livelli possibili sono quattro, elencati in ordine gerarchico nella **TABELLA 8**.

Il livello di isolamento si imposta con il seguente comando SQL:

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE };
```

Omettendo le parole chiave **GLOBAL** e **SESSION** la modifica è valida solo per la transazione successiva; con **SESSION** si imposta il valore per l'intera sessione (fino alla disconnessione dal DBMS), mentre con **GLOBAL** si modifica il valore per il DBMS: tale valore sarà quindi adottato per tutte le connessioni aperte successivamente (ma non su quelle già aperte).

TABELLA 8

Livello di isolamento	Descrizione
<b>READ UNCOMMITTED</b>	A questo livello sono visibili gli aggiornamenti effettuati da altri utenti anche se non ancora consolidati. Questo livello non è transazionale in senso stretto: esso può causare problemi di consistenza dei dati e va utilizzato solo quando non vi sono problematiche di accesso concorrente
<b>READ COMMITTED</b>	A questo livello gli aggiornamenti sono visibili solo dopo il consolidamento
<b>REPEATABLE READ</b>	È la modalità predefinita. Perché un aggiornamento sia visibile deve essere consolidato e la transazione che la legge terminata. La stessa lettura ripetuta all'interno di una transazione darà quindi sempre lo stesso risultato
<b>SERIALIZABLE</b>	Opera come nel caso precedente con l'aggiunta che la semplice lettura di un dato provoca il blocco degli aggiornamenti fino al termine della transazione; in sostanza è come se ogni comando <b>SELECT</b> venisse effettuato ricorrendo alla parola chiave <b>LOCK IN SHARE MODE</b>

## Sintesi

■ **SQL**. *Structured Query Language*, linguaggio per operare su basi di dati.

■ **SELECT**. Comando base di SQL per la formulazione di query; la sua forma generale è:

```
SELECT <lista_campi> FROM <lista_tabelle>
[WHERE <condizione>];
```

Restituisce come risultato una tabella virtuale.

■ **DISTINCT**. Clausola da specificare dopo la parola chiave **SELECT** per eliminare eventuali righe duplicate restituite dalla query.

■ **IN**. Operatore SQL che permette di formulare condizioni per verificare se il valore di un campo appartiene ai valori di un insieme; in pratica costituisce un OR multiplo. Se negato (**NOT IN**) permette di verificare se il valore appartiene al complementare di un insieme.

■ **Like**. Operatore SQL che permette di formulare condizioni per verificare il *pattern-matching* tra il valore di un campo di tipo stringa e un elemento di confronto. Viene utilizzato con il carattere *jolly* «%» per definire corrispondenze parziali.

■ **BETWEEN**. Operatore SQL che permette di definire condizioni per verificare se il valore di un campo appartiene a un intervallo di valori di cui vengono indicati gli estremi.

■ **AS**. Operatore SQL che permette di definire *alias* per assegnare nomi temporanei a tabelle o campi.

■ **ORDER BY**. Parola chiave SQL che permette di definire un criterio di ordinamento che deve

essere applicato alle righe del risultato di una query; il criterio può prevedere più di un campo per ognuno dei quali può essere specificato un ordinamento ascendente (**ASC**) o discendente (**DESC**).

■ **Subquery**. Grazie alla proprietà di chiusura del linguaggio SQL è possibile formulare query all'interno di altre query con una tecnica di nidificazione. Le query vengono risolte a partire da quelle più interne: i risultati di queste forniscono i dati su cui operano le query di livello superiore.

■ **ALL**, **ANY** e **EXIST**. Operatori SQL utilizzati con le query nidificate per verificare rispettivamente se una certa relazione è valida per tutti i valori, per almeno un valore tra quelli forniti dalla subquery, se il risultato di una subquery restituisce almeno una riga.

■ **Subquery correlata**. È una subquery che fa riferimento ad *alias* definiti nelle query esterne.

■ **Join**. Operazione di concatenazione temporanea tra le righe di tabelle distinte in funzione di un criterio specifico (formulato nella clausola **WHERE** della query) che generalmente corrisponde all'uguaglianza tra il valore della chiave primaria di una tabella e il valore della chiave esterna di un'altra tabella che si trova in relazione di integrità referenziale con la prima. In ogni caso è possibile eseguire operazioni di *join* tra coppie di tabelle che non necessariamente si trovino in tale relazione. L'unico vincolo richiesto è quello per cui i campi su cui viene effettuata la *join* siano compatibili nelle due tabelle. Se il criterio di *join* non viene espresso

l'operazione restituisce come risultato il prodotto cartesiano completo tra le due tabelle.

■ **Inner join.** Operatore SQL che implementa l'operatore di *join* all'interno della clausola **FROM** della query.

■ **Self-join.** Operazione di *join* di una tabella con se stessa.

■ **Outer join.** Operatore di *join* che permette di selezionare i dati di due tabelle anche nel caso di mancate corrispondenze tra le righe delle medesime. Sono previsti i due operatori **LEFT OUTER JOIN** (o **LEFT JOIN**) e **RIGHT OUTER JOIN** (o **RIGHT JOIN**) che rispettivamente permettono di selezionare i dati dalla tabella al primo termine della *join* anche se non vi sono corrispondenze nella tabella al secondo termine, e viceversa.

■ **Funzioni di aggregazione.** SQL fornisce alcune funzioni che possono essere usate nel comando di **SELECT** per determinare la somma e la media aritmetica di un insieme (**SUM** e **AVG**), la cardinalità di un insieme di righe (**COUNT**), il massimo e il minimo di un insieme di valori (**MAX** e **MIN**).

■ **Query scalare.** È una query che restituisce un unico valore (ovvero una tabella 1 colonna e 1 riga).

■ **GROUP BY.** Questa clausola SQL, seguita da una lista di campi, permette di partizionare i dati selezionati da una query in gruppi in funzione del valore di tali campi.

■ **HAVING.** Le righe selezionate da una query possono essere partizionate in gruppi a cui applicare una condizione per sceglierne solo alcuni mediante la clausola **HAVING** che può essere utilizzata insieme a una qualsiasi funzione di aggregazione, ma sempre e comunque associata a una clausola **GROUP BY**.

■ **Unione, intersezione e differenza.** SQL prevede gli operatori **UNION**, **INTERSECT** e **EXCEPT** per implementare le classiche operazioni dell'algebra booleana.

■ **INSERT, UPDATE e DELETE.** Sono i comandi previsti da SQL per inserire, modificare e cancellare e i dati nelle tabelle di un database.

■ **CREATE, ALTER e DROP.** Sono i comandi base del DDL di SQL utilizzati rispettivamente per

la creazione, la modifica e l'eliminazione di oggetti del DBMS (database, tabelle, campi, vincoli di integrità, indici, ecc.).

■ **Vista logica.** In SQL una vista è una tabella virtuale utilizzabile a tutti gli effetti come se fosse una tabella reale ma che non fa parte della base di dati se non come query memorizzata ed eseguita quando richiesto.

■ **DB-schema.** È l'insieme delle istruzioni DDL che permettono la creazione della struttura (aspetto intensionale) di una base di dati. Esso viene spesso usato anche per effettuare un *porting* della base di dati tra DBMS distinti.

■ **Trigger.** Un *trigger* intercetta il verificarsi di specifiche condizioni sul valore dei dati indicando al tempo stesso le azioni da intraprendere. Gli eventi per i quali si attiva un *trigger* sono l'esecuzione di una istruzione **INSERT** / **UPDATE** / **DELETE** su una tabella. I *trigger* sono utilizzati per diversi scopi nella progettazione di un database, principalmente per mantenere l'integrità referenziale tra le tabelle, per mantenere l'integrità dei dati di una singola tabella, per monitorare il valore dei campi di una tabella, per creare tabelle di registrazione delle righe che vengono modificate o eliminate.

■ **Accesso concorrente ai dati.** Una problematica classica che un DBMS deve gestire è quella relativa all'accesso concorrente ai dati sia in lettura che scrittura da parte di più utenti. Una situazione tipica di questo genere è il caso in cui due utenti accedono contemporaneamente allo stesso dato con l'intenzione di aggiornarlo: uno dei due lo farà per primo e quando il secondo effettuerà l'aggiornamento troverà una situazione variata rispetto al momento della lettura dei dati, ed eventualmente produrrà una situazione finale incongruente. Altra situazione comune è quella in cui un'applicazione debba effettuare più aggiornamenti logicamente correlati fra loro, tanto da richiedere che tutti gli aggiornamenti siano annullati se anche uno solo di essi dovesse non avere esito positivo.

■ **Locking.** I *lock* sono vincoli di «uso esclusivo» che un utente può ottenere su specifiche tabelle per il tempo necessario a svolgere le operazioni necessarie. L'uso dei *lock* è consigliato solo in contesti in cui non sono supportate le transazioni.

■ **Transazioni.** L'uso delle transazioni permette di rendere effettive le modifiche alla base dati solo al termine di una sequenza di istruzioni: dall'istante di avvio di una transazione gli aggior-

namenti rimangono sospesi (e invisibili ad altri utenti) fino a quando non vengono confermati (operazione di **COMMIT**) o annullati (operazione di **ROLLBACK**).

## QUESITI

**1** La clausola **FROM** di un comando **SELECT** serve a specificare ...

- A ... una lista di campi da restituire.
- B ... una lista di tabelle su cui operare.
- C ... una condizione di selezione delle righe appartenenti alle tabelle su cui operare.
- D Nessuna delle risposte precedenti.

**2** La clausola **WHERE** di un comando **SELECT** serve a specificare ...

- A ... una lista di campi da restituire.
- B ... una lista di tabelle su cui operare.
- C ... una condizione di selezione delle righe appartenenti alle tabelle su cui operare.
- D Nessuna delle risposte precedenti.

**3** La parola chiave **DISTINCT** di un comando **SELECT** serve a specificare ...

- A ... che si desidera eliminare gli eventuali duplicati tra le righe restituite.
- B ... che nella clausola **FROM** non può essere specificata più volte la stessa tabella.
- C ... che i campi specificati debbono essere tutti distinti tra loro.
- D Nessuna delle risposte precedenti.

**4** È possibile ottenere lo stesso risultato di **DISTINCT** usando **GROUP BY** ...

- A ... sì, sempre.
- B ... no, mai.
- C ... sì ma solo se nella clausola **GROUP BY** viene specificato l'intero insieme di campi indicato nel comando **SELECT**.
- D Nessuna delle risposte precedenti.

**5** È possibile specificare più di una volta lo stesso campo in una clausola **SELECT** ...

- A ... sì, sempre.
- B ... no, mai.
- C ... sì, a patto di ridenominarlo con un *alias* usando **AS**.
- D Nessuna delle risposte precedenti.

**6** I campi reperiti da un comando **SELECT** debbono appartenere tutti alle tabelle specificate nella clausola **FROM** ...

- A ... sì, sempre.
- B ... sì, a patto che questi non siano utilizzati anche nella clausola **WHERE**.
- C ... no, se sono campi calcolati.
- D Nessuna delle risposte precedenti.

**7** Il valore **NULL** viene assunto da un campo se ...

- A ... viene cancellato.
- B ... non viene attribuito alcun valore.
- C ... è di tipo stringa e contiene uno o più spazi bianchi.
- D Nessuna delle risposte precedenti.

**8** L'operatore **LIKE** serve a ...

- A ... specificare una condizione relativa a un *pattern-matching* tra valori di tipo stringa.
- B ... ridenominare con un *alias* un campo che segue la parola chiave **SELECT**.
- C ... ridenominare con un *alias* una tabella specificata nella clausola **FROM**.
- D Nessuna delle risposte precedenti.

**9** L'operatore **BETWEEN** serve a ...

- A ... specificare l'inizio e la fine di un intervallo di nomi campi di una tabella senza doverli elencare singolarmente.

- B ... specificare l'inizio e la fine di un intervallo di nomi di tabella di un database senza doverli elencare singolarmente.
- C ... formulare una condizione di ricerca indicando gli estremi di un intervallo di valori in cui deve essere compreso il valore di un campo.
- D Nessuna delle risposte precedenti.

**10** La parola chiave **AS** può essere utilizzata in un comando **SELECT** per ...

- A ... assegnare un *alias* a un campo.
- B ... assegnare un *alias* ad una condizione espressa nella clausola **WHERE**.
- C ... assegnare un *alias* a una tabella.
- D Nessuna delle risposte precedenti.

**11** È possibile definire un ordinamento nelle righe restituite dal comando **SELECT**?

- A Sì, utilizzando la clausola **ORDER BY**.
- B No, non è mai possibile.
- C Solo utilizzando una subquery.
- D Nessuna delle risposte precedenti.

**12** Il linguaggio SQL offre la possibilità di formulare subquery ...

- A ... perché è una caratteristica di tutti i linguaggi di programmazione.
- B ... perché ha sia comandi DDL che DML.
- C ... perché è un linguaggio completo.
- D Nessuna delle risposte precedenti.

**13** Una subquery SQL ...

- A ... è una query dipendente da una query principale dalla quale è separata dal simbolo «;».
- B ... è una query all'interno di un'altra query per cui la query interna fornisce i risultati su cui opera la query esterna.
- C ... è una query la cui esecuzione è condizionata dal verificarsi o meno della condizione espressa nella clausola **WHERE**.
- D Nessuna delle risposte precedenti.

**14** L'operatore **IN** serve a specificare ...

- A ... una condizione di **OR** multiplo.
- B ... una condizione di **AND** multiplo.

- C ... una condizione mista di **AND** e **OR**.
- D Nessuna delle risposte precedenti.

**15** Volendo formulare una condizione per verificare se una certa relazione è valida per almeno un valore tra quelli forniti da una subquery è necessario usare la parola chiave ...

- A ... **HAVING**.
- B ... **EXIST**.
- C ... **ALL**.
- D Nessuna delle risposte precedenti.

**16** Una subquery viene detta «scalare» se ...

- A ... fornisce le righe della tabella risultato in ordine crescente.
- B ... fornisce le righe della tabella risultato in ordine decrescente.
- C ... la tabella risultato ha solo una riga e una colonna.
- D Nessuna delle risposte precedenti.

**17** Una subquery viene detta «correlata» se ...

- A ... viene utilizzata in una operazione di *join* con un'altra tabella.
- B ... fornisce risultati simili a quelli forniti da un'altra query.
- C ... fa riferimento ad *alias* definiti nelle query esterne.
- D Nessuna delle risposte precedenti.

**18** Dire quali delle seguenti affermazioni sono vere circa un'operazione di *join*.

- A Deve coinvolgere almeno tre tabelle.
- B Può coinvolgere due volte una stessa tabella a patto che a questa siano stati assegnati due *alias* diversi.
- C Non può coinvolgere tabelle che non si trovino in relazione di integrità referenziale.
- D I campi su cui viene formulato il criterio di *join* devono essere dello stesso tipo o almeno compatibili (numeri con numeri, stringhe con stringhe, ecc.).

## 19 L'operatore **OUTER JOIN** ...

- A ... permette di selezionare comunque i dati di due tabelle anche nel caso di mancate corrispondenze tra le righe delle medesime.
- B ... non permette di selezionare i dati di due tabelle nel caso di mancate corrispondenze tra le righe delle medesime.
- C ... permette di selezionare i dati di due tabelle anche nel caso di mancate corrispondenze tra le righe delle medesime a patto che le due tabelle coinvolte si trovino in relazione di integrità referenziale tra di loro.
- D Nessuna delle risposte precedenti.

## 20 Dire quali delle seguenti affermazioni sono vere circa la clausola **GROUP BY**.

- A Deve essere necessariamente usata con le funzioni di aggregazione.
- B Deve essere necessariamente usata con le funzioni di aggregazione se nel comando **SELECT** sono elencati anche dei campi semplici.
- C Deve essere necessariamente usata con la clausola **HAVING**.
- D Permette di partizionare i dati selezionati da una query in gruppi in funzione del valore di una lista di campi.

## 21 La condizione **HAVING COUNT(\*)** di un comando **SELECT** esprime una condizione ...

- A ... sulla cardinalità totale delle righe selezionate.
- B ... sulla cardinalità di singoli gruppi in cui sono partizionate le righe selezionate.
- C ... sulla cardinalità dell'insieme delle righe che soddisfano la condizione specificata nella clausola **WHERE**.
- D Nessuna delle risposte precedenti.

## 22 Le funzioni di aggregazione prevedono l'uso della clausola **GROUP BY**?

- A Sì, sempre.
- B No, mai.

- C No se la funzione di aggregazione viene specificata nel comando **SELECT** da sola senza altri campi.
- D Nessuna delle risposte precedenti.

## 23 Quale delle seguenti query SQL consente di ottenere come risultato i cognomi degli studenti che hanno riportato il voto massimo in storia?

- A 

```
SELECT cognome FROM Studenti WHERE  
materia = 'Storia' AND  
voto = (SELECT MAX(voto)  
FROM Studenti WHERE  
materia = 'Storia');
```
- B 

```
SELECT cognome FROM Studenti WHERE  
materia = 'Storia' AND  
voto = (SELECT MAX(voto)  
FROM Studenti);
```
- C 

```
SELECT cognome FROM Studenti WHERE  
materia = 'Storia' AND  
cognome = (SELECT MAX(voto)  
FROM Studenti WHERE  
materia = 'Storia');
```
- D 

```
SELECT cognome FROM Studenti WHERE  
materia = 'Storia' AND  
cognome = (SELECT MAX(cognome)  
FROM Studenti WHERE  
materia = 'Storia');
```

## 24 Quale delle seguenti query SQL consente di ottenere come risultato la città con più di 10 clienti?

- A 

```
SELECT Clienti FROM Citta  
GROUP BY Citta HAVING COUNT(*)>10;
```
- B 

```
SELECT Citta FROM Clienti  
GROUP BY Citta WHERE COUNT(*)>10;
```
- C 

```
SELECT Citta FROM Clienti  
GROUP BY Citta HAVING COUNT(*)>10;
```
- D 

```
SELECT Clienti FROM Citta  
GROUP BY Clienti HAVING COUNT(*)>10;
```

## 25 Una **join** tra due tabelle T1 e T2 senza alcuna condizione produce ...

- A ... il prodotto cartesiano  $T1 \times T2$  delle righe in cui la chiave primaria di T1 ha lo stesso valore della chiave esterna di T2.
- B ... il prodotto cartesiano completo  $T1 \times T2$ .
- C ... il prodotto cartesiano  $T1 \times T2$  delle righe in cui la chiave primaria di T1 ha lo stesso valore della chiave primaria di T2.
- D Nessuna delle risposte precedenti.



**26** Il comando SQL `"DELETE FROM Componenti;"` applicato alla base di dati utilizzata per gli esempi di questo capitolo provoca ...

- A ... l'eliminazione della prima riga della tabella *Componenti*.
- B ... l'eliminazione di tutte le righe della tabella *Componenti*.
- C ... l'eliminazione di tutte le righe della tabella *Componenti* e delle righe correlate della tabella *Composizione*.
- D Nessuna delle risposte precedenti.

**27** Il comando SQL `ALTER` permette ...

- A ... l'alterazione delle righe nelle tabelle del database.
- B ... la modifica di alcune caratteristiche della struttura delle tabelle del database.
- C ... la modifica di alcune caratteristiche della struttura di un qualsiasi oggetto del database.
- D Nessuna delle risposte precedenti.

**28** Indicare per ognuna delle seguenti istruzioni di SQL l'appartenenza alla categoria DDL o DML.

INSERT	
DROP	
UPDATE	DDL
CREATE	
SELECT	DML
ALTER	
DELETE	

**29** Una vista logica in SQL può essere realizzata con l'istruzione ...

- A ... `CREATE VIEW`
- B ... `CREATE TRIGGER`
- C ... `GRANT`
- D ... `REVOKE`


**30** Indicare quali delle seguenti istruzioni SQL possono essere usate nella gestione delle transazioni.

- A `COMMIT`
- B `ROLLON`
- C `ROLLBACK`
- D `REVOKE`
- E `SAVEPOINT`

## ESERCIZI

**1** Con riferimento al database progettato nell'esercizio 1 del capitolo A2 scrivere i comandi DDL necessari per realizzarne il DB-schema e quindi formulare le seguenti query in linguaggio SQL:

- a) per ogni prestito, il valore del prestito e i dati identificativi dei clienti che lo hanno stipulato;
- b) elenco di tutti i clienti che hanno almeno un deposito e almeno un prestito;
- c) elenco di tutti i clienti titolari di almeno un deposito ma non di un prestito;
- d) per ogni filiale, il numero dei titolari di conti correnti (contare ogni cliente una sola volta);
- e) elenco di tutte le filiali che hanno un patrimonio maggiore del più piccolo capitale di delle filiali di Livorno;
- f) il nome della/e filiale/i con il saldo medio più alto;
- g) elenco dei clienti che hanno un deposito presso tutte le filiali di Firenze;
- h) saldo medio dei vari clienti che vivono a Siena e hanno almeno due depositi;
- i) numero dei titolari di deposito per ogni filiale dove il saldo medio dei depositi sia superiore a 6500 euro;
- j) elenco di tutte le filiali che hanno un patrimonio maggiore ad almeno una filiale di Livorno;
- k) elenco dei clienti che hanno un solo deposito presso tutte la filiale «Agenzia 2» di Firenze;
- l) saldo medio delle filiali con saldo medio superiore a 2 milioni di euro;
- m) la filiale con il maggior saldo medio e il relativo saldo medio.

**2**  Con riferimento al database progettato nell'esercizio 2 del capitolo A2 scrivere i comandi DDL necessari per realizzarne il DB-schema e quindi formulare le seguenti query in linguaggio SQL:

- a) elenco che riporta data dell'infrazione, nome dell'agente che l'ha elevata, targa, modello e marca dell'auto multata;
- b) per ogni agente, matricola, nome e numero delle contravvenzioni elevate nell'anno 2011 con il relativo importo totale;
- c) elenco in cui per ogni automobilista si visualizzi codice fiscale, nome, indirizzo e numero di auto possedute;

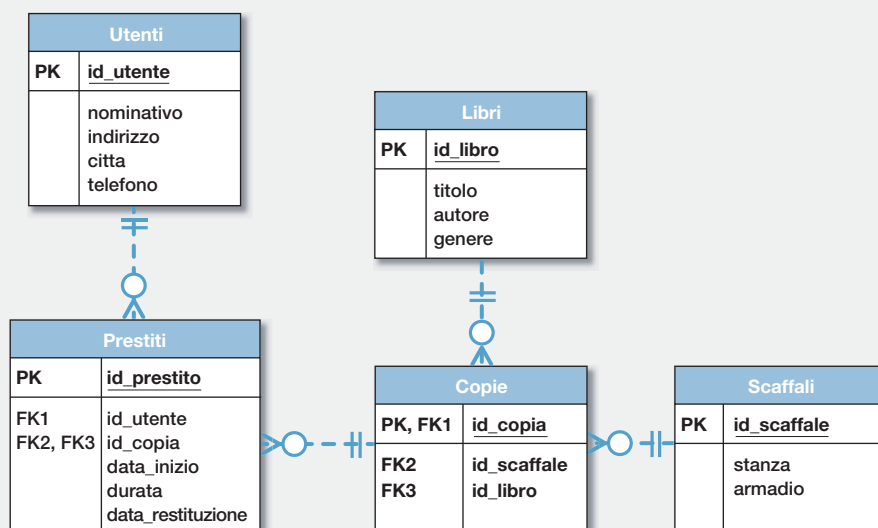


FIGURA 3

- d) per ogni denominazione di infrazione contestata ad auto di marca FIAT, indicare quante multe di quel tipo sono state emesse in un elenco del tipo: tipo\_infrazione, numero\_multe;
- e) nome e indirizzo di automobilisti cui sono state contestate almeno 3 infrazioni per «Divieto di sosta»;
- f) creare una nuova tabella Velocita\_Marzo con le infrazioni per «Eccesso di velocità» del mese di marzo 2011;
- g) targa, modello e marca dell'auto o delle auto più multata/e.

**3** Con riferimento allo schema relazionale di FIGURA 3, relativo alla base di dati per la gestione di una piccola biblioteca, scrivere i comandi DDL necessari per realizzarne il DB-schema e quindi formulare le seguenti query in linguaggio SQL:

- a) numero dei libri presi in prestito dai vari utenti nel corrente mese di gennaio. Fornire un elenco del tipo id\_utente, nome, n\_libri;
- b) elenco delle copie che sono state effettivamente restituite dopo la data presunta di scadenza del prestito (di ciascuna copia mostrare il titolo e il nome dell'utente che l'ha presa in prestito);
- c) una copia viene detta disponibile se non è correntemente in prestito: produrre l'elenco

delle copie non disponibili del libro «I promessi sposi» (di ciascuna di tali copie mostrare il codice);

- d) i titoli dei libri di cui la biblioteca possiede almeno 3 copie (elenco del tipo titolo, n\_copie);
- e) lista degli utenti che hanno avuto in prestito almeno uno dei volumi contenuti negli scaffali della stanza 7;
- f) titolo dei libri le cui copie non sono collocate tutte nel medesimo scaffale;
- g) titolo del/i libro/i più prestato/i tenendo conto dei prestiti complessivi delle varie copie.

**4** Dato lo schema relazionale di FIGURA 4, relativo a una banca dati cinematografica, scrivere i comandi DDL necessari per realizzarne il DB-schema e quindi formulare le seguenti query SQL:

- a) genere/i con il massimo numero di film specificandone il numero;
- b) id\_attore e nominativo degli attori che hanno recitato almeno due volte in film del genere «Horror»;
- c) titolo e numero di premi vinti da ogni film premiato nell'anno 2010;
- d) tra i vari film di genere «Cortometraggio» trovare quello/i di minor durata;
- e) elenco in ordine di anno i titoli dei film in cui ha recitato Brad Pitt;



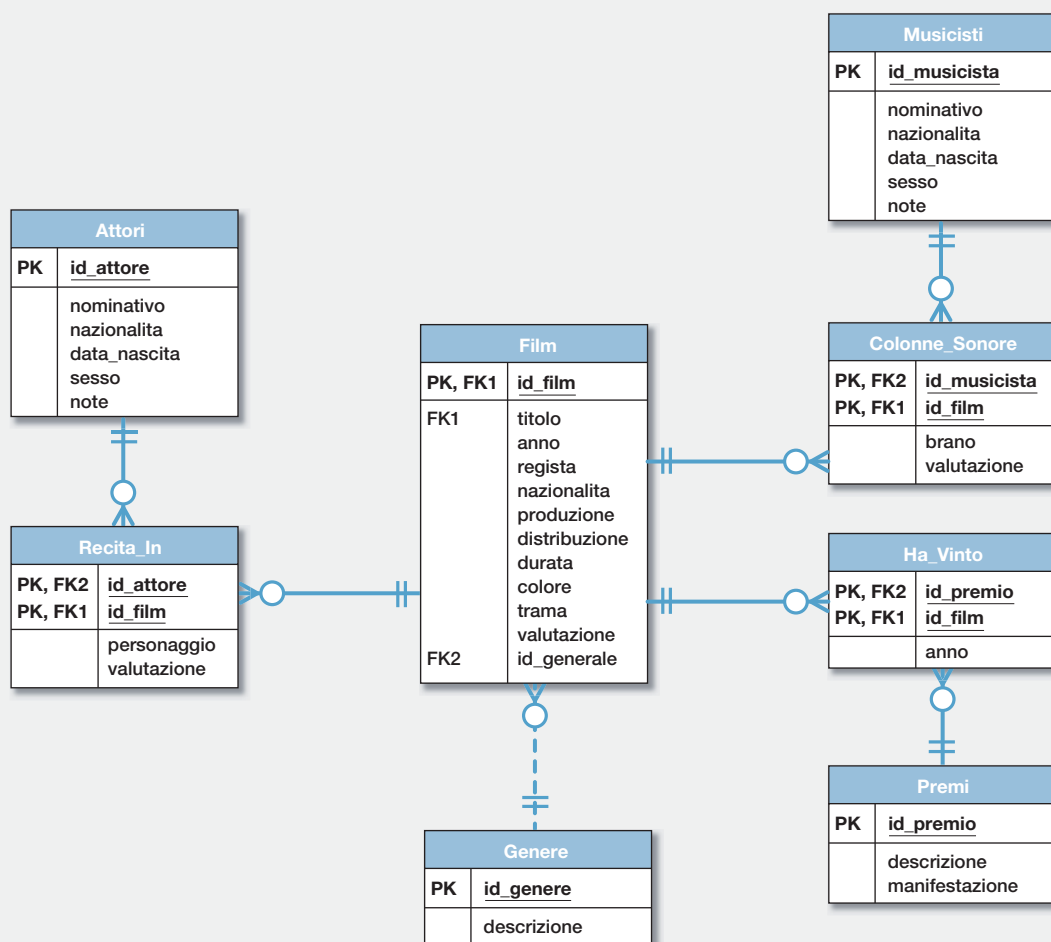


FIGURA 4

- f) elenco dei titoli dei film premiati la cui colonna sonora sia stata scritta da musicisti di nazionalità italiana;
  - g) per ogni attore, il numero di film in cui ha recitato;
  - h) per ogni nazionalità, il numero di premi Oscar vinti dal 2000 al 2011;
  - i) titolo dei film italiani premiati nel 2009 con indicazione di premio (descrizione) e manifestazione;
  - j) anno e titolo dei film in cui abbiano recitato Brad Pitt e Angelina Jolie;
  - k) il/i film/s di maggior durata;
  - l) per ogni attrice si vuole conoscere il numero di film in cui ha recitato quando era ancora minorenne;
  - m) id\_musica e nominativo dei musicisti che hanno scritto brani per colonne sonore di film premiati del genere «Commedia»;
  - n) titolo, nazionalità e numero di attori di film del 2011 il cui cast sia composto da meno di 10 attori;
  - o) dati di attori che non hanno mai recitato in film del genere «Horror»;
  - p) per gli anni tra il 2000 e il 2010, la valutazione media annua ricevuta dagli attori che hanno recitato in film Inglesi (elenco del tipo anno, nominativo, valutazione\_media);
  - q) la/le nazionalità dei film con il maggior il numero di premi vinti dal 2000 al 2009 alla manifestazione degli Oscar con indicazione del numero di premi.
- 5** Un'azienda commerciale vuole gestire i dati relativi agli ordini di materiale ricevuti dai propri clienti. L'azienda vende un certo numero di prodotti, ognuno dei quali è caratterizzato da un codice, un nome, una descrizione, un prezzo di acqui-

sto, un prezzo di vendita e dalla giacenza attuale.

I prodotti sono classificati secondo specifiche categorie che si vogliono gestire separatamente mediante un codice e relativa descrizione.

L'azienda si rivolge a fornitori esterni per l'acquisizione dei prodotti in listino: un prodotto viene acquistato da un solo fornitore che può fornire prodotti diversi. Ogni fornitore è definito da un codice, una ragione sociale, un indirizzo, il CAP, la città, la provincia e il telefono.

Gli stessi attributi che definiscono i fornitori, sono utilizzati per descrivere anche i clienti che inoltrano gli ordini all'azienda. Ogni ordine è relativo a un certo cliente ed è identificato da un codice, da una data di ricezione, dalla data di evasione (se tale campo non è avvalorato l'ordine si intende ancora inevaso) e dalle spese di trasporto. In ogni ordine possono essere richiesti diversi prodotti del listino per ognuno dei quali viene indicata la quantità ordinata.

In base allo schema di FIGURA 5, scrivere i comandi DDL necessari per realizzare la base di dati relativa e quindi formulare le seguenti query SQL:

- a) fornitore/i da cui si acquista il prodotto/i più costoso/i (lista del tipo fornitore, prodotto, prezzo\_acquisto);

- b) dati dei clienti che nell'anno in corso hanno acquistato sia il prodotto P01 che il prodotto P41, ma non il prodotto P35;
- c) ammontare del valore dei prodotti evasi nel mese di gennaio 2009;
- d) lista del tipo cliente, importo totale ordini per l'anno 2010;
- e) per ogni prodotto, quantità ordinata dai clienti nel 2011 suddivisa per provincia;
- f) categoria/e che nel 2010 è stata la più redditizia in termini economici (guadagno).

6 Con riferimento al database progettato nell'esercizio 3 del capitolo A2 scrivere i comandi DDL necessari per realizzare il DB-schema di un database che lo implementa; quindi formulare le seguenti query SQL:

- a) nome, ruolo e data di nascita dei calciatori che hanno militato nella Fiorentina nel 2010;
- b) nome e ruolo dei calciatori che hanno militato nella squadra che nel 2005 ha vinto lo scudetto;
- c) nome, ruolo e totale delle reti realizzate in quel ruolo da ogni giocatore nel corso della sua carriera agonistica in serie A;
- d) nominativo/i e goal di chi ha realizzato in carriera il maggior numero di reti;

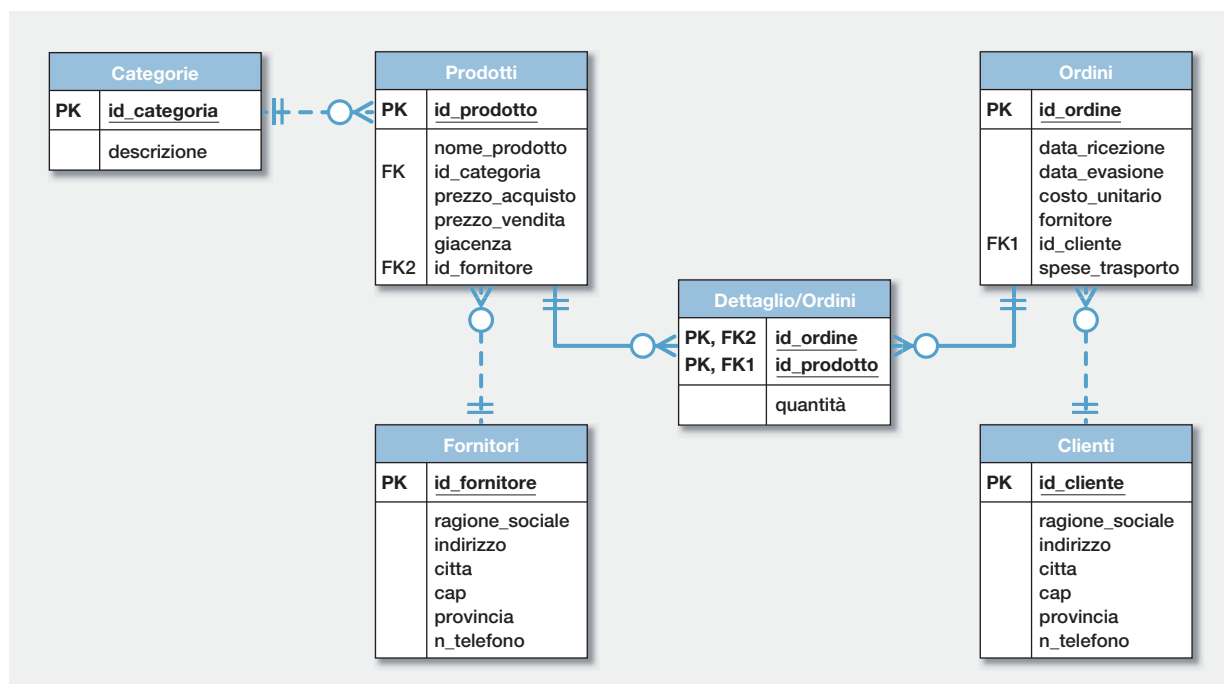


FIGURA 5

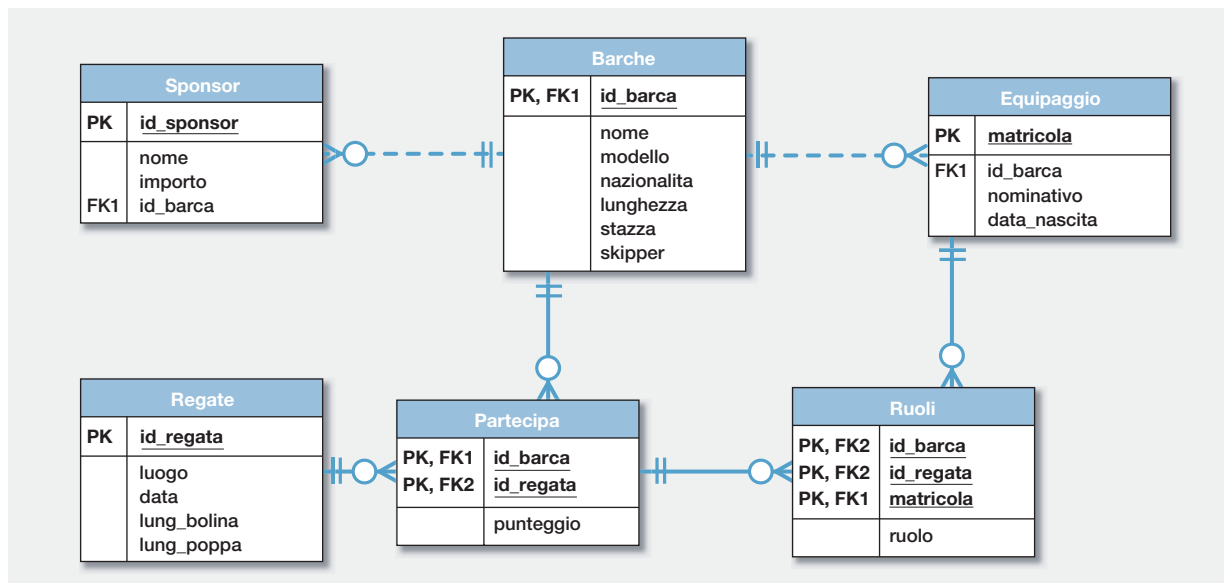


FIGURA 6

- e) nome dei calciatori che in carriera abbiano giocato sia nel Milan che nell'Inter ma non nella Juventus;
- f) nominativo dei calciatori che, nella loro carriera, abbiano vinto almeno due campionati.

**7** La base di dati per la gestione delle regate di una coppa internazionale contiene le informazioni descritte nel seguito. Ogni regata è caratterizzata dal luogo, dalla data e dai dati sul percorso (lunghezza del tratto di bolina e lunghezza del tratto di poppa espresse in miglia nautiche). Le barche concorrenti sono caratterizzate dal nome, dalla nazionalità, dal modello, dalla lunghezza, dalla stazza e dallo skipper. Le barche sono finanziate da sponsor (anche più di uno), e cioè da persone delle quali si conoscono il nome, il codice fiscale, l'indirizzo e l'ammontare della somma fornita. Si conosce inoltre il punteggio totalizzato da una barca in ogni regata cui ha partecipato. Ogni barca ha un equipaggio composto da un numero variabile fra le 15 e le 20 persone, ciascuna con la sua nazionalità e un proprio ruolo (che può variare di regata in regata). In base allo schema di FIGURA 6, che rappresenta la situazione descritta, scrivere i comandi DDL necessari per realizzare il DB-schema di un database che lo implementa e quindi formulare le seguenti query SQL:

- a) elenco delle barche che hanno partecipato alla regate di San Francisco e di Valencia ma non a quella di Bergen;

- b) la barca/barche che alla fine del torneo hanno totalizzato il maggior numero di punti;
- c) per ogni barca una lista del tipo <nome barca>, <totale importo sponsorizzazioni>;
- d) elenco dell'equipaggio della barca «Mercante di Venezia» nella regata di Dover del 20/02/2008 nella forma Nominativo, Ruolo;
- e) tutti i dati di quelle barche che nel loro equipaggio contano almeno tre italiani;
- f) la barca/barche che nelle varie regate hanno totalizzato un punteggio medio maggiore della media dei punteggi totalizzati da tutte le barche.

**8** Con riferimento al database progettato nell'esercizio 4 del capitolo A2 scrivere i comandi DDL necessari per realizzare il DB-schema di un database che lo implementa e formulare le seguenti query SQL:

- a) lista degli esaminandi che non hanno superato prove previste per l'appello d'esame del 22/01/2010 in un elenco del tipo skill\_card, nominativo, tipo\_prova, percentuale;
- b) esaminando/i (skill card e nominativo) che nell'anno 2011 ha/hanno subito il maggior numero di insuccessi nelle prove d'esame;
- c) tipologia/e di prova che risulta/no essere stata/e nel tempo la/le più difficile/i da superare (numero totale di insuccessi);
- d) valutazione media per ogni tipo di prova riportata nei vari appelli del 2010 in un elenco

dei tipo `data_espello`, `tipo_prova`, `percentuale_media`;

- e) elenco di coloro che avendo superato tutti gli esami ECDL hanno conseguito la patente europea per l'uso del computer.

## LABORATORIO

Per il database *Classic models* (<http://www.eclipse.org/birt/phoenix/db/>) sono disponibili online il DB-schema per la generazione in ambiente My-SQL e i file per il caricamento dei dati nelle singole tabelle. Il diagramma di FIGURA 7 rappresenta le tabelle del database e le relazioni tra loro definite.

Eseguire, utilizzando un qualsiasi *client* per il server DBMS My-SQL, le seguenti query in linguaggio SQL sul database:

- 1) elenco dei clienti con il termine *Gift* nella denominazione;
- 2) elenco degli impiegati con indicazione del responsabile;
- 3) elenco dei pagamenti dell'anno 2004 aventi importo superiore a 100 000 con indicazione della nazionalità del cliente;
- 4) elenco degli ordini dell'anno 2004 gestiti dall'impiegato «King»;
- 5) elenco dei clienti gestiti da impiegati residenti negli USA;
- 6) elenco dei clienti che hanno eseguito ordini nel corso dell'anno 2004, senza ripetizioni;
- 7) importo totale dei pagamenti per ciascuno degli anni successivi al 2005.
- 8) elenco degli ordini dell'anno 2004 aventi importo totale superiore a 10 000 e con indicazione per ciascuno di essi dell'importo totale e del numero di articoli ordinati;
- 9) elenco totale degli ordini del cliente numero 333 con indicazione per ognuno del numero complessivo di modelli ordinati;
- 10) elenco dei «saldi» (differenza tra somma dei pagamenti e importo complessivo degli ordini) di tutti i clienti;
- 11) elenco dei pagamenti del 2005 con importo superiore alla media degli importi dei pagamenti del 2004;
- 12) elenco degli uffici con il minimo numero di impiegati;
- 13) elenco dei clienti che nel corso del 2005 ha effettuato un numero di ordini inferiore alla media;
- 14) elenco dei clienti che ha effettuato nel 2005 ordini di importo complessivo superiore alle media degli importi complessivi degli ordini dello stesso anno.

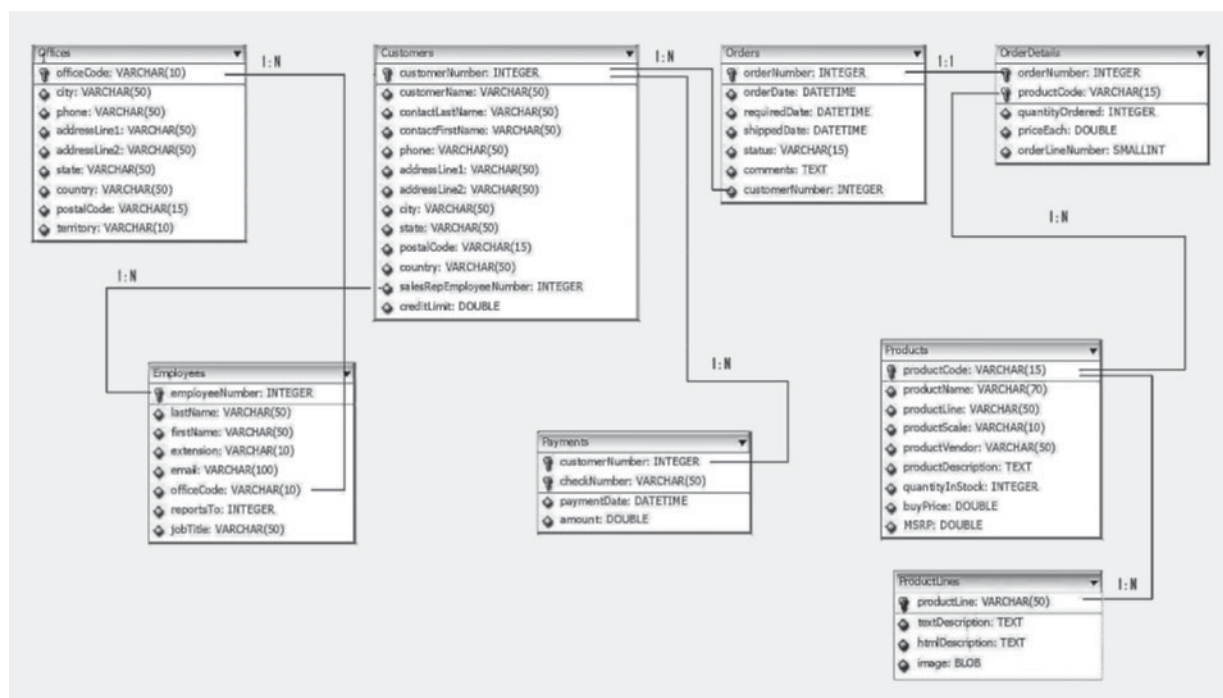


FIGURA 7

**well-know**

ben conosciuto

**useful**

utilizzabile

**to relay**passare, trasmettere,  
comunicare, mandare**statement**

dichiarazione, istruzione

**nowadays**

oggi, al giorno d'oggi

**on the spot**

sul posto, interattivamente

**to envelope**

avvolgere

**early stages**

fasi iniziali

**embedded**

incorporato

**binding**

vincolante

## 1.1 Introduction

MySQL is a relational database server that supports the well-known SQL (Structured Query Language) database language. Therefore, MySQL is named after the language that developers use to store, query, and later update data in a MySQL database. In short, SQL is the native language of MySQL. [...]

## 1.2 Database, Database Server, and Database Language

SQL (Structured Query Language) is a database language used for formulating statements processed by a database server. In this case, the database server is MySQL. The first sentence of this paragraph contains three important concepts: *database*, *database server*, and *database language*. We begin with an explanation of each of these terms.

What is a *database*? [...] A **database** consists of some collection of persistent data that is used by the application systems of some given enterprise and managed by a database-management system. [...]

Data in a database becomes useful only if something is done with it. According to the definition, data in the database is managed by a separate programming system. This system is called a *database server* or *database management system (DBMS)*. [...] The database server alone knows where and how data is stored. [...] A **database server** is a collection of programs that enables users to create and maintain a database.

A database server never changes or deletes the data in a database by itself; someone or something has to give the command for this to happen. Examples of commands that a user could give to the database server are “delete all data about the vehicle with the registration plate number DR-12-DP” or “give the names of all the companies that haven’t paid the invoices of last March.” However, users cannot communicate with the database server directly; an application must present the commands to a database server. An application always exists between the user and the database server. [...]

Commands are relayed to a database server with the help of special languages, called *database languages*. Users enter commands, also known as statements, that are formulated according to the rules of the database language, using special software; the database server then processes these commands. Every database server, regardless of manufacturer, possesses a database language. Some systems support more than one. All these languages are different, which makes it possible to divide them into groups. The *relational database languages* form one of these groups. An example of such a language is SQL. [...]

## 1.4 What is SQL?

As already stated, SQL (Structured Query Language) is a *relational database language*. Among other things, the language consists of statements to insert, update, delete, query, and protect data. The following statements can be formulated with SQL:

- Insert the address of a new employee.
- Delete all the stock data for product ABC.
- Show the address of employee Johnson.
- Show the sales figures of shoes for every region and for every month.
- Show how many products have been sold in London the last three months.
- Make sure that Mr. Johnson cannot see the salary data any longer.

Many vendors already have implemented SQL as the database language for their database server. [...]

We call SQL a relational database language because it is associated with data that has been defined according to the rules of the relational model. [...]

Because SQL is a relational database language, for a long time it has been grouped with the declarative or nonprocedural database languages. By *declarative* and *nonprocedural*, we mean that users (with the help of statements) have to specify only *which* data elements they want, not *how* they must be accessed one by one. Well-known languages such as C, C++, Java, PHP, Pascal, and Visual Basic are examples of procedural languages.

Nowadays, however, SQL can no longer be called a pure declarative language. Since the early 1990s, many vendors have added procedural extensions to SQL.

These make it possible to create procedural database objects such as *triggers* and *stored procedures*; [...] Traditional statements such as IF-THEN-ELSE and WHILE-DO have also been added. Although most of the well-known SQL statements are still not procedural by nature, SQL has changed into a hybrid language consisting of procedural and nonprocedural statements. Recently, MySQL has also been extended with these procedural database objects.

SQL can be used in two ways. First, SQL can be used *interactively*. For example, a user enters an SQL statement on the spot, and the database server processes it immediately. The result is also immediately visible. Interactive SQL is intended for application developers and for end users who want to create reports themselves.

The products that support interactive SQL can be split in two groups: the somewhat old-fashioned products with a terminal-like interface and those with a modern graphical interface. MySQL includes a product with a terminal-like interface that bears the same name as the database server: `mysql`. [...]

First, an SQL statement is entered (`SELECT * FROM PLAYERS`); the result is shown underneath as a table.

However, some products have a more graphical interface available for interactive use, [...] such as MySQL Query Browser, SQLyog, php-MyAdmin, Navicat and WinSQL.

The second way in which SQL can be used is called *preprogrammed* SQL. Here, the SQL statements are embedded in an application that is written in another programming language. Results from these statements are not immediately visible to the user but are processed by the *enveloping* application. Preprogrammed SQL appears mainly in applications developed for end users. These end users do not need to learn SQL to access the data, but they work from simple screens and menus designed for their applications. [...]

In the early stages of the development of SQL, only one method existed for preprogrammed SQL, called *embedded* SQL. In the 1980s, other methods appeared. The most important is called *call level interface* SQL (CLI SQL). Many variations of CLI SQL exist, such as ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity). [...]

The different methods of preprogrammed SQL are also called the *binding styles*.

The statements and features of interactive and preprogrammed SQL are virtually the same. By this, we mean that most statements that can be entered and processed interactively can also be included (embedded) in an SQL application.

[...]

[Rick F. van der Lans, *SQL for MySQL Developers*, Addison Wesley Publishing Company, 2007]

## QUESTIONS

- a** What does the acronym SQL stand for?
- b** What is the difference between a database system and a database?
- c** What are the main types of graphical interface available for using SQL interactively?
- d** How has SQL developed into a hybrid language?

# Accesso a una base di dati in linguaggio Java con JDBC

## DBMS My-SQL e linguaggio Java

My-SQL di Oracle Corporation è un DBMS estremamente diffuso, in particolare per le applicazioni web. My-SQL prevede diversi driver (denominati *connector*) per l'esposizione di API (*Application Program Interface*) verso diversi linguaggi di programmazione, tra cui JDBC per Java.

1. JDBC è indipendente dallo specifico DBMS utilizzato, per cui gli esempi di codice presentati hanno validità generale.
2. Gli esempi sono volutamente ridotti all'essenziale, dato che hanno il solo scopo di dimostrare l'interazione con il DBMS.

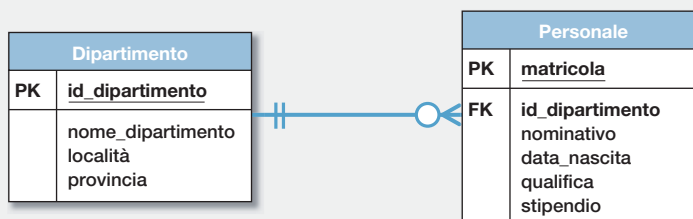
Come è stato già anticipato nei capitoli precedenti, il linguaggio SQL può essere utilizzato come linguaggio «ospite» di un linguaggio di programmazione: questa modalità è in generale relativa all'implementazione di applicazioni software, spesso di tipo gestionale, che prevedono un *front-end* per utenti che interagiscono mediante una GUI (*Graphical User Interface*) con i dati contenuti in un database. Il linguaggio di programmazione permette la realizzazione dell'interfaccia utente e l'implementazione delle procedure di elaborazione dei dati, mentre ai comandi SQL è demandata l'interazione con il database: ricerca, inserimento, aggiornamento e cancellazione dei dati.

### ESEMPIO

Un programma di gestione aziendale integrata consente agli operatori di accedere ai dati relativi alla propria realtà aziendale memorizzati in un database gestito da un DBMS per compiere attività come la gestione della contabilità, della fatturazione, del magazzino e così via.

Nel caso in cui il linguaggio di programmazione ospitante sia Java, il modulo software che permette di accedere a un database è noto come JDBC (*Java DataBase Connectivity*): in questo capitolo vedremo come utilizzare la tecnologia JDBC per interagire con il DBMS My-SQL<sup>1</sup>, un prodotto distribuito gratuitamente da parte di Oracle Corporation.

Gli esempi di questo capitolo<sup>2</sup> sono basati su un semplice database che rappresenta la realtà di un'azienda suddivisa in dipartimenti distribuiti geograficamente, ognuno dei quali ha un certo numero di impiegati. Di seguito se ne riporta il diagramma delle tabelle



e il DB-schema in linguaggio SQL:



```
CREATE DATABASE Azienda;
```

```
USE Azienda;
```



```
CREATE TABLE Dipartimento (  
    id_dipartimento VARCHAR(3) NOT NULL,  
    nome_dipartimento VARCHAR(30),  
    localita VARCHAR(20),  
    provincia VARCHAR(2),  
    CONSTRAINT chiave_primaria PRIMARY KEY (id_dipartimento)  
);
```

```
INSERT INTO Dipartimento (id_dipartimento, nome_dipartimento,  
localita, provincia) VALUES  
( 'D1', 'ALFA', 'Livorno', 'LI'),  
( 'D2', 'BETA', 'Pisa', 'PI'),  
( 'D3', 'GAMMA', 'Piombino', 'LI'),  
( 'D4', 'DELTA', 'Lucca', 'LU'),  
( 'D5', 'OMEGA', 'Firenze', 'FI');
```

```
CREATE TABLE Personale (  
    matricola VARCHAR(5) NOT NULL,  
    id_dipartimento VARCHAR(3),  
    nominativo VARCHAR(50),  
    data_nascita DATE,  
    qualifica VARCHAR(2),  
    stipendio DOUBLE,  
    CONSTRAINT chiave_primaria PRIMARY KEY (matricola),  
    CONSTRAINT dipartimenti_personale FOREIGN KEY (id_dipartimento)  
        REFERENCES Dipartimento(id_dipartimento)  
);
```

```
INSERT INTO Personale (matricola, id_dipartimento, nominativo,  
                        data_nascita, qualifica, stipendio) VALUES  
( '00013', 'D1', 'ROSSI PIERO', '1965-01-15', '01', 1640.00),  
( '00034', 'D2', 'NERI GIOVANNI', '1964-08-05', '02', 2530.00),  
( '00075', 'D4', 'ARNETTI MARIA', '1967-01-15', '01', 1750.00),  
( '04346', 'D3', 'BELLI DANIELA', '1967-02-25', '01', 1740.00),  
( '04434', 'D2', 'VERDI MARCO', '1977-05-09', '03', 3480.00),  
( '04450', 'D3', 'SANDRI DONATA', '1969-05-01', '02', 2470.00),  
( '04532', 'D3', 'GIANNINI PIETRO', '1968-06-04', '01', 1580.00),  
( '04541', 'D3', 'TESINI MARIO', '1968-12-28', '02', 2740.00),  
( '04551', 'D1', 'BIANCHI MAURO', '1968-07-09', '02', 2580.00),  
( '04717', 'D2', 'PIERINI MARIO', '1969-06-20', '01', 1520.00),  
( '04794', 'D2', 'CARLETTI PAOLO', '1971-07-02', '01', 1670.00),  
( '05019', 'D4', 'SOLDANI GIULIO', '1973-03-27', '02', 2480.00),  
( '05462', 'D3', 'LAPINI PAOLO', '1967-01-11', '03', 3960.00),  
( '05477', 'D4', 'BRESCHI CARLA', '1976-05-02', '02', 2600.00);
```



Le varie soluzioni proposte nel seguito del capitolo prevedono una stretta integrazione del linguaggio SQL come ospite del linguaggio Java: l'interazione con il DBMS avviene sempre mediante stringhe che rappresentano comandi SQL che specifici metodi di classi Java inviano al DBMS stesso.

## 1 Architettura client/server e API Java DataBase Connectivity

L'architettura di riferimento in cui si opera con JDBC è il classico modello client/server (FIGURA 1), in cui:

- l'applicazione client e il server DBMS sono normalmente eseguiti su computer distinti che comunicano tramite una rete locale o geografica;
- un singolo server può servire numerosi client;
- un singolo client può utilizzare più server.

Un'analisi più approfondita evidenzia i seguenti aspetti:

- in un'architettura client/server coesistono applicazioni client e server DBMS eterogenei e, di conseguenza, si individuano almeno due componenti distinti: il *front-end* (costituito da GUI o da interfacce web) e il *back-end* (i server DBMS);
- è necessario stabilire come le applicazioni client comunicano con i server DBMS, in particolare come gestire i risultati delle *query*;
- è importante rendere l'accesso ai dati indipendente dallo specifico server DBMS utilizzato in modo che l'eventuale cambiamento non imponga modifiche alle applicazioni client.

**OSSERVAZIONE** Effettuare un accesso ai database indipendente dallo specifico server DBMS utilizzato rappresenta il problema principale nell'implementazione di applicazioni distribuite in rete la cui soluzione richiede funzionalità di adattamento e di conversione che permettano lo scambio di informazione tra sistemi e applicazioni eterogenei.

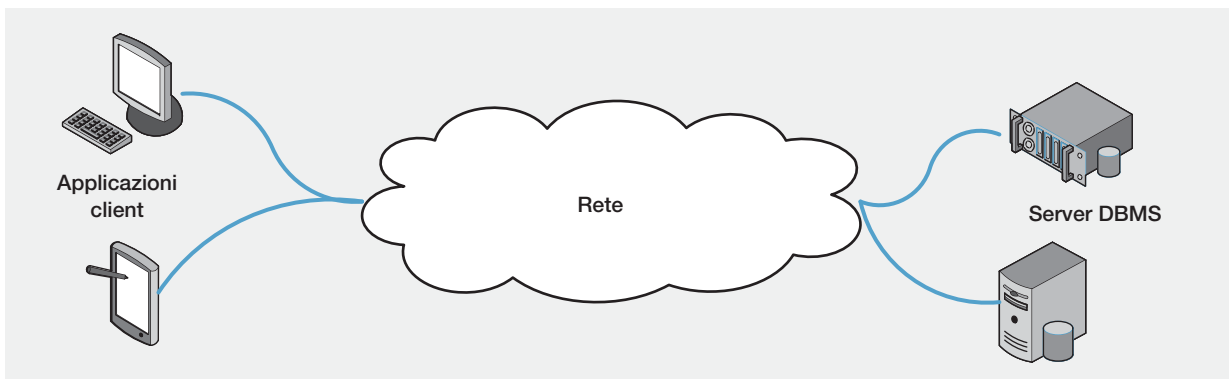


FIGURA 1

Per far fronte a queste problematiche nello sviluppo di applicazioni software si ricorre a tecnologie standard implementate in forma di API (*Application Program Interface*):

- **ODBC (*Open DataBase Connectivity*)**: sviluppata da Microsoft negli anni '90 del secolo scorso, si propone l'intento di definire un'interfaccia standard per l'accesso da codice in linguaggio C/C++ a database in modo indipendente sia dal sistema operativo sia dal DBMS;
- **JDBC (*Java DataBase Connectivity*)**: specificatamente sviluppata come componente standard del *Java Development Kit* a partire dal 1997, permette l'accesso a un database in modo indipendente dal DBMS e dalla piattaforma di esecuzione.

Un'applicazione software che opera su un database è un programma che invoca specifiche funzioni API per accedere ai dati gestiti da un DBMS. La tipica sequenza di interazione è la seguente:

- connessione alla sorgente dei dati (database contenuto in un DBMS);
- esecuzione di comandi/*query* SQL;
- recupero dei risultati ed eventuale gestione degli errori;
- disconnessione dalla sorgente dei dati.

Nel contesto descritto l'architettura software di riferimento è quella illustrata (FIGURA 2), in cui il *driver manager* espone all'applicazione software una API standard, anche se i driver specifici dei vari DBMS dispongono di una API proprietaria.

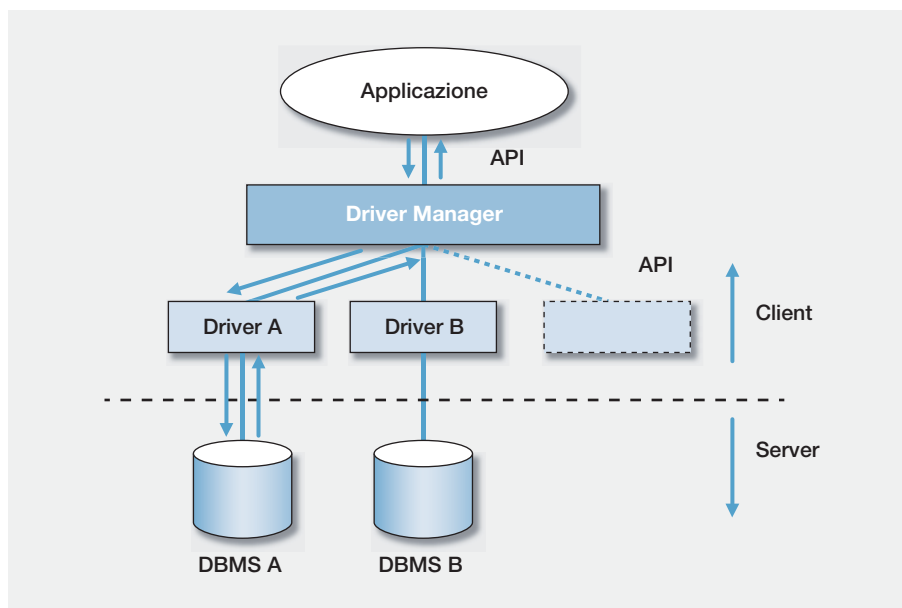


FIGURA 2

**Driver manager.** È un componente software che gestisce la comunicazione tra l'applicazione e i driver specifici dei singoli DBMS; risolve problematiche comuni a tutte le applicazioni:

- selezionare il driver da utilizzare sulla base delle informazioni fornite dall'applicazione;
- gestire l'invocazione delle funzioni dei driver.

**Driver.** Sono generalmente librerie caricate dinamicamente che implementano le funzioni API; ne esiste una specifica per ogni particolare DBMS che ha il compito di:

- nascondere le differenze di interazione dovute ai vari DBMS, sistemi operativi e protocolli di rete impiegati;
- trasformare le invocazioni delle funzioni API nel «dialetto» SQL usato dal DBMS, o nelle corrispondenti funzioni API supportate dal DBMS;
- gestire le transazioni, eseguire i comandi e le *query* SQL, inviare e recuperare i dati, gestire gli errori ecc.

**DBMS.** È il server che gestisce la base di dati; in questo contesto:

- riceve le richieste esclusivamente nello specifico linguaggio supportato;
- esegue i comandi e le *query* SQL ricevute dal driver e restituisce i relativi risultati.

Il primo componente software conforme a questa architettura è stato ODBC (*Open DataBase Connectivity*), introdotto da Microsoft nel 1991 e oggi supportato praticamente da tutti i DBMS relazionali. ODBC presenta le seguenti caratteristiche:

- è una API sviluppata in linguaggio C che richiede il ricorso ad API intermedie per essere usata con altri linguaggi di programmazione; inoltre la scelta del linguaggio C per l'implementazione non lo rende completamente portabile;
- il linguaggio SQL supportato da ODBC comprende un insieme minimale di funzionalità;
- il linguaggio SQL supportato da ODBC comprende sia *query* statiche (per applicazioni software convenzionali), sia *query* dinamiche (per applicazioni software che creano interrogazioni in modo interattivo): nel primo caso eventuali errori SQL sono riportati in fase di compilazione del codice;
- l'interfaccia esposta da ODBC non è sempre di agevole utilizzazione.

Per ovviare a questi inconvenienti nel 1996 Sun Microsystems<sup>3</sup>, allo scopo di realizzare una API standard per l'accesso ai database in linguaggio Java, ha sviluppato la tecnologia JDBC. JDBC è una soluzione «*pure Java*» inclusa nello JDK (*Java Development Kit*, il sistema di sviluppo del linguaggio Java) che espone un'interfaccia flessibile per la gestione dell'interazione con i DBMS garantendo al tempo stesso l'indipendenza dalla piattaforma su cui viene eseguita.

In FIGURA 3 è schematizzata l'architettura generale di JDBC. JDBC prevede i seguenti quattro tipi di driver.

- **Tipo 1 (JDBC-ODBC bridge).** L'accesso al DBMS da parte di JDBC avviene tramite un driver ODBC che esiste in pratica per ogni DBMS relazionale; dato che il driver ODBC non è in genere portabile tra diverse piattaforme di esecuzione, questa soluzione è da adottarsi solo quan-

3. Sun Microsystems è stata acquistata nel 2009 da Oracle Corporation che ha mantenuto il supporto all'evoluzione del linguaggio Java e alle tecnologie correlate.

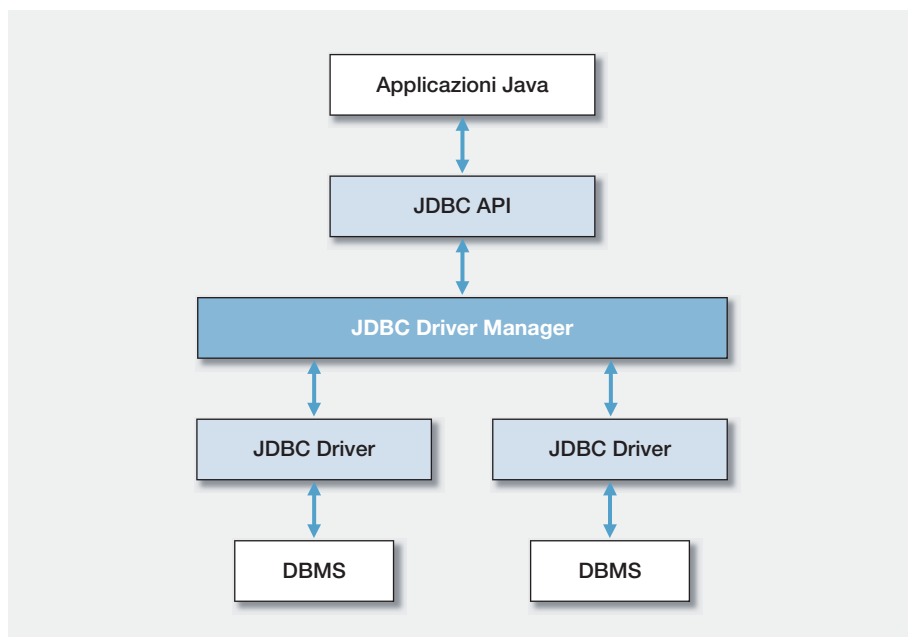


FIGURA 3

do non esistono alternative perché viola il principio «*write once, run anywhere*» che caratterizza le applicazioni Java.

- **Tipo 2 (driver realizzato solo in parte in Java).** Le chiamate inoltrate a JDBC sono convertite in chiamate alla API del DBMS, di conseguenza il driver contiene codice Java che invoca funzioni normalmente codificate in linguaggio C/C++; anche in questo caso il driver non è portabile tra piattaforme di esecuzione diverse.
- **Tipo 3 (driver completamente realizzato in Java e DBMS *middleware*).** Le chiamate inoltrate a JDBC sono elaborate da un driver locale completamente codificato in Java che le trasforma utilizzando un protocollo indipendente da quello del server DBMS in invocazioni per un'applicazione *middleware* che a sua volta si interfaccia con lo specifico DBMS.
- **Tipo 4 (driver *pure Java* con connessione diretta al server DBMS).** Il driver JDBC è in questo caso completamente realizzato in Java e – essendo specifico per il DBMS utilizzato – converte le chiamate JDBC direttamente nel protocollo di rete usato dal server DBMS.

Lo schema di FIGURA 4 sintetizza graficamente le differenze tra i quattro tipi di driver JDBC.

### Middleware

Il termine *middleware* viene utilizzato per riferirsi a un insieme di componenti software intermediari tra applicazioni diverse. I componenti *middleware* sono spesso utilizzati come supporto per sistemi distribuiti complessi.

## 2 Connessione a un DBMS ed elaborazione di comandi e query SQL in linguaggio Java

Nel seguito ci riferiremo a una versione semplificata dell'architettura interna del package *java.sql* che implementa la API JDBC, schematizzata nella FIGURA 5.

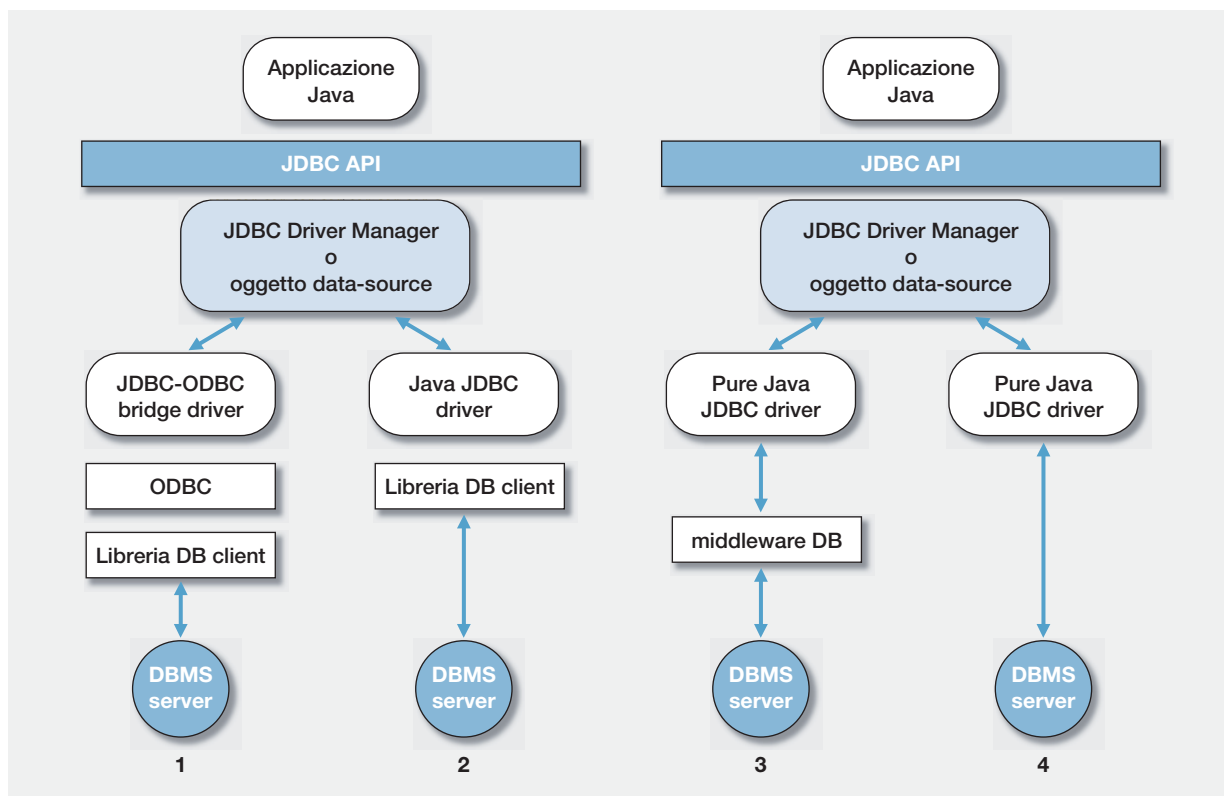


FIGURA 4

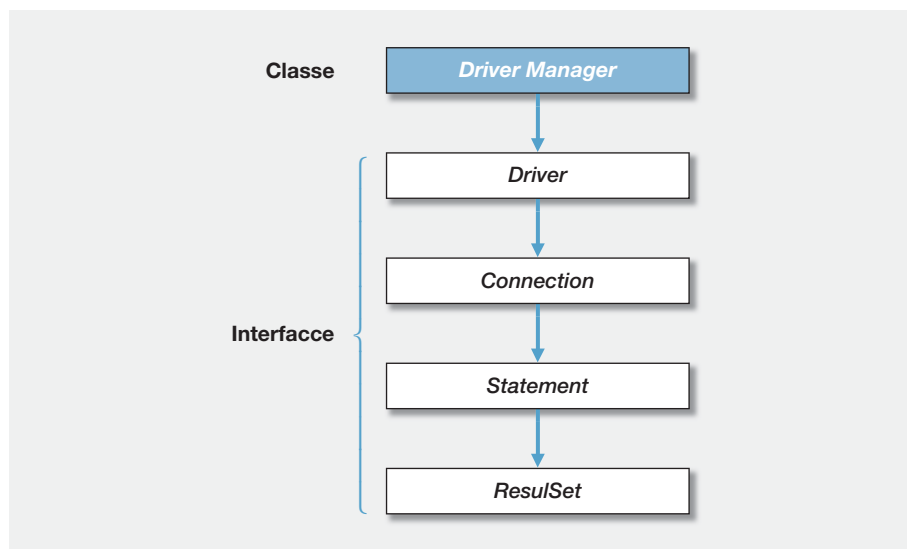


FIGURA 5

Prendiamo in esame i singoli passi previsti dalla API JDBC per l'interazione con un database contenuto in uno specifico DBMS.

## 0. Caricamento del driver

Il *driver manager* mantiene una lista di classi che implementano l'interfaccia *Driver*, ma la specifica implementazione di un driver deve essere registrata nel *driver manager*. Nelle prime versioni di JDBC la classe che

realizza il driver implementando l'interfaccia *Driver* doveva essere esplicitamente caricata in modo da forzarne la registrazione mediante invocazione del metodo statico *registerDriver* della classe *DriverManager* da parte del codice di inizializzazione della classe stessa. In alternativa era ed è possibile registrare direttamente un'istanza del driver.

#### ESEMPIO

La seguente istruzione registra il driver del DBMS My-SQL caricando la rispettiva classe

```
Class.forName("com.mysql.jdbc.Driver");
```

mentre la seguente istruzione registra esplicitamente il driver *bridge* per ODBC:

```
DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
```

**OSSERVAZIONE** Nelle versioni più recenti di JDBC la registrazione dei driver avviene automaticamente in fase di connessione con il server DBMS e non è necessario effettuarla esplicitamente.

## 1. Connessione al server DBMS

Il metodo statico *getConnection* della classe *DriverManager* restituisce un oggetto che implementa l'interfaccia *Connection* e che permette di interagire con il DBMS; oltre allo username e alla password, il metodo *getConnection* richiede come parametro un URL il cui formato dipende dallo specifico DBMS utilizzato.

### Driver JDBC per DBMS My-SQL

Il driver JDBC per il DBMS My-SQL denominato J/Connector può essere liberamente scaricato dal sito [www.mysql.it](http://www.mysql.it). Esso viene fornito nella forma di un package compresso in formato JAR che può essere importato nel progetto della propria applicazione Java.

#### ESEMPIO

L'accesso a un server My-SQL in esecuzione sul computer stesso in cui viene eseguita l'applicazione Java richiede di specificare il seguente URL:

```
jdbc:mysql://localhost:3306/database
```

dove *database* è il nome del database sul quale si intende operare. Nell'ipotesi di accedere al server DBMS con username *root* privo di password<sup>4</sup>, la richiesta di connessione al database *Azienda* si effettua con la seguente istruzione:

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/Azienda",
    "root", "");
```

**OSSERVAZIONE** Nell'URL dell'esempio precedente *localhost* è l'identificatore di rete del server DBMS, mentre 3306 è il numero di porta standard su cui è attivo My-SQL; questa configurazione è tipica per un server DBMS installato sulla stessa macchina che ospita il sistema di sviluppo allo scopo di verificare il codice prodotto.

La connessione al server DBMS avviene invocando il metodo *getConnection* della classe *DriverManager*, che restituisce un oggetto che implementa l'interfaccia *Connection*; se uno dei driver caricati riconosce l'URL fornito dal metodo, il driver stabilisce la connessione, in caso contrario viene sollevata un'eccezione di tipo *SQLException*.

4. Questa configurazione è assolutamente sconsigliata sotto il profilo della sicurezza, ma è accettabile per un server DBMS utilizzato esclusivamente per sviluppare e testare il codice Java che accede a un database.



Il codice del metodo *main* della seguente classe dimostra la connessione al database *Azienda* di esempio gestito da un server DBMS My-SQL in esecuzione sullo stesso sistema che esegue l'applicazione:

```
import java.sql.*;

public class TestConnection {

    public static void main(String[] args)
    // throws ClassNotFoundException
    {
        Connection con;

        String URL = "jdbc:mysql://localhost:3306";
        String database = "Azienda";
        // String driver = "com.mysql.jdbc.Driver";
        String user = "root";
        String password = "";

        try {
            // Class.forName(driver);
            con = DriverManager.getConnection(URL+"/"+database, user, password);
            System.out.println("Connessione al server DBMS effettuata.");
            con.close(); // disconnessione dal server DBMS
        }
        catch (SQLException exception) {
            System.out.println("Errore di connessione al server DBMS!");
        }
    }
}
```

**OSSERVAZIONE** Le righe di codice commentate nell'esempio precedente si riferiscono all'eventuale uso di un driver JDBC di tipo obsoleto, per il quale è necessaria la registrazione esplicita.

## 2. Esecuzione di comandi/query SQL

Per creare uno *statement* SQL – destinato a eseguire un comando o una *query* – si invoca il metodo *createStatement* di un oggetto che implementa l'interfaccia *Connection*; il metodo restituisce un oggetto che implementa l'interfaccia *Statement*.

Per eseguire un comando o una *query* è necessario distinguere tra interrogazioni dei dati e comandi DML/DDL; i metodi resi disponibili dall'interfaccia *Statement* a questo scopo sono riportati nella TABELLA 1.

TABELLA 1

execute	Esegue il comando o la <i>query</i> SQL fornita come parametro; restituisce <i>true</i> se il risultato è un oggetto che implementa l'interfaccia <i>ResultSet</i> , <i>false</i> altrimenti (l'eventuale risultato viene reso disponibile dai metodi <i>getResultSet</i> o <i>getUpdateCount</i> )
executeQuery	Esegue la <i>query</i> SQL fornita come parametro; restituisce un oggetto che implementa l'interfaccia <i>ResultSet</i>
executeUpdate	Esegue il comando SQL fornito come parametro; restituisce un valore intero che rappresenta il numero di record coinvolti nell'esecuzione del comando; può essere utilizzato per l'esecuzione di comandi DDL

Il seguente frammento di codice Java esegue una *query*:

```
Connection con;
ResultSet result;

...
Statement stat = con.createStatement();
result = stat.executeQuery("SELECT * FROM Personale");

mentre quello che segue esegue un comando DML

Connection con;
int result;

...
Statement stat = con.createStatement();
result = stat.executeUpdate("INSERT dipartimento(id_dipartimento, nome_dipartimento,
localita, provincia) VALUES ('D6', 'TETA', 'Rosignano', 'LI');");
```

**OSSERVAZIONE** Il simbolo terminatore dei comandi SQL «;», se non specificato, viene inserito automaticamente dal driver prima di inviare il comando al DBMS per l'esecuzione.

Il metodo *prepareStatement* di un oggetto che implementa l'interfaccia *Connection* consente di creare *statement* predefiniti eseguibili più volte (*prepared statement*); questa tecnica riduce i tempi di esecuzione del comando/*query* per comandi/*query* che devono essere eseguiti molte volte. Utilizzando *statement* di questo tipo è possibile inserire nella stringa che rappresenta il comando SQL uno o più parametri identificati dal simbolo «?». Alcuni metodi previsti dall'interfaccia *PreparedStatement* consentono di valorizzare i parametri – identificati dalla loro posizione nella stringa – prima dell'esecuzione; l'elenco che segue riporta solo quelli relativi ai tipi Java più comunemente utilizzati:

- *setTime*
- *setString*
- *setShort*
- *setNull*
- *setLong*
- *setInt*
- *setFloat*
- *setDouble*
- *setDate*

Il seguente frammento di codice Java crea uno *statement* predefinito con un parametro e lo istanzia con una stringa prima di eseguirlo:

```
Connection con;
ResultSet result;

...
PreparedStatement query = con.prepareStatement("SELECT * FROM Personale WHERE Nominativo = ?");
query.setString(1, 'BIANCHI MAURO');
result = query.executeQuery();
```



5. In alternativa un oggetto che implementa l'interfaccia *ResultSet* è restituito dal metodo *getResultSet* di un oggetto che implementa l'interfaccia *Statement* dopo l'invocazione del metodo *execute*.

### 3. Recupero dei risultati

Il metodo *executeQuery* definito dalle interfacce *Statement* e *PreparedStatement* restituisce un oggetto che implementa l'interfaccia *ResultSet*<sup>5</sup>. Un oggetto di questo può contenere un risultato composto da una sequenza di record che sono iterabili mediante il metodo *next*: l'accesso ai valori dei campi del singolo record avviene mediante metodi specifici che accettano come parametro l'indice posizionale del campo o il suo nome; l'elenco che segue riporta solo quelli relativi ai tipi Java più comunemente utilizzati:

- *getTime*
- *getString*
- *getShort*
- *getLong*
- *getInt*
- *getFloat*
- *getDouble*
- *getDate*

#### ESEMPIO

Il seguente frammento di codice Java esegue una *query* e visualizza il valore di alcuni campi del risultato:

```
Connection con;  
...  
Statement stat = con.createStatement();  
ResultSet result = stat.executeQuery("SELECT * FROM Personale;");  
while (result.next()) {  
    String nominativo = result.getString("nominativo");  
    float stipendio = result.getFloat("stipendio");  
    System.out.println(nominativo + ": " + stipendio);  
}  
result.close();  
stat.close();
```

**OSSERVAZIONE** La prima invocazione del metodo *next* di un oggetto che implementa l'interfaccia *ResultSet* posiziona il «cursore» dei record sulla prima posizione della sequenza dei risultati; ogni successiva invocazione lo posiziona sulla posizione successiva: una volta raggiunta l'ultima posizione il metodo restituisce il valore *false*.

**OSSERVAZIONE** L'invocazione dei metodi *close* sugli oggetti che implementano l'interfaccia *ResultSet* e *Statement* forza il rilascio immediato delle risorse che utilizzano.

Il seguente frammento di codice Java è analogo a quello dell'esempio precedente, ma accede ai valori dei singoli campi specificando la loro posizione nel risultato dell'interrogazione:

```
Connection con;
...
Statement stat = con.createStatement();
ResultSet result = stat.executeQuery("SELECT nominativo, qualifica,
                                     stipendio FROM Personale;");

while (result.next()) {
    String nominativo = result.getString(1);
    float stipendio = result.getFloat(3);
    System.out.println(nominativo + ": " + stipendio);
}
result.close();
stat.close();
```

## 4. Disconnessione dal server DBMS

La disconnessione dal server DBMS si effettua invocando il metodo *close* dell'oggetto restituito dall'invocazione del metodo *getConnection*.

Gli esempi che seguono illustrano l'intera sequenza di interazione di un'applicazione JDBC con un DBMS.



Il codice del metodo *main* della seguente classe visualizza l'elenco in ordine alfabetico del personale del database *Azienda* riportando il dipartimento a cui ogni unità di personale è assegnata (la connessione avviene a un server DBMS My-SQL in esecuzione sullo stesso sistema che esegue l'applicazione):

```
import java.sql.*;

public class TestQuery {

    public static void main(String[] args) {
        Connection con;
        Statement stat;
        ResultSet result;
        String URL = "jdbc:mysql://localhost:3306";
        String database = "Azienda";
        String user = "root";
        String password = "";

        try {
            con = DriverManager.getConnection(URL+"/"+database, user,
                                             password);
            System.out.println("Effettuata connessione al server DBMS.");
            stat = con.createStatement();
            result = stat.executeQuery("SELECT * FROM Personale,
                                     Dipartimento WHERE
                                     Personale.id_dipartimento =
                                     Dipartimento.id_dipartimento
                                     ORDER BY nominativo;");
```



```

        System.out.println("Elenco personale:");
        while (result.next()) {
            String nominativo = result.getString("nominativo");
            String dipartimento = result.getString("nome_dipartimento");
            System.out.println(nominativo + " - " + dipartimento);
        }
        result.close();
        stat.close();
        con.close();
        System.out.println("Effettuata disconnessione dal server DBMS.");
    }
    catch (SQLException exception) {
        System.out.println("Errore!");
    }
}
}

```

Con i dati del database di esempio viene prodotto il seguente output:

```

Effettuata connessione al server DBMS.
Elenco personale:
ARNETTI MARIA - DELTA
BELLI DANIELA - GAMMA
BIANCHI MAURO - ALFA
BRESCHI CARLA - DELTA
CARLETTI PAOLO - BETA
GIANNINI PIETRO - GAMMA
LAPINI PAOLO - GAMMA
NERI GIOVANNI - BETA
PIERINI MARIO - BETA
ROSSI PIERO - ALFA
SANDRI DONATA - GAMMA
SOLDANI GIULIO - DELTA
TESINI MARIO - GAMMA
VERDI MARCO - BETA
Effettuata disconnessione dal server DBMS.

```

## ESEMPIO

Il codice del metodo *main* della seguente classe visualizza il numero delle unità di personale del database *Azienda* prima e dopo l'esecuzione di un comando di eliminazione di alcuni record della tabella *Personale* (la connessione avviene a un server DBMS My-SQL in esecuzione sullo stesso sistema che esegue l'applicazione):

```

import java.sql.*;

public class TestDML {

    public static void main(String[] args) {
        Connection con;
        Statement stat;
        ResultSet result;
        int personale;
        String URL = "jdbc:mysql://localhost:3306";
    }
}

```



```

String database = "Azienda";
String user = "root";
String password = "";

try {
    con = DriverManager.getConnection(URL+"/"+database, user, password);
    System.out.println("Effettuata connessione al server DBMS.");
    stat = con.createStatement();
    result = stat.executeQuery("SELECT COUNT(*) AS numero FROM Personale;");
    result.next(); // posizionamento primo risultato
    personale = result.getInt(1);
    System.out.println("Unità personale: " + personale);
    if (stat.executeUpdate("DELETE FROM Personale
                           WHERE stipendio > 3000;") > 0) {
        System.out.println("Eliminato personale con stipendio maggiore di 3000€.");
    }
    result = stat.executeQuery("SELECT COUNT(*) AS numero FROM Personale;");
    result.next(); // posizionamento primo risultato
    personale = result.getInt(1);
    System.out.println("Unità personale: " + personale);
    result.close();
    stat.close();
    con.close();
    System.out.println("Effettuata disconnessione dal server DBMS.");
}
catch (SQLException exception) {
    System.out.println("Errore!");
}
}

```

Con i dati del database di esempio viene prodotto il seguente output:

```

Effettuata connessione al server DBMS.
Unità personale: 14
Eliminato personale con stipendio maggiore di 3000€.
Unità personale: 12
Effettuata disconnessione dal server DBMS.

```

### 3 Classi CRUD in linguaggio Java; corrispondenza tra tipi SQL e tipi Java

Le operazioni che normalmente si effettuano su una base di dati sono riepilogate dall'acronimo CRUD:

- **Create**: creazione di nuovi record;
- **Read**: ricerca e lettura di record;
- **Update**: aggiornamento di record esistenti;
- **Delete**: eliminazione di record.

## Java Data Objects

*Java Data Objects* (JDO) è una API per la gestione della persistenza degli oggetti in un database relazionale. JDO consente di memorizzare in modo permanente e trasparente, per successivamente ripristinare, oggetti istanze di classi definite in linguaggio Java. JDO realizza un vero proprio *Object-Relational Mapping* (ORM) tra gli oggetti Java tradizionali (POJO: *Plain Old Java Object*) e i database relazionali.

Ogni operazione CRUD è chiaramente associata a uno specifico comando del linguaggio SQL come riportato nella TABELLA 2:

TABELLA 2

Create	INSERT
Read	SELECT
Update	UPDATE
Delete	DELETE

La realizzazione di una classe CRUD in linguaggio Java prevede – oltre alla definizione di specifiche classi per la rappresentazione dei record di dati – lo sviluppo di una classe di gestione dei dati i cui metodi incapsulino l'accesso al database relazionale che rende persistenti i dati stessi.

### ESEMPIO

La seguente classe rappresenta i record della tabella *Personale* del database *Azienda*:



```
import java.sql.*;

public class Personale {
    private String matricola;
    private String dipartimento;
    private String nominativo;
    private String qualifica;
    private Date dataNascita;
    private double stipendio;

    public Personale(String matricola, String dipartimento,
                     String nominativo, String qualifica,
                     Date dataNascita, double stipendio) {
        this.matricola = matricola;
        this.dipartimento = dipartimento;
        this.nominativo = nominativo;
        this.qualifica = qualifica;
        this.dataNascita = dataNascita;
        this.stipendio = stipendio;
    }

    public Personale() {
        this.matricola = "";
        this.dipartimento = "";
        this.nominativo = "";
        this.qualifica = "";
        this.dataNascita = new Date(0);
        this.stipendio = 0.0;
    }

    public Personale(Personale personale) {
        this.matricola = personale.getMatricola();
        this.dipartimento = personale.getDipartimento();
        this.nominativo = personale.getNominativo();
    }
}
```



```

        this.qualifica = personale.getQualifica();
        this.dataNascita = personale.getDataNascita();
        this.stipendio = personale.getStipendio();
    }

    public String getMatricola() {
        return matricola;
    }

    public void setMatricola(String matricola) {
        this.matricola = matricola;
    }

    public String getDipartimento() {
        return dipartimento;
    }

    public void setDipartimento(String dipartimento) {
        this.dipartimento = dipartimento;
    }

    public String getNominativo() {
        return nominativo;
    }

    public void setNominativo(String nominativo) {
        this.nominativo = nominativo;
    }

    public String getQualifica() {
        return qualifica;
    }

    public void setQualifica(String qualifica) {
        this.qualifica = qualifica;
    }

    public Date getDataNascita() {
        return dataNascita;
    }

    public void setDataNascita(Date dataNascita) {
        this.dataNascita = dataNascita;
    }

    public double getStipendio() {
        return stipendio;
    }

    public void setStipendio(double stipendio) {
        this.stipendio = stipendio;
    }

    public String toString() {
        return "Personale(" + matricola + "," + dipartimento + "," +
            nominativo + "," + qualifica + "," + dataNascita +
            "," + stipendio + ')';
    }
}

```

La seguente classe CRUD consente di gestire i record della tabella *Personale* del database *Azienda*:



```
import java.sql.*;

public class GestionePersonale {
    final String URL = "jdbc:mysql://localhost:3306";
    final String database = "Azienda";
    final String user = "root";
    final String password = "";
    private Connection con;

    public GestionePersonale() throws SQLException {
        con = DriverManager.getConnection(URL+"/"+database, user, password);
    }

    /* Inserimento unità di personale nel database */
    public boolean aggiungiPersonale(Personale personale) {
        Statement stat;
        ResultSet result;
        String id_dipartimento;
        String data_nascita;

        try { // ricerca identificatore dipartimento a partire dal nome
            stat = con.createStatement();
            String query = "SELECT id_dipartimento FROM Dipartimento
                           WHERE nome_dipartimento = '"
                           + personale.getDipartimento() + "'";
            result = stat.executeQuery(query);
            result.next();
            id_dipartimento = result.getString(1);
            result.close();
            stat.close();
        }
        catch (SQLException exception) {
            return false;
        }

        try {
            stat = con.createStatement();
            data_nascita = personale.getDataNascita().toString();
            String command = "INSERT INTO Personale(matricola,
                           id_dipartimento, nominativo, data_nascita,
                           qualifica, stipendio)
                           VALUES ('" + personale.getMatricola() + "', '" +
                           id_dipartimento + "', '" +
                           personale.getNominativo() + "', '" +
                           data_nascita + "', '" +
                           personale.getQualifica() + "', " +
                           personale.getStipendio() + ");";

            if (stat.executeUpdate(command) == 0) {
                return false;
            }
            stat.close();
        }
    }
}
```



```

        catch (SQLException exception) {
            return false;
        }

        return true;
    }

    /* Aggiornamento dati unità di personale nel database */
    public boolean aggiornaPersonale(Personale personale) {
        Statement stat;
        ResultSet result;
        String id_dipartimento;
        String data_nascita;

        try { // verifica presenza nel database unità di personale
            stat = con.createStatement();
            String query = "SELECT COUNT(*) AS numero FROM Personale WHERE
                           matricola = '" + personale.getMatricola() + "';";
            result = stat.executeQuery(query);
            result.next();
            if (result.getInt(1) == 0) {
                return false;
            }
            result.close();
            stat.close();
        }
        catch (SQLException exception) {
            return false;
        }

        try { // ricerca identificatore dipartimento a partire dal nome
            stat = con.createStatement();
            String query = "SELECT id_dipartimento FROM Dipartimento WHERE
                           nome_dipartimento = '" +
                           personale.getDipartimento() + "';";
            result = stat.executeQuery(query);
            result.next();
            id_dipartimento = result.getString(1);
            result.close();
            stat.close();
        }
        catch (SQLException exception) {
            return false;
        }

        try {
            stat = con.createStatement();
            data_nascita = personale.getDataNascita().toString();
            String command = "UPDATE Personale SET id_dipartimento='" +
                             id_dipartimento + "', nominativo='" +
                             personale.getNominativo() +
                             "', data_nascita= '" + data_nascita +

```





```

        "', qualifica='" + personale.getQualifica() +
        "', stipendio='" + personale.getStipendio() +
        " WHERE matricola='" + personale.getMatricola()
        + "';";

    if (stat.executeUpdate(command) == 0) {
        return false;
    }
    stat.close();
}

catch (SQLException exception) {
    return false;
}

return true;
}

/* Ricerca nel database i dati di una unità di personale */
public Personale datiPersonale(String matricola) {
    Personale personale;
    Statement stat;
    ResultSet result;
    String nominativo;
    Date data_nascita;
    String nome_dipartimento;
    String qualifica;
    double stipendio;

    try {
        stat = con.createStatement();
        String query = "SELECT * FROM Personale, Dipartimento WHERE
                        Personale.id_dipartimento =
                        Dipartimento.id_dipartimento AND matricola = '" +
                        matricola + "';";
        result = stat.executeQuery(query);
        result.next();
        nominativo = result.getString("nominativo");
        data_nascita = result.getDate("data_nascita");
        nome_dipartimento = result.getString("nome_dipartimento");
        qualifica = result.getString("qualifica");
        stipendio = result.getDouble("stipendio");
        personale = new Personale(matricola, nome_dipartimento,
                                   nominativo, qualifica, data_nascita,
                                   stipendio);

        result.close();
        stat.close();
        return personale;
    }
    catch (SQLException exception) {
        return null;
    }
}

```



```

/* Ricerca nel database i dati di tutte le unità di personale */
public Personale[] elencoPersonale() {
    Personale elenco_personale[];
    Statement stat;
    ResultSet result;
    int numero_personale;
    String matricola;
    String nominativo;
    Date data_nascita;
    String nome_dipartimento;
    String qualifica;
    double stipendio;

    try { // conteggio unità di personale presenti nel database
        stat = con.createStatement();
        String query = "SELECT COUNT(*) AS numero FROM Personale;";
        result = stat.executeQuery(query);
        result.next();
        numero_personale = result.getInt(1);
        if (numero_personale > 0) {
            elenco_personale = new Personale[numero_personale];
        }
        else {
            return null;
        }
        result.close();
        stat.close();
    }
    catch (SQLException exception) {
        return null;
    }

    try {
        stat = con.createStatement();
        String query = "SELECT * FROM Personale, Dipartimento WHERE
                        Personale.id_dipartimento =
                        Dipartimento.id_dipartimento;";
        result = stat.executeQuery(query);
        numero_personale = 0;
        while (result.next()) {
            matricola = result.getString("matricola");
            nominativo = result.getString("nominativo");
            data_nascita = result.getDate("data_nascita");
            nome_dipartimento = result.getString("nome_dipartimento");
            qualifica = result.getString("qualifica");
            stipendio = result.getDouble("stipendio");
            elenco_personale[numero_personale] =
                new Personale(matricola, nome_dipartimento, nominativo,
                              qualifica, data_nascita, stipendio);
            numero_personale++;
        }
    }
}

```

```

        result.close();
        stat.close();
        return elenco_personale;
    }
    catch (SQLException exception) {
        return null;
    }
}

/* Eliminazione unità di personale dal database */
public boolean eliminaPersonale(String matricola) {
    Statement stat;
    ResultSet result;

    try {
        stat = con.createStatement();
        String command = "DELETE FROM Personale WHERE matricola = '" +
                        matricola + "';";
        if (stat.executeUpdate(command) == 0) {
            return false;
        }
        stat.close();
    }
    catch (SQLException exception) {
        return false;
    }
    return true;
}
}

```

## OSSERVAZIONE



In un contesto reale nel codice del metodo *elencoPersonale* della classe dell'esempio precedente potrebbe essere opportuno introdurre un limite alla dimensione massima che l'array restituito può assumere.

Per verificare i metodi della classe CRUD dell'esempio precedente è possibile definire il seguente metodo *main*:

```

public static void main(String args[]) {
    Date data_nascita = new Date(65, 2, 23); // 23-3-1965
    Personale personale = new Personale("01234", "ALFA", "MEINI GIORGIO",
                                         "09", data_nascita, 9999.99);

    GestionePersonale gestione_personale;
    Personale elenco_personale[];

    try {
        gestione_personale = new GestionePersonale();
    }
}

```



```

catch (SQLException exception) {
    System.out.println("Errore connessione server DBMS.");
    return;
}

if (gestione_personale.aggiungiPersonale(personale)) {
    System.out.println("Unità di personale aggiunta al database.");
}
else {
    System.out.println("Errore aggiunta unità di personale.");
}

elenco_personale = gestione_personale.elencoPersonale();
if (elenco_personale != null) {
    System.out.println("Elenco personale: ");
    for (int i=0; i<elenco_personale.length; i++) {
        System.out.println(elenco_personale[i]);
    }
}
else {
    System.out.println("Errore ricerca unità di personale.");
}

personale.setQualifica("00");
personale.setStipendio(999.99);
if (gestione_personale.aggiornaPersonale(personale)) {
    System.out.println("Unità di personale del database aggiornata.");
}
else {
    System.out.println("Errore aggiornamento unità di personale.");
}

personale = gestione_personale.datiPersonale("01234");
if (personale != null) {
    System.out.println("Unità di personale: " + personale);
}
else {
    System.out.println("Errore ricerca unità di personale.");
}

if (gestione_personale.eliminaPersonale("01234")) {
    System.out.println("Unità di personale eliminata dal database.");
}
else {
    System.out.println("Errore eliminazione unità di personale.");
}
}

```

### 3.1 Corrispondenza tra tipi SQL e tipi Java

La scelta del metodo corretto da impiegare per acquisire i valori dei singoli campi da un oggetto che implementa l'interfaccia *ResultSet* richiede di stabilire una corrispondenza tra i tipi dei campi delle tabelle di un database e i tipi delle variabili utilizzate.

Nella TABELLA 3 sono riportate le corrispondenze tra i tipi di dati Java e i più comuni tipi di dati del linguaggio SQL.

TABELLA 3

SQL	Java
CHAR	String
VARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
INTEGER	int
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

#### CHAR e VARCHAR

I tipi del linguaggio SQL **CHAR** e **VARCHAR** sono strettamente collegati tra loro: **CHAR** rappresenta una stringa di caratteri di lunghezza fissa, mentre **VARCHAR** rappresenta una stringa di caratteri a lunghezza variabile. Nel linguaggio Java non è possibile effettuare questa distinzione: entrambi i tipi possono essere rappresentati in Java come `String`, o come `char[]`, anche se la prima soluzione è in generale la più appropriata. Il metodo `getString` definito dall'interfaccia *ResultSet* restituisce un oggetto di tipo `String` ed è adatto per dati sia di tipo **CHAR** che **VARCHAR**.

#### NUMERIC e DECIMAL

I tipi **NUMERIC** e **DECIMAL** del linguaggio SQL sono molto simili: entrambi rappresentano numeri in virgola fissa e la rappresentazione più conveniente in linguaggio Java per tali tipi è la classe *BigDecimal* del package *java.math*. Essa infatti permette operazioni matematiche sul tipo *BigDecimal* anche in combinazione con tipi interi e in virgola mobile. Oltre che con il metodo `getBigDecimal` dell'interfaccia *ResultSet* è possibile accedere a questi tipi di valori invocando il metodo `getString`.

## BINARY e VARBINARY

Questi tipi di dati SQL sono strettamente correlati: **BINARY** rappresenta dati binari di lunghezza fissa, mentre **VARBINARY** rappresenta dati binari di lunghezza variabile. Nel linguaggio Java non si ha la necessità di distinguere tra questi tipi perché essi possono essere rappresentati tramite array di byte in ambedue i casi: il metodo *getBytes* dell'interfaccia *ResultSet* viene usato per recuperare valori di tipo **BINARY** e **VARBINARY**.

## BIT

Il tipo SQL **BIT** rappresenta un singolo bit che può assumere i valori 0 o 1; può essere rappresentato mediante il tipo **boolean** del linguaggio Java restituito dal metodo *getBoolean* dell'interfaccia *ResultSet*.

## INTEGER

Il tipo SQL **INTEGER** e le varianti My-SQL **TINYINT**, **SMALLINT**, **INT** e **BIGINT** rappresentano valori numerici interi che trovano una corrispondenza nei tipi Java **byte**, **short**, **int** e **long** restituiti rispettivamente dai metodi *getBytes*, *getShort*, *getInt* e *getLong* dell'interfaccia *ResultSet*.

## REAL, FLOAT e DOUBLE

Il tipo SQL **REAL** rappresenta numeri in virgola mobile in singola precisione, mentre i tipi **FLOAT** e **DOUBLE** rappresentano numeri in virgola mobile in doppia precisione. È di conseguenza raccomandabile utilizzare per il tipo **REAL** il tipo Java **float** restituito dal metodo *getFloat* dell'interfaccia *ResultSet*, e per i tipi **FLOAT** e **DOUBLE** il tipo Java **double** restituito dal metodo *getDouble* dell'interfaccia *ResultSet*.

## DATE, TIME e TIMESTAMP

Questi tre tipi di dati del linguaggio SQL fanno riferimento a valori temporali: il tipo **DATE** rappresenta date con i valori di giorno, mese e anno, il tipo **TIME** rappresenta un orario con valori di ore, minuti e secondi e il tipo **TIMESTAMP** rappresenta valori in cui sono compresi una data e un orario integrato con il valore dei millisecondi. La classe standard *Date* del package *java.util* permette di gestire dati di tipo data e/o orario, ma non è in corrispondenza diretta con nessuno dei tre tipi SQL per le date e gli orari; per questa ragione nel package *java.sql* sono definite tre sottoclassi di *Date*:

- *Date* per dati SQL di tipo **DATE**;
- *Time* per dati SQL di tipo **TIME**;
- *Timestamp* per dati SQL di tipo **TIMESTAMP**.

# 4 Uso di oggetti RowSet

*RowSet* è un'interfaccia definita nel package *javax.sql* che deriva dall'interfaccia *ResultSet*, ma che espone la tipica interfaccia di un oggetto *JavaBean*: un oggetto che implementa l'interfaccia *RowSet* ha lo scopo di mantenere

### JavaBean

I *JavaBean* sono componenti software realizzati in linguaggio Java seguendo alcune convenzioni:

- hanno un costruttore privo di argomenti;
- il valore degli attributi, definiti «proprietà», viene impostato e acquisito mediante metodi di tipo *get/set*;
- è possibile registrare uno o più *listener* di eventi che il componente genera.

una corrispondenza tra i dati che contiene nella forma di una sequenza di record e la loro sorgente, tipicamente un database relazionale al quale si connette utilizzando JDBC.

Alcuni vantaggi del ricorso a *RowSet* rispetto a *ResultSet* consistono nel fatto che *RowSet* è sempre «navigabile» in entrambe le direzioni e che l'aggiornamento dei dati che un oggetto *RowSet* contiene comporta l'aggiornamento della sorgente dei dati (nel caso di un database relazionale dei record di una o più tabelle).

L'impostazione di alcune proprietà previste dall'interfaccia *RowSet* condizionano il funzionamento degli oggetti che la implementano (TABELLA 4).

TABELLA 4

concurrency	L'accesso concorrente ai dati può essere di tipo CONCUR_READ_ONLY (default, solo lettura dei dati), oppure CONCUR_UPDATABLE (permesso aggiornamento concorrente dei dati)
maxRows	Massimo numero di record gestiti
password	Password per l'accesso al database
queryTimeout	Tempo massimo di attesa per l'esecuzione di un comando (secondi)
readOnly	Accesso ai dati senza possibilità di modifica
type	Lo scorrimento dei dati può essere di tipo TYPE_FORWARD_ONLY (unidirezionale), TYPE_SCROLL_INSENSITIVE (bidirezionale non sensibile al cambiamento della sorgente di dati), o TYPE_SCROLL_SENSITIVE (bidirezionale, sensibile al cambiamento della sorgente di dati)
url	URL per la connessione JDBC al database
username	Username per l'accesso al database

**OSSERVAZIONE** Dato che gli oggetti *RowSet* sono componenti *JavaBeans*, le proprietà sono impostate e acquisite invocando metodi *set/get* specifici.

L'interfaccia *RowSet* prevede due metodi fondamentali per il funzionamento degli oggetti che la implementano:

- **setCommand**: per impostare la *query* SQL che definisce i dati che l'oggetto contiene;
- **execute**: per eseguire la *query* SQL che carica i dati come contenuto dell'oggetto.

I seguenti metodi che l'interfaccia *RowSet* eredita dall'interfaccia *ResultSet* consentono di impostare il valore dei campi del record corrente senza aggiornare i corrispondenti valori nel database, cosa che può essere fatta successivamente invocando il metodo *updateRow*:

- *updateBigDecimal*
- *updateBoolean*
- *updateByte*

- *updateBytes*
- *updateDate*
- *updateDouble*
- *updateFloat*
- *updateInt*
- *updateLong*
- *updateNull*
- *updateShort*
- *updateString*
- *updateTime*
- *updateTimestamp*

**OSSERVAZIONE** Per l'acquisizione del valore dei campi del record corrente sono disponibili i seguenti metodi ereditati dall'interfaccia *ResultSet*:

- *getBigDecimal*
- *getBoolean*
- *getByte*
- *getBytes*
- *getDate*
- *getDouble*
- *getFloat*
- *getInt*
- *getLong*
- *getNull*
- *getShort*
- *getString*
- *getTime*
- *getTimestamp*

L'interfaccia *RowSet* eredita dall'interfaccia *ResultSet* i seguenti metodi per lo scorrimento della sequenza di record che contiene:

- ***afterLast***: posiziona il cursore oltre l'ultimo record della sequenza (come quando termina lo scorrimento della sequenza);
- ***beforeFirst***: posiziona il cursore prima del primo record della sequenza (come quando lo scorrimento della sequenza non è ancora iniziato);
- ***first***: posiziona il cursore sul primo record della sequenza;
- ***last***: posiziona il cursore sull'ultimo record della sequenza;
- ***next***: posiziona il cursore sul prossimo record della sequenza;
- ***previous***: posiziona il cursore sul precedente record della sequenza.

L'eliminazione di un record da un oggetto che implementa l'interfaccia *RowSet* e contemporaneamente dei dati corrispondenti nel database avviene invocando il metodo *deleteRow*.

L'inserimento di un nuovo record in un oggetto che implementa l'interfaccia *RowSet* e contemporaneamente nella/e tabella/e corrispondenti



## Oggetti *RowSet* connessi e disconnessi

Un oggetto *RowSet* può essere di tipo connesso o disconnesso. È *connesso* se usa un driver basato sulla tecnologia JDBC per mantenere una connessione permanente a un database. È *disconnesso* se effettua una connessione intermittente a una sorgente di dati per leggere o scrivere i dati.

Non dovendo mantenere una connessione permanente alla propria sorgente di dati, gli oggetti *RowSet* disconnessi sono molto più «leggeri» rispetto a quelli connessi: questa caratteristica li rende ideali per la gestione di dati in rete, in particolare utilizzando dispositivi mobili.

nel database avviene invocando il metodo *insertRow* dopo il posizionamento sul record speciale di inserimento mediante invocazione del metodo *moveToInsertRow*.

Le classi che implementano l'interfaccia *RowSet* normalmente ne implementano una delle seguenti specializzazioni:

- ***JdbcRowSet***: prevede la connessione permanente a un database mediante un driver JDBC;
- ***CachedRowSet***: prevede che la connessione alla propria sorgente di dati possa avvenire mediante sincronizzazioni discontinue nel tempo;
- ***WebRowSet***: prevede lo stesso comportamento di *CachedRowSet*, oltre a specifiche funzionalità per l'importazione e l'esportazione in formato XML dei dati che contiene;
- ***JoinRowSet***: prevede lo stesso comportamento di *WebRowSet*, oltre alla possibilità di effettuare *join* senza la necessità di connettersi alla sorgente dei dati;
- ***FilteredRowSet***: prevede lo stesso comportamento di *WebRowSet*, oltre alla possibilità di filtrare i dati che risultano visibili senza la necessità di connettersi alla sorgente dei dati.

**OSSERVAZIONE** Nel package *javax.sql* non sono definite classi che implementano l'interfaccia *RowSet*, ma è in ogni caso possibile utilizzare una delle implementazioni standard di riferimento contenute nel package *com.sun.rowset*, come *JdbcRowSetImpl* che implementa l'interfaccia *JdbcRowSet*.

Il modo più semplice per creare un oggetto di classe *JdbcRowSetImpl* è quello di invocare il costruttore fornendo come argomento un oggetto di tipo *Connection* già istanziato.

### ESEMPIO

La seguente classe Java rivisita quella dell'esempio precedente relativo agli oggetti *ResultSet* allo scopo di dimostrare l'uso di oggetti *RowSet*:

```
import java.sql.*;
import javax.sql.*;
import javax.sql.rowset.*;
import com.sun.rowset.*;

public class GestionePersonale {
    final String URL = "jdbc:mysql://localhost:3306";
    final String database = "Azienda";
    final String user = "root";
    final String password = "";
    private Connection con;
    private JdbcRowSet rowset;

    public GestionePersonale() throws SQLException {
        con = DriverManager.getConnection(URL+"/"+database, user, password);
    }
}
```



```

rowset = new JdbcRowSetImpl(con);
rowset.setCommand("SELECT * FROM Personale;");
rowset.execute();
}

/* Inserimento unità di personale nel database */
public boolean aggiungiPersonale(Personale personale) {
    Statement stat;
    ResultSet result;
    String id_dipartimento;

    try { // ricerca identificatore dipartimento a partire dal nome
        stat = con.createStatement();
        String query = "SELECT id_dipartimento FROM Dipartimento
                        WHERE nome_dipartimento = '"
                        + personale.getDipartimento() + "'";
        result = stat.executeQuery(query);
        result.next();
        id_dipartimento = result.getString(1);
        result.close();
        stat.close();
    }
    catch (SQLException exception) {
        return false;
    }

    try {
        rowset.moveToInsertRow();
        rowset.updateString("matricola", personale.getMatricola());
        rowset.updateString("id_dipartimento", id_dipartimento);
        rowset.updateString("nominativo", personale.getNominativo());
        rowset.updateDate("data_nascita", personale.getDataNascita());
        rowset.updateString("qualifica", personale.getQualifica());
        rowset.updateDouble("stipendio", personale.getStipendio());
        rowset.insertRow();
        return true;
    }
    catch (SQLException exception) {
        return false;
    }
}

/* Aggiornamento dati unità di personale nel database */
public boolean aggiornaPersonale(Personale personale) {
    Statement stat;
    ResultSet result;
    String id_dipartimento;
    String data_nascita;

    try { // ricerca identificatore dipartimento a partire dal nome
        stat = con.createStatement();
        String query = "SELECT id_dipartimento FROM Dipartimento WHERE
                        nome_dipartimento = '" + personale.getDipartimento() + "'";
        result = stat.executeQuery(query);
        result.next();
        id_dipartimento = result.getString(1);
        result.close();
        stat.close();
    }
    catch (SQLException exception) {
        return false;
    }

    try {
        rowset.moveToUpdateRow();
        rowset.updateString("matricola", personale.getMatricola());
        rowset.updateString("id_dipartimento", id_dipartimento);
        rowset.updateString("nominativo", personale.getNominativo());
        rowset.updateDate("data_nascita", personale.getDataNascita());
        rowset.updateString("qualifica", personale.getQualifica());
        rowset.updateDouble("stipendio", personale.getStipendio());
        rowset.updateRow();
        return true;
    }
    catch (SQLException exception) {
        return false;
    }
}

```

```

        result = stat.executeQuery(query);
        result.next();
        id_dipartimento = result.getString(1);
        result.close();
        stat.close();
    }
    catch (SQLException exception) {
        return false;
    }

    try {
        rowset.beforeFirst();
        while (rowset.next()) { // ricerca record
            if (rowset.getString("matricola").equals
                (personale.getMatricola())) {
                rowset.updateString("id_dipartimento", id_dipartimento);
                rowset.updateString("nominativo", personale.getNominativo());
                rowset.updateDate("data_nascita", personale.getDataNascita());
                rowset.updateString("qualifica", personale.getQualifica());
                rowset.updateDouble("stipendio", personale.getStipendio());
                rowset.updateRow();
                return true;
            }
        }
        return false;
    }
    catch (SQLException exception) {
        return false;
    }
}

/* Ricerca nel database i dati di una unità di personale */
public Personale datiPersonale(String matricola) {
    Personale personale;
    Statement stat;
    ResultSet result;
    String nominativo;
    Date data_nascita;
    String id_dipartimento;
    String nome_dipartimento;
    String qualifica;
    double stipendio;

    try {
        rowset.beforeFirst();
        while (rowset.next()) { // ricerca record
            if (rowset.getString("matricola").equals(matricola)) {
                id_dipartimento = rowset.getString("id_dipartimento");
                nominativo = rowset.getString("nominativo");
                data_nascita = rowset.getDate("data_nascita");
                qualifica = rowset.getString("qualifica");
                stipendio = rowset.getDouble("stipendio");
            }
        }
    }
    catch (SQLException exception) {
        return null;
    }
}

```



```

        stat = con.createStatement();
        String query = "SELECT nome_dipartimento FROM Dipartimento WHERE
                        id_dipartimento = '" + id_dipartimento + "'";
        result = stat.executeQuery(query);
        result.next();
        nome_dipartimento = result.getString(1);
        result.close();
        stat.close();
        personale = new Personale(matricola, nome_dipartimento,
                                   nominativo, qualifica, data_nascita,
                                   stipendio);

        return personale;
    }
}
return null;
}
catch (SQLException exception) {
    return null;
}
}

/* Ricerca nel database i dati di tutte le unità di personale */
public Personale[] elencoPersonale() {
    Personale elenco_personale[];
    Statement stat;
    ResultSet result;
    int numero_personale;
    String matricola;
    String nominativo;
    Date data_nascita;
    String id_dipartimento;
    String nome_dipartimento;
    String qualifica;
    double stipendio;

    try { // conteggio record
        rowset.beforeFirst();
        numero_personale = 0;
        while (rowset.next()) {
            numero_personale++;
        }
        if (numero_personale > 0) {
            elenco_personale = new Personale[numero_personale];
        }
        else {
            return null;
        }
    }
    catch (SQLException exception) {
        return null;
    }
}

```



```

try {
    rowset.beforeFirst();
    numero_personale = 0;
    while (rowset.next()) {
        matricola = rowset.getString("matricola");
        id_dipartimento = rowset.getString("id_dipartimento");
        nominativo = rowset.getString("nominativo");
        data_nascita = rowset.getDate("data_nascita");
        qualifica = rowset.getString("qualifica");
        stipendio = rowset.getDouble("stipendio");
        stat = con.createStatement();
        String query = "SELECT nome_dipartimento FROM Dipartimento WHERE
                        id_dipartimento = '" + id_dipartimento + "'";
        result = stat.executeQuery(query);
        result.next();
        nome_dipartimento = result.getString(1);
        result.close();
        stat.close();
        elenco_personale[numero_personale] = new Personale(matricola,
                                                            nome_dipartimento,
                                                            nominativo, qualifica,
                                                            data_nascita, stipendio);

        numero_personale++;
    }
    return elenco_personale;
}
catch (SQLException exception) {
    return null;
}
}

/* Eliminazione unità di personale dal database */
public boolean eliminaPersonale(String matricola) {
    Statement stat;
    ResultSet result;

    try {
        rowset.beforeFirst();
        while (rowset.next()) { // ricerca record
            if (rowset.getString("matricola").equals(matricola)) {
                rowset.deleteRow();
                return true;
            }
        }
        return false;
    }
    catch (SQLException exception) {
        return false;
    }
}
}

```

Non essendo cambiata l'interfaccia della classe *GestionePersonale* il *main* per il test della classe non subisce variazioni rispetto al metodo *main* della classe originale.

**OSSERVAZIONE** Il codice dell'esempio precedente – oltre all'oggetto *JdbcRowSet* che comprende i dati della tabella *Personale* del database *Azienda* – utilizza oggetti *ResultSet* temporanei per risolvere l'accesso alla tabella *Dipartimento*.

La mancanza di metodi che permettano di selezionare i singoli record di un oggetto *JdbcRowSet* costringe a effettuare la ricerca dei record di interesse iterando i record contenuti nell'oggetto.

## Eventi e GUI

Gli eventi generati da un oggetto di classe *RowSet* intercettati da uno specifico *RowSetListener* registrato da un componente della GUI di un'applicazione Java (ad esempio un componente del framework *Swing*) permettono di aggiornare la visualizzazione dei dati in modo sincrono con il loro cambiamento anche nel caso che la variazione sia determinata da operazioni estranee al componente stesso.

### 4.1 Notifica degli eventi

L'interfaccia *RowSet* prevede l'implementazione del modello a eventi dei componenti *JavaBean*. Gli oggetti *RowSet* notificano a un eventuale *listener* tre diversi tipi di eventi:

- movimento del cursore dei record (metodo *cursorMoved*);
- aggiornamento, inserimento e cancellazione di un record (metodo *rowChanged*);
- modifica del contenuto dell'oggetto (metodo *rowSetChanged*).

Gli oggetti *listener* che ricevono gli eventi generati da oggetti *RowSet* devono implementare l'interfaccia *RowSetListener* ed essere registrati per ogni specifico oggetto ai cui eventi sono interessati.

#### ESEMPIO

La seguente classe realizza a solo titolo di esempio un visualizzatore degli eventi generati da un oggetto *JdbcRowSet* nel corso del suo funzionamento:

```
import java.sql.*;
import javax.sql.*;
import javax.sql.rowset.*;
import com.sun.rowset.*;

class TestListener implements RowSetListener {
    public void cursorMoved(RowSetEvent event) {
        System.out.println("Spostamento cursore.");
    }

    public void rowChanged(RowSetEvent event) {
        System.out.println("Cambiamento record.");
    }

    public void rowSetChanged(RowSetEvent event) {
        System.out.println("Modifica RowSet.");
    }
}
```

La registrazione del *listener* avviene aggiungendo questa riga nel costruttore della classe *GestionePersonale*:

```
rowset.addRowSetListener(new TestListener());
```

L'esecuzione del metodo *main* della classe produce in output le stringhe di testo «Spostamento cursore» e «Cambiamento record» in corrispondenza dell'esecuzione delle istruzioni di iterazione sui dati e di modifica (inserimento, aggiornamento o cancellazione) dei record.

## 5 Gestione delle transazioni

Al momento della creazione di una connessione JDBC, questa si trova in uno stato di *auto-commit*: ogni singolo comando SQL viene gestito come una transazione per la quale viene eseguito un *commit* automatico dopo la sua esecuzione. Avendo la necessità che un insieme di comandi siano eseguiti in modo atomico è necessario disabilitare la modalità *auto-commit*:

```
Connection con = DriverManager.getConnection(...);
con.setAutoCommit(false);
```

Una volta disabilitato l'*auto-commit*, l'esecuzione dei comandi SQL non modifica il database fino a quando non viene eseguito esplicitamente il metodo *commit*:

```
con.commit();
```

In questo modo, se si verifica un qualsiasi errore, è possibile annullare tutte le modifiche apportate, ma non ancora confermate, eseguendo un *rollback*:

```
con.rollback();
```

**OSSERVAZIONE** Normalmente l'invocazione del metodo *commit* avviene dopo l'esecuzione dell'ultimo comando SQL che costituisce la transazione, mentre l'invocazione del metodo *rollback* avviene come parte del codice di gestione di una eventuale eccezione:

```
try {
    ...
    ...
    ...
    con.commit();
}
catch (SQLException exception) {
    con.rollback();
}
```

### ESEMPIO

Il seguente metodo della classe *GestionePersonale* applica a tutto il personale di una specifica qualifica un aumento percentuale limitando il nuovo stipendio a un valore massimo:

```
public void aumentoStipendio(String qualifica, double percentuale,
                             double massimo) throws SQLException {
    con.setAutoCommit(false);

    try {
        PreparedStatement aumenta = con.prepareStatement("UPDATE Personale SET stipendio =
                                                         (stipendio + stipendio*%/100) WHERE qualifica = ?;"); ▶
```

```

    aumenta.setDouble(1, percentuale);
    aumenta.setString(2, qualifica);
    aumenta.executeUpdate();
    PreparedStatement limita = con.prepareStatement("UPDATE Personale
                                                SET stipendio=? WHERE qualifica = ?
                                                AND stipendio > ?;");

    limita.setDouble(1, massimo);
    limita.setString(2, qualifica);
    limita.setDouble(3, massimo);
    limita.executeUpdate();
    con.commit();
}
catch(SQLException exception){
    con.rollback();
}
con.setAutoCommit(true);
}

```

L'interfaccia *JdbcRowSet* prevede, per la gestione delle transazioni, i metodi indicati nella TABELLA 5.

TABELLA 5

setAutoCommit	Abilita/disabilita la modalità <i>auto-commit</i>
commit	Termina ed esegue il <i>commit</i> della transazione
rollback	Termina e annulla la transazione

## Sintesi

■ **Accesso a database indipendente dal server DBMS utilizzato.** Effettuare un accesso al database indipendentemente dallo specifico server DBMS utilizzato rappresenta il problema principale nell'implementazione di applicazioni distribuite in rete la cui soluzione richiede funzionalità di adattamento e di conversione che permettano lo scambio di informazione tra sistemi e applicazioni eterogenei. Per far fronte a queste problematiche nello sviluppo di applicazioni software si ricorre a tecnologie standard implementate in forma di API (*Application Program Interface*).

■ **ODBC (*Open DataBase Connectivity*).** Tecnologia sviluppata da Microsoft negli anni '90 del secolo scorso con l'intento di definire un'interfaccia standard per l'accesso da codice in lin-

guaggio C/C++ a database in modo indipendente sia dal sistema operativo che dal server DBMS.

■ **JDBC (*Java DataBase Connectivity*).** Tecnologia specificatamente sviluppata come componente standard del *Java Development Kit* a partire dal 1997; permette l'accesso a un database in modo indipendente dal DBMS e dalla piattaforma di esecuzione.

■ **Accesso a database da parte di un'applicazione.** La tipica sequenza di interazione tra un'applicazione e un server DBMS per l'accesso a una base di dati è la seguente:

- connessione alla sorgente dei dati (database contenuto in un DBMS);
- esecuzione di comandi/*query* SQL;



- recupero dei risultati ed eventuale gestione degli errori;
- disconnessione dalla sorgente dei dati.

■ **Driver Manager.** È un componente software che gestisce la comunicazione tra l'applicazione e i driver degli specifici server DBMS. Esso seleziona uno specifico driver per l'interazione con uno specifico DBMS e ne gestisce l'invocazione delle funzioni che esso prevede.

■ **Driver.** Sono generalmente librerie caricate dinamicamente che implementano le funzioni API; ne esiste una specifica per ogni particolare DBMS col compito di:

- nascondere le differenze di interazione dovute ai vari DBMS, sistemi operativi e protocolli di rete impiegati;
- trasformare le invocazioni delle funzioni API nel «dialetto» SQL usato dal server DBMS, o nelle corrispondenti funzioni API supportate dal server DBMS;
- gestire le transazioni, eseguire i comandi e le *query* SQL, inviare e recuperare i dati, gestire gli errori ecc.

■ **JDBC-ODBC bridge.** Driver JDBC che prevede l'accesso al server DBMS tramite un driver ODBC.

■ **Driver JDBC realizzato solo in parte in Java.** Le chiamate inoltrate a JDBC sono convertite in chiamate alla API specifica del server DBMS, di conseguenza il driver contiene codice Java che invoca funzioni normalmente codificate in linguaggio C/C++.

■ **Driver JDBC completamente realizzato in Java con DBMS middleware.** Le chiamate inoltrate a JDBC sono elaborate da un driver locale completamente codificato in Java che le trasforma utilizzando un protocollo indipendente da quello del server DBMS in invocazioni per un'applicazione *middleware*, che a sua volta si interfaccia con lo specifico DBMS.

■ **Driver JDBC pure Java con connessione diretta al server DBMS.** Il driver JDBC è in questo caso completamente realizzato in Java e – essendo specifico per il DBMS utilizzato – converte le chiamate JDBC direttamente nel protocollo di rete usato dal server DBMS.

■ **Uso di API JDBC per l'interazione con un database gestito da uno specifico DBMS.** La tipica sequenza di fasi nell'uso di API JDBC per l'interazione con un database gestito da un server DBMS prevede:

- 0) caricamento del driver (opzionale);
- 1) connessione al server DBMS;
- 2) esecuzione di comandi/*query* SQL;
- 3) recupero dei risultati;
- 4) disconnessione dal server DBMS.

■ **Caricamento del driver.** Il *driver manager* mantiene una lista di classi che implementano l'interfaccia *Driver*, ma la specifica implementazione di un driver deve essere registrata nel *driver manager*. Nelle versioni più recenti di JDBC la registrazione dei driver avviene automaticamente in fase di connessione con il server DBMS e non è necessario effettuarla esplicitamente.

■ **Connessione al server DBMS.** Il metodo statico *getConnection* della classe *DriverManager* restituisce un oggetto che implementa l'interfaccia *Connection* e che permette di interagire con il server DBMS; oltre allo username e alla password, il metodo *getConnection* richiede come parametro un URL il cui formato dipende dallo specifico DBMS utilizzato.

■ **Esecuzione di comandi/*query* SQL.** Per creare uno *statement* SQL – destinato a eseguire un comando o una *query* – è previsto il metodo *createStatement* di un oggetto che implementa l'interfaccia *Connection*; il metodo restituisce un oggetto che implementa l'interfaccia *Statement* che dispone di metodi per eseguire interrogazioni sui dati o comandi DML/DDI.

■ **Prepared statement.** Il metodo *prepareStatement* di un oggetto che implementa l'interfaccia *Connection* consente di creare *statement* predefiniti eseguibili più volte (*prepared statement*); questa tecnica riduce i tempi di esecuzione del comando/*query* per comandi/*query* che devono essere eseguiti molte volte. Utilizzando *statement* di questo tipo è possibile inserire nella stringa che rappresenta il comando SQL uno o più parametri identificati da simbolo «?», i cui valori dovranno essere forniti al momento dell'esecuzione effettiva della *query*.

■ **Recupero dei risultati.** Il metodo *executeQuery* definito dalle interfacce *Statement* e *PreparedStatement* restituisce un oggetto che implementa l'interfaccia *ResultSet*, che può contenere un risultato composto da una sequenza di record iterabile mediante il metodo *next*; l'accesso ai valori dei campi del singolo record avviene mediante metodi specifici che accettano come parametro l'indice posizionale del campo o il suo nome.

■ **Disconnessione dal server DBMS.** La disconnessione dal server DBMS si effettua invocando il metodo *close* dell'oggetto restituito dell'invocazione del metodo *getConnection*.

■ **CRUD (*Create, Read, Update, Delete*).** Acronimo con cui ci si riferisce alle operazioni che normalmente si effettuano su una base di dati: *Create* (inserimento di nuovi record di dati), *Read* (ricerca e lettura di record di dati), *Update* (aggiornamento di record di dati esistenti), *Delete* (eliminazione di record di dati). La realizzazione di una classe CRUD in linguaggio Java deve prevedere – oltre alla definizione di specifiche classi per la rappresentazione dei record di dati – lo sviluppo di una classe di gestione dei dati i cui metodi incapsolino l'accesso al database che rende persistenti i dati stessi.

■ **Oggetti *RowSet*.** *RowSet* è un'interfaccia che deriva dall'interfaccia *ResultSet*, ma che espone la tipica interfaccia di un oggetto *JavaBean*: un oggetto che implementa l'interfaccia *RowSet* ha lo scopo di mantenere una corrispondenza tra i dati che contiene nella forma di una sequenza di record e la loro sorgente, tipicamente un database relazionale al quale si connette utilizzando JDBC. Alcuni vantaggi del ricorso a *RowSet* rispetto a *ResultSet* consistono nel fatto che *RowSet* è sempre «naviga-

bile» in entrambe le direzioni e che l'aggiornamento dei dati che un oggetto *RowSet* contiene comporta l'aggiornamento della sorgente dei dati (nel caso di un database relazionale dei record di una o più tabelle).

■ **Notifica eventi con oggetti *RowSet*.** L'interfaccia *RowSet* prevede l'implementazione del modello a eventi dei componenti *JavaBean*. Gli oggetti *RowSet* notificano a un eventuale *listener* tre diversi tipi di eventi: movimento del cursore dei record (metodo *cursorMoved*), aggiornamento, inserimento e cancellazione di un record (metodo *rowChanged*), modifica del contenuto dell'oggetto (metodo *rowSetChanged*). Gli oggetti *listener* che ricevono gli eventi generati da oggetti che implementano l'interfaccia *RowSet* devono implementare l'interfaccia *RowSetListener* ed essere registrati per ogni specifico oggetto ai cui eventi sono interessati.

■ **JDBC e transazioni.** JDBC prevede la gestione transazionale di tutte quelle operazioni che prevedono una modifica dei dati contenuti nel database. Al momento della creazione di una connessione JDBC questa si trova in uno stato di *auto-commit*: ogni singolo comando SQL viene gestito come una transazione per la quale viene eseguito un *commit* automatico dopo la sua esecuzione. Avendo la necessità che un insieme di comandi siano eseguiti in modo atomico, è necessario disabilitare la modalità *auto-commit*, così che l'esecuzione dei comandi SQL non modifichi il database fino a quando non viene eseguito esplicitamente il *commit*. In questo modo, se si verifica un qualsiasi errore, è possibile annullare tutte le modifiche apportate, ma non ancora confermate, eseguendo un *rollback*.

## QUESITI

### 1 JDBC ...

- A ... è la versione di ODBC specifica per il linguaggio Java.
- B ... è una API Java che consente l'accesso esclusivamente al DBMS My-SQL.
- C ... è una API che permette l'accesso a qualsiasi DBMS.
- D Nessuna delle risposte precedenti è corretta.

### 2 Ordinare temporalmente le fasi della tipica sequenza di interazione di un'applicazione software con un server DBMS.

- A Esecuzione di comandi/query SQL.
- B Disconnessione dal server DBMS.
- C Connessione al server DBMS.
- D Recupero dei risultati ed eventuale gestione degli errori.

### 3 Associare il tipo di driver JDBC alla corretta descrizione:

Tipo 1	driver <i>pure Java</i> con connessione diretta al server DBMS
Tipo 2	ODBC <i>bridge</i>
Tipo 3	driver completamente realizzato in Java e <i>middleware</i> per DBMS specifico
Tipo 4	driver realizzato solo in parte in linguaggio Java

### 4 Associare le seguenti interfacce del package *java.sql* alla relativa funzionalità:

<i>Driver</i>	contenitore del risultato di una <i>query</i>
<i>Connection</i>	componente dipendente dallo specifico server DBMS
<i>Statement</i>	comando/query da inoltrare al server DBMS
<i>ResultSet</i>	gestione dell'interazione con il server DBMS

### 5 Quale eccezione sollevano i metodi delle classi/interfacce del package *java.sql*?

- A *JDBCException*
- B *SQLException*
- C *JavaSQLException*
- D *DBMSException*

### 6 Come viene istanziato un oggetto che implementa l'interfaccia *Connection*?

- A Invocando il metodo *getConnection* della classe *DriverManager*.
- B Utilizzando l'operatore **new** per creare un oggetto di tipo *Connection*.
- C Invocando il metodo *registerDriver* della classe *DriverManager*.
- D Nessuna delle risposte precedenti è corretta.

### 7 Quali sono i parametri del metodo *getConnection* della classe *DriverManager*?

- A Username di accesso al server DBMS.
- B URL dello specifico server DBMS.
- C Password di accesso al server DBMS.
- D Tutte le risposte precedenti sono corrette.

### 8 Come viene normalmente istanziato un oggetto che implementa l'interfaccia *ResultSet*?

- A Come risultato dell'invocazione del metodo *executeQuery* o *getResultSet* dell'interfaccia *Statement*.
- B Utilizzando l'operatore **new** per creare un oggetto di tipo *ResultSet*.
- C Come risultato dell'invocazione del metodo *createStatement* o *preparedStatement* dell'interfaccia *Connection*.
- D Nessuna delle risposte precedenti è corretta.

### 9 Associare i seguenti tipi SQL al corretto metodo dell'interfaccia *ResultSet* per il recupero dei relativi valori:

<b>VARCHAR</b>	<i>getFloat</i>
<b>DATE</b>	<i>getInt</i>
<b>INTEGER</b>	<i>getDate</i>
<b>REAL</b>	<i>getString</i>

**10** Il metodo *next* dell'interfaccia *ResultSet* ...

- A ... permette di iterare i singoli record del risultato di una *query*.
- B ... ripete l'esecuzione di una *query* per aggiornare i dati.
- C ... permette di iterare i singoli campi di un record fornito come risultato di una *query*.
- D *next* non è un metodo dell'interfaccia *ResultSet*.

**11** Cosa significa l'acronimo CRUD?

- A *Copy, Read, Update, Define*.
- B *Create, Read, Update, Delete*.
- C *Common Raw Uplink to DBMS*.
- D Nessuna delle risposte precedenti è corretta.

**12** Associare i seguenti tipi SQL ai tipi Java corrispondenti

BIT	<i>java.math.BigDecimal</i>
DATE	<i>java.sql.Time</i>
TIME	<i>boolean</i>
NUMERIC	<i>java.sql.Date</i>

**13** Quali delle seguenti affermazioni sono vere per un oggetto che implementa l'interfaccia *RowSet*?

- A Implementa tutti i metodi dell'interfaccia *ResultSet*.
- B È definita nel package *javax.sql*.
- C Garantisce l'aggiornamento della sorgente di dati.
- D Rispettano le convenzioni per i componenti *JavaBean*.

**14** Come viene selezionata la sorgente di dati di un oggetto che implementa l'interfaccia *RowSet*?

- A Invocando i metodi *setCommand* e *execute*.
- B Specificandola nell'argomento del costruttore.
- C Invocando il metodo *dataSource*.
- D Nessuna delle risposte precedenti è corretta.

**15** Associare le seguenti interfacce che derivano da *RowSet* con le caratteristiche relative:

<i>JdbcRowSet</i>	possibilità di unione dei dati senza collegamento alla sorgente
<i>JoinRowSet</i>	esportazione/importazione di dati in formato XML
<i>WebRowSet</i>	selezione dei dati visibili rispetto ai dati contenuti
<i>FilteredRowSet</i>	connessione permanente al server DBMS

**16** Quali dei seguenti eventi sono generati da un oggetto che implementa l'interfaccia *RowSet*?

- A *cursorMoved*
- B *rowDeleted*
- C *rowUpdated*
- D *rowInserted*

## ESERCIZI

**1** Si consideri il database rappresentato nel diagramma di FIGURA 6. Dopo avere definito una specifica classe Java per rappresentare oggetti di tipo *Libro*, realizzare una classe Java CRUD i cui metodi consentano di:

- a) ottenere l'elenco dei libri stampati, con indicazione del codice ISBN, del titolo, dell'autore e dell'editore, in un determinato periodo di tempo indicato specificando l'anno di inizio e di fine;
- b) eliminare un libro dal database a partire dall'indicazione del codice ISBN;
- c) inserire un nuovo libro nel database.

**2** Un'associazione di *car-sharing* mantiene un database del parco auto e dei soci al fine di registrare i noleggi e verificare le disponibilità delle vetture; il diagramma del database è quello di FIGURA 7. Dopo avere definito specifiche classi Java per rappresentare oggetti di tipo *Auto* e *Noleggio*, realizzare una classe Java CRUD i cui metodi consentano di:

- a) ottenere l'elenco delle auto disponibili in un determinato periodo compreso tra due date;

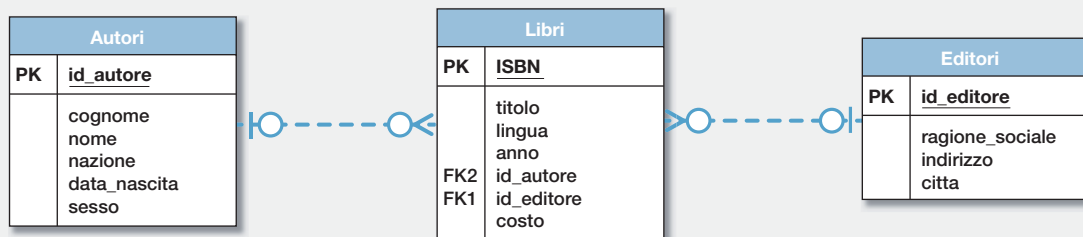


FIGURA 6

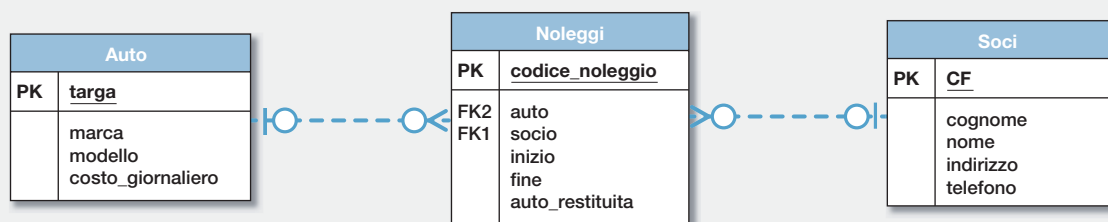


FIGURA 7

- b) ottenere l'elenco dei noleggi effettuati da un socio in un determinato periodo;
- c) inserire un nuovo noleggio;
- d) aggiornare la situazione relativa a un noleggio al momento in cui un'auto viene restituita.

**3** Un'agenzia espressa ha sedi e clienti in varie città: i clienti mittenti consegnano i plichi alla sede di partenza e i clienti destinatari ritirano i plichi nella sede di arrivo. Dalla spedizione alla consegna ogni plico è identificato da un codice numerico: la data/ora di spedizione viene registrata quando

il mittente consegna il plico, mentre la data/ora di consegna viene registrata quando il destinatario ritira il plico. Il sistema informatico dell'agenzia si basa su un database realizzato a partire dal diagramma di FIGURA 8. Dopo avere definito specifiche classi Java per rappresentare oggetti di tipo *Cliente*, *Spedizione* e *Sede*, realizzare una classe Java CRUD i cui metodi consentano di gestire le seguenti operazioni:

- a) ricezione dal mittente di una nuova spedizione: dato che il codice di spedizione è un valore

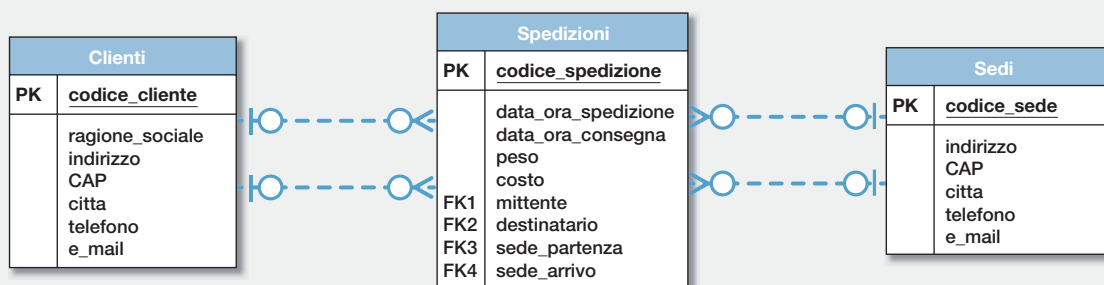


FIGURA 8

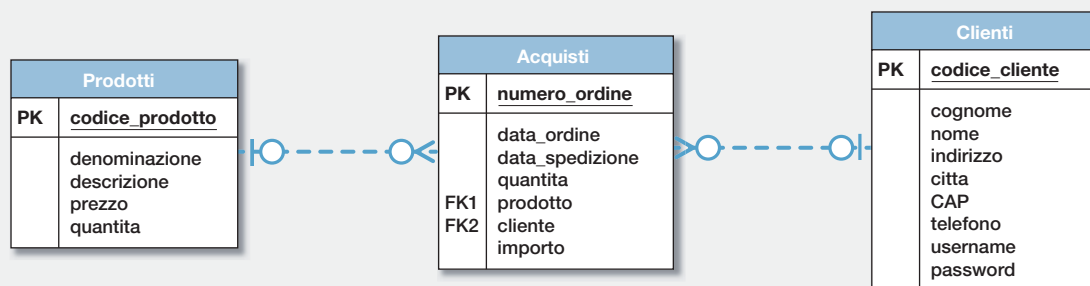


FIGURA 9

- numerico progressivo deve essere restituito il codice assegnato alla spedizione;
- b) elenco delle spedizioni non ancora consegnate con indicazione del codice, della data/ora di spedizione, della ragione sociale del destinatario e della sede di arrivo;
  - c) consegna al destinatario di una spedizione;
  - d) elenco delle spedizioni effettuate oggi da una specifica sede dell'agenzia;
  - e) stato (consegnata o meno) di una specifica spedizione identificata dal codice numerico con indicazione della sede di partenza e di arrivo.
- 4** Un nuovo sito Internet per la vendita di prodotti on-line utilizzerà il database illustrato nel diagramma di FIGURA 9. Dopo avere definito specifiche classi Java per rappresentare oggetti di tipo

*Cliente, Acquisti e Prodotti*, realizzare una classe Java CRUD i cui metodi consentano di gestire le seguenti operazioni:

- a) ottenere l'elenco dei prodotti disponibili per la vendita con indicazione del codice, del prezzo e della quantità disponibile;
- b) variazione della quantità disponibile di uno specifico prodotto;
- c) inserire la data di spedizione di un ordine da evadere di cui è noto il numero;
- d) aggiungere un nuovo prodotto a quelli già in vendita;
- e) ordinare una quantità variabile (inferiore ovviamente alla disponibilità) di un singolo prodotto ottenendo il numero come risultato;
- f) ottenere lo stato (spedito o meno) di un acquisto individuato dal numero di ordine.



# Il linguaggio XML per la rappresentazione dei dati

1. La versione 1.1 di XML risale al 2004, ma non è ancora oggi diffusamente utilizzata.

Lo *Extensible Markup Language* o XML è stato concepito da un gruppo di lavoro del World Wide Web Consortium (W3C) a metà degli anni '90 del secolo scorso, inizialmente come soluzione ad alcune problematiche emerse dall'evoluzione di HTML come linguaggio per la definizione delle pagine web: la versione 1.0 di XML è divenuta una raccomandazione del W3C nel 1998<sup>1</sup>.

**OSSERVAZIONE** XML è un sottoinsieme («profilo») di SGML (*Standard Generalized Markup Language*), un linguaggio per la definizione di linguaggi basati su tag marcatori: XML, come SGML, è un «metalinguaggio» che consente di definire altri linguaggi. Il linguaggio HTML è stato inizialmente sviluppato dal creatore Tim Berners-Lee come un'applicazione di SGML.

La versatilità di XML come metalinguaggio per la definizione di linguaggi specifici lo ha reso uno strumento utilizzato principalmente per definire documenti con marcatori in cui i tag non sono predefiniti e collezioni di dati semi-strutturati rappresentate in formato testuale. È il formato testuale di XML che lo ha reso lo standard più utilizzato per l'interscambio di documenti prodotti con applicazioni software diverse e di dati tra DBMS di diversi produttori.

## World Wide Web Consortium (W3C)

Il World Wide Web Consortium è un'organizzazione non governativa fondata nel 1994 da Tim Berners-Lee con lo scopo di «sviluppare le potenzialità del web».

I membri del W3C sono aziende informatiche e di telecomunicazioni, istituti di ricerca pubblici e privati e università. Le attività del W3C producono «raccomandazioni» che nel campo del web sono dei veri e propri standard.

L'autorevolezza del W3C è data dal fatto che la maggior parte delle tecnologie software del web di vasta diffusione (HTTP, HTML, CSS, XML, PNG, ...) sono state definite dal consorzio.

## ESEMPIO

Il seguente file di testo è un documento XML che rappresenta gli indirizzi di posta elettronica e le password degli utenti di un sistema informatico:

```
<?xml version="1.0" ?>
<users>
  <!-- Nota: le password sono cifrate. -->
  <user>
    <username>Pippo</username>
    <email>pippo32@disney.com</email>
    <password>0A1B2C3D4E5F</password>
  </user>
  <user>
    <username>Pluto</username>
    <email>pluto30@disney.com</email>
    <password>A9B8C7D6E5F4</password>
  </user>
</users>
```

**OSSERVAZIONI** A differenza di HTML, che è stato concepito per la visualizzazione di dati in formato ipertestuale e multimediale, XML è un linguaggio finalizzato alla memorizzazione e all'interscambio di dati, non alla visualizzazione che, eventualmente, avviene applicando a un file XML regole di trasformazione basate sui fogli di stile CSS.

Come risulta evidente dall'esempio precedente un corretto uso di XML – in particolare un'attenta scelta dei nomi dei tag – rende i documenti in questo formato, oltre che ideali per la gestione automatica da parte di applicazioni software, autodescrittivi per la lettura da parte dell'Uomo, anche se il linguaggio è stato spesso criticato per la sua verbosità.

## XHTML

È la versione del linguaggio HTML modificata in modo che ogni pagina HTML sia un documento XML valido.

Dato che le regole per la nidificazione dei tag sono rigorosamente gerarchiche – come in HTML – i documenti in formato XML hanno naturalmente una struttura ad albero che viene utilizzata da molti strumenti che ne consentono la gestione.

### ESEMPIO

Il documento dell'esempio precedente presenta la struttura ad albero di FIGURA 1, che deriva dalla struttura di nidificazione dei tag con l'elemento che racchiude tutti gli altri e che diviene il nodo radice dell'albero.

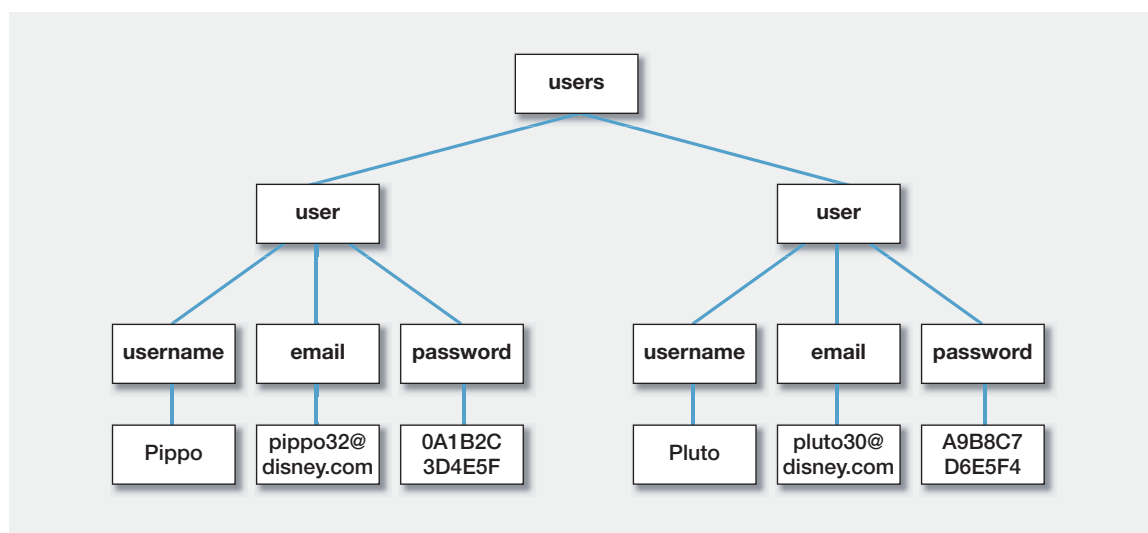


FIGURA 1

## 1 La sintassi del linguaggio XML e la struttura ad albero dei documenti

Le regole di base della sintassi XML non sono complesse:

- tutti i documenti XML devono iniziare con la seguente dichiarazione che identifica la versione di riferimento del linguaggio:

```
<?xml version="1.0" ?>
```



- i nomi dei tag possono iniziare esclusivamente con un carattere alfabetico o con il simbolo «\_»;
- i nomi dei tag possono contenere esclusivamente caratteri alfabetici, cifre numeriche e i simboli «\_», «-» e «.» (sono quindi esclusi gli spazi);
- i nomi dei tag sono sensibili alla differenza tra caratteri minuscoli e maiuscoli;
- tutti i tag devono essere chiusi con un tag corrispondente avente lo stesso nome preceduto dal simbolo «/»;
- la nidificazione dei tag aperti/chiusi deve essere rigorosamente gerarchica;
- l'intero documento XML, a esclusione della dichiarazione iniziale, deve essere compreso in un solo elemento «radice», cioè tra due tag accoppiati di apertura e chiusura dell'elemento;
- i tag possono avere attributi per i quali è possibile specificare valori che devono essere necessariamente compresi tra due simboli «"» o «'»;
- i simboli chiave del linguaggio sono rappresentati mediante le seguenti entità:

<	&lt;
>	&gt;
&	&amp;
"	&quot;
'	&apos;

- un commento è compreso tra i simboli «<!--» e «-->».

#### ESEMPIO

Il seguente file di testo è un documento XML corretto che rappresenta un messaggio SMS:

```
<?xml version="1.0" ?>
<MESSAGE timestamp="07/10/2012 12:00:00">
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text>C&apos;erano molti invitati al tuo compleanno oggi?</text>
</MESSAGE>
```

L'elemento radice è *MESSAGE* e *timestamp* è un suo attributo.

- Un **elemento XML** è quanto è compreso tra un tag di apertura e il corrispondente tag di chiusura (inclusi); un elemento può contenere testo, attributi e altri elementi, oppure essere vuoto.

**OSSERVAZIONE** L'elemento *MESSAGGIO* dell'esempio precedente comprende l'attributo *timestamp* e gli elementi *from*, *to* e *text* che contengono esclusivamente testo.

Esiste una sintassi abbreviata per gli elementi vuoti, o che comprendono solo attributi, che impiega un solo tag che contemporaneamente apre e chiude l'elemento stesso, come nei seguenti esempi:

`<tag/>`

`<tag attributo="..."/>`

Il linguaggio XML prevede un attributo predefinito speciale denominato `xml:lang` per indicare la lingua del testo contenuto in un elemento; i valori più comuni sono i seguenti:

en	Inglese
fr	Francese
de	Tedesco
es	Spagnolo
it	Italiano

#### ESEMPIO

Il documento XML dell'esempio precedente dovrebbe riportare l'attributo per l'indicazione del linguaggio:

```
<?xml version="1.0" ?>
<MESSAGE timestamp="07/10/2012 12:00:00">
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text xml:lang="it">
    C&apos;erano molti invitati al tuo compleanno oggi?
  </text>
</MESSAGE>
```

Dato che un elemento può contenere a sua volta altri elementi, gli strumenti software che gestiscono i documenti XML analizzano l'intero contenuto di ogni elemento. Per evitare che il contenuto di un elemento sia analizzato e permettere che al suo interno possa essere inserita qualsiasi sequenza di caratteri, è possibile qualificarlo come CDATA (*Character DATA*) inserendolo tra i simboli «`<![CDATA[`» e «`]]>`».

#### ESEMPIO

Il seguente file di testo è un documento XML corretto che rappresenta un messaggio SMS:

```
<?xml version="1.0" ?>
<MESSAGE timestamp="07/10/2012 12:00:00">
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text>
    <![CDATA[
```

```

Ti invio un esempio di XML:
<?xml version="1.0" ?>
<users>
  <user>
    <username>Pippo</username>
    <email>pippo32@disney.com</email>
  </user>
  <user>
    <username>Pluto</username>
    <email>pluto30@disney.com</email>
  </user>
</users>
]]>
</text>
</MESSAGE>

```

2. Il formato in cui viene salvato il file deve essere coerente con il tipo di codifica specificato: singolo byte per ASCII, UTF-8, Windows-1252 e ISO-8859-1, multibyte per UTF-16.

**OSSERVAZIONE** La possibilità di definire testo di tipo CDATA consente di inserire codice HTML come contenuto di un elemento di un file XML.

Se il file che contiene un documento XML contiene caratteri che non appartengono all'insieme ASCII, come per esempio i caratteri accentati della nostra lingua, è necessario specificare il tipo di codifica utilizzato di cui la **TABELLA 1** riporta i più utilizzati<sup>2</sup>.

**TABELLA 1**

Windows-1252	Codifica a 8 bit dei caratteri dell'Europa occidentale adottata dai sistemi operativi Windows
ISO-8859-1	Codifica standard a 8 bit dei caratteri dell'Europa occidentale
UTF-8	Codifica Unicode che utilizza da 8 a 32 bit per il singolo carattere
UTF-16	Codifica Unicode che utilizza 16 bit per il singolo carattere adottata dal linguaggio Java

La codifica dei caratteri viene specificata come attributo della dichiarazione XML iniziale:

```
<?xml version="1.0" encoding="..." ?>
```

**ESEMPIO**

Il seguente file di testo è un documento XML corretto che rappresenta un messaggio SMS:

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<MESSAGE timestamp="07/10/2012 12:00:00">
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text>Perché non festeggi il tuo compleanno oggi?</text>
</MESSAGE>

```

In XML gli attributi dovrebbero essere utilizzati esclusivamente per fornire informazioni accessorie e non per rappresentare i dati.

#### ESEMPIO

Se la data/ora di ricezione di un messaggio SMS è un dato che lo caratterizza, l'esempio precedente dovrebbe essere riscritto come segue, trasformando l'attributo *timestamp* negli elementi *date* e *time*:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<MESSAGE>
  <date>07/10/2012</date>
  <time>12:00:00</time>
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text>Perché non festeggi il tuo compleanno oggi?</text>
</MESSAGE>
```

È sempre possibile rappresentare la struttura di un documento XML sintatticamente corretto mediante un diagramma ad albero in cui il nodo radice è l'elemento radice del documento e i nodi dell'albero sono gli elementi stessi, gli attributi degli elementi, il contenuto testuale degli elementi e gli eventuali commenti inseriti nel documento.

#### ESEMPIO

Il seguente file di testo è un documento XML corretto che rappresenta un libro:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libro genere="fantascienza" xml:lang="it">
  <autore>Stefano Benni</autore>
  <titolo>Terra!</titolo>
  <editore>Feltrinelli</editore>
  <anno>1983</anno>
</libro>
```

Esso è rappresentato dal diagramma ad albero di FIGURA 2.

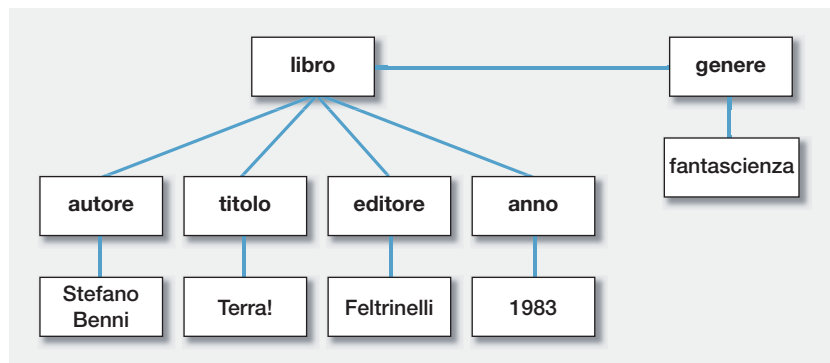


FIGURA 2

### La visualizzazione di un documento XML

I browser, gli editor avanzati e gli ambienti software per gli sviluppatori visualizzano un file contenente un documento XML in formato testuale fornendo strumenti per nascondere il contenuto di un elemento e facilitare di conseguenza la lettura della gerarchia degli elementi del documento stesso. Volendo realizzare una pagina web che visualizzi il contenuto di un documento XML in modo appropriato è necessario specificare un documento in linguaggio XSL (*eXtensible Style-sheet Language*).

XSL comprende XSLT (*eXtensible Style-sheet Language Transformations*), che consente di specificare regole per la trasformazione di un documento XML in un diverso documento XML, o in una pagina HTML o XHTML.

## 2 La definizione di linguaggi XML mediante schemi XSD

► Un documento XML che rispetta le regole sintattiche di base è denominato **ben formato**.

**OSSERVAZIONE** La verifica del rispetto delle regole sintattiche di un documento XML può essere effettuata utilizzando specifici strumenti software: molti browser o editor specializzati per programmatori sono in grado di verificare che un documento XML sia ben formato. All'URL [validator.w3.org](http://validator.w3.org) è disponibile uno strumento utilizzabile on-line per la verifica sintattica di documenti XML e di pagine HTML.

Dato che XML viene utilizzato per definire nuovi linguaggi specifici che utilizzano tag marcatori non predefiniti, anche se un documento XML è ben formato rimane aperto il problema di verificare che impieghi effettivamente i tag previsti e in modo corretto; inoltre il contenuto di un elemento rappresenta spesso un dato che deve rispettare regole di tipizzazione.

### Linguaggi XML

Il successo di XML è legato alla sua flessibilità: spesso le aziende e gli sviluppatori software definiscono schemi XML per i file che costituiscono l'input o l'output delle applicazioni che realizzano.

Per esempio, i formati dei file prodotti dalle applicazioni *Office* di Microsoft sono in formato XML (da cui deriva il carattere «x» che ne termina l'estensione del nome).

In alcuni casi sono stati definiti dei veri e propri linguaggi XML aventi scopi specifici; tra questi ricordiamo: **CML** (*Chemical Markup Language*), che è usato per la definizione di strutture molecolari; **EPUB** (*Electronic Publication*), che è un formato aperto per la pubblicazione di e-book; **GML** (*Geographic Markup Language*) e **KML** (*Keyhole Markup Language*), che sono linguaggi per la definizione di risorse georeferenziate; **MathML**, che è impiegato per la codifica della notazione matematica; **ODF** (*Open Document Format*), che è un formato aperto per i documenti delle applicazioni di utilità; **SVG** (*Scalable Vector Graphics*), che è un formato utilizzato per la grafica vettoriale bidimensionale.

### ESEMPIO

Il seguente documento XML che rappresenta una collezione di libri è ben formato:

```
<?xml version="1.0" ?>
<libri>
  <libro genere="fantascienza">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <editore>Feltrinelli</editore>
    <anno>83</anno>
  </libro>
  <libro genere="romanzo">
    <scrittore>Mark Twain</scrittore>
    <titolo>Le avventure di Huckleberry Finn</titolo>
    <anno>1884</anno>
  </libro>
  <libro>
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <editore/>
    <anno>milleottocentocinquantuno</anno>
  </libro>
</libri>
```

Tuttavia i dati relativi ai singoli libri non sono presenti in modo coerente: per «Moby Dick» l'anno di pubblicazione non è espresso in formato numerico e l'editore non è indicato; per il capolavoro di Mark Twain non è specificato un elemento *editore* e l'autore è definito come contenuto dell'elemento *scrittore*; infine l'anno di pubblicazione di «Terra!» è indicato in forma numerica abbreviata.

Un file di questo tipo risulterebbe di difficile elaborazione da parte di un'applicazione software.

L'esempio precedente evidenzia la necessità di un metodo per definire la struttura che un documento XML relativo a una specifica applicazione deve avere e la tipologia che i dati contenuti negli elementi devono rispettare. Il metodo più comune per definire uno schema di validazione per un documento XML è XSD (*XML Schema Definition*), inizialmente proposto da W3C nel 2001<sup>3</sup>.

Uno schema XSD definisce:

- gli elementi di un documento XML e la loro relazione gerarchica;
- gli eventuali attributi degli elementi;
- il numero e l'ordinamento degli elementi;
- gli eventuali elementi vuoti;
- il tipo dei dati contenuti negli elementi e negli attributi;
- i valori predefiniti o costanti del contenuto degli elementi e degli attributi.

Il linguaggio XSD che consente di definire lo schema di un documento XML è a sua volta un linguaggio XML: ogni schema XSD deve avere come radice un elemento denominato **schema** con un attributo predefinito che riferisce il file contenente gli elementi e i tipi di dati standard utilizzabili:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Inoltre ogni documento XML dovrebbe dichiarare il file contenente lo schema di riferimento inserendo i seguenti attributi predefiniti nella definizione dell'elemento radice:

```
xmlns:xsi="http://www.w3.org/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..."
```

**OSSERVAZIONE** Essendo XML una tecnologia molto diffusa nelle applicazioni web, spesso il file dello schema di riferimento è una risorsa esposta in rete e definita da un URL.

#### ESEMPIO

Un possibile schema XSD per un documento XML che rappresenta un singolo libro potrebbe essere il seguente:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="autore" type="xs:string"/>
        <xs:element name="titolo" type="xs:string"/>
        <xs:element name="editore" type="xs:string"/>
        <xs:element name="anno" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3. La versione 1.1 di XSD è divenuta una raccomandazione W3C nel 2012.

#### Document Type Definition (DTD)

Il primo formato standard per la definizione della struttura di un documento XML è stato DTD: *Document Type Definition*. DTD presentava molte limitazioni – tra cui il fatto di non avere una sintassi compatibile con XML – che sono state superate da XSD: *XML Schema Definition*.

## Validazione di documenti XML con Notepad++

Notepad++ ([notepad-plus-plus.org](http://notepad-plus-plus.org)) è un editor per sviluppatori che utilizzano il sistema operativo Windows, gratuito e molto noto.

Tra i numerosi *plug-in* di cui dispone Notepad++, *XML tools* consente di verificare che un file in formato XML sia ben formato e di validare un file in formato XML rispetto a uno schema XSD.

4. La verifica di uno schema XML nella forma di un documento XSD può essere effettuata con lo strumento software open-source XSV, sviluppato dall'Università di Edimburgo e disponibile on-line sul sito del World Wide Web Consortium.

```
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Lo schema definisce un documento XML che deve avere un elemento radice *libro* con un attributo *genere* di tipo stringa di caratteri e gli elementi *autore*, *titolo* ed *editore* di tipo stringa di caratteri più l'elemento *anno* di tipo numerico intero. Un documento XML deve riferire lo schema a cui si attiene nella definizione dell'elemento radice, come nel seguente documento XML, in cui si assume che lo schema precedente sia stato salvato nel file »libro.xsd«:

```
<?xml version="1.0" ?>
<libro
  xmlns:xsi="http://www.w3.org/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="libro.xsd">
  <autore>Stefano Benni</autore>
  <titolo>Terra!</titolo>
  <editore>Feltrinelli</editore>
  <anno>1983</anno>
</libro>
```

**OSSERVAZIONE** Progettare un linguaggio XML significa definirne lo schema, cioè la sintassi che un qualsiasi documento XML deve rispettare per appartenere al linguaggio.

► Un documento XML **valido** è un documento XML ben formato che rispetta le regole dello schema che riferisce.

La validazione di un documento XML rispetto allo schema riferito viene normalmente effettuata mediante strumenti software automatici<sup>4</sup>.

## 2.1 La definizione degli elementi semplici e degli attributi

Un **elemento semplice** è un elemento XML che può contenere solo testo: non può contenere attributi o altri elementi. Dato che XML è un formato testuale, il testo può rappresentare stringhe di caratteri, oppure valori numerici, date, orari ecc.

In uno schema XSD è possibile imporre un tipo al testo contenuto in un elemento.

La sintassi per definire un elemento semplice in uno schema XSD è la seguente:

```
<xs:element name="..." type="..."/>
```

I tipi più comuni per un elemento semplice sono riportati nella TABELLA 2:

TABELLA 2

xs:string	Stringhe di caratteri
xs:decimal	Valori numerici non necessariamente interi
xs:integer	Valori numerici interi
xs:date	Data nel formato aaaa-mm-gg (a: anno, m: mese, g: giorno)
xs:time	Ora nel formato oo-mm-ss (o: ora, m: minuti, s: secondi)

ESEMPI

Esempi di definizione di elementi semplici:

```
<xs:element name="nome" type="xs:string"/>
<xs:element name="dataNascita" type="xs:date"/>
<xs:element name="età" type="xs:integer"/>
```

e di relativi elementi XML validi:

```
<nome>Walt Disney</nome>
<dataNascita>1901-12-05</dataNascita>
<età>65</età>
```

e invalidi:

```
<nome/>
<dataNascita>5/12/1901</dataNascita>
<età>sessantacinque</età>
```

Un elemento semplice può avere un valore predefinito che assume nel caso che il documento XML non ne specifichi uno, o un valore costante che impedisce che nel documento XML ne possa essere definito uno diverso: per la definizione si utilizzano rispettivamente gli attributi *default* e *fixed*.

ESEMPI

Esempi di definizione di elementi semplici con valori predefiniti e costanti:

```
<xs:element name="colore" type="xs:string" default="rosso"/>
<xs:element name="piGreco" type="xs:decimal" fixed="3.14159"/>
```

e di relativi elementi XML validi:

```
<colore>verde</colore>
<colore/>
<piGreco>3.14159</piGreco>
```

e invalidi:

```
<piGreco>3.1416</piGreco>
```

Nella definizione di un elemento semplice è possibile impostare delle «restrizioni» al tipo base indicato. Per i tipi numerici sono possibili le restrizioni indicate nella TABELLA 3:



TABELLA 3

<code>xs:fractionDigits</code>	Massimo numero di cifre decimali
<code>xs:maxExclusive</code> <code>xs:maxInclusive</code>	Valore massimo (rispettivamente escluso e incluso)
<code>xs:minExclusive</code> <code>xs:minInclusive</code>	Valore minimo (rispettivamente escluso e incluso)
<code>xs:totalDigits</code>	Numero esatto di cifre

## ESEMPI

Esempi di definizione di elementi semplici con restrizioni:

```
<xs:element name="quantità">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minExclusive value="0"/>
      <xs:maxInclusive value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="prezzo">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

e di relativi elementi XML validi:

```
<quantità>5</quantità>
<quantità>10</quantità>
<prezzo>0.99</prezzo>
<prezzo>99</prezzo>
```

e invalidi:

```
<quantità>0</quantità>
<quantità>100</quantità>
<prezzo>1.999</prezzo>
```

### Le espressioni regolari e la definizione dei pattern

Una possibile restrizione per il tipo stringa di caratteri consiste nello specificare un *pattern* cui la stringa deve corrispondere. Per esempio un codice fiscale deve essere composto nell'ordine da 6 caratteri alfabetici maiuscoli, 2 cifre numeriche, 1 carattere alfabetico maiuscolo, 2 cifre numeriche, 1 carattere alfabetico maiuscolo, 3 cifre numeriche e un carattere alfabetico maiuscolo finale.

Il linguaggio XSD consente di definire questo tipo di condizioni mediante la restrizione `xs:pattern` e un valore che rappresenta un'«espressione regolare» standard.

**OSSERVAZIONE** Come risulta evidente nell'esempio precedente l'impostazione di restrizioni a un tipo di base impone l'uso della notazione estesa per definire un elemento semplice:

```
<xs:element name="...">
  <xs:simpleType>
    <xs:restriction base="xs:...">
      ...
      <!--restrizioni -->
      ...
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Per le stringhe di caratteri sono possibili le restrizioni indicate nella TABELLA 4:

TABELLA 4

xs:length	numero esatto di caratteri
xs:maxLength	numero massimo di caratteri
xs:minLength	numero minimo di caratteri

ESEMPI

Esempi di definizione di elementi semplici con restrizioni:

```
<xs:element name="indirizzo">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="10"/>
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="CAP">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

e di relativi elementi XML validi:

```
<indirizzo>Via delle Fornaci, 20 Livorno</indirizzo>
<CAP>57128</CAP>
```

e invalidi:

```
<indirizzo>Via corta</indirizzo>
<CAP>1234</CAP>
```

Una restrizione molto comune del tipo di base stringa di caratteri – l'enumerazione – consente di elencare le stringhe valide.

ESEMPIO

L'elemento *colore* può contenere una sola tra le stringhe: «bianco», «nero», «rosso», «verde», «blu», «celeste», «violetto» e «giallo»:

```
<xs:element name="colore">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="bianco"/>
      <xs:enumeration value="nero"/>
      <xs:enumeration value="rosso"/>
      <xs:enumeration value="verde"/>
```



```

        <xs:enumeration value="blu"/>
        <xs:enumeration value="celeste"/>
        <xs:enumeration value="violetto"/>
        <xs:enumeration value="giallo"/>
    </xs:restriction>
</xs:simpleType>
</xs:element>

```

I seguenti sono elementi XML validi:

```

        <colore>nero</colore>
        <colore>giallo</colore>

```

e invalidi:

```

        <colore>grigio</colore>
        <colore/>

```

**OSSERVAZIONE** Un tipo enumerativo può essere definito autonomamente e successivamente utilizzato per definire uno o più elementi. L'esempio precedente può essere così riscritto:

```

<xs:simpleType name="nomeColore">
    <xs:restriction base="xs:string">
        <xs:enumeration value="bianco"/>
        <xs:enumeration value="nero"/>
        <xs:enumeration value="rosso"/>
        <xs:enumeration value="verde"/>
        <xs:enumeration value="blu"/>
        <xs:enumeration value="celeste"/>
        <xs:enumeration value="violetto"/>
        <xs:enumeration value="giallo"/>
    </xs:restriction>
</xs:simpleType>

<xs:element name="colore" type="nomeColore"/>

```

Un elemento semplice non può prevedere attributi, ma in uno schema XSD un attributo viene definito in modo analogo a un elemento semplice:

```

<xs:attribute name="..." type="..."/>

```

**OSSERVAZIONE** Un attributo è normalmente facoltativo; per renderlo obbligatorio nella definizione deve essere valorizzato con «required» l'attributo *use*:

```

<xs:attribute name="..." type="..." use="required"/>

```

## 2.2 La definizione degli elementi complessi

Un **elemento complesso** è un elemento XML che ricade in una delle seguenti categorie:

- è un elemento vuoto, eventualmente con attributi;
- è un elemento che contiene altri elementi;
- è un elemento che contiene solo testo, ma con attributi;
- è un elemento che contiene sia testo sia altri elementi.

Un elemento complesso può essere definito direttamente, oppure mediante un «tipo» che permette di definire più elementi complessi.

### ESEMPIO

La definizione degli elementi *docente* e *studente*

```
<xs:element name="docente">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="cognome" type="xs:string"/>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="dataNascita" type="xs:date"/>
      <xs:element name="luogoNascita" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="studente">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="cognome" type="xs:string"/>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="dataNascita" type="xs:date"/>
      <xs:element name="luogoNascita" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

può essere riscritta utilizzando il seguente tipo *persona*:

```
<xs:complexType name="persona">
  <xs:sequence>
    <xs:element name="cognome" type="xs:string"/>
    <xs:element name="nome" type="xs:string"/>
    <xs:element name="dataNascita" type="xs:date"/>
    <xs:element name="luogoNascita" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="docente" type="persona"/>
<xs:element name="studente" type="persona"/>
```

**OSSERVAZIONE** L'indicatore `<xs:sequence>` utilizzato nello schema XSD dell'esempio precedente impone che gli elementi complessi *docente* e *studente* contengano ciascuno gli elementi semplici *cognome*, *nome*, *dataNascita* e *luogoNascita* nell'ordine specificato, come nel seguente esempio:

```
<docente>
  <cognome>Meini</cognome>
  <nome>Giorgio</nome>
  <dataNascita>1965-03-23</dataNascita>
  <luogoNascita>Livorno</luogoNascita>
</docente>
```

Un elemento vuoto è un elemento complesso così definito:

```
<xs:element name="elementoVuoto">
  <xs:complexType>
  </xs:complexType>
</xs:element>
```

per il quale sono validi elementi XML come il seguente:

```
<elementoVuoto></elementoVuoto>
<elementoVuoto/>
```

Un elemento vuoto con un attributo è invece un elemento complesso così definito:

```
<xs:element name="elementoVuoto">
  <xs:complexType>
    <xs:attribute name="attributo" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

per il quale sono validi elementi XML come il seguente:

```
<elementoVuoto></elementoVuoto>
<elementoVuoto attributo="valore"/>
```

Il modo più comune per definire un elemento complesso che contiene esclusivamente altri elementi è il ricorso all'indicatore `<xs:sequence>` che nello schema XSD comprende l'elenco ordinato degli elementi contenuti.

#### ESEMPIO

La seguente definizione dell'elemento *indirizzo*

```
<xs:element name="indirizzo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="via" type="xs:string"/>
      <xs:element name="numero">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
```



```

        <xs:minInclusive value="1"/>
    </xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="CAP">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="5"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="città" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>

```

valida elementi XML come questo

```

<indirizzo>
  <via>Via delle Fornaci</via>
  <numero>20</numero>
  <CAP>57128</CAP>
  <città>Livorno</città>
</indirizzo>

```

ma non come quello che segue

```

<indirizzo>
  <via>Via delle Fornaci</via>
  <numero>20</numero>
  <città>Livorno</città>
  <CAP>57128</CAP>
</indirizzo>

```

in quanto l'indicatore `<xs:sequence>` vincola l'ordine degli elementi contenuti nell'elemento complesso *indirizzo*.

Un elemento XML che contiene solo testo è un elemento semplice, ma un elemento che contiene solo testo e prevede uno o più attributi è un elemento complesso con contenuto semplice. Elementi complessi di questo tipo possono essere definiti ricorrendo a una «estensione» di un tipo base:

```

<xs:element name="elementoConAttributo">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="...">
        <attribute name="..." type="..." />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

```

La seguente definizione dell'elemento *misura*

```
<xs:element name="misura">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="unità" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

valida elementi XML come questi

```
<misura unità="cm">100</misura>
<misura unità="mm">0.001</misura>
```

ma non come quello che segue

```
<misura>10</misura>
```

in quanto l'attributo `use="required"` nello schema rende la presenza dell'attributo *unità* obbligatoria.

Per definire un elemento XML che possa contenere sia testo sia elementi è necessario specificare l'attributo `mixed="true"` nella definizione di un tipo complesso.

La seguente definizione dell'elemento *messaggio*

```
<xs:element name="messaggio">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="codice" type="xs:string"/>
      <xs:element name="data" type="xs:date"/>
      <xs:element name="ora" type="xs:time"/>
      <xs:element name="luogo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

permette di validare elementi XML come quello che segue

```
<messaggio>
```

La spedizione <codice>0123456789</codice> è stata consegnata il giorno <data>2012-10-22</data> alle ore <ora>15-15-30</ora> a <luogo>Livorno</luogo>

```
</messaggio>
```

Oltre all'indicatore `<xs:sequence>` esistono altri due indicatori per definire gli elementi contenuti in un elemento complesso:

- `<xs:all>` è analogo a `<xs:sequence>`, ma non impone il rispetto dell'ordine degli elementi;
- `<xs:choice>` permette di elencare due o più elementi di cui uno solo deve necessariamente essere presente.

La seguente definizione dell'elemento *indirizzo*

```
<xs:element name="indirizzo">
  <xs:complexType>
    <xs:all>
      <xs:element name="via" type="xs:string"/>
      <xs:element name="numero">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CAP">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:length value="5"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="città" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

valida entrambi i seguenti elementi XML

```
<indirizzo>
  <via>Via delle Fornaci</via>
  <numero>20</numero>
  <CAP>57128</CAP>
  <città>Livorno</città>
</indirizzo>

<indirizzo>
  <via>Via delle Fornaci</via>
  <numero>20</numero>
  <città>Livorno</città>
  <CAP>57128</CAP>
</indirizzo>
```

in quanto l'indicatore `<xs:all>` non vincola l'ordine degli elementi contenuti nell'elemento complesso *indirizzo*.

Data la seguente definizione dell'elemento complesso *referimentoCliente*

```
<xs:element name="referimentoCliente">
  <xs:complexType>
    <xs:choice>
      <xs:element name="codiceFiscale">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:length value="16"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
```





```

<xs:element name="partitaIVA">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="11"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>

```

sono validi i seguenti elementi XML

```

<referimentoCliente>
  <codiceFiscale>MNEGRG65C23E625Y</codiceFiscale>
</referimentoCliente>

<referimentoCliente>
  <partitaIVA>01487450494</partitaIVA>
</referimentoCliente>

```

ma non è valido il seguente elemento XML

```

<referimentoCliente>
  <codiceFiscale>MNEGRG65C23E625Y</codiceFiscale>
  <partitaIVA>01487450494</partitaIVA>
</referimentoCliente>

```

perché l'indicatore `<xs:choice>` impone di scegliere un solo elemento tra quelli indicati.

Tutti gli indicatori presi in esame consentono una sola occorrenza di ogni singolo elemento: gli attributi *minOccurs* e *maxOccurs* permettono di indicare il numero minimo e massimo di occorrenze di ogni elemento.

**OSSERVAZIONE** Lo speciale valore «*unbounded*» per l'attributo *maxOccurs* specifica l'assenza di un limite al numero di occorrenze dell'elemento.

## Schemi XML estendibili

La definizione di un elemento di tipo `xs:any` nello schema XSD valida un elemento di tipo qualsiasi nel documento XML corrispondente (in particolare se viene specificato l'indicatore `minOccurs="0"` l'elemento diviene, oltre che generico, facoltativo).

Il tipo `xs:any` viene utilizzato per creare schemi XML flessibili che validano documenti con elementi non previsti; in modo analogo per gli attributi si ricorre a `xs:anyAttribute`.

## ESEMPIO

Data la seguente definizione dell'elemento complesso *persona*

```

<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="cognome" type="xs:string"/>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="figlio" type="xs:string"
        minOccurs="0" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

sono validi i seguenti elementi XML

```

<persona>
  <cognome>Meini</cognome>
  <nome>Giorgio</nome>
</persona>

```

```

<persona>
  <cognome>Meini</cognome>
  <nome>Giorgio</nome>
  <figlio>Federico</figlio>
  <figlio>Alessandro</figlio>
</persona>

```

5. Base-64 è una codifica che impiega un sottoinsieme di 64 caratteri ASCII per realizzare un sistema di numerazione in base 64 spesso utilizzato per la rappresentazione testuale del contenuto di file binari.

## 2.3 Tipi di dato predefiniti

La TABELLA 5 riepiloga i tipi di dato predefiniti utilizzabili in uno schema XSD.

TABELLA 5

xs:string	Stringhe di caratteri
xs:normalizedString	Stringhe di caratteri non suddivise su più linee e prive di tabulazioni
xs:token	Stringhe di caratteri non suddivise su più linee, prive di tabulazioni, di spazi iniziali e finali e di spazi ripetuti
xs:float	Valori numerici <i>floating-point</i> a 32 bit
xs:double	Valori numerici <i>floating-point</i> a 64 bit
xs:decimal	Valori numerici non necessariamente interi
xs:byte xs:unsignedByte	Valori numerici rappresentabili con 8 bit, rispettivamente con segno e senza segno
xs:short xs:unsignedShort	Valori numerici rappresentabili con 16 bit, rispettivamente con segno e senza segno
xs:int xs:unsignedInt	Valori numerici rappresentabili con 32 bit, rispettivamente con segno e senza segno
xs:long xs:unsignedLong	Valori numerici rappresentabili con 64 bit, rispettivamente con segno e senza segno
xs:integer	Valori numerici interi
xs:nonNegativeInteger	Valori numerici interi non negativi (compreso lo 0)
xs:negativeInteger	Valori numerici interi negativi
xs:nonPositiveInteger	Valori numerici interi non positivi (compreso lo 0)
xs:positiveInteger	Valori numerici interi positivi
xs:date	Data nel formato aaaa-mm-gg (a: anno, m: mese, g: giorno); un carattere «Z» finale opzionale indica che si tratta della data UTC
xs:time	Ora nel formato oo:mm:ss (o: ora, m: minuti, s: secondi); i secondi possono assumere un valore non intero; un carattere «Z» finale opzionale indica che si tratta dell'ora UTC
xs:dateTime	Data/ora nel formato aaaa-mm-ggT <sup>oo</sup> :mm:ss (a: anno, m: mese, g: giorno, o: ora, m: minuti, s: secondi); i secondi possono assumere un valore non intero; un carattere «Z» finale opzionale indica che si tratta dell'ora UTC
xs:duration	Durata temporale nel formato <div>PaYmMgDT<sup>o</sup>hMmSs</div> (a: anni, m: mesi, g: giorni, o: ore, m: minuti, s: secondi; i secondi possono assumere un valore non intero); le varie sezioni sono opzionali, il carattere «P» inizia obbligatoriamente la rappresentazione del periodo, il carattere «T» inizia obbligatoriamente la parte oraria del periodo
xs:boolean	Valori <i>true/false</i>
xs:hexBinary	Sequenze di byte codificate in esadecimale
xs:base64Binary	Sequenze di byte codificate in base-64 <sup>5</sup>
xs:anyURI	Stringa che rappresenta un URI ( <i>Uniform Resource Identifier</i> )

**OSSERVAZIONI** Relativamente ai tipi di data e ora, la data/ora UTC (*Coordinated Universal Time*) è la data ora riferita al meridiano di Greenwich (Londra).

Ai tipi di data, ora e durata è possibile applicare le restrizioni per i valori enumerati, minimi e massimi.

**ESEMPIO**

Data la seguente definizione dell'elemento complesso *viaggio*

```
<xs:element name="viaggio">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="mezzo">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="auto"/>
            <xs:enumeration value="treno"/>
            <xs:enumeration value="aereo"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="partenza" type="xs:dateTime"/>
      <xs:element name="luogoPartenza" type="xs:string"/>
      <xs:element name="arrivo" type="xs:dateTime"/>
      <xs:element name="luogoArrivo" type="xs:string"/>
      <xs:element name="durata" type="xs:duration"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

è valido il seguente elemento XML

```
<viaggio>
  <mezzo>treno</mezzo>
  <partenza>2012-10-21T10:35:10</partenza>
  <luogoPartenza>Livorno</luogoPartenza>
  <arrivo>2012-10-21T14:40:30</arrivo>
  <luogoArrivo>Milano</luogoArrivo>
  <durata>PT4H5M20S</durata>
</viaggio>
```

**ESEMPIO**

Data la seguente definizione dell'elemento complesso *riferimenti*

```
<xs:element name="riferimenti">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="riferimento" type="xs:anyURI"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

risultano validi i seguenti elementi XML

```
<riferimenti>
  <riferimento>http://www.w3schools.com</riferimento>
  <riferimento>http://www.w3.org/2001/03/webdata/xsv</riferimento>
</riferimenti>

<riferimenti>
</riferimenti>
```

## 2.4 Un esempio completo

Il percorso di un veicolo sulla superficie terrestre è una lista ordinata di posizioni geografiche – latitudine e longitudine espresse come gradi, minuti e secondi – associata alla data/ora in cui il veicolo le ha attraversate.

Il seguente schema XSD, memorizzato nel file «percorso.xsd», definisce la struttura di un documento XML adatto a memorizzare il percorso di un veicolo:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- definizione del tipo latitudine -->
  <xs:complexType name="tipo_latitudine">
    <xs:sequence>
      <xs:element name="gradi">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxExclusive value="90"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="minuti">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxExclusive value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="secondi">
        <xs:simpleType>
          <xs:restriction base="xs:decimal">
            <xs:minInclusive value="0"/>
            <xs:maxExclusive value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="orientamento">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Nord"/>
            <xs:enumeration value="Sud"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```



```

<!-- definizione del tipo latitudine -->
<xs:complexType name="tipo_longitudine">
  <xs:sequence>
    <xs:element name="gradi">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxExclusive value="180"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="minuti">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxExclusive value="60"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="secondi">
      <xs:simpleType>
        <xs:restriction base="xs:decimal">
          <xs:minInclusive value="0"/>
          <xs:maxExclusive value="60"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="orientamento">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Est"/>
          <xs:enumeration value="Ovest"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- definizione del tipo posizione -->
<xs:complexType name="tipo_posizione">
  <xs:sequence>
    <xs:element name="latitudine" type="tipo_latitudine"/>
    <xs:element name="longitudine" type="tipo_longitudine"/>
    <xs:element name="dataOra" type="xs:dateTime"/>
  </xs:sequence>
</xs:complexType>
<!-- definizione dello schema percorso -->
<xs:element name="percorso">
  <xs:complexType>
    <xs:sequence>

```



```

    <xs:element name="posizione" type="tipo_posizione"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

**OSSERVAZIONE** L'elemento radice *percorso* è composto da una lista potenzialmente infinita di elementi *posizione* di tipo *tipo\_posizione*. Il tipo complesso *tipo\_posizione* è costituito da un elemento *latitudine* di tipo *tipo\_latitudine*, da un elemento *longitudine* di tipo *tipo\_longitudine* e da un elemento *dataOra* di tipo predefinito *dateTime*. Entrambi i tipi complessi *tipo\_latitudine* e *tipo\_longitudine* sono a loro volta costituiti dagli elementi *gradi*, *minuti*, *secondi* e *orientamento* definiti come opportune restrizioni di tipi predefiniti. Si noti come il ricorso alla definizione di tipi renda – nonostante la verbosità tipica di XSD – lo schema leggibile.

## ESEMPIO

Il seguente documento XML risulta valido rispetto allo schema precedente:

```

<?xml version="1.0" ?>
<percorso
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="percorso.xsd">
  <posizione>
    <latitudine>
      <gradi>43</gradi>
      <minuti>33</minuti>
      <secondi>0</secondi>
      <orientamento>Nord</orientamento>
    </latitudine>
    <longitudine>
      <gradi>10</gradi>
      <minuti>19</minuti>
      <secondi>0</secondi>
      <orientamento>Est</orientamento>
    </longitudine>
    <dataOra>2012-10-21T17:29:59</dataOra>
  </posizione>
  <posizione>
    <latitudine>
      <gradi>43</gradi>
      <minuti>43</minuti>
      <secondi>0</secondi>
      <orientamento>Nord</orientamento>
    </latitudine>
    <longitudine>
      <gradi>10</gradi>
      <minuti>24</minuti>
      <secondi>0</secondi>
      <orientamento>Est</orientamento>
    </longitudine>
    <dataOra>2012-10-21T17:59:09</dataOra>
  </posizione>
</percorso>

```



### 3 Riferimento ai nodi di un albero XML con XPath

#### Valutazione di espressioni XPath con Notepad++

Tra le funzionalità del *plug-in XML tools* dell'editor per sviluppatori Notepad++ ([notepad-plus-plus.org](http://notepad-plus-plus.org)) vi è la possibilità di valutare espressioni XPath su un documento XML aperto.

*XPath* è un linguaggio specifico per la «navigazione» di documenti XML; dato che un qualsiasi documento XML può essere rappresentato come un albero è stata definita una notazione standard per riferirne singoli nodi, o sottoinsiemi di nodi: le espressioni del linguaggio *XPath* hanno esattamente questo scopo.

**OSSERVAZIONE** Un nodo di un albero che rappresenta un documento XML può corrispondere a un elemento (semplice o complesso), a un attributo, a una stringa di testo o a un commento.

#### ESEMPIO

Dato il documento XML dell'esempio iniziale

```
<?xml version="1.0" ?>
<users>
  <!-- Nota: le password sono cifrate. -->
  <user>
    <username>Pippo</username>
    <email>pippo32@disney.com</email>
    <password>0A1B2C3D4E5F</password>
  </user>
  <user>
    <username>Pluto</username>
    <email>pluto30@disney.com</email>
    <password>A9B8C7D6E5F4</password>
  </user>
</users>
```

L'espressione XPath

`/users/user/username`

individua gli elementi evidenziati che hanno rispettivamente per contenuto le stringhe «Pippo» e «Pluto». Nella rappresentazione grafica dell'albero del documento (FIGURA 3) si osserva che l'espressione definisce il percorso che dal nodo radice congiunge i nodi individuati.

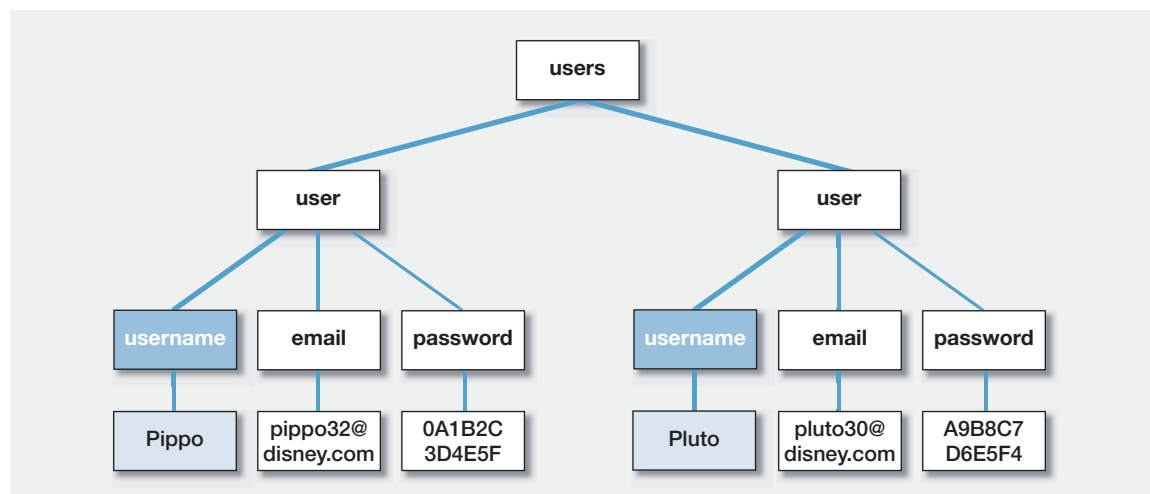


FIGURA 3

► Un'espressione *XPath* – nella sua forma più semplice – è una sequenza di nomi di nodi, ciascuno separato dall'altro dal simbolo «/», che determina uno o più percorsi nell'albero che rappresenta il documento XML.

**OSSERVAZIONE** Una sequenza di nomi di nodi determina più percorsi solo nel caso che esistano nodi distinti aventi lo stesso nome.

Nel linguaggio *XPath* esiste il concetto di «nodo corrente» di un documento XML: si tratta del nodo preso in considerazione in un determinato istante della navigazione dell'albero che rappresenta il documento. La **TABELLA 6** riporta le espressioni fondamentali del linguaggio *Xpath*.

**TABELLA 6**

<i>nodo</i>	Individua i nodi che hanno come nome <i>nodo</i>
/	Indica che il percorso ha inizio dal nodo radice del documento (percorso assoluto)
//	Individua tutti i nodi che nel documento hanno lo stesso nome indipendentemente dalla posizione
.	Indica il nodo corrente
..	Indica il padre del nodo corrente
@	Individua un attributo
*	Individua tutti i nodi elemento
@*	Individua tutti i nodi attributo

**ESEMPIO**

Dato il seguente documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libro genere="fantascienza" xml:lang="it">
  <autore>Stefano Benni</autore>
  <titolo>Terra!</titolo>
  <editore>Feltrinelli</editore>
  <anno>1983</anno>
</libro>
```

esso è rappresentato dal diagramma ad albero di **FIGURA 4**. Le espressioni *XPath* che seguono individuano singoli nodi con i relativi valori:

/libro/titolo	<i>titolo</i>	«Terra!»
/libro/@genere	<i>genere</i>	«fantascienza»
/libro/autore/..anno	<i>anno</i>	«1983»
//editore	<i>editore</i>	«Feltrinelli»

Le espressioni *XPath* che seguono individuano insiemi di nodi con i relativi valori:

/libro/\*



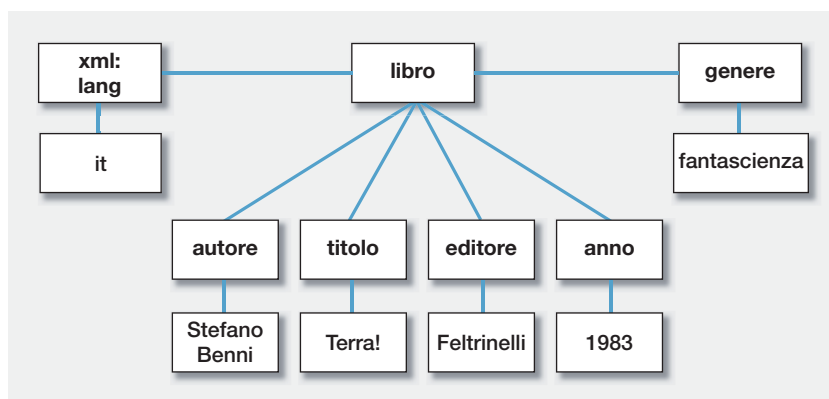


FIGURA 4

```

autore          «Stefano Benni»
titolo          «Terra!»
editore         «Feltrinelli»
anno            «1983»

/libro/@*

genere          «fantascienza»
xml:lang        «it»
  
```

**OSSERVAZIONE** L'espressione *XPath* `//*` seleziona tutti i nodi elemento di un documento XML.

In una espressione *Xpath* è possibile specificare, dopo il nome di un nodo, un **predicato** compreso tra i simboli «`[`» e «`]`»: un predicato è una condizione di selezione dei nodi.

La TABELLA 7 riporta gli operatori relazionali e logici utilizzabili per specificare un predicato di un'espressione del linguaggio *Xpath*.

TABELLA 7

=	Uguale
!=	Diverso
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
or	Or logico
and	And logico

Dato il seguente documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libri>
  <libro genere="fantascienza">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <anno>1983</anno>
  </libro>
  <libro genere="romanzo">
    <autore>Mark Twain</autore>
    <titolo>Le avventure di Huckleberry Finn</titolo>
    <anno>1884</anno>
  </libro>
  <libro genere="romanzo">
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <anno>1851</anno>
  </libro>
</libri>
```

le espressioni *XPath* che seguono individuano insiemi di nodi con i relativi valori:

```
/libri/libro[@genere="romanzo"]/titolo
```

titolo	«Le avventure di Huckleberry Finn»
titolo	«Moby Dick»

```
/libri/libro[anno>1900]/titolo
```

titolo	«Terra!»
--------	----------

Il linguaggio *XPath* comprende un'ampia libreria di funzioni predefinite che operano sugli elementi testuali<sup>6</sup> e che possono essere utilizzate nella definizione dei predicati; la TABELLA 8 riepiloga le funzioni principali.

TABELLA 8

concat	Concatena 2 stringhe di caratteri
contains	Verifica se una stringa di caratteri è sottostringa di un'altra
start-with	Verifica se una stringa di caratteri inizia con una sottostringa specificata
string-length	Restituisce la lunghezza di una stringa di caratteri
normalize-space	Normalizza gli spazi di una stringa di caratteri (elimina gli spazi iniziali e finali e gli spazi ripetuti)
not	Nega un'espressione booleana
round	Approssima un valore numerico al valore intero
floor	Tronca un valore numerico al valore intero

6. Gli elementi testuali di un documento XML possono rappresentare valori numerici: questo aspetto può essere imposto dalla validazione rispetto a uno schema XSD.

## ESEMPIO

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libri>
  <libro genere="fantascienza">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <prezzo>10.50</prezzo>
  </libro>
  <libro genere="romanzo">
    <autore>Mark Twain</autore>
    <titolo>Le avventure di Huckleberry Finn</titolo>
    <prezzo>9.90</prezzo>
  </libro>
  <libro genere="romanzo">
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <prezzo>15.00</prezzo>
  </libro>
</libri>
```

```
/libri/libro[floor(prezzo*1.1)<15]/titolo
```

```
/libri/libro[contains(autore, "Melville")]/titolo
```

titolo	«Moby Dick»
--------	-------------

## ESEMPIO

```
/libri/libro[1]/titolo      titolo      «Terra!»
/libri/libro[last()]/titolo titolo      «Moby Dick»
```

212

L'operatore «|» del linguaggio *XPath* consente di unire il risultato di espressioni distinte.

ESEMPIO

L'espressione *XPath*

```
/libri/libro[@genere="fantascienza"]/autore |  
/libri/libro[@genere="romanzo"]/autore
```

riferita al documento XML dell'esempio precedente individua i seguenti nodi con i relativi valori:

autore	«Stefano Benni»
autore	«Mark Twain»
autore	«Herman Melville»

Gli **assi** del linguaggio *XPath* sono sottoinsiemi dei nodi dell'albero che rappresenta il documento XML specificati rispetto al nodo corrente (TABELLA 9).

TABELLA 9

ancestor	Seleziona tutti i nodi antenati del nodo corrente
ancestor-or-self	Seleziona tutti i nodi antenati del nodo corrente, compreso il nodo corrente stesso
attribute	Seleziona gli attributi del nodo corrente (è equivalente a «@»)
child	Seleziona tutti i nodi figli del nodo corrente (è l'asse predefinito e solitamente non viene indicato nelle espressioni)
descendant	Seleziona tutti i nodi discendenti del nodo corrente
descendant-or-self	Seleziona tutti i nodi discendenti del nodo corrente, compreso il nodo corrente stesso (è equivalente a «//»)
following	Seleziona tutti i nodi corrispondenti agli elementi che nel documento XML seguono l'elemento corrispondente al nodo corrente
following-sibling	Seleziona tutti i nodi fratelli successivi al nodo corrente
parent	Seleziona il nodo padre del nodo corrente (è equivalente a «.»)
preceding	Seleziona tutti i nodi corrispondenti agli elementi che nel documento XML precedono l'elemento corrispondente al nodo corrente
preceding-sibling	Seleziona tutti i nodi fratelli precedenti al nodo corrente
self	Seleziona il nodo corrente stesso (è equivalente a «.»)

► La definizione formale delle espressioni *XPath* prevede che ciascuna di esse sia composta da «passi» separati dal simbolo «/» e che ciascun passo sia costituito da tre componenti:

- la specificazione dell'asse seguita dal simbolo «:» (può essere omessa la specificazione dell'asse *child*);
- l'identificazione di uno o più nodi dell'asse (normalmente espressa mediante il nome del/dei nodo/i);
- un predicato (opzionale).

La seconda componente di un passo può essere espressa utilizzando le funzioni di test dei nodi indicate nella TABELLA 10.

TABELLA 10

node()	Identifica qualsiasi nodo
comment()	Identifica i commenti
text()	Identifica i nodi testuali

ESEMPIO

Con riferimento al documento XML dell'esempio introduttivo del capitolo

```
<?xml version="1.0" ?>
<users>
  <!-- Nota: le password sono cifrate. -->
  <user>
    <username>Pippo</username>
    <email>pippo32@disney.com</email>
    <password>0A1B2C3D4E5F</password>
  </user>
  <user>
    <username>Pluto</username>
    <email>pluto30@disney.com</email>
    <password>A9B8C7D6E5F4</password>
  </user>
</users>
```

l'espressione *XPath* che identifica tutti i nodi *username* nella sua forma estesa è formulata come segue

```
/child::users/child::user/child::username/self::node()
```

oppure

```
/descendant::username//self::node()
```

ESEMPIO

Con riferimento al documento XML di un esempio precedente

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libri>
  <libro genere="fantascienza">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <anno>1983</anno>
  </libro>
  <libro genere="romanzo">
    <autore>Mark Twain</autore>
    <titolo>Le avventure di Huckleberry Finn</titolo>
    <anno>1884</anno>
  </libro>
  <libro genere="romanzo">
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <anno>1851</anno>
  </libro>
</libri>
```

le espressioni *XPath* che seguono individuano insiemi di nodi con i relativi valori:

```
/libri/libro[last()]/preceding-sibling::libro/titolo  
titolo          «Terra!»  
titolo          «Le avventure di Huckleberry Finn»  
  
/libri/libro[titolo="Moby Dick"]/self::node()  
               attribute::genere  
genere          «romanzo»  
  
/libri/libro[1]/following::titolo  
titolo          «Le avventure di Huckleberry Finn»  
titolo          «Moby Dick»
```

## Sintesi

■ **XML (*Extensible Markup Language*).** XML è un metalinguaggio per la definizione di linguaggi basati su tag marcatori; la versatilità di XML lo ha reso uno strumento utilizzato principalmente per definire documenti con marcatori in cui i tag non sono predefiniti e collezioni di dati semistrutturati rappresentate in formato testuale.

■ **Struttura ad albero di un documento XML.** Dato che le regole per la nidificazione dei tag del linguaggio sono rigorosamente gerarchiche, i documenti in formato XML hanno naturalmente una struttura ad albero che viene utilizzata da molti strumenti che ne consentono la gestione.

■ **Elementi XML.** Un elemento di un documento XML è quanto è compreso tra un tag di apertura e il corrispondente tag di chiusura: un elemento può contenere testo, attributi e altri elementi, oppure essere vuoto.

■ **Documenti XML ben formati.** Un documento XML che rispetta le regole sintattiche di base del linguaggio è definito «ben formato».

■ **XSD (*XML Schema Definition*).** Uno schema XSD consente di definire la struttura e i tipi degli elementi che un documento XML deve rispettare per essere considerato valido. La validazione di un documento XML rispetto a uno schema XSD viene effettuata mediante strumenti software automatici; ogni documento XML dovrebbe contenere un riferimento allo schema XSD che lo definisce.

■ **Tipi XSD predefiniti.** In uno schema XSD è possibile definire il tipo del contenuto testuale di un elemento. I tipi predefiniti fondamentali sono le stringhe di caratteri, i valori numerici generali, i valori numerici interi, le date e gli orari; a questi tipi fondamentali è possibile imporre delle restrizioni, per esempio una limitazione del campo di validità per i tipi numerici.

■ **Tipi XSD derivati.** In uno schema XSD è possibile definire nuovi tipi a partire dai tipi fondamentali, dalle loro restrizioni e dalle loro combinazioni; gli elementi di un documento XML possono essere validati rispetto a questi tipi derivati.

■ **Elementi XML complessi.** Un elemento XML complesso è normalmente un elemento che contiene attributi e/o altri elementi: gli indicatori XSD consentono di definire un elemento complesso come sequenza di più elementi, anche complessi.

■ **Espressioni XPath.** *XPath* è un linguaggio per la navigazione dei documenti XML; le espressioni *XPath* riferiscono singoli nodi, o sottoinsiemi di nodi, dell'albero che rappresenta un documento XML. Le espressioni *XPath* più semplici sono simili ai *pathname* che individuano un file, un insieme di file o una directory nella struttura gerarchica del file-system.

■ **Predicati XPath.** La possibilità di impiegare predicati nelle espressioni *XPath* permette di selezionare nodi o sottoinsiemi di nodi dell'albero che rappresenta un documento XML che rispettano specifici criteri imposti al contenuto informativo.

## QUESITI

### 1 XML è ...

- A ... un linguaggio con tag marcatori alternativo rispetto a HTML.
- B ... un metalinguaggio per la definizione di linguaggi con tag marcatori.
- C ... un metalinguaggio da cui è stato derivato HTML.
- D Nessuna delle risposte precedenti è corretta.

### 2 XML viene utilizzato ...

- A ... per la definizione del formato di visualizzazione di dati non strutturati.
- B ... per la memorizzazione e l'interscambio di dati.
- C ... per la definizione di documenti con marcatori i cui tag non sono predefiniti.
- D Nessuna delle risposte precedenti è corretta.

### 3 I tag di un documento XML ...

- A ... possono essere aperti e non chiusi.
- B ... devono essere nidificati in modo gerarchico.
- C ... devono necessariamente prevedere uno o più attributi.
- D Nessuna delle risposte precedenti è corretta.

### 4 Indica che cosa può contenere un elemento XML.

- A Valori testuali.
- B Uno o più attributi.
- C Altri elementi.
- D Commenti.

### 5 Un documento XML può essere rappresentato mediante ...

- A ... un albero.
- B ... una lista.
- C ... una tabella.
- D Nessuna delle risposte precedenti è corretta.

### 6 Quale scopo ha la specificazione dell'«encoding» di un documento XML?

- A Consentire la corretta interpretazione dei caratteri che costituiscono il documento.
- B Specificare il metodo di compressione del documento.
- C Indicare il metodo di cifratura del documento.
- D Nessuna delle risposte precedenti è corretta.

### 7 Per quale motivo si usa un elemento di tipo CDATA in un documento XML?

- A Per evitare che il contenuto dell'elemento sia analizzato.
- B Per forzare l'interpretazione del contenuto dell'elemento.
- C Per inserire come contenuto dell'elemento una struttura dati definita in linguaggio C.
- D Nessuna delle risposte precedenti è corretta.

### 8 Un documento XML «ben formato» è ...

- A ... un documento XML che rispetta le regole sintattiche del linguaggio.
- B ... un documento XML validato rispetto a uno schema XSD.
- C ... un documento XML validato rispetto a uno schema pubblicato sul web.
- D Nessuna delle risposte precedenti è corretta.

### 9 Un documento XML valido è ...

- A ... un documento XML che rispetta le regole sintattiche del linguaggio.
- B ... un documento XML validato rispetto a uno schema XSD.
- C ... un documento XML utilizzato da più di un'applicazione software.
- D Nessuna delle risposte precedenti è corretta.

### 10 XSD è ...

- A ... un linguaggio alternativo a XML.
- B ... un linguaggio per la definizione di schemi per la validazione di documenti XML.
- C ... un linguaggio per la trasformazione e visualizzazione di documenti XML.
- D Nessuna delle risposte precedenti è corretta.

**11** Uno schema XSD definisce il tipo ...

- A ... di tutti gli elementi testuali di un documento XML.
- B ... dei soli valori degli attributi di un documento XML.
- C ... dei soli elementi testuali con formato numerico di un documento XML.
- D Nessuna delle risposte precedenti è corretta.

**12** La restrizione del tipo di un elemento XML ...

- A ... pone ulteriori condizioni alla validità di un elemento testuale di un determinato tipo.
- B ... è applicabile esclusivamente a elementi testuali di tipo numerico.
- C ... è applicabile esclusivamente a elementi testuali di tipo stringa di caratteri.
- D Nessuna delle risposte precedenti è corretta.

**13** Un elemento XML complesso ...

- A ... è un elemento XML che contiene esclusivamente testo.
- B ... è un elemento XML che può contenere altri elementi.
- C ... è un elemento XML che contiene necessariamente attributi.
- D Nessuna delle risposte precedenti è corretta.

**14** L'indicatore `<xs:sequence>` in uno schema XSD ...

- A ... consente di definire un elemento complesso con più elementi nell'ordine specificato.
- B ... consente di definire un elemento semplice il cui contenuto è di tipo sequenziale.
- C ... consente di definire un elemento complesso con più elementi in cui l'ordine non ha importanza.
- D Nessuna delle risposte precedenti è corretta.

**15** Gli attributi *minOccurs* e *maxOccurs* ...

- A ... consentono di definire il numero minimo e massimo di ripetizioni di un elemento.
- B ... consentono di definire il numero minimo e massimo di caratteri del contenuto testuale di un elemento.
- C ... consentono di definire il numero minimo e massimo di elementi contenuti in un elemento complesso.
- D Nessuna delle risposte precedenti è corretta.

**16** Dovendo definire un elemento che specifica una data ...

- A ... è necessario definire un elemento complesso che contiene gli elementi *giorno*, *mese* e *anno*.
- B ... è sufficiente definire un elemento semplice del tipo predefinito *date*.
- C ... è necessario definire un elemento complesso del tipo predefinito *date*.
- D Nessuna delle risposte precedenti è corretta.

**17** Le espressioni del linguaggio *XPath* ...

- A ... riferiscono esclusivamente singoli nodi dell'albero che rappresenta un documento XML.
- B ... riferiscono esclusivamente nodi foglie dell'albero che rappresenta un documento XML.
- C ... riferiscono singoli nodi, o insiemi di nodi dell'albero che rappresenta un documento XML.
- D Nessuna delle risposte precedenti è corretta.

**18** Un predicato del linguaggio *XPath* ...

- A ... seleziona sempre un singolo nodo dell'albero che rappresenta un documento XML.
- B ... seleziona esclusivamente uno o più nodi foglie dell'albero che rappresenta un documento XML.
- C ... seleziona uno o più nodi dell'albero che rappresenta un documento XML.
- D Nessuna delle risposte precedenti è corretta.



## ESERCIZI

- 1** Indicare quali dei seguenti documenti XML sono «ben formati»; in caso contrario indicare il motivo per cui non lo sono.

a) `<?xml version="1.0" ?>`

```
<A>
  <B>...</B>
  <C>...</C>
</A>
<A>
  ...
</A>
```

b) `<?xml version="1.0" ?>`

```
<A>
  <B>...</C>
  <C>...</B>
</A>
```

c) `<?xml version="1.0" ?>`


```
<A>
  <B>...</b>
  <c>...</C>
</A>
```


d) `<?xml version="1.0" ?>`

```
<A x=123>
  <B>...</B>
  <C>...</C>
</A>
```

e) `<?xml version="1.0" ?>`

```
<A y="123">
  <B>...</C>
  <C>...</B>
</A>
```

- 2**  Mediante uno schema XSD progettare un linguaggio XML che consenta di rappresentare una rubrica dei riferimenti a persone (nome, cognome, indirizzo, uno o più numeri di telefono, uno o più indirizzi di posta elettronica, sito web). Produrre un documento XML valido di esempio.

- 3**  Mediante uno schema XSD progettare un linguaggio XML che consenta di rappresentare un prodotto presente nel magazzino di un sito web di commercio di dispositivi elettronici (categoria a scelta tra: audio, video, telefonia e informatica; codice, marca, modello, descrizione, prezzo,


quantità disponibile). Produrre più documenti XML validi di esempio.


- 4** Con riferimento ai prodotti descritti nell'esercizio 3 progettare, mediante uno schema XSD, un linguaggio XML che permetta di rappresentare il contenuto di un «carrello» del sito web di commercio; il «carrello» deve comprendere i dati dell'acquirente coerentemente con quanto specificato nell'esercizio 2. Produrre un documento XML valido di esempio.

- 5** Mediante uno schema XSD progettare un linguaggio XML per la creazione di vocabolari intesi come elenchi di termini e in cui per ogni termine siano specificati uno o più significati. Produrre un documento XML valido di esempio.

- 6** Mediante uno schema XSD progettare un linguaggio XML per la rappresentazione di un giornale composto da articoli; ogni articolo ha un titolo, un sottotitolo, un'intestazione, un autore, una data e un corpo composto da uno o più paragrafi di testo. Produrre un documento XML valido di esempio.

- 7** Mediante uno schema XSD progettare un linguaggio XML per la realizzazione di registri dell'insegnante in cui per ogni studente sono riportati il nome, il cognome, la classe, le date dei giorni di assenza, le valutazioni comprendenti la data, la tipologia (scritta, orale, pratica, test), il punteggio e il voto decimale. Produrre un documento XML valido di esempio.

- 8**  Mediante uno schema XSD progettare un linguaggio XML per la definizione di prove di verifica strutturate composte da quesiti a scelta multipla; per ogni quesito è necessario specificare la richiesta, il punteggio e le possibili risposte. Produrre un documento XML valido di esempio.

- 9**  Progettare mediante uno schema XSD un linguaggio XML per la registrazione dei dati meteorologici (temperatura, pressione, umidità relativa, velocità e direzione del vento) rilevati da una stazione mobile; per ogni registrazione è necessario specificare la posizione (latitudine/longitudine) fornita da un dispositivo GPS interno alla stazio-

ne e la data/ora della misura, deve inoltre essere specificata come attributo l'unità di misura impiegata dallo strumento. Produrre un documento XML valido di esempio.

## 10 Facendo riferimento al seguente documento



XML

```
<?xml version="1.0" ?>
<prodotti>
  <prodotto punti="10">
    <codice>1234567890</codice>
    <descrizione>
      biscotti al burro
    </descrizione>
    <prezzo>5.50</prezzo>
  </prodotto>
  <prodotto punti="0">
    <codice>0123456789</codice>
    <descrizione>
      birra 3/4 litro
    </descrizione>
    <prezzo>4.00</prezzo>
  </prodotto>
  <prodotto punti="20">
    <codice>9876543210</codice>
    <descrizione>
      spaghetti 1/2 Kg
    </descrizione>
```

```
    <prezzo>2.50</prezzo>
  </prodotto>
  <prodotto punti="5">
    <codice>0987654321</codice>
    <descrizione>
      salame confezionato
    </descrizione>
    <prezzo>3.50</prezzo>
  </prodotto>
</prodotti>
```

Scrivere le espressioni *XPath* che selezionano i seguenti elementi:

- descrizione dei prodotti che hanno il prezzo inferiore a 4 €;
- prezzi di tutti i prodotti;
- descrizione del secondo prodotto dell'elenco;
- codice del penultimo prodotto dell'elenco;
- codice della birra;
- prezzo degli spaghetti scontato del 10%;
- descrizione del prodotto con codice 0123456789;
- descrizione del prodotto successivo nell'elenco ai biscotti;
- codici dei prodotti con prezzo superiore a 5 € o inferiore a 3 €;
- punti del prodotto con codice 9876543210;
- codici dei prodotti con punti superiori a 5;
- descrizione dei prodotti con punti uguali a 0.

**at the mercy**

alla mercé

**burial site**

luogo di sepoltura

**bothered**

preoccupato

**starkly**

aspro

**to trip the unwary**

far scappare gli incauti

**encumbered**

gravato

**brokerage houses**

agenzie di intermediazione

## 1 An Eagle's eye view of XML

This chapter introduces you to XML, the Extensible Markup Language. It explains, in general terms, what XML is and how it is used. It shows you how different XML technologies work together, and how to create an XML document and deliver it to readers.

### What Is XML?

XML stands for Extensible Markup Language (often miscapitalized as *eXtensible Markup Language* to justify the acronym). XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document.

It is a meta-markup language that defines a syntax in which other domain-specific markup languages can be written.

### XML is a meta-markup language

The first thing you need to understand about XML is that it isn't just another markup language like HTML, TeX, or troff. These languages define a fixed set of tags that describe a fixed number of elements. If the markup language you use doesn't contain the tag you need, you're out of luck. You can wait for the next version of the markup language, hoping that it includes the tag you need; but then you're really at the mercy of whatever the vendor chooses to include.

XML, however, is a meta-markup language. It's a language that lets you make up the tags you need as you go along. These tags must be organized according to certain general principles, but they're quite flexible in their meaning. For example, if you're working on genealogy and need to describe family names, personal names, dates, births, adoptions, deaths, burial sites, marriages, divorces, and so on, you can create tags for each of these. You don't have to force your data to fit into paragraphs, list items, table cells, or other very general categories.

You can document the tags you create in a schema written in any of several languages, including document type definitions (DTDs) and the W3C XML Schema Language. [...]. For now, think of a schema as a vocabulary and a syntax for certain kinds of documents.

For example, the schema for Peter Murray-Rust's Chemical Markup Language (CML) is a DTD that describes a vocabulary and a syntax for the molecular sciences: chemistry, crystallography, solid-state physics, and the like. It includes tags for atoms, molecules, bonds, spectra, and so on. Many different people in the field can share this schema. Other schemas are available for other fields, and you can create your own.

XML defines the meta syntax that domain-specific markup languages such as MusicXML, MathML, and CML must follow. It specifies the rules for the low-level syntax, saying how markup is distinguished from content, how attributes are attached to elements, and so forth, without saying what these tags, elements, and attributes are or what they mean. It gives the patterns that elements must follow without specifying the names of the elements. For example, XML says that tags begin with a < and end with a >. However, XML does not tell you what names must go between the < and the >.

If an application understands this meta syntax, it at least partially understands all the languages built from this meta syntax. A browser does not need to know in advance each and every tag that might be used by thousands of different markup languages. Instead, the browser discovers the tags used by any given document as it reads the document or its schema. The detailed instructions about how to display the content of these tags are provided in a separate style sheet that is attached to the document.

[...]

XML means you don't have to wait for browser vendors to catch up with your ideas.

You can invent the tags you need, when you need them, and tell the browsers how to display these tags.

### XML describes structure and semantics, not formatting

XML markup describes a document's structure and meaning. It does not describe the formatting of the elements on the page. You can add formatting to a document with a style sheet. The

document itself only contains tags that say what is in the document, not what the document looks like.

By contrast, HTML encompasses formatting, structural, and semantic markup. `<B>` is a formatting tag that makes its content bold. `<STRONG>` is a semantic tag that means its contents are especially important. `<TD>` is a structural tag that indicates that the contents are a cell in a table. In fact, some tags can have all three kinds of meaning. An `<H1>` tag can simultaneously mean 20-point Helvetica bold, a level 1 heading, and the title of the page.

For example, in HTML, a song might be described using a definition title, definition data, an unordered list, and list items. But none of these elements actually have anything to do with music. The HTML might look something like this:

```
<DT> Hot Cop
<DD> by Jacques Morali, Henri Belolo, and Victor Willis
<UL>
<LI> Jacques Morali
<LI> PolyGram Records
<LI> 6:20
<LI> 1978
<LI> Village People
</UL>
```

In XML, the same data could be marked up like this:

```
<SONG>
  <TITLE>Hot Cop</TITLE>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

Instead of generic tags such as `<DT>` and `<LI>`, this example uses meaningful tags such as `<SONG>`, `<TITLE>`, `<COMPOSER>`, and `<YEAR>`. These tags didn't come from any preexisting standard or specification. I just made them up on the spot because they fit the information I was describing. Domain-specific tagging has a number of advantages, not the least of which is that it's easier for a human to read the source code to determine what the author intended.

XML markup also makes it easier for nonhuman automated computer software to locate all of the songs in the document. A computer program reading HTML can't tell more than that an element is a DT. It cannot determine whether that DT represents a song title, a definition, or some designer's favorite means of indenting text.

In fact, a single document might well contain DT elements with all three meanings. XML element names can be chosen such that they have extra meaning in additional contexts. For example, they might be the field names of a database. XML is far more flexible and amenable to varied uses than HTML because a limited number of tags don't have to serve many different purposes. XML offers an infinite number of tags to fill an infinite number of needs.

## Why Are Developers Excited About XML?

XML makes easy many web-development tasks that are extremely difficult with HTML, and it makes tasks that are impossible with HTML possible. Because XML is extensible, developers like it for many reasons. Which reasons most interest you depends on your individual needs, but once you learn XML, you're likely to discover that it's the solution to more than one problem you're already struggling with. This section investigates some of the generic uses of XML that excite developers. [...].

## Domain-specific markup languages

XML enables individual professions (for example, music, chemistry, human resources) to develop their own domain-specific markup languages. Domain-specific markup languages enable practitioners in the field to trade notes, data, and information without worrying about whether or not the person on the receiving end has the particular proprietary pay-ware that was used to create the data. They can even send documents to people outside the profession with a reasonable confidence that those who receive them will at least be able to view the documents.

Furthermore, creating separate markup languages for different domains does not lead to bloatware or unnecessary complexity for those outside the profession. You may not be interested in electrical engineering diagrams, but electrical engineers are.

You may not need to include sheet music in your web pages, but composers do. XML lets the electrical engineers describe their circuits and the composers notate their scores, mostly without stepping on each other's toes. Neither field needs special support from browser manufacturers or complicated plug-ins, as is true today.

## Self-describing data

Much computer data from the last 40 years is lost, not because of natural disaster or decaying backup media (though those are problems too, ones XML doesn't solve), but simply because no one bothered to document how the data formats. A Lotus 1-2-3 file on a 15-year-old 5.25-inch floppy disk might be irretrievable in most corporations today without a huge investment of time and resources. Data in a less-known binary format such as Lotus Jazz may be gone forever.

XML is, at a low level, an incredibly simple data format. It can be written in 100 percent pure ASCII or Unicode text, as well as in a few other well-defined formats. Text is reasonably resistant to corruption. The removal of bytes or even large sequences of bytes does not noticeably corrupt the remaining text. This starkly contrasts with many other formats, such as compressed data or serialized Java objects, in which the corruption or loss of even a single byte can render the rest of the file unreadable.

At a higher level, XML is self-describing. Suppose you're an information archaeologist in the twenty-third century and you encounter this chunk of XML code on an old floppy disk that has survived the ravages of time:

```
<PERSON ID="p1100" SEX="M">
  <NAME>
    <GIVEN>Judson</GIVEN>
    <SURNAME> McDaniel</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>21 Feb 1834</DATE>
  </BIRTH>
  <DEATH>
    <DATE>9 Dec 1905</DATE>
  </DEATH>
</PERSON>
```

Even if you're not familiar with XML, assuming you speak a reasonable facsimile of twentieth-century English, you've got a pretty good idea that this fragment describes a man named Judson McDaniel, who was born on February 21, 1834 and died on December 9, 1905. In fact, even with gaps in or corruption of the data, you could probably still extract most of this information. The same could not be said for a proprietary, binary spreadsheet or word-processor format.

Furthermore, XML is very well documented. The World Wide Web Consortium (W3C)'s XML specification and numerous books tell you exactly how to read XML data. There are no secrets waiting to trip the unwary.

## Interchange of data among applications

Because XML is nonproprietary and easy to read and write, it's an excellent format for the interchange of data among different applications. XML is not encumbered by copyright, patent, trade secret, or any other sort of intellectual property restrictions.

It has been designed to be extremely expressive and very well structured while at the same time being easy for both human beings and computer programs to read and write. Thus, it's an obvious choice for exchange languages.

One such format is the Open Financial Exchange 2.0 (OFX, <http://www.ofx.net/>). OFX is designed to let personal finance programs, such as Microsoft Money and Quicken, trade data. The data can be sent back and forth between programs and exchanged with banks, brokerage houses, credit card companies, and the like.

By choosing XML instead of a proprietary data format, you can use any tool that understands XML to work with your data. You can even use different tools for different purposes, one program to view and another to edit, for example. XML keeps you from getting locked into a particular program simply because that's what your data is already written in, or because that program's proprietary format is all your correspondent can accept.

For example, many publishers require submissions in Microsoft Word. This means that most authors have to use Word, even if they would rather use OpenOffice.org Writer or WordPerfect. This makes it extremely difficult for any other company to publish a competing word processor unless it can read and write Word files. To do so, the company's programmers must reverse-engineer the binary Word file format, which requires a significant investment of limited time and resources. Most other word processors have a limited ability to read and write Word files, but they generally lose track of graphics, macros, styles, revision marks, and other important features.

Word's document format is undocumented, proprietary, and constantly changing, and thus Word tends to end up winning by default, even when writers would prefer to use other, simpler programs. Word offers the option to save its documents in an XML application called WordML instead of its native binary file format. It is far easier to reverse-engineer an undocumented XML format than a binary format. In the future, Word files will much more easily be exchanged among people using different word processors.

## Structured data

XML is ideal for large and complex documents because the data is structured. You specify a vocabulary that defines the elements in the document, and you can specify the relations between elements. For example, if you're putting together a web page of sales contacts, you can require every contact to have a phone number and an e-mail address. If you're inputting data for a database, you can make sure that no fields are missing. You can even provide default values to be used when no data is available.

XML also provides a client-side include mechanism that integrates data from multiple sources and displays it as a single document. (In fact, it provides at least three different ways of doing this, a source of some confusion.) The data can even be rearranged on the fly. Parts of it can be shown or hidden depending on user actions. You'll find this extremely useful when you're working with large information repositories like relational databases.

[E. Harold, *XML 1.1 Bible*, 3<sup>rd</sup> edition, Wiley, 2004]

## QUESTIONS

- a** Is XML a markup language like HTML?
- b** How the browser format and show the content of a XML document?
- c** How can you develop your own domain-specific language with XML?
- d** Why XML document are human-readable without a specific program?
- e** Why XML is the preferred way to exchange data among different application?

# Gli strumenti per la gestione dei dati rappresentati in linguaggio XML

La tabella di un database relazionale è sempre rappresentabile mediante un documento XML avente la seguente struttura:

```
<?xml version="1.0" ?>
<tabella>
  <riga1>
    <campo1>...</campo1>
    <campo2>...</campo2>
    ...
  </riga1>
  <riga2>
    <campo1>...</campo1>
    <campo2>...</campo2>
    ...
  </riga2>
  ...
  <rigaN>
    <campo1>...</campo1>
    <campo2>...</campo2>
    ...
  </rigaN>
</tabella>
```

## ESEMPIO

La tabella

ISBN	titolo	lingua	anno	prezzo
123-456-789	Pinocchio	Italiano	2010	15.00
987-654-321	The Hobbit	Inglese	2001	20.00

può essere rappresentata con il seguente documento XML:

```
<?xml version="1.0" ?>
<libri>
  <libro>
    <ISBN>123-456-789</ISBN>
```

```

    <titolo>Pinocchio</titolo>
    <lingua>Italiano</lingua>
    <anno>2010</anno>
    <prezzo>15.00</prezzo>
  </libro>
  <libro>
    <ISBN>987-654-321</ISBN>
    <titolo>The Hobbit</titolo>
    <lingua>Inglese</lingua>
    <anno>2001</anno>
    <prezzo>20.00</prezzo>
  </libro>
</libri>

```

**OSSERVAZIONE** Uno schema XSD rappresenta l'aspetto intensionale dei dati, mentre un documento XML valido rispetto allo schema ne rappresenta l'aspetto estensionale. Lo schema XSD dell'esempio precedente è il seguente:

```

<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- definizione del tipo libro -->
  <xs:complexType name="tipo_libro">
    <xs:sequence>
      <xs:element name="ISBN" type="xs:string"/>
      <xs:element name="titolo" type="xs:string"/>
      <xs:element name="lingua" type="xs:string"/>
      <xs:element name="anno" type="xs:integer"/>
      <xs:element name="prezzo" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
  <!-- definizione dello schema libri -->
  <xs:element name="libri">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="libro" type="tipo_libro"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

I documenti XML costituiscono una generalizzazione dei database relazionali; infatti se una tabella può essere rappresentata mediante un documento XML, il viceversa non è vero, in quanto gli schemi XSD per i quali i documenti XML validi possono essere rappresentati mediante una tabella sono solo un sottoinsieme di tutti gli schemi possibili.



1. In alcuni casi i documenti XML sono memorizzati in database di tipo relazionale: la trasformazione avviene a opera del DBMS stesso, oppure mediante un'applicazione software specifica.

Negli ultimi anni alla gestione dei dati mediante database relazionali si affianca il ricorso a vere e proprie basi di dati XML, cioè collezioni persistenti di documenti XML non necessariamente memorizzate sotto forma di file testuali<sup>1</sup>.

**OSSERVAZIONE** Il motivo fondamentale per cui si adotta una base di dati XML è la facilità di trasferimento dei dati che questa soluzione permette rispetto a un database tradizionale: in un mondo di applicazioni ormai centrato sulla distribuzione delle risorse nella rete, questo aspetto diviene facilmente uno dei requisiti principali nella progettazione di un nuovo sistema software.

## 1 L'interrogazione di basi di dati XML con il linguaggio XQuery

Se si escludono aspetti meramente prestazionali relativi alla quantità di dati gestibili e alla relativa velocità di elaborazione, la principale limitazione di una base di dati costituita da documenti XML consisteva nell'assenza di un adeguato linguaggio di interrogazione funzionalmente analogo a SQL.

Il linguaggio *XQuery* è un'estensione del linguaggio *XPath* che consente di formulare vere e proprie «query» su uno o più documenti XML che eventualmente generano nuovo codice XML nella forma di elementi, o insiemi di elementi. Il linguaggio *XQuery* **non** ha una sintassi XML.

### BaseX

BaseX è un sistema client/server per la gestione di database XML e al tempo stesso un processore *XPath/XQuery* inizialmente sviluppato all'università di Costanza. Oggi questo sistema implementa la maggior parte delle raccomandazioni del W3C.

BaseX è un progetto software open-source la cui componente client è una GUI molto semplice da utilizzare e al tempo stesso estremamente espressiva nella visualizzazione dei dati, che possono essere rappresentati, oltre che in forma testuale, anche come tabella, come mappa, come albero ecc.

La GUI di BaseX consente di visualizzare in tempo reale il risultato dell'applicazione interattiva di un'espressione *XPath/XQuery* a un documento XML, o a una collezione di documenti XML.

### ESEMPIO

La valutazione della seguente espressione *XPath*

```
/libri/libro[lingua="Italiano"]/titolo
```

sul documento XML dell'esempio di apertura del capitolo produce come risultato il seguente elemento XML:

```
<titolo>Pinocchio</titolo>
```

**OSSERVAZIONE** Espressioni *XPath* con predicati possono selezionare i dati presenti in un documento XML in base a criteri arbitrariamente complessi e rappresentano quindi una forma di query. Per questo motivo un'espressione *XPath* valida è un comando *XQuery* valido; in altre parole il linguaggio *XPath* è un sottoinsieme del linguaggio *XQuery*.

Dato che una base di dati XML può comprendere più documenti XML, il singolo documento viene riferito («aperto» nella terminologia *XQuery*) con la seguente espressione:

```
doc ("...")
```

dove l'argomento è normalmente il nome del file che contiene il documento stesso.

ESEMPIO

Se il nome del file che contiene il documento XML è «libri.xml», l'espressione *XPath* dell'esempio precedente può essere così formulata:

doc("libri.xml")/libri/libro[lingua="Italiano"]/titolo

Le query del linguaggio *XQuery* consistono normalmente nella valutazione delle cosiddette espressioni FLWOR<sup>2</sup>, così denominate dalle iniziali delle clausole che le costituiscono (TABELLA 1).

2. Il termine FLWOR viene usualmente pronunciato come la parola *flower* della lingua inglese.

TABELLA 1

<b>for ... in ...</b>	Consente di legare a una variabile il risultato di ogni singola iterazione su un elemento – normalmente ripetuto – del documento XML che è tipicamente specificato mediante un'espressione <i>XPath</i> ; una singola espressione FLWOR può prevedere più clausole <b>for</b> , ciascuna legata a una diversa variabile di iterazione
<b>let</b>	Assegnazione del valore di una variabile per ogni singola iterazione (opzionale)
<b>where</b>	Applicazione di un filtro specificato da una condizione logica ai risultati dell'iterazione; più condizioni possono essere combinate mediante i connettivi logici <b>and</b> e <b>or</b> (opzionale)
<b>order by</b>	Specificazione di uno o più criteri di ordinamento degli elementi risultanti dall'iterazione; le parole chiave <b>ascending</b> e <b>descending</b> determinano il verso di ordinamento (opzionale)
<b>return</b>	Definizione del contenuto e del formato del risultato

Nella formulazione delle espressioni FLWOR del linguaggio *XQuery* è necessario ricorrere alle variabili, il cui nome è sempre preceduto dal simbolo «\$».

**OSSERVAZIONE** Le variabili del linguaggio *XQuery* sono «immutabili». Una volta assegnato loro un valore, esso non può essere modificato o riassegnato. Il tempo di vita delle variabili legate alle iterazioni di un'espressione FLWOR è limitato alla singola iterazione.

ESEMPIO

La seguente espressione *XQuery* è equivalente all'espressione *XPath* degli esempi precedenti:

for \$l in /libri/libro  
where \$l/lingua="Italiano"  
return \$l/titolo

File *XQuery*

Oltre all'uso interattivo del linguaggio *XQuery* – diretto da parte dell'utente, o mediante incorporazione in un programma sviluppato in un linguaggio ospite – è possibile realizzare dei veri e propri script *XQuery* che possono essere eseguiti da un sistema di gestione di database XML. In questo caso normalmente le espressioni che costituiscono il «corpo» dello script sono precedute da un prologo nel quale sono riportati i *namespace* riferiti, le eventuali funzioni definite dall'utente e, se necessarie, le variabili globali.

## Costruzione di elementi XML nel linguaggio XQuery e trasformazione di documenti XML

Il linguaggio XQuery consente di costruire esplicitamente elementi e attributi XML mediante le parole chiave **element** e **attribute**.

Questa caratteristica del linguaggio XQuery lo rende adatto per la definizione di regole di trasformazione di un documento XML in un diverso documento XML, anche se per questo scopo il W3C ha previsto un linguaggio specifico, XSLT (*Extensible Stylesheet Language Transformations*).

3. Nel caso di output HTML si tratterà quindi di XHTML.

4. L'eventuale uso di questi simboli deve quindi essere sostituito dalle entità `&#123;` e `&#125;`;

### ESEMPIO

La seguente espressione XQuery applicata al documento XML dell'esempio introduttivo del capitolo

```
for $l in /libri/libro
order by $l/anno
return $l/data(titolo)
```

restituisce i seguenti elementi ordinati in base al contenuto dell'elemento anno:

The Hobbit Pinocchio

**OSSERVAZIONE** La funzione predefinita `data()` utilizzata nell'esempio precedente estrae da un elemento XML il solo contenuto testuale; al suo posto può essere utilizzata la funzione di test `text()` che identifica gli elementi di tipo testuale:

```
for $l in /libri/libro
order by $l/anno
return $l/titolo/test()
```

La parola chiave **at** utilizzata nella clausola **for** consente di valorizzare una variabile con il conteggio delle iterazioni effettuate.

Il linguaggio XQuery viene spesso utilizzato, oltre che per selezionare i dati di interesse presenti in un documento XML, per «formattarli», ad esempio in linguaggio HTML, per la successiva visualizzazione.

Fermo restando che il risultato di un'espressione FLWOR deve rappresentare codice XML ben formato<sup>3</sup>, i singoli elementi XML prodotti dalla query devono essere compresi tra i simboli «{» e «}»<sup>4</sup>.

### ESEMPIO

La seguente espressione XQuery applicata al documento XML dell'esempio introduttivo del capitolo

```
<ul>
{
  for $l at $i in /libri/libro
  order by $l/anno
  return <li>{$i}. { $l/data(titolo) } { { $l/data(anno) } } </li>
}
</ul>
```

produce il seguente elenco HTML:

```
<ul>
  <li>1. The Hobbit (2001)</li>
  <li>2. Pinocchio (2010)</li>
</ul>
```

Nella definizione delle query in linguaggio *XQuery*, oltre alla possibilità di usare le funzioni predefinite del linguaggio *XPath*, è possibile calcolare i risultati come valori di un'espressione aritmetica utilizzando gli operatori del linguaggio *XPath* stesso.

In una espressione *XQuery* può essere inserito un commento delimitato dai simboli «(:» e «:»)».

#### ESEMPIO

La seguente espressione *XQuery* applicata al documento XML dell'esempio introduttivo del capitolo

```
<ul>
{
  (: elenco dei libri con indicazione del prezzo scontato del 25% :)
  for $l in /libri/libro
  order by $l/prezzo
  return <li>{$l/data(titolo)} ({ $l/prezzo*0.75})</li>
}
</ul>
```

produce il seguente elenco HTML:

```
<ul>
  <li>Pinocchio (11.25)</li>
  <li>The Hobbit (15)</li>
</ul>
```

### Funzioni definite dell'utente nel linguaggio *XQuery*

Oltre alle centinaia di funzioni predefinite rese disponibili dal linguaggio *XPath*, *XQuery* – come molti linguaggi di programmazione – consente di definire funzioni utilizzabili nella formulazione delle espressioni. Questa caratteristica rende estremamente flessibile l'uso del linguaggio di interrogazione.

Per i tipi XML predefiniti è possibile utilizzare dei «costruttori» per i valori costanti eventualmente necessari nella specificazione di una condizione utilizzando il nome del tipo preceduto dal prefisso «*xs:*» come invocazione di una funzione<sup>5</sup>.

5. Si tratta di fatto di una funzione che trasforma una stringa di caratteri in un valore del tipo specificato.

#### ESEMPIO

Disponendo del seguente documento XML contenuto nel file «*classe\_3A.xml*»

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<classe anno="3" sezione="A">
  <studente>
    <cognome>Rossi</cognome>
    <nome>Mario</nome>
    <sex>M</sex>
    <luogoNascita>Roma</luogoNascita>
    <dataNascita>1999-12-01</dataNascita>
  </studente>
  <studente>
    <cognome>Bianchi</cognome>
    <nome>Maria</nome>
    <sex>F</sex>
    <luogoNascita>Milano</luogoNascita>
    <dataNascita>2000-01-31</dataNascita>
```

```

</studente>
<studente>
  <cognome>Verdi</cognome>
  <nome>Luigi</nome>
  <sex>M</sex>
  <luogoNascita>Firenze</luogoNascita>
  <dataNascita>1999-06-30</dataNascita>
</studente>
<studente>
  <cognome>Neri</cognome>
  <nome>Luisa</nome>
  <sex>F</sex>
  <luogoNascita>Napoli</luogoNascita>
  <dataNascita>2000-06-01</dataNascita>
</studente>
</classe>

```

la seguente espressione FLWOR

```

for $s in doc("classe_3A.xml")/classe/studente
where $s/dataNascita >= xs:date("2000-01-01")
order by $s/cognome, $s/nome
return <nominativo>{$s/data(cognome)}, {$s/data(nome)}</nominativo>

```

restituisce l'elenco in ordine alfabetico dei soli studenti nati dopo il 1° gennaio 2000 incluso:

```

<nominativo>Bianchi, Maria</nominativo>
<nominativo>Neri, Luisa</nominativo>

```

La funzione `distinct-values()` consente di eliminare le ripetizioni da un elenco di valori testuali.

#### ESEMPIO

A partire dal documento XML dell'esempio precedente l'espressione *XPath*

```
data(/classe/studente/luogoNascita)
```

restituisce la sequenza di valori

```
Roma Milano Roma Napoli
```

mentre la seguente

```
distinct-values(/classe/studente/luogoNascita)
```

fornisce la sequenza di valori

```
Roma Milano Napoli
```

La seconda espressione *XPath* è equivalente all'espressione FLWOR

```

for $s in distinct-values(/classe/studente/luogoNascita)
return $s

```

Il ricorso alla funzione `distinct-values()` unito all'uso della clausola `let` per assegnare valori a una variabile consente di effettuare query di raggruppamento sui dati contenuti in un documento XML. In questo contesto risultano spesso utili le funzioni di aggregazione che operano su insiemi di elementi (TABELLA 2).

TABELLA 2

count	Restituisce il numero degli elementi
avg	Restituisce la media dei valori numerici degli elementi
min	Restituisce il minimo dei valori degli elementi
max	Restituisce il massimo dei valori degli elementi
sum	Restituisce la somma dei valori numerici degli elementi

#### ESEMPIO

A partire dal documento XML dell'esempio precedente l'espressione FLWOR

```
for $l in distinct-values(/classe/studente/luogoNascita)
let $s := /classe/studente[luogoNascita = $l]
return <nascita città="{ $l}" studenti="{count($s)}"/>
```

restituisce i seguenti elementi XML:

```
<nascita città="Roma" studenti="2"/>
<nascita città="Milano" studenti="1"/>
<nascita città="Napoli" studenti="1"/>
```

**OSSERVAZIONE** L'operazione di raggruppamento avviene iterando con la variabile specificata per la clausola `for` sui gruppi che si intendono costituire, la cui unicità è garantita dalla funzione `distinct-values()`, e imponendo con la sintassi *XPath* nella clausola `let` che la variabile assegnata sia l'insieme degli elementi appartenenti al gruppo. Nella clausola `return` è possibile fare riferimento al gruppo e invocare funzione di aggregazione sull'insieme degli elementi del gruppo stesso.

#### ESEMPIO

A partire dal seguente documento XML

```
<?xml version="1.0" ?>
<libri>
  <libro>
    <ISBN>123-456-789</ISBN>
    <titolo>Pinocchio</titolo>
    <lingua>Italiano</lingua>
    <anno>2001</anno>
    <prezzo>15.00</prezzo>
  </libro>
```

```

<libro>
  <ISBN>987-654-321</ISBN>
  <titolo>The Hobbit</titolo>
  <lingua>Inglese</lingua>
  <anno>2010</anno>
  <prezzo>20.00</prezzo>
</libro>
<libro>
  <ISBN>111-222-333</ISBN>
  <titolo>The Lord of Rings</titolo>
  <lingua>Inglese</lingua>
  <anno>2011</anno>
  <prezzo>40.00</prezzo>
</libro>
</libri>

```

l'espressione FLWOR

```

for $lingua in distinct-values(/libri/libro/lingua)
let $libri := /libri/libro[lingua = $lingua]
return <libri lingua="{ $lingua }"
      quantità="{count($libri)}"
      valore="{sum($libri/prezzo)}"/>

```

restituisce i seguenti elementi XML

```

<libri lingua="Italiano" quantità="1" valore="15"/>
<libri lingua="Inglese" quantità="2" valore="60"/>

```

**OSSERVAZIONE** Diversamente dal linguaggio SQL è possibile elencare gli elementi di un gruppo, come nella seguente espressione FLWOR

```

for $lingua in distinct-values(/libri/libro/lingua)
let $libri := /libri/libro[lingua = $lingua]
return <libri lingua="{ $lingua }"
      titoli="{data($libri/titolo)}"/>

```

che restituisce i seguenti elementi XML

```

<libri lingua="Italiano" titoli="Pinocchio"/>
<libri lingua="Inglese" titoli="The Hobbit The Lord
                                of Rings"/>

```

Dato che la logica di selezione delle espressioni FLWOR è la stessa del costrutto **SELECT** del linguaggio SQL, è possibile esprimere una condizione di *join* tra due documenti XML nello stesso modo in cui la si esprime tra due tabelle, cioè imponendo come condizione l'equivalenza di un elemento comune.

Disponendo in una base di dati XML del file «libri.xml» contenente il documento XML dell'esempio precedente e il file «autori.xml» contenente il seguente documento XML

```
<?xml version="1.0" ?>
<autori>
  <autore>
    <ISBN>123-456-789</ISBN>
    <cognome>Collodi</cognome>
    <nome>Carlo</nome>
  </autore>
  <autore>
    <ISBN>987-654-321</ISBN>
    <cognome>Tolkien</cognome>
    <nome>John Ronald Reuel</nome>
  </autore>
  <autore>
    <ISBN>111-222-333</ISBN>
    <cognome>Tolkien</cognome>
    <nome>John Ronald Reuel</nome>
  </autore>
</autori>
```

la seguente espressione *XQuery*

```
<libri>
{
  for $l in doc("libri.xml")/libri/libro
  for $a in doc("autori.xml")/autori/autore
  where $l/ISBN = $a/ISBN
  return
    <libro>
      <autore>{$a/data(cognome)}</autore>
      <titolo>{$l/data(titolo)}</titolo>
    </libro>
}
</libri>
```

fornisce come risultato il seguente elemento XML che realizza l'operazione di *join* tra i due documenti XML iniziali:

```
<libri>
  <libro>
    <autore>Collodi</autore>
    <titolo>Pinocchio</titolo>
  </libro>
  <libro>
    <autore>Tolkien</autore>
    <titolo>The Hobbit</titolo>
  </libro>
  <libro>
    <autore>Tolkien</autore>
    <titolo>The Lord of Rings</titolo>
  </libro>
</libri>
```



**OSSERVAZIONE** L'espressione FLWOR dell'esempio precedente può essere così definita in modo equivalente:

```
<libri>
{
  for $l in doc("libri.xml")/libri/libro,
    $a in doc("autori.xml")/autori/autore
  where $l/ISBN = $a/ISBN
  return
    <libro>
    <autore>{$a/data(cognome)}</autore>
    <titolo>{$l/data(titolo)}</titolo>
    </libro>
}
</libri>
```

Nella restituzione del risultato di una query è possibile utilizzare l'espressione condizionale **if ... then ... else ...** per produrre output diversificati.

#### ESEMPIO

La seguente espressione FLWOR applicata al documento XML dell'esempio precedente

```
<libri>
{
  for $l in doc("libri.xml")/libri/libro,
    $a in doc("autori.xml")/autori/autore
  where $l/ISBN = $a/ISBN
  return if ($l/lingua = "Italiano")
  then
    <libro-italiano>
    <autore>{$a/data(cognome)}</autore>
    <titolo>{$l/data(titolo)}</titolo>
    </libro-italiano>
  else
    <libro-straniero>
    <autore>{$a/data(cognome)}</autore>
    <titolo>{$l/data(titolo)}</titolo>
    </libro-straniero>
}
</libri>
```

fornisce come risultato il seguente elemento XML:

```
<libri>
  <libro-italiano>
    <autore>Collodi</autore>
    <titolo>Pinocchio</titolo>
  </libro-italiano>
  <libro-straniero>
```

```

    <autore>Tolkien</autore>
    <titolo>The Hobbit</titolo>
  </libro-straniero>
  <libro-straniero>
    <autore>Tolkien</autore>
    <titolo>The Lord of Rings</titolo>
  </libro-straniero>
</libri>

```

L'espressione condizionale **if ... then ... else ...** può essere utile per verificare se in una sequenza di elementi almeno uno, o tutti soddisfano un requisito: a questo scopo è possibile utilizzare i quantificatori **some** e **every** in congiunzione con la parola chiave **satisfies**.

#### ESEMPIO

La seguente espressione FLWOR applicata al documento XML dell'esempio precedente

```

<libri>
{
  for $libro in doc("libri.xml")/libri/libro
  return if (every $lingua in doc("libri.xml")/libri/libro/lingua
    satisfies ($lingua = "Italiano"))
  then
    <libro>{$libro/data(titolo)}</libro>
  else
    <libro lingua="{ $libro/data(lingua) }">{$libro/data(titolo)}</libro>
}
</libri>

```

fornisce come risultato il seguente elemento XML in cui per ogni elemento *libro* viene specificato l'attributo *lingua*

```

<libri>
  <libro lingua="Italiano">Pinocchio</libro>
  <libro lingua="Inglese">The Hobbit</libro>
  <libro lingua="Inglese">The Lord of Rings</libro>
</libri>

```

perché non tutti i libri selezionati sono in lingua italiana.

**OSSERVAZIONE** Se il documento XML fosse stato il seguente

```

<?xml version="1.0" ?>
<libri>
  <libro>
    <ISBN>123-456-789</ISBN>
    <titolo>I misteri della jungla nera</titolo>
    <lingua>Italiano</lingua>
  </libro>
</libri>

```

```

</libro>
<libro>
  <ISBN>987-654-321</ISBN>
  <titolo>I pirati della Malesia</titolo>
  <lingua>Italiano</lingua>
</libro>
<libro>
  <ISBN>111-222-333</ISBN>
  <titolo>Le tigri di Mompracem</titolo>
  <lingua>Italiano</lingua>
</libro>
<libro>
  <ISBN>999-888-777</ISBN>
  <titolo>Le due tigri</titolo>
  <lingua>Italiano</lingua>
</libro>
</libri>

```

l'espressione FLWOR dell'esempio precedente avrebbe prodotto il seguente risultato:

```

<libri>
  <libro>I misteri della jungla nera</libro>
  <libro>I pirati della Malesia</libro>
  <libro>Le tigri di Mompracem</libro>
  <libro>Le due tigri </libro>
</libri>

```

## 2 API per la gestione di documenti XML con il linguaggio Java

Il linguaggio di programmazione Java prevede una API (*Application Program Interface*) standard per la gestione di documenti XML denominata JAXP (*Java API for XML Processing*). Tra le elaborazioni che le classi del package che costituiscono JAXP consentono di effettuare sono fondamentali le seguenti:

- validazione di un documento XML rispetto a uno schema XSD;
- produzione di un nuovo documento XML ed eventuale salvataggio in un file di testo;
- lettura di un documento XML contenuto in un file di testo o reso disponibile come risorsa in rete ed eventuale verifica della correttezza;
- trasformazione di un documento XML.

### XQuery in Java

Il linguaggio Java prevede una API denominata XQJ (*XQuery API for Java*) che consente la valutazione di espressioni XQuery riferite a uno o più documenti XML da parte del codice di un programma.

► La lettura di un documento XML in una struttura dati che è possibile gestire da parte del codice di un programma è tecnicamente definita *parsing*.

Storicamente il *parsing* di un documento XML da parte di un programma si basa su una delle due seguenti modalità fondamentali:

- **SAX (*Simple API for XML*)**: la lettura del documento è condotta in modo autonomo dal *parser* e avviata dal codice che istanzia una classe di gestione degli «eventi» (*event handler*), i cui metodi sono invocati automaticamente in corrispondenza di specifiche tipologie di elementi presenti nel documento. L'ordine in cui gli elementi sono analizzati dal *parser* non è prevedibile a priori, inoltre questa modalità non consente di modificare il documento XML originale;
- **DOM (*Document Object Model*)**: il documento viene interamente «mappato» in una struttura ad albero che rappresenta la struttura gerarchica degli elementi del documento XML e il codice può liberamente «navigare» i nodi dell'albero. La modalità DOM consente di modificare il documento XML originale e anche di crearne dinamicamente uno completamente nuovo.

**OSSERVAZIONE** La modalità di *parsing* SAX è estremamente efficiente in quanto il documento XML viene letto in modo sequenziale e solo gli elementi intercettati dal gestore degli eventi sono presi in considerazione: il costo dell'efficienza è rappresentato dalla relativa rigidità della tecnica. Viceversa la tecnica di *parsing* DOM richiede la memorizzazione in strutture dati del programma dell'intero albero che rappresenta il documento, cosa che – nel caso di documenti voluminosi – può richiedere una grande quantità di memoria.

I vari package che costituiscono JAXP permettono di effettuare sia il *parsing* con modalità SAX sia quello di tipo DOM.

I package principali di JAXP sono i seguenti:

- *java.xml.parsers*: interfaccia comune per i *parser* dei vari produttori;
- *org.w3c.dom*: API per il *parsing* di tipo DOM;
- *org.xml.sax*: API per il *parsing* di tipo SAX;
- *javax.xml.transform*: API per la gestione delle regole di trasformazione di un documento XML (comprende le API per il *parsing* di tipo StAX);
- *javax.xml.stream*: API per la gestione della lettura e della scrittura dei file in formato XML.

**OSSERVAZIONE** Molte classi JAXP fondamentali devono essere istanziate seguendo la logica del *design pattern Factory-method*: l'invocazione di specifici metodi della classe *Factory* restituisce i riferimenti alle classi istanziate che sono successivamente utilizzate per operare.

### **Factory-method design pattern**

Il *design pattern Factory-method* prevede una classe «creatrice» di oggetti per i quali è definita un'interfaccia che viene concretizzata in modo diverso da parte di sottoclassi distinte.

Nel seguito del capitolo gli esempi fanno riferimento a documenti XML conformi al seguente schema XSD:



```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- definizione del tipo libro -->
  <xs:complexType name="tipo_libro">
    <xs:sequence>
      <xs:element name="autore" type="xs:string"/>
      <xs:element name="titolo" type="xs:string"/>
      <xs:element name="prezzo" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
  <!-- definizione del tipo libro con genere-->
  <xs:complexType name="tipo_libro_genere">
    <xs:complexContent>
      <xs:extension base="tipo_libro">
        <xs:attribute name="genere" type="xs:string"
          use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <!-- definizione dello schema libri -->
  <xs:element name="libri">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="libro" type="tipo_libro_genere"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

#### ESEMPIO

Il seguente è un documento XML valido rispetto allo schema XSD precedente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libri>
  <libro genere="fantascienza">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <prezzo>10.50</prezzo>
  </libro>
  <libro genere="romanzo">
    <autore>Mark Twain</autore>
    <titolo>Le avventure di Huckleberry Finn</titolo>
    <prezzo>9.90</prezzo>
  </libro>
  <libro genere="romanzo">
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <prezzo>15.00</prezzo>
  </libro>
</libri>
```

Lo schema XSD precedente trova una naturale rappresentazione mediante una collezione di oggetti Java istanza della seguente classe:

```
public class Libro {  
    private String genere;  
    private String titolo;  
    private String autore;  
    private float prezzo;  
  
    public Libro() {  
        this.genere = "";  
        this.titolo = "";  
        this.autore = "";  
        this.prezzo = 0;  
    }  
  
    public Libro(String genere, String titolo, String autore,  
                  float prezzo) {  
        this.genere = genere;  
        this.titolo = titolo;  
        this.autore = autore;  
        this.prezzo = prezzo;  
    }  
  
    public Libro(Libro libro) {  
        this.genere = libro.genere;  
        this.titolo = libro.titolo;  
        this.autore = libro.autore;  
        this.prezzo = libro.prezzo;  
    }  
  
    public String getGenere() {  
        return genere;  
    }  
  
    public String getTitolo() {  
        return titolo;  
    }  
  
    public String getAutore() {  
        return autore;  
    }  
  
    public float getPrezzo() {  
        return prezzo;  
    }  
  
    public void setGenere(String genere) {  
        this.genere = genere;  
    }  
}
```



```

public void setTitolo(String titolo) {
    this.titolo = titolo;
}

public void setAutore(String autore) {
    this.autore = autore;
}

public void setPrezzo(float prezzo) {
    this.prezzo = prezzo;
}

public String toString() {
    return "[" + genere + "] " + autore + ", \"" + titolo +
        "\" (" + prezzo + "€)";
}
}

```

## 2.1 Validazione di un documento XML rispetto a uno schema XSD

Nella seguente classe di esempio il metodo *validate* effettua la validazione di un documento XML rispetto a uno schema XSD entrambi forniti come parametri nella forma dei nomi dei file che li contengono:



```

import java.io.*;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.*;
import org.xml.sax.SAXException;

public class Validate {
    public static void validate(String XMLdocument, String XSDschema)
        throws SAXException, IOException {
        // creazione di uno schema XSD a partire dal file
        SchemaFactory factory =
            SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
        File schemaFile = new File(XSDschema);
        Schema schema = factory.newSchema(schemaFile);
        // creazione di un validatore rispetto allo schema XSD
        Validator validator = schema.newValidator();
        // validazione del documento XML
        Source source = new StreamSource(XMLdocument);
        validator.validate(source);
    }

    public static void main(String[] args) throws IOException {
        try {
            Validation.validate(args[0], args[1]);
            System.out.println("Documento XML valido.");
        }
    }
}

```



```

catch (SAXException exception) {
    System.out.println("Documento XML NON valido:");
    System.out.println(exception.getMessage());
}
}
}

```

Il metodo *main* della classe *Validate* illustra l'uso del metodo *validate* della stessa classe effettuando la validazione di un documento XML rispetto a uno schema XSD i cui nomi dei file sono forniti come argomenti della riga di comando. Il metodo statico *validate* genera l'eccezione di tipo *SAXException* nel caso di documento non corretto o non valido; in questo caso il metodo *getMessage* dell'eccezione restituisce una descrizione puntuale dell'errore individuato.

**OSSERVAZIONE** La classe *StreamSource* utilizzata per istanziare un oggetto rappresentante il documento XML da fornire come argomento al metodo di validazione prevede un costruttore che, anziché un file, accetta come argomento un URL generico. Allo stesso modo il metodo *newSchema* della classe *SchemaFactory* accetta come argomento anche un URL generico oltre che un file.

## 2.2 Parsing di un documento XML con SAX

Il *parsing* di un documento XML con modalità SAX richiede la definizione di una classe *event handler* che implementi i metodi delle interfacce *ContentHandler* ed *ErrorHandler*, ciascuno dei quali viene invocato dal *parser* in corrispondenza di specifici eventi che si verificano nel corso dell'analisi del documento (TABELLA 3).

TABELLA 3

characters	Contenuto testuale di un elemento (passato come parametro)
endDocument	Fine del documento
endElement	Fine di un elemento (il nome viene passato come parametro)
endPrefixMapping	Fine di una dichiarazione di <i>namespace</i> (il prefisso viene passato come parametro)
error	Notifica di un errore
fatalError	Notifica di un errore non recuperabile
ignorableWhitespace	Notifica di spazi ignorati
processingInstruction	Notifica di un'istruzione di elaborazione del documento
skippedEntity	Notifica di un'entità non interpretata (il nome viene passato come parametro)
startDocument	Inizio del documento
startElement	Inizio di un elemento (il nome viene passato come parametro)
startPrefixMapping	Inizio di una dichiarazione di <i>namespace</i> (il prefisso viene passato come parametro; eventuali attributi sono passati come parametro di tipo <i>Attributes</i> )
warning	Notifica di un avviso

### Parsing di un documento XML con StAX

I *parser* SAX sono noti come *push parser* perché inviano gli elementi del documento XML al codice in modo asincrono e indipendentemente dallo stato di questo.

Un *pull parser*, invece, permette al codice di richiedere esplicitamente gli elementi del documento XML quando effettivamente necessari.

JAXP comprende un nuovo tipo di *parser* denominato StAX, *Streaming API for XML*, che pur garantendo la stessa efficienza di SAX consente di effettuare il *parsing* in modo sincrono con il codice che necessita dei dati XML. Pur condividendo con SAX alcune limitazioni, come la necessità di analizzare il documento XML sequenzialmente, StAX permette di scrivere, oltre che di leggere, un file in formato XML.



Spesso solo alcune tipologie degli elementi che il *parser* SAX individua sono di interesse dello sviluppatore che implementa un'applicazione di lettura di un documento XML; la classe *DefaultHandler* del package *org.xml.sax.helpers* realizza un *event handler* in cui ogni metodo è definito, ma è privo di codice: la tecnica standard per ottenere un *parser* SAX consiste nel derivare dalla classe *DefaultHandler* una classe che ne ridefinisce i metodi di interesse.

La classe che implementa lo *event handler* viene successivamente fornita come parametro a un oggetto di classe *SAXParser* che viene istanziato utilizzando un oggetto di classe *SAXParserFactory*.

## ESEMPIO

La seguente classe esemplifica la realizzazione di un *parser* SAX per un documento XML conforme allo schema definito in precedenza; scopo del metodo *parseDocument* della classe è quello di costruire una collezione di oggetti di classe *Libro* a partire dal contenuto di un file XML.



```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class Parser extends DefaultHandler {
    private List libri;
    private String testo;
    private Libro libro;

    public Parser() {
        libri = new ArrayList();
    }

    public List getLibri() {
        return libri;
    }

    public void parseDocument(String filename)
        throws SAXException, ParserConfigurationException, IOException {
        // generazione di un'istanza del parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser parser = factory.newSAXParser();
        // parsing del file e registrazione dello event handler
        parser.parse(filename, this);
    }

    /* metodi di gestione degli eventi */
    public void startDocument() {
        System.out.println("Inizio del parsing del documento...");
    }

    public void endDocument() {
        System.out.println("...fine del parsing del documento.");
    }
}
```



```

public void startElement(String globalNamespace, String localNamespace,
                        String elementName, Attributes attributes)
    throws SAXException {
    testo = "";
    if (elementName.equalsIgnoreCase("libro")) {
        // creazione di un nuovo oggetto di classe Libro
        libro = new Libro();
        libro.setGenere(attributes.getValue("genere"));
    }
}

public void characters(char[] characters, int start, int length)
    throws SAXException {
    testo = new String(characters, start, length);
}

public void endElement(String globalNamespace, String localNamespace,
                      String elementName)
    throws SAXException {
    if (elementName.equalsIgnoreCase("libro")) {
        libri.add(libro);
    }
    else if (elementName.equalsIgnoreCase("autore")) {
        libro.setAutore(testo);
    }
    else if (elementName.equalsIgnoreCase("titolo")) {
        libro.setTitolo(testo);
    }
    else if (elementName.equalsIgnoreCase("prezzo")) {
        libro.setPrezzo(Float.parseFloat(testo));
    }
}

public void warning(SAXParseException exception) {
    System.out.println(exception.getMessage());
}

public void error(SAXParseException exception) {
    System.out.println(exception.getMessage());
}

public void fatalError(SAXParseException exception) {
    System.out.println(exception.getMessage());
}

public static void main(String[] args) {
    Parser sax = new Parser();
    try {
        sax.parseDocument(args[0]);
    }
    catch (SAXException | ParserConfigurationException | IOException exception) {
        System.out.println("Errore!");
    }
}

```



```
// iterazione della lista e visualizzazione degli oggetti
System.out.println("Numero di libri: " + sax.libri.size());
Iterator iterator = sax.libri.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next().toString());
}
}
```

Il metodo *main* della classe *Parser* esemplifica l'uso della classe stessa istanziando un oggetto ed effettuando il *parsing* di un documento XML contenuto in un file il cui nome viene fornito come argomento della riga di comando. Per ogni elemento e in modo ricorsivo il *parser* invoca in sequenza i metodi *startElement* ed *endElement*: nel caso di contenuto testuale viene invocato nell'ordine anche il metodo *characters*, cui sono forniti come argomenti un array di caratteri, la posizione del primo carattere utile e la lunghezza del testo contenuto nell'elemento.

Nella classe *Parser* il metodo *startElement* invocato prima di analizzare un elemento *libro* ha il compito di inizializzare un nuovo oggetto di classe *Libro* che viene aggiunto alla collezione dal codice del metodo *endElement* invocato dopo l'analisi di un elemento *libro*. Le proprietà di un oggetto di classe *Libro* sono impostate con i valori degli attributi e degli elementi compresi in un elemento *libro* del file XML (*genere*, *autore*, *titolo* e *prezzo*) nel momento in cui, per ciascuno di essi, viene invocato il metodo *endElement*, tenendo conto che il valore è stato precedentemente memorizzato nella proprietà *testo* dall'invocazione del metodo *characters*.

**OSSERVAZIONE** Il metodo *startElement* nella classe *Parser* dell'esempio precedente ha come parametro un oggetto di classe *Attributes* che rappresenta gli attributi dell'elemento. La TABELLA 4 riepiloga i metodi fondamentali della classe.

TABELLA 4

<code>getLength</code>	Restituisce il numero di attributi dell'elemento
<code>getQName</code>	Restituisce il nome di un attributo specificandone l'indice
<code>getType</code>	Restituisce il tipo di un attributo specificandone l'indice o il nome
<code>getValue</code>	Restituisce il valore di un attributo specificandone l'indice o il nome
<code>getIndex</code>	Restituisce l'indice di un attributo specificandone il nome

## 2.3 Parsing di un documento XML con DOM

Il *parsing* di un documento XML in modalità DOM comporta la costruzione dell'albero i cui nodi sono istanze di classi che implementano l'interfaccia *Node* o una sua interfaccia derivata, come quelle di TABELLA 5, definite nel package *org.w3c.dom*.

TABELLA 5

<code>Document</code>	Intero albero del documento XML
<code>Element</code>	Elemento di un documento XML
<code>Attr</code>	Attributo di un elemento
<code>Text</code>	Contenuto testuale di un elemento

L'interfaccia *Node* definisce, tra gli altri, i seguenti metodi per la navigazione dell'albero DOM (TABELLA 6).

TABELLA 6

<code>appendChild</code>	Aggiunge un nuovo nodo figlio dopo i figli esistenti
<code>getAttributes</code>	Restituisce la collezione degli attributi
<code>getChildNodes</code>	Restituisce la collezione dei nodi figli
<code>getFirstChild</code>	Restituisce il primo nodo figlio
<code>getLastChild</code>	Restituisce l'ultimo nodo figlio
<code>getNextSibling</code>	Restituisce il prossimo nodo fratello
<code>getNodeName</code>	Restituisce il nome del nodo
<code>getNodeValue</code>	Restituisce il valore del contenuto del nodo
<code>getNodeType</code>	Restituisce il tipo di nodo
<code>getParentNode</code>	Restituisce il nodo padre
<code>getPreviousSibling</code>	Restituisce il nodo fratello precedente
<code>hasAttributes</code>	Restituisce vero se sono presenti attributi, falso altrimenti
<code>hasChildNodes</code>	Restituisce vero se esistono nodi figli, falso altrimenti
<code>insertBefore</code>	Aggiunge un nuovo nodo figlio prima del nodo figlio specificato
<code>removeChild</code>	Rimuove il nodo figlio specificato
<code>replaceChild</code>	Sostituisce il nodo figlio specificato con un nuovo nodo figlio
<code>setNodeValue</code>	Imposta il valore del contenuto del nodo

L'interfaccia *Document* derivata da *Node* aggiunge, tra gli altri, i seguenti metodi specifici (TABELLA 7).

TABELLA 7

<code>createAttribute</code>	Crea un nodo attributo
<code>createElement</code>	Crea un nodo elemento
<code>createTextNode</code>	Crea un nodo testuale
<code>getDocumentElement</code>	Restituisce il nodo radice del documento
<code>getElementsByTagName</code>	Restituisce la collezione degli elementi che hanno il nome specificato
<code>getXmlEncoding</code>	Restituisce la codifica del documento XML
<code>getXmlVersion</code>	Restituisce la versione XML del documento

L'interfaccia *Element* derivata da *Node* aggiunge, tra gli altri, i seguenti metodi specifici (TABELLA 8).

TABELLA 8

<code>getAttribute</code>	Restituisce il valore dell'attributo specificato
<code>getElementsByTagName</code>	Restituisce la collezione degli elementi che hanno il nome specificato
<code>getTagName</code>	Restituisce il nome dell'elemento
<code>hasAttribute</code>	Restituisce vero se l'elemento ha l'attributo specificato, falso altrimenti
<code>removeAttribute</code>	Rimuove dall'elemento l'attributo specificato
<code>setAttribute</code>	Aggiunge l'attributo specificato impostandone il valore

## DOM (Document Object Model)

DOM è uno standard W3C per la rappresentazione di documenti XML indipendente sia dal linguaggio di programmazione sia dall'effettiva implementazione del *parser*.

Le specifiche W3C del DOM prevedono tre diversi livelli di implementazione:

- navigazione e manipolazione del contenuto di un documento XML;
- gestione dei *namespace*, filtri per le visite dell'albero e gestione degli eventi;
- caricamento e salvataggio di file XML, valutazione di espressioni *XPath*, validazione rispetto a documenti DTD e schemi XSD.

## JDOM

I package di classi che costituiscono JAXP integrano e adattano le librerie di classi originariamente sviluppate per il *parsing* in modalità SAX e DOM.

In particolare le classi che implementano DOM sono soggette ai requisiti definiti da W3C in modo indipendente dal linguaggio di programmazione; di conseguenza non consentono allo sviluppatore software di adottare uno stile di programmazione completamente coerente con le caratteristiche del linguaggio Java.

JDOM è una rappresentazione a oggetti dei documenti XML realizzata in modo specifico per il linguaggio di programmazione Java. Le classi JDOM sono classi concrete che possono essere istanziate senza l'ausilio di classi *factory* e utilizzano le collezioni standard del linguaggio.

JDOM consente di leggere e scrivere direttamente sorgenti di documenti XML (file, risorse di rete ecc.) e di creare dinamicamente alberi DOM di documenti XML.

JDOM è un progetto *open-source*: i package di classi che lo costituiscono con la relativa documentazione sono liberamente disponibili sul sito <http://www.jdom.org>.

L'interfaccia *Attr* derivata da *Node* aggiunge, tra gli altri, i seguenti metodi specifici (TABELLA 9).

TABELLA 9

<code>getName</code>	Restituisce il nome dell'attributo
<code>getOwnerElement</code>	Restituisce l'elemento a cui appartiene l'attributo
<code>getValue</code>	Restituisce il valore dell'attributo
<code>setValue</code>	Imposta il valore dell'attributo

L'interfaccia *Text* derivata da *Node* aggiunge, tra gli altri, i seguenti metodi specifici (TABELLA 10).

TABELLA 10

<code>getWholeText</code>	Restituisce il contenuto testuale
<code>replaceWholeText</code>	Sostituisce il contenuto testuale

Infine l'interfaccia *NodeList* rappresenta una collezione ordinata di oggetti di tipo *Node* e definisce i seguenti metodi (TABELLA 11).

TABELLA 11

<code>getLength</code>	Restituisce il numero di nodi della collezione
<code>item</code>	Restituisce un nodo della collezione a partire dal suo indice di posizione

L'oggetto di classe *Document* che rappresenta l'albero DOM del documento XML viene istanziato utilizzando un oggetto di classe *DocumentBuilder*, a sua volta istanziato utilizzando un oggetto di classe *DocumentBuilderFactory*.

### ESEMPIO

La seguente classe esemplifica la realizzazione di un *parser* DOM per un documento XML conforme allo schema definito in precedenza; scopo del metodo *parseDocument* della classe è quello di costruire una collezione di oggetti di classe *Libro* a partire dal contenuto di un file XML che viene rappresentato mediante un oggetto di tipo *Document*.

```
import java.io.IOException;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class Parser {
    private List libri;

    public Parser() {
        libri = new ArrayList();
    }
}
```



```

public List parseDocument(String filename)
throws ParserConfigurationException, SAXException, IOException {
    DocumentBuilderFactory factory;
    DocumentBuilder builder;
    Document document;
    Element root, element;
    NodeList nodelist;
    Libro libro;

    // creazione dell'albero DOM dal documento XML
    factory = DocumentBuilderFactory.newInstance();
    builder = factory.newDocumentBuilder();
    document = builder.parse(filename);
    root = document.getDocumentElement();
    // generazione della lista degli elementi "libro"
    nodelist = root.getElementsByTagName("libro");
    if (nodelist != null && nodelist.getLength() > 0) {
        for (int i=0; i<nodelist.getLength(); i++) {
            element = (Element)nodelist.item(i);
            libro = getLibro(element);
            libri.add(libro);
        }
    }
    return libri;
}

private Libro getLibro(Element element) {
    Libro libro;
    String genere = element.getAttribute("genere");
    String titolo = getTextValue(element, "titolo");
    String autore = getTextValue(element, "autore");
    float prezzo = getFloatValue(element, "prezzo");
    libro = new Libro(genere, titolo, autore, prezzo);
    return libro;
}

// restituisce il valore testuale dell'elemento figlio specificato
private String getTextValue(Element element, String tag) {
    String value = null;
    NodeList nodelist;

    nodelist = element.getElementsByTagName(tag);
    if (nodelist != null && nodelist.getLength() > 0) {
        value = nodelist.item(0).getFirstChild().getNodeValue();
    }
    return value;
}

// restituisce il valore intero dell'elemento figlio specificato
private int getIntValue(Element element, String tag) {
    return Integer.parseInt(getTextValue(element, tag));
}

```



```
// restituisce il valore numerico dell'elemento figlio specificato
private float getFloatValue(Element element, String tag) {
    return Float.parseFloat(getTextValue(element, tag));
}

public static void main(String[] args) {
    List libri = null;
    Parser dom = new Parser();

    try {
        libri = dom.parseDocument(args[0]);
    }
    catch (ParserConfigurationException | SAXException | IOException exception) {
        System.out.println("Errore!");
    }

    // iterazione della lista e visualizzazione degli oggetti
    System.out.println("Numero di libri: " + libri.size());
    Iterator iterator = libri.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next().toString());
    }
}
}
```

Il metodo *main* della classe *Parser* esemplifica l'uso della classe stessa istanziando un oggetto ed effettuando il *parsing* di un documento XML contenuto in un file il cui nome viene fornito come argomento della riga di comando. Il metodo *parseDocument* della classe analizza sequenzialmente la collezione di elementi *libro* presenti nell'albero DOM che rappresenta il documento XML e per ciascuno di essi invoca il metodo *getLibro*, che restituisce un oggetto di classe *Libro* allo scopo di aggiungerlo alla lista che il metodo restituisce.

L'albero costruito da un *parser* DOM consente la libera «navigazione» dei nodi da parte del codice: questa caratteristica è alla base della flessibilità garantita dalla modalità di *parsing* DOM rispetto alla modalità SAX.

## ESEMPIO

Il metodo *printDOM* della classe *DOMTree* visualizza un documento XML generico a partire dall'albero DOM, che in questo caso rappresenta il contenuto di un file in formato XML, effettuandone la visita ricorsiva:

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class DOMTree {
    private Document document;

    public DOMTree(String filename)
        throws ParserConfigurationException, SAXException, IOException {
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
```



```

// creazione dell'albero DOM dal documento XML
factory = DocumentBuilderFactory.newInstance();
builder = factory.newDocumentBuilder();
document = builder.parse(filename);
}

public void printDOM(Node node)
{
    int type = node.getNodeType();
    switch (type) {
        // nodo radice del documento
        case Node.DOCUMENT_NODE:
            {
                printDOM(((Document)node).getDocumentElement());
                break;
            }
        // nodo generico del documento
        case Node.ELEMENT_NODE:
            {
                System.out.print("<");
                System.out.print(node.getNodeName());
                NamedNodeMap attributes = node.getAttributes();
                for (int i=0; i<attributes.getLength(); i++) {
                    printDOM(attributes.item(i));
                }
                System.out.println(">");
                if (node.hasChildNodes()) {
                    NodeList children = node.getChildNodes();
                    for (int i=0; i<children.getLength(); i++) {
                        printDOM(children.item(i));
                    }
                }
                System.out.print("</");
                System.out.print(node.getNodeName());
                System.out.println(">");
                break;
            }
        // nodo attributo
        case Node.ATTRIBUTE_NODE:
            {
                System.out.print(" "+node.getNodeName()+"=\""+
                    ((Attr)node).getValue()+"\"");
                break;
            }
        // nodo entita'
        case Node.ENTITY_REFERENCE_NODE:
            {
                System.out.print("&");
                System.out.print(node.getNodeName());
                System.out.print(";");
                break;
            }
    }
}

```





```

// nodo CDATA
case Node.CDATA_SECTION_NODE:
{
    System.out.print("<![CDATA[");
    System.out.print(node.getNodeValue());
    System.out.println("]]>");
    break;
}

// nodo testuale
case Node.TEXT_NODE:
{
    System.out.println(node.getNodeValue());
    break;
}

// nodo commento
case Node.COMMENT_NODE:
{
    System.out.print("<!--");
    System.out.print(node.getNodeValue());
    System.out.println("-->");
    break;
}
}

}

public static void main(String argv[]) {
    DOMTree tree = new DOMTree(argv[0]);
    tree.printDOM(tree.document);
}
}

```

Il metodo *main* della classe *DOMTree* esemplifica l'uso della classe stessa costruendo un albero DOM del documento XML contenuto nel file il cui nome viene fornito come argomento della riga di comando e successivamente visualizzandolo in formato XML.

**OSSERVAZIONE** Nel codice dell'esempio precedente viene utilizzato per contenere gli attributi di un nodo un oggetto istanza di una classe che implementa l'interfaccia *NamedNodeMap* che rappresenta una collezione di nodi che possono essere acceduti per nome. La seguente tabella riepiloga i metodi fondamentali definiti dall'interfaccia:

**TABELLA 12**

<code>getLength</code>	Restituisce il numero di nodi della collezione
<code>getNamedItem</code>	Restituisce il nodo specificandone il nome
<code>item</code>	Restituisce il nodo specificandone l'indice
<code>removeNamedItem</code>	Elimina il nodo specificato dal nome
<code>setNamedItem</code>	Aggiunge il nodo specificato

## 2.4 Creazione di un documento XML con DOM

La modalità di *parsing* DOM consente, oltre che di leggere un documento XML contenuto in un file o reso disponibile come risorsa accessibile in rete, di creare dinamicamente un albero DOM che può eventualmente essere trasformato in un documento XML.



### ESEMPIO

Il costruttore della seguente classe *GenerateXML* invoca il metodo *createDocument* per costruire un albero DOM che rappresenta la collezione di oggetti di classe *Libro* fornita come parametro; il metodo *printToFile* della stessa classe trasforma l'albero DOM in un file XML:

```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

public class GenerateXML {
    Document document;

    public GenerateXML(List libri) throws ParserConfigurationException {
        createDOMTree(libri);
    }

    private void createDOMTree(List libri)
        throws ParserConfigurationException {
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
        Element root;
        Iterator iterator;
        Libro libro;
        Element element;

        factory = DocumentBuilderFactory.newInstance();
        builder = factory.newDocumentBuilder();
        document = builder.newDocument();
        root = document.createElement("libri");
        document.appendChild(root);
        // per ogni oggetto Libro crea un nodo figlio del nodo radice
        iterator = libri.iterator();
        while (iterator.hasNext()) {
            libro = (Libro)it.next();
            element = createLibroElement(libro);
            root.appendChild(element);
        }
    }

    // metodo ausiliario per la creazione di un elemento XML Libro
    private Element createLibroElement(Libro libro) {
        Text text;

        Element book = document.createElement("libro");
        book.setAttribute("genere", libro.getGenere());
```

```

        Element author = document.createElement("autore");
        text = document.createTextNode(libro.getAutore());
        author.appendChild(text);
        book.appendChild(author);
        Element title = document.createElement("titolo");
        text = document.createTextNode(libro.getTitolo());
        title.appendChild(text);
        book.appendChild(title);
        Element price = document.createElement("prezzo");
        text = document.createTextNode(Float.toString(libro.getPrezzo()));
        price.appendChild(text);
        book.appendChild(price);
        return book;
    }

    private void printToFile(String filename)
    throws TransformerException {
        TransformerFactory factory;
        Transformer transformer;
        DOMSource source;
        StreamResult stream;

        factory = TransformerFactory.newInstance();
        transformer = factory.newTransformer();
        source = new DOMSource(document);
        stream = new StreamResult(new File(filename));
        transformer.transform(source, stream);
    }

    public static void main(String args[]) {
        Libro libro;
        GenerateXML xml = null;
        List libri = new ArrayList();
        // inizializzazione della lista
        libro = new Libro("fantasy", "The Hobbit", "Tolkien", 15);
        libri.add(libro);
        libro = new Libro("fantasy", "The Lord of rings", "Tolkien", 25);
        libri.add(libro);
        try {
            xml = new GenerateXML(libri);
            xml.printToFile(argv[0]);
        }
        catch (ParserConfigurationException | TransformerException exception) {
            System.out.println("Errore!");
        }
    }
}

```

**OSSERVAZIONE** Il codice del metodo *printToFile* della classe *GenerateXML* dell'esempio precedente impiega le classi che JAXP rende disponibili come implementazione del linguaggio di trasformazione XSLT (*eXtensible Stylesheet Language Transformations*).

■ **Documenti XML e DB relazionali.** I documenti XML costituiscono una generalizzazione dei database relazionali; infatti, se una tabella può essere rappresentata mediante un documento XML, il viceversa non è vero, in quanto gli schemi XSD per i quali i documenti XML validi possono essere rappresentati mediante una tabella sono solo un sottoinsieme di tutti gli schemi possibili.

■ **Linguaggio XQuery.** Il linguaggio XQuery è un'estensione del linguaggio XPath che consente di formulare vere e proprie interrogazioni su uno o più documenti XML che eventualmente generano nuovo codice XML nella forma di elementi, o insiemi di elementi. Il linguaggio XQuery **non** ha una sintassi XML.

■ **Interrogazioni XQuery.** Le interrogazioni del linguaggio XQuery consistono normalmente nella valutazione delle cosiddette espressioni FLWOR, così denominate dalle iniziali delle clausole che le costituiscono: **for in, let, where, order by** e **return**. Nella formulazione delle espressioni FLWOR del linguaggio XQuery è necessario ricorrere alle variabili, il cui nome è sempre preceduto dal simbolo «\$». Le variabili del linguaggio XQuery sono «immutabili»: una volta assegnato loro un valore, esso non può essere modificato o riassegnato; il tempo di vita delle variabili legate alle iterazioni di un'espressione FLWOR è limitato alla singola iterazione.

■ **Raggruppamento in XQuery.** La funzione `distinct-values()` consente di eliminare le ripetizioni da un elenco di valori testuali. Il ricorso alla funzione `distinct-values()`, unito all'uso della clausola **let** per assegnare valori a una variabile, consente di effettuare query di raggruppamento sui dati contenuti in un documento XML. In questo contesto risulta spesso utilizzare le funzioni di aggregazione che operano su insiemi di elementi: `count` (conteggio), `avg` (media), `min` (minimo), `max` (massimo) e `sum` (somma).

■ **Join in XQuery.** Dato che la logica di selezione delle espressioni FLWOR è la stessa del costruito SELECT del linguaggio SQL, è possibile esprimere una condizione di *join* tra due docu-

menti XML nello stesso modo in cui la si esprime tra due tabelle, cioè imponendo come condizione l'equivalenza di un elemento comune.

■ **Espressioni condizionali in XQuery.** Nella restituzione del risultato di una query è possibile utilizzare l'espressione condizionale **if ... then ... else ...** per produrre output diversificati. L'espressione condizionale **if ... then ... else ...** può essere utile per verificare se in una sequenza di elementi almeno uno o tutti soddisfano un requisito; a questo scopo è possibile utilizzare i quantificatori **some** e **every** in congiunzione con la parola chiave **satisfies**.

■ **XML e Java.** Il linguaggio di programmazione Java prevede una API (*Application Program Interface*) standard per la gestione di documenti XML denominata JAXP (*Java API for XML Processing*). Tra le elaborazioni che le classi dei package che costituiscono JAXP consentono di effettuare sono fondamentali le seguenti: validazione di un documento XML rispetto a uno schema XSD, produzione di un documento XML ed eventuale salvataggio in un file di testo, lettura di un documento XML contenuto in un file di testo o reso disponibile come risorsa in rete ed eventuale verifica della correttezza, trasformazione di un documento XML. Molte classi JAXP fondamentali devono essere istanziate seguendo la logica del *design pattern Factory-method*: l'invocazione di specifici metodi della classe *Factory* restituisce i riferimenti alle classi istanziate che sono successivamente utilizzate per operare.

■ **Parsing di documenti XML.** La lettura di un documento XML in una struttura dati che è possibile gestire da parte del codice di un programma è tecnicamente definita *parsing*; storicamente il *parsing* di un documento XML da parte di un programma si basa su una delle due seguenti modalità fondamentali: SAX (*Simple API for XML*) e DOM (*Document Object Model*). I vari package che costituiscono JAXP permettono di effettuare sia il *parsing* con modalità SAX sia quello di tipo DOM.

■ **Parsing SAX.** Nella modalità di *parsing* SAX la lettura del documento è condotta in modo

autonomo dal *parser* e avviata dal codice che istanzia una classe di gestione degli «eventi» i cui metodi sono invocati automaticamente in corrispondenza di specifiche tipologie di elementi presenti nel documento; l'ordine in cui gli elementi sono analizzati dal *parser* non è prevedibile a priori, inoltre questa modalità non consente di modificare il documento XML originale.

## QUESITI

### 1 La tabella di un database relazionale ...

- A ... non può mai essere rappresentata mediante un documento XML.
- B ... può essere sempre rappresentata mediante un documento XML.
- C ... può essere rappresentata mediante un documento XML solo se rispetta alcune condizioni.
- D Nessuna delle risposte precedenti è vera.

### 2 Quali delle seguenti affermazioni sono vere?

- A Uno schema XSD rappresenta l'aspetto intensionale dei dati.
- B Uno schema XSD rappresenta l'aspetto estensionale dei dati.
- C Un documento XML rappresenta l'aspetto intensionale dei dati.
- D Un documento XML rappresenta l'aspetto estensionale dei dati.

### 3 Il linguaggio XQuery ...

- A ... è un'estensione del linguaggio XPath.
- B ... è espresso con una sintassi XML.
- C ... è un'estensione del linguaggio XSLT.
- D Nessuna delle risposte precedenti è vera.

### 4 Le espressioni FLWOR del linguaggio XQuery ...

- A ... sono sempre costituite da 5 clausole (**for**, **let**, **where**, **order by** e **return**).
- B ... sono sempre costituite da almeno 3 clausole obbligatorie (**for**, **where** e **return**).
- C ... sono sempre costituite da almeno 2 clausole obbligatorie (**for** e **return**).
- D Nessuna delle risposte precedenti è vera.

**Parsing DOM.** Nella modalità di *parsing* DOM il documento viene interamente «mappato» in una struttura ad albero che rappresenta la struttura gerarchica degli elementi del documento XML e il codice può liberamente «navigare» i nodi dell'albero. La modalità DOM consente di modificare il documento XML originale e anche di crearne dinamicamente uno completamente nuovo.

### 5 Le variabili del linguaggio XQuery utilizzate in un'espressione FLWOR ...

- A ... possono essere riassegnate come le variabili di un qualsiasi linguaggio di programmazione.
- B ... una volta assegnato loro un valore, non possono essere riassegnate.
- C ... hanno un tempo di vita esteso alla valutazione dell'intera espressione.
- D Nessuna delle risposte precedenti è vera.

### 6 L'inserimento di un commento in un'espressione del linguaggio XQuery ...

- A ... si effettua utilizzando i simboli «<!--» e «-->».
- B ... si effettua utilizzando i simboli «(:» e «:»).
- C ... si effettua utilizzando i simboli «/\*» e «\*/».
- D Nessuna delle risposte precedenti è vera.

### 7 La funzione `distinct-values()` del linguaggio XQuery ...

- A ... non inserisce in un elenco eventuali elementi ripetuti più volte.
- B ... inserisce una sola volta in un elenco eventuali elementi ripetuti più volte.
- C ... inserisce in un elenco singole occorrenze, ma solo degli elementi ripetuti più volte.
- D Nessuna delle risposte precedenti è vera.

### 8 Associare le seguenti funzioni di aggregazione XPath/XQuery con la relativa funzione:

count	media dei valori numerici
avg	valore minimo
min	somma dei valori numerici
max	valore massimo
sum	conteggio dei valori

**9** Un'operazione di *join* tra due documenti XML ...

- A ... non è realizzabile con il linguaggio *XQuery*.
- B ... è realizzabile solo con la versione 2 del linguaggio *XQuery*.
- C ... è realizzabile con il linguaggio *XPath*, ma non con il linguaggio *XQuery*.
- D Nessuna delle risposte precedenti è vera.

**10** Quali elaborazioni consentono di effettuare le classi dei package che costituiscono JAXP?

- A Validazione di un documento XML rispetto a uno schema XSD.
- B Produzione di un nuovo documento XML.
- C Lettura di un documento XML ed eventuale verifica della correttezza.
- D Trasformazione di un documento XML.

**11** Un parser SAX ...

- A ... costruisce l'intero albero del documento XML.
- B ... analizza il documento XML generando specifici eventi in corrispondenza di specifiche tipologie di elementi.

- C ... permette di modificare un documento XML.
- D Nessuna delle risposte precedenti è vera.

**12** Un parser DOM ...

- A ... costruisce l'intero albero del documento XML.
- B ... analizza il documento XML generando specifici eventi in corrispondenza di specifiche tipologie di elementi.
- C ... permette di «navigare» un documento XML.
- D Nessuna delle risposte precedenti è vera.

## ESERCIZI

**1** Dato il documento XML riportato in basso e relativo a un sito web di autonoleggio, formulare le seguenti interrogazioni in linguaggio *XQuery*:

- a) elenco di tutte le classi presenti nel documento incluso in un tag `<classi>`;
- b) elenco per ogni classe dei modelli presenti con la sola indicazione della denominazione;

```
<?xml version="1.0"?>
<lista>
  <classe>
    <livello>A</livello>
    <modello>
      <denominazione>FIAT Panda</denominazione>
      <alimentazione>benzina</alimentazione>
      <cilindrata>900</cilindrata>
      <costo valuta="Euro">60.0</costo>
    </modello>
    <modello>
      <denominazione>FIAT Cinquecento</denominazione>
      <alimentazione>diesel</alimentazione>
      <cilindrata>1200</cilindrata>
      <costo valuta="Euro">80.0</costo>
    </modello>
  </classe>
  <classe>
    <livello>C</livello>
    <modello>
      <denominazione>Mercedes classe B</denominazione>
      <alimentazione>diesel</alimentazione>
      <cilindrata>2000</cilindrata>
      <costo valuta="Euro">120.0</costo>
    </modello>
  </classe>
</lista>
```

- c) elenco dei modelli con costo inferiore ai 100 € incluso in un tag `<modelli>` e con indicazione del costo;
- d) elenco delle classi in cui sono compresi modelli con costo superiore a 70 € incluso in un tag `<classi>`; per ogni classe devono essere riportati i modelli con indicazione del costo.

## 2 Dato il seguente documento XML relativo a un sito web di vendita di auto usate

```
<?xml version="1.0"?>
<lista>
  <auto>
    <venditore>Giorgio Meini</venditore>
    <produttore>Citroen</produttore>
    <modello>Picasso</modello>
    <anno>2002</anno>
    <colore>grigio-metallizzato</colore>
    <prezzo valuta="Euro">2500</prezzo>
  </auto>
  <auto>
    <venditore>FiorenzoFormichi</venditore>
    <produttore>Toyota</produttore>
    <modello>RAV 4</modello>
    <anno>2004</anno>
    <colore>blu-metallizzato</colore>
    <prezzo valuta="Euro">7500</prezzo>
  </auto>
</lista>
```

formulare le seguenti interrogazioni in linguaggio *XQuery*:

- a) elenco delle automobili con il venditore e il modello;
- b) elenco delle automobili con il modello e il prezzo ordinate per prezzo crescente;
- c) elenco dei modelli con indicato il numero di auto in vendita con colore metallizzato (si noti che i modelli possono ripetersi per venditori diversi).

## 3 Dato il seguente documento XML relativo a un sito web di vendita di libri

```
<?xml version="1.0"?>
<catalog>
  <book>
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
  </book>
  <book>
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
  </book>
  <book>
    <author>Corets, Eva</author>
    <title>Oberon's Legacy</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-03-10</publish_date>
  </book>
  <book>
    <author>Corets, Eva</author>
    <title>The Sundered Grail</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-09-10</publish_date>
  </book>
  <book>
    <author>Randall, Cynthia</author>
    <title>Lover Birds</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-09-02</publish_date>
  </book>
  <book>
    <author>Thurman, Paula</author>
    <title>Splash Splash</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-11-02</publish_date>
  </book>
  <book>
    <author>Knorr, Stefan</author>
    <title>Creepy Crawlies</title>
    <genre>Horror</genre>
    <price>4.95</price>
    <publish_date>2000-12-06</publish_date>
  </book>
  <book>
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
    <publish_date>2000-11-02</publish_date>
  </book>
</catalog>
```

formulare le seguenti interrogazioni in linguaggio XQuery:

- a) elenco HTML dei soli titoli dei libri con autore e prezzo in ordine di prezzo crescente;
- b) elenco HTML dei soli titoli dei libri pubblicati dopo il 2000 con data di pubblicazione in ordine di titolo;
- c) elenco HTML dei soli titoli dei libri raggruppati per genere;
- d) tabella HTML con il titolo e il prezzo del libro più costoso e del libro meno costoso.

## LABORATORIO

- 1** A partire dal database relazionale definito dal DB-schema riportato in basso e implementato utilizzando un DBMS My-SQL, sviluppare un programma in linguaggio Java che a partire da un file XML come il seguente

```
<?xml version="1.0"?>
<elenco>
  <personaggio>
    <nome>Paperone</nome>
```

```
    <città>Paperopoli</città>
  </personaggio>
  <personaggio>
    <nome>Paperino</nome>
    <città>Paperopoli</città>
  </personaggio>
  <personaggio>
    <nome>Qui</nome>
    <città>Paperopoli</città>
  </personaggio>
  <personaggio>
    <nome>Quo</nome>
    <città>Paperopoli</città>
  </personaggio>
  <personaggio>
    <nome>Qua</nome>
    <città>Paperopoli</città>
  </personaggio>
  <personaggio>
    <nome>Topolino</nome>
    <città>Topolinia</città>
  </personaggio>
  <personaggio>
    <nome>Pippo</nome>
    <città>Topolinia</città>
  </personaggio>
</elenco>
```

```
CREATE DATABASE Disneyland;
```

```
USE Disneyland;
```

```
CREATE TABLE Citta (
  sigla CHAR(2) NOT NULL,
  denominazione VARCHAR(16) NOT NULL,
  CONSTRAINT chiave_primaria PRIMARY KEY (sigla)
);
```

```
INSERT INTO Citta (sigla, denominazione) VALUES
('PA', 'Paperopoli'),
('TO', 'Topolinia');
```

```
CREATE TABLE Personaggi (
  nome VARCHAR(32) NOT NULL,
  citta CHAR(2) NOT NULL,
  CONSTRAINT chiave_primaria PRIMARY KEY (nome),
  CONSTRAINT chiave_esterna FOREIGN KEY (citta) REFERENCES Citta(sigla)
);
```



aggiorni, utilizzando le API JDBC e JAXP, il contenuto della tabella *Personaggi* del database con il contenuto del file XML.

- 2 Con riferimento al database relazionale definito nell'esercizio precedente sviluppare un programma Java che, utilizzando le API JDBC e JAXP, crei un file XML con un formato coerente con lo schema XSD riportato in basso.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="elenco">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="personaggio" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome" type="xs:string"/>
              <xs:element name="città" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## CHAPTER 3 SAX

XML is fundamentally about data; programming with XML, then, has to be fundamentally about getting at that data. That process, called parsing, is the basic task of the APIs I'll cover in the next several chapters. This chapter describes how an XML document is parsed, focusing on the events that occur within this process. These events are important, as they are all points where application-specific code can be inserted and data manipulation can occur.

I'm also going to introduce you to one of the two core XML APIs in Java: SAX, the Simple API for XML (<http://www.saxproject.org>). SAX is what makes insertion of this application-specific code into events possible. The interfaces provided in the SAX package are an important part of any programmer's toolkit for handling XML. Even though the SAX classes are small and few in number, they provide a critical framework for Java and XML to operate within. Solid understanding of how they help in accessing XML data is critical to effectively leveraging XML in your Java programs.

### Setting Up SAX

I'm increasingly of the "learning is best done by doing" philosophy, so I'm not going to hit you with a bunch of concept and theory before getting to code. SAX is a simple API, so you only need to understand its basic model, and how to get the API on your machine; beyond that, code will be your best teacher.

### Callbacks and Event-Based programming

SAX uses a *callback model* for interacting with your code; you may also have heard this model called *event-based programming*. Whatever you call it, it's a bit of a departure for object-oriented developers, so give it some time if you're new to this type of programming.

In short, the parsing process is going to hum along, tearing through an XML document. Every time it encounters a tag, or comment, or text, or any other piece of XML, it *calls back* into your code, signaling that an event has occurred. Your code then has an opportunity to act, based on the details of that event.

For example, if SAX encounters the opening tag of an element, it fires off a `startElement` event. It provides information about that event, such as the name of the element, its attributes, and so on, and then your code gets to respond. You, as a programmer, have to write code for each event that is important to you—from the start of a document to a comment to the end of an element. This process is summed up in Figure 3-1.

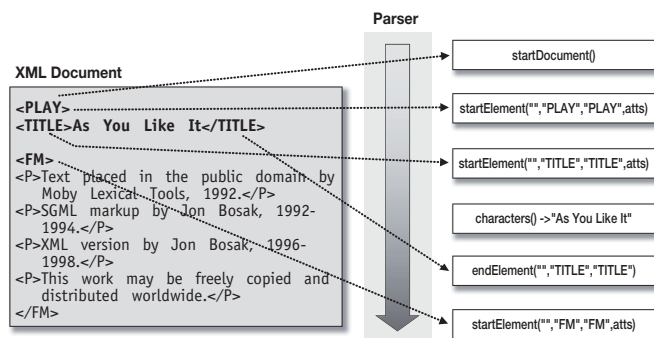


Figure 3.1. The parsing process is controlled by the parser and your code listens for events, responding as they occurs.

What's different about this model is that your code is not active, in the sense that it doesn't ever instruct the parser, "Hey, go and parse the next element." It's passive, in that it waits to be called, and then leaps into action. This takes a little getting used to, but you'll be an old hand by the end of the chapter.

### The SAX API

Unsurprisingly, the SAX API is made up largely of interfaces that define these various callback methods. You would implement the `ContentHandler` interface, and provide an implementation for the `characters()` method (for example) to handle events triggered by character processing. Figure 3-2 provides a visual overview of the API; you'll see that it's remarkably simple.

#### toolkit

cassetta degli strumenti

#### framework

struttura

#### leveraging

sfruttamento

#### to hum

canticchiare

#### tearing

correndo

#### to fire off

sparare

#### to leap

saltare

#### old hand

veterano

#### to trigger

innescare

#### tradeoff

compromesso

#### binding

legame

#### to aim

rivolgere

#### converse

contrario

#### to supply

fornire

#### facet

sfaccettatura

#### indictment

rinvio a giudizio

#### shortcoming

difetto

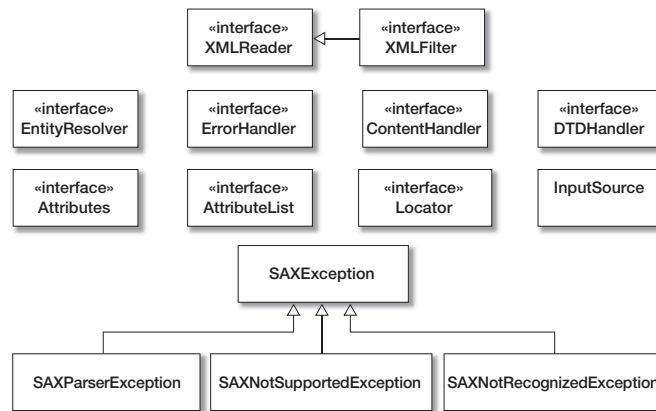


Figure 3.2. SAX is a powerful API. Even though it's largely interfaces and a few helper classes.

Keep in mind that a SAX-compliant parser will not implement many of these interfaces (`EntityResolver`, `ContentHandler`, `ErrorHandler`, etc.); that's the job of you, the programmer. The parser from your vendor will implement the `XMLReader` interface—and a few helper interfaces like `Attributes`—and provide parsing behavior; everything else is left up to you.

## CHAPTER 5 DOM

SAX is just one of several APIs that allow XML work to be done within Java. This chapter and the next will widen your API knowledge as I introduce the Document Object Model, commonly called the DOM. This API is quite a bit different from SAX, and complements the Simple API for XML in many ways. You'll need both, as well as the other APIs and tools in the rest of this book, to be a competent XML developer.

Because DOM is fundamentally different from SAX, I'll spend a good bit of time discussing the concepts behind DOM, and why it might be used instead of SAX for certain applications. Selecting any XML API involves tradeoffs, and choosing between DOM and SAX is certainly no exception. I'll move on to possibly the most important topic: code. I'll introduce you to a utility class that serializes DOM trees and will provide a pretty good look at the DOM structure and related classes. This will get you ready for some more advanced DOM work.

### The Document Object Model

The DOM, unlike SAX, has its origins in the World Wide Web Consortium (W3C; online at <http://www.w3.org>). Whereas SAX is public domain software, developed through long discussions on the XML-dev mailing list, DOM is a standard—just like the actual XML specification. The DOM is designed to represent the content and model of XML documents across all programming languages and tools. On top of that specification, there are several *language bindings*. These bindings exist for JavaScript, Java, CORBA, and other languages, allowing the DOM to be a cross-platform and cross-language specification.

### DOM Levels and Modules

In addition to being different from SAX in regard to standardization and language bindings, the DOM is organized into “levels” instead of versions. DOM Level One is an accepted recommendation, and you can view the completed specification at <http://www.w3.org/TR/REC-DOM-Level-1>. Level 1 details the functionality and navigation of content within a document. DOM Level 2, which was finalized in November of 2000, adds core functionality to DOM Level 1. There are also several additional DOM modules and options aimed at specific content models, such as XML, HTML, and CSS. These less-generic modules begin to “fill in the blanks” left by the more general tools provided in DOM Level 1. You can view the current DOM Level 2 Recommendation at <http://www.w3.org/TR/DOM-Level-2-Core>. This is actually the recommendation for the DOM Core; all the supplemental modules are represented by their own specifications:

[...]

The most recent entrant on the DOM scene, DOM Level 3 is a recommendation that builds upon the DOM Level 2 core. It's housed online at <http://www.w3.org/TR/DOM-Level-3-Core>,

and is mostly concerned with adding support for the XML InfoSet to DOM. However, it also provides a nice bootstrapping implementation, which you'll see in more detail later in this chapter. Support for DOM Level 3 in parsers is a bit sporadic, and you'll often see only portions of the specification implemented (bootstrapping being the most common one).

[...]

## DOM Concepts

In addition to fundamentals about the DOM specification, I want to give you a bit of information about the DOM programming structure itself. At the core of DOM is a tree model. Remember that SAX gave you a piece-by-piece view of an XML document, reporting each event in the parsing lifecycle as it happened. DOM is in many ways the converse of this, supplying a complete in-memory representation of the document. The document is supplied to you in a tree format, and all of this is built upon the DOM `org.w3c.dom.Node` interface. Deriving from this interface, DOM provides several XML-specific interfaces, like `Element`, `Document`, `Attr`, and `Text`. So, in a typical XML document, you might get a structure that looks like Figure 5-1.

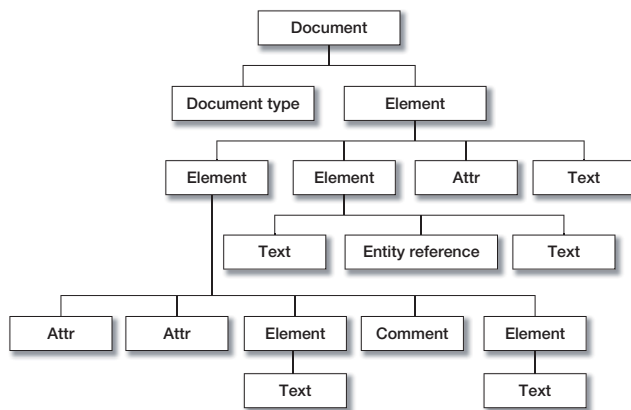


Figure 5.1. DOM documents are often called trees, for reasons obvious from the diagram.

A tree model is followed in every sense. This is particularly notable in the case of the `Element` nodes that have textual values (as in the `Title` element). Instead of the textual value of the node being available through the `Element` node (through, for example, a `getText()` method), there is a child node of type `Text`. So you would get the value of an element from the child `Text` node, rather than the `Element` node itself. While this might seem a little odd, it does preserve a very strict tree model in DOM, and allows tasks like walking the tree to be very simple algorithms, without a lot of special cases. Because of this model, all DOM structures can be treated either as their generic type, `Node`, or as their specific type (`Element`, `Attr`, etc.). Many of the navigation

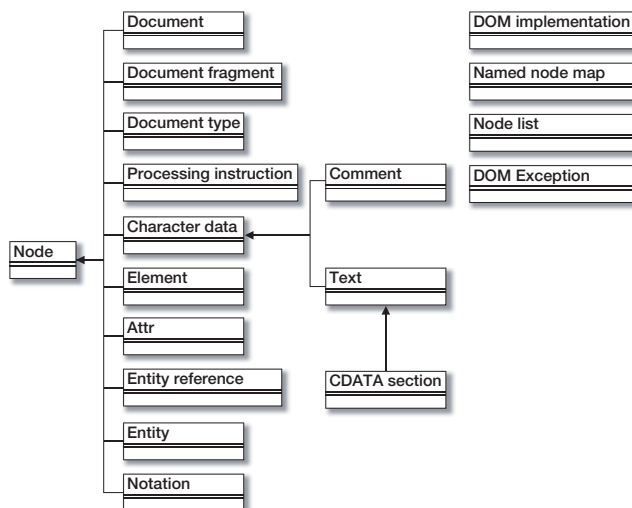


Figure 5.2. Most of the DOM model deals with XML representations; there are also a few exceptions and list classes.

methods, like `getParent()` and `getChildren()`, operate on that basic `Node` interface, so you can walk up and down the tree without worrying about the specific structure type. Another facet of DOM to be aware of is that, like SAX, it defines its own list structures. You'll need to use the `NodeList` and `NamedNodeMap` classes when working with DOM lists, rather than Java collections. Depending on your point of view, this isn't a positive or negative, just a fact of life. Figure 5-2 shows a simple UML-style model of the DOM core interfaces and classes, which you can refer to throughout the rest of the chapter.

## When SAX Sucks

There are a lot of times when using DOM isn't a choice, but a requirement. You'd do well to recognize those situations quickly, and not waste time trying to decide which API to use. Here are the times you essentially *must* use DOM:

### *You need random access to your document*

In SAX, you get information about the XML document as the parser does, and lose that information when the parser does. When the second element in a document comes along, SAX cannot access information in the fourth element, because that fourth element hasn't been parsed yet. When the fourth element does come along, SAX can't look back on that second element.

### *You need access to siblings of an element*

Moving laterally between elements is also difficult with the SAX parsing model. The access provided in SAX is largely hierarchical, as well as sequential. You are going to reach leaf nodes of the first element, then move back up the tree, then down again to leaf nodes of the second element, and so on. At no point is there any clear indication of what level of the hierarchy you are at. There is no concept of a sibling element, or of the next element at the same level, or of which elements are nested within which other elements.

These are hardly indictments of SAX; rather, they are issues that affect what API you use for a specific application. While the above issues are shortcomings for representing, say, an HTML document that must be styled with an XSL style-sheet, they are advantages for extracting data from a 10,000 line catalog entry where you don't need random access to the document.

[B. McLaughlin & J. Edelson, *Java & XML*, 3<sup>rd</sup> edition, O'Reilly, 2006]

## QUESTIONS

- a** What is the callback model that SAX use?
- b** Does DOM work like SAX?
- c** Which representation of an XML document is used by DOM?
- d** What is the difference between the `Node` class and the `Element`, `Attr`, `Text`, ... classes used in DOM?
- e** When are you obliged to use DOM instead of SAX?

# B

## Pagine web dinamiche con linguaggio PHP

**B1**

**Il linguaggio PHP e le *form* HTML**

**B2**

**La programmazione a oggetti  
nel linguaggio PHP**

**B3**

**Accesso a una base di dati in linguaggio PHP**

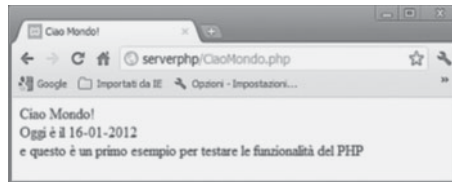


**B4**

**Sviluppo di pagine web dinamiche  
con IDE *NetBeans***

# Il linguaggio PHP e le *form* HTML

Osservando la seguente pagina web



si può pensare a una pagina realizzata in linguaggio HTML che visualizza un messaggio con un riferimento temporale relativo alla data di sistema del computer utilizzato.

È possibile però rilevare alcuni dettagli che portano a pensare che il codice che produce tale risultato non sia stato scritto utilizzando solo tag HTML:

## PHP

Il PHP (acronimo ricorsivo di «PHP: Hypertext Preprocessor», preprocessore di ipertesti; ma originariamente acronimo di «Personal Home Page») è nato nel 1994 a opera del programmatore danese Rasmus Lerdorf.

Esso è un linguaggio di scripting interpretato con una sintassi molto simile a quella del linguaggio C e viene distribuito con licenza open source specifica. Originariamente concepito per la programmazione di pagine web dinamiche, è attualmente utilizzato principalmente per sviluppare applicazioni web lato server. L'esecuzione del codice PHP sul server produce codice HTML da inviare al browser client che ha richiesto la pagina.

Molti siti famosi sono stati scritti in PHP: Wikipedia, Facebook, Flickr, Drupal, ecc.

Inoltre è il linguaggio utilizzato per molti sistemi CMS (*Content Management System*) come Joomla, Wordpress e Drupal.

- il suffisso del nome del file presente nell'area degli indirizzi non è né «.htm» né «.html» ma «.PHP»;
- sempre nell'area degli indirizzi, davanti al nome del file è presente il riferimento «serverPHP/» che non è relativo alla cartella che contiene il file «CiaoMondo.PHP», ma all'invocazione del server web a cui è demandato il compito di elaborarlo (nel caso fosse stato un file HTML avremmo potuto visualizzarlo tramite il browser, senza la necessità di un server web);
- con il solo linguaggio HTML non vi è la possibilità di rilevare e visualizzare la data del sistema in uso.

Esaminiamo quindi il codice della pagina web:

```
<!doctype html>
<html>
  <head>
    <title>Ciao Mondo!</title>
  </head>
  <body>
    <!-- Inizio script PHP -->
    <?php
      echo "Ciao Mondo! <br>";
      echo "Oggi è il " . Date("d-m-Y") . "<br>";
      echo "e questo è un primo esempio per testare
        le funzionalità del PHP <br>";
    ?> <!-- Fine script PHP -->
  </body>
</html>
```

Nel codice della pagina sono state utilizzate sia tag HTML che istruzioni di un diverso linguaggio, infatti:

- le prime e le ultime righe sono scritte in linguaggio HTML;
- le istruzioni comprese tra «<?php» e «?>» terminano tutte col simbolo «;» e sono scritte in linguaggio PHP;
- l'istruzione `echo` produce in output una stringa di caratteri formattata come se fosse codice HTML (qualsiasi browser opera sui *tag* di tale linguaggio);
- la funzione `Date` del linguaggio PHP restituisce la data corrente di sistema;
- il simbolo «.» è l'operatore PHP che permette di concatenare stringhe di caratteri.

**OSSERVAZIONE** La pagina web «CiaoMondo.php» avrebbe potuto essere scritta come segue (si notino i commenti in stile linguaggio C++ con l'uso di «//»):

```
<?php //Inizio script PHP
echo "<!doctype html>";
echo "<html>";
echo "<head>";
echo "<title>Ciao Mondo!</title>";
echo "</head>";
echo "<body>";
echo "Ciao Mondo! <br>";
echo "Oggi è il " . Date("d-m-Y") . "<br>";
echo "e questo è un primo esempio per testare
      le funzionalità del PHP <br>";
echo "</body>";
echo "</html>";
//Fine dello script PHP
?>
```

Le istruzioni PHP sono elaborate dal server e solo il loro output viene inserito nella pagina web inviata al browser: in linea di principio tutto il codice HTML potrebbe essere prodotto con istruzioni `echo`, ma i programmatori trovano più pratico alternare codice HTML «statico» e codice PHP «dinamico»

In questo capitolo vedremo le caratteristiche base del linguaggio PHP e come questo intergisce con HTML nella realizzazione di pagine WEB dinamiche.

## 1 Architetture software client-server

Prima di presentare le caratteristiche del linguaggio PHP, analizziamo il funzionamento di un'architettura client-server i cui elementi fondamentali sono i seguenti:



## Apache

Nato nel 1995, Apache HTTP Server, o più comunemente Apache, è il nome (scelto in onore dell'omonima tribù di nativi americani) della piattaforma server web più diffusa e in grado di operare su vari sistemi operativi (UNIX, Linux, Microsoft Windows, ecc.).

Operativamente è un *demone* (programma costantemente in funzione in attesa di richieste di servizi) in ambiente UNIX, o un servizio in ambiente Windows, che permette l'accesso a uno o più siti web, gestendo varie caratteristiche di sicurezza e potendo ospitare diverse estensioni per pagine dinamiche come PHP.

Lo sviluppo e la gestione di Apache sono curati da Apache Software Foundation che lo distribuisce con una licenza open source specifica.

- **client**: software (e per estensione il computer su cui è eseguito) che istanzia l'interfaccia utente di un'applicazione connettendosi tramite un'infrastruttura di rete a un server;
- **server**: software (e per estensione il computer su cui esso è eseguito) che, oltre alla gestione logica del sistema (fornitura di servizi), deve implementare le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza dei dati;
- **interazione client/server**: un client richiede un servizio a un server mediante uno specifico protocollo; il server, se trova soddisfatte tutte le condizioni per fornire il servizio richiesto lo eroga, altrimenti risponde al client con un messaggio di errore.

In questo contesto particolare, prendiamo in esame un'architettura client-server finalizzata alla gestione di un servizio web dove sulla macchina server è eseguito un server web che si occupa di fornire su richiesta dell'utente (client) file di qualsiasi tipo, tra cui pagine web che lato client sono visualizzate dal browser dell'utente. Le informazioni inviate dal server web all'utente e viceversa viaggiano sulla rete locale o attraverso Internet e sono gestite dal protocollo HTTP come mostrato nella FIGURA 1.

**OSSERVAZIONE** Normalmente un server web risiede su sistemi hardware dedicati<sup>1</sup>, ma può essere eseguito su un computer dove risiedono anche altri servizi, o su computer utilizzati anche per altri scopi, se viene installato il software specifico (Apache HTTP Server, Microsoft Internet Information Server, ecc.). Ad esempio si può installare un server web su un normale PC allo scopo di testare in locale il proprio sito web, oppure per consentire l'accesso ai propri documenti da altri computer, sia in una rete LAN, sia tramite Internet.

Si noti che una stessa macchina può configurarsi allo stesso tempo sia come server che come client se da un browser installato su una

1. Una server farm ospita tipicamente migliaia di computer server, ciascuno dei quali gestisce decine o centinaia di siti web.

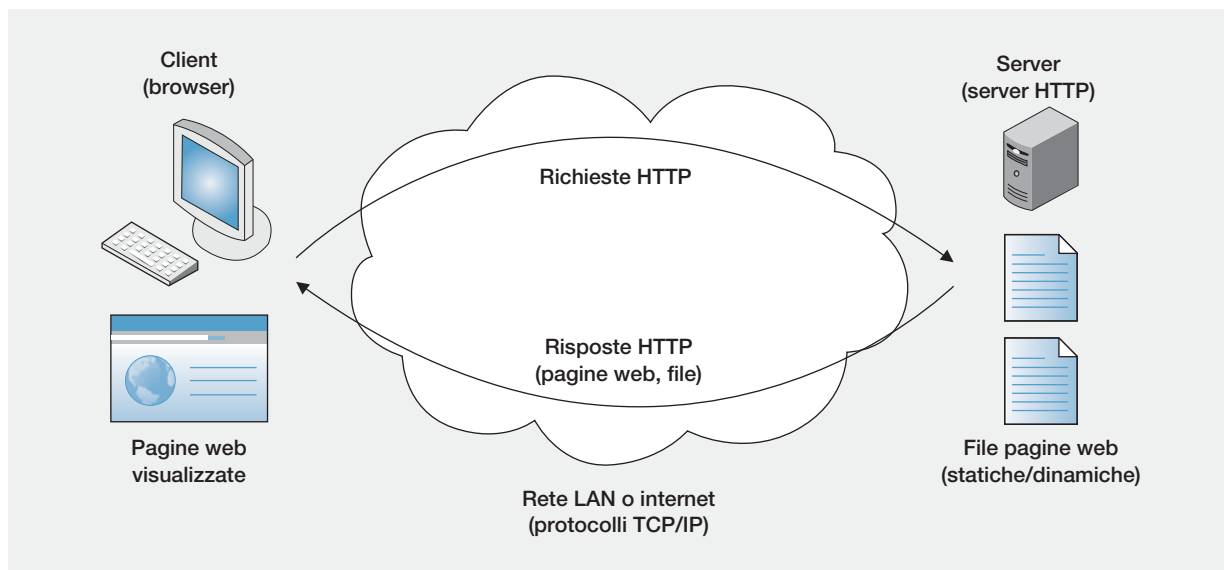


FIGURA 1

macchina su cui è in funzione un server web viene invocato l'indirizzo *localhost*, l'effetto è quello di inoltrare tale richiesta al server web locale che risponde alle richieste del browser come se fosse un client remoto. *localhost* è quindi il prefisso dell'indirizzo web per le pagine memorizzate nel computer locale. Se le pagine contenenti codice PHP sono invece memorizzate su un server remoto è necessario fare riferimento a esso con il relativo indirizzo web, o con l'indirizzo IP che lo identifica.

Quando si parla di architetture client-server sono spesso citati i **linguaggi di scripting client-side** e i **linguaggi di scripting server-side**.

I primi sono linguaggi che permettono l'incorporazione di comandi all'interno del codice HTML per l'implementazione di programmi che vengono eseguiti direttamente dal browser. Un tipico esempio in questo senso è rappresentato dal linguaggio JavaScript.

I secondi sono linguaggi che permettono di sviluppare programmi eseguiti direttamente sul server. La programmazione lato server è in generale un insieme di tecniche che consentono di generare risorse in tempo reale e che tramite un server possono essere rese fruibili ai client senza che esista alcun file statico corrispondente a esse: in questo modo sono realizzati siti web di tipo dinamico. Questo risultato può essere ottenuto generando su richiesta flussi di dati che producono output secondo formati tipici del web (HTML, XHTML, CSS, ecc.). La programmazione lato server permette di espletare funzioni quali l'autenticazione degli utenti, l'accesso a basi di dati remote, ecc. In questo modo è possibile sviluppare applicazioni simili a quelle tradizionali anche se con alcune limitazioni dovute all'interazione tra il client e il server. I grandi vantaggi di questo tipo di approccio sono principalmente due:

- la possibilità di utilizzare le applicazioni praticamente in ogni luogo dove sia disponibile una connessione alla rete Internet;
- l'utente può utilizzare le applicazioni senza dover installare sul computer client alcun software: è sufficiente un browser standard.

In questo contesto si distingue tra **pagine web statiche** e **pagine web dinamiche**:

- le prime sono pagine contenute in file esistenti sul server codificate utilizzando i linguaggi HTML e CSS (più eventuali script in linguaggio JavaScript) e sono immutabili nel tempo fino a quando non si decide di modificarle riscrivendo parte del loro codice;
- le seconde sono il prodotto di script eseguiti dal server e non esistono fisicamente file HTML corrispondenti: sul server sono presenti programmi che, quando eseguiti, inviano il proprio output (ogni riga del quale formattata come codice HTML) al browser del client che lo ha attivato invocando la pagina che lo contiene. Dal momento che in generale la sequenza delle istruzioni eseguite in un programma e il relativo output dipende dai dati che esso riceve in input, ne deriva che le pagine visualizzate lato client possono cambiare in maniera dinamica.

Relativamente all'esempio riportato in apertura di capitolo abbiamo la seguente situazione:

- sul server esiste il file «CiaoMondo.php» che contiene il seguente codice:

```
<!doctype html>
<html>
  <head>
    <title>Ciao Mondo!</title>
  </head>
  <body>
    <!-- Inizio script PHP -->
    <?php
      echo "Ciao Mondo! <br>";
      echo "Oggi è il " . Date("d-m-Y") . "<br>";
      echo "e questo è un primo esempio per testare
        le funzionalità del PHP <br>";
    ?> <!-- Fine script PHP -->
  </body>
</html>
```

- con il riferimento «serverphp/CiaoMondo.php» specificato dall'utente nell'aver degli indirizzi del browser del computer client viene inoltrata la richiesta di esecuzione del file PHP al server web attivo sul server;
- il server HTTP in esecuzione sul computer «serverphp» riceve la richiesta e:
  - verifica l'esistenza del file richiesto (in caso negativo restituisce un messaggio di errore del tipo «Oggetto non trovato: errore 404»);
  - attiva l'interprete PHP per eseguire le singole istruzioni contenute nel file che, se come in questo caso, sono sintatticamente corrette, producono un output che viene inviato al browser del computer client:

```
<!doctype html>
<html>
  <head>
    <title>Ciao Mondo!</title>
  </head>
  <body>
    Ciao Mondo! <br>
    Oggi è il 18/01/2012 <br>
    e questo è un primo esempio per testare le funzionalità
    del PHP <br>
  </body>
</html>
```

L'output prodotto dall'interprete PHP è una sequenza di righe di testo che costituiscono una pagina web in linguaggio HTML. Nel caso specifico la dinamicità della pagina è data dal fatto che eseguendo lo script PHP in giorni diversi cambia la data visualizzata.

**OSSERVAZIONE** L'interprete PHP si limita a tradurre ed eseguire le sole istruzioni del linguaggio. Le righe che riportano codice HTML sono invece inviate al browser così come sono codificate, senza alcuna elaborazione.

È possibile passare da codice PHP a codice HTML e viceversa in qualsiasi contesto.

#### ESEMPIO

Per produrre in output dieci diversi tag `<br>` si può scrivere:

```
<?php for ($i=0;$i<10;$i++) { ?>
    <br>
<?php } ?>
```

Come in altri linguaggi di programmazione in PHP è possibile includere file che contengono codice PHP. Nell'economia di un'applicazione PHP è possibile memorizzare parti di codice comune in un file che può essere incluso in un punto qualsiasi di uno script utilizzando la parola chiave **include**:

```
<?php
...
include ("common.php");
...
?>
```

**OSSERVAZIONE** La parola chiave **require** ha lo stesso effetto di **include**, con la differenza che interrompe l'esecuzione in caso di errore.

#### Inclusione univoca di codice

Le parole chiave **include\_once** e **require\_once** sono analoghe a **include** e **require**, ma oltre a includere il codice PHP contenuto nel file indicato ne impediscono la replicazione, cosa che tipicamente avviene quando si includono più file che a loro volta includono lo stesso file.

## 2 La sintassi del linguaggio PHP

PHP è un linguaggio di scripting di uso generale che viene prevalentemente usato per sviluppare programmi lato server ed è disponibile per i principali server web (Apache, Microsoft IIS, ecc.). Per PHP sono disponibili inoltre numerose librerie di funzioni che permettono la rapida realizzazione di applicazioni complesse.

L'integrazione di blocchi di istruzioni PHP all'interno del codice HTML, quando viene usato come linguaggio di *scripting server side*, può avvenire con diverse modalità di *tagging* che ne delimitano l'inizio e la fine:

- standard: `<?php ... ?>`;
- JavaScript: `<script language="PHP"> ... </script>`.

Un programma PHP è quindi costituito da uno o più blocchi di istruzioni delimitate dai tag che ne indicano inizio e fine; ogni istruzione termina col simbolo «;».

Le strutture di controllo del flusso di esecuzione sono praticamente le stesse dei linguaggi C/C++ e Java:

- **if-else**
- **switch-case**
- **while**
- **do-while**
- **for**
- **foreach**

I commenti di un programma PHP seguono, all'interno di ogni blocco, le regole dei linguaggi C/C++ e Java con l'uso di `/*...*/` per commenti su più righe e `//` per commentare un'unica riga. Per una sola riga priva di codice è possibile utilizzare `#`.

I principali operatori algebrici e di assegnamento del PHP sono simili a quelli dei linguaggi C/C++ e Java con l'aggiunta di quelli che permettono la concatenazione di stringhe:

Operatore	Significato
=	assegnamento
+	somma
-	sottrazione
*	moltiplicazione
/	divisione
%	resto
++	autoincremento
--	autodecremento
+=	assegnamento combinato a somma
-=	assegnamento combinato a differenza
*=	assegnamento combinato a moltiplicazione
/=	assegnamento combinato a divisione
%=	assegnamento combinato a resto
.	(punto) concatenazione di stringhe
.=	assegnamento combinato a concatenazione stringhe

2. Gli operatori di uguaglianza, disuguaglianza e identità si applicano anche agli array.

Allo stesso modo gli operatori di relazione previsti sono i seguenti<sup>2</sup>:

Operatore	Significato
==	uguaglianza
!=	<> disuguaglianza
===	identità (uguaglianza e stesso tipo)
<	minore
>	maggiore
<=	minore o uguale
>=	maggiore o uguale

Infine gli operatori logici:

Operatore	Significato
&& and	AND
 or	OR
^ xor	XOR
!	NOT

L'output di un programma PHP viene gestito tramite le istruzioni **echo** e **print**. I due costrutti sono praticamente identici, infatti le due istruzioni che seguono

```
echo "Alfa Beta Gamma";  
print "Alfa Beta Gamma";
```

producono lo stesso output: la stringa "Alfa Beta Gamma".

La differenza fondamentale risiede nel fatto che **echo** è in grado di stampare uno o più valori, mentre **print** ne gestisce uno alla volta:

```
echo "Uno ", "Due ", "Tre ";  
print "Uno Due Tre";
```

All'interno di uno script può presentarsi la necessità di interrompere il flusso delle istruzioni a seguito del verificarsi di determinate condizioni: in questi casi si ricorre alle istruzioni **die** o **exit** che sono identiche sia nella sintassi che nelle funzionalità: determinano l'arresto immediato del flusso delle istruzioni e consentono di inserire nella pagina web un eventuale messaggio.

#### ESEMPIO

Il codice che segue:

```
<?php  
echo "Primo messaggio";  
die("Script interrotto!");  
echo "Secondo messaggio";  
?>
```

prevede due **echo** per la visualizzazione di due messaggi di cui solo il primo viene visualizzato: il comando **die** terminando l'esecuzione dello script fa in modo che il secondo comando **echo** non sia eseguito.

Spesso questi comandi sono utilizzati per la gestione di errori nell'esecuzione delle istruzioni:

```
<?php  
...  
if($alfa>40) die("numero troppo grande");  
...  
?>
```

o per gestire gli errori nell'invocazione delle funzioni:

```
<?php  
$nomefile = '/test/beta.txt';  
$file = fopen($nomefile, 'r')  
    or exit("Impossibile aprire il file $nomefile");  
?>
```

Il secondo script è relativo al tentativo di apertura in lettura del file «beta.txt» memorizzato nella directory «test»: se la funzione **fopen** ritorna un valore falso perché l'operazione di apertura del file non ha avuto successo, il programma termina visualizzando un messaggio di errore. L'uso del comando **or** dopo l'invocazione di una funzione è tipico della gestione degli errori in PHP.

### 3 Le variabili del linguaggio PHP

Nel linguaggio PHP le variabili hanno il nome che inizia sempre col simbolo «\$» e non necessitano di una dichiarazione esplicita prima del loro utilizzo. I nomi delle variabili sono *case-sensitive*. In un qualsiasi punto all'interno di un programma PHP è possibile dichiarare e iniziare a utilizzare una variabile semplicemente nominandola. Una variabile assume un certo tipo in funzione del valore che le viene assegnato ed è possibile assegnarle in punti diversi tipi differenti.

#### ESEMPIO

Con le istruzioni:

```
...  
$alfa=5;  
...  
$alfa="ABCDEF";  
...
```

la variabile `$alfa` viene prima dichiarata e inizializzata al valore intero 5 e quindi successivamente modificata in tipo stringa con l'assegnazione del valore "ABCDEF".

**OSSERVAZIONE** La possibilità di nominare e inizializzare contestualmente l'uso di una variabile in un punto qualsiasi di un programma, se da una parte permette di scrivere il codice più velocemente, dall'altra può invogliare a un uso non pianificato e poco corretto delle variabili con possibile perdita di chiarezza del significato del programma. Inoltre questo meccanismo rende possibile l'introduzione di errori talvolta di non facile individuazione: un eventuale errore nella digitazione del nome di una variabile in una istruzione fa sì che il PHP la interpreti come una nuova variabile senza segnalare alcun errore.

Il linguaggio PHP prevede l'uso di **variabili dinamiche** che possono essere dichiarate come nell'esempio che segue.

#### ESEMPIO

Definita la variabile `$alfa` con

```
$alfa="Salve";
```

si crea una variabile il cui nome sia il valore della variabile `$alfa`:

```
$$alfa="Ciao";
```

PHP interpreta `$$alfa` sostituendo per primo il valore della variabile più interna: `$alfa` diventa quindi `Salve`, per cui si ha la variabile `$Salve` il cui valore è "Ciao".

**OSSERVAZIONE** PHP prevede variabili predefinite dedicate principalmente alla gestione di informazioni di servizio relative al server su cui il programma viene eseguito, il dialogo HTTP con i relativi parametri e le variabili di ambiente.

## Tipi di dato

Nonostante sia un linguaggio a «tipizzazione debole» che non richiede che ogni variabile sia definita con un proprio tipo, PHP mette a disposizione quattro tipi scalari di base, riportati nella TABELLA 1.

TABELLA 1

Tipo di dato	Descrizione
boolean	Rappresenta un valore booleano che può assumere i valori TRUE o FALSE.
integer	La sua rappresentazione dipende dalla piattaforma del sistema in uso, ma generalmente è rappresentato su 32 byte e può contenere valori da -2147483648 a +2147483647. Oltre alla notazione in base 10 è possibile utilizzare le notazioni: <ul style="list-style-type: none"><li>• in base 8 facendo precedere il numero da uno zero: <code>a\$=0123</code> (83 in base 10);</li><li>• in base 16 facendo precedere il numero da 0x: <code>b\$=0XEA2</code> (3746 in base 10).</li></ul>
double	È in virgola mobile su 64 bit a «precisione doppia»: i limiti teorici sono da -1.7976931348623157E+308 a -2.2250738585072014E-308 e da 2.2250738585072014E-308 a 1.7976931348623157E+308, oltre allo zero. La sintassi usata per esprimere questo tipo di numeri prevede due varianti: <ul style="list-style-type: none"><li>• notazione decimale: <code>a\$=23.561</code>;</li><li>• notazione esponenziale: <code>b\$=23561e-3</code>;</li></ul>
string	Rappresenta una sequenza di caratteri ASCII di lunghezza massima pari a 32 caratteri. La sintassi con cui si possono esprimere le costanti di tipo stringa prevede che le medesime siano racchiuse tra apici singoli («'») o doppi («"»).

Esiste inoltre il tipo speciale `NULL` che definisce una variabile senza alcun valore: la variabile esiste ma non è avvalorata.

**OSSERVAZIONE** PHP permette di inserire all'interno delle costanti stringa delle variabili che vengono automaticamente espanso se la costante ospite è delimitata da simboli «"».

Ad esempio, dopo aver eseguito le seguenti istruzioni

```
$beta=10;  
$alfa="il numero $beta ha due cifre";
```

la variabile `$alfa` conterrà la stringa "il numero 10 ha due cifre"; mentre dopo l'istruzione

```
$alfa='il numero $beta ha due cifre';
```

la variabile `$alfa` conterrà la stringa 'il numero \$beta ha due cifre';



PHP permette di inserire nelle stringhe di caratteri sequenze di *escape*. Per esempio l'output prodotto dalle due seguenti istruzioni:

```
$beta="a tutti"  
echo "Ciao\t$beta\n";
```

visualizza "Ciao" seguito da una tabulazione e quindi da "a tutti" seguito da una riga vuota.

Nella TABELLA 2 sono riportate le sequenze di *escape* supportate da PHP.

TABELLA 2

Sequenza di escape	Significato
\n	Line feed
\r	Carriage Return
\t	Tabulazione orizzontale
\\	Simbolo «\»
\\$	Simbolo «\$»
\"	Simbolo «"»
\123	Notazione per la rappresentazione ottale del codice ASCII di un carattere
\x12	Notazione per la rappresentazione esadecimale del codice ASCII di un carattere

Come abbiamo visto, quando si definisce implicitamente una variabile non occorre specificarne il tipo, ma l'assegnazione di un valore determina il tipo della variabile. È possibile impostare esplicitamente il tipo di una variabile utilizzando una tecnica di *casting* simile a quella del linguaggio C/C++.

**ESEMPIO**

Nella seguente istruzione

```
$alfa=(int)"12345ABCD";
```

se avessimo ommesso l'operatore di cast (**int**) PHP avrebbe creato una variabile stringa, invece, in questo modo, è stata creata una variabile intera il cui valore è 12345.

Nella TABELLA 3 vengono mostrati i principali operatori di cast.

TABELLA 3

Operatore	Cast	Verifica
(bool), (boolean)	A booleano	is_bool()
(int), (integer)	A intero	is_long()
(real), (double)	A numero in virgola mobile	is_float()
(string)	A stringa	is_string()
(array)	Ad array	is_array()
(unset)	A NULL	is_null()

Nella terza colonna della tabella sono state riportate le funzioni che permettono di controllare il tipo di una variabile: ritornano `TRUE` o `FALSE` a seconda che il tipo sia verificato o meno.

## Le costanti

Per dichiarare una costante in PHP si utilizza la seguente sintassi:

```
define("<nome_costante>",<valore>);
```

### ESEMPIO

Per il valore  $\pi$  si può usare la seguente costante:

```
define("PI_GRECO",3.14);
```

Tradizionalmente i nomi delle costanti sono scritti con tutte le lettere maiuscole in modo da agevolare la rapida individuazione nel codice.

## 4 Gli array del linguaggio PHP

In PHP gli array sono vettori indicizzabili sia attraverso un valore numerico il cui valore iniziale è lo zero, sia per mezzo di una chiave di tipo stringa. In quest'ultimo caso il vettore memorizza delle associazioni chiave/valore e viene detto array associativo. A differenza di altri linguaggi di programmazione PHP accetta che gli elementi di un vettore non siano tutti dello stesso tipo.

La maniera più veloce per definire un vettore è quella di utilizzare la parola chiave `array` che accetta come argomenti una sequenza di valori oppure delle coppie `chiave => valore` nel caso di array associativi.

### ESEMPIO

Usando la definizione che segue

```
$colori = array('bianco', 'nero', 'blu', 'verde', 'rosso');
```

ciascuno dei valori elencati può essere riferito tramite un indice numerico da 0 a 4. L'istruzione

```
echo $colori[2];
```

visualizza la stringa «blu».

Lo stesso tipo di memorizzazione si ottiene con la definizione:

```
$settimana =array(
    0 => "Lunedì",
    1 => "Martedì",
    2 => "Mercoledì",
    3 => "Giovedì",
    4 => "Venerdì",
    5 => "Sabato",
    6 => "Domenica");
```



Con la seguente dichiarazione si realizza un array associativo:

```
$elementi = array(  
    "Fe" => "Ferro",  
    "Na" => "Sodio",  
    "Al" => "Alluminio",  
    "K"  => "Potassio",  
    "Mn" => "Manganese");
```

e l'istruzione

```
echo $elementi["Na"];
```

visualizza la stringa «Sodio».

**OSSERVAZIONE** È possibile utilizzare una sintassi del tipo

```
$formazione = array( 1 => 'Buffon', 'Panucci', 'Nesta',  
                     'Cannavaro', 'Coco', 'Ambrosini',  
                     'Tacchinardi', 'Perrotta',  
                     'Totti', 'Inzaghi', 'Vieri',  
                     'CT' => 'Trapattoni');
```

che ha il significato di assegnare al primo elemento dell'array l'indice 1 e a tutti gli altri che seguono un valore incrementato di 1, escluso l'ultimo a cui viene associata la chiave «CT».

Esiste un metodo per aggiungere un valore all'array che può essere usato anche in alternativa ai precedenti per definire l'array:

```
$colori[] = 'giallo';
```

In questo modo viene aggiunto un nuovo elemento di indice 5 all'array `$colori` dell'esempio precedente. Questa sintassi è valida anche per definire un array; infatti, se l'array `$colori` non fosse stato definito, questa istruzione lo avrebbe definito creando l'elemento con indice 0. È possibile indicare direttamente l'indice, anche in modo non consecutivo, come nel seguito:

```
$colori[3] = 'arancio';  
$colori[7] = 'viola';
```

**OSSERVAZIONE** Dopo le istruzioni precedenti, la prima delle quali modifica l'elemento di indice 3, si ha un nuovo elemento di indice 7 con il valore «viola». È da notare che, dopo queste istruzioni, l'array non ha alcun valore nella posizione di indice 6. In ogni caso un'eventuale istruzione come la seguente

```
$colori[] = 'grigio';
```

assegnerebbe l'elemento di indice 8.

In PHP è possibile creare strutture complesse di dati utilizzando gli array a più dimensioni, cioè un array nel quale uno o più elementi sono a loro volta array.

#### ESEMPIO

Per memorizzare in un array e successivamente visualizzare i dati anagrafici di più persone (nome, cognome, data di nascita e città di residenza) si ricorre al codice che segue:

```
$persone = array(
    array('nome' => 'Marco',
        'cognome' => 'Bianchi',
        'data_nascita' => '1964/06/25',
        'residenza' => 'Firenze'),
    array('nome' => 'Paolo',
        'cognome' => 'Rossi',
        'data_nascita' => '1965/05/06',
        'residenza' => 'Roma'),
    array('nome' => 'Gianni',
        'cognome' => 'Neri',
        'data_nascita' => '1963/10/16',
        'residenza' => 'Milano'));
echo $persone[0]['cognome']; // visualizza 'Bianchi'
echo $persone[1]['residenza']; // visualizza 'Roma'
echo $persone[2]['nome']; // visualizza 'Gianni'
```

In questo modo viene definito un array formato a sua volta da tre array ognuno dei quali ha un indice numerico a partire da 0. All'interno dei singoli array tutte le chiavi sono associative.

In questo caso i tre array «interni» hanno la stessa struttura, ma in realtà è possibile dare a ciascuno di essi una struttura diversa, come nell'esempio che segue:

```
$persone = array( 1 => array('nome' => 'Mario Rossi',
    'residenza' => 'Roma',
    'ruolo' => 'impiegato'),
    2 => array('nome' => 'Paolo Bianchi',
    'data_nascita' => '1967/05/06',
    'residenza' => 'Torino'),
    'numero_elementi' => 2);
echo $persone[1]['residenza']; // visualizza 'Roma'
echo $persone['numero_elementi']; // visualizza '2'
```

L'array `$persone` è formato da due array interni, ai quali sono stati assegnati gli indici 1 e 2 e da un terzo elemento, che non è un array ma una variabile intera con chiave associativa *numero\_elementi*. I due array che costituiscono i primi due elementi hanno una struttura diversa: il primo è formato dagli elementi *nome*, *residenza* e *ruolo*, il secondo è formato da *nome*, *data\_nascita* e *residenza*.

In PHP gli elementi degli array possono essere scorsi utilizzando il ciclo **foreach**.

Per visualizzare gli elementi dell'array

```
$elementi = array(
    "Fe" => "Ferro",
    "Na" => "Sodio",
    "Al" => "Alluminio",
    "K" => "Potassio",
    "Mn" => "Manganese");
```

è possibile utilizzare il seguente ciclo

```
foreach ($elementi as $e) {
    echo $e."<br";
}
```

che visualizza i singoli elementi ognuno su una singola riga:

```
Ferro
Sodio
Alluminio
Potassio
Manganese
```

In alternativa il seguente codice

```
foreach ($elementi as $chiave=>$valore) {
    echo "$chiave=$valore<br>";
}
```

visualizza gli elementi del vettore in questo modo:

```
Fe=Ferro
Na=Sodio
Al=Alluminio
K=Potassio
Mn=Manganese
```

## 4.1 Array correlati al web

Per quanto riguarda le attività sul web, PHP prevede degli specifici array associativi denominati «superglobali» e dedicati a specifiche funzionalità. Questi array hanno nomi predefiniti: `$_GET`, `$_POST`, `$_REQUEST`, `$_COOKIE` e `$_SESSION`. I primi tre sono relativi al passaggio di dati tra pagine web, mentre gli altri due, come vedremo in un successivo paragrafo, sono dedicati alla gestione di dati che debbono essere comuni alle pagine esplorate durante una sessione di utilizzo del browser da parte dell'utente.

`$_GET` è un array superglobale creato automaticamente con l'esistenza di una stringa di interrogazione all'interno di un URL, o all'invio dei dati di una *form* col metodo *GET* che usa come veicolo l'URL della pagina richiesta. Qualsiasi informazione inviata con la stringa d'interrogazione in una coppia nome/valore viene caricata nell'array associativo `$_GET` della pagina richiesta.

#### ESEMPIO

L'URL

<http://serverphp/page.php?nome=Pippo&citta=Topolinia>

richiede al server *serverphp* la pagina *page.php* generando nell'array `$_GET` gli elementi «Pippo» con chiave di accesso «nome» e «Topolinia» con chiave di accesso «citta».

**OSSERVAZIONE** Il vantaggio del ricorso a `$_GET` è rappresentato dalla possibilità di accedere alle informazioni generate da una pagina dall'interno di uno script da essa invocato. L'array `$_GET` viene aggiornato a ogni richiesta di una pagina quindi, volendo conservare alcuni valori attraverso la navigazione tra più pagine, è necessario memorizzare in esso i valori per ogni pagina invocata nella sequenza delle richieste effettuate.

L'array superglobale `$_POST` è simile al precedente con la differenza che invece di usare come mezzo di trasporto dei dati la stringa di interrogazione dell'URL, usa il metodo *POST* del protocollo HTTP. Dal momento che i dati trasferiti non sono visibili nella URL rimangono invisibili all'utente aumentando il livello di sicurezza. `$_POST` viene usato principalmente dai moduli presenti nelle pagine web, ma può essere usato indipendentemente dai *form*.

`$_REQUEST` è un array superglobale che contiene gli elementi di entrambi gli array `$_GET` e `$_POST`. L'unico svantaggio è che le chiavi dell'array debbono avere nomi diversi, altrimenti una chiave potrebbe riferirsi a un dato di uno qualsiasi dei due array.

## URL, URI e IRI

URL (*Uniform Resource Locator*) è una stringa di caratteri che identifica e localizza una risorsa presente nella rete locale o in Internet (normalmente una pagina ospitata su un server): è comunemente chiamato «indirizzo web» in quanto viene digitato nell'area degli indirizzi del browser per accedere alla risorsa stessa. Esempi di URL sono i seguenti:

<http://www.google.com>  
<http://server/index.html>  
<http://server/index.php>  
<ftp://zanichelli.it/file.txt>

Il prefisso di un URL, che viene spesso ommesso, identifica il protocollo di accesso alla risorsa.

Un URL è un caso particolare di URI (*Uniform Resource Identifier*) che consente di identificare non solo una risorsa accessibile in rete, ma anche risorse locali come i file, o prive di una localizzazione fisica come gli indirizzi di posta elettronica (in questo caso l'URI assume il nome di URN (*Uniform Resource Name*)).

Il prefisso di un URI definisce uno schema formalmente registrato dalla *Internet Assigned Numbers Authority* (IANA) che solo nel caso di URL corrisponde al protocollo di accesso.

Le stringhe di definizione degli URI (e quindi degli URL e degli URN) sono composte da caratteri ASCII: questa limitazione è superata dagli IRI (*Internationalized Resource Identifier*) che possono essere codificati con caratteri Unicode: la conversione di un IRI in URI è resa automatica da uno specifico algoritmo.

## 5 Le funzioni del linguaggio PHP

PHP prevede una notevole varietà di librerie di funzioni che coprono moltissime esigenze per chi sviluppa software (accesso a database, gestione di documenti PDF, produzioni di grafici, ecc.). Senza voler essere esaustivi, elenchiamo di seguito alcuni gruppi di funzioni standard del linguaggio PHP classificate per tipologia<sup>3</sup>.

3. La documentazione delle funzioni è consultabile nel manuale ufficiale del linguaggio disponibile all'URL [www.php.net/manual/funcref.php](http://www.php.net/manual/funcref.php).

## Funzioni per le variabili

Nome funzione	Descrizione
<code>empty</code>	Verifica se la variabile è vuota oppure no. Per «vuota» si intende che può contenere una stringa vuota o un valore numerico pari a 0, ma può anche essere non definita o essere impostata al valore <code>NULL</code> (l'eventuale indicazione di una variabile non definita, in questo caso, non genera errore). Restituisce un valore booleano.
<code>isset</code>	Verifica se la variabile è definita. Una variabile risulta non definita quando non è stata inizializzata o è stata impostata al valore <code>NULL</code> . Restituisce un valore booleano.
<code>is_null</code>	Verifica se la variabile equivale a <code>NULL</code> , ma genera un errore se viene invocata su una variabile non definita. Restituisce un valore booleano.
<code>is_int</code> <code>is_integer</code> <code>is_long</code>	Le tre funzioni sono equivalenti, verificano se la variabile è di tipo numerico intero. Restituiscono un valore booleano.
<code>is_float</code> <code>is_double</code> <code>is_real</code>	Le tre funzioni sono equivalenti, verificano se la variabile è di tipo numerico in virgola mobile. Restituiscono un valore booleano.
<code>is_string</code>	Verifica se la variabile è una stringa. Restituisce un valore booleano.
<code>is_array</code>	Verifica se la variabile è un array. Restituisce un valore booleano.
<code>is_numeric</code>	Verifica se l'argomento contiene un valore numerico. È importante la distinzione fra questa funzione e <code>is_int()</code> o <code>is_float()</code> , perché queste ultime, nel caso di una stringa che contiene valori numerici, restituiscono falso, mentre <code>is_numeric()</code> restituisce vero. Restituisce un valore booleano.
<code>gettype</code>	Restituisce una stringa che rappresenta il tipo di dato dell'argomento, per esempio: <b>boolean</b> , <b>integer</b> , <b>double</b> , <b>string</b> , <b>array</b> .
<code>print_r</code>	Inserisce nella pagina web informazioni relative al contenuto della variabile. È utile in fase di debug. Se l'argomento è un array sono visualizzate le chiavi e i valori relativi.
<b><code>unset</code></b>	Distrugge la variabile specificata. In realtà non si tratta di una funzione, ma di un'istruzione del linguaggio. Dopo <code>unset()</code> , l'esecuzione di <code>empty()</code> o <code>is_null()</code> sulla stessa variabile restituirà vero, mentre <code>isset()</code> restituirà falso.

## Funzioni per gli array

Nome funzione	Descrizione
<code>count</code>	Restituisce un intero che rappresenta il numero di elementi dell'array.
<code>sort</code> <code>rsort</code>	Ordinano gli elementi di un array, la prima in modo crescente, la seconda decrescente. Queste funzioni modificano direttamente l'array passato come argomento che perderà la sua composizione originale. Le chiavi vanno perse: l'array risultato ha chiavi numeriche a partire da 0 secondo il nuovo ordinamento.
<code>in_array(valore, array)</code>	Cerca il valore all'interno dell'array. Restituisce un valore booleano.
<code>array_key_exists(valore, array)</code>	Cerca il valore fra le chiavi (non fra i valori) dell'array. Restituisce un valore booleano.
<code>array_search(valore, array)</code>	Cerca il valore nell'array e ne restituisce la chiave o, se la ricerca non va a buon fine, il valore <code>FALSE</code> .
<code>array_merge(array, array)</code>	Fonde gli elementi di due (o più) array. Gli elementi con indici numerici vengono accodati l'uno all'altro e le chiavi rinumerate. Le chiavi associative invece vengono mantenute e, nel caso vi siano più elementi con la stessa chiave associativa, l'ultimo sostituisce i precedenti. Restituisce l'array risultante dalla fusione.
<code>array_pop(array)</code>	Estrae l'ultimo elemento dell'array da cui viene eliminato. Restituisce l'elemento estratto.
<code>array_push(array, valore)</code>	Accoda il valore indicato all'array. Equivale all'uso dell'istruzione di accodamento <code>\$array[]=\$valore</code> . Restituisce il numero degli elementi dell'array dopo l'accodamento.

## Funzioni per data e ora

Nome funzione	Descrizione
<code>time()</code>	Fornisce il <i>timestamp</i> <sup>4</sup> relativo al momento in cui viene eseguita. Restituisce un valore intero.
<code>checkdate(mese, giorno, anno)</code>	Verifica se i valori forniti costituiscono una data valida. Restituisce un valore booleano.
<code>mktime(ore, minuti, secondi, mese, giorno, anno)</code>	Sulla base dei parametri forniti restituisce un intero che rappresenta un <i>timestamp</i> . Può essere utilizzata per effettuare calcoli sulle date.
<code>date(formato [,timestamp])</code>	<p>Considera il <i>timestamp</i> fornito (se non è indicato usa quello attuale) e fornisce una data formattata secondo le specifiche indicate nel primo parametro di tipo stringa. Per esempio le due istruzioni</p> <pre>\$data = mktime(0,0,0,2,29,2012); echo date('d-m-Y',\$data);</pre> <p>visualizzano "29-2-2012".</p>

### L'operatore @

In PHP il simbolo «@» premesso all'invocazione di una funzione (in generale alla valutazione di un'espressione) impedisce la visualizzazione dei messaggi di errore.

4. Il *timestamp* PHP è il numero di secondi trascorsi dalla mezzanotte del 1 gennaio 1970.

Le specifiche per definire il formato utilizzato nella funzione `date` si basano su caratteri di cui elenchiamo i valori più usati (TABELLA 3).

TABELLA 3

Codice	Descrizione
Y	anno su 4 cifre
y	anno su 2 cifre
n	mese numerico (1-12)
m	mese numerico su 2 cifre (01-12)
F	mese testuale ('January' – 'December')
M	mese testuale su 3 lettere ('Jan' – 'Dec')
d	giorno del mese su due cifre (01-31)
j	giorno del mese (1-31)
w	giorno della settimana, numerico (0=domenica, 6=sabato)
l	giorno della settimana, testuale ('Sunday' – 'Saturday')
D	giorno della settimana su 3 lettere ('Sun' – 'Sat')
H	ora su due cifre (00-23)
G	ora (0-23)
i	minuti su due cifre (00-59)
s	secondi su due cifre (00-59)



## Funzioni per le stringhe

Nome funzione	Descrizione
<code>strlen(stringa)</code>	Restituisce un numero intero che rappresenta la lunghezza della stringa, cioè il numero di caratteri che la compongono.
<code>trim(stringa)</code> <code>ltrim(stringa)</code> <code>rtrim(stringa)</code>	Eliminano gli spazi in una stringa, rispettivamente all'inizio e alla fine, solo all'inizio, solo alla fine. Restituiscono la stringa modificata.
<code>substr(stringa, intero [, intero])</code>	Restituisce una porzione della stringa, in base al secondo parametro (che indica l'inizio della porzione da estrarre), e all'eventuale terzo parametro, che indica quanti caratteri devono essere estratti. Se il terzo parametro non viene indicato, viene restituita tutta la parte finale della stringa a partire dal carattere indicato. Per esempio, con <code>substr('AlfaBeta', 3, 2)</code> si ottiene 'aBe'.
<code>str_replace(stringa, stringa, stringa)</code>	Effettua una sostituzione della prima stringa con la seconda all'interno della terza. Restituisce la terza stringa modificata. Per esempio, con <code>str_replace('p','t','pippo')</code> si ottiene «titto».
<code>strpos(stringa, stringa)</code>	Ricerca la posizione della seconda stringa all'interno della prima. Restituisce un intero che rappresenta la posizione della stringa cercata. Se la seconda stringa non è presente nella prima, restituisce il valore booleano <code>FALSE</code> . Per esempio: <code>strpos('Lorenzo', 're')</code> restituisce 2.
<code>strstr(stringa, stringa)</code>	Ricerca la seconda stringa all'interno della prima, e restituisce la parte della prima stringa a partire dal punto in cui ha trovato la seconda. Se la ricerca non va a buon fine, restituisce il valore booleano <code>FALSE</code> . Per esempio: <code>strstr('Lorenzo', 're')</code> restituisce «renzo».
<code>strtolower(stringa)</code> <code>strtoupper(stringa)</code>	Convertono tutti i caratteri alfabetici nelle corrispondenti lettere minuscole/maiuscole. Restituiscono la stringa modificata.
<code>explode(stringa, stringa [, intero])</code>	Scompone la seconda stringa in un array usando i caratteri della prima come separatori degli elementi. Il terzo parametro può servire a indicare il numero massimo di elementi che l'array deve contenere (se la scomposizione della stringa ne genera un numero maggiore, la parte finale della stringa sarà interamente contenuta nell'ultimo elemento). Restituisce l'array delle sottostringhe risultato della scomposizione. Per esempio: <code>explode(' ', 'ciao Marco')</code> restituisce un array di due elementi in cui il primo è «ciao» e il secondo «Marco».

### 5.1 Funzioni definite dall'utente

Il linguaggio PHP permette di definire funzioni da parte dell'utente mediante la parola chiave **function** (l'eventuale risultato viene restituito ricorrendo alla parola chiave **return**).

#### ESEMPI

La seguente funzione visualizza, quando invocata, il contenuto della variabile passata come parametro.

```
function messaggio($m){  
    echo $m;  
}
```

Le due seguenti istruzioni:

```
$m="Messaggio di prova";  
messaggio($m);
```

visualizzano il testo "Messaggio di prova".

La funzione che segue incrementa di un giorno la data fornita come parametro. La variabile `$gdt` è un array associativo creato con la funzione `getdate` che estrae da `$dt` giorno (`'mday'`), mese (`'mon'`) o anno (`'year'`). Il valore del giorno (`$gdt[mday]`) viene incrementato di 1, quindi la funzione `mktime` crea un *timestamp* che rappresenta la data del giorno successivo.

```
function giornoSuccessivo($dt) {
    $gdt = getdate($dt);
    $gdt['mday'] = $gdt['mday'] + 1;
    return mktime(0, 0, 0, $gdt['mon'], $gdt['mday'], $gdt['year']);
}
```

Le seguenti istruzioni

```
$data = mktime(0,0,0,2,29,2012);
echo date('d-m-Y',giornoSuccessivo($data));
```

dopo aver definito la variabile `$data` assegnandole il valore 29/2/2012 visualizzano «01-03-2012».

La seguente funzione, dati come argomenti una stringa e un carattere separatore, scompone la stringa in sottostringhe delimitate dal separatore memorizzando ognuna di esse in un array.

```
function scomponStringa($stringa,$sep){
    $arr=array();
    $temp=strtok($stringa,$sep);
    while($temp)
    {
        $arr[]=$temp;
        $temp=strtok($sep);
    }
    return $arr;
}
```

La funzione `strtok` ogni volta che viene invocata restituisce la sottostringa a sinistra del primo carattere separatore che incontra e lascia nella variabile `$stringa` la parte destra della stringa iniziale; se il separatore non viene incontrato restituisce la stringa intera e lascia la variabile `$stringa` vuota.

Con le seguenti istruzioni

```
$s="uno^tre^cinque^sette";
$elementi=scomponStringa($s,'^');
foreach ($elementi as $e) {
    echo $e."<br>";
}
```

si ottiene la seguente visualizzazione:

```
uno
tre
cinque
sette
```

## 5.2 Passaggio di parametri alle funzioni

Il passaggio di parametri alle funzioni avviene normalmente per valore. Il passaggio di variabili per riferimento è possibile utilizzando la notazione tipica del linguaggio C++, cioè premettendo al nome del parametro il simbolo «&».

### ESEMPIO

La seguente funzione PHP calcola nei parametri `x1` e `x2` passati per riferimento le radici dell'equazione di secondo grado di coefficienti `a`, `b` e `c` passati per valore, restituendo il valore vero in caso di successo, falso in caso di insuccesso (la funzione `sqrt` restituisce la radice quadrata dell'argomento):

```
function radici($a, $b, $c, &$x1, &$x2) {  
    $d = $b*$b - 4*$a*$c;  
    if (($d < 0) or ($a == 0))  
        return FALSE;  
    else {  
        $x1 = (-$b + sqrt($d)) / (2*$a);  
        $x2 = (-$b - sqrt($d)) / (2*$a);  
        return TRUE;  
    }  
}
```

È possibile definire funzioni per le quali non è obbligatorio che tutti gli argomenti previsti siano passati al momento della chiamata. Infatti, se nella firma della funzione si specifica un valore di default per un certo parametro, quel parametro è facoltativo nella chiamata della funzione e, se non viene specificato, la funzione impiega il relativo valore di default.

### ESEMPIO

Consideriamo la seguente funzione che visualizza i dati anagrafici di una persona:

```
function anagrafica($nome, $indirizzo, $CF='non rilevato') {  
    echo "Nome: $nome<br>";  
    echo "Indirizzo: $indirizzo<br>";  
    echo "Codice fiscale: $cf<br>";  
}
```

Per il terzo parametro della funzione è previsto un valore di default (la stringa «non rilevato») il quale verrà assegnato alla variabile `$CF` se all'atto dell'invocazione della funzione saranno specificati solo i primi due argomenti. Se vengono forniti tutti e tre gli argomenti, il valore di default viene ignorato. Vediamo due esempi di chiamata della funzione:

```
anagrafica('Mario Rossi', 'via Roma 2', 'RSSMRA69B01H501Y');  
anagrafica('Paolo Bianchi', 'via Milano 9');
```

Nel primo caso otterremo la seguente visualizzazione:

```
Nome: Mario Rossi  
Indirizzo: via Roma 2  
Codice fiscale: RSSMRA69B01H501Y
```

Nel secondo caso invece viene impiegato il valore di default del parametro:

```
Nome: Paolo Bianchi  
Indirizzo: via Milano 9  
Codice fiscale: non rilevato
```

### 5.3 Visibilità e *lifetime* delle variabili

Le variabili definite e utilizzate all'interno di una funzione sono visibili solo all'interno della funzione e cessano di esistere fuori dal contesto di esecuzione della stessa. Eventuali variabili definite all'interno di una funzione con lo stesso nome di una variabile già definita esternamente, «nascondono» la variabile esterna fino a quando la funzione non termina.

Premettendo, all'interno di una funzione, la parola chiave `global` al nome di una variabile, si riferenzia una variabile omonima definita esternamente.

#### ESEMPIO

L'esecuzione del seguente codice

```
function test(){  
    $alfa="tre";  
    echo $alfa."<br>";  
    global $alfa;  
    echo $alfa."<br>";  
}
```

```
$alfa="uno";  
test();
```

visualizza:

```
tre  
uno
```

La variabile `$alfa` interna alla funzione nasconde la variabile omonima esterna. Successivamente la dichiarazione `global $alfa;` rende visibile la variabile esterna all'interno della funzione.

## 6 La gestione di *form* HTML con il linguaggio PHP; validazione dell'input e passaggio di dati tra pagine web

Generalmente in una *form* HTML viene indicato l'indirizzo di una pagina in grado di elaborare i dati inseriti tramite la *form* stessa. L'URL può essere relativo a una pagina che comprende uno script che legge i valori inseriti dall'utente, li elabora e produce una pagina web come risultato. La FIGURA 2 schematizza l'interazione tra una *form* HTML e uno script di gestione dei dati.

In questo paragrafo vedremo come da una pagina web sia possibile richiamare un'altra effettuando contestualmente un passaggio di dati tra esse e come validare gli input forniti dall'utente.

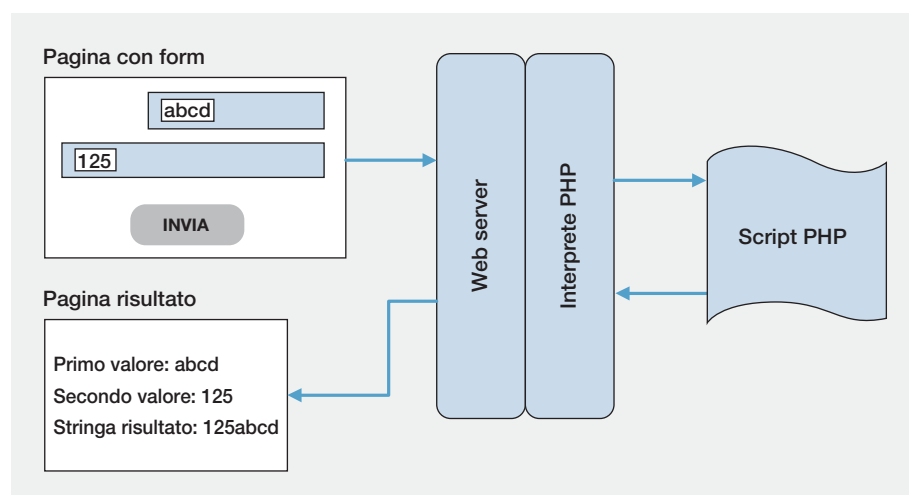


FIGURA 2

### 6.1 Interazione tra pagine web

La pagina web a fianco, denominata «DatiPersonal.html», è codificata interamente in linguaggio HTML e consente all'utente di inserire, secondo varie modalità (campi di testo, pulsanti mutuamente esclusivi, opzioni combinate, elenchi a scorrimento, ecc.) i dati richiesti e di inviarli a una pagina web che comprende codice PHP denominata «DatiPersonal.php»:

```
<html>
<head>
  <title>Richiesta dati personali</title>
</head>
```



```
<body>  
  <form method="POST" action="DatiPersonalizzati.php">  
    Cognome: <input type="text" size="12" maxlength="24"  
      name="cognome"><br>  
    Nome:               <input type="text" size="12"  
      maxlength="24"  
      name="nome"><br><br>  
  
    Sesso: <br>  
    Maschio:      <input type="radio" value="maschio"  
      name="sesso"><br>  
    Femmina: <input type="radio"  
      value="femmina" name="sesso"><br><br>  
    Scegli i cibi che preferisci:<br>  
    Bistecca: <input type="checkbox" value="bistecca"  
      name="cibo[]"><br>  
    Pizza:     <input type="checkbox"  
      value="pizza"  
      name="cibo[]"><br>  
    Pollo:     <input type="checkbox"  
      value="pollo"  
      name="cibo[]"><br><br>  
  
    <textarea rows="5" cols="20" name="citazione">  
      Inserisci la tua citazione preferita!  
    </textarea><br><br>  
    Seleziona il tuo livello di istruzione scolastica:<br>  
    <select name="istruzione">  
      <option value="scuola media">Medie</option>  
      <option value="scuola superiore">Superiori</option>  
      <option value="università">Studi universitari</option>  
    </select><br><br>  
    Seleziona la parte del giorno che preferisci:<br>  
    <select name="partegiorno" size="3">  
      <option value="mattino">Mattino</option>  
      <option value="pomeriggio">Pomeriggio</option>  
      <option value="notte">Notte</option>  
    </select><br><br>  
    <input type="reset" value="Reset">  
    <input type="submit" value="Ok">  
  </form>  
</body>  
</html>
```

Il tag HTML che consente di predisporre una pagina per l'inserimento di dati da parte dell'utente (*form*) è **form**, che consente di specificare l'attributo *method* per scegliere tra la modalità GET (inserimento dei valori nell'URL come stringa di interrogazione) o POST (invio dei valori mediante le funzionalità del protocollo HTTP) di inoltre dei valori inseriti/selezionati dall'utente alla pagina di destinazione specificata dall'attributo *action*.

All'interno del tag **form** è possibile utilizzare i tag **input**, **textarea** e **select/option** per realizzare rispettivamente campi di testo e scelte alternative (sia mutuamente esclusive sia multiple), aree di testo ed elenchi di voci.

**OSSERVAZIONE** Per questi tag HTML si specifica quasi sempre l'attributo *name*, che consente di definire il nome a cui viene associato il valore inserito o selezionato dall'utente: nel caso che la pagina di destinazione contenga codice PHP i nomi definiti sono le chiavi per gli array associativi superglobali `$_GET` o `$_POST`, a seconda del metodo di inoltro prescelto. Se è possibile una selezione multipla di valori, per l'attributo *name* può essere specificato un array utilizzando la coppia di simboli «[]». L'attributo *name* può non essere specificato se non si è interessati alla scelta effettuata dall'utente nella pagina di destinazione.

Il tag **input** non prevede tag di chiusura ed è caratterizzato dall'attributo *type* di cui nella TABELLA 4 si riportano i valori più comuni.

TABELLA 4

button	Visualizza un pulsante normalmente utilizzato per l'invocazione di uno script di codice eseguito lato client.
checkbox	Visualizza una casella per la selezione del valore indicato nell'attributo <i>value</i> ; più elementi di questo tipo aventi lo stesso valore dell'attributo <i>name</i> specificato come array consentono la selezione multipla di più valori.
radio	Visualizza un <i>radio-button</i> per la selezione del valore indicato nell'attributo <i>value</i> : se esistono più elementi di questo tipo aventi lo stesso valore dell'attributo <i>name</i> , la selezione è mutuamente esclusiva.
reset	Visualizza un pulsante il cui effetto è quello di ripristinare i valori iniziali di tutti i componenti della <i>form</i> .
submit	Visualizza un pulsante il cui effetto è quello di richiedere al server la pagina specificata nell'attributo <i>action</i> della <i>form</i> passando i valori inseriti/selezionati dall'utente.
text	Visualizza un campo di testo; è possibile specificare l'attributo <i>size</i> per indicarne la dimensione espressa in caratteri e l'attributo <i>maxlength</i> per specificare il numero massimo di caratteri digitabili

Il tag **textarea** visualizza un'area le cui dimensioni espresse in righe e colonne di caratteri sono definite dagli attributi *rows* e *cols* all'interno della quale è possibile digitare testo.

Il tag **select**, per il quale l'attributo *size* definisce il numero di voci visibili, visualizza un elenco di opzioni tra le quali è possibile effettuare una selezione: le singole opzioni sono specificate con il tag **option**, il cui attributo *value* identifica il valore inviato alla pagina di destinazione.

**OSSERVAZIONE** Per inizializzare gli elementi che compongono una *form* è possibile utilizzare l'attributo *value* per i tag **input** di tipo *text* e l'attributo *checked* impostato a «true» per i tag **input** di tipo *checkbox* e *radio*.

Se la pagina di destinazione «DatiPersonali.php» è la seguente:

```
<html>
<head>
  <title>Visualizzazione dati personali</title>
</head>
<body>
  <?php
    $nome = $_POST["cognome"];
    $cognome = $_POST["nome"];
    $sesso = $_POST["sesso"];
    $cibo = $_POST["cibo"]; // array
    $citazione = $_POST["citazione"];
    $istruzione = $_POST["istruzione"];
    $partegiorno = $_POST["partegiorno"];

    echo "Ciao, $cognome $nome<br>";
    echo "Sei $sesso e ti piace mangiare:<br><br>";
    foreach ($cibo as $c) {
      echo "$c<br>";
    }
    echo "<br>La tua citazione preferita &grave:<br>";
    echo "<em>$citazione</em><br><br>";
    echo "Ti senti pi&ugrave a tuo agio di: $partegiorno
      ed il tuo livello di istruzione &grave:
      $istruzione<br>";

  ?>
</body>
</html>
```



la pressione del pulsante «Ok» della pagina contenente la *form* dopo averla compilata nel seguente modo



carica la seguente pagina dinamica creata dallo script PHP:



I dati inseriti dall'utente nella *form* sono trasferiti utilizzando il metodo POST del protocollo HTTP al server dove lo script PHP li recupera dall'array superglobale `$_POST`, utilizzando i nomi specificati come attributi *name* negli elementi della *form*.

**OSSERVAZIONE** `$_POST["cibo"]` è in realtà un array che contiene i cibi selezionati dall'utente nella *form*: questo consente la selezione multipla.

La pagina visualizzata è il risultato dell'elaborazione dello script PHP da parte del server; l'analisi del codice HTML ricevuto dal browser produce il seguente risultato:

```
<html>
  <head>
    <title>Visualizzazione dati personali</title>
  </head>
  <body>
    Ciao, James Bond<br>
    sei maschio e ti piace mangiare:<br>
    <br>
    bistecca<br>
    pollo<br>
    <br>
    la tua citazione preferita &grave;:<br>
    <em>Agitato, non mescolato!</em><br>
    <br>
    ti senti più a tuo agio di: notte ed il tuo livello
    di istruzione &grave;: università<br>
  </body>
</html>
```

**OSSERVAZIONE** Dato che l'input fornito dall'utente diviene parte del codice della pagina dinamica, può accadere che la codifica non sia coerente con le convenzioni del linguaggio HTML relativamente ai caratteri speciali: la funzione `htmlspecialchars` del linguaggio PHP ha esattamente lo scopo di effettuare la trasformazione di una stringa di testo in una stringa coerente con le convenzioni HTML. Il codice della pagina «Dati Personali.php» dovrebbe essere più correttamente il seguente:



```
<html>
<head>
  <title>Visualizzazione dati personali</title>
</head>
<body>
  <?php
    $nome = htmlentities($_POST["cognome"], ENT_HTML5,
                        'ISO-8859-1');
    $cognome = htmlentities($_POST["nome"], ENT_HTML5,
                          'ISO-8859-1');
    $sesso = htmlentities($_POST["sesso"], ENT_HTML5,
                        'ISO-8859-1');

    $cibo = $_POST["cibo"]; // array
    $citazione = htmlentities($_POST["citazione"],
                          ENT_HTML5, 'ISO-8859-1');
    $istruzione = htmlentities($_POST["istruzione"],
                          ENT_HTML5, 'ISO-8859-1');
    $partegiorno = htmlentities($_POST["partegiorno"],
                          ENT_HTML5, 'ISO-8859-1');

    echo "Ciao, $cognome $nome<br>";
    echo "Sei $sesso e ti piace mangiare:<br><br>";
    foreach ($cibo as $c) {
      echo htmlentities($c, ENT_HTML5,
                      'ISO-8859-1')."<br>";
    }
    echo "<br>La tua citazione preferita &grave:<br>";
    echo "<em>$citazione</em><br><br>";
    echo "Ti senti pi&ugrave a tuo agio di: $partegiorno ed il tuo livello di istruzione &grave: $istruzione<br>";

  ?>
</body>
</html>
```

Il parametro `ENT_HTML5` specifica che la conversione deve essere effettuata secondo le convenzioni della versione 5 del linguaggio HTML, mentre il parametro `ISO-8859-1` specifica che la codifica delle stringhe segue lo standard «Western European – Latin 1», che comprende le lingue inglese, italiana, francese, spagnola e tedesca.

**OSSERVAZIONE** Nell'esempio precedente la funzione `htmlentities` viene applicata direttamente alla ricezione dei dati, in quanto questi non sono elaborati dal codice PHP, ma solo inseriti nel codice HTML generato. Nel caso più comune in cui i dati debbano essere elaborati dallo script PHP, l'applicazione della funzione `htmlentities` deve avvenire solo sulle stringhe di testo che costituiscono l'output dello script.

Specificando nel tag **form** il valore GET per l'attributo *method*, il passaggio dei valori inseriti/selezionati dall'utente avviene in modo visibile aggiungendo una stringa di interrogazione all'URL di richiesta della pagina:

```
http://localhost/DatiPersonali.php?cognome=Bond&nome=James&
sesso=maschio&cibo%5B%5D=bistecca&cibo%5B%5D=pollo&
citazione=Agitato%2C+non+mescolato%21%0D%0A%09%09%09&
istruzione=universit%E0&partegiorno=notte
```

**OSSERVAZIONE** I valori inseriti/selezionati dall'utente sono implicitamente trasformati nella stringa di interrogazione secondo le convenzioni URL che prevedono la codifica esadecimale dei caratteri non alfanumerici preceduta dal simbolo «%». Questo tipo di codifica può essere effettuato esplicitamente utilizzando la funzione PHP `urlencode`; in questo caso la decodifica deve avvenire esplicitamente utilizzando la funzione `urldecode`.

Naturalmente utilizzando il metodo GET lo script PHP deve essere modificato per recuperare i valori dall'array superglobale `$_GET` anziché `$_POST`:

```
...
$nome = $_GET["cognome"];
$cognome = $_GET["nome"];
$sesso = $_GET["sesso"];
$cibo = $_GET["cibo"]; // array
$citazione = $_GET["citazione"];
$istruzione = $_GET["istruzione"];
$partegiorno = $_GET["partegiorno"];
...
```

**OSSERVAZIONE** Utilizzando nello script PHP l'array superglobale `$_REQUEST` non si ha la necessità di modificarlo per il cambiamento del metodo di inoltro dei dati.

Nel caso che l'utente non effettui alcune selezioni nella pagina che visualizza la *form*, la pagina generata dinamicamente contiene alcuni errori dovuti alla mancata definizione delle chiavi associative dell'array superglobale `$_POST` o `$_GET` (o `$_REQUEST`):



Allo scopo di evitare situazioni di errore dovute al mancato inserimento di dati da parte dell'utente in una pagina contenente una *form*, è importante utilizzare le funzioni PHP per variabili, array e stringhe al fine di validare l'input ricevuto.



## ESEMPIO

Il seguente script PHP valida l'input ricevuto dalla pagina contenente la *form* prima del suo uso; in caso di dati mancanti termina l'esecuzione visualizzando un messaggio di errore:

```
<html>
  <head>
    <title>Visualizzazione dati personali</title>
  </head>
  <body>
    <?php
      if (!isset($_POST["cognome"]) ||
          !isset($_POST["nome"]) ||
          !isset($_POST["sesso"]) ||
          !isset($_POST["cibo"]) ||
          !isset($_POST["citazione"]) ||
          !isset($_POST["istruzione"]) ||
          !isset($_POST["partegiorno"]))
        die("Dati personali non corretti.");

      $nome = htmlentities($_POST["cognome"], ENT_HTML5, 'ISO-8859-1');
      $cognome = htmlentities($_POST["nome"], ENT_HTML5, 'ISO-8859-1');
      $sesso = htmlentities($_POST["sesso"], ENT_HTML5, 'ISO-8859-1');
      $cibo = $_POST["cibo"]; // array
      $citazione = htmlentities($_POST["citazione"], ENT_HTML5, 'ISO-8859-1');
      $istruzione = htmlentities($_POST["istruzione"], ENT_HTML5, 'ISO-8859-1');
      $partegiorno = htmlentities($_POST["partegiorno"], ENT_HTML5, 'ISO-8859-1');
      if (strlen($nome) == 0 ||
          strlen($cognome) == 0 ||
          count($cibo) == 0 ||
          strlen($citazione) == 0)
        die("Dati personali non completi.");

      echo "Ciao, $cognome $nome<br>";
      echo "Sei $sesso e ti piace mangiare:<br><br>";
      foreach ($cibo as $c) {
        echo htmlentities($c, ENT_HTML5, 'ISO-8859-1')."<br>";
      }
      echo "<br>La tua citazione preferita &egrave:<br>";
      echo "<em>$citazione</em><br><br>";
      echo "Ti senti pi&ugrave; a tuo agio di: $partegiorno ed il tuo livello di istru-
        zione &egrave: $istruzione<br>";
    ?>
  </body>
</html>
```

È pratica comune da parte dei programmatori PHP strutturare una pagina in modo che richiami se stessa con modalità diverse in funzione del fatto che riceva o meno dati. Il codice che segue integra in un'unica pagina la *form* HTML per l'inserimento dei dati da parte dell'utente e lo script PHP che li visualizza:



```
<html>
<head>
  <title>Dati personali</title>
</head>
<body>
  <?php
    if (!isset($_POST["invio"])) { // verifica invio dati
    ?>
    <form method="POST" action="DatiPersonali.php">
      Cognome: <input type="text" size="12" maxlength="24"
        name="cognome"><br>
      Nome:      <input type="text" size="12"
        maxlength="24"
        name="nome"><br><br>
      Sesso: <br>
      Maschio:      <input type="radio" value="maschio"
        name="sesso"><br>
      Femmina: <input type="radio" value="femmina"
        name="sesso"><br><br>
      Scegli i cibi che preferisci:<br>
      Bistecca: <input type="checkbox" value="bistecca"
        name="cibo[]"><br>
      Pizza:      <input type="checkbox"
        value="pizza"
        name="cibo[]"><br>
      Pollo:      <input type="checkbox"
        value="pollo"
        name="cibo[]"><br><br>
      <textarea rows="5" cols="20" name="citazione">
        Inserisci la tua citazione preferita!
      </textarea><br><br>
      Seleziona il tuo livello di istruzione scolastica:<br>
      <select name="istruzione">
        <option value="scuola media">Medie</option>
        <option value="scuola superiore">Superiori</option>
        <option value="università">Studi universitari</option>
      </select><br><br>
      Seleziona la parte del giorno che preferisci:<br>
      <select name="partegiorno" size="3">
        <option value="mattino">Mattino</option>
        <option value="pomeriggio">Pomeriggio</option>
        <option value="notte">Notte</option>
```

## Sicurezza e controllo dell'input

Uno stretto controllo sulla validità e sul contenuto dei dati ricevuti è essenziale ai fini della sicurezza del sito web costituito dalle pagine dinamiche create con script in linguaggio PHP.

In particolare una tecnica di attacco molto comune, denominata *code injection* (nel caso di script in linguaggio PHP comunemente nota come *cross-site scripting* o *SQL injection* a seconda che il codice inserito sia in linguaggio JavaScript o SQL), prevede l'inserimento di codice nelle stringhe fornite come input, codice che potrà successivamente essere eseguito da un DBMS cui il sito si interfaccia, o inserito nella pagina dinamica restituita al browser.

La migliore difesa che il programmatore PHP può opporre a questa tecnica di attacco è il controllo sistematico delle stringhe ricevute in input al fine di verificarne la congruità (formato, dimensione, ...) con le informazioni attese.

```

</select><br><br>
<input type="reset" value="Reset">
<input type="submit" value="Ok" name="invio">
</form>
<?php
}
else {
    if (!isset($_POST["cognome"]) ||
        !isset($_POST["nome"]) ||
        !isset($_POST["sesso"]) ||
        !isset($_POST["cibo"]) ||
        !isset($_POST["citazione"]) ||
        !isset($_POST["istruzione"]) ||
        !isset($_POST["partegiorno"]))
        die("Dati personali non corretti.");

    $nome = htmlentities($_POST["cognome"], ENT_HTML5,
        'ISO-8859-1');
    $cognome = htmlentities($_POST["nome"], ENT_HTML5,
        'ISO-8859-1');
    $sesso = htmlentities($_POST["sesso"], ENT_HTML5,
        'ISO-8859-1');
    $cibo = $_POST["cibo"]; // array
    $citazione = htmlentities($_POST["citazione"], ENT_HTML5,
        'ISO-8859-1');
    $istruzione = htmlentities($_POST["istruzione"],
        ENT_HTML5, 'ISO-8859-1');
    $partegiorno = htmlentities($_POST["partegiorno"],
        ENT_HTML5, 'ISO-8859-1');

    if (strlen($nome) == 0 ||
        strlen($cognome) == 0 ||
        count($cibo) == 0 ||
        strlen($citazione) == 0)
        die("Dati personali non completi.");

    echo "Ciao, $cognome $nome<br>";
    echo "Sei $sesso e ti piace mangiare:<br><br>";
    foreach ($cibo as $c) {
        echo htmlentities($c, ENT_HTML5, 'ISO-8859-1')."<br>";
    }
    echo "<br>La tua citazione preferita &grave:<br>";
    echo "<em>$citazione</em><br><br>";
    echo "Ti senti pi&ugrave a tuo agio di: $partegiorno ed il tuo
        livello di istruzione &grave: $istruzione<br>";
}
?>
</body>
</html>

```

**OSSERVAZIONE** Il codice della pagina è strutturato in due sezioni: la prima contiene il codice HTML per la generazione della *form*, mentre la seconda contiene lo script PHP per la ricezione, la verifica e la visualizzazione dei dati. Il costrutto condizionale PHP che seleziona la sezione da elaborare e inviare al client verifica la presenza tra i dati ricevuti del nome associato al pulsante di inoltro dei dati della *form* («invio»): se risulta definito, la *form* è stata inoltrata e viene elaborata la sezione PHP; in caso contrario significa che la *form* non è stata ancora inoltrata e viene elaborata la sezione HTML che la visualizza.

La seguente *form* è funzionalmente analoga a quella presentata in precedenza, ma viene visualizzata con un aspetto più piacevole:

Richiesta dati personali

localhost/dati\_personali.html

Inserisci i tuoi dati anagrafici:

Cognome:

Nome:

Sesso: ☐ Maschio ☐ Femmina

Scegli i cibi che preferisci:

Bistecca: ☐

Pizza: ☐

Pollo: ☐

Inserisci la tua citazione preferita!

Seleziona il tuo livello di istruzione scolastica:

Media

Seleziona la parte del giorno che preferisci:

Mattino

Pomeriggio

Notte

Cancella dati:

Invia dati:

Le aree che comprendono uno o più elementi di inserimento/selezione dei dati sono state ottenute con l'uso del tag `fieldset`, mentre le frasi che ne identificano la funzione sono visualizzate ricorrendo al tag `legend`, interno al tag `fieldset` cui si riferisce:



```
<html>  
  <head>  
    <title>Richiesta dati personali</title>  
  </head>  
  <body>  
    <form method="POST" action="DatiPersonali.php">  
      <fieldset>  
        <legend>Inserisci i tuoi dati anagrafici:</legend>  
        Cognome: <input type="text" size="12" maxlength="24"  
          name="cognome"><br>  
        Nome:   <input type="text" size="12"  
          maxlength="24"  
          name="nome"><br><br>
```

```
Sesso: <br>
Maschio: <input type="radio" value="maschio"
        name="sesso"><br>
Femmina: <input type="radio" value="femmina"
         name="sesso"><br>
</fieldset><br>
<fieldset>
    <legend>Scegli i cibi che preferisci:</legend>
Bistecca: <input type="checkbox" value="bistecca"
           name="cibo[]"><br>
Pizza:      <input type="checkbox" value="pizza"
            name="cibo[]"><br>
Pollo:      <input type="checkbox" value="pollo"
            name="cibo[]"><br>
</fieldset><br>
<textarea rows="5" cols="20" name="citazione">
    Inserisci la tua citazione preferita!
</textarea><br><br>
<fieldset>
    <legend>
        Seleziona il tuo livello di istruzione scolastica:
    </legend>
    <select name="istruzione">
        <option value="scuola media">Medie</option>
        <option value="scuola superiore">Superiori</option>
        <option value="università">Studi universitari</option>
    </select><br>
</fieldset><br>
<fieldset>
    <legend>Seleziona la parte del giorno che preferisci:
    </legend>
    <select name="partegiorno" size="3">
        <option value="mattino">Mattino</option>
        <option value="pomeriggio">Pomeriggio</option>
        <option value="notte">Notte</option>
    </select><br>
</fieldset><br>
<label for="reset">Cancella campi:</label>
<input type="reset" value="Reset" id="reset"><br>
<label for="submit">Invia dati:</label>
<input type="submit" value="Ok" id="submit">
</form>
</body>
</html>
```



**OSSERVAZIONE** Le stringhe «Cancella dati» e «Invia dati» sono state associate rispettivamente ai pulsanti di *reset* e *submit* utilizzando il tag `label`. A questo scopo per entrambi i tag `input` relativi ai pulsanti è stato specificato l'attributo *id* riferito dall'attributo *for* del corrispondente tag `label`.

Esistono altri modi per richiamare una pagina dal codice di un'altra pagina.

Una possibilità è quella di inserire il riferimento a una pagina HTML o a uno script PHP all'interno di un tag che realizza un link.

**ESEMPIO**

La seguente istruzione:

```
echo "<a href='cerca_ordine.php'>Cerca ordine</a>";
```

ha come effetto quello di realizzare nella pagina web dinamica in fase di creazione un collegamento ipertestuale sulla stringa "Cerca ordine" che attiva lo script PHP

```
"cerca_ordine.php"
```

**OSSERVAZIONE** Il codice

```
<a href="elenco_commesse.php?idOrdine=<?php echo $idOrdine; ?>">
<?php echo ($idOrdine); ?></a>
```

realizza nella pagina web dinamica in fase di creazione un link sulla stringa che mostra un numero di ordine contenuto nella variabile `$idOrdine` che attiva lo script PHP "elenco\_commesse.php" con il passaggio contestuale del numero dell'ordine stesso. Infatti con la stringa:

```
"elenco_commesse.php?idOrdine=<?php echo $idOrdine; ?>"
```

viene creato un URL che comprende una stringa di interrogazione: supponendo che il numero d'ordine sia 105 l'URL diviene «elenco\_commesse.php?idOrdine=105»). Nello script "elenco\_commesse.php" è possibile accedere al numero dell'ordine facendo riferimento all'elemento `$_GET['idOrdine']`.

**OSSERVAZIONE**

...  
...  
...

Questa è la tipica situazione in cui una buona pratica di programmazione richiede di utilizzare la funzione PHP `urlencode`:

```
<a href="elenco_commesse.php?idOrdine=<?php echo
urlencode($idOrdine); ?>"> <?php echo ($idOrdine); ?></a>
```

Anche il comando PHP `header` può essere utilizzato per passare il controllo a un'altra pagina. Esso assume particolare importanza in quanto permette di aggiungere a quelli predefiniti *header* addizionali alla risposta HTTP da rendere al browser insieme (o in sostituzione) al risultato dello script.

Il comando `header` di PHP consente di far inviare dal server delle «risposte» (*response headers*) relative a una determinata pagina web.

**OSSERVAZIONE** L'utilizzo di `header` richiede attenzione in quanto lo script non deve generare alcun output prima della sua esecuzione, altrimenti viene generato un errore.

Una delle principali funzioni che è possibile realizzare con `header` è il *redirect*: con questa tecnica è possibile attivare un collegamento direttamente mediante il linguaggio PHP. Un *redirect* eseguito con PHP consente di eseguire uno script prima di richiedere al server l'URL di una nuova pagina.

Lo header HTTP che consente di effettuare un redirect è «Location».

Per evitare di generare output dopo `header` può essere opportuno interrompere l'esecuzione del codice con il comando `exit`.

## HTTP header

Il browser (client) durante la navigazione invia delle «richieste» e il server ritorna delle «risposte». Queste richieste e queste risposte sono definite *header* e contengono al loro interno informazioni, alcune delle quali sono essenziali per consentire la regolare navigazione, mentre altre sono puramente informative.

### ESEMPI

```
...
<?php
header("Location: elenco_ordini.php");
exit;
?>
...

...
<?php
header('Refresh: 5; url=http://www.miosito.it/');
exit;
?>
...
```

## 6.2 La gestione e validazione degli input nelle pagine web

In generale la realizzazione di una *form* si deve fissare la propria attenzione su due aspetti, quello relativo alla facilità dell'inserimento dei dati e quello della loro conformità e validazione.

Per esempio potrebbe essere necessario controllare che in un campo «sito web» sia stata inserita una stringa corrispondente a un URL, oppure che in un campo «età» vi sia effettivamente un numero, senza decimali e dal valore inferiore a 110.

L'inserimento di una data potrà essere facilitato dall'uso di un calendario a comparsa attivato dal *focus* del campo, la scelta di un colore sarà più age-

vole potendo effettuare la selezione da una tavolozza anziché richiederne il poco intuitivo corrispondente valore esadecimale. Tradizionalmente queste attività sono gestite con l'utilizzo di codice lato client, tipicamente utilizzando JavaScript.

**OSSERVAZIONE** I controlli di coerenza, oltre che per mezzo di script lato client, dovrebbero sempre essere effettuati anche lato server.

Il linguaggio HTML offre agli sviluppatori specifiche categorie di *input type* (valori numerici, indirizzi e-mail, date, ...) dove il controllo formale viene eseguito direttamente lato client.

**OSSERVAZIONE** Il ricorso a questi tipi di input è estremamente interessante quando utilizzati su dispositivi mobili, come smartphone e tablet. Essi infatti presentano una tastiera touch-screen la cui configurazione si modifica dinamicamente in funzione del tipo di campo di input richiesto. Ad esempio un campo che ammette solo un valore numerico farà aprire sul dispositivo mobile una tastiera virtuale solo numerica.

## Campi di input speciali

Nel seguito sono elencati alcuni tipi di input che generano campi speciali.

### ■ Indirizzi di posta elettronica

E-mail: `<input type="email" name="email_utente">`

Il browser verifica il corretto formato dell'indirizzo di posta elettronica.

### ■ Indirizzi web

Sito internet: `<input type="url" name="web_utente">`

Il browser verifica il corretto formato dell'URL digitato.

### ■ Numeri

Voto: `<input type="number" name="voto" min="0" max="10">`

Questo tipo di input ha la possibilità di impostare gli attributi *min*, *max* e *step*. Gli attributi *min*, *max* indicano il minimo e il massimo dell'intervallo di valori ammissibili, mentre *step* indica il passo dell'incremento o decremento del valore del campo impostabile per mezzo dei tasti +/- presenti nel campo visualizzato.

### ■ Intervalli numerici

Voto: `<input type="range" name="voto" min="1" max="10">`

Questo tipo di input ha gli stessi attributi del tipo numerico, si differenzia dal precedente per la visualizzazione grafica in quanto viene rappresentato come una barra scorrevole.

## ■ Date

Data di nascita: `<input type="date" name="data_nascita">`

Il browser visualizza un calendario per la selezione della data, in alternativa è possibile inserire una stringa che rappresenta la data.

## ■ Numeri telefonici

Telefono: `<input type="tel" name="telefono">`

Il browser verifica il corretto formato del numero telefonico; in alcuni casi visualizza una tastiera telefonica per la composizione.

## ■ Colori

Colore preferito: `<input type="color" name="colore_preferito">`

Il browser visualizza una tavolozza per la selezione del colore che viene rappresentato da una stringa con i valori esadecimali dei colori fondamentali Rosso, Verde e Blu (RGB: Red, Green e Blue).

## ■ Password

Passowrd: `<input type="password" name="password">`

Il browser non visualizza i caratteri digitati sostituendoli con un simbolo generico come «\*».

## Attributi per campi di input

I tipi di input descritti evitano, con le loro funzionalità, di dover scrivere script di controllo e permettono un'interazione migliore con l'utente. Con essi sono stati introdotti alcuni attributi utilizzabili contestualmente che possono essere classificati come segue (TABELLA 5).

TABELLA 5

Tipologia attributi	Descrizione
Booleani	Sono attributi che possono assumere il valore true/false.
Enumerati	Questi attributi possono assumere un insieme finito di valori. Il valore nullo è accettato e indicato con la stringa vuota ("").
Generici	Questi attributi possono assumere un qualsiasi valore.

Nel seguito sono descritti alcuni attributi utili per la gestione di campi di input e la loro validazione.

## ■ autofocus

L'attributo *autofocus* è booleano e consente di impostare il «focus» su uno specifico elemento della *form* al momento del caricamento di una pagina web. Un esempio classico è quello della *home page* di Google: il focus è automaticamente impostato sul campo per la ricerca. Ovviamente un solo elemento per pagina può avere l'attributo autofocus.

Nella seguente *form*

```
<form action="..." method="get">
  Nome utente: <br>
    <input type="text" name="nome_utente" autofocus>
    <input type="submit" value="Invio">
</form>
```

viene impostato il focus sul campo di input.

### ■ *placeholder*

Il valore dell'attributo *placeholder* è visualizzato all'interno di un campo di input, o di un'area di testo, fino a quando il campo è vuoto e non assume il focus.

Esso dovrebbe essere inizializzato con un valore ammissibile per il campo di input, ovvero, contenere un esempio del tipo di valore che l'utente deve scrivere nel campo. È bene evitare usi impropri, come quello di sostituzione della label che descrive il tipo di dato richiesto dal campo.

Nella seguente *form*

```
<form method="post" action="...">
  Parola chiave:
    <input type="text" placeholder="articolo, sezione, ..."
      name="parola_chiave" maxlength="50">
    <input type="submit" value="Invio">
</form>
```

il campo di input previsto per una ricerca bibliografica presuppone l'inserimento del titolo di un articolo di una sezione, ecc.

### ■ *form*

Questo attributo serve per specificare a quale *form* il campo di input fa riferimento. Richiede come valore l'*id* della *form* a cui si riferisce.

Nella seguente *form*

```
<form action="..." method="get" id="id_form">
  <input type="text" name="nome_utente">
  <input type="submit" value="Invio">
</form>
<input type="text" name="cognome_utente" form="id_form">
```

nonostante l'input *cognome\_utente* sia esterno alla *form id\_form* premendo "Invio" si invia anche il dato contenuto nel campo *cognome\_utente* dal momento che impostando l'attributo *form* al valore si specifica l'appartenenza dell'input alla *form*.

## ■ *required*

È un attributo booleano e serve a rendere obbligatoria la compilazione dell'elemento a cui è applicato. La condizione viene valutata all'invio della *form*.

### ESEMPIO

```
<form method="post" action="...">
  Commento:
  <textarea name="commento"
    placeholder="Scrivi qui il tuo commento (max 300 caratteri)"
    maxlength="300" required>
  </textarea>
  <input type="reset" value="Reset">
  <input type="submit" value="Invio">
</form>
```

## ■ *title*

L'attributo *title* associa un *tooltip* informativo a un campo di input che viene visualizzato quando il puntatore del mouse si trova in corrispondenza del campo stesso.

### ESEMPIO

```
Password: <input type="password" name="password"
  title="La password deve avere almeno 8 caratteri
  di cui almeno 2 di tipo numerico.">
```

## ■ *autocomplete*

Questo attributo consente il completamento automatico di un campo da parte del browser: quando l'utente inizia a digitare il valore di un campo di input viene visualizzato un box con una lista di suggerimenti. Questa lista viene aggiornata automaticamente in funzione dei valori inseriti dall'utente nel contesto della sua interazione con le pagine web e memorizzata sul computer client in forma crittografata.

Tale funzionalità a volte comoda, altre volte risulta inadeguata e fastidiosa. L'attributo *autocomplete* è di tipo enumerato e accetta i seguenti valori:

- *on*: indica che l'input può essere completato in maniera automatica dal browser;
- *off*: indica che l'input deve essere inserito manualmente ogni volta che ne è richiesta la compilazione;
- non specificato: indica di usare il valore di default del browser.

### ESEMPIO

```
<form method="post" action="...">
  User name:
  <input type="text" name="username" autocomplete="on"
    placeholder="Inserire username">
  <input type="reset" value="Reset">
  <input type="submit" value="Invio">
</form>
```

## Espressioni regolari

L'attributo *pattern* permette di specificare un modello per la validazione della stringa di testo inserita dall'utente in un campo di input.

Utilizzando l'attributo *pattern*, per esempio, è possibile specificare che un codice fiscale deve essere obbligatoriamente composto da 6 caratteri di testo, seguiti da 2 cifre numeriche, da 1 carattere di testo, da 2 cifre numeriche, da 1 carattere di testo, da 3 cifre numeriche e infine da 1 carattere di testo.

Il valore dell'attributo *pattern* è una «espressione regolare» e deve essere espressa seguendo la sintassi standard prevista per le *regex* (*regular expression*), che non è specifica del linguaggio HTML.

## ■ *multiple*

L'attributo *multiple* è di tipo booleano e permette all'utente l'inserimento di più valori per uno stesso input (ad esempio un insieme di indirizzi e-mail a cui inviare contemporaneamente un messaggio).

ESEMPIO

```
<form>
  E-mail a cui inviare il messaggio:
  <input type="email" multiple name="email[]" />
  <input type="reset" value="Reset">
  <input type="submit" value="Invio">
</form>
```

**OSSERVAZIONE** Nell'esempio precedente l'attributo name ha un valore di tipo array in modo che la corrispondente variabile PHP possa contenere i valori multipli specificati.

ESEMPIO

Il codice HTML che segue definisce una pagina web che contiene una *form* per l'inserimento dei dati relativi a un acquisto effettuato su un sito di *e-commerce*:



```
<html>
  <head>
    <title>Acquisto</title>
  </head>
  <body>
    <form id="acquisto" method="POST" action="acquisto.php">
      <fieldset>
        <legend>Dati acquirente</legend>
        <ul>
          <li>
            <label for="nominativo">Nominativo</label>
            <input id="nominativo" name="nominativo" type="text"
              title="Cognome e nome" required autofocus>
          </li>
          <li>
            <label for="email">E-mail</label>
            <input id="email" name="email" type="email"
              placeholder="nome@dominio.it" required>
          </li>
          <li>
            <label for="telefono">Telefono</label>
            <input id="telefono" name="telefono" type="tel" required>
          </li>
        </ul>
      </fieldset>
      <fieldset>
        <legend>Indirizzo consegna</legend>
        <ul>
```



```

<li>
  <label for="indirizzo">Indirizzo</label>
  <textarea id="indirizzo" name="indirizzo" rows="2" required>
</textarea>
</li>
<li>
  <label for="CAP">CAP</label>
  <input id="CAP" name="CAP" type="text" size="5" maxlength="5" required>
</li>
<li>
  <label for="provincia">Provincia</label>
  <input id="provincia" name="provincia" type="text" size="2"
    maxlength="2" required>
</li>
</ul>
</fieldset>
<fieldset>
  <legend>Carta di credito</legend>
  <ul>
    <li>
      <fieldset>
        <legend>Tipo Carta</legend>
        <ul>
          <li>
            <input id="visa" name="tipocarta" type="radio" checked>
            <label for="visa">VISA</label>
          </li>
          <li>
            <input id="cartasi" name="tipocarta" type="radio">
            <label for="cartasi">CartaSi</label>
          </li>
          <li>
            <input id="mastercard" name="tipocarta" type="radio">
            <label for="mastercard">Mastercard</label>
          </li>
        </ul>
      </fieldset>
    </li>
    <li>
      <label for="numero_carta">Numero Carta</label>
      <input id="numero_carta" name="numero_carta" type="text"
        size="20" required>
    </li>
    <li>
      <label for="sicurezza">Codice sicurezza</label>
      <input id="sicurezza" name="sicurezza" type="text" size="4" required>
    </li>
    <li>
      <label for="intestatario">Nominativo</label>
      <input id="intestatario" name="intestatario" type="text"
        placeholder="Intestatario carta" required>

```





```

        </li>
    </ul>
    </fieldset><br>
</form>
<input type="submit" value="Acquista" form="acquisto">
</body>
</html>

```

La pagina web contenente la *form* visualizzata dal browser è la seguente:

## 7

# Gestione dei *cookies* e delle sessioni in linguaggio PHP

Il protocollo HTTP è privo di stato (*stateless*): una volta che una pagina web è stata inviata dal server al client, per il codice PHP non è più possibile accedere ai dati relativi alla pagina stessa. Lo sviluppatore di applicazioni basate su script server side necessita di memorizzare informazioni persistenti per più pagine: un esempio classico in questo senso consiste nel mantenere traccia di un utente che ha effettuato il *login*. Il linguaggio PHP rende disponibili due metodi per la memorizzazione di dati attraverso pagine web diverse: i *cookie* e le sessioni.

## 7.1

### Cookie

I *cookie* (letteralmente «biscottini») sono dei file che il server web memorizza nel file-system della macchina client. I *cookie* hanno nomi e valori, una validità temporale ed eventualmente impostazioni di sicurezza.

```
$dato="Questa stringa viene memorizzata nel cookie"
setcookie("Alfa",$dato);
setcookie("Beta",$dato,time()+60*60*24*30);
```

La prima invocazione della funzione `setcookie` crea un *cookie* denominato "Alfa" sulla macchina client e vi memorizza la stringa contenuta nella variabile `$dato` senza impostare alcuna scadenza temporale<sup>5</sup>. La seconda invocazione opera come la precedente impostando però la validità temporale del *cookie* "Beta" utilizzando la funzione `time` seguita da un'espressione numerica che a partire dall'orario di sistema ne fissa la validità per trenta giorni successivi il momento della memorizzazione (60 secondi × 60 minuti × 24 ore × 30 giorni).

L'array superglobale `$_COOKIE` permette l'accesso ai valori dei *cookies* così memorizzati

ES.

L'istruzione `$delta=$_COOKIE("Beta")` assegna alla variabile `$delta` il contenuto del *cookie* "Beta".

Un *cookie* può essere eliminato impostando una scadenza precedente all'orario corrente.

**OSSERVAZIONE** Tramite l'uso dei *cookie*, è quindi possibile realizzare un meccanismo mediante il quale le applicazioni lato server possono memorizzare e recuperare informazioni lato client (rappresentato dal browser) in modo da poter associare a ogni utente uno «stato». In effetti utilizzando i *cookie* è possibile rendere persistenti delle variabili nell'arco di più accessi successivi, variabili che possono essere sfruttate per il mantenimento di informazioni relative all'utente. Il meccanismo dei *cookie* tuttavia presenta delle limitazioni intrinseche che lo rendono inadatto ad applicazioni complesse.

5. In questo caso la validità del *cookie* termina con la disconnessione del browser dal sito.

### Cookie

I *cookie* del protocollo HTTP sono file inviati dal server al client (normalmente il browser) contenenti un'associazione nome/valore e una validità temporale che il client invia automaticamente al server ogni volta che avviene una connessione allo stesso dominio.

Dato che sono memorizzati nel sistema client possono essere eliminati o modificati – anche a scopo malevolo – fuori dal controllo del server: sono per questo motivo considerati poco sicuri.

## 7.2 Sessioni

Per gestire dati persistenti in genere si preferisce fare ricorso al meccanismo delle sessioni.

Una sessione consiste in una serie di accessi a pagine PHP, effettuati in un determinato arco di tempo durante il quale viene mantenuto uno «stato» costituito da variabili persistenti.

**OSSERVAZIONE** Ogni sessione è individuata da un identificatore univoco utilizzato per l'associazione tra il client e la relativa sessione: i dati relativi a una sessione risiedono sul server e non sul client offrendo maggiori possibilità di controllo e un livello di sicurezza superiore rispetto ai *cookie*.

Per utilizzare le sessioni in PHP sono previste tre funzioni base: `session_start`, `session_unset` e `session_destroy`.

La funzione `session_start`, viene invocata per creare una nuova sessione o per ripristinarla nel caso sia stata creata in precedenza. Questa funzione tenta anche di impostare o recuperare, nel browser client, un *cookie* contenente l'identificativo di sessione, per cui è necessario che venga invocata all'inizio degli script PHP.

**OSSERVAZIONE** Nel caso che il browser non consenta la memorizzazione di *cookie*, le sessioni, che normalmente richiedono un *cookie* per la memorizzazione dell'identificativo di sessione che deve essere inviato al server ogni volta che il browser ricarica una nuova pagina del sito, devono utilizzare un meccanismo diverso per recuperarlo. L'interprete PHP può essere configurato per aggiungere l'identificativo di sessione come stringa di interrogazione a ogni URL richiesto dal browser: questa modalità è considerata insicura, per cui gli script PHP devono effettuare controlli aggiuntivi, per esempio sulla coerenza dell'indirizzo IP del sistema client.

Per registrare dei valori e renderli permanenti durante una sessione è sufficiente memorizzarli nell'array superglobale `$_SESSION`.

**ESEMPIO**

Se abbiamo definito una variabile `$nomeutente` e vogliamo renderne persistente il valore per tutta la durata della sessione è sufficiente eseguire la seguente riga di codice:

```
$_SESSION['nomeutente']=$nomeutente;
```

La funzione `session_unset` viene utilizzata per distruggere tutti i valori di sessione registrati fino al momento della sua invocazione. Per annullare una sola variabile nell'ambito di una sessione è invece possibile usare la funzione standard `unset` nella forma `unset($_SESSION['variabile']);`.

**OSSERVAZIONE** Non si deve invocare la funzione `unset` direttamente su `$_SESSION` perché in questo modo si disabilita la possibilità di registrare variabili di sessione nell'array superglobale `$_SESSION`.

Infine, la funzione `session_destroy` viene utilizzata per resettare una sessione e distruggere i dati relativi a essa. Tipicamente `session_destroy` viene invocata al momento del *logout* da parte di un utente.

**OSSERVAZIONE** Una sessione viene identificata tramite una stringa che prende il nome di SID (*Session Identifier*). Questa viene scambiata tra client e server e identifica il file sul server che contiene i dati relativi alla sessione stessa.

La sicurezza dei file di sessione è un aspetto sistemistico: è necessario configurare il server in modo che nessuno possa accedere da remoto a tali file e che nessuna applicazione «estranea» possa aprirli in lettura e/o scrittura.

Il seguente codice PHP avvia una sessione e memorizza un valore al suo interno:

```
<?php
session_start();
$ora=time();
$_SESSION['ora']=$ora;
...
?>
```

Una volta stabilita la sessione PHP è in grado di gestirne automaticamente la persistenza tra richieste di pagine diverse all'interno del ciclo di attività del browser.

La seguente pagina inizializza una sessione PHP registrandone il tempo di inizio:

```
<?php
    session_start();
    $_SESSION["ora"] = time();
?>
<html>
    <head>
        <title>Inizio sessione</title>
    </head>
    <body>
        <a href="time.php">Tempo sessione</a>
    </body>
</html>
```

La selezione del *link* da parte dell'utente richiama la pagina "time.php" che visualizza il tempo trascorso dall'inizio della sessione:

```
<?php
    session_start();
?>
<html>
    <head>
        <title>Tempo sessione</title>
    </head>
    <body>
        <?php
            $ora = time();
            $tempo = $ora - $_SESSION["ora"];
            echo "Sono trascorsi $tempo secondi dall'inizio della
                sessione.";
            session_unset();
            session_destroy();
        ?>
    </body>
</html>
```

■ **PHP.** *Hypertext Preprocessor* è un linguaggio di *scripting server side* utilizzato principalmente per sviluppare applicazioni web. L'esecuzione del codice PHP sul server produce codice HTML da inviare al browser dell'utente che ne fa richiesta. Il linguaggio PHP utilizza una sintassi *C-like*.

■ **Client.** Software che istanzia l'interfaccia utente di un'applicazione (tipicamente un browser) connettendosi tramite un'infrastruttura di rete a un server.

■ **Server.** Software (e per estensione il computer su cui è eseguito) che, oltre alla gestione logica del sistema (fornitura di servizi), deve implementare le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza dei dati.

■ **Interazione client/server.** Un client richiede un servizio a un server mediante uno specifico protocollo; il server, se trova soddisfatte tutte le condizioni per fornire il servizio richiesto lo eroga, altrimenti risponde al client con un messaggio di errore.

■ **Server web.** Un server web è un software finalizzato alla gestione di un servizio web. Esso viene generalmente installato su una macchina server e si occupa di fornire su richiesta del client pagine web che sono visualizzabili tramite un browser. Le informazioni inviate dal server web al client e viceversa viaggiano in rete e sono gestite dal protocollo HTTP.

■ **Localhost.** Uno stesso computer può configurarsi allo stesso tempo sia come server che come client. Infatti se da un browser installato su un computer su cui è in funzione un server web viene invocato *localhost* l'effetto è quello di inoltrare tale richiesta al server web locale che risponderà alle richieste del browser esattamente come se fosse un server remoto.

■ **Pagine web statiche.** Pagine contenute in file memorizzati su server codificate in linguaggio HTML (con eventuali script in linguaggio JavaScript) che sono immutabili nel tempo fino a quando non si riscrive parte del loro codice.

■ **Pagine web dinamiche.** Pagine che sono il prodotto di script eseguiti sul server, il cui output è formattato come codice HTML inviato al browser del client; dal momento che la sequenza delle istruzioni eseguite da un programma dipende dai dati che esso riceve in input e che l'output cambia di conseguenza, ne deriva che le pagine visualizzate lato client possono cambiare in maniera dinamica.

■ **Script PHP.** Uno script PHP è costituito da uno o più blocchi di istruzioni delimitate dai tag `<?php` e `?>` che ne indicano inizio e fine; ogni istruzione termina col simbolo «;». In una pagina PHP possono trovarsi sia istruzioni PHP che codice HTML: questo non viene elaborato dall'interprete PHP, ma inviato al client così come è. L'output di uno script PHP è generalmente una pagina web dinamica visualizzata dal browser del client che ne ha richiesto l'esecuzione.

■ **Inclusione di file.** Una caratteristica del linguaggio PHP è quella relativa alla possibilità di includere file contenenti codice PHP. Nell'economia di un'applicazione PHP è possibile memorizzare parti di codice comune in un file che poi può essere incluso in un punto qualsiasi di uno script utilizzando la parola chiave `include`, oppure `require`.

■ **Output.** L'output di uno script PHP viene prodotto tramite le istruzioni `echo` e `print`, la cui differenza fondamentale risiede nel fatto che `echo` è in grado di effettuare l'output contemporaneo di più dati, mentre `print` di uno solo alla volta.

■ **Variabili PHP.** Nel linguaggio PHP le variabili hanno il nome che inizia sempre col simbolo «\$» e non necessitano di una dichiarazione esplicita prima del loro utilizzo. In un qualsiasi punto all'interno di uno script PHP è possibile dichiarare e iniziare a utilizzare una variabile semplicemente nominandola. Una variabile assume un certo tipo in funzione del valore che le viene assegnato ed è possibile assegnarle in punti diversi valori di tipo diverso. Il linguaggio PHP prevede l'uso di variabili dinamiche che iniziano con un doppio simbolo «\$» per cui, ad esempio la variabile `$$alfa` viene interpretata sostituendo per primo il valore della variabile più interna `$alfa` che di-

venta il nome della variabile associata al simbolo «\$» più esterno.

■ **Costanti PHP.** Per dichiarare una costante in PHP si utilizza la parola chiave **define**.

■ **Array PHP.** In PHP gli array sono vettori indicizzabili sia attraverso un valore numerico il cui valore iniziale è zero, sia per mezzo di una stringa (chiave). In questo caso il vettore memorizza delle associazioni chiave/valore e viene detto array associativo. A differenza di altri linguaggi di programmazione PHP accetta che gli elementi di un vettore non siano necessariamente tutti dello stesso tipo. Un array a più dimensioni è un array nel quale uno o più elementi sono a loro volta degli array. In PHP gli array possono essere visitati in sequenza con il costrutto **foreach**.

■ **Array PHP correlati al web.** Per quanto riguarda le attività web, PHP prevede degli specifici array associativi denominati superglobali e dedicati a specifiche funzionalità. Tali array hanno nomi specifici: **\$\_GET**, **\$\_POST**, **\$\_REQUEST**, **\$\_FILES**, **\$\_COOKIE** e **\$\_SESSION**. I primi tre sono relativi al passaggio di dati tra pagine web, mentre gli altri due alla gestione di *cookie* e sessioni.

■ **\$\_GET.** È un array associativo superglobale creato automaticamente con l'esistenza di una stringa di interrogazione all'interno dell'URL della pagina. Qualsiasi informazione inviata con la stringa d'interrogazione è una coppia nome/valore che viene caricata nell'array associativo **\$\_GET** della pagina invocata.

■ **\$\_POST.** È un array superglobale simile al precedente con la differenza che invece di ricevere dati dalla stringa di interrogazione dell'URL, usa il metodo HTTP POST. Dal momento che i dati trasferiti non sono visibili nell'URL sono trasparenti all'utente, aumentando quindi il livello di sicurezza.

■ **\$\_REQUEST.** È un array che contiene gli elementi degli array **\$\_GET** e **\$\_POST**.

■ **Funzioni PHP.** PHP prevede una notevole varietà di librerie di funzioni che coprono moltissime esigenze per chi sviluppa software (accesso a database, gestione di documenti PDF, produzioni di grafici, ecc.). Oltre alle funzioni generiche sulle variabili, PHP prevede funzioni specifiche per operare su array, su date e orari e su dati di tipo strin-

ga. PHP supporta, come molti altri linguaggi, la possibilità di definire funzioni da parte dell'utente che possono essere dichiarate con la parola chiave **function**.

■ **Passaggio di parametri.** Il passaggio di parametri alle funzioni avviene come modalità base per valore. Il passaggio di valori per riferimento avviene premettendo al nome della variabile il simbolo «&».

■ **Visibilità e scope delle variabili.** Le variabili definite e utilizzate all'interno di una funzione sono visibili solo all'interno della funzione e cessano di esistere fuori dal contesto di esecuzione della stessa. Eventuali variabili definite all'interno di una funzione con lo stesso nome di una variabile già definita esternamente, «nascondono» la variabile esterna fino a quando la funzione non termina. Premettendo, all'interno di una funzione, la parola chiave **global** al nome di una variabile, si riferenzia una variabile omonima definita esternamente.

■ **Form.** Il linguaggio HTML mediante il tag **form** consente la creazione di *form* per l'inserimento e/o la selezione di dati di input da parte dell'utente. Attributi fondamentali di una *form* sono la pagina destinataria dei dati e il metodo di passaggio: GET, che aggiunge una stringa di interrogazione all'URL di richiesta della pagina, o POST, che utilizza il metodo POST del protocollo http. All'interno di una *form* è possibile prevedere più tag **input**, ciascuno dei quali visualizza un campo per l'inserimento/selezione di un valore; è inoltre necessario un tag **input** di tipo *submit*, che visualizza un pulsante di conferma dell'inoltro dei dati della *form* alla pagina di destinazione (normalmente una pagina contenente uno script PHP che elabora i dati ricevuti recuperandoli dagli array superglobali **\$\_GET**, **\$\_POST** o **\$\_REQUEST**).

■ **Interazione tra pagine web.** Oltre all'invio dei dati di una *form* è possibile invocare una pagina presente sul server e contenente uno script in linguaggio PHP mediante un collegamento ipertestuale, oppure effettuando un *redirect* mediante l'istruzione PHP **header**.

■ **Validazione dati form di input.** La validazione dei dati inseriti nelle *form* dall'utente avviene generalmente lato server. Per diminuire il traffico di dati dal client verso il server e viceversa è pos-

sibile utilizzare JavaScript per effettuare controlli lato client. Le ultime versioni di HTML hanno introdotto nuovi tipi di campi di input per permettere la validazione lato client senza la necessità di impiegare JavaScript.

**Cookie.** I *cookie* (letteralmente «biscottini») sono dei file che il server web memorizza nel file-system della macchina client. I *cookie* hanno nomi e valori, una validità temporale ed eventualmente impostazioni di sicurezza. Tramite l'uso dei *cookie*, è possibile realizzare un meccanismo mediante il quale le applicazioni lato server possono memorizzare e recuperare informazioni lato client (rappresentato dal browser) in modo da poter associare a ogni utente uno «stato». In effetti utilizzando i *cookie* è possibile rendere persistenti delle variabili nell'arco di più accessi successivi, variabili che possono essere sfruttate per il mantenimento

di informazioni relative all'utente. Il meccanismo dei *cookie* tuttavia presenta delle limitazioni intrinseche che lo rendono inadatto ad applicazioni complesse.

**Sessioni.** Per gestire dati persistenti in genere si fa ricorso al meccanismo delle sessioni. Una sessione consiste in una serie di accessi a pagine PHP, effettuati in un determinato arco di tempo durante il quale viene mantenuto uno stato costituito da variabili persistenti. Ogni sessione è individuata da un identificatore univoco utilizzato per l'associazione tra il client e la relativa sessione: i dati relativi a una sessione risiedono sul server e non sul client offrendo maggiori possibilità di controllo e un livello di sicurezza superiore rispetto ai *cookie*. Da uno script PHP è possibile accedere ai dati di una sessione attraverso l'array superglobale `$_SESSION`.

## QUESITI

**1** Il significato dell'acronimo PHP è ...

- A ... Personal Hypertext Processor.
- B ... Hypertext Preprocessor.
- C ... Personal Home Page.
- D ... Private Home Page.

**2** Quali delle seguenti affermazioni circa il PHP sono vere?

- A È un linguaggio fortemente tipizzato.
- B È un linguaggio di *scripting server side*.
- C È un linguaggio di *scripting client side*.
- D Può essere utilizzato insieme ad HTML e JavaScript.

**3** Gli script server-side PHP sono delimitati da ...

- A ... `<script>...</script>`
- B ... `<?php>...</?>`
- C ... `<?php...?>`
- D ... `<&>...</&>`

**4** Come è possibile visualizzare il messaggio «Hello World» in PHP?

- A `echo "Hello World";`
- B `"Hello World";`
- C `Document.Write("Hello World");`
- D Nessuna delle risposte precedenti.

**5** Con quale simbolo iniziano le variabili PHP?

- A %
- B &
- C !
- D Nessuno dei precedenti.

**6** Quale dei seguenti non è un operatore di confronto PHP?

- A `!=`
- B `>=`
- C `<=>`
- D `===`

**7** Quale dei seguenti nomi di variabile non è corretto?

- A `$mia-Var`
- B `$miaVar`



C \$mia\_Var

D Sono tutte corrette.

**8** Indicare che cosa viene visualizzato con le seguenti istruzioni PHP:

```
$var = 'a';  
$VAR = 'b';  
echo "$var$VAR";
```

A aa

B ab

C bb

D ba

**9** Indicare che cosa viene visualizzato con le seguenti istruzioni

```
$padre = 'madre';  
$madre = 'figlio';  
echo "$$padre";
```

A figlio

B madre

C madrefiglio

D error

**10** Qual è il metodo corretto per includere il file «time.php»?

A <?php include\_file("time.php"); ?>

B <% include file="time.php" %>

C <?php require("time.php"); ?>

D <!--include file="time.php"-->

**11** Qual è il modo corretto per dichiarare una funzione in PHP?

A function miaFunzione()

B create miaFunzione()

C newfunction miaFunzione()

D Nessuno dei precedenti.

**12** Come è possibile accedere ai dati di una *form* inviati col metodo GET?

A Request.Form

B \$\_GET["valore\_chiave"]

C Request.QueryString

D \$GET["valore\_chiave"]

**13** In PHP le funzioni `die()` e `exit()` operano esattamente nello stesso modo.

A Vero.

B Falso.

**14** In PHP gli array possono essere ordinati con la funzione ...

A ... `vetsort()`

B ... `arrsort()`

C ... `sort()`

D Tutte le precedenti.

**15** Quali delle seguenti funzioni sono utilizzabili in PHP per determinare il tipo di una variabile?

A `gettype()`

B `is_double()`

C `get_type()`

D `is_date()`

**16** Indicare quali delle seguenti affermazioni circa gli array PHP sono vere.

A Hanno un numero predefinito di elementi stabilito all'atto della dichiarazione.

B Possono contenere dati di tipo non omogeneo.

C I loro elementi possono essere acceduti per numero di posizione occupata.

D I loro elementi possono essere visitati *solo* tramite il costrutto `foreach`.

E I loro elementi possono essere acceduti tramite il valore di una chiave associativa.

**17** Come è possibile fare riferimento al valore «d» nel seguente vettore PHP?

```
$a = array(  
    'a',  
    3 => 'b',  
    1 => 'c',  
    'd'  
);
```

A `a[0]`

B `a[1]`

C `a[2]`

D `a[3]`

E `a[4]`



**18** Che cosa verrà visualizzato dalle seguenti istruzioni PHP?

```
if ('2' == '02') {  
    echo 'vero';  
} else {  
    echo 'falso';  
}
```

- A vero
- B falso

**19** Che cosa verrà visualizzato dalle seguenti istruzioni PHP?

```
$a = array();  
if ($a == null) {  
    echo 'vero';  
} else {  
    echo 'falso';  
}
```

- A vero
- B falso

**20** Che cosa verrà visualizzato dalle seguenti istruzioni PHP?

```
$str="3euro";  
$a=20;  
$a+=$str;  
print($a);
```

- A 23euro
- B 203euro
- C 320euro
- D 23

**21** Il nome della pagina richiesta nel tag `<form method=...>` viene definito con l'attributo ...

- A ... method
- B ... type
- C ... action
- D Nessuno degli attributi precedenti.

**22** Per scorrere gli elementi di un array conviene utilizzare il costrutto ...

- A ... `switch`
- B ... `foreach`
- C ... `while`
- D Nessuno dei precedenti.

**23** Il passaggio «trasparente» di dati tra pagine web è possibile col metodo ...

- A ... GET
- B ... POST
- C ... REQUEST
- D Nessuno dei precedenti.

**24** L'attributo `required` in un tag che prevede un input di dati permette di ...

- A ... controllare lato client che il dato in questione sia comunque avvalorato.
- B ... controllare lato server che il dato in questione sia comunque avvalorato.
- C ... controllare lato client che il dato in questione abbia un valore di default.
- D Nessuno dei precedenti.

**25** L'attributo `autofocus` in un tag che prevede un input di dati permette di ...

- A ... posizionare il prompt su quel campo al caricamento della pagina.
- B ... posizionare il prompt su quel campo al *submit* dei dati.
- C ... posizionare il prompt su quel campo quando lo si seleziona col mouse.
- D Nessuno dei precedenti.

**26** Per controllare che un campo di input contenga valori appartenenti a un certo intervallo, HTML prevede il tipo di input ...

- A ... number
- B ... range
- C ... search
- D Nessuno dei precedenti.

**27** Un array associativo è un array che permette di accedere ai propri elementi in funzione ...

- A ... del loro indice di posizione.
- B ... del valore di una chiave associata.
- C ... del valore contenuto in essi.
- D Nessuno dei precedenti.

**28** Le funzioni PHP accettano l'omissione di qualche parametro all'atto della loro invocazione ...

- A ... no, mai.
- B ... sì, sempre.
- C ... sì, a patto che nella loro firma sia stato previsto un valore di default per tali parametri.

**29** L'istruzione `$array[]=$alfa` ...

- A ... non è ammessa nel PHP.
- B ... accoda il valore della variabile `$alfa` all'array.
- C ... inserisce il valore della variabile `$alfa` come primo elemento dell'array.
- D ... inserisce il valore della variabile `$alfa` nella prima posizione libera dell'array.

**30** Un *cookie* è un meccanismo ...

- A ... per validare i dati di una *form*.
- B ... mediante il quale le applicazioni lato server posso memorizzare e recuperare informazioni lato client.
- C ... mediante il quale le applicazioni lato client posso memorizzare e recuperare informazioni lato server.
- D Nessuno dei precedenti.

**31** Indicare quali delle seguenti affermazioni sono vere circa le sessioni.

- A Non sono previste dal linguaggio PHP.
- B Sono meccanismi per gestire dati persistenti durante la navigazione tra pagine web dinamiche diverse.
- C I dati gestiti da una sessione vengono memorizzati in file lato client.
- D I dati gestiti da una sessione vengono memorizzati in file lato server.
- E La funzione `session_unset()` viene utilizzata per terminare una sessione.
- F I dati di una sessione vengono memorizzati nel vettore associativo `$_SESSION`.

## ESERCIZI

**1** Codificare più script PHP che producano in output:

- a) la tavola pitagorica rappresentata in una tabella;
- b) la tabellina di un numero ricevuto in input tramite una *form*. Lo script deve verificare che l'utente abbia inserito un numero, in caso contrario deve segnalare l'errore;
- c) il fattoriale di un numero ricevuto in input tramite una *form*. Lo script deve verificare che il numero sia compreso tra 1 e 10, in caso contrario deve segnalare l'errore;
- d) tutti i numeri primi minori di un valore fornito tramite una *form* (un numero intero si dice primo se è maggiore di 1 e divisibile solamente per se stesso e per 1).

**2** Codificare uno script PHP che riceva una stringa in input tramite una *form* produca in output:

- a) il numero di consonanti e vocali che contiene;
- b) il numero di caratteri di tipo numerico che contiene;
- c) la frequenza con cui ogni carattere appare nella stringa.

**3** Codificare una *form* HTML che invoca lo script "stamp.php". La *form* deve contenere quattro campi di testo: uno per il nome, uno per un indirizzo di posta elettronica e due per la password (il secondo serve per verificare che l'utente inserisca la stessa password due volte). Lo script "stamp.php" restituisce i dati ricevuti in una tabella solo se le due password inviate coincidono; in caso negativo, lo script deve inviare un messaggio di errore che illustri all'utente quanto accaduto.

**4** Codificare uno script PHP che, data una lista di nomi contenuti in una sola stringa, generi un array con i nomi inseriti e visualizzi sia la stringa iniziale che l'array ottenuto (la stringa viene fornita da una *form*).

## LABORATORIO

**1** Realizzare una *form* HTML per il menu di un ristorante on-line. La *form* deve contenere una serie di pietanze che possono essere scelte dall'utente e il relativo prezzo. La *form* deve richiamare lo script "conto.php" che si occupa di calcolare il prezzo (incluso il costo del servizio a domicilio) del pasto ordinato. Inoltre lo script PHP restituisce in una tabella il menu ordinato dall'utente con il relativo costo.

**2** Data la *form* riportata nel box in basso e il relativo codice HTML

- scrivere uno script PHP che visualizzi in una tabella le informazioni inserite dall'utente modificando il codice della *form*;
- modificare lo script PHP del punto precedente in modo visualizzare i valori passati dall'utente solo se la password associata al nome è corretta (la password e il nome utente sono memorizzate in due costanti dello script).

**3** Data la seguente *form* e il relativo codice HTML



```
<html>
<head>
<title>Animale preferito</title>
</head>
<body>

<form method="post" action="">

Seleziona l'animale preferito<br>
<select name="animale[]" size="3" multiple>
<option value="1"> Cane
<option value="2" selected> Gatto
<option value="3"> Coniglio
<option value="4"> Criceto
<option value="5"> Canarino
<option value="6"> Pesce rosso
</select>
```

```
<html>
<head>
<title>Richiesta argomenti</title>
</head>

<body>
<h1>Seleziona scelta</h1>

<hr>
<form method="post" action="">

Nome: <input type="text" name="nome"><br>
Password: <input type="password" name="password"><p>

Quali argomenti desideri approfondire?<br>
<input type="checkbox" name="argomento[]" value="HTML"> HTML <br>
<input type="checkbox" name="argomento[]" value="PHP"> PHP <br>
<input type="checkbox" name="argomento[]" value="ASP"> ASP <br>
<input type="checkbox" name="argomento[]" value="multimedia">
  Oggetti multimediali <br>
<hr>

<input type="submit" value="Invio">
<input type="reset" value="Cancella tutto!">

</form>
</body>
</html>
```

### Art. 1. Sistemi di codificazione

Le persone fisiche, le persone giuridiche e le società, associazioni e altre organizzazioni di persone o di beni prive di personalità giuridica sono iscritte all'anagrafe tributaria secondo appositi sistemi di codificazione.

### Art. 2. Numero di codice fiscale delle persone fisiche

Il numero di codice fiscale delle persone fisiche è costituito da un'espressione alfanumerica di sedici caratteri. I primi quindici caratteri sono indicativi dei dati anagrafici di ciascun soggetto secondo l'ordine seguente:

- tre caratteri alfabetici per il cognome;
- tre caratteri alfabetici per il nome;
- due caratteri numerici per l'anno di nascita;
- un carattere alfabetico per il mese di nascita;
- due caratteri numerici per il giorno di nascita e il sesso;
- quattro caratteri (uno alfabetico e tre numerici) per il comune italiano o per lo stato estero di nascita.

Il sedicesimo carattere, alfabetico, ha funzioni di controllo.

### Art. 7. Carattere alfabetico di controllo

Il sedicesimo carattere ha funzione di controllo della esatta trascrizione dei primi quindici caratteri. Esso viene determinato nel modo seguente: ciascuno degli anzidetti quindici caratteri, a seconda che occupi posizione di ordine pari o posizione di ordine dispari, viene convertito in un valore numerico in base alle corrispondenze indicate rispettivamente ai successivi punti 1) e 2).

1) Per la conversione dei sette caratteri con posizione di ordine pari:

A o 0 = 0	F o 5 = 5	K = 10	P = 15	U = 20
B o 1 = 1	G o 6 = 6	L = 11	Q = 16	V = 21
C o 2 = 2	H o 7 = 7	M = 12	R = 17	W = 22
D o 3 = 3	I o 8 = 8	N = 13	S = 18	X = 23
E o 4 = 4	J o 9 = 9	O = 14	T = 19	Y = 24
				Z = 25

2) Per la conversione degli otto caratteri con posizione di ordine dispari:

A o 0 = 1	F o 5 = 13	K = 2	P = 3	U = 16
B o 1 = 0	G o 6 = 15	L = 4	Q = 6	V = 10
C o 2 = 5	H o 7 = 17	M = 18	R = 8	W = 22
D o 3 = 7	I o 8 = 19	N = 20	S = 12	X = 25
E o 4 = 9	J o 9 = 21	O = 11	T = 14	Y = 24
				Z = 23

I valori numerici così determinati vengono addizionati e la somma si divide per il numero 26. Il carattere di controllo si ottiene convertendo il resto di tale divisione nel carattere a esso corrispondente nella sotto indicata tabella:

0 = A	5 = F	10 = K	15 = P	20 = U
1 = B	6 = G	11 = L	16 = Q	21 = V
2 = C	7 = H	12 = M	17 = R	22 = W
3 = D	8 = I	13 = N	18 = S	23 = X
4 = E	9 = J	14 = O	19 = T	24 = Y
				25 = Z

```

<input type="reset" value="Annulla">
<input type="submit" value="Ok">
</form>
</body>
</html>

```

realizzare uno script PHP che visualizzi l'animale/i selezionato/i.

- 4** Realizzare una pagina che visualizzi la seguente form:

e uno script PHP che acquisiti i valori inviati dalla form li visualizzi in modo che i generi scelti vengano memorizzati tutti in una unica stringa separati da «,».

- 5** Realizzare la seguente form



in modo che quando si preme «Invia» venga creata una nuova form con numero di campi di input

uguale al numero inserito. Per esempio, inserendo 2 dovrà essere visualizzata la form:

Lo script deve inserire i nomi forniti in input in un array e visualizzarli in ordine alfabetico.

- 6** Nell'ipotesi di mantenere direttamente nel codice PHP utilizzando un array associativo la rubrica dei numeri telefonici interni del personale di un'organizzazione, realizzare una pagina web HTML che consenta, mediante l'uso di una form, l'inserimento di un nominativo e una pagina web dinamica in linguaggio PHP che fornisca il relativo numero di telefono, o un messaggio di errore. Estendere le funzionalità dell'applicazione web permettendo all'utente di inserire un numero telefonico per ricercare il nominativo della persona a cui si riferisce.
- 7** Un decreto ministeriale del 23 dicembre 1976 intitolato «Sistemi di codificazione dei soggetti da iscrivere all'anagrafe tributaria» stabilisce le caratteristiche formali del codice fiscale e le modalità per verificarne la validità (box nella pagina a fianco). Realizzare una pagina web HTML che consenta, mediante l'uso di una form, l'inserimento di un codice fiscale e una pagina web dinamica in linguaggio PHP che ne verifichi la validità fornendo come risultato l'esito della verifica.

## CHAPTER 1

### The Good Parts

This book has been a rather long time in the making. I have been using PHP for many years now, and have grown to love it more and more for its simplistic approach to programming, its flexibility, and its power. Of all the programming languages I have used throughout my over 20-year career, PHP is my favorite, hands down. PHP has grown from a small set of functions to a very large volume of functions, external interfaces, and add-on tools. Some programmers may be overwhelmed by its apparent vastness, but I hope to show you in this book that most of the PHP world can indeed be of great use. [...]

### Why PHP?

[...] PHP is a widely used language and has experienced much growth in recent years in the enterprise market. Web environments like Facebook, Flickr, portions of Yahoo!, and Wikipedia all use PHP in a significant way, and web content management systems like Drupal, Joomla, and WordPress are also powered by PHP. IBM is also showing a lot of interest in integrating its technologies with PHP. For these reasons, it makes sense for the community to assist beginning and intermediate programmers in becoming familiar with all the best areas of this language.

### A Brief History of PHP

Let's start with a brief history of the language. Personal Home Page (PHP), initially known as PHP Tools, was launched in June 1995 by Rasmus Lerdorf. It was originally launched as open source software and remains so to this day. Database integration was implemented in version 2.0 in 1996, and the product has grown by leaps and bounds ever since. Its worldwide use is higher than any other web development language. As of this writing, the latest version of PHP is 5.3, which was released on June 30, 2009.

### PHP's Place in the World

PHP is one of the most widely used programming languages in the world. To think that it has grown this much in such a short period of time is quite impressive; in just 15 years or so, it has grown to be one of the major players in the web development world.

In the last several years, many members of the PHP community have been debating whether the language is enterprise ready: can it be trusted to handle the big projects and weights? Given the recent focus on PHP from companies like IBM and Microsoft, and the fact that it powers the largest websites in the world (Facebook and Yahoo!), one could argue that it is already in the enterprise. This debate will be resolved over time, but with version 5.3 just recently having been released, it is a safe bet to say that if it isn't, it very soon will be.

### What Is PHP?

So what is PHP anyway? It is a scripting language, mostly used on the server side, that can be employed to generate Hypertext Markup Language (HTML) information dynamically. PHP is connected to a web server, generally Apache or Internet Information Server (IIS), and once it has finished generating proper HTML, it sends its creation back to the web server for delivery to the requesting client.

I say "mostly used" on the server side because you can use PHP in many other areas, including command line, desktop PC, and client server environments, just to name a few. However, it is most commonly used in the web server environment.

PHP developers can also integrate PHP with one of many different database tools like MySQL, SQLite, PostgreSQL, DB2, MS SQL, ORACLE, and so on, to make their created content as dynamic as possible. In reality, what is produced is still a static HTML file, but it is produced on the fly and therefore seems to be dynamic. Actually, one could argue that since the content is dynamically drawn out of a database or some other data source, PHP is in fact creating dynamic content.

**overwhelmed**

scoraggiato, intimidito, sopraffatto, sommerso

**by leaps and bounds**

a passi da gigante

**to argue**

discutere, argomentare

**delivery**

consegna, distribuzione

**proof**

testimonianza, prova, dimostrazione

**environment**

ambiente

**on the fly**

al volo, dinamicamente

**therefore**

quindi, pertanto

**untoward**

increscioso, sconveniente

**bearing on**

incidenza su, rapporto con

## What Has Been Accomplished with PHP?

Now, saying all these things about PHP and not having any proof would be untoward for sure, so let's take a quick highlight tour of what has been built and accomplished with PHP. Some of the major and most popular web locations in the world are powered at some level by PHP. Table 1-1 includes a short list of popular websites, their Uniform Resource Locators (URLs), and a brief description of what each does.

TABLE 1.1 Sampling of major websites that use PHP

Website name	Description	URL
Facebook	Social networking	<a href="http://www.facebook.com">http://www.facebook.com</a>
Flickr	Photograph sharing	<a href="http://www.flickr.com">http://www.flickr.com</a>
Wikipedia	Online collaborative encyclopedia	<a href="http://www.wikipedia.org">http://www.wikipedia.org</a>
SugarCRM	Customer relationship management tool	<a href="http://www.sugarcrm.com">http://www.sugarcrm.com</a>
Dotproject	Project management tool	<a href="http://www.dotproject.org">http://www.dotproject.org</a>
Drupal	Website construction template engine	<a href="http://drupal.org">http://drupal.org</a>
Interspire	Newsletter and email marketing product	<a href="http://www.interspire.com">http://www.interspire.com</a>

This is only the proverbial tip of the iceberg, and is not in any way meant to be an exhaustive list; it is simply a short list of examples of what has been built with PHP. If you have been to any of these websites, you can see what this powerful language can accomplish.

## Basic PHP Setup

By now you might be anxious to try PHP out for yourself, so we'll go through a quick installation discussion here and have you saying, "Hello, world" in no time.

The basic method of PHP development is to build PHP code on top of web server software like Apache or IIS. There is a "stack" of software that is generally used for a fully functional development environment: either LAMP or WAMP. LAMP stands for Linux/Apache/MySQL/PHP, but there are variations to this, as one would expect. You could be using PostgreSQL instead of MySQL for the database and therefore the acronym would be LAPP, but you get the idea. The other acronym – WAMP – stands for Windows/Apache/MySQL/PHP.

Typically, the OS has no real bearing on the functionality of the written code. PHP written in the Windows environment will certainly operate just as well on a Linux box and vice versa. The only thing to be cautious of is if you are doing OS-level commands like CHMOD (for changing file permissions) or CHOWN (for changing file ownerships) in Linux and want to do the same in a different OS. Just be sure to test your code well in this, and all, instances.

Since there are so many different platforms and components to setting up a full PHP development environment, we won't go into detail on how to establish that environment here. Be sure to go to <http://www.php.net/downloads.php> for a full listing of the latest stable releases for the many and varied platforms.

There are also some all-in-one installation packages for Windows; one is called XAMPP (X for cross-platform, A for Apache, M for MySQL, P for PHP, and P for Perl) [...]

Peter B. MacIntyre, *PHP: The Good Parts*, O'Reilly, April 2010

## QUESTIONS

- Why should beginning and intermediate users be assisted in becoming familiar with PHP?
- What bearing on the functionality of the PHP code does a specific OS have?
- Give the names and descriptions of at least three websites based upon PHP technology.
- What is the difference between "LAMP" and "WAMP" acronyms?

## 2 Kick-Starting OOP

In this chapter we will learn how to create objects, define their attributes (or properties) and methods. Objects in PHP are always created using a “class” keyword.

In this chapter we will learn the details of classes, properties, and methods. We will also learn the scope of methods and about modifiers and the benefits of using interfaces. This chapter will also introduce us to other basic OOP features in PHP. As a whole, this chapter is one of the better resources for you to kick-start OOP in PHP.

### Let's Bake Some Objects

As I said before, you can create an object in PHP using the class keyword. A class consists of some properties and methods, either public or private. Let's take the `Emailer` class [...]. We will discuss here what it actually does:

```
<?
//class.emailer.php
class Emailer
{
    private $sender;
    private $recipients;
    private $subject;
    private $body;
    function __construct($sender)
    {
        $this->sender = $sender;
        $this->recipients = array();
    }
    public function addRecipients($recipient)
    {
        array_push($this->recipients, $recipient);
    }
    public function setSubject($subject)
    {
        $this->subject = $subject;
    }
    public function setBody($body)
    {
        $this->body = $body;
    }
    public function sendEmail()
    {
        foreach ($this->recipients as $recipient)
        {
            $result = mail($recipient, $this->subject, $this->body,
                           "From: {$this->sender}\r\n");
            if result) echo "Mail successfully sent to ($recipient)<br/>";
        }
    }
}
?>
```

In this code, we started with `class Emailer`, which means that the name of our class is `Emailer`. While naming a class, follow the same naming convention as variables, i.e. you can't start with a numeric letter, etc.

#### kick-start

calcio d'inizio

#### to kick-start

avviare adesso,  
cominciare subito

#### wondering

chiedendosi, pensando di

#### instance

istanza

#### newly

recentemente, appena,  
da poco, di recente

#### benefit

beneficio, vantaggio



After that we declared the properties of this class. There are four properties here, namely, `$sender`, `$recipient`, `$subject`, and `$body`. Please note that we declare each of them with a keyword `private`. A private property means that this property can only be accessed internally from this class. Properties are nothing but variables inside a class.

If you remember what a method is, it is just a function inside the class. In this class there are five functions, `__construct()`, `addRecipient()`, `setSubject()`, `setBody()`, and `sendEmail()`. Please note that the last four methods are declared `public`. That means when someone instantiates this object, they can access these methods.

The `__construct()` is a special method inside a class which is called a constructor method. Whenever a new object is created from this class, this method will execute automatically. So if we have to perform some preliminary tasks in our object while initiating it, we will do so from this constructor method. For example, in the constructor method of this `Emler` class we just set the `$recipients` as a blank array and we also set the sender name.

## Accessing Properties and Methods from Inside the Class

Are you wondering how a function can access the class properties from inside its content? Let's see using the following code:

```
public function setBody($body)
{
    $this->body = $body;
}
```

There is a private property named `$body` inside our class, and if we want to access it from within the function, we must refer to it with `$this`. `$this` means a reference to a current instance of this object. So we can access the body property with `$this->body`. Please note that we have to access the properties (i.e. class variables) of a class using a `"->"` following the instance.

Similarly, like properties, we can access any member method from inside another member method in this format.

For example, we can invoke `setSubject` method as `$this->setSubject()`.

Please note that `$this` keyword is only valid inside the scope of a method, as long as it is not declared as static. You can not use `$this` keyword from outside the class. We will learn more about `"static"`, `"private"`, `"public"` keywords in the Modifiers section later this chapter.

## Using an Object

Let's use the newly created `Emler` object from inside our PHP code. We must note some things before using an object. You must initiate an object before using it. After initiating, you can access all its public properties and methods using `"->"` after the instance. Let's see using the following code:

```
<?
    $emailerobject = new Emler("hasin@pageflakes.com");
    $emailerobject->addRecipients("hasin@somewherein.net");
    $emailerobject->setSubject("Just a Test");
    $emailerobject->setBody("Hi Hasin, How are you?");
    $emailerobject->sendEmail();
?>
```

In the above code piece, we first created an instance of `Emler` class to a variable name `$emailerobject` in the first line. Here, there is something important to note: we are supplying a sender address while instantiating this:

```
$emailerobject = new Emler("hasin@pageflakes.com");
```

Remember we had a constructor method in our class as `__construct($sender)`.

When initiating an object, we said that the constructor method is called automatically. So while initiating this `Emailer` class we must supply the proper arguments as declared in the constructor method. For example the following code will create a warning:

```
<?
    $emailer = new Emailer();
?>
```

When you execute the above code, it shows the warning as follows:

```
Warning: Missing argument 1 for emailer::__construct(),
called in C:\OOP with PHP5\Codes\ch1\class.emailer.php on line 42
and defined in <b>C:\OOP with PHP5\Codes\ch1\class.emailer.php</b>
on line <b>9</b><br />
```

See the difference? If your class had no constructor method or a constructor with no arguments, you can instantiate it with the above code.

## Modifiers

You have seen that we used some keywords like `private` or `public` in our class. So what are these and why do we need to use them? Well, these keywords are called modifiers and were introduced in PHP5. They were not available in PHP4. These keywords help you to define how these variables and properties will be accessed by the user of this class. Let's see what these modifiers actually do.

**Private:** Properties or methods declared as `private` are not allowed to be called from outside the class. However any method inside the same class can access them without a problem. In our `Emailer` class we have all these properties declared as `private`, so if we execute the following code we will find an error.

```
<?
    include_once("class.emailer.php");
    $emobject = new Emailer("hasin@somewherein.net");
    $emobject->subject = "Hello world";
?>
```

The above code upon execution gives a fatal error as shown below:

```
<b>Fatal error</b>: Cannot access private property emailer::$subject
in <b>C:\OOP with PHP5\Codes\ch1\class.emailer.php</b> on line
<b>43</b><br />
```

That means you can't access any `private` property or method from outside the class.

**Public:** Any property or method which is not explicitly declared as `private` or `protected` is a `public` method. You can access a `public` method from inside or outside the class.

**Protected:** This is another modifier which has a special meaning in OOP. If any property or method is declared as `protected`, you can only access the method from its subclass. We will learn details about subclass later in this chapter. But to see how a protected method or property actually works, we'll use the following example.

To start, let's open `class.emailer.php` file (the `Emailer` class) and change the declaration of the `$sender` variable. Make it as follows:

```
protected $sender
```

Now create another file name `class.extendedemailer.php` with the following code:

```
<?
class ExtendedEmailer extends emailer
{
    function __construct(){}
    public function setSender($sender)
    {
        $this->sender = $sender;
    }
}
?>
```

Now use this object like this:

```
<?
include_once("class.emailer.php");
include_once("class.extendedemailer.php");
$xemailer = new ExtendedEmailer();
$xemailer->setSender("hasin@pageflakes.com");
$xemailer->addRecipients("hasin@somewherein.net");
$xemailer->setSubject("Just a Test");
$xemailer->setBody("Hi Hasin, How are you?");
$xemailer->sendEmail();
?>
```

Now if you look carefully at the code of the `ExtendedEmailer` class, you will find that we accessed the `$sender` property of its parent (which is actually `Emailer` class).

We have been able to access that property only because it was declared as [protected](#).

One more benefit we get here, is that the property `$sender` is still inaccessible directly from outside the scope of these two classes. That means if we execute the following code, it will generate a fatal error.

```
<?
include_once("class.emailer.php");
include_once("class.extendedemailer.php");
$xemailer = new ExtendedEmailer();
$xemailer->sender = "hasin@pageflakes.com";
?>
```

Upon execution, it gives the following error:

```
<b>Fatal error</b>: Cannot access protected property
extendedEmailer::$sender in <b>C:\OOP with
PHP5\Codes\chl\test.php</b> on line <b>5</b><br />
```

[...]

Hasin Hayder, *Object-Oriented Programming with PHP5*, PACKT Publishing, 2007

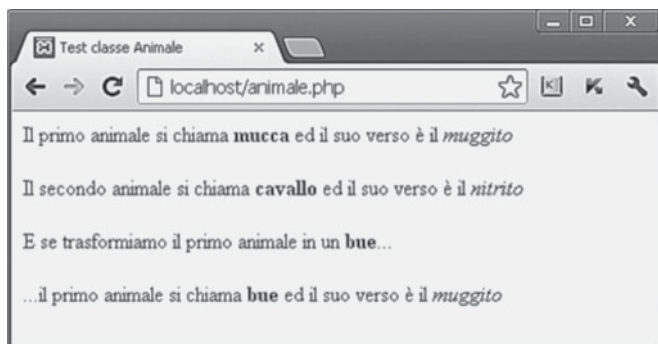
## QUESTIONS

- a** What does the `$this` notation mean?
- b** What does the `protected` modifier mean?
- c** What is a constructor method in PHP?
- d** What is the difference between the “public” and “private” modifiers?

# La programmazione a oggetti nel linguaggio PHP

# B2

La seguente pagina web dinamica



è generata dal seguente codice PHP/HTML:

```
<?php
class Animale {

    // attributi
    private $nome;
    private $verso;

    // costruttore
    public function Animale($nome, $verso) {
        $this->nome = $nome;
        $this->verso = $verso;
    }

    // metodi
    public function setNome($nome) {
        $this->nome = $nome;
    }

    public function setVerso($verso) {
        $this->verso = $verso;
    }

    public function getNome() {
        return $this->nome;
    }
}
```



```

        public function getVerso() {
            return $this->verso;
        }
    }
?>

<html>
<head>
    <title>Test classe Animale</title>
</head>
<body>
    <?php
        $animale1 = new Animale("mucca", "muggito");
        echo "Il primo animale si chiama <strong>".
            $animale1->getNome().
            "</strong> ed il suo verso è il <em>".
            $animale1->getVerso()."</em><br><br>";

        $animale2 = new Animale("cavallo", "nitrito");
        echo "Il secondo animale si chiama <strong>".
            $animale2->getNome().
            "</strong> ed il suo verso è il <em>".
            $animale2->getVerso()."</em><br><br>";

        echo "E se trasformiamo il primo animale in un
            <strong>bue</strong>...<br><br>";
        $animale1->setNome("bue");
        echo "...il primo animale si chiama <strong>".
            $animale1->getNome().
            "</strong> ed il suo verso è il <em>".
            $animale1->getVerso()."</em><br><br>";
    ?>
</body>
</html>

```

Il codice è stato suddiviso logicamente in due parti:

- la prima relativa alla definizione della «classe» `Animale`, nella quale si individuano le sezioni relative agli attributi e ai metodi (compreso il costruttore);
- l'altra relativa alla visualizzazione della pagina web; in questa seconda parte la classe `Animale` viene utilizzata come modello per creare «oggetti» che comprendono le variabili e le funzioni interne invocabili con notazioni come `$animale1->...`

Questa struttura è realizzata in accordo con quella classica della programmazione orientata agli oggetti (OOP, *Object Oriented Programming*) che le ultime versioni del linguaggio PHP consentono di implementare.

# 1 Classi e oggetti nel linguaggio PHP

Analizziamo ora come il linguaggio PHP gestisce gli elementi fondamentali della programmazione orientata agli oggetti.

**Classe.** Il concetto di classe è quello standard di modello per la descrizione di un tipo di oggetti e viene realizzato definendone le «proprietà», o «attributi», e le caratteristiche della sua interfaccia attraverso i «metodi». Anche nel linguaggio PHP, la creazione di un nuovo oggetto è resa possibile dall'operatore **new**, che invoca automaticamente il «costruttore» per l'inizializzazione del valore degli attributi dell'oggetto.

**Proprietà (o attributi).** Una proprietà, o attributo, è una variabile, o una costante, PHP il cui valore in un certo istante determina, in parte, lo stato di un oggetto istanza di una classe.

ES.

Il diverso valore degli attributi *nome* e *verso* della classe *Animale* determina lo stato degli oggetti *\$animale1* e *\$animale2*.

**Metodi.** Anche in PHP i metodi sono realizzati tramite funzioni che, applicate agli oggetti, implementano l'aspetto operativo di una classe. L'invocazione di un metodo avviene con una istruzione del tipo:

```
$nomeOggetto->nomeMetodo(parametro1, parametro2, ...);
```

**Incapsulamento.** La classica tecnica dell'incapsulamento che realizza il principio di *information hiding*, separando l'interfaccia esposta da un oggetto all'esterno dalla sua implementazione interna, viene implementato anche in PHP classificando i membri di una classe (attributi e metodi) in pubblici (**public**) o privati (**private**).

## 1.1 Classi e oggetti nel linguaggio PHP

Nell'esempio di apertura del capitolo è stata utilizzata una sola classe, ma nello sviluppo di un'applicazione PHP è possibile utilizzare più classi, ognuna delle quali finalizzata alla creazione di uno specifico tipo di oggetti. In generale è preferibile inserire la definizione di una classe in un file distinto da includere nel file dello script che ne istanzia gli oggetti.

ESEMPIO

Con riferimento all'esempio introduttivo il codice della classe *Animale* può essere inserito in un file denominato «animale.php»; lo script che genera la pagina web dinamica diviene:

```
<html>
<head>
<title>Test classe Animale</title>
```

```

</head>
<body>
<?php
    require_once("animale.php");
    $animale1 = new Animale("mucca", "muggito");
    echo "Il primo animale si chiama <strong>".
        $animale1->getNome().
        "</strong> ed il suo verso è il <em>".
        $animale1->getVerso()."</em><br><br>";

    $animale2 = new Animale("cavallo", "nitrito");
    echo "Il secondo animale si chiama <strong>".
        $animale2->getNome().
        "</strong> ed il suo verso è il <em>".
        $animale2->getVerso()."</em><br><br>";

    echo "E se trasformiamo il primo animale in un
        <strong>bue</strong>...<br><br>";
    $animale1->setNome("bue");
    echo "...il primo animale si chiama <strong>".
        $animale1->getNome().
        "</strong> ed il suo verso è il <em>".
        $animale1->getVerso()."</em><br><br>";

?>
</body>
</html>

```

Nel codice dello script:

- `require_once("animale.php")` è l'istruzione che include il file di definizione della classe `Animale`;
- `$animale1 = new Animale("mucca", "muggito")` è l'istruzione di creazione dell'oggetto `$animale1` istanza della classe `Animale` a cui sono forniti i parametri «mucca» e «muggito» come valori iniziali degli attributi *nome* e *verso*;
- `$animale2 = new Animale("cavallo", "nitrito")` è l'istruzione di creazione dell'oggetto `$animale2` istanza della classe `Animale` a cui sono forniti i parametri «cavallo» e «nitrito» come valori iniziali degli attributi *nome* e *verso*;
- `$animale1->getNome()` e `$animale2->getNome()` sono le espressioni relative rispettivamente all'applicazione del metodo `getNome` agli oggetti `$animale1` e `$animale2` per ottenere il valore dell'attributo *nome*;
- `$animale1->getVerso()` e `$animale2->getVerso()` sono le espressioni relative all'applicazione del metodo `getVerso` rispettivamente agli oggetti `$animale1` e `$animale2` per ottenere il valore dell'attributo *verso*;
- `$animale1->setNome("bue")` è l'istruzione che invoca il metodo `setNome` all'oggetto `$animale1` per modificare il valore dell'attributo *nome* in «bue».

**OSSERVAZIONE** Dal momento che gli attributi *nome* e *verso* della classe `Animale` sono stati dichiarati di tipo **private**, è necessario ricorrere ai metodi `getNome`, `getVerso`, `setNome` e `setVerso` per accedere ad essi e modificarne il valore. Secondo una prassi consolidata i metodi sono denominati con il prefisso `get/set` seguito dal nome dell'attributo.

Nel caso in cui un attributo sia dichiarato di tipo **public** (pratica sconsigliata perché è sempre bene che sia il codice di un metodo a controllare l'accesso al valore di un attributo allo scopo di garantirne la validità dei valori) è possibile fare direttamente riferimento a esso con una notazione come la seguente:

```
$nomeOggetto->nomeAttributo
```

Per esempio, se gli attributi *nome* e *verso* della classe `Animale` fossero stati entrambi dichiarati di tipo **public**, sarebbe stato possibile accedervi con espressioni e istruzioni come le seguenti:

```
animale1->verso  
animale1->nome="bue"
```

Effettuando l'analisi del codice PHP che definisce la classe `Animale`

```
<?php  
class Animale {  
  
    // attributi  
    private $nome;  
    private $verso;  
  
    // costruttore  
    public function Animale($nome, $verso) {  
        $this->nome = $nome;  
        $this->verso = $verso;  
    }  
  
    // metodi  
    public function setNome($nome) {  
        $this->nome = $nome;  
    }  
  
    public function setVerso($verso) {  
        $this->verso = $verso;  
    }  
  
    public function getNome() {  
        return $this->nome;  
    }  
  
    public function getVerso() {  
        return $this->verso;  
    }  
}  
?>
```



i commenti evidenziano le sezioni «attributi», «costruttore» e «metodi»:

- **attributi**: sono definite come variabili di tipo **private** le proprietà *nome* e *verso*;
- **metodi**: sono definite le funzioni di tipo **public** per il recupero (*get*) e l'impostazione (*set*) del valore degli attributi privati *nome* e *verso*;
- **costruttore**: è definito il costruttore, il metodo avente lo stesso nome della classe invocato automaticamente al momento della creazione di un oggetto mediante l'operatore **new** che in questo caso inizializza gli attributi *nome* e *verso* con il valore dei parametri omonimi. Nelle versioni più recenti del linguaggio PHP la dichiarazione del costruttore con lo stesso nome della classe è deprecata: dovrebbe essere usato il nome predefinito `__construct`.

#### ESEMPIO

Nel caso della classe `Animale` si dovrebbe avere il seguente costruttore:

```
// costruttore
public function __construct($nome, $verso) {
    $this->nome = $nome;
    $this->verso = $verso;
}
```

La variabile predefinita `$this` riferisce l'oggetto su cui il metodo, o il costruttore, viene invocato; viene normalmente utilizzata per fare riferimento agli attributi di un oggetto nel codice dei metodi di una classe.

**OSSERVAZIONE** Le considerazioni svolte per la visibilità degli attributi valgono anche per i metodi: un metodo di tipo **public** può essere invocato dal codice esterno alla classe, mentre un metodo di tipo **private** può essere invocato esclusivamente dal codice di un metodo della classe stessa.

Il linguaggio PHP permette la definizione di un metodo «distruttore» così dichiarato all'interno della definizione di una classe:

```
public function __destruct() {...}
```

Come il metodo costruttore viene invocato automaticamente al momento della creazione di un oggetto, il metodo distruttore viene invocato automaticamente quando un oggetto cessa di esistere, per esempio quando alla variabile che lo riferisce viene assegnato un valore o un oggetto diverso. Compito del codice del distruttore è quello di recuperare l'area di memoria assegnata all'oggetto e rilasciare eventuali risorse assegnate all'oggetto stesso che altrimenti rimarrebbero inutilizzabili, come, per esempio, un file aperto dal codice del costruttore o di un metodo e che deve necessariamente essere chiuso correttamente al momento dell'eliminazione dell'oggetto stesso.



La versione della classe `Animale` che segue – definita nel file «animale.php» – esemplifica l'uso del metodo distruttore, che in questo caso inserisce nella pagina dinamica un messaggio:

```
<?php
class Animale {

    // attributi
    private $nome;
    private $verso;

    // costruttore
    public function __construct($nome, $verso) {
        $this->nome = $nome;
        $this->verso = $verso;
    }

    // distruttore
    public function __destruct() {
        echo "L'oggetto <em>". $this->nome.
            "</em> &grave; stato distrutto.<br>";
    }

    // metodi
    public function setNome($nome) {
        $this->nome = $nome;
    }

    public function setVerso($verso) {
        $this->verso = $verso;
    }

    public function getNome() {
        return $this->nome;
    }

    public function getVerso() {
        return $this->verso;
    }
}
?>
```

Il seguente script PHP

```
<html>
<head>
    <title>Test classe Animale</title>
</head>
<body>
    <?php
        require_once("animale.php");
        $animale = new Animale("mucca", "muggito");
```



```

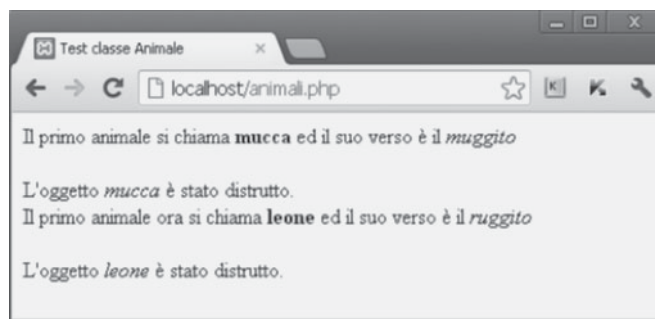
echo "Il primo animale si chiama <strong>".
    $animale->getNome().
    "</strong> ed il suo verso è il <em>".
    $animale->getVerso()."</em><br><br>";

$animale = new Animale("leone", "ruggito");
echo "Il primo animale ora si chiama <strong>".
    $animale->getNome().
    "</strong> ed il suo verso è il <em>".
    $animale->getVerso()."</em><br><br>";

?>
</body>
</html>

```

produce la seguente pagina web dinamica:



che esemplifica il comportamento dei metodi costruttore e distruttore:

- la prima riga della pagina è generata dopo che l'oggetto *animale* è stato creato;
- l'istruzione `$animale = new Animale("leone", "ruggito")` crea un nuovo oggetto utilizzando la stessa variabile *animale*: dato che questa riferiva precedentemente un oggetto diverso (l'animale denominato «mucca»), ne implica automaticamente la distruzione che visualizza il messaggio presente nella seconda riga della pagina;
- la terza riga visualizza il nuovo stato dell'oggetto riferito dalla variabile *animale*;
- l'ultima riga riporta il messaggio relativo alla distruzione automatica dell'oggetto riferito dalla variabile *animale* al termine dello script.

**OSSERVAZIONE** Se non sono richieste operazioni specifiche non è necessario dichiarare esplicitamente il distruttore di una classe, perché il linguaggio PHP ne genera automaticamente uno predefinito che ha il compito di recuperare l'area di memoria assegnata all'oggetto stesso in modo del tutto trasparente, una volta che questo non è più riferito da alcuna variabile.

In tutti quei casi in cui è pratico utilizzare messaggi per visualizzare lo stato degli oggetti istanza di una classe, conviene definire un metodo che restituisce una stringa di descrizione dello stato dell'oggetto.

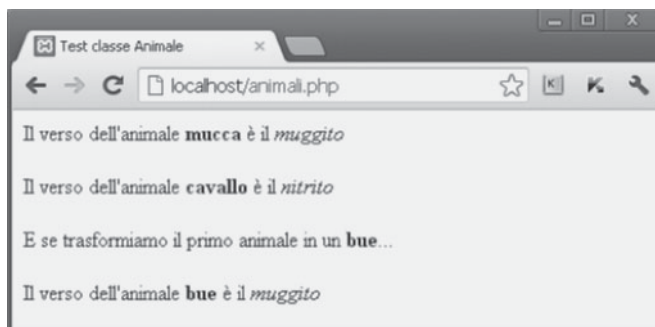
Alla definizione della classe `Animale` può essere aggiunto il seguente metodo:

```
public function toString() {
    return "Il verso dell'animale <strong>".
        $this->getNome().
        "</strong> &grave; il <em>".
        $this->getVerso()."</em><br>";
}
```

Il seguente script PHP

```
<html>
<head>
    <title>Test classe Animale</title>
</head>
<body>
    <?php
        require_once("animale.php");
        $animale1 = new Animale("mucca", "muggito");
        echo $animale1->toString();
        echo "<br>";
        $animale2 = new Animale("cavallo", "nitrito");
        echo $animale2->toString();
        echo "<br>";
        echo "E se trasformiamo il primo animale in un
            <strong>bue</strong>...<br><br>";
        $animale1->setNome("bue");
        echo $animale1->toString();
    ?>
</body>
</html>
```

produce la seguente pagina web dinamica:



**OSSERVAZIONE** Se allo scopo di produrre una stringa che descriva lo stato di un oggetto si dichiara un metodo denominato con il nome predefinito `__toString`, per esempio

```
public function toString() {
    return "Il verso dell'animale <strong>".
        $this->getNome().
        "</strong> &grave; il <em>".
        $this->getVerso()."</em><br>";
}
```

è possibile impiegare nelle istruzioni di output (**echo** e **print**) direttamente il nome della variabile che riferisce l'oggetto, come nella seguente istruzione:

```
echo $animale1;
```

## 1.2 Uguaglianza e copia di oggetti

Il linguaggio PHP fornisce due distinti operatori per verificare l'uguaglianza tra variabili: «==» e «===». Il primo verifica il contenuto delle variabili (uguaglianza), mentre il secondo verifica che, oltre ad avere lo stesso contenuto, le variabili abbiano anche lo stesso tipo (identità).

Una distinzione simile viene applicata alle variabili che riferiscono oggetti: l'operatore di uguaglianza verifica semplicemente che gli attributi di due oggetti diversi istanza di una stessa classe assumano gli stessi valori, mentre l'operatore di identità verifica che si tratti dello stesso oggetto.

**OSSERVAZIONE** Due oggetti identici sono ovviamente anche uguali.

### ESEMPIO

Dato il seguente frammento di script PHP riferito alla classe `Animale` degli esempi precedenti

```
$animale1 = new Animale("mucca", "muggito");  
$animale2 = new Animale("leone", "ruggito");  
$animale3 = new Animale("mucca", "muggito");  
$animale4 = $animale1;
```

- le variabili `$animale1` e `$animale2` riferiscono oggetti che non sono né uguali né identici, per cui le espressioni logiche

```
$animale1==$animale2 e $animale1=== $animale2
```

sono entrambe false;

- le variabili `$animale1` e `$animale3` riferiscono oggetti che sono uguali (stessi valori per gli stessi attributi), ma non sono identici (si riferiscono a oggetti diversi), per cui l'espressione logica

```
$animale1==$animale3
```

è vera, mentre l'espressione logica

```
$animale1=== $animale3
```

è falsa;

- le variabili `$animale1` e `$animale4` riferiscono lo stesso oggetto, per cui le espressioni logiche

```
$animale1==$animale4 e $animale1=== $animale4
```

sono entrambe vere.

**OSSERVAZIONE** Anche in PHP è necessario effettuare una separazione concettuale tra la variabile che riferisce un oggetto e l'oggetto stesso perché, come dimostra l'esempio precedente, variabili diverse possono riferire uno stesso oggetto. In altre parole, un oggetto è istanza di una classe, mentre una variabile che riferisce un oggetto è il mezzo tramite il quale è possibile accedere a esso.

Nel linguaggio PHP un oggetto che non è più riferito da alcuna variabile viene automaticamente distrutto.

In molti casi è necessario creare un nuovo oggetto sulla base delle informazioni che sono memorizzate in un altro oggetto istanza della stessa classe; questa è classica operazione di «clonazione» viene realizzata nel linguaggio PHP tramite l'operatore **clone**:

```
$nuovo_oggetto = clone $vecchio_oggetto;
```

#### ESEMPIO

Dato il seguente frammento di script PHP riferito alla classe `Animale` degli esempi precedenti

```
$animale1 = new Animale("mucca", "muggito");  
$animale2 = $animale1;  
$animale3 = clone $animale1;
```

- la seconda istruzione fa in modo che due variabili distinte (`$animale2` e `$animale1`) riferiscano lo stesso oggetto;
- la terza istruzione crea un nuovo oggetto riferito dalla variabile `$animale3` e avente gli stessi valori degli attributi dell'oggetto riferito dalla variabile `$animale1`.

## 1.3 Attributi e metodi statici; costanti

Il linguaggio PHP permette la definizione di attributi e metodi il cui ciclo di vita si riferisce alla classe stessa piuttosto che ai singoli oggetti istanziati:

**Attributi statici.** Attributi a cui è necessario accedere direttamente utilizzando il nome della classe; un attributo statico deve essere inizializzato contestualmente alla dichiarazione.

**Metodi statici.** Metodi che possono essere invocati direttamente utilizzando il nome della classe, anche senza avere istanziato oggetti. La variabile predefinita `$this` non può essere utilizzata nel codice di un metodo statico; la parola chiave **self**:: consente di riferire la classe cui il metodo appartiene.

Nella seguente classe PHP



```
<?php
class Pubblicazione {

    // attributi
    private $titolo;
    private $numero_pagine;
    private static $costo_per_pagina = 0.02;

    // costruttore
    public function __construct($titolo, $numero_pagine) {
        $this->titolo = $titolo;
        $this->numero_pagine = $numero_pagine;
    }

    // metodi
    public function setTitolo($titolo) {
        $this->titolo = $titolo;
    }

    public function setNumero_pagine($numero_pagine) {
        $this->numero_pagine = $numero_pagine;
    }

    public static function setCosto_per_pagina($costo) {
        self::$costo_per_pagina = $costo;
    }

    public function getTitolo() {
        return $this->titolo;
    }

    public function getNumero_pagine() {
        return $this->numero_pagine;
    }

    public static function getCosto_per_pagina() {
        return self::$costo_per_pagina;
    }

    public function prezzo() {
        return self::$costo_per_pagina*$this->numero_pagine;
    }

    public function __toString() {
        return "Pubblicazione: ".$this->titolo.
            " pagine: ".$this->numero_pagine.
            " costo pagina: ".self::$costo_per_pagina.
            " prezzo: ".$this->prezzo();
    }
}
?>
```

l'attributo `$costo_per_pagina` è statico e fissato inizialmente al valore di 0.05; anche modificando tale valore tutti gli oggetti della classe condividono il medesimo costo per pagina. I metodi statici `setCosto_per_pagina` e `getCosto_per_pagina` permettono di accedere all'attributo `$costo_per_pagina`.

**OSSERVAZIONE** Il significato di metodi e attributi statici è che essi fanno parte della classe stessa e non degli oggetti istanza della classe: essi esistono indipendentemente dall'avere o meno istanziati oggetti della classe. È per questo motivo che è necessario un modo per accedere alle proprietà e alle funzioni statiche senza riferire un oggetto specifico dato dalla parola chiave **self**:<sup>1</sup> che, impiegata nel codice di un metodo, riferisce la classe a cui il metodo appartiene, contrariamente alla variabile `$this` che riferisce l'oggetto su cui il metodo è invocato.

1. In generale la notazione «::» consente di invocare i metodi statici di una classe premettendo il nome della classe stessa.

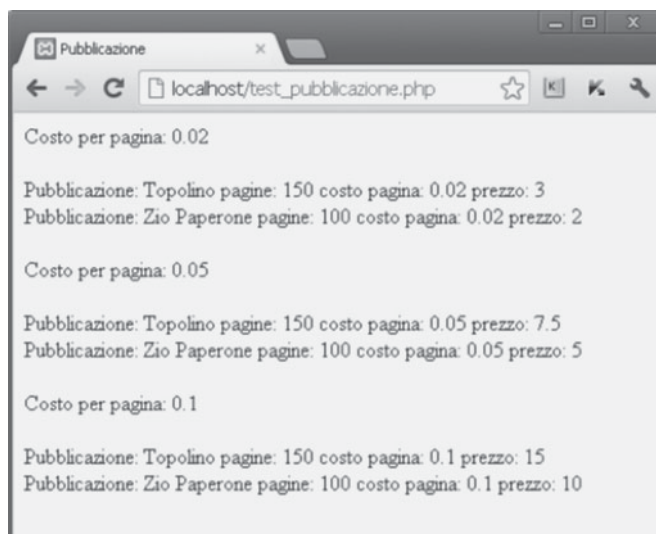


## ESEMPIO

Il seguente script permette di verificare le funzionalità della classe `Pubblicazione`

```
<html>
  <head>
    <title>Pubblicazione</title>
  </head>
  <body>
    <?php
      require_once("pubblicazione.php");
      echo "Costo per pagina: ".Pubblicazione::getCosto_per_pagina()."<br><br>";
      $pubblicazione1 = new Pubblicazione("Topolino",150);
      $pubblicazione2 = new Pubblicazione("Zio Paperone",100);
      echo $pubblicazione1."<br>";
      echo $pubblicazione2."<br><br>";
      Pubblicazione::setCosto_per_pagina(0.05);
      echo "Costo per pagina: ".Pubblicazione::getCosto_per_pagina()."<br><br>";
      echo $pubblicazione1."<br>";
      echo $pubblicazione2."<br><br>";
      $pubblicazione1->setCosto_per_pagina(0.1);
      echo "Costo per pagina: ".Pubblicazione::getCosto_per_pagina()."<br><br>";
      echo $pubblicazione1."<br>";
      echo $pubblicazione2."<br><br>";
    ?>
  </body>
</html>
```

visualizzando la seguente pagina web dinamica:





**OSSERVAZIONE** Nell'esempio precedente si osserva che:

- l'invocazione del metodo statico `Pubblicazione::getCosto_per_pagina()` restituisce il valore corrente dell'attributo statico `costo_per_pagina`;
- modificando il valore dell'attributo statico `costo_per_pagina` a livello di classe con l'istruzione `Pubblicazione::setCosto_per_pagina(0.05)` il cambiamento viene recepito da tutti gli oggetti istanza della classe;
- modificando il valore dell'attributo statico `costo_per_pagina` a livello di singolo oggetto con l'istruzione `$pubblicazione1->setCosto_per_pagina(0.1)` il cambiamento viene recepito da tutti gli oggetti istanza della classe.

Un caso particolare di attributi statici è rappresentato dalle costanti dichiarate utilizzando la parola chiave `const`; dato che una costante è implicitamente un attributo che non può cambiare valore, essa rappresenta un elemento statico condiviso da tutti gli oggetti istanza della classe in cui è dichiarato.

**ESEMPIO**

Lo script PHP che segue esemplifica l'uso delle costanti:

```
<?php
class Costanti {
    const PI_GRECO = 3.1416;
}
echo Costanti::PI_GRECO;
?>
```

## 1.4 I metodi «magici»

Il linguaggio PHP prevede per le classi metodi intrinsecamente definiti denominati «magici». Sono caratterizzati dal fatto che il loro nome inizia con il prefisso «`__`» e che vengono invocati automaticamente in specifiche circostanze (per esempio il metodo `__construct` viene invocato al momento della creazione di un nuovo oggetto).

I metodi magici sono definiti di tipo *overloading* se hanno la possibilità di creare dinamicamente nuovi attributi o metodi di una classe.

La **TABELLA 1** riporta alcuni dei principali metodi magici per le classi del linguaggio PHP.

I metodi magici di tipo *overloading* sono automaticamente applicati a metodi e attributi «inaccessibili», cioè inesistenti, o acceduti senza rispettare il loro campo di visibilità. Accedendo a un attributo o invocando un metodo pubblico i metodi magici di tipo *overloading* non sono chiamati in causa; invece accedendo a un attributo o invocando un metodo privato dall'esterno della classe sono automaticamente invocati i metodi magici di tipo *overloading*.

TABELLA 1

Metodo	Invocato quando ...	Overloading
<code>__construct</code>	... viene creato un nuovo oggetto istanza della classe	No
<code>__destruct</code>	... viene distrutto un oggetto istanza della classe	No
<code>__set</code>	... viene modificato un attributo della classe inaccessibile	Sì
<code>__get</code>	... viene acceduto un attributo della classe inaccessibile	Sì
<code>__isset</code>	... viene invocata la funzione predefinita <code>isset</code> con argomento un attributo della classe inaccessibile	Sì
<code>__unset</code>	... viene invocata la funzione predefinita <code>unset</code> con argomento un attributo della classe inaccessibile	Sì
<code>__toString</code>	... viene utilizzato il riferimento a un oggetto istanza della classe nel contesto di un'espressione che richiede una stringa	No
<code>__clone</code>	... viene clonato un oggetto istanza della classe utilizzando l'operatore <code>clone</code>	No

I principali metodi di questo tipo sono `__get` e `__set`, invocati quando un'istruzione tenta di leggere o scrivere il valore di un attributo inaccessibile.

Per esempio l'esecuzione dello script PHP

```
<?php
class Studente {
    public $nome;
}

$studente = new Studente();
$studente->nome = "Marco";
$studente->email = "marco@scuola.it";
?>
```

contrariamente a quanto ci si potrebbe aspettare per confronto con altri linguaggi di programmazione orientati agli oggetti, non genera un errore. L'accesso diretto del codice esterno alla classe `Studente` all'attributo `nome` è legale in quanto esso è pubblico, ma l'attributo `email` è inesistente.

L'istruzione

```
$studente->email = "marco@scuola.it";
```

invoca automaticamente il metodo magico `__set` con parametri «email» e «marco@scuola.it» che crea un nuovo attributo denominato `email` a cui assegna il valore «marco@scuola.it».

**OSSERVAZIONE** Un attributo creato dal metodo magico `__set` esiste per il solo oggetto su cui è stato invocato e non per la classe di cui l'oggetto è istanza. Nel caso dell'esempio precedente l'attributo *email* esiste per il solo oggetto `$studente`.

Sempre facendo riferimento alla classe `Studente` e all'oggetto `$studente` dello script precedente, l'esecuzione dell'istruzione

```
echo $studente->telefono;
```

invoca automaticamente il metodo magico `__get` con parametro «telefono» che in questo caso – data l'inesistenza dell'attributo `telefono` – genera un messaggio di errore senza interrompere l'esecuzione dello script. È possibile «personalizzare» il comportamento dei metodi magici `__set` e `__get` definendoli esplicitamente nel codice della classe come nel seguente script, che implementa una classe per i cui oggetti è possibile definire dinamicamente attributi in modo indipendente l'uno dall'altro:

```
<?php
class Studente {
    public $nome;
    private $dati = array();

    public function __set($dato, $valore) {
        $this->dati[$dato] = $valore;
    }

    public function __get($dato) {
        if (isset($this->dati[$dato]))
            return $this->dati[$dato];
        else
            return "";
    }

    public function __unset($dato) {
        unset($this->dati[$dato]);
    }
}

$studente = new Studente();
$studente->nome = "Marco";
$studente->email = "marco@scuola.it";
$studente->telefono = "0123456789";
?>
```

È possibile fare riferimento agli attributi «dinamici» della classe con istruzioni del tipo:

```
echo "Nome: ".$studente->nome."<br>";
echo "E-mail: ".$studente->email."<br>";
echo "Telefono: ".$studente->telefono."<br>";
```

**OSSERVAZIONE** La ridefinizione del metodo magico `__unset` nella classe `Studente` consente di «eliminare» un attributo dinamico non più necessario.

Tra i metodi magici non di tipo *overloading* sono stati già introdotti i metodi `__construct`, `__destruct` e `__toString`; prendiamo ora in esame il metodo `__clone` che viene automaticamente invocato quando si ricorre all'operatore `clone` per copiare un oggetto: la ridefinizione di questo metodo magico consente di adattare la copia alle caratteristiche specifiche della classe di cui l'oggetto è istanza.

#### ESEMPIO

Se un oggetto ha come attributo un altro oggetto, la clonazione non comporta la clonazione dell'oggetto attribuito; il seguente script

```
<?php
class Data {
    public $giorno;
    public $mese;
    public $anno;
}

class Persona {
    public $cognome;
    public $nome;
    public $sesso;
    public $data_nascita;
    public $luogo_nascita;
}

$personal = new Persona();
$personal->nome = "Giorgio";
$personal->cognome = "Meini";
$personal->sesso = "M";
$personal->data_nascita->giorno = 23;
$personal->data_nascita->mese = 3;
$personal->data_nascita->anno = 1965;
$personal->luogo_nascita = "Livorno";
$persona2 = clone $personal;
$persona2->nome = "Fiorenzo";
$persona2->cognome = "Formichi";
$persona2->data_nascita->giorno = 6;
$persona2->data_nascita->mese = 10;
$persona2->data_nascita->anno = 1954;
?>
```

crea due oggetti distinti in cui la data di nascita è lo stesso oggetto; la variazione dell'attributo `data_nascita` nell'oggetto `$persona2` comporta la stessa variazione nell'oggetto `$personal`. La ridefinizione del metodo magico `__clone` nella classe `Persona` permette di evitare questo inaspettato effetto collaterale della clonazione: ►

```

<?php
class Data {
    public $giorno;
    public $mese;
    public $anno;
}

class Persona {
    public $cognome;
    public $nome;
    public $sesso;
    public $data_nascita;
    public $luogo_nascita;

    public function __clone() {
        $data = new Data();
        $data->giorno = $this->data_nascita->giorno;
        $data->mese = $this->data_nascita->mese;
        $data->anno = $this->data_nascita->anno;
        $this->data_nascita = $data;
    }
}

$personal = new Persona();
$personal->nome = "Giorgio";
$personal->cognome = "Meini";
$personal->sesso = "M";
$personal->data_nascita->giorno = 23;
$personal->data_nascita->mese = 3;
$personal->data_nascita->anno = 1965;
$personal->luogo_nascita = "Livorno";
$persona2 = clone $personal;
$persona2->nome = "Fiorenzo";
$persona2->cognome = "Formichi";
$persona2->data_nascita->giorno = 6;
$persona2->data_nascita->mese = 10;
$persona2->data_nascita->anno = 1954;

?>

```

Il metodo magico `__clone` viene invocato dopo la creazione del nuovo oggetto e viene in questo caso utilizzato per creare un nuovo oggetto di classe `Data` distinto da quello dell'oggetto originale clonato.

## 2 Ereditarietà e classi astratte

### 2.1 Ereditarietà

Il linguaggio PHP permette di implementare l'ereditarietà secondo lo schema classico della OOP in cui si permette di definire una nuova classe (classe «derivata») a partire da una classe preesistente in modo da mantenere attributi e metodi della classe originale («superclasse») per eventualmente aggiungerne di nuovi.



A partire dalla seguente classe Animale

```
<?php
class Animale {

    // attributi
    private $nome;
    private $specie;

    // costruttore
    public function __construct($nome, $specie) {
        $this->nome=$nome;
        $this->specie=$specie;
    }

    // metodi
    public function setNome($nome) {
        $this->nome=$nome;
    }

    public function setSpecie($specie) {
        $this->specie=$specie;
    }

    public function getNome() {
        return $this->nome;
    }

    public function getSpecie() {
        return $this->specie;
    }
}
?>
```

è possibile derivare le classi AnimaleTerrestre e AnimaleAcquatico

```
<?php
class AnimaleTerrestre extends Animale {

    // attributi
    private $ambiente;

    // costruttore
    public function __construct($nome, $specie, $ambiente) {
        parent::__construct($nome, $specie);
        $this->ambiente=$ambiente;
    }

    // metodi
    public function setAmbiente($ambiente) {
        $this->ambiente=$ambiente;
    }
}
```



```

    public function getAmbiente() {
        return $this->ambiente;
    }

    public function __toString() {
        return "L'animale ".$this->getNome().
            " si trova in ".$this->getSpecie().
            " e vive nell'ambiente ".$this->getAmbiente();
    }
}

class AnimaleAcquatico extends Animale {
    // attributi
    private $tipo_acqua;

    // costruttore
    public function __construct($nome, $specie, $tipo_acqua) {
        parent::__construct($nome, $specie);
        $this->tipo_acqua = $tipo_acqua;
    }

    // metodi
    public function setTipo_acqua($tipo_acqua) {
        $this->tipo_acqua = $tipo_acqua;
    }

    public function getTipo_acqua() {
        return $this->tipo_acqua;
    }

    public function __toString() {
        return "L'animale ".$this->getNome().
            " si trova in ".$this->getSpecie().
            " e vive in acqua ".$this->getTipo_acqua();
    }
}
?>

```

Lo script PHP che segue include il file «animale.php» contenente le precedenti definizioni delle classi

```

<html>
<head>
    <title> Test ereditarietà </title>
</head>
<body>
    <?php
        require_once("animale.php");
        $animale1 = new AnimaleTerrestre("leone", "mammifero", "savana");
        $animale2 = new AnimaleTerrestre("volpe", "mammifero", "bosco");
        $animale3 = new AnimaleAcquatico("balena", "mammifero", "salata");
        $animale4 = new AnimaleAcquatico("trota", "pesce", "dolce");
    >

```

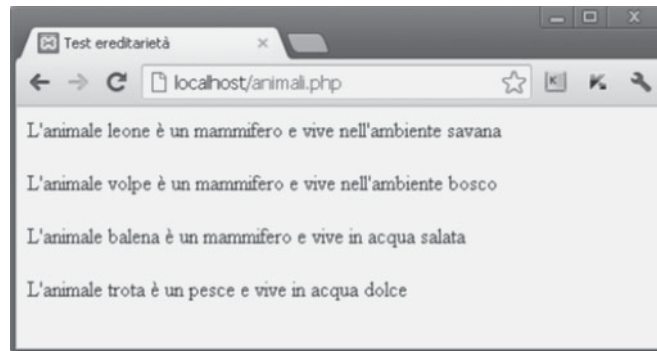
```

        echo $animale1."<br><br>";
        echo $animale2."<br><br>";
        echo $animale3."<br><br>";
        echo $animale4."<br><br>";

    ?>
</body>
</html>

```

e visualizza la seguente pagina dinamica:



Con riferimento al codice dell'esempio precedente:

- la parola chiave **extends** utilizzata nella definizione di una classe permette di derivare attributi e metodi dalla superclasse specificata;
- le classi derivate `AnimaleTerrestre` e `AnimaleAcquatico` ereditano dalla classe `Animale` gli attributi *nome* e *specie* e i metodi `getNome`, `SetNome`, `getSpecie` e `setSpecie`;
- la classe derivata `AnimaleTerrestre` introduce un nuovo attributo *ambiente* e i relativi metodi di accesso;
- la classe derivata `AnimaleAcquatico` introduce un nuovo attributo *tipo\_acqua* e i relativi metodi di accesso;
- il costruttore di entrambe le classi derivate ha come parametri anche i valori degli attributi della superclasse: l'istruzione `parent::__construct(...)` permette di invocare il costruttore della superclasse per inizializzare gli attributi ereditati.

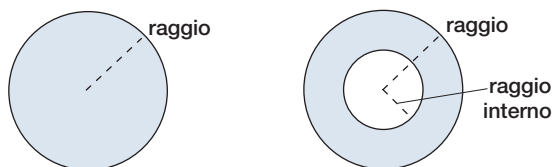
**OSSERVAZIONE** Nel contesto dell'ereditarietà è previsto il tipo di visibilità **protected** che rende accessibile un attributo o un metodo al codice di tutte le classi derivate.

In una classe derivata è possibile ridefinire utilizzando lo stesso nome i metodi ereditati dalla superclasse (*overriding*); i metodi ridefiniti nascondono quelli della superclasse che possono eventualmente essere invocati utilizzando la notazione

`parent::metodo(...)`



Il codice PHP che segue definisce una gerarchia di classi relative a due figure geometriche, il cerchio e la corona circolare:



```
<?php
class Cerchio {
    const PI_GRECO = 3.1416;
    private $raggio;

    public function __construct($raggio) {
        $this->raggio=$raggio;
    }

    public function setRaggio($raggio) {
        $this->raggio=$raggio;
    }

    public function getRaggio() {
        return $this->raggio;
    }

    public function area() {
        return $this->raggio*$this->raggio*Cerchio::PI_GRECO;
    }

    public function circonferenza() {
        return $this->raggio*Cerchio::PI_GRECO*2;
    }
}

class CoronaCircolare extends Cerchio {
    private $raggio_interno;

    public function __construct($raggio, $raggio_interno) {
        parent::__construct($raggio);
        $this->raggio_interno=$raggio_interno;
    }

    public function getRaggio_interno() {
        return $this->raggio_interno;
    }

    public function area() {
        $area_interna=$this->raggio_interno*$this->raggio_interno*Cerchio::PI_GRECO;
        return parent::area()-$area_interna;
    }

    public function circonferenza_interna() {
        return $this->raggio_interno*Cerchio::PI_GRECO*2;
    }
}
?>
```

Il metodo `area` è stato ridefinito nella classe `CoronaCircolare`; nel codice che lo implementa è stato invocato l'omonimo metodo della superclasse utilizzando la parola chiave **parent**.

**OSSERVAZIONE** Per impedire che una classe possa essere estesa è sufficiente definirla come **final class**.

La parola chiave **final** premessa alla definizione di un metodo impedisce che esso possa essere ridefinito in una classe derivata.

## 2.2 Classi astratte

Spesso lo scopo di una gerarchia di classi è fattorizzare componenti comuni a più classi al livello più alto della gerarchia in modo tale che siano ereditati dalle classi di livello più basso: in questo contesto le classi astratte consentono di realizzare la fattorizzazione dei componenti comuni anche nel caso in cui la superclasse risultante non sia «ben definita» e quindi inutilizzabile per istanziare oggetti.

L'uso della classi astratte nel linguaggio PHP è disciplinato dalle seguenti regole:

- una classe viene definita astratta premettendo alla sua dichiarazione la parola chiave **abstract**;
- una classe astratta può avere costruttori, ma non può essere istanziata;
- una classe astratta può avere uno o più «metodi astratti», cioè metodi privi del codice e qualificati con la parola chiave **abstract**; l'implementazione dei metodi astratti è vincolante per le classi derivate;
- se una classe deriva da una classe astratta deve fornire un'implementazione per tutti i metodi astratti ereditati;
- le classi astratte possono essere superclassi o classi derivate di altre classi, astratte e non.

### ESEMPIO

Con riferimento alla classe `Animale` dell'esempio precedente, essa potrebbe essere più propriamente così definita:

```
<?php
abstract class Animale {

    private $nome;
    private $specie;

    public function __construct($nome, $specie) {
        $this->nome=$nome;
        $this->specie=$specie;
    }

    public function setNome($nome) {
        $this->nome=$nome;
    }

    public function setSpecie($specie) {
        $this->specie=$specie;
    }
}
```



```

public function getNome() {
    return $this->nome;
}

public function getSpecie () {
    return $this->specie;
}

abstract function __toString();
}
?>

```

dove la classe Animale:

- viene definita astratta premettendo alla sua dichiarazione la parola chiave **abstract**;
- ha un costruttore ma non può essere istanziata;
- ha il metodo astratto `__toString` che è privo di codice e obbliga a fornire una sua definizione in ogni classe non astratta derivata dalla classe Animale.

### 3 Gestione delle eccezioni

Le eccezioni sono eventi che si manifestano in fase di esecuzione del codice in corrispondenza al verificarsi di errori o attivate dal codice stesso in corrispondenza di determinate situazioni.

Il linguaggio PHP prevede sia l'intercettazione e la gestione delle eccezioni generate, che la possibilità di generarne. In PHP le eccezioni sono istanze della classe predefinita `Exception`, o di una classe da essa derivata.

La gestione delle eccezioni consente di ricevere notifiche relative all'esecuzione di uno script nel caso in cui si verifichino situazioni impreviste e successivamente di disporre dei dettagli relativi alla tipologia di errore verificatosi. Gli oggetti che rappresentano le eccezioni hanno un contenuto informativo che prevede come minimo il codice dell'evento (errore), il nome del file e il numero di riga dell'istruzione che ha generato l'eccezione. Infatti il linguaggio PHP prevede un sistema per la gestione delle eccezioni che pone una particolare attenzione relativamente agli errori prodotti dall'esecuzione degli script, come gli errori sintattici, o generati dall'esecuzione delle istruzioni.

In ogni caso le eccezioni non sono generate necessariamente da errori nel codice, ma anche da comportamenti non voluti o da esiti non previsti in seguito a una richiesta da parte di un client nei confronti del server, come per esempio la richiesta di accesso a un file che non è più disponibile.

Un primo esempio di gestione delle eccezioni in linguaggio PHP è il seguente script:

```

<html>
<head>
  <title> Test eccezioni </title>
</head>
<body>
  <?php
    function divisione($m, $n) {
      if ($n == 0)
        throw new Exception("Divisione per zero.");
      else
        return $m/$n;
    }

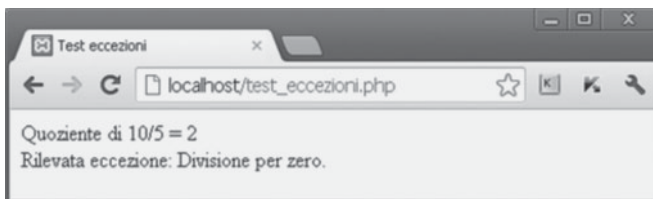
    try {
      echo "Quoziente di 10/5 = ".divisione(10,5)."<br>";
      echo "Quoziente di 10/0 = ".divisione(10,0)."<br>";
    } catch (Exception $e) {
      echo "Rilevata eccezione. ".$e->getMessage().
        "<br>";
    }

  ?>
</body>
</html>

```



che genera la seguente pagina web dinamica:



**OSSERVAZIONE** In assenza della gestione dell'eccezione l'esecuzione dello script PHP avrebbe segnalato un errore al tentativo di esecuzione della seconda divisione.

Esaminando il codice si osserva che:

- il codice della funzione `divisione` controlla che il parametro  $n$  abbia un valore diverso da zero; solo in questo caso viene utilizzato come divisore della divisione richiesta;
- se la funzione `divisione` rileva un valore del divisore  $n$  non accettabile provvede a generare un'eccezione eseguendo l'istruzione `throw new Exception(...)` fornendo come parametro un messaggio esplicativo;
- nel codice sono effettuate due diverse invocazioni della funzione `divisione`: una con un divisore ammissibile (5) e l'altra con un divisore non accettabile (0); il costrutto `try...catch` consente di controllare il corretto esito di esecuzione di un frammento di codice in cui la funzione viene

utilizzata prevedendo eventuali azioni in corrispondenza del verificarsi di eccezioni. La sezione **try** del costrutto comprende il frammento di codice da eseguire in modo controllato, mentre la sezione **catch** viene eseguita solo nel caso che sia intercettata un'eccezione, o il verificarsi di un errore, nell'esecuzione del frammento di codice: in questo caso l'esecuzione del frammento non viene completata;

- indipendentemente dal fatto che si verifichi o meno un'eccezione, il resto dello script viene eseguito e terminato normalmente.

Se le eccezioni potenzialmente generate da un frammento di codice non sono gestite, al verificarsi di un errore possono crearsi due diversi tipi di situazione:

- l'errore è tipo *fatal*: viene emesso un messaggio di errore e lo script viene terminato in maniera anomala;
- l'errore è di tipo *warning*: viene emesso un messaggio di errore, ma l'esecuzione dello script prosegue.

Nel codice della sezione **catch** eseguito al momento in cui l'eccezione viene intercettata è possibile accedere ai metodi resi disponibili dall'oggetto di tipo eccezione; la classe predefinita `Exception` del linguaggio PHP rende disponibili i metodi indicati nella **TABELLA 2**.

**TABELLA 2**

<code>getCode()</code>	Restituisce un valore numerico che costituisce il codice dell'eccezione (nel caso di eccezioni generate con un'istruzione <b>throw</b> è possibile fornire il valore come secondo parametro del costruttore dell'oggetto eccezione)
<code>getFile()</code>	Restituisce il nome e il percorso del file dello script in cui si è verificata l'eccezione
<code>getLine()</code>	Restituisce il numero della riga di codice che ha generato l'eccezione
<code>getMessage()</code>	Restituisce il messaggio di descrizione dell'eccezione (nel caso di eccezioni generate con un'istruzione <b>throw</b> è possibile fornire il valore come primo parametro del costruttore dell'oggetto eccezione)
<code>getTrace()</code>	Restituisce in forma di array numerosi dettagli relativi all'eccezione
<code>getTraceAsString()</code>	È simile al metodo precedente, ma restituisce i dettagli in forma di un'unica stringa

**ESEMPIO**

Il codice che segue è relativo all'implementazione di un meccanismo di gestione delle eccezioni per notificare all'utente l'assenza di una risorsa, nel caso specifico, un file di codice:

```
<?php
    try {
        if (!@include("file.php")) {
            throw new Exception("File non presente.");
        }
    }
    Catch (Exception $e) {
        echo $e->getMessage();
    }
?>
```

■ **Classe.** In PHP, secondo la prassi della programmazione OOP, una classe rappresenta un modello formale per la descrizione di un certo tipo di oggetti definendone gli attributi e i metodi. A partire dalla definizione di una classe è possibile creare («istanziare») oggetti simili che condividono lo stesso insieme di attributi e lo stesso insieme di metodi.

■ **Oggetto.** Istanza di una classe il cui valore degli attributi ne determina lo stato; agli oggetti specifici sono applicati i metodi della classe. A ogni oggetto viene assegnata al momento della creazione un'area di memoria in seguito recuperata e resa nuovamente disponibile per essere riutilizzata quando l'oggetto risulta non essere più accessibile. Nel linguaggio PHP un oggetto viene istanziato con una invocazione del tipo:

```
NomeClasse $nomeOggetto = new NomeClasse(...);
```

■ **Attributi.** Un attributo (o proprietà) è una variabile, o una costante, il cui valore in un determinato istante contribuisce a determinare lo stato di un oggetto. Come avviene per le variabili, i nomi degli attributi nel linguaggio PHP iniziano col simbolo «\$». Il riferimento agli attributi di una classe avviene tramite la variabile predefinita `$this` nella forma `$this->nome_attributo`.

■ **Metodi.** I metodi sono funzioni applicate agli oggetti che realizzano l'aspetto operativo di una classe permettendo l'accesso e la gestione dello stato interno determinato dagli attributi. A differenza degli attributi che assumono valori distinti per i diversi oggetti istanza di una stessa classe, il codice dei metodi è comune a tutti gli oggetti di una classe. L'invocazione di un metodo avviene in PHP con una istruzione del tipo:

```
$nomeOggetto->nomeMetodo(...);
```

■ **Accessibilità dei membri di una classe.** Il linguaggio PHP permette di definire diversi livelli di accessibilità dei membri (attributi e metodi) di una classe: privato, protetto e pubblico. Mediante i livelli di accessibilità viene implementata l'interfaccia degli oggetti rispettando il principio di *information hiding*. Una corretta tecnica di programmazione presuppone la dichiarazione privata degli

attributi di una classe ai quali accedere, per la loro gestione, tramite metodi pubblici per le operazioni *get* e *set*.

■ **Membri statici di una classe.** Il linguaggio PHP permette di definire i membri di una classe (attributi o metodi) di tipo statico. Membri di questo tipo appartengono alla classe e non al singolo oggetto: modificare il valore di un attributo statico significa modificare tale valore per tutte le istanze della classe. Si può far riferimento ai membri statici di una classe con le seguenti notazioni:

```
NomeClasse::nomeAttributo
NomeClasse::nomeMetodo(...)
```

All'interno della classe il riferimento a tali membri è reso possibile dalla parola chiave **self** (che indica la classe stessa) nella seguente forma:

```
self::nomeMembro
```

■ **Costruttore.** Particolare metodo di una classe che viene invocato all'atto della creazione di un oggetto invocando l'operatore **new**. Nel linguaggio PHP, il costruttore ha il nome standard `__construct(...)` e «restituisce» un nuovo oggetto istanza della classe stessa. La sua funzione è quella classica di inizializzazione del valore degli attributi di un oggetto al momento della creazione. Se non viene implementato esplicitamente PHP crea un oggetto invocando un costruttore di default.

■ **Distruzione.** Speciale metodo invocato automaticamente quando un oggetto cessa di esistere (per esempio quando l'unica variabile che lo riferisce viene riassegnata). Compito del distruttore è quello di recuperare l'area di memoria assegnata all'oggetto ed eventualmente rilasciare eventuali risorse che altrimenti rimarrebbero inutilizzabili perché assegnate a un oggetto che non esiste.

■ **Uguaglianza tra oggetti.** Il linguaggio PHP prevede due operatori per verificare l'uguaglianza tra variabili: «==» e «===». Nel caso di variabili assegnate il primo operatore è relativo alla verifica del contenuto delle variabili (uguaglianza), mentre il secondo operatore verifica che le variabili siano anche dello stesso tipo (identità). La differen-

za è estesa alle variabili che riferiscono oggetti nel senso che l'operatore di uguaglianza verifica se gli attributi di due oggetti diversi istanze di una stessa classe assumono gli stessi valori, mentre l'operatore di identità verifica se si tratta o meno dello stesso oggetto: due oggetti identici sono ovviamente anche uguali.

■ **Clonazione.** È l'operazione che permette la creazione di un nuovo oggetto sulla base delle informazioni che sono presenti in un altro oggetto istanza della stessa classe. PHP fornisce uno specifico operatore per clonare oggetti:

```
$nuovo_objetto = clone $oggetto_da_clonare
```

■ **Metodi magici.** Sono metodi predefiniti delle classi del linguaggio PHP che vengono invocati automaticamente in situazioni specifiche (creazione di un oggetto, distruzione di un oggetto, accesso ad attributi non definiti ...). Sintatticamente sono identificati dal prefisso «`__`».

■ **Ereditarietà.** È un meccanismo fondamentale della OOP che permette la creazione di nuove classi (classi derivate) a partire da classi preesistenti (classi base o superclassi) ereditandone attributi e/o metodi, aggiungendone di nuovi, o ridefinendone altri. L'ereditarietà è finalizzata alla creazione di gerarchie di classi e grazie a essa si estende la possibilità di riutilizzare componenti comuni a più classi della stessa gerarchia. In PHP una classe derivata viene dichiarata con la seguente sintassi:

```
class classeDerivata extends superclasse
```

■ **Overriding.** Il linguaggio PHP permette di sovrascrivere un metodo di una superclasse ridefinendolo in una sottoclasse con lo stesso nome. Il metodo della superclasse rimane comunque di-

sponibile e, nel codice della sottoclasse, è possibile invocarlo con la seguente sintassi:

```
parent::metodo
```

■ **Classi astratte.** Una classe astratta si limita a definire l'interfaccia di alcuni (o tutti) metodi dichiarandoli, ma non implementandoli; le classi derivate dovranno obbligatoriamente implementare il codice di questi metodi «astratti». In questo modo una classe astratta «fattorizza» le operazioni comuni a tutte le sue sottoclassi. Una classe astratta non viene creata per istanziare oggetti, ma esclusivamente per derivare altre classi che specificheranno le operazioni in essa solo dichiarate. È infatti impossibile istanziare oggetti a partire da una classe astratta in quanto, essendovi metodi astratti, la loro eventuale invocazione non potrebbe essere eseguita. Il linguaggio PHP permette la definizione di classi astratte premettendo alla loro dichiarazione la parola chiave **abstract**.

■ **Eccezioni.** Sono eventi che si presentano in fase di esecuzione di un programma al verificarsi di situazioni anomale, o di errori. Un'eccezione può essere intercettata e gestita dal codice, oppure il programma termina la sua esecuzione. Nel linguaggio PHP l'intercettazione e gestione delle eccezioni si basa sul costrutto **try/catch**, che permette di definire un blocco di istruzioni la cui esecuzione deve avvenire in modalità «controllata» per intercettare eventuali eccezioni che potrebbero verificarsi: ogni singola clausola **catch** definisce un blocco di istruzioni eseguite per gestire la situazione anomala o di errore connessa alla specifica eccezione intercettata. In PHP le eccezioni sono oggetti istanza di classi che derivano dalla classe predefinita `Exception` e che comprendono un contenuto informativo relativo alla eccezione che rappresentano.

## QUESITI

**1** Una classe nel linguaggio PHP è ...

- A ... il meccanismo attraverso il quale vengono classificati vari tipi di oggetti.
- B ... un modello formale per la descrizione di un certo tipo di oggetti definendone gli attributi e i metodi.
- C ... un modello formale per la descrizione di un certo tipo di oggetti definendone i metodi, ma non gli attributi.
- D Nessuna delle risposte precedenti.

**2** Quali delle seguenti affermazioni relative a un oggetto del linguaggio PHP sono vere?

- A È l'istanza di una specifica classe.
- B È un'entità autonoma, con propri valori per gli attributi che lo caratterizzano.
- C È un'entità autonoma con proprie operazioni che possono differire anche da quelle di altri oggetti simili.
- D Le proprietà di un oggetto descrivono il suo stato.
- E I metodi di un oggetto descrivono il suo stato.
- F I metodi si riferiscono alle funzionalità di un oggetto.

**3** Qual è la funzione dell'operatore PHP `new`?

- A Creare una nuova classe.
- B Creare un nuovo metodo.
- C Creare un nuovo oggetto.
- D Creare un nuovo script.

**4** È possibile fare in modo che due variabili distinte riferiscano lo stesso oggetto?

- A Sì, sempre.
- B No, mai.
- C Sì, ma solo se i riferimenti sono entrambi dello stesso tipo dell'oggetto che riferiscono.
- D Sì, ma solo se i riferimenti non sono entrambi dello stesso tipo dell'oggetto che riferiscono.

**5** Nel linguaggio PHP lo spazio di memoria assegnato agli attributi di un oggetto istanziato con l'operatore `new` viene liberato ...

- A ... automaticamente dal supporto a tempo di esecuzione del linguaggio.
- B ... solo al termine dell'esecuzione dello script PHP.
- C ... automaticamente dal distruttore della classe.
- D Nessuna delle risposte precedenti.

**6** Nel linguaggio PHP la variabile predefinita `$this` viene usata per fare riferimento ...

- A ... ai metodi della classe.
- B ... agli attributi della classe.
- C ... alla classe stessa.
- D Nessuna delle risposte precedenti.

**7** Nel linguaggio PHP la parola chiave `self` viene usata per fare riferimento ...

- A ... ai metodi della classe.
- B ... agli attributi della classe.
- C ... alla classe stessa.
- D Nessuna delle risposte precedenti.

**8** Quali delle seguenti è la forma generale per applicare un metodo a un oggetto?

- A `$nomeOggetto->nomeMetodo (...)`
- B `nomeMetodo (...)->$nomeOggetto`
- C `NomeClasse::$nomeMetodo (...)`
- D `nomeMetodo (nomeOggetto)`

**9** Quali delle seguenti affermazioni relative all'ereditarietà in PHP sono vere?

- A Permette di ereditare nella classe derivata solo attributi della classe base.
- B Permette estendere la classe base con attributi e metodi della classe derivata.
- C Permette di ereditare nella classe derivata attributi e metodi della classe base eventualmente estendendoli con nuovi attributi e metodi.
- D Permette di inserire attributi o metodi della classe derivata nella classe base.



**10** Se un metodo di una classe derivata sovrascrive un metodo della superclasse, con quale notazione PHP è possibile far riferimento al metodo della superclasse?

- A `self::metodo`
- B `parent->metodo`
- C `parent::metodo`
- D Non è possibile riferire un metodo sovrascritto.

**11** Quali delle seguenti affermazioni a proposito di un costruttore PHP sono vere?

- A È un metodo che viene eseguito automaticamente all'atto della creazione di un oggetto.
- B È un metodo che deve essere obbligatoriamente definito.
- C È un metodo che ha come nome il nome della classe stessa.
- D È un metodo magico denominato `__construct`.

**12** Quali delle seguenti istruzioni PHP permette di definire un nuovo oggetto `$beta` uguale a un oggetto esistente `$alfa`?

- A `$beta = $alfa;`
- B `$beta === $alfa;`
- C `$beta = clone $alfa;`
- D Nessuna delle risposte precedenti.

**13** Quali delle seguenti affermazioni circa i «metodi magici» sono vere?

- A Sono metodi predefiniti delle classi del linguaggio PHP.
- B Sono metodi che non possono in alcun caso essere sovrascritti.
- C Il loro nome inizia sempre con il prefisso «`__`».
- D Alcuni di essi permettono di gestire i membri «inaccessibili» di una classe.

**14** Quali delle seguenti affermazioni circa una classe astratta PHP sono vere?

- A È una classe in cui alcuni metodi sono dichiarati, ma non implementati.
- B È una classe in cui non tutti gli attributi sono definiti.

- C È una classe da cui non possono essere istanziati oggetti.
- D È una classe che viene utilizzata come classe base.

**15** Il valore degli attributi di una classe dichiarati privati ...

- A ... può essere modificato mediante specifici metodi della classe stessa.
- B ... può essere modificato solo dal costruttore.
- C ... non può essere modificato.
- D ... può essere comunque modificato direttamente con un'espressione del tipo  
`$oggetto->attributo = espressione;`

**16** Quali delle seguenti affermazioni relative a un attributo con livello di accessibilità `protected` sono vere?

- A La sua accessibilità è equivalente al livello `public`.
- B La sua accessibilità diretta è possibile solo a livello di classe e classe derivata.
- C La sua accessibilità è equivalente al livello `private`.
- D La sua accessibilità diretta non mai è possibile.

**17** I membri statici di una classe sono ...

- A ... le costanti.
- B ... gli attributi e i metodi che appartengono non alle singole istanze, ma alla classe.
- C ... membri che possono essere riferiti con una notazione del tipo

`NomeClasse->nomeMembro`

- D Nessuna delle risposte precedenti.

**18** Indicare quali delle seguenti affermazioni sono vere relativamente alle eccezioni nel linguaggio PHP ...

- A ... non sono previste.
- B ... sono oggetti istanza della classe PHP `Exception` o di sue derivate.
- C ... possono essere gestite mediante il costrutto `try/catch`.
- D ... possono essere generate con il costrutto `throw`.

## ESERCIZI

**1** Implementare in linguaggio PHP una classe *Bike* i cui oggetti rappresentano biciclette da noleggiare nell'arco di un giorno. Ogni oggetto *Bike* deve avere almeno le seguenti caratteristiche: matricola (numero intero che identifica una bicicletta), misura (piccola, media, grande), tipo (*mountain-bike*, corsa, passeggio), l'ora di inizio del noleggio (espressa come data/ora). Il costo per ora o frazione delle biciclette è fissato in 5 € per le bici da passeggio, 7 € per le *mountain-bike* e 8 € per quelle da corsa. La classe deve esporre i seguenti metodi:

- costruttore che ha come parametri il tipo e la misura;
- i metodi *get/set* per ogni attributo;
- un metodo *toString* che restituisca una stringa con i valori degli attributi dell'oggetto su cui è invocato;
- un metodo che restituisca l'importo del noleggio e che accetta come parametro l'ora di restituzione della bicicletta e fornisce come risultato l'importo da pagare per il noleggio.

**2** Facendo riferimento alla classe *Bike* realizzata nell'esercizio precedente, implementare in linguaggio PHP una classe *NoleggioBike* i cui oggetti rappresentano dei punti di noleggio delle biciclette, ciascuno dei quali prevede tre distinte rastrelliere ognuna delle quali può contenere più oggetti di tipo *Bike*. La prima rastrelliera è dedicata alle *mountain-bike*, la seconda alle bici da corsa e la terza a quelle da passeggio. La classe deve esporre i seguenti metodi:

- costruttore avente come parametro il numero di biciclette presenti in ciascuna rastrelliera;
- *getBike*: ha come parametro il tipo di bicicletta e la misura volute e restituisce, se disponibile, il primo oggetto di quel tipo lasciando vuota la posizione precedentemente occupata nella rastrelliera;
- *setBike*: ha come parametro un oggetto *Bike* e inserisce una bicicletta nella prima posizione libera della rastrelliera di tipo coerente con la bicicletta;

- *getN*: restituisce il numero di biciclette disponibili di un certo tipo e una certa misura;
- *toString*: restituisce una stringa contenente l'elenco di tutte le biciclette contenute in uno specifico tipo di rastrelliera fornito come parametro.

**3** Un'azienda commercializza alcuni prodotti i cui dati caratteristici sono descritti da:



- codice;
- descrizione;
- aliquota IVA;
- prezzo di vendita.

Implementare in linguaggio PHP una classe per rappresentare gli oggetti di tipo *Prodotto* prevedendo:

- i metodi *get/set* per ogni attributo;
- un costruttore che permetta di istanziare oggetti di tipo *Prodotto* definendone valori specifici per i vari attributi;
- un metodo *toString* che restituisca una stringa contenente una descrizione delle caratteristiche di un prodotto.

Scrivere uno script PHP contenuto in una pagina HTML che consenta di verificare tutte le funzionalità della classe.


**4** Facendo riferimento alla classe *Prodotto* realizzata nell'esercizio precedente, implementare in linguaggio PHP una classe *Fattura* i cui oggetti rappresentano fatture di vendita emesse a clienti. Ogni fattura deve contenere i dati del cliente a cui è intestata (ragione sociale, indirizzo completo, partita IVA), l'elenco dei prodotti ordinati con la relativa quantità venduta. La classe deve esporre i seguenti metodi oltre al costruttore:



- *getProdotto*: ha come parametro la posizione di un prodotto nell'ambito di una fattura e restituisce il prodotto medesimo;
- *setProdotto*: inserisce un nuovo prodotto nella lista dei prodotti in fattura specificandone la quantità fatturata, ha come parametri un oggetto *Prodotto* e un valore intero che ne rappresenta la quantità;
- *eliminaProdotto*: elimina dalla fattura un prodotto di cui venga fornito come parametro il relativo codice;

- *toString*: restituisce una stringa contenente l'elenco di tutti i prodotti fatturati con la relativa quantità;
- *totaleImponibile*: restituisce l'importo totale dato dalla somma della quantità dei singoli prodotti moltiplicata per il relativo prezzo unitario;
- *importIva*: restituisce l'importo totale dato dalla somma degli imponibili dei singoli prodotti moltiplicata per la relativa aliquota IVA;
- *totaleFattura*: restituisce il totale generale della fattura come somma del totale imponibile più il totale IVA.

Scrivere uno script PHP contenuto in una pagina HTML che consenta di verificare tutte le funzionalità della classe.

**5**  In riferimento all'esercizio precedente si crei la classe *FatturaAccompagnatoria* che estenda la classe *Fattura* per riunire in uno stesso oggetto

gli elementi della fattura e le informazioni sul trasporto dei prodotti venduti. Una *FatturaAccompagnatoria* prevede dati aggiuntivi rispetto a una fattura normale:

- il numero dei pacchi contenenti i prodotti fatturati;
- il nome dell'azienda che effettua il trasporto delle merci;
- l'importo dovuto per la consegna delle merci.

Sono quindi necessari i seguenti metodi:

- un opportuno costruttore;
- i metodi *get/set* per ogni attributo;
- il metodo *totaleFattura* che restituisce il totale generale della fattura come somma del totale generale della fattura più le spese di spedizione.

Scrivere uno script PHP contenuto in una pagina HTML che consenta di verificare tutte le funzionalità della classe.

# Accesso a una base di dati in linguaggio PHP

# B3

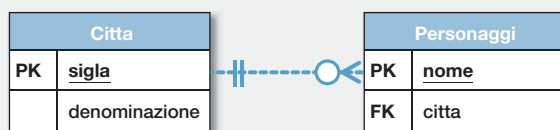
La possibilità di generare pagine web dinamiche non sarebbe una delle tecniche alla base dei siti moderni senza la possibilità di accedere a un DBMS da parte del linguaggio di *scripting server-side*.

## ESEMPIO

Il sito web di Trenitalia consente di consultare gli orari dei treni memorizzati in un database gestito da un DBMS e le prenotazioni effettuate sono a loro volta memorizzate in un database.

Molti siti che utilizzano PHP come tecnologia lato server sono basati su DBMS My-SQL, un prodotto distribuito con licenza open-source da parte di Oracle Corporation.

Gli esempi di questo capitolo<sup>1</sup> sono basati su un semplice database di cui si riporta il diagramma delle tabelle



e il DB-schema in linguaggio SQL:



```
CREATE DATABASE Disneyland;
```

```
USE Disneyland;
```

```
CREATE TABLE Citta (  
    sigla CHAR(2) NOT NULL,  
    denominazione VARCHAR(16) NOT NULL,  
    CONSTRAINT chiave_primaria PRIMARY KEY (sigla)  
);
```

```
INSERT INTO Citta (sigla, denominazione) VALUES  
( 'PA', 'Paperopoli'),  
( 'TO', 'Topolinia');
```

1. Gli esempi sono graficamente essenziali, avendo il solo scopo di dimostrare l'interazione con il DBMS; sono inoltre sempre riferiti al server *localhost* normalmente utilizzato dai programmatori PHP in fase di sviluppo e di test del codice.

## DBMS My-SQL e linguaggio PHP

My-SQL di Oracle Corporation è estremamente diffuso come DBMS per le applicazioni web. Si tratta di un servizio che può essere eseguito sullo stesso computer che ospita il server web, oppure su un diverso computer che per motivi di efficienza dovrebbe trovarsi sulla stessa rete locale.

My-SQL prevede diversi driver (denominati *connector*) per l'esposizione di API (*Application Program Interface*) verso diversi linguaggi di programmazione, tra cui PHP (in alcuni casi si tratta di interfacce verso librerie sviluppate per il linguaggio C e non di driver nativi).

Specifiche estensioni dell'interprete PHP consentono di utilizzare le API delle librerie *mysql* e *mysqli*, oppure della libreria PDO: *PHP Data Object*. La documentazione ufficiale del linguaggio PHP sconsiglia lo sviluppo di nuove applicazioni utilizzando la libreria *legacy mysql*.

```
CREATE TABLE Personaggi (
    nome VARCHAR(32) NOT NULL,
    citta CHAR(2) NOT NULL,
    CONSTRAINT chiave_primaria PRIMARY KEY (nome),
    CONSTRAINT chiave_esterna FOREIGN KEY (citta) REFERENCES
    Citta(sigla)
);
```

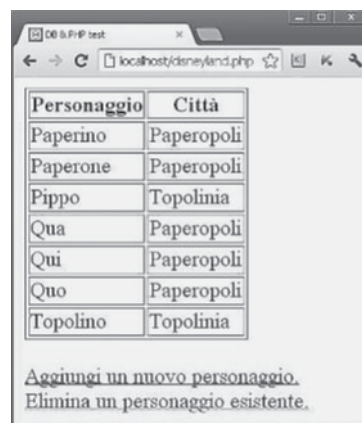
```
INSERT INTO Personaggi (nome, citta) VALUES
('Paperone', 'PA'),
('Paperino', 'PA'),
('Qui', 'PA'),
('Quo', 'PA'),
('Qua', 'PA'),
('Topolino', 'TO'),
('Pippo', 'TO');
```

**OSSERVAZIONE** Il database *Disneyland* viene generato con le città di Paperopoli e Topolinia (nella tabella *Citta*) e un certo numero di personaggi che le abitano (nella tabella *Personaggi*).

Le varie soluzioni proposte nel seguito del capitolo prevedono una stretta integrazione del linguaggio SQL come ospite del linguaggio PHP: l'interazione con il DBMS avviene sempre mediante stringhe costruite dinamicamente che rappresentano comandi SQL che funzioni o metodi PHP inviano al DBMS stesso.

## 1 L'interfaccia del linguaggio PHP con il DBMS My-SQL

Utilizzando il linguaggio PHP è possibile generare una pagina dinamica come la seguente contenente il risultato di una query effettuata su un database di un server My-SQL:



Personaggio	Città
Paperino	Paperopoli
Paperone	Paperopoli
Pippo	Topolinia
Qua	Paperopoli
Qui	Paperopoli
Quo	Paperopoli
Topolino	Topolinia

[Aggiungi un nuovo personaggio.](#)  
[Elimina un personaggio esistente.](#)

**OSSERVAZIONE** Le informazioni riportate nella pagina dinamica sono il risultato della seguente query SQL:

```
SELECT nome, denominazione FROM Personaggi, Citta
WHERE Personaggi.citta = Citta.sigla
ORDER BY nome
```

effettuata sul database *Disneyland*.

## 1.1 L'interfaccia originale *mysql*

Fin dalle prime versioni il linguaggio PHP dispone di una libreria di funzioni denominata «mysql» per interagire con DBMS My-SQL; nella TABELLA 1 sono riportate le funzioni fondamentali della libreria.

TABELLA 1

Funzione	Descrizione
<code>mysql_connect</code>	Connessione a un server DBMS specificando nome (o indirizzo IP) del computer su cui è in esecuzione, <i>username</i> e <i>password</i> . Restituisce un identificativo della connessione, oppure <i>FALSE</i> in caso di errore
<code>mysql_select_db</code>	Selezione di un database su cui operare. Restituisce <i>TRUE</i> in caso di successo, oppure <i>FALSE</i> in caso di errore
<code>mysql_query</code>	Esecuzione di un comando SQL. Per i comandi DML restituisce <i>TRUE</i> in caso di successo, oppure <i>FALSE</i> in caso di errore. Per le query restituisce un riferimento al risultato, oppure <i>FALSE</i> in caso di errore
<code>mysql_affected_rows</code>	Restituisce il numero di righe coinvolte dall'ultimo comando DML eseguito
<code>mysql_num_rows</code>	Restituisce il numero di righe del risultato di una query
<code>mysql_fetch_row</code>	Recupero di una riga del risultato di una query in forma di array indicizzato. Restituisce un array, oppure <i>FALSE</i> se non vi sono ulteriori righe nel risultato
<code>mysql_fetch_assoc</code>	Recupero di una riga del risultato di una query in forma di array associativo avente come chiavi i nomi delle colonne. Restituisce un array, oppure <i>FALSE</i> se non vi sono ulteriori righe nel risultato
<code>mysql_fetch_array</code>	Recupero di una riga del risultato di una query in forma combinata di array indicizzato e associativo. Restituisce un array, oppure <i>FALSE</i> se non vi sono ulteriori righe nel risultato
<code>mysql_close</code>	Chiusura della connessione al server DBMS. Restituisce <i>TRUE</i> in caso di successo, oppure <i>FALSE</i> in caso di errore

Il codice dello script «disneyland.php» che genera la pagina dinamica con la tabella che elenca i personaggi memorizzati nel database *Disneyland* e la relativa città è il seguente (il server DBMS My-SQL è in questo caso eseguito sullo stesso computer del server web e l'utente «root» è privo di password<sup>2</sup>):

## Array super-globale `$_SERVER`

Gli esempi di questo capitolo sono tutti riferiti al computer locale identificato dal nome «localhost»: anche se il server web e il server My-SQL sono eseguiti dallo stesso computer, gli script PHP non possono funzionare in una configurazione client/server in cui il computer client e il computer server sono distinti. Per ovviare a questa difficoltà il linguaggio PHP rende disponibile l'array super-globale `$_SERVER` i cui elementi contengono molte informazioni relative alla connessione di rete gestita dal server HTTP, in particolare gli elementi con chiave «SERVER\_NAME» e «SERVER\_ADDR» ne restituiscono rispettivamente il nome e l'indirizzo IP. Per rendere completamente funzionanti gli script proposti si può sostituire la stringa «localhost» nel codice (sia PHP sia HTML) con il nome di dominio del computer server (ad esempio [www.server.net](http://www.server.net)), oppure utilizzare l'espressione `$_SERVER[SERVER_NAME]` o in alternativa `$_SERVER[SERVER_ADDR]` per far generare dinamicamente dal codice PHP il nome di dominio o l'indirizzo IP del server al posto di impiegare la stringa «localhost».

2. Questa configurazione è assolutamente sconsigliata sotto il profilo della sicurezza, ma è accettabile per un server DBMS utilizzato esclusivamente per sviluppare e testare il codice PHP; in ogni caso i parametri di accesso al DBMS presenti nel codice PHP non sono visibili all'utente perché non sono presenti nel codice HTML inviato al browser.



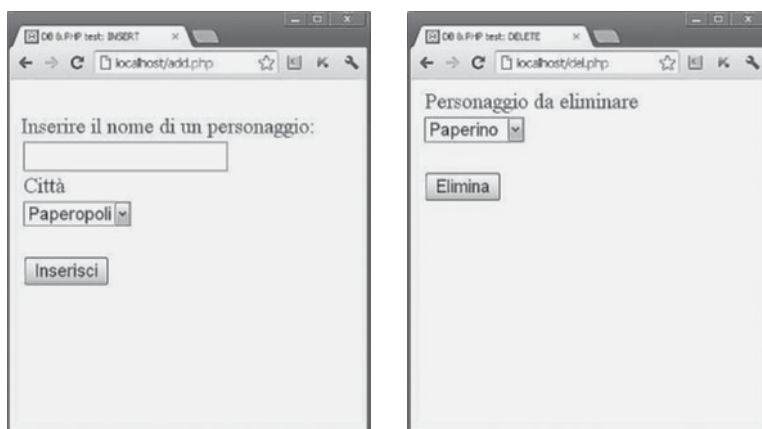
```
<html>
<head>
  <title>DB & PHP test</title>
</head>
<body>
<?php
  $connection = mysql_connect("localhost","root","");
  mysql_select_db("Disneyland",$connection);
  $query = "SELECT nome,denominazione FROM Personaggi,Citta
           WHERE Personaggi.citta = Citta.sigla ORDER BY
           nome";
  $result = mysql_query($query,$connection);
  if (mysql_num_rows($result) != 0)
  {
    echo "<table border>";
    echo "<tr>";
    echo "<th>Personaggio</th>";
    echo "<th>Città</th>";
    echo "</tr>";
    while ($row = mysql_fetch_array($result))
    {
      echo "<tr>";
      echo "<td>$row[nome]</td>";
      //echo "<td>$row[0]</td>";
      echo "<td>$row[denominazione]</td>";
      //echo "<td>$row[1]</td>";
      echo "</tr>";
    }
    echo "</table>";
  }
  else
    echo "Nessun personaggio è presente nel
        database.";
  mysql_close($connection);
?>
<br>
<a href="http://localhost/add.php">
  Aggiungi un nuovo personaggio.
</a>
<br>
<a href="http://localhost/del.php">
  Elimina un personaggio esistente.
</a>
</body>
</html>
```

**OSSERVAZIONE** La tabella viene generata dinamicamente dallo script PHP producendo una riga della tabella per ogni riga del risultato me-

diante un ciclo che recupera le singole righe del risultato della query. L'array risultante ha un elemento per ogni colonna del risultato della query: le istruzioni commentate interne al ciclo esemplificano l'accesso indicizzato in sostituzione di quello per chiave associativa.

In caso di errori nell'interazione con il DBMS è prassi consolidata dei programmatori PHP visualizzare la stringa che contiene il comando SQL fornito come argomento alla funzione `mysql_query` mediante un'istruzione `echo` o `print`<sup>3</sup>.

I due collegamenti ipertestuali inseriti nella pagina di visualizzazione dei personaggi invocano una pagina dinamica generata da uno script PHP contenente una *form* rispettivamente per l'aggiunta di un nuovo personaggio («add.php»), o per l'eliminazione di un personaggio esistente («del.php»):



Il codice della pagina «add.php» è il seguente:

```
<html>
<head>
  <title>DB & PHP test: INSERT</title>
</head>
<body>
  <form action="insert.php" method="GET"><br>
    Inserire il nome di un personaggio:
    <input type="text" name="personaggio"> <br>
    Città<br>
    <select name="citta">
      <option value="PA">Paperopoli</option>
      <option value="TO">Topolinia</option>
    </select><br><br>
    <input type="submit" value="Inserisci">
  </form>
</body>
</html>
```



3. Per semplicità l'invocazione delle funzioni è effettuata negli esempi iniziali del capitolo senza verificare eventuali condizioni di errore. Nel paragrafo 1.4 sono indicate le tecniche per il controllo degli errori.



**OSSERVAZIONE** Il codice della pagina «add.php» è esclusivamente HTML; l'elenco che consente di selezionare la città del personaggio è infatti istanziato staticamente con le sole città Paperopoli e Topolinia presenti nella tabella *Citta* del database *Disneyland*. Questa scelta è coerente con il fatto che non esiste una modalità accessibile all'utente per modificare il contenuto di questa tabella.

I dati della *form* della pagina «add.php» sono inviati utilizzando il metodo GET alla pagina «insert.php» costituita da uno script PHP che effettua l'inserimento nel database del nuovo personaggio utilizzando il comando SQL **INSERT**:



```
<html>
<head>
  <title>DB & PHP test: INSERT</title>
</head>
<body>
<?php
  $personaggio = $_GET["personaggio"];
  $citta = $_GET["citta"];
  $connection = mysql_connect("localhost","root","");
  mysql_select_db("Disneyland",$connection);
  $query = "SELECT * FROM Personaggi WHERE nome =
            '$personaggio'";
  $result = mysql_query($query,$connection);
  if (mysql_num_rows($result) != 0)
    echo "Il personaggio $personaggio &egrave; gi&agrave;
        presente nel database!";
  else
  {
    $query = "INSERT INTO Personaggi VALUES
              ('$personaggio','$citta')";
    $result = mysql_query($query,$connection);
    echo "Il personaggio $personaggio &egrave; stato
        aggiunto al database!";
  }
  mysql_close($connection);
?><br><br>
  <a href="http://localhost/disneyland.php">
    Visualizza elenco personaggi.
  </a>
</body>
</html>
```

**OSSERVAZIONE** Per prevenire errori dovuti alla duplicazione della chiave primaria della tabella *Personaggi*, prima di effettuare l'operazione di inserimento viene verificato se il nome del personaggio è già presente

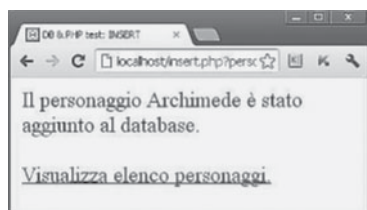
nella tabella. Non è invece possibile in questo caso avere errori di integrità referenziale in quanto la sigla della città viene scelta dall'utente tra le due possibilità («PA» e «TO») presenti nella tabella *Citta* al momento della generazione del database.

#### ESEMPIO

L'aggiunta del personaggio «Archimede» con selezione della città «Paperopoli» visualizza la seguente pagina dinamica invocata mediante l'URL:

`http://localhost/insert.php?personaggio=Archimede&citta=PA`

che comprende la stringa di interrogazione con i valori associati ai nomi *personaggio* e *citta*:



La selezione del collegamento ipertestuale invoca lo script PHP «disneyland.php» che visualizza l'elenco dei personaggi comprendente il nuovo personaggio aggiunto al database:

Personaggio	Città
Archimede	Paperopoli
Paperino	Paperopoli
Paperone	Paperopoli
Pippo	Topolinia
Qua	Paperopoli
Qui	Paperopoli
Quo	Paperopoli
Topolino	Topolinia

[Aggiungi un nuovo personaggio.](#)  
[Elimina un personaggio esistente.](#)

Il codice della pagina «del.php», che contiene la *form* per l'eliminazione di un personaggio dal database, genera dinamicamente l'elenco dei personaggi tra cui l'utente seleziona quello dal eliminare:

```
<html>
  <head>
    <title>DB & PHP test: DELETE</title>
  </head>
  <body>
```



```

<?php
    $connection = mysql_connect("localhost","root","");
    mysql_select_db("Disneyland",$connection);
    $query = "SELECT nome FROM Personaggi ORDER BY nome";
    $result = mysql_query($query,$connection);
    if (mysql_num_rows($result) != 0)
    {
?>
<form action="delete.php" method="GET" ><br>
    Personaggio da eliminare<br>
    <select name="personaggio">
        <?php
            while ($row = mysql_fetch_array($result))
                echo "<option value=\"\$row[0]\">$row[0]</option>";
                // echo "<option value=\"\$row[nome]\">$row[nome]
                // </option>";

        ?>
    </select><br><br>
    <input type="submit" value="Elimina">
</form>
<?php
    }
    else
        echo "Nessun personaggio &egrave; presente nel
        database.";
    mysql_close($connection);
?>
</body>
</html>

```

**OSSERVAZIONE** La riga di codice commentata rappresenta l'accesso alternativo per chiave associativa, anziché per indice, all'array che contiene una riga del risultato della query (`row[nome]` invece di `row[0]`).

La *form* generata dallo script, utilizzando il metodo GET, invia alla pagina «delete.php» il personaggio selezionato dall'utente; lo script PHP della pagina ne effettua l'eliminazione dal database utilizzando il comando SQL **DELETE**:



```

<html>
<head>
    <title>DB & PHP test: DELETE</title>
</head>
<body>
<?php
    $personaggio = $_GET["personaggio"];
    $connection = mysql_connect("localhost","root","");
    mysql_select_db("Disneyland",$connection);

```

```

$query = "DELETE FROM Personaggi WHERE nome = '$personaggio'";
$result = mysql_query($query,$connection);
echo "Il personaggio $personaggio &egrave; stato
      eliminato dal database.";
mysql_close($connection);
?><br><br>
<a href="http://localhost/disneyland.php">
  Visualizza elenco personaggi.
</a>
</body>
</html>

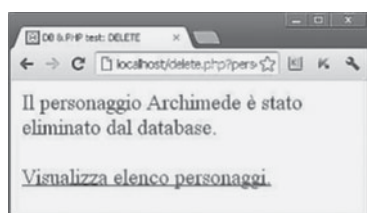
```

#### ESEMPIO

L'eliminazione del personaggio «Archimede» visualizza la seguente pagina dinamica invocata mediante l'URL:

`http://localhost/delete.php?personaggio=Archimede`

che comprende la stringa di interrogazione con il valore associato al nome *personaggio*:



La selezione del collegamento ipertestuale invoca lo script PHP «disneyland.php» che visualizza l'elenco aggiornato dei personaggi.

## 1.2 L'interfaccia avanzata *mysqli*

Le versioni più recenti del linguaggio PHP dispongono di una nuova libreria denominata «mysqli» (*mysql improved*) per l'interfacciamento con DBMS My-SQL. Caratteristica di questa libreria è il fatto di esporre sia un approccio procedurale non diverso da quello originale, sia un approccio orientato agli oggetti: i due approcci possono inoltre essere utilizzati nel contesto dello stesso script.

L'approccio procedurale è basato sulle seguenti funzioni fondamentali indicate nella **TABELLA 2** ed è del tutto analogo a quello della libreria originale *mysql*.

**OSSERVAZIONE** La funzione `mysqli_connect` comprende la selezione del database su cui operare; per verificare un eventuale errore di connessione al DBMS è necessario invocare la funzione `mysqli_connect_errno`.

TABELLA 2

Funzione	Descrizione
<code>mysqli_connect</code>	Connessione a un server DBMS specificando nome (o indirizzo IP) del computer su cui è in esecuzione, <i>username</i> , <i>password</i> e <i>database</i> . Restituisce un identificativo della connessione
<code>mysqli_connect_errno</code>	Restituisce l'eventuale codice numerico di errore della connessione, il valore 0 se la connessione è stata stabilita correttamente
<code>mysqli_query</code>	Esecuzione di un comando SQL. Per i comandi DML restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore. Per le query restituisce un riferimento al risultato, oppure <code>FALSE</code> in caso di errore
<code>mysqli_affected_rows</code>	Restituisce il numero di righe coinvolte dall'ultimo comando DML eseguito, il valore -1 in caso di errore
<code>mysqli_num_rows</code>	Restituisce il numero di righe del risultato di una query
<code>mysqli_fetch_row</code>	Recupero di una riga del risultato di una query in forma di array indicizzato. Restituisce un array, oppure <code>NULL</code> se non vi sono ulteriori righe nel risultato
<code>mysqli_fetch_assoc</code>	Recupero di una riga del risultato di una query in forma di array associativo avente come chiavi i nomi delle colonne. Restituisce un array, oppure <code>NULL</code> se non vi sono ulteriori righe nel risultato
<code>mysqli_fetch_array</code>	Recupero di una riga del risultato di una query in forma combinata di array indicizzato e associativo. Restituisce un array, oppure <code>NULL</code> se non vi sono ulteriori righe nel risultato
<code>mysqli_free_result</code>	Rilascio della memoria del risultato di una query. Non restituisce nulla
<code>mysqli_close</code>	Chiusura della connessione al server DBMS. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore

Inoltre la libreria *mysqli* dispone di funzioni per la gestione delle transazioni (TABELLA 3).

TABELLA 3

Funzione	Descrizione
<code>mysqli_autocommit</code>	Fornendo come argomento <code>TRUE</code> o <code>FALSE</code> abilita o disabilita la modalità <b>AUTOCOMMIT</b> del DBMS My-SQL. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore
<code>mysqli_commit</code>	Esegue il <b>COMMIT</b> dei comandi eseguiti in precedenza. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore
<code>mysqli_rollback</code>	Esegue il <b>ROLLBACK</b> dei comandi eseguiti in precedenza. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore

4. Nella trattazione sono elencati solo le proprietà e i metodi fondamentali delle classi; per un elenco esaustivo si rimanda alla documentazione ufficiale del linguaggio PHP disponibile sul sito <http://php.net>.

L'uso delle funzioni della libreria *mysqli* è del tutto analogo a quello delle funzioni della libreria *mysql*. Nel seguito prendiamo in esame esclusivamente l'approccio orientato agli oggetti consentito dalla libreria *mysqli* che è basata su tre classi di oggetti fondamentali<sup>4</sup>:

- *mysqli*, che rappresenta la connessione con il DBMS My-SQL;
- *mysqli\_result*, che rappresenta il risultato di una query;
- *mysqli\_stmt*, che rappresenta un comando SQL «parametrico».

Un oggetto di classe *mysqli* espone le proprietà indicate nella TABELLA 4 e i metodi descritti nella TABELLA 5.

TABELLA 4

Proprietà	Descrizione
<code>\$affected_rows</code>	Contiene il numero di righe coinvolte dall'ultimo comando DML eseguito; il valore è -1 in caso di errore
<code>\$connect_errno</code>	Contiene l'eventuale codice numerico di errore della connessione; il valore è 0 se la connessione è stata stabilita correttamente
<code>\$errno</code>	Contiene l'eventuale codice numerico di errore dell'ultimo comando eseguito; il valore è 0 se il comando è stato eseguito correttamente

TABELLA 5

Metodi	Descrizione
<code>__construct</code>	È il costruttore della classe. Effettua la connessione a un server DBMS di cui viene fornito come argomento il nome (o l'indirizzo IP) del computer su cui è in esecuzione, l' <i>username</i> e la <i>password</i> dell'utente e il <i>database</i> su cui si intende operare
<code>autocommit</code>	Fornendo come argomento <code>TRUE</code> o <code>FALSE</code> abilita o disabilita la modalità <b>AUTOCOMMIT</b> del DBMS My-SQL. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore
<code>close</code>	Chiude la connessione al server DBMS: restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore
<code>commit</code>	Esegue il <b>COMMIT</b> dei comandi eseguiti in precedenza. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore
<code>prepare</code>	Predispose un comando SQL parametrizzato per l'esecuzione (i parametri sono indicati nel comando SQL per mezzo di simboli «?»). Restituisce un oggetto di classe <i>mysqli_stmt</i> , oppure <code>FALSE</code> in caso di errore
<code>query</code>	Esegue un comando SQL. Per i comandi DML restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore. Per le query restituisce un riferimento al risultato, oppure <code>FALSE</code> in caso di errore
<code>rollback</code>	Esegue il <b>ROLLBACK</b> dei comandi eseguiti in precedenza. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore

Un oggetto di classe *mysqli\_result* espone le proprietà indicate nella TABELLA 6 e i metodi descritti nella TABELLA 7.

TABELLA 6

Proprietà	Descrizione
<code>\$num_rows</code>	Contiene il numero di righe del risultato della query
<code>\$field_count</code>	Contiene il numero di colonne del risultato della query

TABELLA 7

Metodi	Descrizione
<code>fetch_all</code>	Restituisce un array in cui ogni elemento è un array che rappresenta una singola riga del risultato di una query
<code>fetch_array</code>	Restituisce una riga del risultato di una query in forma combinata di array indicizzato e associativo, oppure <code>NULL</code> se non vi sono ulteriori righe nel risultato
<code>fetch_assoc</code>	Restituisce una riga del risultato di una query in forma di array associativo avente come chiavi i nomi delle colonne, oppure <code>NULL</code> se non vi sono ulteriori righe nel risultato
<code>fetch_row</code>	Restituisce una riga del risultato di una query in forma di array indicizzato, oppure <code>NULL</code> se non vi sono ulteriori righe nel risultato
<code>free</code>	Rilascia la memoria del risultato di una query

## ESEMPIO

Gli script PHP presentati nel precedente paragrafo sono qui di seguito riproposti utilizzando la libreria *mysqli* con interfaccia orientata agli oggetti.



- «disneyland.php» visualizza l'elenco dei personaggi con la relativa città:

```
<html>
<head>
  <title>DB & PHP test</title>
</head>
<body>
<?php
  $connection = new mysqli("localhost","root","","Disneyland");
  $query = "SELECT nome,denominazione FROM Personaggi,Citta
           WHERE Personaggi.citta = Citta.sigla ORDER BY nome";
  $result = $connection->query($query);
  if ($result->num_rows != 0)
  {
    echo "<table border>";
    echo "<tr>";
    echo "<th>Personaggio</th>";
    echo "<th>Città</th>";
    echo "</tr>";
    while ($row = $result->fetch_array())
    {
      echo "<tr>";
      echo "<td>$row[nome]</td>";
      //echo "<td>$row[0]</td>";
      echo "<td>$row[denominazione]</td>";
      //echo "<td>$row[1]</td>";
      echo "</tr>";
    }
    echo "</table>";
  }
  $result->free();
  $connection->close();
?><br>
<a href="http://localhost/add.php">
  Aggiungi un nuovo personaggio.
```



```

    </a><br>
    <a href="http://localhost/del.php">
        Elimina un personaggio esistente.
    </a>
</body>
</html>

```

- «add.php» e «insert.php» aggiungono al database un nuovo personaggio inserito dall'utente che ne seleziona la città di appartenenza:

```

<html>
<head>
    <title>DB & PHP test: INSERT</title>
</head>
<body>
    <form action="insert.php" method="GET"><br>
        Inserire il nome di un personaggio:
        <input type="text" name="personaggio"> <br>
        Città&grave;:<br>
        <select name="citta">
            <option value="PA">Paperopoli</option>
            <option value="TO">Topolinia</option>
        </select><br><br>
        <input type="submit" value="Inserisci">
    </form>
</body>
</html>

```

```

<html>
<head>
    <title>DB & PHP test: INSERT</title>
</head>
<body>
    <?php
        $personaggio = $_GET["personaggio"];
        $citta = $_GET["citta"];
        $connection = new mysqli("localhost","root","", "Disneyland");
        $query = "SELECT * FROM Personaggi WHERE nome = '$personaggio'";
        $result = $connection->query($query);
        if ($result->num_rows != 0)
            echo "Il personaggio $personaggio &grave; gi&grave; presente nel database.";
        else
        {
            $query = "INSERT INTO Personaggi VALUES ('$personaggio','$citta')";
            $connection->query($query);
            echo "Il personaggio $personaggio &grave; stato aggiunto al database.";
        }
        $result->free();
        $connection->close();
    ?><br><br>
    <a href="http://localhost/disneyland.php">
        Visualizza elenco personaggi.
    </a>
</body>
</html>

```





- «del.php» e «delete.php» eliminano dal database un personaggio selezionato dall'utente:

```
<html>
<head>
  <title>DB & PHP test: DELETE</title>
</head>
<body>
  <?php
    $connection = new mysqli("localhost","root","","Disneyland");
    $query = "SELECT nome FROM Personaggi ORDER BY nome";
    $result = $connection->query($query);
    if ($result->num_rows != 0) {
  ?>
  <form action="delete.php" method="GET" ><br>
    Personaggio da eliminare<br>
    <select name="personaggio">
      <?php
        while ($row = $result->fetch_array())
          //echo "<option value=\"$row[0]\">$row[0]</option>";
          echo "<option value=\"$row[nome]\">$row[nome]</option>";
      ?>
    </select><br><br>
    <input type="submit" value="Elimina">
  </form>
  <?php
    }
    else
      echo "Nessun personaggio &egrave; presente nel database.";
    $result->free();
    $connection->close();
  ?>
</body>
</html>

<html>
<head>
  <title>DB & PHP test: DELETE</title>
</head>
<body>
  <?php
    $personaggio = $_GET["personaggio"];
    $connection = new mysqli("localhost","root","","Disneyland");
    $query = "DELETE FROM Personaggi WHERE nome = '$personaggio'";
    $result = $connection->query($query);
    $connection->close();
    echo "Il personaggio $personaggio &egrave; stato eliminato dal database.";
  ?><br><br>
  <a href="http://localhost/disneyland.php">
    Visualizza elenco personaggi.
  </a>
</body>
</html>
```

Gli oggetti di classe *mysqli\_stmt* sono istanziati dal metodo `prepare` di un oggetto di classe *mysqli* e consentono di definire comandi SQL parametrici<sup>5</sup> che possono essere eseguiti più volte, fornendo valori distinti per i parametri: questa modalità è molto efficiente nel caso che sia necessario ripetere comandi strutturalmente simili.

5. Sono trattati solo i più comuni comandi SQL DML, ma un oggetto di classe *mysqli\_stmt* permette di gestire query SQL parametriche; in questo caso i parametri possono essere anche relativi ai risultati della query.

**ESEMPIO** Il ricorso a comandi parametrici è molto utile se si devono generare le righe di una tabella di un database a partire dai valori contenuti in un file inviato dal browser al server.

Un oggetto di classe *mysqli\_stmt* espone le proprietà illustrate nella TABELLA 8 e i metodi descritti nella TABELLA 9.

TABELLA 8

Proprietà	Descrizione
<code>\$affected_rows</code>	Contiene il numero di righe coinvolte dall'ultimo comando eseguito; il valore è -1 in caso di errore
<code>\$errno</code>	Contiene l'eventuale codice numerico di errore dell'ultimo comando eseguito; il valore è 0 se il comando è stato eseguito correttamente
<code>\$param_count</code>	Contiene il numero di parametri del comando parametrico

TABELLA 9

Proprietà	Descrizione
<code>bind_param</code>	Istanza il comando parametrico sostituendo valori effettivi ai parametri, per ogni valore è necessario definire il tipo mediante un carattere («i» = <i>integer</i> , «d» = <i>double</i> , «s» = <i>string</i> ). Restituisce TRUE in caso di successo, oppure FALSE in caso di errore
<code>close</code>	Conclude l'uso del comando parametrico. Restituisce TRUE in caso di successo, oppure FALSE in caso di errore
<code>execute</code>	Esegue il comando parametrico con i parametri istanziati. Restituisce TRUE in caso di successo, oppure FALSE in caso di errore

**ESEMPIO** Il codice che segue inserisce nella tabella *Personaggi* i personaggi contenuti in un array indicizzato i cui elementi sono array associativi di due elementi con chiave «personaggio» e «sigla\_citta»:

```
<?php
...
$personaggi = array(
    array('personaggio' => "Gambadilegno", 'sigla_citta' => "TO"),
    array('personaggio' => "Banda Bassotti", 'sigla_citta' => "PA"),
    array('personaggio' => "Minni", 'sigla_citta' => "TO"),
    array('personaggio' => "Paperina", 'sigla_citta' => "PA")
);
```

```

...
...
...
$connection = new mysqli("localhost","root","", "Disneyland");
$command = $connection->prepare("INSERT INTO Personaggi VALUES (?,?)");
foreach ($personaggi as $personaggio)
{
    $command->bind_param("ss", $personaggio['personaggio'],
                        $personaggio['sigla_citta']);
    $command->execute();
}
$command->close();
$connection->close();
...
?>

```

**OSSERVAZIONE** La stringa che definisce il comando parametrico argomento del metodo `prepare` ha due simboli «?» nella posizione in cui devono essere inseriti i valori dei parametri. Il primo argomento del metodo `bind_param` definisce due parametri di tipo stringa («ss») i cui valori sono i successivi parametri del metodo stesso. L'invocazione del metodo `execute` esegue il comando SQL parametrizzato, ogni volta con valori diversi.

### 1.3 Inserimento nella tabella di un database dei dati contenuti in un file inviato dal browser al server

Il tipo «file» per il tag `input` di una *form* HTML consente all'utente di selezionare un file da inviare al server utilizzando il metodo «POST»: se la pagina di destinazione dell'azione della *form* è uno script PHP, questo riceve le informazioni relative al file nell'array superglobale `$_FILES`, con chiave uguale al valore dell'attributo *name*; le informazioni ricevute sono elementi di un array associativo (TABELLA 10).

TABELLA 10

Elemento	Descrizione
<code>\$name</code>	Contiene il nome del file sul computer client
<code>\$type</code>	Contiene il tipo MIME del file se specificato nella <i>form</i> HTML
<code>\$size</code>	Contiene la dimensione in byte del file
<code>\$tmp_name</code>	Contiene il nome, completo del percorso, del file sul computer server <sup>6</sup>
<code>\$error</code>	Contiene il codice di errore relativo al caricamento del file: il valore <code>UPLOAD_ERR_OK (0)</code> indica l'assenza di errori

6. Il server HTTP salva il file in una directory temporanea dove può essere elaborato da uno script PHP e da dove dovrebbe essere eliminato al termine dell'elaborazione; se il file deve essere salvato permanentemente in una posizione del *file-system* del server, deve esservi copiato dallo script PHP.

La seguente *form* HTML permette di inviare un file in formato testuale contenente personaggi da aggiungere al database *Disneyland*:



```
<html>
<head>
  <title>DB & PHP test: UPLOAD</title>
</head>
<body>
  <form enctype="multipart/form-data"
    action="insert_from_file.php"
    method="POST">
    Selezionare il file dei personaggi da inviare:<br>
    <input type="file" name="personaggi"><br><br>
    <input type="submit" value="Inoltra">
  </form>
</body>
</html>
```

**OSSERVAZIONE** L'attributo *enctype* del tag `form` può assumere i seguenti valori: «application/x-www-form-urlencoded» (i dati sono codificati per l'invio come nelle stringhe di interrogazione dell'URL), «multipart/form-data» (i dati non sono codificati per l'invio), «text/plain» (i dati non sono codificati per l'invio, ma gli spazi sono convertiti in simboli «+»). Se una *form* effettua l'upload di un file deve necessariamente definire come valore dell'attributo *enctype* «multipart/form-data».

Lo script PHP che segue («insert\_form\_file.php»), se invocato dall'inoltro di un file in formato CSV contenente per ogni riga di testo il nome di un personaggio seguito – dopo il simbolo separatore «,» – dalla sigla della città in cui vive, carica nella tabella *Personaggi* del database *Disneyland* i personaggi che non violino l'unicità della chiave primaria della tabella stessa e l'integrità referenziale relativa alla tabella *Citta*:



```
<html>
<head>
  <title>DB & PHP test: UPLOAD</title>
</head>
<body>
  <?php
    if ($_FILES["personaggi"]["error"] == UPLOAD_ERR_OK)
    {
      $connection = new mysqli("localhost","root","",
                             "Disneyland");
      $command = $connection->prepare("INSERT INTO
                                     Personaggi VALUES
                                     (?,?)");
      $personaggi = file($_FILES["personaggi"]["tmp_name"],
                        FILE_IGNORE_NEW_LINES |
                        FILE_SKIP_EMPTY_LINES);
    }
  >
```

## Multipurpose Internet Mail Extension (MIME)

Lo standard MIME nasce per definire le estensioni di contenuto dei messaggi di posta elettronica e i formati degli allegati. Con il tempo la modalità di definizione dei formati dello standard MIME è divenuta uno standard per la definizione del formato dei dati scambiati sulla rete Internet indipendentemente dal protocollo utilizzato. Questa evoluzione dello standard prende il nome di *Internet Media Type* e prevede una classificazione dei formati in categorie:

- *application*: formati per specifiche applicazioni software (per esempio PDF);
- *audio*: formati per audio (per esempio MP3);
- *image*: formati per immagini (per esempio JPEG);
- *message*: protocolli per lo scambio di messaggi (per esempio HTTP);
- *model*: formati per modelli tridimensionali (per esempio VRML);
- *multipart*: formati per oggetti internamente costituiti da più parti;
- *text*: formati testuali (per esempio CSV e HTML);
- *video*: formati per video (per esempio MPEG).

```

foreach ($personaggi as $linea)
{
    $dati = explode(",", $linea);
    $personaggio = trim($dati[0]);
    $citta = trim($dati[1]);
    $command->bind_param("ss", $personaggio, $citta);
    if ($command->execute())
        echo "Il personaggio $personaggio &grave;
            stato aggiunto al database.<br>";
    else
        echo "Errore: il personaggio $personaggio
            NON &grave; stato aggiunto
            al database.<br>";
}
unlink($_FILES["personaggi"]["tmp_name"]);
$command->close();
$connection->close();
}
else
    echo "Errore di caricamento del file.";
?><br>
<a href="http://localhost/disneyland.php">
    Visualizza elenco personaggi.
</a>
</body>
</html>

```

7. In caso di file di grandi dimensioni si dovrebbe evitare il ricorso a questa funzione, sostituendola con un ciclo che legge una singola riga del file per volta.

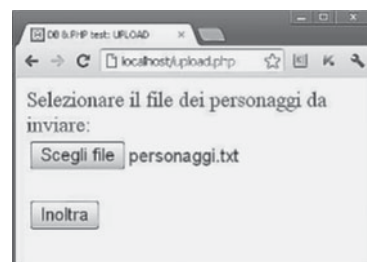
**OSSERVAZIONE** La funzione *file* carica l'intero contenuto di un file testuale in un array indicizzato<sup>7</sup>: ogni elemento dell'array contiene una riga del file. Le opzioni specificate come secondo parametro indicano che le righe eventualmente vuote devono essere ignorate e che i caratteri CR e/o LF devono essere eliminati dall'ultimo elemento di ogni riga.

La funzione *unlink* elimina il file specificato come argomento.

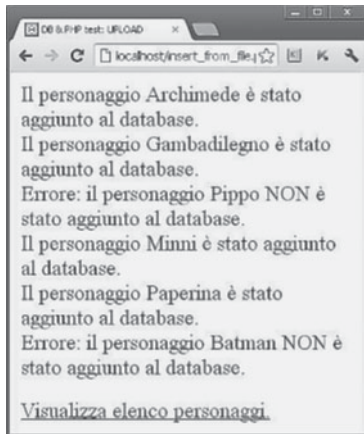
## ESEMPIO

Selezionando il file «personaggi.txt» con il seguente contenuto:

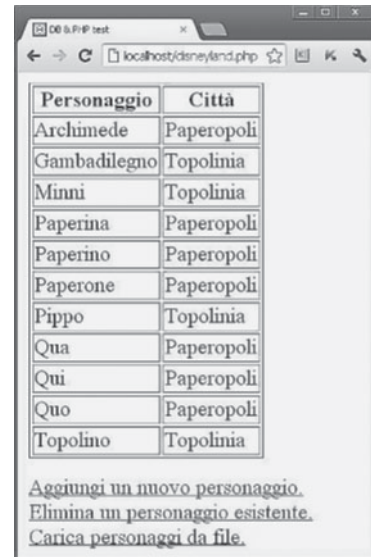
Archimede, PA  
 Gambadilegno, TO  
 Pippo, TO  
 Minni, TO  
 Paperina, PA  
 Batman, GC



si ottiene la seguente pagina dinamica come risultato:

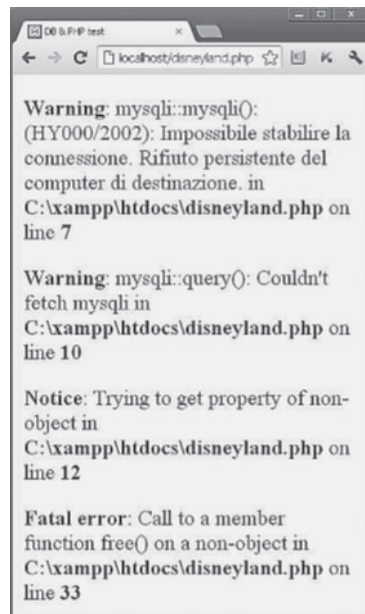


La visualizzazione dell'elenco dei personaggi conferma l'inserimento nella tabella del database:



## 1.4 Controllo degli errori e degli input

In caso di non funzionamento del DBMS My-SQL lo script PHP di visualizzazione dei personaggi del database *Disneyland* produce un risultato simile al seguente, che risulta incomprensibile all'utente:



L'operatore di controllo degli errori del linguaggio PHP («@») impedisce la generazione dei messaggi di errore nell'output inviato al browser in caso di errore nella valutazione di un'espressione o nell'invocazione di una funzione.

**OSSERVAZIONE** Normalmente il fallimento della valutazione di un'espressione o dell'invocazione di una funzione comporta una situazione di errore non recuperabile: in questo caso è necessario verificare la potenziale situazione di errore e definire un comportamento alternativo che spesso consiste nella terminazione dell'esecuzione dello script PHP e nella visualizzazione di un messaggio di errore (a questo scopo si utilizzano le istruzioni **exit** o **die**).

Una versione migliore sotto questo aspetto dello script PHP di visualizzazione dei personaggi del database *Disneyland* è la seguente:



```
<html>
<head>
  <title>DB & PHP test</title>
</head>
<body>
  <?php
    $connection = @ new mysqli("localhost","root","", "Disneyland");
    if ($connection->connect_error)
      die ("Errore di connessione con il DBMS.");
    $query = "SELECT nome,denominazione FROM Personaggi,Citta WHERE
              Personaggi.citta = Citta.sigla ORDER BY nome";
    $result = @ $connection->query($query);
    if ($connection->errno)
    {
      @ $connection->close();
      die ("Errore nell'esecuzione della query");
    }
    if (@ $result->num_rows != 0)
    {
      echo "<table border>";
      echo "<tr>";
      echo "<th>Personaggio</th>";
      echo "<th>Città</th>";
      echo "</tr>";
      while ($row = @ $result->fetch_array())
      {
        echo "<tr>";
        echo "<td>$row[0]</td>";
        echo "<td>$row[1]</td>";
        echo "</tr>";
      }
      echo "</table>";
    }
    @ $result->free();
    @ $connection->close();
  ?><br>
```



```

<a href="http://localhost/add.php">
    Aggiungi un nuovo personaggio.
</a><br>
<a href="http://localhost/del.php">
    Elimina un personaggio esistente.
</a><br>
<a href="http://localhost/upload.php">
    Carica personaggi da file.
</a>
</body>
</html>

```

Dato che l'inoltro di dati a uno script PHP avviene anche nel caso di valori numerici mediante stringhe di caratteri, sia utilizzando il metodo «GET» sia il metodo «POST», uno stretto controllo sui dati ricevuti è una pratica che previene numerosi errori potenziali e alcuni attacchi malevoli. Il linguaggio rende disponibile in particolare la funzione `filter_input` per validare/«filtrare» i dati forniti come input mediante gli array superglobali `$_GET` o `$_POST`.

La funzione `filter_input` – che restituisce il valore corretto dell'input, oppure `FALSE` o `NULL` in caso di errore – ha i seguenti parametri:

- tipo di input: `INPUT_GET` per l'array `$_GET`, o `INPUT_POST` per l'array `$_POST`;
- nome della variabile: chiave associativa dell'elemento dell'array superglobale;
- filtro: uno tra quelli riportati nelle seguenti TABELLE 11 e 12.

TABELLA 11

Filtri di validazione	
<code>FILTER_VALIDATE_EMAIL</code>	Verifica di un indirizzo e-mail
<code>FILTER_VALIDATE_FLOAT</code>	Verifica di un valore numerico <i>floating-point</i>
<code>FILTER_VALIDATE_INT</code>	Verifica di un valore numerico intero
<code>FILTER_VALIDATE_IP</code>	Verifica di un indirizzo IP
<code>FILTER_VALIDATE_URL</code>	Verifica di una stringa URL

TABELLA 12

Filtri di sanitizzazione	
<code>FILTER_SANITIZE_EMAIL</code>	Verifica/correzione di un indirizzo e-mail
<code>FILTER_SANITIZE_NUMBER_FLOAT</code>	Verifica/correzione di un valore numerico <i>floating-point</i>
<code>FILTER_SANITIZE_NUMBER_INT</code>	Verifica/correzione di un valore numerico intero
<code>FILTER_SANITIZE_STRING</code>	Verifica/correzione di una stringa di caratteri (eliminazione di eventuali tag HTML)
<code>FILTER_SANITIZE_URL</code>	Verifica/correzione di una stringa URL

## SQL injection

Con questo nome si identifica una tecnica di attacco malevolo ad un sito web con pagine dinamiche gestite mediante un linguaggio di *scripting* lato server come PHP. Esso viene condotto inserendo codice SQL eventualmente dannoso nei campi di una *form* HTML: nel caso che il testo inserito nei campi venga direttamente utilizzato per comporre una stringa che rappresenta un comando in linguaggio SQL da inviare al DBMS si ottiene l'esecuzione del codice «iniettato». Il controllo delle stringhe fornite come input è essenziale per evitare questo tipo di attacco.



**OSSERVAZIONE** I filtri di validazione non alterano il dato di input: si limitano a segnalare l'eventuale non conformità al formato specificato; i filtri di sanitizzazione modificano il dato di input in coerenza con il formato specificato eliminando, per esempio, caratteri non richiesti.

ESEMPIO

I dati che l'utente inserisce mediante la seguente *form* HTML

```
<html>
<head>
  <title>Dati</title>
</head>
<body>
  <form method="GET" action="dati.php">
    <label for="nominativo">Nominativo</label>
    <input id="nominativo" name="nominativo" type="text" required><br>
    <label for="email">Indirizzo posta elettronica</label>
    <input id="email" name="email" type="text" required><br>
    <label for="web">Sito web</label>
    <input id="web" name="web" type="text" required><br>
    <label for="altezza">Altezza (cm)</label>
    <input id="altezza" name="altezza" type="text" required><br>
    <label for="peso">Peso (Kg)</label>
    <input id="peso" name="peso" type="text" required><br>
    <input type="submit" value="Inoltra dati">
  </form>
</body>
</html>
```



The screenshot shows a web browser window with the title 'Dati'. The address bar shows 'localhost/dati'. The form contains the following fields and values:

Field Label	Value
Nominativo	Pippo
Indirizzo posta elettronica	pippo@disney.com
Sito web	http://www.disney.com
Altezza (cm)	201
Peso (Kg)	99.9

At the bottom of the form is a submit button labeled 'Inoltra dati'.

possono essere validati dal seguente script PHP:

```

<html>
  <head>
    <title>Dati</title>
  </head>
  <body>
    <?php
      $nominativo = filter_input(INPUT_GET, 'nominativo', FILTER_SANITIZE_STRING);
      if ($nominativo)
        echo "Nominativo: $nominativo";
      else
        echo "Nominativo NON corretto";
      echo "<br>";
      $email = filter_input(INPUT_GET, 'email', FILTER_VALIDATE_EMAIL);
      if ($email)
        echo "Indirizzo posta elettronica: $email";
      else
        echo "Indirizzo posta elettronica NON corretto";
      echo "<br>";
      $web = filter_input(INPUT_GET, 'web', FILTER_VALIDATE_URL);
      if ($web)
        echo "Sito web: $web";
      else
        echo "Sito web NON corretto";
      echo "<br>";
      $altezza = filter_input(INPUT_GET, 'altezza', FILTER_VALIDATE_INT);
      if ($altezza)
        echo "Altezza: $altezza cm";
      else
        echo "Altezza NON corretta";
      echo "<br>";
      $peso = filter_input(INPUT_GET, 'peso', FILTER_VALIDATE_FLOAT);
      if ($peso)
        echo "Peso: $peso Kg";
      else
        echo "Peso NON corretto";
    ?>
  </body>
</html>

```

## 2 Gestione degli utenti e delle password con DBMS My-SQL e linguaggio PHP

La gestione degli utenti – con le relative credenziali di accesso, per esempio username e password – è una delle problematiche più comuni nella realizzazione di un sito web. I dati relativi al singolo utente devono essere memorizzati in una specifica tabella di un database ospitato dal DBMS di supporto del sito web stesso. A questo proposito al database *Disneyland* è stata aggiunta una nuova tabella *Utente* creata con il seguente comando SQL:

8. La personalizzazione delle credenziali di accesso al DBMS di supporto al sito web presuppone la gestione da parte del DBA di utenti distinti per il DBMS. Nella presente trattazione non viene presa in considerazione questa possibilità e nei campi *DB\_username* e *DB\_password* della tabella *Utenti* sono sempre memorizzate le credenziali relative all'utente predefinito del DBMS My-SQL.

**USE** Disneyland;

```
CREATE TABLE Utente (  
    username VARCHAR(16) NOT NULL,  
    password VARCHAR(256) NOT NULL,  
    DB_username VARCHAR(16) NOT NULL,  
    DB_password VARCHAR(256) NOT NULL,  
    CONSTRAINT chiave_primaria PRIMARY KEY (username)  
);
```

I campi *username* e *password* consentono la memorizzazione delle credenziali di accesso al sito web, mentre i campi *DB\_username* e *DB\_password* sono utilizzati per memorizzare le credenziali di accesso al DBMS associate al singolo utente<sup>8</sup>.

Le password saranno memorizzate nel database in forma cifrata utilizzando la funzione PHP `crypt`; il loro riconoscimento avverrà cifrando la password digitata dall'utente e confrontando il risultato con quello presente nel database: l'accesso sarà consentito se risulteranno uguali, negato altrimenti.

**OSSERVAZIONE** La dimensione assegnata dei campi relativi alle password non intende consentire password eccessivamente lunghe, ma l'algoritmo di cifratura implementato dalla funzione `crypt` può generare stringhe più lunghe della stringa fornita come input.

È necessario prima di tutto realizzare uno script PHP che consenta di inserire nella tabella *Utente* i dati relativi agli utenti del sito web:



```
<?php  
    if (!isset($_POST['username']) || !isset($_POST['password']))  
    {  
?  
    <html>  
        <head>  
            <title>Nuovo utente</title>  
        </head>  
        <body>  
            <form method="POST" action="nuovo_utente.php">  
                Username <input name="username" type="text"><br>  
                Password<input name="password" type="password"><br><br>  
                <input type="submit" value="Registra utente">  
            </form>  
        </body>  
    </html>  
<?php  
    }  
    else  
    {  
?  
    }  
}
```



```

<html>
  <head>
    <title>Nuovo utente</title>
  </head>
  <body>
    <?php
      $username = $_POST['username'];
      $password = $_POST['password'];
      if (strlen($username) != 0 && strlen($password) != 0)
      {
        $password = crypt($password, 0); // cifratura della
                                          // password

        $connection = new mysqli ("localhost","root","",
                                   "Disneyland");

        $query = "SELECT * FROM Utenti WHERE username =
                  '$username'";
        $result = $connection->query($query);
        if ($result->num_rows != 0)
          echo "L'utente $username &grave; gi&agrave;
              presente nel database.";

        else
        {
          $query = "INSERT INTO Utenti (username,
                                       password, DB_username, DB_password)
                   VALUES ('$username', '$password',
                             'root', '')";
          $connection->query($query);
          echo "L'utente $username &grave; stato
              aggiunto al database.";
        }
        $result->free();
        $connection->close();
      }
      else
        echo "Username/password non validi.";
    ?>
  </body>
</html>
<?php
}
?>

```

Lo script visualizza la seguente *form* HTML

che richiede l'inserimento dell'username e della password che sono inviati allo stesso script, denominato «nuovo\_utente.php», utilizzando il metodo HTTP POST per effettuarne l'inserimento nella tabella *Utenti* del database *Disneyland*.

**OSSERVAZIONE** Lo script «nuovo\_utente.php» non deve essere disponibile all'utente generico del sito, ma esclusivamente all'amministratore. Questa discriminazione, per esempio, può essere realizzata a livello sistemistico impostando dei privilegi di accesso al server HTTP che ospita il sito.

La gestione dell'accesso degli utenti al sito viene regolata inizializzando una nuova sessione al momento del riconoscimento della validità della password e, di conseguenza, dell'identità dell'utente che l'ha fornita effettuando il *login*: nessuna pagina del sito deve risultare accessibile se non è stato effettuato un *login* valido.

**OSSERVAZIONE** Le pagine del sito prevedranno per l'utente la possibilità di concludere la sessione effettuando esplicitamente un *logout*; inoltre – trascorso un certo tempo dal momento del *login* – la sessione sarà automaticamente invalidata.

Il seguente script PHP, denominato «login.php», consente a un utente del sito di effettuare il *login*:



```
<?php
    if (!isset($_POST['username']) || !isset($_POST['password']))
    {
?>
    <html>
        <head>
            <title>Login</title>
        </head>
        <body>
            <form method="POST" action="login.php">
                Username <input name="username" type="text"><br>
                Password<input name="password" type="password"><br><br>
                <input type="submit" value="Accedi">
            </form>
        </body>
    </html>
<?php
    }
    else
    {
?>
```



```

<html>
<head>
  <title>Login</title>
</head>
<body>
  <?php
    $username = $_POST['username'];
    $password = $_POST['password'];
    if (strlen($username) != 0 && strlen($password) != 0)
    {
      $connection = new mysqli("localhost","root","",
                              "Disneyland");
      $query = "SELECT * FROM Utenti WHERE username =
                '$username'";
      $result = $connection->query($query);
      if ($result->num_rows == 0)
      {
        echo "Utente $username sconosciuto: ";
        echo "<a href=\"http://localhost/login.php\">
              riprova.</a><br>";
      }
      else
      {
        $user_row = $result->fetch_array();
        // cifratura e verifica della password
        $password = crypt($password, 0);
        if ($password == $user_row['password'])
        {
          echo "Password corretta: ";
          echo "<a href=\"http://localhost/
                disneyland.php\"> accedi.</a><br>";
          // distruzione eventuale sessione
          // precedente
          session_start();
          session_unset();
          session_destroy();
          // inizializzazione nuova sessione
          session_start();
          $_SESSION['username'] = $username;
          $_SESSION['start_time'] = time();
          $_SESSION['DB_username'] =
            $user_row['DB_username'];
          $_SESSION['DB_password'] =
            $user_row['DB_password'];
          echo "<a href=\"http://localhost/logout.
                php\"> [$username logout]</a>";
        }
      }
    }
  }

```

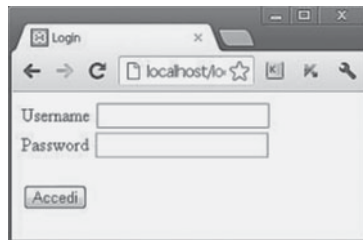
```

else
{
    echo "Password errata: ";
    echo "<a href=\"http://localhost/login.php\"> riprova.</a>";
}
}
$result->free();
$connection->close();
}
else
{
    echo "Username/password non validi: ";
    echo "<a href=\"http://localhost/login.php\"> riprova.</a>";
}
}

?>
</body>
</html>
<?php
}
?>

```

Lo script visualizza la seguente *form* HTML



che richiede l'inserimento dello username e della password, che sono inviate allo stesso script utilizzando il metodo HTTP POST, per effettuare il confronto della password digitata e cifrata con la cifratura della password associata all'username digitato memorizzata nella tabella *Utenti* del database *Disneyland*. Solo nel caso di password corretta viene creata e inizializzata una nuova sessione PHP, memorizzando le seguenti informazioni disponibili alle pagine del sito che saranno successivamente richieste dall'utente:

- username dell'utente;
- username per l'accesso al DBMS dell'utente;
- password per l'accesso al DBMS dell'utente;
- *timestamp* dell'ora di effettuazione del *login*.

**OSSERVAZIONE** Le credenziali per l'accesso al DBMS sono utilizzate da tutti gli script PHP invocati dall'utente che richiedono un'interazione

con il database. Il valore del *timestamp*, invece, viene utilizzato dalle pagine del sito per verificare se dal momento del *login* è trascorso un periodo di tempo tale da giustificare la chiusura automatica della sessione per motivi di sicurezza (*logout*).

Le pagine del sito devono impedire la loro visualizzazione in assenza di un *login* valido; a questo scopo il file «session.php» contiene il seguente script PHP che tutte le pagine del sito includono come prima riga di codice:

```
<?php
    session_start();
    if (!isset($_SESSION['start_time']))
    {
        header('Location:http://localhost/login.php');
        die;
    }
    else
    {
        $now = time();
        $time = $now - $_SESSION['start_time'];
        if ($time > 3600) // 3600s => 1h
        {
            header('Location:http://localhost/logout.php');
            die;
        }
    }
?>
```



In assenza di un *login* valido la sessione non risulta inizializzata e l'elemento di chiave «start\_time» dell'array superglobale \$\_SESSION non è di conseguenza definito, per cui la richiesta della pagina viene reindirizzata alla pagina «login.php». Nel caso in cui la sessione risulti inizializzata da un tempo superiore a un'ora (3600 secondi) la richiesta della pagina viene invece reindirizzata alla pagina «logout.php» che termina la sessione PHP. In entrambi i casi un'istruzione **die** termina lo script PHP, per cui la pagina il cui codice segue l'inclusione viene elaborata dal server HTTP e inviata al browser solo se una sessione risulta inizializzata da meno di un'ora.



#### ESEMPIO

Il codice della pagina principale del sito è ora il seguente:

```
<?php
    require "session.php";
?>
<html>
    <head>
        <title>DB & PHP test</title>
```





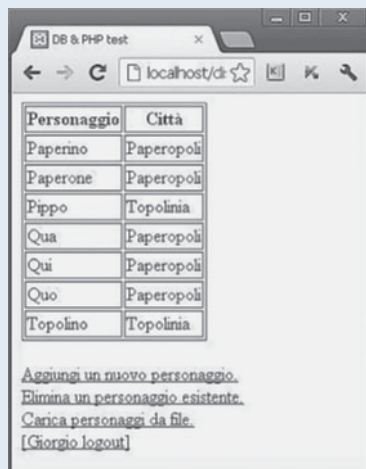
```

</head>
<body>
  <?php
    // connessione al database con credenziali di sessione
    $connection = new mysqli("localhost",$_SESSION['DB_username'],
                             $_SESSION['DB_password'], "Disneyland");
    $query = "SELECT nome,denominazione FROM Personaggi,Citta WHERE
              Personaggi.citta = Citta.sigla ORDER BY nome";
    $result = $connection->query($query);
    if ($result->num_rows != 0)
    {
        echo "<table border>";
        echo "<tr>";
        echo "<th>Personaggio</th>";
        echo "<th>Città</th>";
        echo "</tr>";
        while ($row = $result->fetch_array())
        {
            echo "<tr>";
            echo "<td>$row[nome]</td>";
            echo "<td>$row[denominazione]</td>";
            echo "</tr>";
        }
        echo "</table>";
    }
    $result->free();
    $connection->close();
  ?><br>
  <a href="http://localhost/add.php">
    Aggiungi un nuovo personaggio.
  </a><br>
  <a href="http://localhost/del.php">
    Elimina un personaggio esistente.
  </a><br>
  <a href="http://localhost/upload.php">
    Carica personaggi da file.
  </a><br>
  <?php
    echo "<a href='\"http://localhost/logout.php\"'>
          [$_SESSION[username] logout]</a>";
  ?>
</body>
</html>

```

**OSSERVAZIONE** Rispetto alla versione originale della pagina «disneyland.php» è ora richiesta l'inclusione dello script «session.php», che reindirizza la pagina in caso di sessione non valida o scaduta, e l'accesso al DBMS avviene mediante le credenziali di sessione che sono specifiche dell'utente che ha effettuato il *login*. Inoltre, all'elenco dei collegamenti

alle pagine del sito che consentono di effettuare le operazioni di inserimento, eliminazione e caricamento da file dei personaggi, ne viene aggiunto uno per effettuare il *logout*:



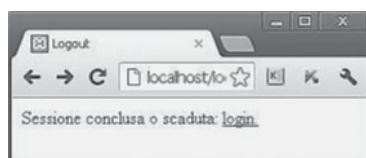
Le stesse modifiche sono state apportate a tutte le pagine del sito.

Lo script «logout.php» ha il solo compito di terminare la sessione PHP corrente invalidando il contenuto dell'array superglobale `$_SESSION`

```
<?php
// distruzione sessione corrente
session_start();
session_unset();
session_destroy();
$_SESSION = array();
?>
<html>
<head>
<title>Logout</title>
</head>
<body>
  Sessione conclusa o scaduta:
  <a href="http://localhost/login.php">login.</a>
</body>
</html>
```



e di visualizzare un collegamento alla pagina di *login*:



### 3 PDO (PHP Data Objects) per l'accesso a DBMS

#### ODBC (Open DataBase Connectivity)

ODBC è una API sviluppata da Microsoft negli anni '90 del secolo scorso con l'intento di definire un'interfaccia standard per l'accesso di codice in linguaggio C/C++ a DBMS in modo indipendente sia dal sistema operativo, sia dal DBMS stesso.

Praticamente tutti i DBMS esistenti – compreso My-SQL – dispongono di un «driver» ODBC che rende possibile l'accesso delle applicazioni ai dati mediante l'interfaccia standard. La API di ODBC è basata sull'uso del linguaggio SQL come ospite del linguaggio di sviluppo dell'applicazione.

#### PDO e indipendenza dal DBMS

L'adozione di PDO da parte dello sviluppatore consente di realizzare applicazioni web in linguaggio PHP indipendenti dal DBMS utilizzato. Questo consente in teoria al committente di scegliere liberamente il DBMS da utilizzare per l'applicazione. Purtroppo la parziale dipendenza del linguaggio SQL dal DBMS utilizzato limita questo vantaggio.

Le funzioni e le classi dell'interfaccia *mysqli* consentono l'accesso da codice PHP a un DBMS My-SQL: questo limite è superato da PDO (PHP Data Objects), un'estensione del linguaggio che consente di accedere a un qualsiasi DBMS per il quale sia stato realizzato uno specifico driver.

**OSSERVAZIONE** Esistono driver PDO per i più diffusi DBMS: Microsoft SQL-server, IBM DB2, My-SQL, Oracle DB, PostgreSQL, ... Inoltre esiste un driver per la API standard ODBC (*Open DataBase Connectivity*), che consente di connettersi virtualmente a un qualsiasi DBMS.

La classe principale – il cui costruttore realizza la connessione con il DBMS specificato – è *PDO*, di cui nella TABELLA 13 si elencano i metodi fondamentali.

TABELLA 13

Metodo	Descrizione
<code>__construct</code>	È il costruttore della classe. Effettua la connessione a un server DBMS di cui viene fornito come argomento il tipo, il nome (o indirizzo IP) del computer su cui è in esecuzione, l' <i>username</i> e la <i>password</i> dell'utente e il <i>database</i> su cui si intende operare
<code>beginTransaction</code>	Disabilita la modalità <b>AUTOCOMMIT</b> del DBMS. Restituisce <b>TRUE</b> in caso di successo, oppure <b>FALSE</b> in caso di errore
<code>commit</code>	Esegue il <b>COMMIT</b> dei comandi eseguiti in precedenza. Restituisce <b>TRUE</b> in caso di successo, oppure <b>FALSE</b> in caso di errore
<code>exec</code>	Esegue un comando SQL di tipo DML. Restituisce il numero di righe interessate dal comando
<code>prepare</code>	Predispose un comando SQL parametrizzato per l'esecuzione (i parametri sono indicati nel comando SQL per mezzo di un nome simbolico preceduto dal simbolo «:»). Restituisce un oggetto di classe <i>PDOStatement</i>
<code>query</code>	Esegue una query SQL. Restituisce un oggetto di classe <i>PDOStatement</i> come risultato
<code>rollBack</code>	Esegue il <b>ROLLBACK</b> dei comandi eseguiti in precedenza. Restituisce <b>TRUE</b> in caso di successo, oppure <b>FALSE</b> in caso di errore

**OSSERVAZIONE** Tutti i metodi della classe *PDO* sollevano un'eccezione di tipo *PDOException* in caso di errore.

Per chiudere la connessione al DBMS di un oggetto di classe *PDO* è sufficiente assegnare il valore **NULL** all'oggetto stesso.

Gli oggetti di classe *PDOStatement* possono rappresentare sia un comando SQL parametrizzato, sia il risultato di una query; nella TABELLA 14 sono riportati i principali metodi della classe.

TABELLA 14

Metodo	Descrizione
<code>bindParam</code>	<p>Istanza il comando parametrico sostituendo valori effettivi ai parametri. Per ogni valore è necessario definire il tipo:</p> <ul style="list-style-type: none"> <li>• <code>PDO::PARAM_STR</code>: stringa di caratteri;</li> <li>• <code>PDO::PARAM_INT</code>: valore numerico intero.</li> </ul> <p>Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore</p>
<code>closeCursor</code>	<p>Conclude l'uso del risultato di una query liberando le risorse allocate. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore</p>
<code>columnCount</code>	<p>Restituisce il numero di colonne del risultato di una query</p>
<code>execute</code>	<p>Esegue il comando parametrico con i parametri istanziati. Restituisce <code>TRUE</code> in caso di successo, oppure <code>FALSE</code> in caso di errore</p>
<code>fetch</code>	<p>Recupera la riga successiva del risultato di una query. Un parametro determina la modalità di restituzione:</p> <ul style="list-style-type: none"> <li>• <code>PDO::FETCH_ASSOC</code>: array associativo;</li> <li>• <code>PDO::FETCH_NUM</code>: array indicizzato;</li> <li>• <code>PDO::FETCH_BOTH</code>: array associativo e indicizzato;</li> <li>• <code>PDO::FETCH_OBJ</code>: oggetto con proprietà aventi il nome delle colonne.</li> </ul> <p>In caso di errore restituisce <code>FALSE</code></p>
<code>fetchAll</code>	<p>Restituisce un array in cui ogni elemento è un array che rappresenta una singola riga del risultato di una query</p>
<code>fetchColumn</code>	<p>Restituisce il valore di una singola colonna del risultato di una query</p>
<code>rowCount</code>	<p>Restituisce il numero di righe coinvolte dall'ultimo comando SQL di tipo DML eseguito. Non è utilizzabile per ottenere il numero di righe del risultato di una query</p>



## ESEMPIO

La pagina «disneyland.php» che visualizza i personaggi del database *Disneyland* può essere così realizzata utilizzando PDO:

```
<html>
  <head>
    <title>DB & PHP test</title>
  </head>
  <body>
    <?php
      try {
        $connection = new PDO("mysql:host=localhost;dbname=Disneyland","root","");
        $query = "SELECT nome,denominazione FROM Personaggi,Citta WHERE
                  Personaggi.citta = Citta.sigla ORDER BY nome";
        $result = $connection->query($query);
        if ($row = $result->fetch(PDO::FETCH_OBJ))
        {
          echo "<table border>";
          echo "<tr>";
          echo "<th>Personaggio</th>";
```

```

        echo "<th>Citt&grave;</th>";
        echo "</tr>";
        echo "<tr>";
        echo "<td>$row->nome</td>";
        echo "<td>$row->denominazione</td>";
        echo "</tr>";
        while ($row = $result->fetch(PDO::FETCH_OBJ))
        {
            echo "<tr>";
            echo "<td>$row->nome</td>";
            echo "<td>$row->denominazione</td>";
            echo "</tr>";
        }
        echo "</table>";
    }
    $result->closeCursor();
    $connection = NULL;
}
catch (PDOException $exception)
{
    echo "Errore nell'accesso al database.<br>";
    die;
}

?><br>
<a href="http://localhost/add.php">
    Aggiungi un nuovo personaggio.
</a><br>
<a href="http://localhost/del.php">
    Elimina un personaggio esistente.
</a><br>
<a href="http://localhost/upload.php">
    Carica personaggi da file.
</a>
</body>
</html>

```

La pagina «insert.php» che aggiunge un nuovo personaggio al database esemplifica l'esecuzione di un comando SQL di tipo DML:

```

<html>
<head>
    <title>DB & PHP test: INSERT</title>
</head>
<body>
<?php
    $personaggio = $_GET["personaggio"];
    $citta = $_GET["citta"];
    try {
        $connection = new PDO("mysql:host=localhost;dbname=Disneyland","root","");
        $query = "SELECT COUNT(*) FROM Personaggi WHERE nome = '$personaggio'";
        $result = $connection->query($query);
        if ($result->fetchColumn() > 0)
            echo "Il personaggio $personaggio &grave; gi&grave; presente nel database.";
        else
        {

```

```

        $command = "INSERT INTO Personaggi VALUES ('$personaggio','$citta)";
        $connection->exec($command);
        echo "Il personaggio $personaggio &grave; stato aggiunto al database.";
    }
    $result->closeCursor();
    $connection = NULL;
}
catch (PDOException $exception)
{
    echo "Errore nell'accesso al database.<br>";
    die;
}
?><br>
<a href="http://localhost/disneyland.php">
    Visualizza elenco personaggi.
</a>
</body>
</html>

```

**OSSERVAZIONE** Gli script dell'esempio precedente illustrano alcune tecniche di programmazione per gestire l'indisponibilità del numero di righe del risultato di una query. La tabella visualizzata dalla pagina «disneyland.php» viene impostata solo se il risultato della query non è vuoto, condizione verificata dalla struttura condizionale che comprende la classica struttura iterativa per il recupero delle singole righe del risultato. Nello script «insert.php» per verificare l'esistenza del personaggio nel database viene eseguita la query SQL

```
SELECT COUNT(*) FROM Personaggi WHERE nome = '...';
```

e verificato che il risultato scalare risulti uguale a 0.



#### ESEMPIO

La pagina «insert\_from\_file.php» esemplifica la gestione dei comandi SQL parametrizzati utilizzando PDO:

```

<html>
<head>
    <title>DB & PHP test: UPLOAD</title>
</head>
<body>
<?php
    if ($_FILES["personaggi"]["error"] == UPLOAD_ERR_OK)
    {
        try
        {
            $connection = new PDO("mysql:host=localhost;dbname=Disneyland","root","");
            $statement = $connection->prepare("INSERT INTO Personaggi VALUES
                (:personaggio,:citta)");
            $personaggi = file($_FILES["personaggi"]["tmp_name"],
                FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
            foreach ($personaggi as $linea)

```

```

{
    $dati = explode(",", $linea);
    $personaggio = trim($dati[0]);
    $citta = trim($dati[1]);
    $statement->bindParam(":personaggio", $personaggio, PDO::PARAM_STR);
    $statement->bindParam(":citta", $citta, PDO::PARAM_STR);
    if ($statement->execute())
        echo "Il personaggio $personaggio &egrave; stato aggiunto al database.<br>";
    else
        echo "Errore: il personaggio $personaggio NON &egrave; stato aggiunto al database.<br>";
}
$connection = NULL;
}
catch (PDOException $exception)
{
    echo "Errore nell'accesso al database.<br>";
    die;
}
}
else
    echo "Errore caricamento del file.";
?><br>
<a href="http://localhost/disneyland.php">
    Visualizza elenco personaggi.
</a>
</body>
</html>

```

## Sintesi

■ **Siti web e DBMS.** La possibilità di generare pagine web dinamiche non sarebbe una delle tecniche alla base dei siti moderni senza la possibilità di accedere a un DBMS da parte del linguaggio di *scripting server-side*; molti siti che utilizzano PHP come tecnologia lato server sono basati su DBMS My-SQL, un prodotto distribuito con licenza open-source da parte di Oracle Corporation.

■ **SQL Embedded.** Le soluzioni per accedere a un database da uno script lato server prevedono una stretta integrazione del linguaggio SQL come ospite del linguaggio PHP. L'interazione con il DBMS avviene sempre mediante stringhe costruite dinamicamente che rappresentano comandi SQL che funzioni o metodi PHP inviano al DBMS stesso.

■ **Interfaccia *mysqli*.** Fin dalle prime versioni del linguaggio PHP la libreria di funzioni denomi-

nata «*mysql*» consente di interagire con un DBMS My-SQL.

■ **Interfaccia avanzata *mysqli*.** Le versioni più recenti del linguaggio PHP dispongono di una libreria denominata «*mysqli*» (*mysql improved*) per l'interfacciamento con DBMS My-SQL. Caratteristica di questa libreria è il fatto di esporre sia un approccio procedurale non diverso da quello originale della libreria «*mysql*», sia un approccio orientato agli oggetti basato sulle classi *mysqli*, che rappresenta la connessione con il DBMS My-SQL, *mysqli\_result*, che rappresenta il risultato di una query, e *mysqli\_stmt*, che rappresenta un comando SQL parametrico. Un oggetto di classe *mysqli* espone metodi specifici per la gestione delle transazioni.

■ **Controllo degli errori.** L'operatore @ del linguaggio PHP premesso a un'espressione impe-

disce l'inserimento di eventuali messaggi di errore nell'output prodotto da uno script, situazione incomprensibile all'utente che lo sviluppatore deve evitare; al tempo stesso tutte le condizioni che potenzialmente generano errori devono essere verificate e opportunamente gestite dal codice.

■ **Controllo degli input.** Dato che l'inoltro di dati a uno script PHP avviene, anche nel caso di valori numerici, mediante stringhe di caratteri utilizzando sia il metodo «GET» sia il metodo «POST», uno stretto controllo sui dati ricevuti è una pratica che previene numerosi errori potenziali e alcuni attacchi malevoli.

■ **Gestione delle password.** La gestione degli utenti – con le relative credenziali di accesso, per esempio username e password – è una delle problematiche più comuni nella realizzazione di un sito web. I dati relativi al singolo utente devono essere memorizzati in una specifica tabella di un database ospitato dal DBMS di supporto del sito web stesso; le password devono essere memorizzate nel database in forma cifrata: il loro riconoscimento avviene cifrando la password digitata dall'utente e confrontando il risultato con quello presente nel database, consentendo l'accesso solo se risulteranno uguali, negato altrimenti.

■ **Login e sessioni.** La gestione dell'accesso degli utenti al sito viene regolata inizializzando

una nuova sessione PHP al momento del riconoscimento della validità della password e, di conseguenza, dell'identità dell'utente che l'ha fornita effettuando il *login*: nessuna pagina del sito deve risultare accessibile se non è stato effettuato un *login* valido. Le pagine del sito prevederanno per l'utente la possibilità di concludere la sessione effettuando esplicitamente un *logout*; inoltre – trascorso un certo tempo dal momento del *login* – la sessione sarà automaticamente invalidata.

■ **PHP Data Objects.** Le funzioni e le classi dell'interfaccia *mysqli* consentono l'accesso da codice PHP a un DBMS My-SQL. Questo limite è superato da PDO (*PHP Data Objects*), un'estensione del linguaggio che consente di accedere a un qualsiasi DBMS per il quale sia stato realizzato uno specifico driver. Esistono driver PDO per i più diffusi DBMS: Microsoft SQL-server, IBM DB2, My-SQL, Oracle DB, PostgreSQL, ... Inoltre esiste un driver per la API standard ODBC (*Open DataBase Connectivity*), che consente di connettersi virtualmente a un qualsiasi DBMS. La classe principale di PDO è *PDO*, il cui costruttore realizza la connessione con il DBMS indicato e il database specificato; gli oggetti di classe *PDOStatement*, invece, possono rappresentare sia un comando SQL parametrizzato sia il risultato di una query. I metodi delle classi *PDO* e *PDOStatement* sollevano un'eccezione di tipo *PDOException* in caso di errore.

## QUESITI

**1** L'accesso a un DBMS My-SQL da parte di uno script PHP ...

- A ... avviene scrivendo direttamente i comandi SQL nel codice PHP.
- B ... avviene fornendo come parametro a funzioni o metodi PHP una stringa che rappresenta un comando SQL.
- C ... avviene senza utilizzare il linguaggio SQL.
- D Nessuna delle precedenti.

**2** La funzione `mysql_query` dell'interfaccia originale *mysql* è utilizzata per ...

- A ... eseguire esclusivamente comandi SQL di tipo DML.

B ... eseguire esclusivamente query SQL.

C ... eseguire sia comandi SQL di tipo DML sia query SQL.

D Nessuna delle precedenti.

**3** Le singole righe del risultato di una query SQL eseguita utilizzando l'interfaccia originale *mysql* possono essere ...

A ... memorizzate esclusivamente in un array indicizzato.

B ... memorizzate esclusivamente in un array associativo.

C ... memorizzate in un array indicizzato, in un array associativo o in una combinazione dei due.

D Nessuna delle precedenti.



**4** La gestione delle transazioni con la libreria avanzata *mysqli* ...

- A ... non può essere effettuata in nessun modo.
- B ... può essere effettuata solo inviando i comandi SQL COMMIT/ROLLBACK al DBMS.
- C ... può essere effettuata utilizzando specifici metodi della classe *mysqli*.
- D Nessuna delle precedenti.

**5** Il risultato fornito dal metodo *mysqli\_query* della classe *mysqli* fornendo come argomento una query SQL ...

- A ... è una stringa di caratteri.
- B ... è un array.
- C ... è un oggetto di classe *mysqli\_result*.
- D ... è un oggetto di classe *mysqli\_stmt*.

**6** Indicare le affermazioni vere relativamente a un comando SQL parametrizzato utilizzando l'interfaccia avanzata *mysqli*.

- A È un oggetto di classe *mysqli\_stmt*.
- B È generato dal metodo *prepare* della classe *mysqli*.
- C Espone il metodo *bind* per istanziare i parametri.
- D Espone il metodo *execute* per eseguire il comando.

**7** La funzione PHP *filter\_input* ...

- A ... consente di validare i dati forniti come input a uno script con metodi standard.
- B ... consente di filtrare i dati forniti come input a uno script invocando una funzione definita dall'utente.

- C ... consente di escludere dall'input di uno script i dati non utilizzati.
- D Nessuna delle precedenti.

**8** Una corretta gestione delle password prevede che ...

- A ... siano memorizzate in forma cifrata e decifrate per essere confrontate con quanto digitato dall'utente.
- B ... siano memorizzate in forma cifrata e confrontate con la cifratura di quanto digitato dall'utente.
- C ... siano memorizzate in chiaro e cifrate al momento del confronto con quanto digitato dall'utente.
- D Nessuna delle precedenti.

**9** L'estensione PDO del linguaggio PHP ...

- A ... consente la connessione a un qualsiasi DBMS.
- B ... consente le connessione esclusivamente al DBMS My-SQL.
- C ... consente la connessione a un qualsiasi DBMS utilizzando i driver ODBC.
- D ... consente la connessione ai DBMS prodotti da Oracle.

## ESERCIZI

**1** Dato il database rappresentato nel diagramma di FIGURA 1, realizzare le seguenti pagine web utilizzando il linguaggio PHP:

- a) visualizzazione in una tabella dell'elenco dei libri stampati, con indicazione del codi-



FIGURA 1



FIGURA 2

- ce ISBN, del titolo, dell'autore e dell'editore, in un determinato periodo di tempo indicato dall'utente inserendo l'anno di inizio e di fine;
- b) eliminazione di un libro dal database con indicazione del codice ISBN;
- c) inserimento di un nuovo libro nel database con selezione dell'autore e dell'editore da elenchi predefiniti.

**2** In riferimento all'esempio del paragrafo 4.2 del capitolo A2, realizzare in linguaggio PHP uno script per la prenotazione di un posto da parte di un utente registrato, a partire dalla visualizzazione dell'elenco dei treni disponibili per il viaggio da una stazione di partenza a una stazione di arrivo in una particolare data.

**3** Un'associazione di *car-sharing* mantiene un database del parco auto e dei soci al fine di registrare i noleggi e verificare le disponibilità delle vetture; il diagramma del database è quello di FIGURA 2. Realizzare le seguenti pagine web utilizzando il linguaggio PHP:

- a) visualizzazione dell'elenco delle auto disponibili in un determinato periodo compreso tra due date;

- b) visualizzazione dei noleggi effettuati da un socio in un determinato periodo;
- c) inserimento di un nuovo noleggio;
- d) aggiornamento della situazione relativa a un noleggio al momento in cui un'auto viene restituita.

**4** Un'agenzia espressi ha sedi e clienti in varie città: i clienti mittenti consegnano i plichi alla sede di partenza e i clienti destinatari ritirano i plichi nella sede di arrivo. Dalla spedizione alla consegna ogni plico è identificato da un codice numerico: la data/ora di spedizione viene registrata quando il mittente consegna il plico, mentre la data/ora di consegna viene registrata quando il destinatario ritira il plico. Il sistema informatico dell'agenzia è interamente organizzato su un'interfaccia web e si basa su un database realizzato a partire dal diagramma di FIGURA 3. Realizzare le seguenti pagine web utilizzando il linguaggio PHP:

- a) ricezione dal mittente di una nuova spedizione; dato che il codice di spedizione è un valore numerico progressivo, lo script deve visualizzare il codice assegnato alla spedizione;

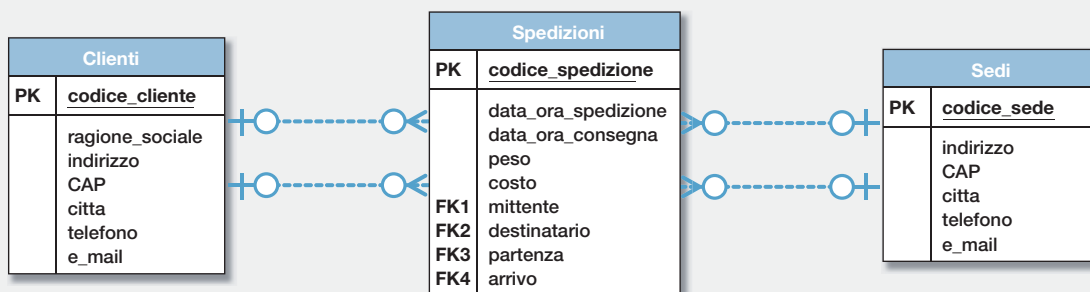


FIGURA 3

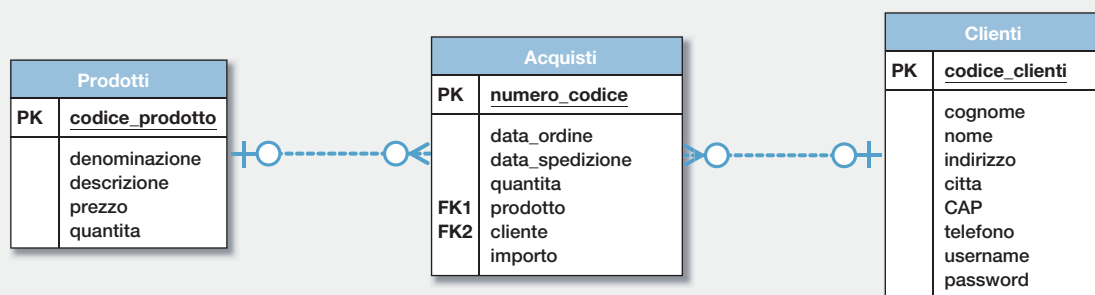


FIGURA 4

- b) elenco delle spedizioni non ancora consegnate con indicazione del codice, della data/ora di spedizione, della ragione sociale del destinatario e della sede di arrivo;
- c) consegna al destinatario di una spedizione;
- d) elenco delle spedizioni effettuate oggi da una sede dell'agenzia selezionata dall'elenco delle sedi;
- e) stato (consegnata o meno) di una specifica spedizione identificata dal codice numerico con indicazione della sede di partenza e di arrivo.

Tenendo conto che le pagine (a), (b), (c) e (d) possono essere richieste solo dal personale dell'agenzia, integrare il database e il codice degli script in modo che esse siano protette mediante una procedura di *login*.

## LABORATORIO

- 1 Un nuovo sito Internet per la vendita di prodotti online utilizzerà il database illustrato nel diagramma di FIGURA 4 e gestito con DBMS My-SQL. Tutte le attività sono realizzate mediante un'applicazione web utilizzando un normale browser e la tecnologia *server-side* PHP. Le pagine dinamiche del sito devono consentire:

- al responsabile del magazzino la visualizzazione dell'elenco dei prodotti disponibili per la vendita con indicazione del codice, del prezzo e della quantità disponibile e la possibilità di modificare la quantità disponibile di un singolo prodotto;
- al responsabile delle spedizioni di inserire la data di spedizione di un ordine da evadere di cui è noto il numero;
- al responsabile del marketing di aggiungere un nuovo prodotto a quelli già in vendita;
- al responsabile dei clienti di inserire un ordine effettuato telefonicamente da un cliente;
- a un cliente che abbia effettuato un *login* valido, di ordinare una quantità variabile (inferiore ovviamente alla disponibilità) di un singolo prodotto<sup>9</sup>, trascurando le problematiche di pagamento, e la visualizzazione dello stato (spedito o meno) di un acquisto individuato dal numero di ordine;
- a un nuovo cliente la registrazione nel sito.

Dopo averne definito il DB-schema in linguaggio SQL ed avere creato il database realizzare gli script PHP che generano le pagine dinamiche elencate.

9. Tenendo conto che il numero d'ordine è un campo contatore, la pagina deve visualizzare al termine dell'operazione il numero di ordine generato dal DBMS.

## Chapter 7

### Database interaction

It would make little sense these days to have a static website that doesn't change unless you physically alter the HTML or PHP of each file. There is usually a need to store and retrieve dynamic information as it relates to the content of a website or web application. In this chapter, we will look at how to make your pages draw some of their content from a database.

#### MySQLi Object Interface

The most popular database platform used with PHP is the MySQL database. If you look at the MySQL website you will discover that there are a few different versions of MySQL you can use. We will look at the freely distributable version known as the *community server*. PHP has a number of different interfaces to this database tool as well, so we will look at the object-oriented interface known as MySQL Improved extension (MySQLi). If you read the previous chapter on OOP with PHP, the use of this interface should not be overly foreign.

First, let's use the very basic database schema I hinted at in the previous chapter by extending the example of a rudimentary guestbook page we started with. We'll add the ability to actually save the entries into the database table. Here is the structure of the *guests* table:

```
table:      guests
guestid    int(11)
fname      varchar(30)
lname      varchar(40)
comments   text
```

And here is the SQL code to create it:

```
CREATE DATABASE 'website' ;
USE 'website' ;

CREATE TABLE 'guests' (
  'guestid' INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  'fname' VARCHAR( 30 ) NOT NULL ,
  'lname' VARCHAR( 40 ) NOT NULL ,
  'comments' TEXT NOT NULL
)
```

Since this object-oriented interface is built into PHP with a standard installation configuration (you just have to activate the MySQL extension in your PHP environment), all you have to do to start using it is instantiate its class, as in the following code:

```
$mydb = new mysqli('localhost', 'dbuser', 'dbpassword', 'dbname');
```

In our example, we have a database named *website*, and we will pretend that our username is *petermac* with the password *1q2w3e4r*. The actual code that we use is:

```
$mydb = new mysqli('localhost', 'petermac', '1q2w3e4r', 'website');
```

This gives us access to the database engine itself within the PHP code; we will specifically access tables and other data later. Once this class is instantiated into the variable *\$mydb*, we can use methods on that object to do our database work.

This chapter assumes an understanding of the SQL command language and will not spend time covering it. There are many online resources and printed books that can assist you with crafting SQL code.

[...] will take the values from the *\$\_POST* array and save them into the database. Here is the full listing of the code:

```
$mydb = new mysqli('localhost', 'petermac', '1q2w3e4r', 'website');

$sql = "INSERT INTO guests (fname, lname, comments)
VALUES ('$_POST[fname]', '$_POST[lname]', '$_POST[comments]')";

if ($mydb->query($sql) == TRUE) {
    echo "Guest entry saved successfully.";
```

**overly foreign**

troppo sconosciuto  
completamente  
estraneo

**to hint at**

accennare a, alludere a

**for the sake  
of simplicity**

per motivi di semplicità

**to tidy up**

mettere in ordine,  
riordinare

```

    } else {
        echo "INSERT attempt failed, please try again later, or call tech support" ;
    }
}

$mydb->close();

```

For the sake of simplicity and clarity, we are not concerned here with security of the content coming from the user (`$_POST` array). Be sure to review Chapter 9 on security, particularly the section titled “Cross-Site Scripting (XSS) and SQL Injection” [...] before you use any of your SQL code on a public site.

First, we instantiate the `MySQLi` class into an object with the variable identifier `$mydb`. Next, we build our SQL command string and save it to a variable called `$sql`. Then we call the query method of the class, and at the same time test its return value to determine if it was successful (`true`) and comment to the screen accordingly. Last, we call the close method on the class to tidy up and destroy the class from memory.

### Retrieving Data for Display

In another area of your website, you may want to draw out a listing of your guests and show a short clip of their comments. We can accomplish this by employing the same `MySQLi` class and working with the result set that is generated from a `SELECT` SQL command. There are many ways to display the information in the browser, and we’ll look at one example of how this can be done. Notice that the returned result is a different object than the `$mydb` that we first instantiate. PHP instantiates the result object for you and fills it with any returned data. Here is the code:

```

$mydb = new mysqli('localhost', 'petermac', '1q2w3e4r', 'website');
$sql = "SELECT * FROM Guests ORDER BY lname, fname";
$result = $mydb->query($sql);

while( $row = $result->fetch_assoc() ){
    echo $row['fname']." ".$row['lname'];
    echo " made these comments: ".substr($row['comments'],0,150);
    echo "<br/>";
}

$result->close();
$mydb->close ();

```

Here, we are using the query method call and storing the returned information into the variable called `$result`. Then, we use a method of the result object called `fetch_assoc` to provide one row of data at a time, and we store that single row into the variable called `$row`. This continues while there are rows to process, and within that while loop, we are dumping content out to the browser window. Finally, we close both the result and the database objects.

One of the most useful methods I have found in `MySQLi` is `multi_query`; this method allows you to run multiple SQL commands in the same statement. If you want to do an `INSERT`, and then an `UPDATE` statement based on similar data, you can do it all in one method call.

We have, of course, just scratched the surface of what the `MySQLi` class has to offer. You can find the documentation for the class at <http://www.php.net/mysqli>, and you will see the extensive list of methods that are part of this class. Each result class is documented within the appropriate subject area.

P. MacIntyre, *PHP, the good parts*, O’Reilly, 2010

### QUESTIONS

- a Can PHP code interact with a different DBMS than MySQL by using the `MySQLi` interface?
- b What parameters does the `mysqli` class require?
- c Which method of the `mysqli` class executes an SQL command?
- d Is the result object of a query the same object instance of the `mysqli` class used to connect to a database?
- e What does the `fetch_assoc` method of the result object of a query provide?

# Indice analitico

## A

@, operatore, 390  
ACID (*Atomicity, Consistency, Isolation, Durability*), 69, 80  
algebra relazionale, 65, 69, 80, 94  
analista, 8  
Apache, 266  
– Derby, 20  
API (*Application Program Interface*), 146, 147  
– JDBC, 178  
architettura *client/server*, 146, 265  
archivio di dati, 14, 24  
– v. *anche* file  
associazioni, 33, 78  
– binarie  
– – 1:1, 33  
– – 1:N, 34  
– – M:N, 34  
– parziali, 34  
– totali, 34  
atomicità, 69, 80  
*atomicity*, 69, 80  
attributo/i, 36, 78, 327, 330, 351  
– non primo, 44  
– primo, 44  
– statici, 335

## B

base di dati, 18, 19, 24  
– accesso, 144, 177  
– con JDBC, 144  
– amministratore, 21, 24  
– relazionale, 30, 45, 48, 79, 253  
– – linguaggi, 64  
– – normalizzazione, 48, 49, 79  
– – – prima forma normale, 49  
– – – seconda forma normale, 49  
– – – terza forma normale, 49  
– – progettazione, 48, 58  
– sistema di gestione, 11, 18  
– – accesso in linguaggio PHP, 357  
– – architettura logica, 22, 25  
– – gestione dei privilegi, 123  
– – indipendenza  
– – – fisica, 20  
– – – logica, 20  
– – integrazione, 20  
– – integrità, 20  
– – livello fisico  
– – – di memorizzazione, 23, 25  
– – livello logico  
– – – globale, 23, 25  
– – – utente, 22, 25  
– – monoutente, 67  
– – multiutente, 67  
– – PDO (*PHP Data Objects*), 388  
– – progettazione  
– – – fisica, 21  
– – – logica, 21  
– – programmatori di applicazioni, 21  
– – transazioni, 67  
– – – atomicità (*atomicity*), 69  
– – – consistenza (*consistency*), 69  
– – – gestione, 68  
– – – isolamento (*isolation*), 69  
– – – persistenza (*durability*), 69  
– – utenti  
– – – amministratori, 21  
– – – avanzati, 21  
– – – finali, 21  
BaseX, 226  
BCNF (*Boyce-Codd Normal Form*), 55  
*Begin Of Transaction* (BOT), 68  
BOT (*Begin Of Transaction*), 68  
Boyce-Codd, forma normale, 55, 80  
Boyce-Codd Normal Form (BCNF), 55, 80

## C

calcolo relazionale, 65  
cardinalità, 42  
CCM (*Concurrency Control Manager*), 68  
Chen, Peter, 33  
chiave, 43, 79  
– candidata, 44, 79  
– esterna, 36, 44, 79  
– – integrità referenziale, 44  
– primaria, 36, 44, 79  
– – univocità, 44  
chiusura, proprietà di, 100  
ciclo di vita, 6  
classe/i, 327, 351  
– astratte, 352  
– membri  
– – accessibilità, 351  
– – statici, 351  
classi CRUD, 157  
*client*, 266, 310  
*client/server*, 266, 310  
clonazione, 352  
Codd, Edgar, 42  
comandi  
– DDL, 114  
– DML, 124  
– SQL, 152, 178  
*commit*, 68, 80  
*conceptual modeling*, 33  
*Concurrency Control Manager* (CCM), 68  
*consistency*, 69, 80  
consistenza, 69, 80  
*cookie*, 306, 312  
costruttore, 330, 351  
*Create*, 157  
crisi del software, 6  
CRUD (*Create, Read, Update, Delete*), 179

## D

*Data Base Administrator* (DBA), 21  
*Data Base Management System* (DBMS), 11, 18, 23, 24  
– v. *anche* base di dati, sistema di gestione

*Data Definition Language* (DDL), 21, 64  
*Data Manipulation Language* (DML), 21, 65  
– autonomi, 21  
– ospitati, 21  
*database*, v. base di dati  
dati, 3, 14, 23  
– accesso concorrente, 128, 132  
– aspetto  
– – estensionale, 12, 23  
– – intensionale, 12, 23  
– estensione, 13  
– file, 14  
– – operazioni sui, 15  
– – organizzazione, 15  
– gestione, 23  
– approccio basato su *file system*, 23  
– – approccio fondato su DBMS, 23  
– indipendenza, 24  
– – fisica, 24  
– – logica, 24  
– integrazione, 24  
– integrità, 24  
– intensione, 13  
– istanze, 12  
– modello, 21  
– schema, 12  
*DB\_password*, 380  
*DB\_username*, 380  
DB-schema, 121, 132  
DBA (*Data Base Administrator*), 21  
DBMS (*Data Base Management System*), 11, 18, 23, 24, 148  
– basati su *file system*, 20  
– connessione al server, 149, 151, 178  
– disconnessione dal server, 155, 179  
– *middleware*, 149, 178  
DBMS My-SQL, 144, 357, 358, 392  
– *connector*, 357  
– driver JDBC, 151  
– linguaggio PHP, 357

- interfaccia, 358
- controllo degli errori, 375, 392
- controllo degli input, 375, 393
- gestione degli utenti, 379, 393
- gestione delle password, 379, 393
- *mysql*, 359
- *mysqli*, 365
- DDL (*Data Definition Language*), 21, 24, 64
- decomposizione senza perdita, 50
- Delete*, 157
- diagrammi E/R (entità/relazioni), 33, 78
- associazioni, 33, 78
- binarie
- – 1:1, 33
- – 1:N, 34
- – M:N, 34
- parziali, 34
- totali, 34
- attributi, 36, 78
- entità, 33, 78
- dependenza funzionale, 52, 79
- banale, 52
- completa, 52
- parziale, 52
- distruttore, 351
- DML (*Data Manipulation Language*), 21, 24, 65
- Document Object Model* (DOM), 237, 245, 253
- Document Type Definition* (DTD), 191
- DOM (*Document Object Model*), 237, 245, 253
- modalità di *parsing*, 237, 254
- domini, 42
- driver*, 148, 150, 178
- JDBC, 148, 151
- *pure Java*, 149, 178
- – connessione diretta al server DBMS, 149, 178
- – realizzazione in Java
- – completa (DBMS *middleware*), 149, 178
- – ODBC *bridge*, 148, 178
- – parziale, 149, 178
- Driver Manager*, 147, 150, 178
- DTD (*Document Type Definition*), 191
- durabilità, 69, 80
- durability*, 69, 80

**E**

- eccezioni, 352
- End of Transaction* (EOT), 68
- entità, 33, 78
- EOT (*End of Transaction*), 68

- ereditarietà, 352
- Extensible Markup Language* (XML), 184

**F**

- Factory-method design pattern*, 237
- file, 14, 24
- apertura, 15, 24
- cancellazione, 15, 24
- chiusura, 15, 24
- creazione, 15, 24
- di dati, 14
- – operazioni sui, 15
- inserimento, 15, 24
- modalità di accesso, 24
- modifica, 15, 24
- organizzazione, 15, 24
- – casuale (*random*), 24
- – diretta, 24
- – fisica, 15
- – indicizzata (*indexed*), 24
- – logica, 15
- – – accesso casuale (*random*), 16
- – – accesso diretto, 16
- – – indicizzata (*indexed*), 17
- – – sequenziale, 16
- – sequenziale, 24
- *record*, 14, 24
- registrazione, 14, 24
- ricerca, 15, 24
- FK (*Foreign Key*), 37
- Foreign Key* (FK), 37
- funzioni di aggregazione, 107, 132

**G**

- geo-database*, 119
- Gis, sistemi, 119
- grado, 42

**H**

- HTML, 311
- file dal browser al server, 372
- *form*, 264, 286, 287, 311
- gestione con PHP, 286
- *input*, 288, 311
- *option*, 288
- *select*, 288
- *textarea*, 288
- HTTP, *header*, 299

**I**

- incapsulamento, 327
- informazione, 3, 4, 23
- inner join*, 132
- integrità referenziale, 121
- interazione *client/server*, 266, 310
- Internationalized Resource Identifier* (IRI), 279

- Internet Media Type*, 373
- *application*, 373
- *audio*, 373
- *image*, 373
- *message*, 373
- *model*, 373
- *multipart*, 373
- *text*, 373
- *video*, 373
- IRI (*Internationalized Resource Identifier*), 279
- isolamento, 80
- isolation*, 69

**J**

- Java, 144, 149
- API per la gestione di documenti XML, 236
- classi CRUD, 157
- elaborazione di comandi, 149
- *query* SQL, 149
- tipi, 157
- – corrispondenza con tipi SQL, 157, 166
- XQuery, 236
- Java API for XML Processing* (JAXP), 236, 253
- Java Data Objects* (JDO), 158
- Java DataBase Connectivity* (JDBC), 144, 146, 147, 177
- JavaBean*, 167
- JAXP (*Java API for XML Processing*), 236, 253
- JDBC (*Java DataBase Connectivity*), 144, 146, 147, 177
- *driver*, 148, 151, 178
- – JDBC-ODBC *bridge*, 148, 178
- – *pure Java*, 149, 178
- – – connessione diretta al server DBMS, 149, 178
- – – realizzazione in Java
- – – completa (DBMS *middleware*), 149, 178
- – – parziale, 149, 178
- transazioni, 176, 179
- JDBC-ODBC *bridge*, 148, 178
- JDO (*Java Data Objects*), 158
- JDOM, 246
- join*, 100, 131

**L**

- linguaggio/i
- di modellizzazione, 9
- di *scripting client-side*, 267
- di *scripting server-side*, 267, 357, 392
- grafici, 66
- naturali, 66
- per DBMS, 80
- relazionale, 65

- – completo, 69
- livello
- fisico di memorizzazione, 23
- logico
- – globale, 23
- – utente, 22
- localhost*, 267, 310
- locking*, 128, 132
- Logging/Recovery Manager* (LRM), 68
- login*, 382, 393
- logout*, 385, 393
- LRM (*Logging/Recovery Manager*), 68

**M**

- mapping*, 94
- metodi, 327, 330, 351
- statici, 335
- middleware*, 149, 178
- MIME (*Multipurpose Internet Mail Extension*), 373
- modellazione concettuale, 33
- modello relazionale, 41, 79
- Multipurpose Internet Mail Extension* (MIME), 373
- mysql*, 359, 392
- MySQL, 94
- mysql improved* (*mysqli*), 365, 392
- mysqli* (*mysql improved*), 365, 392

**N**

- 1NF (*1st Normal Form*), 49
- 2NF (*2nd Normal Form*), 49
- 3NF (*3rd Normal Form*), 49
- 1st Normal Form* (1NF), 49
- 2nd Normal Form* (2NF), 49
- 3rd Normal Form* (3NF), 49
- Notepad++, 192

**O**

- ODBC (*Open DataBase Connectivity*), 147, 177, 388, 393
- oggetto/i, 351
- uguaglianza, 351
- Open DataBase Connectivity* (ODBC), 147, 177, 388
- operatore/i
- @, 390
- di differenza, 112
- di intersezione, 112
- di unione, 112
- relazionali, 69
- derivati, 70, 81
- – congiunzione, 70, 74, 81
- – divisione, 70, 75, 81
- – *equi-join*, 75
- – intersezione, 70, 73, 81
- – *join*, 70, 75, 81
- – primitivi, 70, 81
- – differenza, 70, 71, 81



- – – prodotto cartesiano, 70, 72, 81
  - – – proiezione, 70, 73, 81
  - – – restrizione, 70, 81
  - – – ridenominazione, 70, 73, 81
  - – – selezione, 70, 72, 81
  - – – unione, 70, 71, 81
  - outer join*, 132
  - overriding*, 352
- P**
- pagine web, 286
  - colori, 301
  - date, 301
  - dinamiche, 267, 310
  - indirizzi
  - di posta elettronica, 300
  - web, 300
  - input
  - campi di
  - – attributi, 301
  - – *autocomplete*, 303
  - – *autofocus*, 301
  - – *form*, 302
  - – *multiple*, 304
  - – *placeholder*, 302
  - – *required*, 303
  - – speciali, 300
  - – *title*, 303
  - *form* di, 287, 311
  - gestione, 299
  - validazione degli, 299, 286
  - interazione tra, 286, 311
  - intervalli numerici, 300
  - numeri, 300
  - telefonici, 301
  - passaggio di dati, 286
  - password, 301
  - statiche, 267, 310
  - parsing*, 236
  - DOM, 237, 254
  - SAX, 237, 253
  - password, 301, 380
  - PDO (PHP Data Objects), 388, 393
  - accesso a DBMS, 388
  - indipendenza dal DBMS, 388
  - PDOException*, 388, 393
  - PDOStatement*, 388, 393
  - persistenza, 80
  - PHP (PHP Hypertext Preprocessor), 264, 310
  - *abstract*, 347
  - accesso a un DBMS, 357
  - *array*, 275, 311
  - correlati al web, 278, 311
  - attributi statici, 335
  - classi, 327
  - astratte, 342, 347
  - *clone*, 335
  - *const*, 338
  - controllo
  - degli errori, 392
  - degli input, 393
  - *cookies*, 306
  - copia di oggetti, 334
  - costanti, 275, 311
  - *define*, 311
  - *die*, 271, 385
  - *echo*, 265, 271, 310
  - ereditarietà, 342
  - *exit*, 271
  - *extends*, 345
  - file inviato dal browser al server, 372
  - *foreach*, 277, 311
  - *form* HTML, 286
  - gestione di, 286
  - *function*, 282, 311
  - funzioni, 279, 311
  - definite dall'utente, 282
  - passaggio di parametri, 284, 311
  - per *array*, 280
  - per data e ora, 281
  - per stringhe, 282
  - per variabili, 280
  - gestione delle eccezioni, 348
  - *global*, 285, 311
  - *header*, 299, 311
  - *include*, 269, 310
  - inclusione
  - di file, 310
  - univoca di codice, 269
  - linguaggio, 264
  - metodi
  - *magic*, 338, 352
  - statici, 335
  - *mysql*, 359
  - *mysqli*, 365
  - *new*, 327
  - oggetti, 327
  - operatore @, 281, 392
  - *output*, 310
  - *parent*, 345
  - *print*, 271, 310
  - *private*, 327
  - programmazione a oggetti, 325
  - *public*, 327
  - *require*, 269, 310
  - *return*, 282
  - *script*, 264, 310
  - *self*, 335
  - sessioni, 306, 307, 312
  - sintassi, 269
  - strutture di controllo del flusso, 269
  - *throw new*, 349
  - tipi di dato, 273
  - *try...catch*, 349
  - uguaglianza di oggetti, 334
  - variabili, 272, 285, 310, 311
  - dinamiche, 272
  - *lifetime*, 285
  - *scope*, 311
  - visibilità, 285, 311
  - PHP Data Objects (PDO), 388
  - PK (Primary Key), 37
  - prepared statement*, 178
  - prima forma normale (1NF), 52, 80
  - Primary Key (PK), 37
  - progettazione
  - concettuale, 9
  - di un sistema informatico, 10
  - fisica, 10
  - logica, 10
  - proprietà, 327
- Q**
- QBE (Query By Example), 66
  - query*
  - nidificate, 100
  - scalare, 110, 132
  - SQL, 152, 178
  - Query By Example (QBE), 66
- R**
- raggruppamento, clausola di, 107
  - RDBMS (Relational Data Base Management System), 47
  - Read*, 157
  - record*, 14, 24
  - campo, 14
  - *field*, 14
  - registrazione, 14, 24
  - Relational Data Base Management System (RDBMS), 47
  - relazione, 42, 79
  - con attributi, 43
  - rollback*, 68, 80
  - RowSet*, 167, 179
  - *afterLast*, 169
  - *beforeFirst*, 169
  - *CachedRowSet*, 170
  - *execute*, 168
  - *FilteredRowSet*, 170
  - *first*, 169
  - interfaccia, 167
  - *JdbcRowSet*, 170
  - *last*, 169
  - metodi, 168, 169
  - *next*, 169
  - notifica degli eventi, 175
  - oggetti, 167, 170, 179
  - connessi, 170
  - disconnessi, 170
  - *previous*, 169
  - *setCommand*, 168
  - *WebRowSet*, 170
  - modalità di *parsing*, 237, 253
  - schema, 43
  - scripting*
  - *client-side*, 267
  - *server-side*, 267, 357, 392
  - seconda forma normale (2NF), 53, 80
  - self-join*, 100, 132
  - server, 266, 310
  - web, 266, 310
  - sessioni, 393
  - Simple API for XML (SAX), 237, 253
  - sistema/i
  - informatico/i, 2, 5, 6, 23
  - analisi
  - dell'esistente, 8
  - preliminare, 8
  - ciclo di vita, 6, 23
  - raccolta delle richieste degli utenti, 23
  - realizzazione, 23
  - componenti
  - applicative, 10
  - infrastrutturali, 10
  - dati, 3, 14
  - informazione, 3
  - nuovi requisiti, 8
  - progettazione, 10
  - concettuale, 9, 23
  - fisica, 10, 23
  - logica, 10, 23
  - realizzazione, 10
  - richieste degli utenti, 8
  - informativo/i, 2, 5, 6, 23
  - dati, 3, 12
  - aspetto estensionale, 12
  - aspetto intensionale, 12
  - file, 14
  - informazione, 3
  - scopo
  - decisionale, 5
  - operativo, 5
  - software, crisi del, 6
  - SQL (Structured Query Language), 65, 91, 131
  - accesso concorrente ai dati, 128, 132
  - ALL, 106, 131
  - ALTER, 114, 123, 132
  - ALTER TABLE, 122, 123
  - ANY, 106, 131
  - AS, 99, 102, 131
  - ASC, 97, 131
  - BETWEEN, 98, 131
  - BINARY, 167
  - BIT, 116, 167
  - CHAR, 166
  - chiusura, 100
  - clausola di raggruppamento, 107
  - comandi DDL, 114
  - comandi DML, 124
  - COMMIT, 129, 133
- S**
- savepoint*, 129
  - SAX (Simple API for XML), 237, 253



- CONSTRAINT, 115
  - CREATE, 114, 121, 132
  - CREATE DATABASE, 121
  - CREATE INDEX, 121
  - CREATE TABLE, 121
  - CREATE TRIGGER, 127
  - CREATE VIEW, 121
  - data, 117
  - DATE, 117, 167
  - DATETIME, 117
  - dati numerici, 116
  - DB-schema, 121, 132
  - DECIMAL, 166
  - DEFAULT, 115
  - DELETE, 124, 125, 126, 132
  - DELETE-FROM-WHERE, 125
  - DESC, 97, 131
  - DISTINCT, 97, 131
  - DOUBLE, 116, 167
  - DROP, 114, 123, 124, 132
  - *embedded*, 392
  - EXCEPT, 113, 114, 132
  - EXIST, 106
  - EXIT, 131
  - FLOAT, 116, 167
  - FOR EACH ROW, 127
  - FOR UPDATE, 130
  - FOREIGN KEY, 115, 123
  - FROM, 101, 132
  - funzioni di aggregazione, 107, 132
  - GLOBAL, 130
  - GRANT, 123
  - GROUP BY, 108, 109, 132
  - HAVING, 109, 132
  - HAVING COUNT, 110
  - IN, 131
  - *injection*, 375
  - INNER JOIN, 102
  - *inner join*, 132
  - INSERT, 124, 126, 132
  - INTEGER, 167
  - integrità referenziale, 121
  - INTERSECT, 113, 132
  - intersezione, 132
  - INTO, 99
  - *join*, 100, 131
  - LEFT JOIN, 104, 132
  - LEFT OUTER JOIN, 104, 132
  - LIKE, 97, 131
  - *lite*, 20
  - livello di isolamento, 130
  - LOCAL, 129
  - LOCK IN SHARE MODE, 130
  - LOCK TABLES, 128
  - *locking*, 128, 132
  - LOW\_PRIORITY, 129
  - *mapping*, 94
  - NEW, 127
  - NOT IN, 131
  - NOT NULL, 115
  - NULL, 96
  - NUMERIC, 166
  - OLD, 127
  - ON DELETE CASCADE, 123
  - ON UPDATE CASCADE, 123
  - operatori
    - di differenza, 112
    - di intersezione, 112
    - di unione, 112
  - orari, 117
  - ORDER BY, 131
  - *outer join*, 132
  - PRIMARY KEY, 115
  - *query*, 152, 178
  - – nidificate, 100
  - – scalare, 110, 132
  - REAL, 167
  - REFERENCES, 115
  - RENAME, 123, 124
  - RENAME TABLE, 124
  - REVOKE, 123
  - RIGHT JOIN, 104, 132
  - RIGHT OUTER JOIN, 104, 132
  - ROLLBACK, 129, 133
  - ROLLBACK TO SAVEPOINT, 130
  - *savepoint*, 129
  - SELECT, 94, 95, 131, 132
  - SELECT-FROM-WHERE, 94, 131
  - *self-join*, 100, 132
  - SESSION, 130
  - SET AUTOCOMMIT, 129
  - START TRANSACTION, 129
  - stringhe di caratteri, 120
  - *subquery*, 100, 131
  - – correlate, 106, 131
  - TIME, 167
  - TIMESTAMP, 167
  - tipi, 157
    - corrispondenza con tipi
      - Java, 157, 166
      - di dato, 115
    - transazioni, 129, 133
    - *trigger*, 126, 132
    - UNION, 112, 132
    - unione, 132
    - UNIQUE, 121
    - UNLOCK TABLES, 129
    - UNSIGNED, 116
    - UPDATE, 124, 126, 132
    - UPDATE-SET-WHERE, 126
    - utilizzo, 92
    - VARBINARY, 167
    - VARCHAR, 166
    - vista, 121, 132
    - WHERE, 96, 100, 131
    - ZEROFILL, 116
  - standard relazionale, 47
  - StAX (*Streaming API for XML*), 241
  - stored procedure*, 126
  - Streaming API for XML* (StAX), 241
  - stringhe di caratteri, 120
  - Structured Query Language* (SQL), 65
  - subquery*, 100, 131
  - correlata, 131, 106
  - superchiave, 43, 79
- T**
- tabelle, 43
    - compatibili, 71
  - terza forma normale (3NF), 54, 80
  - timestamp*, 384
  - Transaction Processing*, 68
  - transazioni, 67, 80, 129, 133
    - atomicità (*atomicity*), 69
    - consistenza (*consistency*), 69
    - isolamento (*isolation*), 69
    - persistenza (*durability*), 69
  - trigger*, 126, 132
- U**
- Uniform Resource Identifier* (URI), 279
  - Uniform Resource Locator* (URL), 279
  - unione, 132
  - Update*, 157
  - URI (*Uniform Resource Identifier*), 279
  - URL (*Uniform Resource Locator*), 279
  - username*, 380
  - utente/i, 67
  - richieste degli, 8
- V**
- vista, 121, 132
- W**
- W3C (*World Wide Web Consortium*), 184
  - World Wide Web Consortium* (W3C), 184
- X**
- XHTML, 185
  - XML (*Extensible Markup Language*), 184, 215, 253
    - attributi, 192
    - documento/i, 224, 253
      - ben formato, 190, 215
      - creazione con DOM, 251
      - gestione con API Java, 236
      - *parsing*, 253
      - – con DOM, 244
    - – con SAX, 241
    - – con StAX, 241
    - struttura ad albero, 185, 215
    - validazione
      - con Notepad++, 192
      - rispetto a uno schema XSD, 240
      - valido, 192
      - visualizzazione, 189
    - elemento/i, 186, 215
      - complessi, 197, 215
      - semplici, 192
    - espressioni regolari, 194
    - linguaggi per applicazioni specifiche, 190
    - pattern, 194
    - schemi
      - estendibili, 202
      - XSD, 190
      - – definizione di linguaggi, 190
      - sintassi, 185
      - strumento/i per la gestione dei dati, 224
      - tipi di dato predefiniti, 203
    - XPath
      - riferimento ai nodi di un albero, 208
    - XQuery
      - costruzione di elementi, 228
      - interrogazione di basi di dati, 226
      - trasformazione di documenti, 228
  - XML Schema Definition (XSD), 215
  - XPath, 208
    - espressioni, 215
    - predicati, 215
    - riferimento ai nodi di un albero XML, 208
    - valutazione di espressioni con Notepad++, 208
  - XQJ (*XQuery API for Java*), 236
  - XQuery, 226, 253
    - espressioni condizionali, 253
    - file, 227
    - funzioni definite dell'utente, 229
    - interrogazioni, 253
    - *join*, 253
    - raggruppamento, 253
  - XQuery API for Java (XQJ), 236
  - XSD (*XML Schema Definition*), 215
    - schema, 191
    - tipi derivati, 215
    - tipi predefiniti, 215

Fiorenzo Formichi Giorgio Meini

# Corso di informatica

per Informatica

Basi di dati relazionali e linguaggio SQL

Linguaggio XML

Pagine web dinamiche con linguaggio PHP

Un corso di informatica focalizzato sulla pratica di laboratorio, mediante esempi graduali e attinenti ad aspetti professionalmente rilevanti. I contenuti proposti e gli esempi applicativi sono illustrati attraverso i linguaggi di programmazione più diffusi nella pratica professionale.



## Nel libro

- Il testo è diviso in **due sezioni**: la prima dedicata a un tema fondamentale (*Basi di dati relazionali e linguaggio SQL. Linguaggio XML*), la seconda a una tematica web (*Pagine web dinamiche con linguaggio PHP*).
- Una **sintesi di fine capitolo** introduce i quesiti e gli esercizi.
- **Brani in inglese**, tratti da testi di riferimento della disciplina, sono accompagnati da un glossario.

## Idee per il tuo futuro

Stai per finire la scuola,  
che cosa fare adesso?

- Vale la pena laurearsi?
- Che cosa chiedono ai test di ammissione?
- E se volessi studiare all'estero?
- Come si scrive un curriculum?

In questo libro, otto pagine  
per l'orientamento e nel sito

[www.ideeperiltuofuturo.it](http://www.ideeperiltuofuturo.it)

dati, informazioni e consigli  
sull'università e il mondo  
del lavoro.

L'accesso alle risorse digitali  
protette è personale e non  
condivisibile.