



Capitolo 12

Scegliere un database per PHP

- 12.1 **Cos'è un database?**
- 12.2 **Perché un database?**
- 12.3 **Scegliere un database**
- 12.4 **Funzioni avanzate**
- 12.5 **Database supportati da PHP**
- 12.6 **Astrazione del database (o no)**
- 12.7 **L'obiettivo: MySQL**
- 12.8 **Sommario**

I database e PHP si sposano come torta e gelato, Trinidad e Tobago, uova fresche e prosciutto, tanto per rendere l'idea.

Dopo tutto, cos'è il Web? La creazione di vasti archivi di informazione disponibili a un pubblico più o meno vasto. Non che non ci sia abbondanza di piccoli siti brochureware; ma più è grande e più spesso viene aggiornata la sorgente dei dati, più il valore comparativo viene fornito dal Web rispetto agli altri mezzi di comunicazione.

Forse il vantaggio maggiore di PHP rispetto a prodotti simili è la scelta insuperata e la facilità di connessione ai database che offre. Come descritto nel paragrafo "Scegliere un database" di questo capitolo, PHP supporta connessioni native ai database più popolari, open source e prodotti commerciali simili. Alla fine è possibile includere quasi tutti i database che apriranno la loro API al pubblico. Per qualsiasi database non supportato, c'è il supporto ODBC generico.

12.1 Cos'è un database?

Un *database* è un'applicazione separata che memorizza una raccolta di dati. Ciascun database possiede una o più API distinte per creare, accedere, gestire, cercare, e replicare i dati che contiene. È possibile utilizzare altri tipi di memorizzazione dei dati, come i file nel filesystem o ampie tabelle hash in memoria, ma quando si parla di database si intende un'applicazione autonoma come Oracle o SQL Server o Sleepycat.



12.2 Perché un database?

Se si decide di utilizzare PHP, probabilmente presto o tardi servirà un database. Anche per qualcosa di piccolo, come un Weblog personale, occorre pensare seriamente ai vantaggi che offre l'utilizzo di un database invece delle pagine statiche o dei file di testo inclusi.

Mantenimento e scalabilità

Avere PHP che assembla le pagine in tempo reale da un modello e un database è un'esperienza unica. Una volta provata, non si tornerà mai più a gestire un sito HTML statico di qualsiasi dimensione. Con lo sforzo di programmazione di una pagina, si possono produrre un numero infinito di pagine uniformi. Se ne cambia una e tutte le altre vengono modificate di conseguenza.

Oggi esistono siti Web con centinaia di migliaia di pagine separate (sicuramente non c'è nessuno che le gestisce tutte a mano). Se si possiede un concetto Web che potrebbe svilupparsi in più di alcune dozzine di pagine, si dovrebbe pensare di passare all'utilizzo dei database il più presto possibile.

Portabilità

Poiché un database è un'applicazione e non una parte del sistema operativo, è possibile trasferire facilmente la sua struttura e i contenuti da una macchina a un'altra, o (in certi casi) anche da una piattaforma a un'altra. Questo è particolarmente importante per gli imprenditori, che possono sviluppare un progetto senza possibilità di controllo dell'ambiente nel quale verrà inserito: possono consegnare un pacchetto di PHP più lo schema del database MySQL copiato in un tarball o file zip. Esistono anche programmi PHP molto conosciuti, come vBulletin, che mantengono buona parte del codice in un database per renderlo più facilmente distribuibile.

Evitare programmazioni difficili

Certe cose che si possono eseguire con PHP non andrebbero fatte, perché implicano mosse di programmazione sgradevoli o rischiose.

Si supponga di essere il comandante della nave spaziale *Enterprise* e di mantenere il diario del Capitano. Ogni episodio è contenuto in un file di testo identificato da una data spaziale unica, inserito da PHP in un modello; il comandante è un uomo dello spazio molto impegnato con intere galassie da esplorare; non c'è il tempo di scrivere il diario ogni giorno. Si desidera inserire su ogni pagina i collegamenti *Avanti* e *Indietro* generati automaticamente per chi vuole leggere il diario in ordine cronologico. È abbastanza facile utilizzare PHP per trovare l'inserimento precedente, ma qualsiasi sforzo per localizzare

quello successivo può diventare velocemente un ciclo infinito, perché è più facile provare qualcosa che esiste rispetto a qualcosa che non esiste. Per contro, se si inseriscono i dati di log in un database, l'intero lavoro diventerà banale. Il database indicherà qual è l'ultimo inserimento in ogni dato momento.

Esistono altri tipi di lavori di programmazione che un database è altamente ottimizzato a compiere, e dandogliene l'occasione è poi possibile sfruttarli. Per esempio, non si devono mai ordinare i dati impostati sul lato PHP, ma occorre imparare a scrivere le proprie query in modo che ritornino preordinate. Questi problemi di efficienza saranno descritti più dettagliatamente nel Capitolo 18.

Ricerca

Sebbene per le stringhe sia possibile cercare file di testo multipli (specialmente sulle piattaforme UNIX), non è quello che molti sviluppatori Web vorrebbero fare spesso. Dopo la ricerca su un centinaio di file, il compito diventa lento e difficile da gestire. I database esistono per facilitare le ricerche. Con un unico comando si può trovare qualsiasi cosa, da un numero di identificazione, a un ampio blocco di testo, a un'immagine in formato JPEG.

In alcuni casi, l'informazione ha valore solo quando viene inserita in un database di ricerca. Per esempio, poche persone vorrebbero leggere un lungo elenco di testo riguardante i registi e i loro film; ma occasionalmente molte desiderano eseguire una ricerca in un database a caccia di tali informazioni. Qui si può obiettare che è la ricerca, tanto quanto l'informazione stessa, che crea il valore.

Sicurezza

Un database, se utilizzato con la sue password, offre un ulteriore livello di sicurezza.

Si supponga di utilizzare PHP per gestire i file dei clienti di una società, contenenti informazioni sui prezzi pagati da ciascun cliente per i prodotti e i reclami eseguiti. Queste informazioni sarebbero oro per la concorrenza e sarebbe imbarazzante se fossero disponibili a tutti; ma devono essere inserite in Internet così da essere accessibili al proprio esercito di venditori. Se PHP scrive ogni nuovo record del cliente in un file di testo, occorre fornire all'utente del server HTTP l'accesso in scrittura alla directory più sensibile (di solito nessuno o tutti). Non è una buona idea. Facendo invece in modo che PHP scriva in un database, si possono mantenere i permessi di accesso alle directory in sola lettura, e anche richiedere una seconda password prima che il database possa essere modificato.

Oppure si consideri il caso di un sito di contenuto con un vasto numero di visitatori, un numero più ridotto di scrittori e una manciata di editori. Si possono impostare facilmente i livelli di permesso di accesso al database

per ogni gruppo, in modo che i visitatori possano solo leggere il contenuto del database (formattato nelle pagine Web), gli scrittori possano sfogliare e modificare solo i loro inserimenti, e gli editori possono sfogliare/modificare/cancellare qualsiasi cosa nel sito.

Architettura multilivello

Finora sono stati considerati solo i siti cosiddetti *a due tier*: PHP prende dati puri da alcuni tipi di sistemi di memorizzazione e li trasforma in HTML. Tuttavia, uno dei propositi di PHP è diventare il “collante” nello sviluppo a tre o più tier. Se si possiede qualcosa di più complesso della semplice architettura a due tier, allora si ha *veramente* bisogno di un database.

Un’architettura a diversi tier è un numero arbitrario di sotto sistemi di software collegati da un sito Web sul front end e uno o più database sul back end. Un’architettura a diversi tier abbastanza comune è quella di un grande sito di e-commerce, con carrelli della spesa collegati a sistemi di ricevimento ordini collegati a procedure di gestione delle catene di rifornimento, più database dei prodotti e dei clienti, addebiti delle carte di credito, FAQ, motori di consigli, strumenti di analisi delle registrazioni Web, memorizzazione di proxy e chissà cos’altro si nasconde dietro le quinte. In queste condizioni, è necessario disporre di caratteristiche avanzate di database, come descritto nel paragrafo “Funzioni avanzate” di questo capitolo.

Il potenziale rovescio della medaglia: le prestazioni

Si potrebbe essere preoccupati delle prestazioni. È vero che un sito basato su un database sullo stesso hardware e software sarà sempre più lento di un sito statico. È vero che certi database sono più veloci di altri. Tuttavia, la questione reale è se i margini di prestazione possano essere mai rilevati. Se si sta parlando di aggiungere millesimi di secondi alla propria latenza, a chi importa? Alcune delle preoccupazioni riguardanti le prestazioni che si leggono su Internet sono così gonfiate che rasentano l’assurdo.

Una volta che si necessita di entrare nel database anche per *una sola parte* di informazione per pagina, è quasi sempre vantaggioso inserire tutti i dati nel database. La maggioranza delle informazioni addizionali di una query al database viene caricata anticipatamente quando si stabilisce una connessione. Se si dovesse eseguire questa operazione per ottenere un nome o un titolo, il download di qualche migliaio di parole di testo è praticamente gratuito.

In conclusione, solamente i test delle prestazioni potranno stabilire se un database è troppo lento per le proprie impostazioni e operazioni, tutto il resto sono solo chiacchiere. Molti siti Web completamente basati su database ottengono facilmente sotto secondi di latenza, che dovrebbero essere sufficienti per la maggioranza degli scopi.

12.3 Scegliere un database

Sebbene i database (anche quelli relazionali) esistano da lungo tempo, fino a poco tempo fa erano abbastanza costosi o limitati nelle funzionalità. Di conseguenza, anche molti programmatori esperti non hanno mai dovuto imparare molto su come scegliere un database per una particolare esigenza. Per questo motivo, vale la pena esaminare gli elementi fondamentali che portano a questa decisione.

Si potrebbe non avere scelta

Realisticamente, si potrebbe non avere molta scelta. Molte persone stanno cercando specificatamente il modo più veloce per mettere online i loro vecchi database, invece di avere il lusso di decidere prima il linguaggio di scripting e poi di scegliere un database.

Inoltre, le scelte di SO, server Web e linguaggi di programmazione possono prendere automaticamente alcune decisioni. Un'applicazione Java personalizzata su una piattaforma UNIX "Big Iron" non andrà molto d'accordo con Microsoft SQL Server (in teoria è possibile, ma in pratica bisogna essere un masochista, anche se l'altra strada non è così male). Più grande diventa il sistema, più le scelte saranno limitate da decisioni precedenti.

La buona notizia è che PHP si impegna a supportare molti database e altri server di back-end. Può aiutare unire le estremità slegate di un'architettura cresciuta organicamente nel tempo, come hanno fatto molte. In PHP alcune funzioni esistono esclusivamente per aiutare ad allineare i dati in un database più moderno.

Flat file, relazionale, relazionale a oggetti

I database sono come gli strumenti da cucina: più lo strumento è semplice, più l'operatore deve essere bravo per ottenere un buon risultato. I cuochi esperti possono creare cibi deliziosi utilizzando solo un coltello molto affilato e poche e vecchie pentole; mentre per raggiungere gli stessi risultati i dilettanti devono tirare fuori sofisticati attrezzi ed elettrodomestici.

La stessa cosa vale per i database. Può diventare abbastanza ridicolo leggere le discussioni delle persone sui difetti impliciti di questo o di quel database, sapendo che l'esperienza del singolo utente viene riflessa da questo software più che da qualsiasi altro. Basta questo per dire che molti capolavori tecnici si trovano nelle tabelle hash più semplici, mentre i disordini segreti eseguiti in malo modo ribollono sui più recenti e grandi DBMS abilitati a Java e orientati agli oggetti.

Con PHP si possono utilizzare tre tipi principali di database: *flat file*, *relazionali* e *relazionali a oggetti*.

I flat file o database *di hash*, come Gnu DBM e Berkeley DB (noto come Sleepycat DB2), vengono utilizzati soprattutto da o in altri programmi come i server di posta elettronica. Forniscono il mezzo più leggero e veloce per la memorizzazione e la ricerca di dati come le coppie nome utente/password o i messaggi di posta elettronica datati. I programmatori C di vecchia scuola solitamente hanno la maggior esperienza con questo tipo di database.

Questi database non creano da soli una rappresentazione di relazioni più complesse tra punti di dati. Questa operazione viene invece eseguita dal programma client di accesso. Sebbene i risultati possano essere estremamente impressionanti, dipende tutto dall'esperienza del programmatore.

Oggi la varietà relazionale è il tipo di database più comune. Le persone hanno idee diverse su ciò che costituisce un database relazionale, e quindi non si scenderà nei particolari. Di conseguenza, si affermerà arbitrariamente che i database che parlano un SQL scorrevole sono relazionali. La maggior parte dei database popolari utilizzata comunemente con PHP è relazionale. Per una descrizione più dettagliata dei database relazionali consultare il Capitolo 13.

Ma ci sono relazionali e relazionali. Alcuni database commerciali molto popolari, come Filemaker Pro e Microsoft Access, non sono stati progettati per essere utilizzati sul back end di un sito Web di produzione. Nonostante abbiano un livello di supporto ODBC abbastanza elevato, e di conseguenza PHP può in teoria ottenere i dati da essi, sono stati progettati principalmente per la facilità di utilizzo più che per la velocità. Ancora peggio, la maggior parte degli utenti di questi prodotti si rifiuta di servirsi di quello che sono le funzioni relazionali, preferendo ripetere informazioni di testo in ogni elemento invece di creare una tabella separata che rappresenta una relazione. Infine, questi database di solito non dispongono dei thread, dei blocchi e di altre caratteristiche di produzione. Ci sono alcune persone che provano a utilizzare Microsoft Access con PHP, poiché lo hanno scritto nei forum e nelle mailing list PHP, ma chiaramente non per siti pubblici con un traffico rilevante. Tuttavia, gli autori conoscono sviluppatori che utilizzano Access o Filemaker Pro come strumenti di sviluppo sui loro portatili, così da poter programmare sull'aereo, e ci sono sempre progetti che utilizzano i dati legacy da questi database semi relazionali. Si può scoprire che l'impiego migliore di questi tipi di database sarà quello di realizzare un prototipo delle loro controparti Web. Nonostante tutti i difetti di Access, molti sviluppatori sostengono che dispone di buone funzioni di visualizzazione di dati/relazione.

Infine, ci sono i database orientati agli oggetti e quelli relazionali a oggetti, modelli di accesso dei dati nuovi e ancora in fase di sviluppo. Il database orientato agli oggetti è progettato per lavorare più uniformemente con i linguaggi di programmazione orientati agli oggetti; mentre il relazionale a oggetti è un ibrido utilizzato per i tipi di dati (come i dati astronomici e genetici) che non vengono serviti bene dai normali database relazionali. Tuttavia, PHP stesso non richiede uno stile di programmazione a oggetti né ai suoi utenti né ai programmi con i quali comunica. E nonostante in PHP5 sia presente una vasta

quantità di nuove funzioni a oggetti, gli sviluppatori riconosceranno comunque che la potenza del linguaggio si trova nelle root procedurali più semplici. Questo non significa che non è possibile utilizzare PHP con alcune di esse, ma la necessità assoluta di comportarsi in questo modo è discutibile.

ODBC/JDBC e API nativa a confronto

Esistono due API generiche standard di accesso al database: *Open Database Connectivity* (ODBC) e *Java Database Connectivity* (JDBC). ODBC è strettamente collegata a Microsoft e JDBC è ancora più saldamente legata a Sun Microsystems. Tuttavia, altre società hanno implementato questi standard nei loro prodotti, con l'aggiunta di driver specifici per ciascun programma client.

ODBC e JDBC sono più o meno mutuamente esclusive. Una cosa chiamata “*ponte ODBC-JDBC*” viene utilizzata per permettere ai programmi Java di accedere ai database ODBC, ma è molto lenta. Esistono anche driver proprietari che eseguono lo stesso lavoro più velocemente.

Ci sono anche database a cui possono accedere i client attraverso le loro API invece che sfruttando ODBC o JDBC. Questa operazione è sicuramente più veloce poiché ci sono meno livelli nello stack. La maggior parte dei database open source rientra in questa categoria. Alcuni hanno anche i driver ODBC o JDBC. Così per esempio, PHP può accedere a MySQL con una API nativa, mentre un sotto sistema Java può utilizzare lo stesso database con JDBC. Prima di realizzare un qualsiasi schema di accesso multiplo, occorre assicurarsi che i driver di cui si ha bisogno siano disponibili e che possano essere acquistati e gestiti.

Database scambiabili

Sebbene ODBC sia più lento delle API native, ha il vantaggio di essere uno standard aperto. Di conseguenza, il codice PHP scritto con i comandi ODBC funzionerà con qualsiasi database compatibile con ODBC. Questa funzione è molto utile se si deve iniziare un progetto con un database che non scala, come Microsoft Access, e in seguito si deve passare a un database più potente dal punto di vista industriale. Sebbene entrambi siano ottimi prodotti nelle loro categorie, può diventare un lavoro lungo passare da un database leggero come Mini SQL (noto come mSQL) a una suite per server pronta come DB2 di IBM (di nuovo, un buon programmatore a cui vengono dati il tempo e le risorse può realizzare qualsiasi applicazione relativamente facile da portare, mentre uno sviluppatore inesperto o precipitoso può non essere in grado di farlo).

12.4 Funzioni avanzate

Questo paragrafo descrive le funzioni di database SQL specifiche con le quali si può non avere ancora familiarità. Si spera che sia possibile comprendere, anche da una descrizione così veloce, se si ha veramente bisogno di una o più di queste funzioni.

GUI

I database variano enormemente dal punto di vista degli strumenti di interfaccia utente. Le scelte spaziano dalle più rigide interazioni a riga di comando ai massicci kit di strumenti di sviluppo forniti da Java. Si paga per ciò che si desidera, sia in contanti che in prestazioni. Cercare l'interfaccia più chiara che soddisfi le proprie esigenze, poiché una GUI può aumentare enormemente i costi.

Un'alternativa a basso costo alla GUI incorporata è un'interfaccia Web. Queste spesso vengono personalizzate, ma ci sono anche prodotti di terze parti che possono soddisfare le proprie esigenze. Per esempio, MySQL possiede diverse interfacce basate sul Web disponibili gratuitamente, che si possono trovare in Freshmeat (www.freshmeat.net) o SourceForge (<http://sf.net>). La più famosa è probabilmente PHPMyAdmin, disponibile sul sito Web di PHPMyAdmin (www.phpmyadmin.org).

Subquery

Una *subquery* o *sotto selezione* è una dichiarazione SELECT inclusa, come questa:

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);
```

Esistono modi per aggirare la mancanza di sotto selezioni, e non tutti ne hanno bisogno. Tuttavia, possono far risparmiare un po' di tempo se la necessità di eseguire grandi selezioni, inserimenti e cancellazioni è alta.

SELECT INTO

SELECT INTO è una comoda funzione se si desidera spostare frequentemente i dati da una tabella all'altra. La sintassi può variare leggermente. Un metodo è:

```
SELECT INTO table2(col2, col3, col7) lastname, firstname, state FROM table1
WHERE col5 = NULL;
```

Un altro modo per ottenere lo stesso risultato è:

```
INSERT INTO table2(col2, col3, col7) SELECT lastname, firstname, state FROM
table1 WHERE col5 = NULL;
```

Unioni complesse

Un'*unione* è un modo di cercare qualcosa attraverso le tabelle utilizzando valori condivisi per accoppiare le tabelle. La forma più semplice è:

```
SELECT * FROM table1,table2 WHERE table1.id=table2.id;
```

Questo produce i contenuti completi di qualsiasi riga nelle due tabelle che condividono i numeri di ID. Esistono altri tipi di unioni più specifici ed estesi, compresi sinistra o destra, diritto o incrociato, interno ed esterno, ma sono poco utilizzati.

Le unioni sono molto pratiche e fanno risparmiare tempo, a volte sono quasi essenziali, ma in pratica pochi hanno veramente bisogno di modelli più esoterici. Quindi, non si deve rifiutare un database incontrollabile perché manca la giusta unione esterna.

Thread e blocchi

I *thread* e i *blocchi* sono molto importanti per siti multi tier e a due tier con molti collaboratori. Impediscono che due chiamate al database si scontrino una con l'altra, fornendo il controllo editoriale solo a una singola transazione alla volta.

Un esempio che spiega chiaramente il valore dei thread e dei blocchi è un sito Web che vende biglietti per concerti rock. Ovviamente, non si desidera che due persone acquistino lo stesso posto per la stessa manifestazione a causa di un errore del database. Il database ha bisogno di alcuni modi per riconoscere le richieste uniche e permettere che solo un utente (o thread) esegua le modifiche in un qualsiasi momento stabilito, mentre gli altri vengono bloccati finché non è stata completata la prima transazione.

A meno che non si sia sicuri che il progetto (per esempio, un Web log) avrà un solo utente alla volta, si faccia attenzione nell'affidarsi a un database senza thread.

Database transazionali

Questo termine si riferisce a un progetto di database che cerca di massimizzare l'integrità dei dati. Il paradigma *transazionale* conta su assegnazioni e rollback. Le transazioni concluse con successo verranno assegnate al database. Quelle che non hanno avuto successo non verranno salvate, o il database verrà riportato alla sua condizione precedente.

In generale, le transazioni diventano più utili in situazioni in cui si desidera venga assegnato tutto o niente a un gruppo di inserimenti. Un sistema di e-commerce può utilizzare i rollback in situazioni dove viene rifiutata la carta di credito di un cliente, scegliendo di non registrare le informazioni sul

cliente, l'ordine d'acquisto, la modifica dell'inventario e così via. I rollback sono anche utili in caso di alterazione dei dati, come quando un server di database subisce un blocco hardware.

Un progetto alternativo di integrità dei dati viene chiamato *atomico*. I sostenitori del paradigma atomico dichiarano che è molto più veloce e ugualmente sicuro, ma le transazioni possono risultare più semplici per numerosi programmatori, poiché c'è più logica a livello del database.

Procedure e trigger

Le *procedure* sono query o routine memorizzate e precompilate sul server di database. Una comune procedura è quella che seleziona tutti gli indirizzi di posta elettronica dei clienti che hanno fatto acquisti un particolare giorno. Se si utilizzano le stesse dichiarazioni di selezione più volte, le procedure possono raccogliere in modo pratico e veloce.

I *trigger* sono procedure che si verificano quando alcuni eventi vengono registrati dal database. A seconda del database, si può scrivere un trigger per inviare un messaggio di posta elettronica con l'estratto conto ai clienti o ai soci del sito, e impostarlo perché il messaggio venga spedito ogni domenica a mezzanotte. Un altro utilizzo pratico consiste nello spedire un messaggio di posta elettronica all'amministratore del database ogni volta che viene registrato un errore. Relativamente pochi database utilizzano i trigger, poiché sfruttano una grande quantità di energia programmatica e molti cicli supplementari per rintracciare i possibili eventi.

Indici

Gli *indici* rappresentano un modo per velocizzare le ricerche di grandi set di dati. Si supponga di dover trovare il particolare file di un cliente in una vasta pila di file gettati a caso sulla propria scrivania. Oppure si potrebbe cercare il file in un gruppo di raccoglitori sistemati in ordine alfabetico. Ovviamente, il sistema dei file in raccoglitori sarà più veloce, perché manterrà i documenti preordinati in piccole cartelle. In una nutshell questo è ciò che fa un indice.

Se si hanno milioni di utenti sarà estremamente lento per un database trovarne uno in base al cognome, perché il programma dovrà esaminare ciascuna voce presente nel campo del cognome e confrontarla con la stringa che si sta cercando. Un indice posizionato su quella colonna permetterà al database di eseguire la ricerca solo in una parte del set di dati, e questo renderà la ricerca più veloce.

Tuttavia, gli indici non devono essere usati da chiunque. Innanzitutto, rallentano la scrittura e accelerano la lettura. Non necessariamente accelerano ogni tipo di query, e il set di dati potrebbe non essere grande abbastanza per mostrare una differenza di velocità apprezzabile. Gli indici aiuteranno molto

se si possiede un bilione di voci, ma potrebbero non aiutare affatto se le voci sono solo 500.

Chiavi esterne e limitazioni di integrità

La struttura relazionale di un database è spesso implicita nei modi in cui i campi di una tabella si riferiscono agli ID di riga di un'altra, ma il database non farà necessariamente niente di utile per assicurare che la struttura venga rispettata quando si eseguono le modifiche. Un modo in cui il database può essere utile è attraverso le *cancellazioni a cascata*: cancellando automaticamente le righe che dipendono da altre righe che sono state eliminate (questo a volte è implementato come trigger). Per esempio, se si cancella il record di un paziente dell'ospedale, si può desiderare che anche tutte le righe orfane nella tabella corrispondente alle sue visite vengano automaticamente eliminate. In alternativa, un sistema di database può semplicemente impedire la cancellazione delle righe genitore a meno che prima non vengano cancellate le potenziali orfane. Se questo tipo di limitazione è un salvavita o semplicemente una noiosa restrizione dipende da quanto è importante che la struttura relazionale sia del tutto affidabile e coerente, e quanto frequente sia il bisogno di questo tipo di operazioni pericolose. La maggior parte di queste funzioni può essere implementata, sebbene in modo meno chiaro ed efficace, con una combinazione di chiavi tradizionali e il codice client che si utilizza per manipolare i dati.

Replica del database

Quando i dati memorizzati aumentano, è necessario pensare al potenziamento. Per un determinato periodo, è semplicemente possibile spostare il server di database su macchine più veloci con più processori e dischi più capienti, ma prima o poi un database in crescita dovrà essere replicato su più di un server.

Per eseguire questa operazione sono necessari alcuni mezzi che tengono automaticamente sincronizzati i diversi server. Questo normalmente comprende un sistema di registrazione, e spesso una relazione *master-slave* tra i server di database. Un database è il *master*, e in esso vengono inseriti tutti i nuovi dati. Un registro tiene traccia di queste modifiche in ordine cronologico. Tutti gli altri server sono *slave*, che mettono a disposizione i dati invece di conservarli. Leggono periodicamente il registro master ed eseguono le stesse modifiche al loro interno.

Il passo successivo è un tipo di meccanismo *failover*, per il quale uno slave può diventare il server di database master se il master si blocca. Si pensi attentamente al livello di sicurezza che devono avere i dati, poiché è molto costoso.

12.5 Database supportati da PHP

Se prima d'ora non è mai stato selezionato un database, all'inizio la vasta scelta di prodotti supportati da PHP può dare le vertigini. La tabella 12.1 fornisce una prima introduzione ai vari database più facilmente disponibili per gli utenti di PHP, con annotazioni sui driver e sulle licenze.

Tabella 12.1 Database supportati da PHP.

DATABASE	TIPO	SUPPORTO	PIATTAFORMA	LICENZA	NOTE
Adabas D	R	ODBC (disapprovato)	U, W	C	Tedesco, distribuito con SuSE Linux
DBA/DBM	FF	Livello di astrazione	U	OS, C	Sleepycat, Gnu DBM, cdb
dBase	P	Solo importazione	W	C	No SQL
Empress	R	ODBC	U, W	C	Aziendale, disponibile il driver JDBC
filepro	P	Solo importazione	U, W	C	Non per la produzione
IBM DB2	R	ODBC	U, W	C	Aziendale, disponibile il driver JDBC
Informix	R	Nativo	U, W	C	Aziendale
Interbase	R	Nativo	U, W	C	Aziendale, disponibile il driver JDBC
MS Access	R	ODBC	W	C	Non per la produzione
MS SQL Server	R	Nativo	W	C	Aziendale
mSQL	R	Nativo	U	Sh	Molto piccolo
MySQL	R	Nativo	U, W	C, OS	Diverse licenze
Oracle	R	Nativo	U, W	C	Aziendale
Oracle8	R	Nativo	U, W	C	Aziendale, integrazione di Java
PostgreSQL	O-R	Nativo	U	OS	Supporto commerciale disponibile
Solid	R	ODBC (disapprovato)	U, W	C	db incluso, azienda finlandese
Sybase	R	Nativo	U, W	C	Aziendale

FF= Flat file; R = Relazionale; O-R = Relazionale a oggetti; U = Unix; W = Windows, C = Commerciale; OS = Open Source; Sh = Shareware.

12.6 Astrazione del database (o no)

L'astrazione del database, cioè la scrittura di funzioni wrapper o di classi al posto dell'utilizzo di nudi comandi PHP, è uno di quegli argomenti di programmazione quasi religiosi. Alcuni eccellenti programmatori giurano su questo e ne fanno motivo di discussione. Altri, semplicemente per esperienza, pensano suoni meglio di come in realtà funzioni. Si conoscono gruppi PHP che hanno membri in entrambe le categorie.

La verità è che è un problema più per PHP che per qualsiasi altro linguaggio di programmazione, perché PHP è molto efficace dal punto di vista della connettività multipla al database. Le installazioni aziendali di Java si riducono spesso alla scelta tra due o tre prodotti di database simili con connettività

JDBC; così gli sviluppatori Java non discutono affatto sui meriti della portabilità del database. Analogamente, ODBC è l'opzione standard per gli sviluppatori ASP. Il problema dell'astrazione del database sorge fundamentalmente perché PHP si collega a così tanti prodotti di database con veloci API native.

Gli argomenti a favore dell'astrazione del database si riducono fundamentalmente a questo: è possibile scambiare i database senza modificare molto il codice, risparmiando a volte un po' di lavoro di battitura. Gli argomenti contro l'astrazione del database si riducono fundamentalmente a questo: se si devono scambiare i database, probabilmente si è già sprecato molto tempo; e la pratica limita alle funzioni SQL più semplici e comuni utilizzate in un modello fisso, invece che consentire la codifica in modo da sfruttare tutti i vantaggi del set di funzioni del particolare database.

Per esempio, una delle caratteristiche migliori di Oracle è la possibilità di prendere un intero set di risultati e inserirlo in un array di PHP numerico e a una sola dimensione. Nessuno degli altri principali database utilizzati con PHP possiede questa funzione: sono soltanto in grado di prendere una singola riga alla volta. Per implementarlo da soli in modo portabile si dovrebbero aggiungere informazioni addizionali alla propria query, poiché occorre prendere ogni singola riga e trasformarla in un valore in un secondo array. Naturalmente, è possibile eseguire un'operazione del genere, ma occorre chiedersi se è necessario.

Inutile dire che il passaggio del database non avviene: lo si dovrebbe fare personalmente più di una volta. Tuttavia, le menti esperte ritengono che si deve essere in grado di collegare differenti database ogni volta che lo si desidera. Senza dilungarsi su ciò che è ovvio, non è un compito banale cambiare i database. Non è qualcosa che gli sviluppatori vorranno prendere in considerazione a meno che la situazione non sia assolutamente disastrosa. In tutti i casi in cui si era coinvolti personalmente, il cambio del database faceva parte di una completa riscrittura del sito e comportava una tremenda quantità di dolore nel frenetico tentativo di consultare i manuali da 1200 pagine relativi ai database, e in interminabili tentativi senza risultato di trasferire i dati. Tutto ciò suggerisce che la portabilità del database in teoria potrebbe essere una buona idea, ma in pratica solitamente significa che in precedenza è stato fatto un errore di cattiva architettura. Agli albori del Web, le cattive decisioni riguardanti l'architettura erano una cosa naturale perché nessuno poteva prevedere quali tecnologie e prodotti sarebbero durati. Da quando la percentuale di modifica è diminuita e sono emersi chiaramente i leader nelle varie categorie di prodotti, questo tipo di operazione di salvataggio potrebbe diventare meno necessaria.

Come sempre, il consiglio è soddisfare innanzitutto le proprie esigenze ed essere scettici sui consigli non richiesti. Ricordare che molti pacchetti PHP open source e commerciali che incorporano l'astrazione di database possono avere scopi diversi dai propri: spesso servono per procurarsi la maggior utenza possibile installata e sono disposti ad accettare un codice di database non ottimale per raggiungere tali obiettivi. Si potrebbe semplicemente avere

la necessità di focalizzarsi su come far girare un particolare sito al meglio. Non è necessario prendere in considerazione qualsiasi consiglio proveniente da chi non ha mai dovuto realmente cambiare i database. Quando le persone parlano della loro esperienza di sostituzione dei database, è meglio scoprire quanto grande è stata la modifica in termini di programmazione: migrare da MS SQL Server a MySQL è meno impegnativo che passare da MySQL a Oracle. Quindi prendere una decisione su cosa è meglio fare nella propria particolare situazione.

SUGGERIMENTO *Esiste una situazione particolare in cui si deve accettare l'astrazione del database anche se non è la decisione tecnica giusta. L'esperienza degli autori insegna che molti clienti o capi inesperti desiderano disporre della possibilità futura di passare da un database perfettamente funzionante a Oracle o DB2. Questa è una preoccupazione di ingegneria sociale delicata, perché investono nell'idea che il proprio business online crescerà enormemente e diventerà una vera impresa. Inutile spiegare loro che quando avranno bisogno di Oracle o DB2, il sole avrà consumato tutta la sua energia e l'Himalaya sarà diventato una pianura. Si consiglia di sorridere e di implementare funzioni di wrapper del database.*

12.7 L'obiettivo: MySQL

Gli autori di questo libro, come molti altri team che abbiano mai scritto un libro su PHP, amano MySQL e lo utilizzeranno in tutti gli esempi presenti nella Parte seconda. MySQL è probabilmente il database più veloce, economico, facile e affidabile; possiede anche la maggior parte delle funzioni necessarie e, questa è la vera differenza, funziona più o meno bene nelle implementazioni di UNIX e di Windows.

Nonostante tutto questo amore e fiducia in MySQL, bisogna riconoscere che a volte non riesce a soddisfare tutte le esigenze. Più avanti nel libro verranno descritte due alternative, anche se in modo poco approfondito. Nel Capitolo 34 verrà esaminato Postgre SQL, un database open source che aspira al progetto relazionale a oggetti descritto in precedenza. Per la maggior parte degli scopi, l'utilizzo di Postgre SQL equivale a voler mettere una puntina con un martello, ma se semplicemente sono necessarie alcune di queste funzioni, è un'implementazione potente (e gratuita) che svolge molto bene il suo lavoro.

Il Capitolo 35 è dedicato a Oracle. Probabilmente si è già a conoscenza del fatto che Oracle è un prodotto commerciale, ed è lecito chiedersi perché PHP supporti un prodotto così diverso dal punto di vista filosofico. Sia per meriti tecnici, che per abilità del personale marketing, che per ipnosi di massa, Oracle è onnipresente nelle impostazioni commerciali; e l'ottimo supporto di PHP ha aperto le porte dell'open source a numerose organizzazioni che altrimenti non avrebbero mai visto la luce.

12.8 **Sommario**

Il grande vantaggio del Web è la sua abilità nel mettere a disposizione del pubblico velocemente e a buon prezzo grandi quantità di informazioni. Questa caratteristica è stata estremamente migliorata dal recente aumento di disponibilità di database economici e affidabili.

Poiché molti programmatori esperti non hanno mai dovuto scegliere prima d'ora un database, sono stati descritti alcuni punti fondamentali che si devono prendere in considerazione quando è necessario prendere una decisione. Questi includono il progetto di database di base (flat file, relazionali o relazionali a oggetti), l'API o il driver, e la facilità di portabilità futura. Anche le caratteristiche opzionali, come le transazioni o l'interfaccia grafica, devono rientrare nella scelta del database. PHP supporta molti database di diverso tipo, in modo da avere un'ottima opportunità di trovare esattamente il set di caratteristiche necessario.

