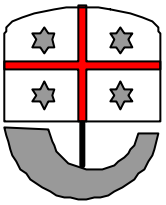

REGIONE LIGURIA

Servizio Programmi e
Strutture Culturali

LE BASI DI DATI E IL LORO IMPIEGO NELLE BIBLIOTECHE

testo di Beppe Pavoletti (Servizio Programmi e Strutture Culturali)

*L'autore di questo testo è raggiungibile all'indirizzo email giuseppe.pavoletti****@regione.liguria.it****. **Commenti, suggerimenti e segnalazioni di errori ed imprecisioni saranno graditissimi**.*
Il sito del Servizio Programmi e Strutture Culturali è: http://www.regione.liguria.it/menu/0611_fr.htn
Il Catalogo Collettivo delle Biblioteche Liguri (CBL) è consultabile all'URL: http://opac.regione.liguria.it/cgi-win/hiweb.exe/a3 oppure http://www.regione.liguria.it/cbl/.

V. 3 - aggiornamento: 18 luglio 2001.
Versione precedente: 2.2.1 12.1.2001

SOMMARIO

1. BASI DI DATI IN GENERALE[1]

Non è difficile comprendere che la nozione di *base di dati* o *archivio di dati* non si riscontra solo in contesti informatici. Infatti sono basi di dati tutti gli archivi, anche cartacei, destinati a contenere **informazioni strutturate** e che da tempo immemorabile sono in uso sia presso gli uffici che presso i privati. Per informazioni strutturate intendiamo quelle in cui determinate informazioni si ripetono, appunto, in una struttura regolare: ad esempio, un indirizzario non consiste in un testo libero come quello di una lettera o di una poesia, ma contiene informazioni ben definite come nome, cognome, via, città, provincia, codice postale. Si noti, poi, che anche in una lettera d'ufficio si potrebbe individuare una struttura (e quindi non solo un testo libero): data, indirizzo, oggetto, testo.

Possiamo quindi mettere in evidenza, indipendentemente da specifici riferimenti ai computer, alcuni elementi significativi:

- distinzione tra **struttura** e **dati**: la struttura deve essere definita prima dei dati ed è quella che determina la natura di un archivio (un indirizzario si distingue dal catalogo di una biblioteca perché prevede informazioni concettualmente diverse, e non perché materialmente vi si trovano dati differenti; infatti uno stesso dato, ad esempio un nome di persona, potrebbe essere materialmente presenti in entrambi gli archivi, ma non potrebbe avere lo stesso ruolo); **questa distinzione è forse la più importante e deve essere costantemente tenuta presente nella lettura di tutto ciò che segue**
- presenza di due diverse **tipologie di dati**: dati che si potrebbero definire *atomici, puntuali*, e che sono quelli che costituiscono in certo modo un tutto unico (ad esempio un cognome, un numero di telefono, una città, un codice fiscale), e dati che si potrebbero definire *testuali*, ossia costituiti da testi di una certa lunghezza e con una loro articolazione interna (ad esempio il titolo di un libro); in molti archivi sono presenti in modo prevalente o esclusivo dati di un tipo, ma non mancano archivi che accostano dati di entrambi i tipi (si pensi all'esempio della lettera d'ufficio prima citato); inoltre talvolta lo stesso dato può essere visto in entrambi i nomi: ad esempio, il titolo di un libro può essere visto come un testo, ma anche come una entità atomica
- presenza assai frequente di **dati duplicati**: ad esempio, in un indirizzario generalmente il nome della stessa città si presenterà più volte, nel catalogo di una biblioteca si presenterà più volte lo stesso autore o lo stesso soggetto
- necessità di effettuare sui dati diverse **operazioni**, spesso collegate tra loro: cancellazione, aggiunta, modifica; ricerca di un particolare dato o di un sottoinsieme dell'archivio; ordinamento dell'archivio secondo un certo criterio (ad esempio, ordinamento dell'indirizzazio secondo il cognome, oppure secondo la città, oppure secondo la provincia ecc.); tali operazioni sono collegate nel senso che, ad esempio, la cancellazione o modifica di un dato presuppone la sua ricerca all'interno dell'archivio, oppure può essere desiderabile effettuare certe operazioni, come l'ordinamento, solo su un sottoinsieme dell'archivio stesso.

Si vede subito che la gestione cartacea degli archivi di dati presenta significativi limiti rispetto a tutti i punti evidenziati sopra.

In realtà i primi due, essendo di carattere puramente concettuale, sarebbero in linea di principio indipendenti dalle particolari tecniche utilizzate. Tuttavia di fatto spesso la gestione cartacea degli archivi finisce per oscurare questi aspetti, per cui chi utilizza questi archivi cartacei tende percepire non tanto la loro struttura logica, quanto il loro aspetto materiale, ossia quello di testo scritto su carta che contiene delle informazioni in una determinata successione. Così, ad esempio, alcuni bibliotecari, più o meno consapevolmente, concepiscono la notizia bibliografica non come la combinazione di vari elementi concettualmente distinti (intestazione per autore, descrizione ecc.), ma come un testo scritto (su carta) che inizia con l'intestazione autore, prosegue con la descrizione, e così via, quasi che l'intestazione non fosse uno specifico dato con la sua identità, ma fosse essenzialmente la prima riga un testo.

---

[1] La trattazione teorica delle basi di dati e del modello relazionale è basata soprattutto su *[Database 1997]*. Utile è anche l'esposizione, più semplice, che si trova in *[Datenbanken 1998]* o in *[Dati 1996]* nonché quella, ancora più semplice, di due volumi pensati soprattutto per i bibliotecari, cioè *[Informatica 1983]* e *[Rappresentazione 1988]*.

I limiti degli archivi cartacei rispetto al terzo e al quarto punto sono invece molto più sostanziali. Per quanto riguarda la duplicazione dei dati, non c'è quasi nessuna via di scampo, poiché i dati ripetuti vanno fisicamente scritti sulla carta per poter essere letti ed utilizzati. Ci sarebbe, in linea di principio, un'altra possibilità: si potrebbe infatti non riscrivere tutto il dato, ma utilizzare un codice che rimanda ad un altro archivio in cui si trova il dato completo. Per esempio, nel caso dell'indirizzario si potrebbe assegnare un numero a ciascuna città, e poi nell'indirizzo non scrivere il nome della città, ma solo il numero; per avere il nome della città, basterebbe andare a vedere nell'archivio delle città, a quale nome corrisponde quel dato numero. Si capisce benissimo, però, a quali complicazioni pratiche conduce questo metodo in una gestione cartacea, soprattutto se gli archivi hanno una struttura complessa e se i dati sono molti.

Per quanto riguarda poi le operazioni sui dati, esse richiedono la manipolazione fisica dei supporti (ad esempio schede, registri ecc.). In particolare, si noti che: avere un ordinamento diverso da quello standard richiede la duplicazione fisica dell'archivio o di una parte di esso, a meno di rinunciare all'ordinamento di partenza (ad esempio, se i dati si trovano su schede, è possibile ordinare le schede come si vuole, ma ogni volta in un solo modo, a meno di avere più copie delle schede stesse); l'ordinamento è una operazione manuale lunga e soggetta ad errori; l'accesso rapido ad un dato è possibile solo se questo dato rientra nel criterio di ordinamento, altrimenti è necessaria una ricerca sequenziale, cioè una scansione di tutto l'archivio dall'inizio alla fine; l'estrazione di un sottoinsieme di dati implica una duplicazione fisica degli stessi (come quando da un archivio si estrae una lista di nomi battuta a macchina su un foglio), oppure lo spostamento materiale dei dati con rischio di dispersione e danneggiamento dei supporti (come quando in un catalogo a schede si estrae un cassetto per consultarlo altrove); le operazioni di modifica e aggiunta di dati possono creare incongruenze nell'archivio, in misura dipendente dal supporto: ad esempio, se l'archivio è scritto in un registro, non è evidentemente possibile aggiungere nuovi dati tra quelli già esistenti, ma solo in fondo al registro, per cui di tanto in tanto è necessario riscrivere tutto l'archivio con i nuovi dati ordinati nella posizione corretta. Chi ne ha voglia, non dovrebbe trovare difficoltà ad esplicitare altri limiti e problemi derivanti dalla gestione cartacea e manuale degli archivi di dati.

Risulta quindi estremamente desiderabile utilizzare i computer per superare questi inconvenienti; anche i computer, però, non risolvono i problemi per magia, ma solo se sono dotati di hardware e software adeguati alla bisogna. Dell'hardware qui non ci occupiamo, perché l'hardware necessario alla gestione degli archivi di dati fa parte da molti anni della dotazione di qualsiasi computer: si tratta essenzialmente di video, tastiera e memorie di massa (come hard disk o CD-ROM) oltre, beninteso, a CPU e RAM. Ci sono, ovviamente, questioni di prestazioni, alle quali però accenneremo in seguito.

È invece il software quello di cui dobbiamo maggiormente occuparci. Infatti anche con un computer non si possono gestire efficacemente basi di dati se non con dei software appropriati.

Alcuni, a questo scopo, usano semplicemente un programma di scrittura: creano quindi un documento e vi inseriscono dei dati, ad esempio una serie di registrazioni bibliografiche di aspetto analogo alle schede cartacee. Questa tecnica può andare bene solo per esigenze di livello del tutto elementare, perché ripropone molti dei problemi degli archivi cartacei: ad esempio, come si fa ad ordinare efficacemente una base dati di questo genere? Inoltre il programma di scrittura, non essendo progettato per questo scopo, non sa assolutamente nulla della struttura dell'archivio, la quale può risultare evidente a chi legga il testo, ma non può propriamente essere gestita dal software, appunto perché esso non ha i mezzi per rappresentarla. Sarebbe anzi più esatto dire che in questo modo non si creano vere e proprie basi di dati, ma solo dei testi in forma di elenco.

Altri invece utilizzano i fogli elettronici per gestire le loro basi di dati. Questi programmi si avvicinano un poco di più a quello che sarebbe necessario allo scopo, ma sono ben lontani dal soddisfare i requisiti che sarebbero desiderabili. Essi infatti mancano proprio del requisito fondamentale della distinzione tra struttura e dati: l'unica struttura che mettono a disposizione è quella del tutto generica della tabella organizzata in righe e in colonne,

dove però ogni riga e ogni colonna può contenere qualsiasi cosa (nulla mi vieta di mettere in una colonna via via un nome, una data, un numero, un testo, un'altra data ecc.). Questo, in alcuni fogli elettronici, può anche determinare gravi inconvenienti: ad esempio in Microsoft Excel si può ordinare una colonna indipendentemente dalle altre; questa funzionalità, se usata a sproposito, può rendere inutilizzabili tutti i dati. Si pensi al solito indirizzario creato inserendo i nomi nella prima colonna, i cognomi nella seconda e così via: se si ordina solo la prima colonna, si perde completamente la corrispondenza tra i nomi e tutti gli altri dati ! Anche dei fogli elettronici si può quindi dire che non siano in grado di gestire vere e proprie basi di dati.

A questo scopo, infatti sono stati sviluppati appositi programmi, che si distinguono essenzialmente in due categorie: **Data Base Management System (DBMS)** e **Information Retrieval Systems (IRS)**. Per quanto riguarda i DBMS, se ne sono succeduti diversi tipi, ed in particolare i DBMS gerarchici, i DBMS reticolari e il **DBMS relazionali (RDBMS)** che sono quelli oggi generalmente usati e dei quali ci occuperemo nel seguito. Pertanto si intende che con la sigla DBMS, se non risulta diversamente, indicheremo i DBMS relazionali. Nelle sezioni successive verranno esposte le caratteristiche principali di RDBMS e IRS. Avvertiamo che queste due tipologie di software verranno illustrate nelle loro caratteristiche più tipiche e quindi, per così dire, in forma pura, ma si deve tener presente che di fatto molti prodotti commerciali tendono sempre più ad assumere caratteristiche ibride.

1.1 Elementi strutturali

In questo paragrafo vogliamo introdurre alcuni concetti che non corrispondono tanto a componenti software direttamente utilizzabili e visibili, ma piuttosto ad elementi strutturali dei sistemi software di archiviazione.

Lo standard ANSI/SPARC, che si propone di concettualizzare a livello generale i sistemi di gestione delle basi di dati, utilizza a questo scopo alcune nozioni che sono già emerse nella nostra esposizione, distinguendo tra:

- **piano esterno** che comprende la visione dei dati dal punto di vista dell'utente e i mezzi a sua disposizione per la loro manipolazione
- **piano concettuale** che comprende la struttura logica dei dati e le regole di integrità
- **piano interno** come comprende elementi come i tipi di dati, le caratteristiche dei campi, il formato di memorizzazione, gli indici, e a sua volta fa riferimento ai servizi del sistema operativo per la manipolazione fisica dei dati attraverso l'accesso ai supporti di memorizzazione e alla rete

A questo punto possiamo introdurre due concetti molto utili per descrivere, a livello astratto, i sistemi di gestione degli archivi di dati sono quelli di Data Definition Language = Linguaggio di Definizione dei Dati (DDL) e di Data Manipulation Language = Linguaggio di Manipolazione dei Dati (DML)[2]. Non si deve pensare che in ogni prodotto questi elementi siano presentati come veri e propri linguaggi, tuttavia le funzioni svolte dal DDL e dal DML devono essere presenti, altrimenti non si potrebbe neppure parlare di vera e propria gestione di basi di dati.

Il DDL ha lo scopo di definire i dati, ossia di determinare quali dati si trovano in un archivio (in una tabella di RDBMS o nel file piatto di un IRS - per questi concetti v. capitoli seguenti), stabilendo ad esempio che esiste il campo *titolo* che accetta dati di tipo testo con una lunghezza massima di 100 carattere, il campo *prezzo* che accetta numeri interi, il campo *data di ingresso* che accetta dati di tipo data ecc. Il risultato delle istruzioni formulate tramite il DDL è il **dizionario dei dati** (detto anche **catalogo di sistema**), cioè la descrizione esplicita e completa della struttura dell'archivio, ossia dei dati che esso può accettare. I dati contenuti nel dizionario dei dati vengono anche detti *metadati*, perché sono dati che si riferiscono ad altri dati. Spesso il dizionario dei dati è registrato a sua volta come un database, e quindi interrogabile, visualizzabile ecc.

[2] Di solito i termini DDL e DML vengono introdotti soprattutto nella trattazione degli RDBMS, ma non sembra che in linea di principio ci sia alcunché che impedisca di introdurli ad un livello più generale, come appunto qui.

Il DML ha invece lo scopo di permettere agli utenti di eseguire manipolazioni dei dati, ed in particolare: ricerca di dati, creazione di dati, cancellazione di dati, modifica di dati. La parte del DML che presiede alla ricerca dei dati viene anche detta *linguaggio di interrogazione* o *linguaggio di query* (*query language*).

Oltre al DDL e al DML si parla di DCL, cioè Data Control Language: si tratta del linguaggio che permette di assegnare agli utenti i permessi di fare determinate operazioni. Si deve infatti tenere presente che quando un database è utilizzato da molti, se ognuno potesse fare qualsiasi operazione, compreso cancellare tutto il database, non mancherebbero di accadere dei disastri. C'è quindi una figura particolare, cioè il **DBA = Data Base Administrator**, che assegna a ciascuno le sua autorizzazioni, e inoltre può intervenire sul database per modificarlo o eliminare malfunzionamenti. Naturalmente il DBA non può essere chiunque: si tratta in genere di tecnici specializzati, con una profonda conoscenza, teorica e pratica, di tutto quanto riguarda le basi di dati. Ma chi assegna al DBA le sue autorizzazioni, che non sono in generale illimitate? A questo provvedere una specie di super DBA, detto di solito **amministratore di sistema** (*system administrator*) o **superutente** (*superuser*), che ha la possibilità di effettuare qualunque operazione, compresa la cancellazione dell'intero database e la disinstallazione del programma di gestione. A seconda dei casi e delle esigenze, può accadere che una stessa persona svolga sia le funzioni di DBA che quelle di amministratore di sistema. Si noti che nei database per uso personale spesso queste figure non ci sono perché il programma non permette di gestire le autorizzazioni, oppure è l'utente stesso che, magari senza accorgersene, ha anche le funzioni di DBA e amministratore di sistema, e quindi può compiere qualsiasi operazione.

Non in tutti i programmi il DDL e il DML sono messi a disposizione sotto forma di linguaggi in senso stretto, cioè come istruzioni da formulare con una specifica sintassi: a volte vengono presentate all'utente delle interfacce di più agevole utilizzo che però in linea di principio si potrebbero tradurre in un DDL in senso stretto. Per fare un esempio concreto, consideriamo il noto software CDS-ISIS, dove il linguaggio di ricerca si presenta a tutti gli effetti come un linguaggio (nel quale si possono formulare comandi come: *informatica/(15) * biblioteche(1)*), ma le altre parti del DML sono espresse attraverso le istruzioni del menu di editing (modifica record, nuovo record ecc.) oppure attraverso istruzioni del linguaggio di programmazione Isis-Pascal, mentre il DDL consiste nelle istruzioni della FDT.

Il DDL e il DML di un software di gestione di archivi di dati rappresentano completamente le potenzialità del software stesso: non ha evidentemente alcun senso, infatti, pensare che un software di questo genere possa trattare dati non descrivibili nel suo DDL o effettuare operazioni non descrivibili nel suo DML.

Il DDL e il DML fanno riferimento a quella che viene chiamata la **struttura logica** dell'archivio, cioè ai dati come sono intesi e utilizzati dagli utenti e dagli sviluppatori, ad esempio: titolo, autore, nome, cognome, stipendio lordo, data di nascita ecc. Tuttavia, poiché i dati devono essere fisicamente registrati sulle memorie di massa, l'archivio deve avere, come abbiamo visto illustrando lo schema ANSI/SPARC, anche una **struttura fisica**. Una descrizione della struttura fisica di un archivio potrebbe essere di questo genere (attenzione: si tratta di una descrizione completamente inventata e riportata a solo scopo esemplificativo): *il file inizia con una intestazione di 34 bytes; i bit nelle posizioni 0-31 contengono la data di aggiornamento sotto forma di un intero lungo rappresentante il numero di secondi trascorsi dal 1 gennaio 1900; i bit nelle posizioni 32-35 sono impostati a 0 e riservati per uso futuro* ecc. Normalmente l'utente non ha bisogno di conoscere la struttura fisica dell'archivio che sta utilizzando, a meno che non sia particolarmente interessato ad approfondire l'aspetto tecnico, ed in particolare a scrivere egli stesso software per la gestione a basso livello dell'archivio[3]. Comunque, la documentazione di alcuni software, come CDS-ISIS e dBase IV, contiene la descrizione della struttura fisica degli archivi, mentre per altri software, come Microsoft Access, questa non è nota. Si noti che generalmente la struttura fisica è comune a

[3] Contrariamente a quanto si potrebbe pensare, non è particolarmente difficile, anche per un programmatore dilettante, realizzare semplici procedure che accedono a basso livello ad un archivio per effettuare operazioni elementari, specialmente letture sequenziali: la cosa è particolarmente facile per il formato dBase, che è molto semplice, è un po' più difficile con CDS-ISIS. Naturalmente realizzare un completo motore di gestione di basi dati è cosa ben diversa, che richiede competenze di altissimo livello.

tutti gli archivi gestiti da un determinato programma, anche se - come ovvio - gli archivi non possono essere tutti fisicamente identici, dal momento che contengono dati diversi.

Questo discorso ci conduce alla distinzione tra **storage manager** e **query processor**. Lo storage manager è la parte del sistema di archiviazione che si occupa di immagazzinare fisicamente i dati sulle memorie di massa, nonché di leggerli e scriverli (per crearli, cancellarli o modificarli). La presente versione di questo documento non tratta della struttura fisica degli archivi di dati, cioè del formato in cui i dati vengono fisicamente immagazzinati sulle memorie di massa: questo non vuol dire che la questione non sia importante, anzi, è assolutamente fondamentale per il conseguimento di adeguate prestazioni ed affidabilità. Si deve infatti sempre ricordare che qualunque operazione effettuata da un utente su di un archivio di dati si riconduce in ultima analisi a operazioni fisiche di lettura e/o scrittura effettuata dallo storage manager sulle memorie di massa in base ai comandi impartiti dall'utente. L'argomento, alquanto complesso, è comunque trattato in numerose pubblicazioni citate in bibliografia.

Lo storage manager può a sua volta essere concepito come suddiviso in diversi altri componenti, ossia:

- il gestore dei metodi di accesso ai dati, che è il vero e proprio gestore dell'accesso alle memorie di massa
- il *buffer manager*, responsabile del trasferimento dei dati tra memoria di massa e memoria centrale (e viceversa)
- il controllore dell'affidabilità, che ha il compito di assicurare il mantenmento del contenuto della base dati in presenza di malfunzionamenti
- il controllore della consistenza, che si occupa di fare un modo che si possano avere molti accessi contemporanei alla base dati senza che si generino errori e perdite di consistenza (l'argomento, che è particolarmente importante, sarà trattanto più estesamente di seguito)

Il query processor è invece il componente che si occupa di elaborare le interrogazioni, o query, formulate dall'utente per poi affidarsi allo storage manager per il recupero fisico dei dati. Si noti che le query fanno riferimento alla struttura logica dell'archivio e non a quella fisica: ad esempio, si può interrogare un database per ricercare tutti i libri che hanno un certo titolo o un certo autore, o magari anche per recuperare il record numero 3075, ma nessuno verrà in mente di richiedere il recupero di 12.213 bytes a partire dalla posizione 117.274 dell'archivio[4], e normalmente questo non è neanche possibile. Queste operazioni vengono invece effettuate dallo storage manager in base ai risultati dell'elaborazione eseguita dal query processor. Un componente normalmente presente nell'ambito del query processor è l'**ottimizzatore di query**, che rielabora la query impostata dall'utente per ricondurla alla forma migliore per renderne l'esecuzione più efficiente.

Si deve ricordare che in questo contesto il termine query ha un senso più ampio dell'interrogazione in senso stretto, cioè di quella formulata allo scopo di conoscere il contenuto della base dati, a si può interpretare nel senso di qualsiasi richiesta rivolta al software di gestione, non solo quelle di ritrovamento dei dati, ma anche quelle di modifica o aggiunta. In particolare, tutte le istruzioni in linguaggio SQL, che verrà trattato in seguito, vengono dette query.

1.2 Il modello entità/associazioni (E/R).

Nel paragrafo precedente abbiamo dato una descrizione astratta di un generico sistema di gestione di basi di dati. Vogliamo ora rivolgerci ai dati stessi per tentare di darne ancora una descrizione astratta, un modello, utile a permetterne il trattamento tramite software, che non è possibile se non si definisce in modo rigoroso l'oggetto dell'elaborazione. Il più noto modello dei dati è il **modello entità/associazioni**, designato anche con la sigla E/R, dall'inglese *entity/relationship*. Molti osserveranno che l'espressione *entità/associazioni* è alquanto inconsueta, essendo normalmente utilizzata invece quella *entità/relazioni*. Quest'ultima espressione però introduce una ambiguità: infatti il termine *relazione*, che qui corrisponde a *relationship* è lo stesso che poi nella trattazione dei DBMS relazionali viene utilizzato per designare le relazioni nel senso della teoria degli insiemi (in inglese *relation*). L'uso del

[4] A nessuno, si intende, tranne che a coloro che hanno letto questa dispense e ne sono stati spinti a queste morbose curiosità.

termine *associazioni* mantiene l'espressività della locuzione e nello stesso tempo evita l'ambiguità[5]. Il modello E/R non è la descrizione di un software, ma solo una descrizione astratta e formalizzata dei dati che poi può essere presa a base, in modo più o meno completo e rigoroso, dalle diverse implementazioni software. Per quanto le implementazioni tramite i DBMS relazionali possano apparire più naturali, è del tutto possibile implementare il modello anche tramite un information retrieval.

Una **entità** è una cosa distinguibile da altre tramite le sue proprietà. La delimitazione di una entità è in gran parte relativa alle esigenze del sistema informativo che si deve costruire: ad esempio, in un sistema di gestione del personale una persona sarà normalmente una entità, ma in un sistema di documentazione medica potrebbe essere utile considerare come entità le varie parti del corpo di una persona. Nel campo biblioteconomico, entità saranno i vari componenti di una notizia bibliografica, come il titolo, la descrizione, il soggetto, la copia fisica[6]. Una entità possiede un insieme di proprietà (ad esempio l'entità bibliografica titolo potrebbe possedere le proprietà: natura, lingua e data), e ogni proprietà possiede un valore preso da un certo insieme di valori possibili (nell'esempio precedente potremmo avere: natura=monografia, lingua=italiano e data=1996). Le proprietà sono dette anche **attributi**, e d'ora in poi verranno designate con questa espressione. Come osservato, un attributo non può avere qualsiasi valore, ma solo uno tra un certo insieme di valori possibili, detto **dominio** (in inglese anche *value set*) dell'attributo (per esempio, l'attributo *anno corrente* può avere come valore un numero intero non superiore a quello dell'anno corrente, e quindi non può avere valori come *5e+3, 4011, 7/12, Roma*).

Un insieme di entità dello stesso tipo che condividono gli stessi attributi è detto, appunto, **insieme di entità** (*entity set*). Alcune entità possono appartenere a più insiemi, per esempio se è stata definita l'entità *bibliotecario* e l'entità *lettore* ci può essere un bibliotecario che ha anche il ruolo di lettore (quando prende un libro in prestito).

Gli attributi possono essere:

- **semplici** o **composti**; quelli semplici non possono essere ulteriormente suddivisi, mentre quelli composti possono essere suddivisi in ulteriori attributi; per esempio la data, nell'esempio precedente, è un attributo semplice, ma in SBN è composto da due parti, che sono a loro volta attributi a tutti gli effetti, cioè il tipo di data (data certa, data incerta ecc.) e l'anno
- **a singolo valore** o **multivalore**; i primi possono avere un solo valore, i secondi molti; ad esempio, nel nostro esempio bibliografico, l'attributo *natura* è a singolo valore (un titolo può essere di una monografia, di un periodico, di uno spoglio ecc., ma solo di una cosa per volta), mentre l'attributo *inventario* può essere multivalore, poiché ad un titolo possono corrispondere diverse copie di un documento, ognuna con il suo numero di inventario; si tenga presente che a volte un attributo multivalore può essere meglio rappresentato da una associazione[7]
- **null**; quando una specifica entità non ha un valore per un certo attributo, perché non applicabile o sconosciuto, si dice che questo attributo ha valore *null*; non bisogna confondere il valore *null* con lo 0, che è un numero ben preciso e quindi un valore come qualunque altro
- **derivati**; sono derivati quegli attributi il cui valore può essere calcolato a partire da altri; per esempio, il titolo, oltre all'attributo multivalore *inventario* potrebbe avere l'attributo *numero copie*; è chiaro che il valore di questo attributo può essere calcolato semplicemente contando il numero di valori di *inventario*

Una **associazione** collega diverse entità, che possono appartenere allo stesso o a diversi insiemi di entità. Per esempio, se abbiamo gli insiemi di entità *titoli* e *lettori*, possiamo associare a ogni lettore i titoli dei libri che ha

---

[5] In realtà in inglese esiste anche il termine *association*, che significa associazione in senso generico.

[6] Non è difficile accorgersi del fatto che l'analisi del dato bibliografico che sta alla base di SBN è di fatto una applicazione del modello E/R, e può anzi essere utilmente presa ad esempio di una applicazione molto ampia e dettagliata di questo modello. Anche i recenti *Functional Requirements for Bibliographic Records* elaborati dall'IFLA sono largamente ispirati al modello E/R

[7] Gli utenti di CDS-ISIS noteranno che gli attributi multivalore sono quelli che il software rappresenta, in modo molto intuitivo, tramite i campi ripetibili.

in prestito[8]. **Una associazione è anche una relazione nel senso della teoria degli insiemi**, ma il concetto di relazione sarà esposto nel capitolo sugli RDBMS. A questo punto ci si può chiedere perché allora non usare il termine *relazione* anche qui, invece di usare *associazioni.* Il motivo è che il contesto è diverso: il termine *relazione* viene usato nel contesto dell'algebra relazionale, che è una teoria matematica, mentre il termine *associazione* viene usato nel contesto modello E/R che è un modello dei dati e di per sé può essere trattato senza esplicitare la sua connessione con l'algebra relazionale. C'è poi anche un motivo didattico: evitare il termine *relazioni* nel contesto del modello E/R può servire a non indurre qualcuno a pensare che i DBMS relazionali si chiamino così perché mettono in relazione diverse tabelle: vedremo successivamente che non è questa l'origine del nome (anche se naturalmente è vero che i DBMS relazionali permettono di collegare più tabelle).

Una associazione può possedere degli **attributi descrittivi**, che specificano alcune particolari caratteristiche di quella associazione[9].

Le **cardinalità di mappatura** o **rapporti di cardinalità** esprimono il numero di entità di un insieme di entità con le quali una entità può essere messa in corrispondenza da una associazione. Si distinguono quattro casi, che esemplificheremo con riferimento agli ipotetici insiemi A e B:

- **uno a uno**, quando una entità in A è associata al massimo ad una entità in B
- **uno a molti**, quando una entità in A può essere associata a più entità in B, mentre una entità in B può essere associata solo ad una entità in A
- **molti a uno**, quando una entità in B può essere associata a più entità in A, mentre una entità in A può essere associata solo ad una entità in B
- **molti a molti**, quando una entità in A può essere associata a più entità in B e viceversa

Di grande importanza è la nozione di **dipendenze esistenziali**, che si hanno quando l'esistenza di una entità è dipendente da quella di un altra. In questo caso la prima è detta *entità dipendente* e l'altra *entità dominante*. Per esempio, se è stato definito l'insieme di entità *prestito*, è evidente che una entità prestito sarà dipendente sia rispetto alle entità documento che alle entità lettore: infatti un prestito non può esistere se non collega un documento ad un lettore, mentre un documento può esistere anche se nessuno lo prende in prestito e un lettore può esistere anche senza documenti in prestito. Le dipendenze esistenziali sono importanti per la normalizzazione, che verrà trattata nel capitolo sugli RDBMS.

Dal punto di vista della progettazione del database, le entità devono essere distinte attraverso i loro attributi. Questo ci conduce al concetto di **chiave**, che è veramente un concetto chiave!

Definiamo innanzitutto una **superchiave** come un insieme di attributi che, presi nel loro insieme, identificano in modo univoco una entità in un insieme di entità. Ad esempio, gli attributi *titolo, data, dimensioni, ISBN* identificano univocamente una entità bibliografica, poiché è impossibile che due entità abbiano lo stesso valore per tutti questi attributi contemporaneamente. Vediamo subito, però, che una superchiave così definita non è molto interessante, perché contiene molti attributi ridondanti: infatti, basterebbe l'ISBN da solo a identificare univocamente una entità, mentre non basterebbe nessuno degli altri tre attributi, né alcuna loro combinazione. Formalmente, si può dire che **per qualsiasi superchiave, ogni suo sovrainsieme è ancora una superchiave**, ossia se prendiamo una superchiave a nostro piacere e le aggiungiamo degli attributi qualsiasi otteniamo ancora una superchiave, evidentemente più o meno ridondante.

Una superchiave tale che nessun suo sottoinsieme proprio è ancora una superchiave, e che quindi contiene solo gli attributi minimi indispensabili per identificare univocamente una entità, è detta **chiave candidata**: di

[8] Si intende che questo è un esempio piuttosto rozzo, scelto per ragioni di semplicità: sarebbe meglio infatti definire un terzo insieme di entità, quello dei *documenti fisici*, associato da una parte ai titoli e dall'altra ai lettori

[9] Le note al legame utilizzate in SBN sono esempi di questi attributi descrittivi

fatto, sono queste le chiavi veramente interessanti. Una chiave candidata scelta dal progettista come identificatore principale delle entità di un insieme di entità è detta **chiave primaria**.

In ogni caso, la principale proprietà di una chiave, sia essa superchiave o chiave candidata, è **due entità non possono avere gli stessi valori per tutti gli attributi di una chiave**.

Non tutte le chiavi, peraltro, rientrano in questa categoria: ad esempio le note chiavi OCLC sono costruite apposta in modo presentare valori duplicati, per aiutare il ritrovamento di elementi simili (titoli o autori) soprattutto allo scopo di individuare dati duplicati per errore, ma non letteralmente identici, bensì con qualche differenza di forma.

Come si è visto, una chiave può essere composta di più attributi, ma spesso si crea un attributo apposta per svolgere la funzione di chiave primaria: due esempi sono il BID e il VID in SBN.

Se un insieme di entità non ha attributi sufficienti per creare una chiave primaria si chiama **insieme debole di entità** (*weak entity set*), altrimenti si chiama **insieme forte di entità** (*strong entity set*). Alcuni software, come Microsoft Access, tentano di "scoraggiare" la creazione di insiemi deboli di entità offrendosi di generare automaticamente una chiave primaria, se essa non è stata definita dal progettista della base dati, aggiungendo un apposito attributo[10].

2. RDBMS

Gli RDBMS sono strumenti che si sono dimostrati, nel corso del tempo, estremamente potenti e flessibili. Tra i prodotti che rientrano in questa categoria possiamo citare sia dei software per PC, come i classici dBase III, dBase IV e gli attuali Fox, Paradox, Access, Visual dBase, sia dei software concepiti soprattutto per sistemi più grandi (anche in alcuni casi possono essere utilizzati anche su PC), come Microsoft SQL Server, Oracle, Informix, Ingres, Sybase, DB2, Interbase, Adabas, PostgreSQL, My-SQL, Progress.

La definizione teorica degli RDBMS si deve a Edward F. Codd, che la enunciò nel 1970 con il saggio *A relational model for large shared data banks*. Si tratta di una definizione estremamente precisa e rigorosa perché si basa sui concetti della teoria degli insiemi. Infatti i DBMS relazionali non si chiamano così, come forse molti credono, perché consentono di mettere in relazioni dati diversi, ma perché le strutture di dati su cui operano sono **relazioni** nel senso definito nell'ambito della teoria degli insiemi.

Ecco che cos'è una relazione: **dati uno o più insiemi non necessariamente distinti** (ossia si può prendere più volte lo stesso insieme) **si definisce prodotto cartesiano l'insieme che contiene tutte le n-uple** (ossia: coppie se gli insieme sono due, triple se sono tre, quadruple se sono quattro; invece di n-uple si può dire anche t-uple) **ordinate ottenute prendendo, sempre nello stesso ordine, un elemento da ciascun insieme** (un elemento dal primo insieme, un elemento dal secondo e così via fino all'ultimo, poi di nuovo un elemento dal primo ecc.) **e che non contiene nessun altro elemento; si definisce relazione un qualunque sottoinsieme di un prodotto cartesiano (incluso il prodotto cartesiano stesso)**[11].

[10] Come si vedrà meglio in seguito, in Access e in tutti gli altri RDBMS, gli insiemi di entità sono le tabelle.

[11] Si noterà che questa definizione sembra presupporre l'**assioma di scelta**; infatti si assume che il risultato dell'operazione di estrazione degli elementi dagli insiemi di partenza, per la quale non viene dato alcun metodo costruttivo, sia a sua volta un insieme. Questo sembra determinare una situazione singolare: da una parte è noto che l'assioma di scelta non è dimostrabile né refutabile in base agli altri assiomi della teoria degli insiemi, per cui è possibile costruire una teoria degli insiemi **senza** assioma di scelta; d'altra parte, è un fatto che i DBMS relazionali esistono e funzionano proprio secondo questo principio. Che cosa dovrebbe fare quindi chi rifiuta l'assioma di scelta? Forse negare l'esistenza dei DBMS relazionali ? In realtà il paradosso è solo apparente: il problema si porrebbe se si trattasse di insiemi

L'**arietà** di una relazione è il numero di insiemi presi a base per la costruzione della relazione.

Osserviamo, di passaggio, che la nozione di relazione ha applicazioni che vanno ben oltre i DBMS relazionali: ad esempio le funzioni, concetto fondamentale dell'analisi, si possono definire come relazioni.

Dunque i DBMS relazioni operano su relazioni. In particolare, non è difficile comprendere che la struttura della base dati viene determinata dalla scelta degli insiemi dei quali viene fatto il prodotto cartesiano. Per il ben noto indirizzario, questi insiemi potrebbero essere quello dei nomi, quello dei cognomi, quello degli indirizzi e quello delle città. Una registrazione del database sarebbe una n-upla composta da un nome, un cognome, un indirizzo e una città.

Le relazioni su cui operano i DBMS vengono normalmente rappresentate come **tabelle** composte di righe e di colonne. Attenzione però: queste non sono come le tabelle dei fogli elettronici, nei quali ogni cella non è altro che un contenitore generico dove si può mettere qualsiasi cosa. In una tabella di DBMS le **colonne** rappresentano gli insiemi di partenza, mentre le righe rappresentano le n-uple. Le righe di una tabella di DBMS sono dette più comunemente **record**. Le colonne vengono anche dette **attributi** della tabella. L'insieme dei possibili valori assunti da un attributo è il **dominio** dell'attributo. Sarebbe più esatto dire che l'attributo è il criterio in base al quale viene definito l'insieme, ma nella terminologia più comune gli attributi vengono chiamati **campi**. Nel seguito utilizzeremo normalmente la terminologia corrente, e quindi i termini *record* e *campo* (le altre espressioni, come *n-uple, attributi, dominio* vengono impiegate prevalentemente, anche se non esclusivamente, quando si tratta l'argomento da un punto di vista teorico). Per quanto riguarda le *tabelle*, esse spesso vengono chiamate con questo nome ma, anche a seconda del particolare prodotto che si usa, possono essere chiamate anche **archivi** o **file di database**[12]**.** Da notare che un file di database di per sé non coincide con un file del sistema operativo: ad esempio, in dBase III (e in tutta la famiglia xBase) una tabella è anche un file del sistema operativo, ma in Microsoft Access una tabella è contenuta in un file del s.o. non da sola ma insieme a molti altri elementi. Osserviamo ancora che per **database** o **base di dati** non si intende la singola tabella come tale, ma un insieme di tabelle logicamente correlate, eventualmente con l'aggiunta di altri elementi quali schermate di inserimento dati, formati video e stampa ecc. Ovviamente, occasionalmente può benissimo accadere che un database consti di una sola tabella. Tanto per completare la confusione terminologica, talvolta, soprattutto nel linguaggio informale, si dice anche *database* per indicare il DBMS, ma qui non seguiremo questo uso che può più facilmente indurre in confusione.

Dovrebbe essere ovvio, a questo punto, che **le tabelle sono insiemi di entità** nel senso del modello E/R. Le **associazioni** (*relationship*) **a loro volta sono relazioni,** poiché associano entità prese di volta in volta da diversi insiemi di entità.

La struttura di una tabella, ossia i campi (colonne) che contiene viene detta **tracciato record**. Come già detto, la lista completa di tutti campi con tutte le loro caratteristiche (tipo di dati, eventuali vincoli sui dati consenti ecc.) viene detta **dizionario dei dati.** La stessa terminologia **si applica anche agli information retrieval** di cui parleremo nel capitolo successivo.

Circa la struttura fisica con cui le tabelle vengono memorizzate da parte del RDBMS, non ci sono regole fisse, ma sono i progettisti che tentano varie soluzioni al fine soprattutto di migliorare le prestazioni e l'affidabilità del prodotto. Tuttavia si può dire che i campi di una tabella sono **a lunghezza fissa**, ossia occupano sempre lo

---

infiniti, ma poiché quelli di cui si occupano i DBMS relazionali sono solo insiemi finiti è sempre possibile, in linea di principio, effettuarne la costruzione anche solo enumerandone gli elementi.

[12] Si noti che con un RDBMS si possono effettuare tutte le operazioni che si fanno con un foglio elettronico; infatti l'unica struttura di dati di un foglio elettronico è una tabella generica, che quindi può essere gestita dal DBMS relazionale; come già evidenziato, non è vero invece il contrario; in realtà, gli unici vantaggi dei fogli elettronici si riscontrano quando si devono effettuare operazioni molto semplici, perché non obbligano a definire separatamente la struttura dell'archivio

stesso spazio indipendentemente dal fatto che contengano dati oppure no. Inoltre la struttura del database è strettamente vincolante, ed è la stessa per tutta la tabella, come risulta ovvio dalle spiegazioni precedente. In altri termini, se per una tabella sono stati definiti tre campi, ma in un record o due ne servirebbe un quarto, non si può inserirlo solo per quei particolari record, ma bisogna definirlo per tutti oppure rinunciarvi (vedremo in seguito che queste restrizioni possono non valere per gli information retrieval).

Pressoché in tutti i RDBMS alle tabelle contenenti i dati sono associati degli **indici** per velocizzare la ricerca. Essi funzionano in modo analogo agli indici di un libro: per cercare qualcosa, invece di scorrere tutti i dati, si cerca rapidamente nell'indice il rimando al punto in cui si trova la cosa cercata. Gli indici sono gestiti attraverso varie tecnologie che hanno tutte quattro scopi principali: aumentare la velocità di accesso alle voci degli indici, diminuire le dimensioni degli indici, aumentare la velocità di creazione e aggiornamento degli indici e migliorare l'affidabilità di questi ultimi (ossia evitare che negli indici si generino voci errate o addirittura errori così gravi da renderne impossibile l'uso).

2.1 Operazioni degli RDBMS

Naturalmente gli RDBMS sono utili perché consentono di effettuare numerose potenti operazioni finalizzate alla gestione delle basi di dati.

In primo luogo, bisogna mettere in evidenza i seguenti punti:

- la distinzione tra struttura e dati è realizzata con assoluta chiarezza; è evidente infatti che i valori di un attributo non sono l'attributo, oppure che le righe di una tabella (record contenenti i dati) non sono le colonne (campi ella tabella)
- anche la differenziazione dei tipi di dati può essere spinta fin dove lo si ritiene utile: infatti il tipo di dati si può considerare equivalente al dominio dell'attributo (ad esempio: un attributo definito in modo tale che i suoi valori siano tratti dall'insieme dei numeri interi avrà evidentemente il tipo *numero intero*), che può essere scelto a piacimento.

Non meno importante, anzi essenziale, è il fatto che **un RDBMS può gestire più tabelle**, e quindi i dati che compongono un archivio possono venire separati in più tabelle opportunamente collegate tra di loro; naturalmente questi collegamenti vengono previsti in sede di progettazione della base dati (è il caso di ricordare che vi sono quei semplici programmi, tra cui il più noto è probabilmente il modulo database di Microsoft Works che operano sì su tabelle, ma solo su una per volta, possono ancora essere definiti RDBMS, ma estremamente semplificati, perché usano solo alcune delle operazioni che sono possibili sulle relazioni).

Gli RDBMS possono effettuare sulle tabelle che, ricordiamolo, trattate formalmente sono relazioni nel senso prima definito, tutte le operazioni definite dall'algebra relazionale. Alcune operazioni agiscono solo su una relazione, e quindi sono dette **operazioni unarie**, mentre altre agiscono su due relazioni, e quindi sono dette **operazioni binarie**. Le operazioni relazionali hanno una proprietà molto importante: **il risultato di una operazione relazionale è ancora una relazione**. Gli **argomenti** di una operazione sono le relazioni o la relazione a cui essa si applica.

Le **operazioni unarie** sono le seguenti:

- **Selezione** o **Selezione theta** o **Restrizione** (simbolo: ↓): seleziona le tuple che rispondono ad una data caratteristica (ad esempio in una certa colonna contengono un certo valore, oppure non contengono quel valore, oppure contengono un valore superiore a uno dato)
- **Proiezione** (simbolo: →): restituisce un sottoinsieme degli attributi di una relazione; ad esempio, se di una relazione con gli attributi *titolo, tipo data, data, ISBN* vogliamo trattare solo gli ISBN selezioniamo questo attributo (colonna) con una proiezione

- **Rinominazione** (simbolo: ↑): assegna un nome ad una relazione; è utile per assegnare un nome alle relazioni risultanti dall'esecuzione di operazioni relazionali

Per introdurre le operazioni binarie, bisogna prima osservare che esse sono possibili solo su relazioni **compatibili**. Due relazioni sono compatibili quando hanno la stessa arietà e il dominio dell'attributo i-esimo delle due relazioni è lo stesso.

Le principali[13] **operazioni binarie** sono le seguenti:

- **Unione** (simbolo: ∪): restituisce una relazione che contiene tutte le tuple presenti nei suoi argomenti, con l'eliminazione di quelle duplicate perché presenti in entrambi
- **Differenza insiemistica** (simbolo: -): restituisce le tuple che sono presenti in una relazione ma non in un'altra (come evidente, questa operazione non gode della proprietà commutativa)
- **Prodotto cartesiano** (simbolo: ×): è la relazione costruita prendendo ogni possibile coppia di tuple dai suoi argomenti; la sua struttura sarà la concatenazione degli attributi degli argomenti, ossia verranno prima gli attributi di un argomento e poi quelli dell'altro
- **Intersezione insiemistica** (simbolo: ∩): restituisce una relazione che contiene solo le tuple presenti in entrambi gli argomenti, prese una sola volta ciascuna
- **Join naturale** (simbolo: ⋈): combina un prodotto cartesiano e una selezione; infatti esegue prima il prodotto cartesiano dei suoi argomenti, e poi una selezione sul risultato imponendo l'uguaglianza sugli attributi che compaiono in entrambe le relazioni; infine vengono eliminate le tuple duplicate; evidentemente il join naturale presuppone che gli argomenti abbiano almeno un attributo in comune, altrimenti non si potrebbe effettuare la selezione; **il join naturale ha una importanza fondamentale nell'uso degli RDBMS**, come vedremo in seguito

Ci sono poi altre operazioni che sono dette **operazioni estese**:

- **Proiezione generalizzata** o **Estensione** (simbolo: →, come per la proiezione): si tratta di una proiezione, nella quale la lista degli attributi può consistere in qualunque operazione aritmetica su costanti o attributi; può trattarsi, ad esempio, della differenza tra due attributi dell'argomento, oppure di una certa percentuale del valore di un attributo; si tratta di quelli che i sw RDBMS chiamano spesso *campi calcolati*
- **Aggregazione** è una relazione che risulta dall'applicazione, ad una relazione preesistente, di funzioni di aggregazione come la media o la somma dei valori di un attributo
- **Join esterno**: è una estensione del join che serve per trattare i casi in cui alcune informazioni sono mancanti; vi possono essere infatti, negli attributi comuni agli argomenti, alcuni valori che sono presenti solo in una delle relazioni, e che vanno persi nel join naturale; ci sono tre tipi di join esterno:
    - **Join esterno sinistro:** al risultato del join naturale aggiunge le tuple della relazione di sinistra (cioè del primo argomento del join naturale) assegnando valori *null* agli attributi provenienti dalla relazione di destra
    - **Join esterno destro**: stessa cosa, tranne che tuple aggiunte sono quelle della relazione di destra e gli attributi riempiti con valori *null* sono quelli della relazione di sinistra
    - **Join esterno completo**: effettua sia le operazioni del join esterno sinistro che quelle del join esterno destro

Avevamo osservato che il risultato di una operazione relazione è ancora una relazione. In una terminologia più concreta e vicina a quello che si vedere operando con gli RDBMS, si può dire che il risultato è ancora un tabella, ma non una tabella fisicamente registrata nel database al pari di quelle di partenza, bensì quella che si può definire una *tabella logica*, ossia il risultato di queste operazioni è a tutti gli effetti una tabella con righe, colonne, attributi ecc., tabella che però è derivata da quelle fisicamente presenti nel database (di solito gli

[13] Possono in realtà essere definite ulteriori operazioni, che qui non si trattano specificamente.

RDBMS commerciali permettono di registrare come tabella fisica il risultato di una union o di un join, ma questa è solo una comodità aggiuntiva). Questa tabella poi può servire come partenza per ulteriori operazioni..

Tra le operazioni descritte sopra, il join merita una trattazione particolare. Esso infatti permette di risolvere il problema della duplicazione dei dati, portando quindi a basi dati di struttura più razionale, non ridondante e con dati assai meglio controllati (questo sarà illustrato più ampiamente nel paragrafo sulla normalizzazione). Per illustrare il join ricorriamo ad un esempio, ossia il solito indirizzario che prevede i campi: nome cognome, via, città. Avevamo già osservato che se si utilizza una sola tabella, vi saranno in generale nomi di città che si ripetono molte volte, per cui devono essere inseriti ripetutamente. Questo non solo comporta una perdita di tempo, ma anche molte possibilità di errori: ad esempio errori di digitazione, o inserimenti di forme diverse dello stesso dato. Per rimediare, bisognerà innanzitutto progettare diversamente la base dati. Infatti non useremo più una sola tabella, ma due: la prima, che chiameremo *Nomi*, conterrà i campi nome, cognome e via, mentre la seconda, che chiameremo *Città* conterrà il campo città. Questa seconda tabella conterrà inolt re un campo, che chiameremo *ID*, destinato ad ospitare un identificatore univoco di ciascun record, ad esempio un numero, scelto quindi in modo tale che non vi possano essere due record con lo stesso numero. Anche la tabella *Nomi* avrà bisogno di un ulteriore campo, che chiameremo *Riferimento*, destinato appunto a contenere un riferimento alla tabella *Città*: questo campo sarà dello stesso tipo del campo *ID* della tabella Città, per cui conterrà dati che sono possibili valori di quel campo (ossia numeri, poiché avevamo ipotizzato che gli identificatori univoci fossero numeri). A questo punto è chiaro come funziona il tutto: per indicare che di un indirizzo fa parte una certa città, si riporterà in questo campo il numero che nella tabella *Città* corrisponde al nome desiderato (per esempio, se la città è Genova, e Genova corrisponde al numero 64, si riporterà 64). Il RDBMS poi penserà ad effettuare il join tra le due tabelle, accoppiando a ciascun record di *Nomi* il record di *Città* che contiene, nel campo *ID*, il numero contenuto nel campo *Riferimento* nel record in questione. Il risultato finale sarà una tabella che presenta la città appropriata per ogni indirizzo.

Un esempio sarà utile per chiarire meglio i concetti. Ecco innanzitutto la tabella *Nomi*:

Nome	Cognome	Via	Riferimento
Beppe	Pavoletti	Via Trieste	1
Pippo	Rossi	Via del Campo	64

Ed ecco la tabella *Città*:

ID	Denominazione
1	Acqui Terme
64	Genova
119	New York

Ed ecco il risultato del join:

Nome	Cognome	Via	Riferimento	ID	Denominazione
Beppe	Pavoletti	Via Trieste	1	1	Acqui Terme
Pippo	Rossi	Via del Campo	64	64	Genova

Si tratta, evidentemente, di un **join naturale**, di cui è presentato il risultato completo, nel quale gli attributi *Riferimento* e *ID* saranno probabilmente di scarsa utilità per chi deve esaminare il risultato del join, ma qualsiasi RDBMS consente di effettuare una **proiezione** per selezionare i campi da visualizzare nel risultato, ed eventualmente anche una **ridenominazione** per scegliere, se necessario, un nome più appropriato per questi campi, sicché il risultato del join precedente potrebbe essere visualizzato, ad esempio, in questo modo:

Nome	Cognome	Via	Città

Beppe	Pavoletti	Via Trieste	Acqui Terme
Pippo	Rossi	Via del Campo	Genova

Ricordiamo che la tabella *Città* conteneva anche il record "New York", che però non compare nel risultato del join perché ad esso non viene fatto nessun riferimento dalla tabella *Nomi*. In genere gli RDBMS permettono di effettuare anche un **join esterno** per visualizzare nel risultato di un join anche i record non collegati in una o nell'altra delle tabelle. Ad esempio in questo caso si potrebbe effettuare un **join esterno sinistro** visualizzando anche gli indirizzi senza città (il campo *Città* sarebbe *null*) oppure un **join esterno destro** visualizzando le città per le quali non esistono indirizzi (rimarrebbero vuoti i primi tre campi), ma probabilmente questo non sarebbe molto utile, se non pare fare controlli sulla coerenza dei dati. In altri casi invece è necessario operare in questo modo: ad esempio nel catalogo di una biblioteca normalmente bisogna visualizzare anche i titoli non associati ad alcun autore.

Riprendendo quest'ultimo esempio, è chiaro che il catalogo di una biblioteca sarebbe composto da un insieme di tabelle (come titoli, autori, soggetti ecc.), collegate poi tra loro dagli opportuni join.

Ci si può domandare se l'inserimento dei dati in un RDBMS richieda effettivamente la digitazione di codici univoci che servano da riferimento ai dati contenuti in altre tabelle: in realtà non è così, perché i RDBMS mettono a disposizione tecniche per indicare i collegamenti in modo molto più semplice, ma di questo si parlerà nel capitolo dedicato all'interfaccia utente.

È invece importante osservare che è possibile inserire i dati in una tabella indipendentemente rispetto alle altre. Nell'esempio precedente, si potrebbero inserire tutte insieme le città che interessano, verificando accuratamente la forma dei nomi su atlanti o repertori geografici, per cui gli addetti all'inserimento dei nomi dovrebbero solo scegliere la città appropriata dall'elenco già esistente. Questa possibilità trova molteplici applicazioni, come si vedrà meglio nel seguito.

Il discorso sul join non sarebbe completo se non si illustrasse il fatto che c'è un rapporto tra le tabelle tra cui fare i join e le cardinalità di mappatura previste dal modello E/R. Faremo riferimento a un join tra due tabelle, e per maggior chiarezza utilizzeremo le espressioni *tabella di destra* e *tabella di sinistra* (nell'esempio sopra, la tabella di sinistra sarebbe *Nomi* e quella di destra sarebbe *Città*).

- **associazione uno a uno**: si ha quando ciascun dato (tra quelli interessati al join) della tabella di sinistra corrisponde ad un solo dato della tabella di destra **e viceversa**; un esempio potrebbe essere la corrispondenza tra persone e codici fiscali: ad una persona corrisponde un solo codice fiscale e ad un codice fiscale corrisponde una sola persona; questo tipo di collegamento si rappresenta **con una sola tabella** (ossia in realtà non si fa nessun join); infatti utilizzare due tabelle sarebbe certamente possibile, ma normalmente del tutto inutile, non essendoci comunque alcuna ridondanza di dati
- **associazione uno a molti** (oppure 1 a n): si ha quando a ciascun dato della tabella di sinistra corrisponde ad un solo dato della tabella di destra, ma ad un dato della tabella di destra possono corrispondere molti dati della tabella di sinistra; l'esempio che abbiamo visto prima rientra in questo caso: infatti un indirizzo comprende una sola città, ma ad una città possono corrispondere molti indirizzi; questo tipo di collegamento si rappresenta **con due tabelle**, ciascuna delle quali contiene un capo atto a fare da riferimento reciproco, esattamente come nel caso dell'indirizzario visto prima
- **associazione molti a molti** (oppure m a n): si ha quando a ciascun dato della tabella di sinistra corrispondono molti dati della tabella di destra e viceversa; un tipico esempio è il collegamento tra autori e titoli: un titolo può avere molti autori, e un autore può essere autore di molti titoli; questo tipo di collegamento si rappresenta **con tre tabelle**: la terza tabella avrà almeno due campi, dei quali il primo conterrà un riferimento (tramite un identificatore univoco come visto sopra) ai dati della tabella di sinistra, e il secondo un analogo riferimento alla tabella di destra; quindi se si vuole collegare il titolo identificato dal numero 7 con l'autore identificato dal numero 219, la terza tabella avrà un record che nei suoi due campi

conterrà i dati rispettivamente 7 e 219; se poi al titolo si vuole collegare ancora un altro autore, identificato dal numero 1001, alla terza tabella sarà aggiunto un record che conterrà 7 e 1001

Ovviamente qualsiasi RDBMS degno di questo nome è in grado di gestire tutti i tre tipi di join.

Vediamo infine come un RDBMS può gestire le operazioni di modifica del database e di ricerca. Per quanto riguarda la modifica (aggiunta, cancellazione o variazione di record), l'accesso fisico ai dati, ossia le operazioni di lettura e scrittura sui file, sono procedure specifiche dei vari prodotti, i quali si possono differenziare tra loro quanto a velocità ed affidabilità. Molto importanti sono soprattutto la velocità e l'affidabilità quando si hanno grandi quantità di dati e un gran numero di utenti che lavorano sul database. A proposito della ricerca si può dire che in linea generale un RDBMS puro tenda a risultare più rigido e limitato di quanto ci si potrebbe aspettare, soprattutto se è necessario compiere ricerche su testi: infatti questo tipo di programma non è particolarmente orientato alla ricerca per singole parole, magari con il troncamento e gli operatori di prossimità. Può anche avvenire, soprattutto con applicazioni non molto recenti, che sia possibile solo una ricerca (peraltro rapidissima) per chiave univoca o per campo completo, anche se di solito i programmi effettivamente disponibili cercano di superare questi limiti e anche di avvicinarsi alla potenza di ricerca degli information retrieval (v. seguito).

Nel capitolo delle operazioni di modifica rientra anche tutto ciò che riguarda l'integrità del database, ossia la coerenza dei suoi dati. Ad esempio, nel nostro indirizzario un record della tabella *Nomi* senza riferimento alla città sarebbe palesemente privo di senso, quindi si può desiderare che il programma impedisca del tutto l'immissione di tali record. E che succede, inoltre, se l'operatore, che aveva tutte le migliori intenzioni di completare i dati in modo corretto, non può completare l'immissione perché si blocca il suo computer, o la rete, o per un malfunzionamento di qualunque tipo ?

Ai problemi del primo tipo si può ovviare perché gli RDBMS permettono di definire, in fase di progetto di un database, dei **vincoli di integrità**, ossia di impedire di compiere determinate operazioni che infrangerebbero la coerenza del database. Prendendo ancora come esempio l'indirizzario, osserviamo che avere dei nomi di città cui non è collegato alcun indirizzo non costituisce un problema, per cui si possono tranquillamente cancellare tutti gli indirizzi di una determinata città. Costituisce un problema, invece, avere degli indirizzi senza l'indicazione della città, per cui non si può permettere di cancellare una città senza verificare se vi sono degli indirizzi collegati. Perciò se viene impostato un vincolo di questo genere il RDBMS potrebbe cancellare automaticamente tutti gli indirizzi collegati non appena si cancella una città dalla tabella *Città* oppure (più saggiamente, per evitare cancellazioni dissennate) permettere la cancellazione di un record di questa tabella solo se prima sono stati cancellati tutti i record collegati nella tabella *Nomi*.

Ai problemi del secondo tipo (interruzione anomala delle operazioni) si può ovviare con il controllo delle **transazioni.** Per **transazione** si intende un insieme di operazioni destinate a produrre un certo risultato ben determinato, ad esempio la modifica di un record, oppure la cancellazione di un altro, oppure la creazione di un terzo e così via. Si vede facilmente che, tanto per scegliere uno di questi esempi, la modifica di un record comporta una serie di operazioni come: l'accesso a una tabella, l'individuazione del record, il suo caricamento in memoria, la modifica dei dati, il salvataggio degli stessi in modo permanente, l'aggiornamento degli indici, eventualmente la modifica di altri record nella stessa o in altre tabelle sulla base dei vincoli di integrità. Il controllo delle transazioni prevede che la varie modifiche alla base dati che possono essere generate nel corso di una transazione non vengano scritte permanentemente nel database finché la transazione è ancora in corso. Quando la transazione è conclusa, allora viene data (automaticamente oppure esplicitamente dall'operatore) una istruzione, usualmente denominata **commit**, che registra permanentemente nella base dati tutte le modifiche derivanti dalla transazione. Se però la transazione non si conclude regolarmente, allora viene data (anche in questo caso automaticamente o esplicitamente da parte dell'operatore) un'altra istruzione, usualmente detta **rollback**, che elimina ogni traccia della transazione interrotta, riportando la base dati esattamente nello stato in cui era prima questa iniziasse. Un tipico caso in cui viene eseguito un rollback è

quando il RDBMS riparte dopo essere stato terminato in modo anormale, ad esempio per uno spegnimento del computer.

Queste tecniche di controllo delle transazioni sono implementate pressoché su tutti gli RDBMS di qualche pretesa, anche quelli per PC, ma di solito nell'impiego per così dire casalingo non se ne fa uso. Per contro, esse sono pressoché indispensabili in qualsiasi base dati di livello professionale, e anzi hanno un ruolo assolutamente centrale, tanto che il tipo di elaborazione degli RDBMS viene detta **elaborazione transazionale** (anche in contrapposizione all'**elaborazione analitica** dei sistemi di data warehouse di cui parleremo in seguito). L'elaborazione transazionale è caratterizzata dalla presenza di numerosi utenti che contemporaneamente possono eseguire una grande quantità di operazioni che, in sé considerate sono semplici e atomiche (per esempio interrogare un database per estrarre una riga da una tabella), e devono essere eseguite con grande rapidità ma nello stesso tempo mantenendo l'integrità dei dati.

Quando molti utenti accedono ad una base dati in scrittura, è necessario prendere gli opportuni provvedimenti per mantenere sempre la coerenza dei dati, altrimenti potrebbe avvenire che uno modifica un record e contemporaneamente un altro, sullo stesso record, effettua modifiche incompatibili con quelle del primo. Di solito si ricorre al **blocco** (lock) **dei record**: quando un utente accede ad un record per modificarlo, il motore relazionale o di information retrieval lo blocca, ossia impedisce ad altri utenti di accedere in scrittura allo stesso record.

L'argomento è di tale importanza che su di esso vengono svolte ricerche oltremodo approfondite con tecniche matematiche sofisticate[14], e anche qui vogliamo darne una trattazione un poco più estesa. Notiamo innanzitutto che la gestione della **concorrenza** (cioè l'accesso simultaneo di molti utenti) è un argomento strettamente collegato con la gestione delle transazioni, perché entrambi riguardano il mantenimento dell'integrità e della coerenza dei dati[15].

Le transazioni, per poter dare buoni risultati, devono essere caratterizzate dalle seguenti proprietà, dette **proprietà acide** per il motivo che vedremo subito:

- **atomicità** (**A**tomicity): ogni transazione costituisce un tutto unico indivisibile, per cui o viene eseguita per intero (e quindi chiusa dal commit) oppure non viene eseguita per nulla (ossia, se era iniziata, viene annullata dal rollback
- **coerenza** (**C**onsistency): ogni transazione deve lasciare la base dati in uno stato coerente (ad esempio, in un sistema di gestione dei prestiti della biblioteca, se un libro risulta restituito non può contemporaneamente risultare che un lettore lo ha in prestito)
- **isolamento** (**I**solation): l'esecuzione di una transazione deve essere indipendente dalla esecuzione contemporanea di altre transazioni, ossia diverse transazioni eseguite contemporaneamente devono avere lo stesso risultato di ciascuna transazione eseguita da sola
- **permanenza** (**D**urability): l'effetto di una transazione che eseguito il commit deve essere permanente, e quindi non andare più perso (fatto salvo, si intende, che un'altra transazione successiva può andare a modificare i dati)

Come si vede, l'acronimo dei termini inglesi è la parola ACID, da cui appunto l'espressione proprietà acide.

Senza il controllo di concorrenza però non sarebbe possibile assicurare il mantenimento delle proprietà acide: infatti senza controllo di concorrenza ogni applicazione e ogni utente leggerebbe e scriverebbe nel database senza tener conto delle operazioni degli altri utenti e applicazioni. Anche intuitivamente, si capisce subito che è come quando più persone devono occuparsi dello stesso lavoro o usare gli stessi materiali: se non si mettono

[14] Per una esposizione chiara e al contempo non banale si veda *[Basi 1999]*.

[15] Alcuni RDBMS di alta qualità, come MySQL (http://www.mysql.org/) implementano la gestione della concorrenza ma non quella delle transazioni, e sono indicati soprattutto quando gli accessi in lettura prevalgono su quelli in scrittura; per informazioni su MySQL si veda anche *[Datenbanken 2000]*

d'accordo esplicitamente, o non usano qualche criterio condiviso su come procedere è improbabile che il lavoro vada a buon fine.

Nel campo dei database, la mancanza del controllo di concorrenza dà luogo a diversi inconvenienti che sono stati dettagliatamente analizzati dagli studiosi. Tra questi ne citiamo alcuni:

- **lettura sporca** (*dirty read*): si ha quando una transazione va a leggere una valore risultato di uno stato intermedio di un'altra transazione che però poi fallisce con un abort; in questo caso la prima transazione opera su un valore che in realtà non è stato registrato nel database (vengono violate le proprietà dell'indipendenza, perché la seconda transazione dipende da uno stato intermedio della prima e non dal suo risultato finale, dell'atomicità, per lo stesso motivo, e in fondo anche quella della persistenza, perché viene utilizzato un dato che non è persistente)
- **letture inconsistenti**: se una transazione va a leggere ripetutamente lo stesso dato che nel frattempo viene modificato da un'altra transazione, leggerà valori diversi senza motivo apparente (anche qui vengono violate la proprietà dell'indipendenza e dell'atomicità, e di conseguenza anche quella della consistenza)
- **perdita di aggiornamento** (*lost update*): se due transazioni leggono lo stesso dato e lo modificano, il risultato della prima transazione che scrive il dato modificato va perso perché viene sovrascritto dalla seconda transazione (violate l'indipendenza e l'atomicità, e di conseguenza la persistenza)
- **aggiornamento fantasma** (*ghost update*): se una transazione legge diversi dati sui quali contemporaneamente sta operando un'altra transazione può avvenire che legga alcuni dati già aggiornati dalla seconda transazione, e altri non ancora aggiornati, per cui i risultati delle elaborazioni della prima transazione saranno incoerenti rispetto al contenuto effettivo del database (violate l'indipendenza e l'atomicità, e per conseguenza la consistenza)

È evidente che una base dati in cui si verificano fenomeni di questo genere non è per niente affidabile. Su questo argomento sono stati fatti, nel corso degli anni, studi molto approfonditi, dei quali qui riassumiamo solo alcuni risultati di maggiore importanza dal punto di vista pratico per poter comprendere il funzionamento dei sistemi reali.

Osserviamo innanzitutto che l'ideale a cui si tende è quello della **serializzazione**, il che significa che bisogna fare in modo che le operazioni si svolgano **come se** le transazioni venissero tutte eseguite rigorosamente l'una dopo l'altra e non contemporaneamente. Come abbiamo già osservato, a questo si provvede attraverso l'uso dei lock. In pasrticolare, si distinguono tre primitive:

- **r_lock** o lock in lettura
- **w_lock** o lock in scrittura
- **unlock**, ossia rilascio dei lock

Ogni operazione di lettura deve essere preceduta da un r_lock e seguita da un unlock. Questo lock si dice condiviso, perché sulla stessa risorsa (ad esempio una riga di una tabella) possono essere contemporaneamente essere presenti più lock di questo tipo. Ogni operazione di scrittura invece deve essere preceduta da un w_lock e seguita da un unlock. Questo lock si dice esclusivo perché sulla stessa risorsa non può esserne attivo più di uno (non si può andare in molti a scrivere contemporaneamente sullo stesso record!). Il lock vengono in modo trasparente alle applicazioni: questo significa che il programma applicativo non deve preoccuparsi di acquisire e rilasciare i lock, ma sono le transazioni stesse che li richiedono ad un compnente del RDBMS detto lock manager, che li concede o no a seconda della compatibilità con i lock precedenti.

Sulle transazioni si impone poi la seguente fondamentale restrizione, detta **locking a due fasi** o **2PL** (dall'inglese *two phase locking*):

una transazione, dopo aver rilasciato un lock, non può acquisirne altri.

Pertanto in ogni transazione si distingue una fasce crescente, nella quale la transazione acquisizione i lock, e una fasecalante, nella quale la transazione rilascia i lock.

2.1.1 Il linguaggio SQL

Il linguaggio SQL (Structured Query Language = Linguaggio strutturato di interrogazione) è stato concepito inizialmente dalla IBM ala fine degli anni '70, diventando presto uno standard di fatto nel'ambito degli RDBMS. In seguito SQL diventò uno standard ufficiale per la prima volta nell'ottobre 1986, come standard ANSI X3.135-1986, noto come SQL-86. Questo standard nel 1987 fu poi adottato anche dall'ISO. O standard fu poi perfezionato ed ampliato dall'ANSI nel 1989 (SQL-89), dall'ANSI e dall'ISO congiuntamente nel 1992 (SQL-92) e infine dall'ISO/IEC nel 1999 (SQL3). Nel corso di questo periodo, SQL ha confermato il suo successo diventando di gran lunga il principale linguaggio per la gestione delle basi di dati nell'ambito RDBMS. Esso viene supportato, spesso con alcune differenze tra l'una e l'altra implementazione da tutti i principali RDBMS funzionanti nei più diversi ambienti, ad esempio Access, DB2, Oracle, Ingres, Informix, SQL/DS, Sybase, SQL Server, PostgresSQL, mSQL ecc. Si tratta comunque di tutti i prodotti di alto livello, adatti anche per applicazioni impegnative, mentre possono ancora esservi prodotti concepiti per uso individuale o per piccole applicazioni che hanno un supporto SQL limitato o anche del tutto assente.

SQL non è un linguaggio di programmazione completo concepito per impieghi generali, ma un linguaggio specializzato per la gestione dei database, per cui consente di effettuare tutte le operazioni sulle tabelle, come creazione, eliminazione e modifica di una tabella, modifica dei dati, estrazione di un sottoinsieme delle righe di una tabella selezionato secondo un certo criterio, modifica di un tale sottoinsieme secondo un certo criterio (ad esempio sostituire ogni occorrenza di un certo valore con un altro valore), join, union ecc. Anche se la versione attuale dello standard è alquanto complessa, si può dire che SQL è un linguaggio molto semplice e chiaro rispetto alla sua potenza, e comunque sono certamente semplici e chiare almeno le istruzioni principali.

Non è qui luogo per una illustrazione completa del SQL, per cui limiteremo a qualche accenno[16]. Si può dire che l'istruzione principale sia *Select*, che serve appunto a selezionare dei dati da una o più tabelle. Select accetta numerosi parametri, tra cui in particolare la tabella su cui operare, le colonne (campi) da estrarre e i criteri di selezione delle righe (record). Alcuni esempi che utilizzano le tabelle già viste in precedenza, cioè *Nomi* e *Città*. Negli esempi le parole chiave del SQL sono riportare in maiuscolo solo per chiarezza: ciò però non è obbligatorio. Anche gli a capo sono liberi e qui sono stati inseriti solo per la leggibilità.

Per estrarre tutti i campi di tutti i record dalla tabella *Nomi* useremo l'istruzione

```
SELECT *
FROM Nomi;
```

Per estrarre dalla tabella *Nomi* solo i nomi e i cognomi di quelli che si chiamano Beppe useremo invece la seguente istruzione

```
SELECT Nome,Cognome
FROM Nomi
WHERE Nome = 'Beppe';
```

Per estrarre tutti i record e i campi dalla tabella *Città* ordinandoli per Denominazione ecco l'istruzione adatta:

```
SELECT *
FROM Città
ORDER BY Denominazione;
```

[16] Per esposizioni più complete si veda, ad esempio, *[Datenbanken 1998]* o *[SQL 1991]* (non più molto aggiornato, ma semplice e sintetico), mentre una esposizione molto approfondita si ha in *[SQL-99 1999]*.

Con la seguente istruzione si effettua il join tra le due tabelle (nella prima versione, cioè quella che mostra anche i campi di legame); si noti la sintassi *tabella.colonna* per riferirsi ad una specifica colonna di una specifica tabella:

```
SELECT *
FROM Nomi,Città
WHERE Nomi.Riferimento = Città.Id
ORDER BY Denominazione,Cognome,Nome;
```

Si tratta evidentemente di un join naturale. In realtà l'istruzione riportata ha valore solo indicativo, perché nell'attuale versione di SQL si dovrebbe usare una sintassi un poco diversa che permette di distinguere i vari tipi di join, ma che qui non approfondiamo, non essendo questo documento un manuale di SQL. Si noti anche che specifici prodotti potrebbero avere estensioni proprietarie del linguaggio, o implementarne solo un sottoinsieme, o anche implementare alcune istruzioni in modo non standard.

Molto potenti sono anche le funzioni di aggregazione, come *count()* che restituisce il numero di righe che soddisfano la query, *avg(), sum(), min()* e *max()* che calcolano rispettivamente la media, la somma, il minimo e il massimo dei valori numerici di una colonna. Ad esempio

```
SELECT COUNT(nome)
FROM nomi
WHERE nome = 'baciccia';
```

restituisce il numero di righe in cui l'attributo *nome* ha il valore *baciccia*. Da ricordare la clausola *group by*:

```
SELECT nome, COUNT(nome)
FROM nomi
GROUP BY nome;
```

ci informa sul numero di righe in cui compare ciascun nome.

Un altro elemento importante del linguaggio SQL sono le **viste** o **views**, create attraverso l'istruzione `CREATE VIEW`. Si tratta del risultato di una select che viene presentato come se fosse una tabella autonoma, ed è utilizzabile come tale. Le viste servono per presentare i dati in modo più chiaro e adatto alle esigenze degli utenti, e per limitare l'accesso di questi ultimi solo a determinate parti del database. Le viste infatti permettono di limitare l'accesso non solo a certe tabelle, ma anche a certe righe di una tabella, quelle appunto che sono il risultato di una select. In alcuni prodotti, come Microsoft Access, le viste apparentemente non esistono, ma solo perché non sono chiamate con questo nome. In Access le query di selezione non sono altro che viste, poiché si tratta del risultato dell'esecuzione di una istruzione SQL presentato come una tabella[17] e, anzi, quello che manca è la possibilità di eseguire in modo interattivo delle istruzioni SQL.

SQL mette a disposizione numerosi tipi di dati, come testo, numeri interi, numeri in virgola mobile, date ecc. Merita richiamare l'attenzione sul tipo di dati *array* (presente solo nelle versioni più recenti), che è una lista ordinata di valori. Una colonna definita come *array* non conterrà, in una riga, un singolo valore, ma una lista di valori. Questo crea dei problemi con la teoria della normalizzazione, che sarà illustrata poco oltre: una simile colonna, strettamente parlando, non rispetta il requisito minimo dell'atomicità dei valori, e quindi non è neppure in prima forma normale, e quindi non potrebbe avere alcun ruolo in un database relazionale! È quindi alquanto singolare che proprio SQL, il linguaggio per eccellenza associato agli RDBMS permetta queste cose, per cui il ruolo di questo tipo di dati e l'opportunità di usarlo sono ancora soggetti a discussione.

[17] Queste viste in Access vengono però create senza usare esplicitamente l'istruzione `CREATE VIEW`.

Ci sono diversi modi per utilizzare SQL con un RDBMS:

- **modalità interattiva**, che consiste nel digitare direttamente le istruzioni SQL, che vengono immediatamente eseguite dal RDBMS; sostanzialmente rientra in questa modalità anche la possibilità di registrare le stesse istruzioni SQL che verrebbero digitate in modalità interattiva in modo che poi in qualsiasi momento esse possano venire richiamate per l'esecuzione (molti RDBMS, soprattutto quelli per PC, mettono a disposizione delle schermate con cui comporre la query in modo facilitato, senza neppure dover conoscere il SQL; la query composta in tal modo viene poi tradotta in SQL ad uso del RDBMS)
- **embedded SQL**, che consiste nell'incorporare le istruzioni SQL in programmi scritti in un linguaggio di programmazione generale, detto *linguaggio host*,, di solito il C o il Cobol, ma anche altri[18], che consente un controllo più completo di tutte le operazioni; i programmi scritti in questo modo vengono innanzitutto elaborati da un precompilatore, che traduce tutte le istruzioni SQL incomporate nel programma in una forma accettabile dal compilatore vero e proprio, che produrrà un codice oggetto in cui le orignarie istruzioni SQL sono diventate riferimenti esterni; successivamente il linker userà una apposita libreria (che normalmente fa parte del RDBMS e non dell'ambiente di sviluppo del linguaggio host) per risolvere questi riferimenti; questa è stata finora la tecnica più usata per progetti di un certo impegno
- **moduli SQL**, tecnica fornita da SQL3; consiste in estensioni del linguaggio che permettono di dichiarare variabili, assegnar loro dei valori, controllare il flusso dell'esecuzione e quindi scrivere moduli di programma completi e autosufficienti, senza bisogno di un linguaggio host, il quale peraltro è necessario per richiamare i moduli
- **SQL/CLI (Call Level Interface)**; anche questa tecnica è messa a disposizione da SQL3,e prevede l'uso di un linguaggio host, il quale richiama una libreria SQL esterna (che normalmente fa parte del RDBMS e non dell'ambiente di sviluppo del linguaggio host) tramite chiamate a funzioni delle quali le istruzioni SQL da eseguire sono parametri; si noti la differenza con l'embedded SQL: là le istruzioni SQL erano direttamente inserite tra le istruzioni del linguaggio, qui invece compariono come parametri delle funzioni di libreria; secondo alcuni (ad esempio gli autori di *[SQL-99 1999]*) questa tecnica finirà per sostituire l'embedded SQL; si noti che le librerie di cui si parla saranno delle DLL in Windows e degli shared objects in Unix

A seconda del prodotto che si usa, si possono avere a disposizione una o più di queste tecniche.

2.2 Normalizzazione

Quanto detto finora non ci permette di rispondere a una domanda fondamentale per la corretta progettazione delle basi dati: con quale criterio si devono distribuire i dati tra le diverse tabelle? Infatti finora abbiamo considerato come date le tabelle su cui operare, ma non sappiamo perché sono state definite proprio in un certo modo e non in un altro.

Consideriamo l'esempio del paragrafo precedente: se l'attributo *Via* fosse stato assegnato alla tabella *Città*, evidentemente sarebbe stato necessario prevedere, in questa tabella, una riga per ciascuna via, e poiché una stessa città ha molte vie, i nomi delle città avrebbero potuto trovarsi ripetuti molte volte nella tabella, con grande duplicazione di dati. In un caso semplice come questo, è sufficiente un po' di esperienza nella progettazione delle basi dati per evitare questi errori, ma tutto l'argomento è stato ampiamente studiato sul piano teorico, fino a darne una esposizione completa e rigorosa.

In particolare, si intende per **decomposizione** la suddivisione dei dati tra le diverse tabelle, mentre la **normalizzazione** è la tecnica per ottenere una buona decomposizione, tale cioè da evitare il più possibile duplicazioni di dati e da facilitare le operazioni che si devono compiere sul database. La decomposizione deve avere alcune proprietà, in particolare essere **senza perdita di informazioni** (*lossless decomposition*), ossia essere tale che, attraverso i join opportuni, si possano ricostruire tutte le informazioni che sono oggetto

[18] SQL richiede il supporto per almeno uno dei seguenti linguaggi: ADA, C, Cobol, Fortran, Mumps, Pascal, PL/I, ma in realtà pochi prodotti supportano ADA, Mumps e PL/I. È inoltre possibile che specifici prodotti supportino ulteriori linguaggi

dell'elaborazione, mentre in una decomposizione con perdita si avranno o tuple in meno o tuple in più. Inoltre la decomposizione deve **preservare le dipendenze**, cioè essere tale che tra i dati decomposti vi siano tutte le dipendenze che devono sussistere tra i dati oggetto dell'elaborazione (le dipendenze verranno illustrate sotto).

Uno schema di relazione si decompone senza perdita su due relazioni se l'insieme degli attributi comuni alle due relazioni è chiave per almeno una delle relazioni decomposte[19]. Supponiamo ad esempio di voler rappresentare la situazione dei prestiti in una biblioteca tramite uno schema di relazione sugli attributi: *inventario, titolo, nome, cognome, codice fiscale*. Questo schema presenza molte ridondanze, perché se un lettore ha in prestito più libri i suoi dati compariranno più volte e se una edizione è presente in più copie il titolo comparirà più volte. Proviamo allora a decomporlo in due relazioni, una su *inventario, titolo, cognome* e l'altra su *nome, cognome, codice fiscale*. Le due relazioni potrebbero avere un contenuto del genere:

Inventario	Titolo	Cognome
100	I promessi sposi	Rossi
334	I Buddembrook	Bianchi
256	Discorso sul metodo	Pavoletti
1245	American psycho	Verdi
667	La Bibbia	Rossi

Cognome	Nome	Codice fiscale
Rossi	Baciccia	xxxxxxxxxxxxxxxxxx
Bianchi	Gino	xxxxxxxxxxxxxxxxxx
Pavoletti	Beppe	xxxxxxxxxxxxxxxxxx
Verdi	Giuseppe	xxxxxxxxxxxxxxxxxx
Rossi	Giobatta	xxxxxxxxxxxxxxxxxx

Se ora vogliamo effettuare un join, abbiamo a disposizione solo l'attributo *cognome*, che è l'unico comune alle due relazioni, ma si vede immediatamente che con un join su *cognome* sia La Bibbia che I Promessi Sposi risulteranno in prestito tanto a Rossi Baciccia quanto a Rossi Giobatta, il che non solo non è affatto ovvio in generale, ma nel caso presente è impossibile, visto che a ciascuno dei due titoli è associato un solo inventario, per cui ne esiste una sola copia che non può essere in prestito contemporaneamente a due lettori. Non è difficile capire che in questa decomposizione la condizione che abbiamo illustrato sopra non è verificata, perché *cognome* non è chiave in nessuna delle due relazioni. Intutivamente, si può dire che la decomposizione non ci mette in condizioni di distinguere tra lettori diversi con lo stesso cognome, e quindi ci porta a concludere che questi lettori abbiano tutti in prestito le stesse cose. Una decomposizione corretta dal punto di vista della perdita sarebbe inevce stata quella su *titolo, inventario, codice fiscale* e *codice fiscale, cognome, nome* perché il codice fiscale, essendo per definizione univoco, è chiave della seconda relazione.

Per quanto riguarda invece la conservazione delle dipendenze, bisogna evitare che in seguito alla decomposizione scompaiano delle dipendenze che valevano nello schema originale. Qui la regola è che ogni dipendenza funzionale dello schema originario deve coinvolgere attributi che compaiano tutti insieme in uno degli schemi risultati dalla decomposizione. La decomposizione dell'esempio riportato sopra non conserva neppure le dipendenze. Infatti nello schema originario cè una dipendenza da *inventario* a *nome, cognome*, perché ogni esemplare può essere in prestito a una sola persona per volta, mentre dopo la decomposizione questo non vale più.

[19] Si noti che questa è una condizione sufficiente ma non necessaria, poiché possono esserci decomposizioni senza perdita che non la rispettano. Tuttavia è una condizione di notevole importanza pratica, perché quando è verificata garantisce che la decomposizione sia senza perdita.

Veniamo ora finalmente a trattare delle **dipendenze**, che sono anche la premessa per la teoria della normalizzazione. Si distinguono tre tipi di dipendenze, che esporremo di seguito, cioè **dipendenze funzionali**, **dipendenze multivalore** e **dipendenze di join**. Nella esposizione utilizzeremo le lettere greche, come α e β, per indicare gli attributi e la sintassi $t_1[\alpha]$ per indicare il valore dell'attributo α della tupla t_1.

- **Dipendenze funzionali**. Siano dati gli attributi α e β della relazione *r*. Si ha una dipendenza funzionale tra α e β se per ogni copia di tuple in *r* si ha che se $t_1[\alpha]=t_2[\alpha]$ allora $t_1[\beta]=t_2[\beta]$. La dipendenza funzionale ha come simbolo: $\alpha \rightarrow \beta$[20]. Ad esempio, nell'ambito dei dati bibliografici si ha dipendenza funzionale tra ISBN e titolo proprio, poiché non può avvenire che allo stesso ISBN corrispondano titoli diversi. Alcune dipendenze funzionali possono essere implicate da altre, ad esempio se $\alpha \rightarrow \beta$ e $\beta \rightarrow \gamma$ allora $\alpha \rightarrow \gamma$. L'insieme di tutte le dipendenze funzionali implicate da un certo insieme F di tali dipendenze è detto **chiusura** di questo insieme, ed è denotata da F^+. Se è possibile rimuovere un attributo da una dipendenza funzionale senza modificare la chiusura dell'insieme di dipendenze, questo attributo è detto **estraneo**. Se per un insieme di dipendenze F, esiste un insieme di dipendenze F_C tale che:
  - ◊ nessuna dipendenza funzionale in F_C contiene attributi estranei
  - ◊ ogni lato sinistro di una dipendenza in F_C è unico
  - ◊ F implica logicamente tutte le dipendenze in F_C e F_C implica logicamente tutte le dipendenze in F

  allora F_C è una **copertura canonica** per F, ed ha la stessa chiusura di F. La copertura canonica è quindi una sorta di insieme minimale di dipendenze, nel quale non vi sono elementi che possono essere rimossi.
- **Dipendenze multivalore.** Le dipendenze multivalore, a differenza delle differenze funzionali, non vietano l'esistenza di determinate tuple, ma impongono invece l'esistenza di alcune tuple data l'esistenza di alcune altre. Tra gli attributi α e β, che fanno parte dello schema di relazione (cioè dell'insieme di attributi) R esiste una dipendenza multivalore se per ogni coppia di tuple tali che $t_1[\alpha]=t_2[\alpha]$ esistono le tuple t_3 e t_4 con le seguenti proprietà: $t_1[\alpha]=t_2[\alpha]=t_3[\alpha]=t_4[\alpha]$, $t_3[\beta]=t_1[\beta]$,$t_3[R - \beta]=t_2[R - \beta]$, $t_4[\beta]=t_2[\beta]$,$t_4[R - \beta]=t_1[R - \beta]$. Intuitivamente, l'esistenza di una dipendenza multivalore significa che l'associazione tra α e β è indipendente da quella tra α e R-β, il che si ha quando un attributo ha rapporti indipendenti con altri due insiemi di attributi. Immaginiamo, ad esempio, che il sistema di prestito di un software per gestione biblioteche contenga una tabella con i seguenti dati: inventario del libro, nome del lettore, indirizzo del lettore, e che consenta di indicare, se necessario, più indirizzi per lo stesso lettore. Ogni volta che un lettore con più indirizzi, ad esempio due, ha un libro in prestito, bisognerà ripetere due volte sia il nome che l'inventario: si tratta di una conseguenza indesiderata che dimostra che la tabella non è ancora ben normalizzata, e che deriva dal fatto che i rapporti tra lettore ed indirizzo sono indipendenti da quelli tra lettore e inventario del documento preso in prestito.
- **Dipendenze di join**. Sia R uno schema di relazione, e sia R_1, R_2, ..., R_n una sua decomposizione senza perdita, per cui $R = R_1 \bowtie R_2 \bowtie ... \bowtie R_n$. Diciamo che una relazione *r*(R), cioè una relazione avente lo schema R soddisfa la dipendenza di join *(R_1, R_2, ..., R_n) se *r* è uguale al join naturale delle sue proiezioni della forma $\pi_{R_n}(r)$, cioè il join tra $\pi_{R_1}(r)$, $\pi_{R_2}(r)$ e così via. Una dipendenza di join è **banale** se uno degli R_i è R stesso. Intuitivamente, si può dire che se la dipendenza multivalore stabilisce l'indipendenza tra coppie di associazioni, la dipendenza di join stabilisce l'indipendenza tra tutti i membri di un insieme di associazioni, e quindi è una estensione della dipendenza multivalore.

Armati di questi concetti, possiamo cominciare ad elencare le diverse forme normali che sono state definite e studiate. L'ordine in cui sono introdotte non è casuale, ma corrisponde a livelli via via superiori di

[20] Lo stesso simbolo è usato anche il logica per denotare l'implicazione materiale. La dipendenza funzionale **non** è però una implicazione materiale, perché altrimenti sussisterebbe anche quando il primo membro dell'espressione, cioè $t_1[\alpha]=t_2[\alpha]$, è falso. Non è neppure un bicondizionale, perché in tal caso sussisterebbe anche quando entrambi i membri dell'esrpressione sono falsi.

normalizzazione, e quindi ad una decomposizione più perfetta. Inoltre ogni livello include i precedenti: ad esempio, ogni schema di relazione in quarta forma normale è anche in terza forma normale ecc. Quale forma normale utilizzare di fatto in uno specifico progetto dipende peraltro dalle caratteristiche dei dati da trattare e anche da considerazioni pratiche: le strutture più normalizzate in genere sono più complicate da gestire, per cui può essere che in certi casi, ad esempio per dati di importanza marginale, questa maggiore complicazione non bilanci i vantaggi che si ottengono. Nel seguito si parlerà di relazioni o schemi di relazione in una certa forma normale. La struttura di un database si definisce in un certa normale se tutte le relazioni che la compongono sono in quella forma normale: per esempio se un database è composto solo da relazioni in BCNF si dirà che il database è in BCNF, ma se una di queste relazioni fosse invece in 3NF allora si dirà che il database è in 3NF (anche le relazioni in BCNF sono in 3NF).

1. **Prima forma normale (1NF)**. Uno schema di relazione è in 1NF se rispetta il requisito dell'**atomicità** dei dati, ossia se gli elementi del dominio di tutti gli attributi sono considerati unità indivisibili. L'atomicità dipende in certa misura dai requisiti di progetto: un dato può essere considerato atomico in un certo contesto, ma in un altro può essere opportuno suddividerlo in diverse componenti. Ad esempio, di solito un attributo di tipo *data* è considerato atomico, ma può darsi che per qualche scopo sia utile definire *giorno*, *mese* e *anno* come attributi distinti. Comunque ci sono dei casi in cui è molto difficile considerare atomici certi dati: si pensi ad un database bibliografico in cui sia stato definito un unico attributo per contenere titolo, autore, soggetto, descrizione fisica, inventario, numero standard e prezzo. Uno schema che non rispetti neppure la 1NF è inutilizzabile, almeno nei termini della teoria delle basi dati relazionali, e può trovarsi in archivi progettati da persone inesperte e comunque prive di qualunque cognizione, anche intuitiva, di cosa sia e come si gestisca una base di dati. Può anche essere l'effetto del tentativo, più o meno ingenuo, di utilizzare un RDBMS come se fosse un information retrieval. Talvolta per uno schema può essere sufficiente conformarsi solo alla 1NF: è il caso di uno schema con un solo attributo, nel quale evidentemente non è possibile alcuna decomposizione. A parte questo caso, però, la conformità alla 1NF non è assolutamente sufficiente per una buona progettazione.
2. **Seconda forma normale (2NF)**. È considerata di interesse puramente storico, nel senso che non viene più ritenuto di alcun valore pratico avere uno schema in 2NF. Essa fa riferimento al concetto di **dipendenza parziale**, che si ha quando per una dipendenza funzionale $\alpha \rightarrow \beta$ esiste un sottoinsieme proprio γ di α tale che $\gamma \rightarrow \beta$. Uno schema di relazione è in 2NF se ognuno dei suoi attributi risponde ai seguenti criteri: appare in una chiave candidata e non è parzialmente dipendente da una chiave candidata. Si può anche esprimere dicendo che un attributo che non è chiave deve dipendere da tutti i campi della chiave primaria, e implica che due o più tabelle non possono avere la stessa chiave primaria.
3. **Terza forma normale (3NF)**. Uno schema è in 3NF rispetto ad un insieme F di dipendenze se per tutte le dipendenze funzionali in F^+ vale almeno una delle seguenti condizioni: $\alpha \rightarrow \beta$ è una dipendenza banale (cioè β è un sottoinsieme di α), oppure α è una superchiave per lo schema, oppure ogni attributo in $\beta - \alpha$ è contenuto in una chiave candidata. Detto in altri termini, per ogni dipendenza non banale $\alpha \rightarrow \beta$, α contiene una chiave oppure ogni attributo in β è contenuto in almeno una chiave (si noti che le condizioni possono anche verificarsi entrambe). Si può anche esprimere dicendo che nessun attributo che non è una chiave dipende da un altro attributo che non è chiave. Per fare un esempio supponiamo che in un sistema bibliotecario ogni biblioteca abbia individuato un diverso fornitore di libri per ciascuna classe della CDD. Questo può essere espresso tramite lo schema in 3NF comprendente gli attributi *biblioteca*, *CDD*, *fornitore*, che contiene le seguenti dipendenze: da *CDD* a *fornitore* e da *biblioteca* *fornitore* a *CDD*.
4. **Forma normale di Boyce-Codd (BCNF = *Boyce-Codd Normal Form*)**. Uno schema è in BCNF rispetto ad un insieme F di dipendenze se per tutte le dipendenze funzionali in F^+ vale almeno una delle seguenti condizioni: $\alpha \rightarrow \beta$ è una dipendenza banale (cioè β è un sottoinsieme di α), oppure α è una superchiave per lo schema. In termini più semplici per ogni dipendenza non banale $\alpha \rightarrow \beta$, α contiene una chiave per la relazione. Come si può vedere, è ammessa una condizione in meno rispetto alla 3NF, per cui la BCNF è più restrittiva rispetto alla 3NF. Un esempio è una relazione in cui vi sia una dipendenza tra un attributo e l'insieme di tutti gli altri, come avviene se un attributo è l'ISBN e gli altri sono gli elementi della descrizione bibliografica. Se a questa relazione aggiungessimo come ulteriore attributo il numero di

inventario, essa non sarebbe più in BCNF, perché non c'è dipendenza funzionale tra ISBN e inventario (non vale che se per due tuple è uguale l'ISBN sia uguale il numero di inventario). **La 3NF e la BCNF sono le forme normali di maggiore uso pratico, nel senso che nella maggior parte dei casi reali di progettazione di basi di dati non si cerca di spingere la normalizzazione oltre queste forme.**

5. **Quarta forma normale (4NF)**. La 4NF utilizza la nozione di dipendenza multivalore illustrata in precedenza. Diciamo innanzitutto che una dipendenza multivalore è **banale** se è soddisfatta da tutte le relazioni su uno schema R. Uno schema è in 4NF rispetto ad un insieme F di dipendenze funzionali e multivalore se per tutte le dipendenze multivalore in F^+ vale almeno una delle seguenti condizioni: la dipendenza multivalore da α a β è banale, oppure α è una superchiave per lo schema. In termini più intuitivi, si può dire che la 4NF proibisce le dipendenze uno a molti multiple e indipendenti (cioè in cui gli attributi dipendenti sono indipendenti tra loro) tra campi della chiave primaria e campi che non sono chiave. Per fare un esempio, consideriamo le schema dato come esempio nella esposizione della dipendenza multivalore, cioè uno schema, che potrebbe utilizzato in un software di prestito bibliotecario, con i seguenti attributi: *inventario del libro*, *nome del lettore*, *indirizzo del lettore*, e che inoltre deve consentire di indicare, se necessario, più indirizzi per lo stesso lettore. Questo schema non è ben normalizzato, perché se un lettore ha più indirizzi bisogna ripetere per ciascun indirizzi tutti i dati del lettore e l'inventario del libro. Poiché lo schema contiene dipendenze multivalore si può decomporre in due schemi, entrambi in 4NF: il primo conterrà *inventario* e *nome*, il secondo *nome* e *indirizzo*.
6. **Forma normale di join di proiezione (PJNF = *Project-Join Normal Form*)** detta anche **Quinta forma normale (5NF)**. Uno schema R è in PJNF rispetto ad un insieme F di dipendenze funzionali, multivalore e di join se per tutte le dipendenze di join in F^+ vale almeno una delle seguenti condizioni: la dipendenza di join è banale, oppure ogni decomposizione R_n dello schema coinvolta nel join è una superchiave di R. In termini intuitivi, questa forma richiede di eliminare ogni ridondanza scomponendo lo schema in parti più picccole possibili. Per fare il solito esempio bibliotecario, consideriamo un software di prestito da impiegare in un sistema bibliotecario, nel quale è necessario indicare quale biblioteca del sistema ha fatto un certo prestito. Potremmo avere, per questo scopo, una relazione sullo schema: *biblioteca, lettore, inventario, ISBN*. Questo schema può essere decomposto in tre schemi in PJNF, tra i quali vale una dipendenza di join che permette di ricostruire l'insieme delle informazioni. Gli schemi sono *(inventario, lettore), (inventario, biblioteca), (inventario, ISBN)*. Questa decomposizione è abbastanza intuitiva: basta infatti osservare che un inventario identifica la biblioteca che l'ha dato in prestito, il lettore che l'ha preso in prestito, e l'ISBN del documento cui è assegnato l'inventario. La PJNF non è molto utilizzata, perché determina una struttura di database assai complessa, ed è ancora oggetto di ricerche perché le sue proprietà non sono ancora state completamente individuate.
7. **Forma normale di chiave di dominio (DKNF = *Domain-Key Normal Form*)**. Questa forma normale presuppone innanzitutto le nozioni di **dominio** e di **chiave** già introdotte in precedenza. Più in dettaglio, se A è un attributo e **dom** un insieme di valori, la **dichiarazione di dominio** A $\in$ **dom** richiede che i valori di A in tutte le tuple siano membri di **dom**. Se K è un attributo che fa parte dello schema R, la **dichiarazione di chiave key**(K) richiede che K sia una superchiave per R. Tutte le dichiarazione di chiave sono dipendenze funzionali, ma non tutte le dipendenze funzionali sono dichiarazioni di chiave. Introduciamo inoltre la nozione di **restrizione generale** (*general constraint*), che è un predicato valido per tutte le relazioni su un dato schema. Un tale predicato viene espresso in una qualche forma su cui ci si accordi, come la logica del primo ordine, per cui può avere una forma come: *per tutti gli attributi che godono della proprietà x, vale anche la proprietà y* (formalmente: *" A x(A) É y(A)*, dove A indica un attributo qualsiasi). Le dipendenze sono restrizioni generali , ma non tutte le restrizioni generali sono dipendenze. Un esempio di restrizione generale, per uno schema che prevede gli attributi *ISBN* e *codice paese di pubblicazione* potrebbe essere: *se le prime due cifre di <ISBN> sono = 88 allora < codice paese di pubblicazione> = IT*. Lo scopo per cui è utile progettare database in DKNF è quello di permettere di testare le restrizioni generali solo in termini di restrizioni di chiave e restrizioni di dominio. In termini più rigorosi, la cosa si può esprimere come segue: sia **D** un insieme di restrizioni di dominio e **K** un insieme di restrizioni di chiave per lo schema R. Sia **G** l'insieme delle restrizioni generali per R. Lo schema R è in DKNF se **D** ⊨ **K** implica logicamente **G**, ossia tutte le restrizioni generali - e quindi anche le dipendenze -

che valgono per lo schema si possono derivare dall'unione delle dichiarazioni di dominio (che sono restrizioni sul dominio) e delle dichiarazioni di chiave (che sono dipendenze funzionali). Possiamo quindi dire che invece di uno schema *ISBN, codice paese di pubblicazione* potremmo avere un insieme di schemi in DKNF per ognuno dei quali valgono le restrizioni generali: *la prima parte dell'ISBN è x* e *il codice di paese è y*, ad esempio: se la prima parte dell'ISBN è 88 il codice di paese è IT, se la prima parte è 3 il codice di paese è DE, e cosi' via.

2.3 Applicazioni dei RDBMS

Come si può facilmente intuire, gli RDBMS vengono utilizzati nei casi in cui i dati si prestano ad essere rappresentati sotto forma di tabella, ossia in cui c'è un certo numero di elementi di base, atomici, che si ripetono con certe combinazioni. Si pensi ad esempi quali *Nome, Cognome, Indirizzo*, oppure *Codice prodotto, Nome prodotto, Descrizione prodotto, Prezzo*, oppure ancora *Identificativo cliente, Identificativo prodotto, Numero fattura, Data, Prezzo*. Si noti che il considerare atomico o no un elemento spesso non è un giudizio assoluto, ma dipende dall'uso che se ne vuole fare: ad esempio in un contesto un nome di persona potrebbe essere considerato un tutt'uno, mentre in un altro potrebbe essere necessario considerare separatamente il nome e il cognome. Negli esempi fatti poc'anzi c'è un dato che abbiamo chiamato *Descrizione prodotto* e che pone problemi interessanti. Ci possiamo infatti chiedere: quale genere di descrizione vogliamo ? una formula sintetica e standardizzata, oppure una descrizione libera e più ampia ? Inoltre: ci interessa poter ricercare le singole parole della descrizione ? Torneremo in seguito su questi problemi, che investono direttamente i database bibliografici.

In concreto quindi gli RDBMS vengono impiegati in primo luogo per attività amministrative, quali contabilità fatturazione gestione magazzino, gestione personale, gestione clienti e simili (non a caso di solito nei manuali di questi prodotti ci sono esempi come quelli citati), mentre sono meno utilizzati negli impieghi che potremmo definire di tipo documentario ossia dove bisogna gestire testi di lunghezza variabile e magari non tutto strutturati, e dove le ricerche sono poco prevedibili e orientate soprattutto al testo. Infatti gli RDBMS sono tipicamente orientati a ricercare con la massima velocità un elemento in una colonna di tabella piuttosto che combinazioni di parole in un testo.

Gli RDBMS sono poi particolarmente efficienti quando è necessario effettuare dei controlli rigorosi sull'immissione dei dati: infatti, grazie al meccanismo del join, è sufficiente mantenere una tabella con i dati da controllare, e tutti gli utenti trarranno poi i dati da questa tabella. A seconda del contesto poi, si può impedire all'utente di aggiungere dati a questa tabella, oppure permettergli di aggiungerne solo dopo avere effettuato una ricerca per verificare se il dato esiste già (come avviene ad esempio in SBN).

Ci sono anche impieghi (ad esempio i cataloghi delle biblioteche) in cui si possono utilizzare con successo sia RDBMS che information retrieval (di cui si parlerà nell'apposito capitolo).

Si deve anche ricordare che i prodotti commerciali tendono ad espandere sempre più le funzionalità degli RDBMS, per cui può essere che vengano implementate funzionalità che tradizionalmente sono piuttosto tipiche degli information retrieval (e viceversa).

3. INFORMATION RETRIEVAL

Gli *information retrieval*, detti anche IRS (Information Retrieval Systems) sono programmi in certo qual modo complementari agli RDBMS, nel senso che gestiscono quei tipi di basi dati per i quali gli RDBMS sono meno indicati, anche se in entrambi i campi vi sono prodotti piuttosto ibridi. Tra i principali IRS commerciali citiamo il classico Stairs della IBM, e poi molti prodotti più moderni come CDS-ISIS, Fulcrum, Basis, Highway.

Manca una definizione degli IRS così rigorosa come quella degli RDBMS, comunque si può dire che in linea generale gli IRS sono programmi per la gestione di basi dati testuali, e soprattutto - come dice il loro stesso nome - per la **ricerca** di informazioni in basi dati testuali.

Le basi dati testuali sono quelle in cui esiste un unico tipo di dati, e cioè appunto il **testo**, ossia una successione di caratteri alfabetici o numerici. Naturalmente una base dati testuale può contenere quelli che **per noi** sono numeri o date, ma che per il programma sono semplicemente particolari successioni di caratteri. Non va dimenticato infatti che non vi sono caratteri alfabetici, ma anche caratteri numerici, per cui un testo può essere una successione da caratteri come "abcdxyz" o anche una successione di caratteri come "756334" o ancora una come "10-6-1996". La conseguenza di questo è che un IRS "puro" non è in grado di compiere operazioni matematiche né operazioni su date (come calcolare quanti giorni intercorrono tra due date) e neppure ordinare dei dati in ordine numerico o di data (ad esempio, ordinerebbe i numeri non in ordine di grandezza ma carattere per carattere, dando risultati come: 0004456, 1, 126, 1234, 21, 35556, 78).

L'altro punto principale è che gli IRS sono orientati molto più alla ricerca delle informazioni che alla loro produzione, al contrario quindi di quanto avviene con gli RDBMS. Infatti in generale essi consentono di effettuare ricerche sofisticatissime, mentre possono incontrare difficoltà ad effettuare controlli in fase di produzione dei dati.

In qualche caso, le funzionalità di produzione dei dati potrebbero addirittura mancare o avere un ruolo del tutto secondario. Per comprendere questo bisogna però introdurre una distinzione importantissima: quella da basi dati testuali **non strutturate** e basi dati testuali **strutturate**. Le prime sono semplicemente testi liberi come quello di una lettera, di una relazione, di una poesia o di un romanzo. Un IRS orientato alle basi dati non strutturate permette di fare ricerche su questi testi, ma è di scarsa utilità che permetta anche di produrli, visto che di programmi atti alla produzione di testi ce ne sono infiniti. Gli IRS di questo genere possono essere in grado di effettuare ricerche su vari formati di dati testuali, ad esempio testo ASCII, Winword, Wordstar, Wordperfect ecc. (anzi, i programmi di scrittura includono sempre alcune funzionalità di information retrieval, spesso molto elementari ma in alcuni casi anche abbastanza sofisticate). Questi programmi però non ci interessano, e quindi non ne parleremo più oltre, perché sono evidentemente inadatti a qualsiasi impiego specificamente bibliotecario[21]. Ben più interessanti per i nostri scopi sono invece gli IRS che gestiscono basi dati testuali strutturate, nelle quali cioè il testo viene inserito in una struttura predefinita di campi e sottocampi. I sottocampi, entità sconosciuta agli RDBMS, sono parti in cui viene ulteriormente suddiviso un campo, allo scopo di permettere di far riferimento, a seconda delle esigenze al campo intero oppure ad una sua parte specifica (per fare un esempio concreto nel popolare database *Teca*, basato sull'IRS CDS-ISIS, il campo *Titolo* è suddiviso nei sottocampi *Titolo proprio, indicazione generale del materiale Complemento del titolo* ecc.). Osserviamo che negli IRS vi è l'usanza di indicare i campi con numeri piuttosto che con nomi, come avviene negli RDBMS, ma si tratta appunto di una pura e semplice usanza, tanto che in alcuni IRS i campi vengono invece designati con nomi. Gli IRS per la gestione di basi dati testuali strutturate devono prevedere qualche funzionalità di inserimento dei dati, perché questo inserimento deve avvenire nell'ambito di una certa struttura, e non sotto forma di generico testo libero. Queste funzionalità possono essere più o meno sviluppate. In particolare, può avvenire che vi siano degli IRS destinati principalmente ad importare, tramite appositi formati di scambio (sull'argomento v. il capitolo 5), dati prodotti in origine con altri software, e nei quali perciò le funzionalità native di inserimento dei dati sono scarsamente significative (mentre magari vi sono sofisticate funzionalità di importazione e conversione).

Al fine di evitare equivoci, sarà bene chiarire ulteriormente che una base dati testuale strutturata **non è affatto** una tabella di RDBMS, ma è invece un insieme di stringhe (il termine *stringa* indica una sequenza di caratteri) contrassegnate da appositi marcatori che ne individuano la posizione nella struttura. Da questo consegue che in un RDBMS i campi sono spesso a **lunghezza fissa**, per cui occupano sempre lo stesso

[21] Per la precisione non ci interessano gli IRS che trattano **solo** archivi non strutturati; non si esclude però che gli altri possano essere adibiti anche a trattare archivi non strutturati o poco strutturati: infatti in diversi contesti documentari può avere grande importanza l'indicizzazione full text di testi completi, ad esempio di articoli scientifici, testi letterari ecc.

numero di caratteri anche quando non contengono dati (e quindi si evita di definire campi troppo lunghi che determinerebbero un grande spreco di spazio su disco), mentre in un IRS i campi sono tipicamente a **lunghezza variabile**, e quindi occupano esattamente tanto spazio quanto è richiesto dai dati contenuti (più i delimitatori di campo e/o sottocampo). Comunque vi sono DBMS che possono trattare anche campi testuali a lunghezza variabile (in SQL corrispondono al tipo di dati *varchar*), per cui la distinzione tra campi a lunghezza fissa e variabile non si può considerare pienamente rappresentativa di quella tra IRS e RDBMS.

In un RDBMS ogni record (ossia ogni riga di una tabella) ha la stessa struttura, per cui questa struttura viene memorizzata nel database una sola volta; se i campi sono a lunghezza fissa, ogni record ha la stessa lunghezza, e ogni campo ha la stessa lunghezza in tutti i record, e questo consente di individuare immediatamente la posizione di inizio di qualsiasi campo in qualsiasi record. Ad esempio, se in una tabella ogni riga è lunga 470 caratteri, e il primo campo è lungo 85 caratteri, per sapere dove inizia il secondo campo del record 2317 è sufficiente effettuare il semplice calcolo [(470 * 2316) + 85], dal quali risulta che il campo desiderato inizia alla posizione 1088605 dei dati. Un calcolo di questo genere viene effettuato dal computer in un tempo trascurabile, per cui l'accesso è velocissimo, e questo risultato si ottiene con una struttura molto semplice (sono ovviamente più significativi i tempi necessari per leggere il record e trasferirlo in memoria): basta infatti che oltre ai dati venga registrata, in una posizione predefinita e quindi immediatamente accessibile, la struttura della tabella. In un IRS non si può usare lo stesso meccanismo, perché i campi sono a lunghezza variabile, per cui la posizione in cui inizia il secondo campo del record 2317 dipende da quanti dati sono contenuti nei record precedenti. Un metodo per trovare questa posizione sarebbe leggere tutto il database dall'inizio finché non si incontra il punto desiderato, ma è chiaro che con questa tecnica i tempi di accesso sarebbero enormi, e quindi il programma non avrebbe alcuna utilità pratica, se non con archivi molto piccoli. Gli IRS usano invece dei sistemi di indici nei quali - in generale - viene registrata la posizione di inizio di ciascun record e, per ciascun record, la posizione di inizio di ciascun campo. Sempre parlando in generale, almeno il primo livello dell'indice deve essere composto di elementi a lunghezza fissa, perché altrimenti si riproporrebbe il problema di come individuare rapidamente ciascun elemento. Se questi indici sono ben progettati, anche gli IRS permettono di accedere con estrema rapidità a qualsiasi record desiderato, e lo stesso naturalmente vale per gli RDBMS che gestiscono campi a lunghezza variabile.

Per fare un esempio, il ben noto IRS CDS-ISIS ha un indice con struttura a tre livelli. Il primo livello comprende elementi a lunghezza fissa che indicano la posizione di inizio di ciascun record. Questo livello è memorizzato in un file fisico separato rispetto a quello principale [22]. Il secondo livello è anch'esso a lunghezza fissa e contiene, tra l'altro, la posizione di inizio dei campi variabili e il numero di campi nel record. Il terzo elemento è l'elenco dei campi del record, ed è a lunghezza variabile perché la descrizione di ogni campo ha lunghezza fissa, ma varia il numero di campi (un campo che in quel record non contiene dati non si considera presente ma vuoto, come nei DBMS, ma del tutto inesistente). Risulta quindi chiaro che questa struttura non è orientata alla tabella, ossia ad un insieme di record, ma al singolo record, per cui in linea di principio ogni record potrebbe avere una struttura del tutto diversa da quella degli altri [23].

Per quanto riguarda le operazioni degli IRS, possiamo dire che non ci sono né il join né un linguaggio standardizzato (o quasi) come SQL. Il join non è evidentemente applicabile nel contesto degli IRS, mentre un linguaggio standard non è impossibile per principio, ma certamente la sua elaborazione viene resa difficoltosa dal fatto che mentre in tutti i DBMS si può fare riferimento alla tabella, e quindi a righe e colonne, negli IRS non c'è un tale elemento comune.

---

[22] Si tratta del file con estensione .xrf, mentre il file principale è quello con estensione .mst

[23] Gli utenti di CDS-ISIS potrebbero osservare che esiste però la FDT (Field Definition Table) che contiene la struttura dei record che vale per tutto il database; la FDT però serve essenzialmente per controllare l'immissione dei dati e come documentazione per l'utente ma non determina assolutamente cosa può essere scritto nel database: tramite l'Isis-Pascal o l'importazione da ISO 2709 si possono tranquillamente scrivere campi non previsti dalla FDT, che poi possono essere visualizzati con le varie tecniche standard di Isis (non possono essere modificati tramite le normali worksheet di inserimento dati); inoltre le modifiche alla FDT non cambiano minimamente il contenuto del database

Ovviamente gli IRS permettono di effettuare operazioni come creazione, modifica, cancellazione, visualizzazione, stampa e ordinamento dei dati, nonché altre che costituiscono di solito il loro punto di forza, come la ricerca e l'esportazione/importazione, delle quali parleremo più dettagliatamente in seguito.

Spesso gli IRS permettono di fare anche operazioni che ad un osservatore disattento potrebbero sembrare simili al join: infatti in molti prodotti di questo genere un campo può contenere un riferimento a dati di altri record dello stesso database o di altri, per cui questi dati vengono estratti quando si richiama il record di partenza per la visualizzazione od altre elaborazioni. Ancora più lontana dal join è la tecnica di copiare in un campo un dato preso da un altro archivio, cosa che permette di implementare, ad esempio, liste di autorità. Si può fare in modo che chi inserisce i dati non sia neppure consapevole dell'uso di due archivi distinti, ma che semplicemente scelga il dato che gli interessa tra quelli che gli vengono presentati, con un **apparenza**, quindi, simile a quella dei DBMS quando presentano i dati da scegliere per il join. Per rimarcare la differenza tra le due tecniche, basti pensare che nel join qualunque modifica nei dati della lista di autorità viene automaticamente riflessa nella tabella risultato del join perché si tratta in realtà dello stesso dato, mentre nella copiatura ciò non avviene, perché il dato di origine e quello copiato sono fisicamente distinti (anche se non è impossibile creare procedure che effettuano automaticamente l'aggiornamento del dato copiato). Si deve osservare, tuttavia, che alcune operazioni effettuabili tramite gli IRS sono effettivamente operazioni relazionali (nel capitolo sulle applicazioni alle biblioteche vedremo un esempio di prodotto cartesiano rappresentato con CDS-ISIS). Non per questo viene a cadere la distinzione tra RDBMS e IRS, perché questi ultimi si basano su strutture di dati che non corrispondono alle entità dell'algebra relazionale, e quindi le operazioni relazionali sono in certo qual modo un prodotto derivato: ad esempio il prodotto cartesiano può venire implementato da CDS-ISIS in fase di output dei dati, ma non può venire rappresentato nei dati stessi.

Quando poc'anzi si è parlato di indici, l'espressione si riferiva ad indici che rappresentano la struttura del record, cioè la posizione dei singoli campi che compongono il record. Gli IRS hanno però anche indici che vengono utilizzati per la ricerca, ossia per localizzare con la massima rapidità i record che corrispondono a una certa condizione di ricerca. La più diffusa tecnica per la gestione di questi indici è quella dell'**inverted file** (o lista invertita), utilizzata, tra l'altro, da CDS-ISIS. Un inverted file è una lista di chiavi di indice, cioè delle parole o stringhe o altri dati che sono resi accessibili tramite l'indice, ad ognuna delle quali è associato un elenco di puntatori al record, campo, eventuale ricorrenza del campo (se questo è ripetibile) e posizione in cui il termine si trova (ovviamente se il termine si trova in più record l'inverted file conterrà le informazioni utili per identificare ciascun record). Spesso l'inverted file contiene anche l'indicazione del numero di occorrenze del termine (dette **postings**). Vi sono numerose tecniche di rappresentazione di un inverted file tramite sistemi di codifica che hanno lo scopo di ridurre l'occupazione di spazio necessaria per immagazzinare l'inverted file.

3.1 **Applicazioni degli information retrieval**

Da quanto appena detto, si può facilmente immaginare che le applicazioni tipiche degli IRS non saranno certamente quelle in cui eccellono i RDBMS (gestione personale, contabilità, fatturazione ecc.), ma sono quelle che si possono far rientrare nell'ambito della **documentazione**. Infatti in questo ambito è per lo più necessario rendere ricercabili in modo sofisticato testi più o meno strutturati; a seconda del contesto, è necessario un grado più o meno elevato di strutturazione, così come è necessario sviluppare in modo più o meno sofisticato quanto riguarda l'immissione dei dati.

Le prime applicazioni che possono venire in mente sono quelle per la catalogazione di biblioteche, archivi o musei, ma gli IRS non si limitano a questo. Ad esempio possono essere utilizzati per sistemi di documentazione aziendale [24], in particolare per il fatto che, non essendo vincolati ad una struttura tabellare, si prestano alla gestione di documenti molto eterogenei. Disponendo di un IRS abbastanza potente, una azienda potrebbe creare un sistema informativo comprendente da una parte il catalogo della biblioteca e quello dell'archivio

[24] Con il termina *aziendale* non ci si riferisce solo ad aziende in senso stretto, ma più in generale ad organizzazioni più o meno grosse, che possono quindi essere pubbliche amministrazioni, associazioni ecc.

(storico e/o corrente), dall'altro documenti full-text come relazioni, rapporti tecnici, documentazione interna ecc. Ogni tipo di documento potrebbe venire archiviato con il tracciato record più appropriato, ma essere ricercabile con modalità comuni. Inoltre ai documenti full-text potrebbero venire aggiunti campi strutturati ad indicare, ad esempio, l'autore o il soggetto. Non solo, ma l'IRS potrebbe gestire documenti multimediali (immagini digitalizzate, grafici, video ecc.) e interagire con il RDBMS utilizzato per la parte gestionale (es. bilancio) per acquisirne dati online oppure in batch.

Talvolta si sceglie anche di utilizzare contemporaneamente un RDBMS e un IRS per trattare gli stessi dati: il primo per la produzione ed il controllo dei dati, il secondo per la ricerca e la consultazione. In questo caso, i dati vengono trasferiti ad intervalli regolari dal DBMS all'IRS. Questa soluzione consente di avere il meglio di entrambe le tecnologie a prezzo, evidentemente, di costi maggiori: bisogna infatti come minimo acquistare e gestire due software invece di uno, ma quasi sempre la soluzione prevede anche due macchine diverse ognuna con il suo sistema operativo (i due sistemi operativi possono essere uguali o diversi, a seconda dei particolari software che si usano). Tuttavia questa soluzione è sempre più diffusa, e viene usata ad esempio nell'indice SBN: infatti l'ambiente in cui vengono prodotti i dati è un mainframe IBM con sistema operativo MVS e il RDBMS DB2, mentre quello che viene utilizzato per la consultazione pubblica su Internet è una macchina Unix di fascia intermedia con l'IRS Basis. Anche in diversi poli SBN (come quello piemonte) vengono impiegati un RDBMS per il database di produzione e un IRS per l'OPAC Web. Spesso anche nelle biblioteche gli utenti non consultano più l'archivio di produzione, ma l'OPAC Web (che può essere locale o su Internet) che presenta maggiore facilità d'uso e spesso anche prestazioni più sofisticate.

4. **INTERROGAZIONE DI BASI DI DATI**

Si potrebbe dire che l'interrogazione delle basi di dati è un po' il punto saliente di tutta la faccenda. A che servirebbe, infatti, inserire dati con la massima cura e precisione se poi gli stessi dati non si potessero ritrovare, quando servono, con la massima efficacia, rapidità e secondo criteri diversi a seconda delle necessità ? Bisogna quindi dedicare particolare attenzione a questo argomento.

Quanto diremo si può riferire, in linea generale, sia agli IRS che agli RDBMS, mentre vi possono essere differenze significative tra un prodotto e l'altro. Certi tipi di ricerca sono però tipici piuttosto degli IRS che degli RDBMS, anche se non in modo assoluto, per cui questo sarà messo in evidenza nel corso della trattazione.

La condizione preliminare per poter effettuare ricerche che non siano solo sequenziali, cioè che non comportino la lettura di tutti i dati a partire dall'inizio finché non si trova quello cercato, è la presenza di **indici**. Come già osservato, la tecnica utilizzata, tra le diverse possibili, per la costruzione degli indici dipende dallo specifico prodotto che si considera. Qui ci limitiamo ad alcune osservazioni di carattere generale. Innanzitutto, è di notevole interesse pratico il fatto che in alcuni prodotti gli indici vengono aggiornati non appena si immettono dati, mentre in altri vengono aggiornati solo quando si dà un apposito comando, che di solito viene dato alla fine di una sessione di immissione dati. È chiaro che in quest'ultimo caso i nuovi dati immessi non saranno ricercabili se non dopo l'aggiornamento degli indici, e non immediatamente dopo l'immissione. Nella maggior parte dei programmi, inoltre, è possibile non solo aggiornare gli indici, ma anche rigenerarli completamente: di solito è consigliabile compiere questa operazione con una certa frequenza per migliorare le prestazioni e l'affidabilità, e ridurre l'occupazione di spazio su disco. Si deve infatti tener presente che non vi sono solo i dati ad occupare spazio disco, ma anche gli indici, le cui dimensioni sono spesso anche maggiori di quelle dei dati. Per la verità, questo era un problema più preoccupante qualche anno fa che non oggi, a meno che non si abbia a che fare con archivi estremamente grandi oppure con computer piuttosto vecchi, perché è difficile mettere in crisi la capienza degli hard disk odierni. Per fare un esempio concreto, un archivio di 100.000 record bibliografici gestito con Isis/Teca può arrivare ad occupare tra dati ed indici 200-300 Mb di spazio, quindi una piccola frazione dell'hard disk anche del più modesto PC oggi in vendita. Il problema

dell'occupazione di spazio è però più rilevante in archivi di dimensioni molto grandi, soprattutto se indicizzati full-text: in questi casi di può arrivare anche a decine di Gb di spazio disco.

Poiché la ricerca utilizza gli indici, è evidente che non si può ricercare quello che non è indicizzato, proprio come in un libro non si può cercare un argomento se c'è solo l'indice dei nomi, né cercare un nome se c'è solo l'indice degli argomenti. Tuttavia non possiamo qui spiegare le caratteristiche degli indici di specifici archivi, perché ci può essere la più grande varietà spiegazioni, oltre al fatto che in diversi casi gli indici sono personalizzabili, possiamo però illustrare le diverse tecniche di indicizzazione possibili.

In particolare, possiamo distinguere essenzialmente i seguenti tre casi:

- **l'indice corrisponde al contenuto di un campo tutto intero** (o eventualmente di un sottocampo, dove i sottocampi sono previsti, il che avviene normalmente negli IRS); per esempio, se viene indicizzato il campo titolo, le chiavi dell'indice (cioè i dati ricercabili) corrisponderanno ai titoli (di solito è possibile, introducendo opportuni caratteri di marcatura, escludere dalla chiave le parole non significative che si trovassero all'inizio del campo, ad esempio gli articoli); un indice di questo tipo consente ricerche per chiave intera, ossia - per riprendere l'esempio di poco fa - per titolo intero, e normalmente anche per parte iniziale della chiave, ma non per parole o per parti intermedie, per cui non è molto flessibile; è molto adatto quando si effettuano ricerche per identificatori univoci, come il codice fiscale, l'ISBN, il codice prodotto in un magazzino, il numero della fattura e simili
- **l'indice corrisponde alle singole parole di un campo** (o eventualmente a specifiche parole o parti identificate da specifici caratteri di marcatura inseriti insieme ai dati; questa variante è tipica soprattutto di alcuni IRS); questo tipo di indice occupa notevole spazio su disco ma è estremamente flessibile perché permette di cercare un testo conoscendone solo alcune, o anche solo una parola, per cui è utilissimo, ed anzi quasi indispensabile, per la ricerca su campi testuali, soprattutto se lunghi e di contenuto variabile; è utile ricordare che generalmente vengono esclude dall'indicizzazione alcune parole generiche, come gli articoli e le congiunzioni; le parole escluse dall'indicizzazione sono dette **stopword**, e l'elenco delle stopword è detto **stoplist**; la ricerca per parole è tipica degli IRS, ma in generale è possibile anche negli RDBMS, dove però può essere meno efficiente quanto a prestazioni
- **l'indice corrisponde ad una espressione formulata in un linguaggio messo a disposizione dal RDBMS o IRS**; nei primi due tipi di indicizzazione che abbiamo descritto, ogni chiave di indice corrispondeva letteralmente ad un dato presente nell'archivio; a volte però può essere necessario o utile ricercare i dati in una forma diversa da quella nella quale sono stati inseriti; ad esempio, consideriamo un database bibliografico che contiene i nomi degli autori: i lettori che interrogano il database potrebbero cercare questi nomi in diversi modi; potrebbero cercare solo il cognome, e allora è sufficiente una indicizzazione per parole, ma se volessero cercare il nome completo, come lo cercherebbero ? mettendo una virgola dopo il cognome, come nelle intestazioni per autore (Manzoni, Alessandro), oppure secondo l'uso comune senza la virgola (Manzoni Alessandro), o magari con il nome prima del cognome (Alessandro Manzoni) ? se vogliamo permettere ai lettori di ricercare in tutti e tre i modi, dobbiamo avere i mezzi, appunto apposite istruzioni, per creare, in base ai dati presenti nell'archivio, le tre diverse stringhe cui abbiamo fatto riferimento; l'indicizzazione su espressione è un mezzo potentissimo per rendere più flessibile la ricerca, e si può trovare tanto negli RDBMS quanto negli IRS (ad esempio è presente tanto in dBase III e successori quanto in CDS-ISIS); le potenzialità in questo campo dipendono più dagli specifici prodotti utilizzati che dalla loro tipologia

Vedendo la cosa con riguardo al contenuto del database, possiamo distinguere il caso in cui vengono indicizzati solo alcuni campi, e non tutti, oppure solo alcune parole dei campi e quello in cui viene indicizzato l'intero contenuto del database. Se il contenuto è testuale questo tipo di indicizzazione viene detta **full-text**. Si deve però ancora distinguere l'indicizzazione full-text su testi controllati, ossia redatti da specialisti secondo norme standardizzate, come il testo di una notizia bibliografica (descrizione ISBD, intestazione RICA ecc.), e i testi liberi. Anche in campo documentario, può avvenire che vi siano database che contengono non solo le registrazioni bibliografiche, ma anche testi completi di documenti come articoli o altri. Un importante caso di indicizzazione full-text di testi liberi è quello dei motori di ricerca di Internet, che indicizzano l'intero contenuto

dei siti Web. L'indicizzazione full-text di testi liberi rende necessari alcuni accorgimenti in ricerca, dei quali parleremo più oltre.

Una volta che abbiamo a disposizione tutti gli indici che ci servono, o almeno tutti quelli che il nostro programma ci permette di creare, possiamo finalmente effettuare la ricerca. Per valutare se la ricerca ha avuto successo, si tiene conto principalmente di due parametri:

- il **richiamo** è il rapporto tra i record ritrovati e quelli pertinenti all'ambito della ricerca presenti nel database; per record pertinenti non si intendono quelli che corrispondono all'espressione di ricerca, che vengono sempre ritrovati tutti a meno che non vi siano errori nel programma, nei dati o negli indici, ma quelli che sul piano semantico rientrano nello scopo della ricerca; per esempio, chi cercasse le opere di Manzoni con una delle stringhe ricordate poc'anzi non troverebbe i record in cui compare la forma "Manzoni A.", che pure corrispondono anch'essi ad opere di Manzoni
- la **rilevanza** è il rapporto tra i record effettivamente attinenti all'oggetto della ricerca e il totale di quelli ritrovati; infatti alcune formulazioni di ricerca possono portare al ritrovamento di record che in realtà non erano desiderati: per esempio, chi cercasse le opere di Alessandro Manzoni utilizzando il solo cognome troverebbe anche quelle dello scrittore Carlo Manzoni[25], che presumibilmente non erano tra quelle desiderate; la percentuale di record non pertinenti rispetto a tutti quelli ritrovati è detta **rumore**

In linea di principio, nulla vieta di ottenere nel contempo il 100 % di richiamo e il 100 % di rilevanza, cosa che equivale a trovare tutti e solo i record pertinenti alla ricerca effettuata. Questo, anzi, succede spesso con le ricerche per chiave univoca: se si vuole sapere a chi appartiene un certo codice fiscale, bisogna semplicemente fare una ricerca per codice fiscale e si troveranno tutte e sole le persone cui quel codice appartiene. Anche con ricerche molto semplici, soprattutto - ma non solo - su piccoli database, si può ottenere un tale risultato. Le cose tendono a cambiare con ricerche sofisticate, come quelle che si hanno nel campo delle biblioteche e della documentazione (ad esempio: cercare tutti gli articoli che trattano di DSP prodotti da ditte europee e adatti all'impiego negli analizzatori di Fourier). Qui di fatto avviene che se si vuole il massimo di richiamo bisogna quasi sempre rinunciare alla rilevanza, ossia rassegnarsi a ritrovare un certo numero di record non pertinenti. Viceversa, se si vuole eliminare del tutto il rumore bisogna di solito rassegnarsi ad un peggioramento del richiamo, ossia a non ritrovare alcuni record che sarebbero pertinenti. Questo avviene perché per aumentare il richiamo bisogna rendere meno specifica la ricerca, ma questo può facilmente aumentare il rumore, mentre per aumentare la rilevanza bisogna rendere più specifica la ricerca, cosa che però può escludere alcuni record che invece sarebbero pertinenti alla ricerca stessa. Quando si deve scegliere tra ottimizzare il richiamo e ottimizzare la rilevanza, di solito di sceglie il primo quando si cerca su piccoli database o quando la ricerca è comunque tale da non ritrovare un eccessivo numero di record, mentre si sceglie il secondo nel caso opposto. Ad esempio, può accadere che una ricerca su un certo database, restituisca 100.000 record; a questo punto, è difficile che ci sia qualcuno disposto ad esaminare tutti i 100.000 record, quand'anche fossero tutti rigorosamente pertinenti allo scopo della ricerca, per cui normalmente si preferirà specificare la ricerca introducendo ulteriori parametri per selezionare, tra questo insieme, i record maggiormente interessanti.

Ma quali sono le cause che peggiorano il richiamo e aumentano il rumore (ossia diminuiscono la rilevanza) ? Il peggioramento del richiamo in genere è causato da:

- ricerche formulate in modo troppo specifico
- ricerche formulate in modo errato sul piano semantico, ossia tali da identificare un insieme di record diverso da quello desiderato, oppure con condizioni incoerenti (si pensi a una ricerca dei libri pubblicati nella DDR **dopo** il 1990, quando la DDR non esisteva più)
- ricerche formulate in modo improprio sul piano sintattico (vi possono essere ricerche formalmente corrette ma tali da non poter mai restituire alcun record proprio in virtù della loro stessa formulazione)
- errori di inserimento dei dati nel database; può trattarsi di banali errori di battitura, che sono particolarmente nefasti perché, essendo imprevedibili, possono rendere del tutto introvabili alcuni record

[25] Carlo Manzoni è uno scrittore umoristico, creatore del personaggio del signor Veneranda

(se Manzoni è stato scritto in qualche record in modo errato, chi può prevedere tutti i possibili errori per verificare la presenza di quel record ?), oppure ad esempio - per rimanere nell'ambito bibliografico - di errori di soggettazione o di classificazione

- database non ben progettato, o comunque non progettato per il tipo di ricerca che si sta eseguendo; ad esempio, la maggior parte dei database bibliografici non si prestano alla ricerca per generi letterari, o per temi e personaggi delle opere letterarie, quindi sarebbe piuttosto ingenuo chi, volendo trovare tutti i romanzi e racconti di vampiri, cercasse la parola "vampiro"

Le cause che determinano il peggioramento del rumore sono in certo qual modo simili:

- ricerche formulate in modo troppo generico
- ricerche formulate in modo errato sul piano semantico, ossia tali da identificare un insieme di record diverso da quello desiderato, oppure con condizioni tali che nella loro combinazione individuano un insieme di record troppo ampio
- errori di inserimento dei dati nel database; può trattarsi ancora una volta di errori di battitura: infatti il risultato dell'errore può casualmente rientrare nell'ambito di una ricerca che in realtà non c'entra niente (si pensi, ad esempio, ad autori che si distinguano solo per la presenza di lettere doppie nel cognome); analogo effetto possono avere non solo gli errori di soggettazione o di classificazione, ma anche una politica di soggettazione e classificazione troppo generica (per esempio, se nel catalogo gli indici Dewey vengono inseriti solo fino alla terza cifra, sarà impossibile fare ricerche veramente specifiche sulla classificazione)
- database non ben progettato, ed in particolare con un tracciato record troppo sommario che inserisca nello stesso campo dati eterogenei (si pensi a un database bibliografico che preveda un unico campo per tutta la descrizione ISBD: questo non permette di fare ricerche solo su una parte della descrizione, per cui il rumore è quasi inevitabile, essendoci spesso parole che compaiono in parti diverse della descrizione con ruoli differenti)

I risultati della ricerca si possono migliorare di molto utilizzando strategie di ricerca alternative, in modo che se una non dà buon esito si può tentare con un altra per verificare se il risultato è lo stesso oppure è migliore: ad esempio, se una ricerca sul titolo completo non restituisce niente, si può tentare una ricerca sul titolo troncato oppure sulle parole del titolo (v. seguito per l'illustrazione di questi tipi di ricerca). Molto importante è comunque conoscere le caratteristiche dell'archivio su cui si opera: ad esempio, se l'archivio contiene pochi dati semplici e molto standardizzati di solito è difficile avere molto rumore, per cui è meglio concentrarsi sull'aumento del richiamo. Il caso opposto è invece quello della indicizzazione full-text su testi liberi, che può facilmente condurre a una enorme quantità di rumore, come avviene quasi sempre quando si cerca con una unica parola molto comune. Bisogna quindi fare in modo da rendere più specifica la ricerca: tuttavia, poiché si tratta di testi liberi, il loro linguaggio non è standardizzato, per cui può succedere che improvvisamente si passi dal trovare troppo al non trovare niente del tutto. Non c'è una soluzione semplice a questo problema: spesso si devono fare diversi tentativi combinando formulazioni atte a restringere la ricerca con altre che permettano di tener conto dalla variabilità della terminologia dei testi indicizzati.

Un altro aspetto importante della ricerca sono le **prestazioni**: infatti la ricerca non dovrebbe solo restituire un risultato esatto e rispondente al contenuto del database, ma anche restituirlo in breve tempo. Le prestazioni in ricerca dei vari programmi dipendono comunque dalla progettazione degli stessi, per cui l'utente in genere non ci può fare molto. Può tuttavia avvenire che alcuni tipi di ricerca siano sensibilmente più lenti di altri, per cui se si conosce questa caratteristica del programma che si sta usando si può cercare di effettuare prima le ricerche che offrono migliori prestazioni, e solo dopo quelle che offrono prestazioni inferiori. Si tenga presente che facendo ricerche su di un singolo computer o su una piccola rete difficilmente si incontreranno problemi di prestazioni (a meno che non si stia utilizzando hardware inadeguato), mentre questi sorgono più facilmente quando vi sono molti utenti che contemporaneamente interrogano il database, specialmente se questo è molto grande (in grossi sistemi si possono avere anche centinaia o migliaia di interrogazioni contemporanee).

4.1 Query booleane

Le query booleane si chiamano così perché suddividono l'insieme dei record presenti nel database in due sottoinsiemi: quello dei record che soddisfano alle condizioni impostate e quelli che non vi soddisfano[26]. In altri termini, per ogni record vi sono due possibilità: o soddisfa la condizione di ricerca o non la soddisfa, e il software di interrogazione provvede a individuare qual è il caso per ogni record.

Vediamo ora quali sono esattamente le tecniche utilizzate per la ricerca. Il linguaggio di ricerca in realtà varia da un software all'altro (con la parziale eccezione di SQL che, sia pure con diverse varianti, è uno standard, ma solo per i RDBMS), per cui non è possibile spiegare qui il linguaggio di ricerca di tutti i programmi disponibili. Tuttavia i vari linguaggi fanno in gran parte le stesse cose, anche se le rappresentano in modo diverso, per cui ad esempio l'operatore *and* (v. seguito) può essere rappresentato in un caso dalla parola AND, in un altro dal segno *, e in un altro ancora dal segno &[27]. Quello che illustreremo ora sono appunto i diversi comandi ed operatori che si possono normalmente utilizzare, esposti però in forma generale e senza riferimento alla loro rappresentazione in specifici linguaggi di ricerca. Quando non viene specificato altrimenti, si intende che questi comandi possono trovarsi sia negli RDBMS che negli IRS. Si deve però tener presente che in alcuni programmi potrebbero non essere supportati tutti i tipi di ricerca qui elencati (per saperlo con certezza si deve fare riferimento alla documentazione del programma).

Le istruzioni di ricerca possono quindi riassumersi nel modo che segue:

- ricerca per **singolo dato**, dove il dato può essere una parola, un numero, una data ecc.; rientra in questo caso anche la ricerca di una stringa, composta di più parole, da ritrovare letteralmente (in alcuni programmi si deve usare, in questo caso, un particolare carattere - di solito le virgolette - per delimitare la stringa); è evidentemente il tipo di ricerca più semplice perché non richiede l'immissione di alcunché se non il solo dato che si vuole cercare, ma spesso non è sufficiente, per cui bisogna impiegare tecniche più sofisticare
- **troncamento a destra** o **ricerca per radice**; **troncamento a sinistra** o **ricerca per desinenza**; si applicano ai testi, e permette di ricercare tutte le stringhe che cominciano nel modo indicato, come ad esempio: biblioteca, biblioteche, bibliotecari, bibliotecarie ecc. (troncamento a destra); si può avere anche il troncamento a sinistra, che cerca tutte le parole che finiscono con la stringa data, e anche la ricerca di una sottostringa all'interno di una parola (che ad esempio è possibile in SQL); il troncamento a destra permette di cercare in una volta sola tutte le forme di un termine (singolare, plurale, maschile, femminile), nonché più termini semanticamente affini che hanno la stessa radice, ma può provocare molto rumore perché spesso ritrova anche termini di cui non era tenuto conto; alcuni programmi, per diminuire il rumore, impongono un numero minimo di caratteri prima del troncamento; il rischio di rumore è evidentemente molto più accentuato nel troncamento a destra e nella ricerca di sottostringhe all'interno delle parole, per cui questi tipi di ricerca sono di uso più raro
- **operatori di confronto**; sono operatori come *maggiore di*, *minore di*, *compreso tra*, *diverso da* che si applicano soprattutto ai numeri e alle date; alcuni programmi consentono di applicarli anche alle stringhe: in questo caso una espressione come *maggiore di xyz* significa *ciò che segue xyz nell'ordine alfabetico*; spesso questi operatori sono utilizzati in associazione con altri per specificare condizioni aggiuntive, come quando si cerca un titolo e in più si impone che la data di pubblicazione sia successiva o precedente ad un certo anno
- **identificatore di campo**; permette di limitare la ricerca a uno o più campi specifici; in alcuni linguaggi di ricerca, ad esempio SQL, è impossibile fare una ricerca senza specificare anche il campo, in altri, come quello di CDS-ISIS, la specificazione è facoltativa; l'uso dell'identificatore di campo è molto utile per limitare il rumore, perché consente di neutralizzare gli effetti di dati materialmente identici ma che in campi

[26] Questo a prima vista potrebbe non risultare così ovvio, perché quello che viene presentato all'utente che effettua la ricerca è solo l'insieme dei record che soddisfano le condizioni di ricerca, mentre quello dei record che non vi soddisfano non viene mostrato.

[27] Avviene anche, soprattutto in applicazioni orientate al pubblico e non agli addetti, che il vero linguaggio di ricerca venga "nascosto" dietro espressioni più informali e vicine al linguaggio comune; ad esempio, può esserci un menu con l'opzione *Cerca tutte le parole* che, come si vedrà nel seguito, corrisponde ad una ricerca in and.

diversi hanno diversa rilevanza per la ricerca (ad esempio, quando si cerca un nome di città come luogo di pubblicazione è irrilevante lo stesso nome che figura nel titolo o nel soggetto, o addirittura come cognome dell'autore); esso però richiede una buona conoscenza della struttura del database, altrimenti si possono avere risultati completamente falsati, ad esempio cercando un dato in un campo in cui non potrebbe mai trovarsi

- **ricerca per somiglianza**; si applica ai testi e consiste nel fatto che alcuni programmi permettono di cercare i tutti i termini che "somigliano" a quello dato, dove la somiglianza può essere intesa secondo vari criteri; per esempio si possono cercare in una sola volta le forme maschili, femminili, singolari e plurali, oppure forme grammaticalmente varianti, come le voci di un verbo o un termine declinato (si noti che rispetto al troncamento questo metodo, se ben implementato, può avere il vantaggio di escludere termini parzialmente uguali a quello dato ma in realtà del tutto diversi); è anche possibile cercare le parole di suono simile a quella data; questo tipo di ricerca evidentemente tende ad aumentare il numero dei record ritrovati, per cui se non usata in modo accorto può produrre rumore; è utile per ricerche sofisticate in ambito documentario
- i tipi di ricerca fin qui illustrati si applicano a ricerche costituite da **un solo termine** (dove per termine si intende anche una stringa presa come una unica condizione di ricerca); se però si vogliono impiegare **più termini** bisogna indicare al programma in che modo deve combinarli; le regole secondo cui i termini vanno combinati sono dette **operatori**, mentre i termini da combinare sono detti **operandi**; vediamo quali sono gli operatori che si possono utilizzare
  - i più noti e diffusi sono gli **operatori booleani**, così chiamati dal nome del logico e matematico George Boole, che nel secolo scorso per primo elaborò un calcolo logico formalizzato in modo analogo ai calcoli aritmetici, e che sono i seguenti:
    - **AND**; ritrova i record nei quali sono presenti **tutti** gli operandi; serve per restringere l'ambito della ricerca; infatti una ricerca in and, rispetto alla ricerca con uno solo degli operandi, può al massimo trovare lo stesso numero di record (se gli operandi sono presenti sempre e solo tutti insieme), ma non un numero maggiore; esagerare con l'and può portare a non trovare più nulla, perché si sono messi in and termini che in realtà non si trovano mai insieme
    - **OR**; ritrova i record nei quali è presente **almeno uno degli operandi**; serve per allargare l'ambito della ricerca; infatti una ricerca in or, rispetto alla ricerca con uno solo degli operandi, può al minimo trovare lo stesso numero di record (se gli operandi sono presenti sempre e solo tutti insieme), ma non un numero minore; su grossi database un uso sconsiderato dell'or può aumentare oltre misura il rumore; tuttavia questo operatore può essere utile, ad esempio, per consentire la ricerca di sinonimi o varianti di un termine, o anche per effettuare in una sola volta ricerche diverse delle quali si vogliono vedere i risultati tutti insieme (ad esempio, se si vogliono vedere insieme tutte le edizioni di opere di Verga e tutte le edizioni di opere di Fogazzaro)
    - **NOT**; nega il suo operando, ossia trova tutti i record nei quali l'operando non è presente; vi sono tuttavia due varianti del not; esso può essere utilizzato come operatore unario, ossia applicato ad un solo operando, e quindi trova semplicemente tutti i record nei quali non c'è l'operando, oppure può essere utilizzato tra due operandi, per cercare i record nei quali è presente il primo ma non il secondo (inteso in questo modo, il not è in realtà un *and not*); a seconda del programma che si usa, può essere ammesso un senso o l'altro del not, o anche entrambi; il not come operatore unario ha una utilità piuttosto limitata (che senso avrebbe cercare tutti i libri **non** scritti da un certo autore ?), soprattutto quando si nega un valore tra un limitato numero di valori possibili; ad esempio, se sappiamo che un campo può contenere solo la sigla di una provincia ligure, e quindi un valore scelto tra quattro possibili, è più rapido e comodo cercare *not GE* piuttosto che *IM or SP or SV*; il not inteso come *and not* ha l'effetto di restringere la ricerca, ed è utile quando un termine può essere presente in contesti eterogenei, ma associato a termini diversi a seconda del contesto: con il not si può limitare la ricerca al contesto che effettivamente interessa; le ricerche con il not tuttavia richiedono cautela perché, con dati non facilmente prevedibili a priori, possono facilmente restringere la ricerca in modo eccessivo, e quindi peggiorare il richiamo
    - **XOR** (eXclusive OR); ritrova i record in cui è presente almeno uno degli operandi, ma non più di uno; in realtà è un operatore derivato, perché si può definire in questo modo: *a XOR b = (a NOT b)*

OR (b NOT a); viene usato raramente, e molti programmi non lo supportano, anche perché lo si può facilmente sostituire, se necessario, usando l'or e il not nel modo indicato

- c'è poi una famiglia di operatori che si possono considerare come **estensioni dell'and**, perché impongono ulteriori restrizioni, oltre a quelle dell'and, che devono essere tutte soddisfatte; si trovano principalmente negli IRS, perché servono essenzialmente per la ricerca su testi, soprattutto se si tratta di testi liberi e/o con indicizzazione full-text; questi operatori sono necessari perché a volte il semplice and non è sufficientemente specifico: infatti due o più termini possono trovarsi tutti presenti in un record, e anche in un campo, ma senza alcun legame tra loro, per cui il record che viene ritrovato non ha in realtà alcun rapporto con l'oggetto della ricerca; gli operatori di cui parliamo vengono chiamati **operatori di prossimità** (o anche **di adiacenza**) perché impongono che i termini cercati non devono solo essere tutti presenti, ma devono anche essere vicini tra di loro; si tratta di una famiglia di operatori, perché la prossimità viene intesa in modi diversi: ad esempio, l'operatore può imporre che i termini siano immediatamente adiacenti, oppure che siano separati da non più di *n* parole, oppure da esattamente *n* parole; inoltre si può anche imporre che i termini siano presenti in un determinato ordine; i migliori programmi prevedono tutte queste varianti degli operatori di prossimità, e permettono anche di scegliere liberamente il numero di parole ammesso o imposto come distanza tra i termini di ricerca, in modo che l'utente possa personalizzare al massimo grado la ricerca per meglio adattarla alle sue esigenze ed alle caratteristiche dei dati in cui sta cercando; in questa categoria di operatori si possono anche far rientrare quelli che impongono che i termini si trovino tutti in un determinato campo o (nei programmi che prevedono i campi ripetibili) in una stessa ricorrenza del campo
- un'altra possibilità particolarmente sofistica è quella delle **sottoquery**, tipiche soprattutto degli RDBMS ed in particolare del SQL: al posto di un termine, si può utilizzare una intera espressione del linguaggio di ricerca (che è appunto la sottoquery), per cui nella ricerca viene considerato l'insieme dei record restituiti da questa espressione; ad esempio, se si vogliono cercare i residenti a Genova che hanno una età maggiore di tutti i residenti a Savona, in SQL si può usare la seguente espressione di ricerca (la sottoquery corrisponde alla parte tra parentesi): *select * from genova where età > all (select età from savona);* [28] (naturalmente *genova* e *savona* devono essere due tabelle del database, e *età* un attributo (e quindi una colonna) di queste tabelle
- citiamo infine le **parentesi**, che, in presenza di più operatori in una stessa istruzione di ricerca determinano l'ordine di esecuzione degli operatori; ad esempio *(a AND b) OR c* trova i record in cui sono presenti contemporaneamente *a* e *b*, e tutti quelli in cui è presente *c*, mentre *a AND (b OR c)* trova i record in cui è presente *a* insieme a *b* oppure *a* insieme a *c*; i migliori programmi permettono di usare diversi livelli di parentesi (ossia parentesi all'interno di altre parentesi); naturalmente i programmi hanno un ordine di precedenza degli operatori da utilizzare se sono dati più operatori in assenza di parentesi

Concludiamo osservando che i programmi consentono di combinare liberamente diversi operatori per cui, ad esempio, uno degli operandi di un *and* può essere un termine troncato, oppure in una stessa espressione di ricerca possono esserci contemporaneamente operatori booleani ed operatori di prossimità.

4.2 Ranked query

Il termine inglese **ranked query** è di difficile traduzione, per cui spesso si preferisce lasciarlo nella forma originale, e così verrà fatto anche in questo documento. Comunque l'inglese *rank* significa rango, grado, posizione in una scala di valutazione, disposizione ordinata. Questo suggerisce già l'idea che sta alla base delle ranked query: non determinare, per ogni record, se soddisfa le condizioni di ricerca o no, ma determinare la sua rilevanza rispetto alla ricerca impostata dall'utente, rilevanza che può essere più o meno grande, per cui tutti i record (a parte, ovviamente, quelli per i quali non viene riconosciuta alcuna rilevanza) vengono collocati appunto secondo una scala di rilevanza. Si tratta di un tipo di ricerca che si applica soprattutto agli IRS.

[28] Di fatto, alcuni software, soprattutto se non molto recenti, non permetterebbero di usare una parola con l'accento, in questo caso *età*, come nome di attributo

La ricerca non viene impostata combinando i termini con degli operatori, ma dando una lista di termini ed espressioni atte a descrivere l'argomento che interessa. I record restituiti possono anche non contenere tutti i termini contenuti nella query, appunto perché potrebbero essere di una qualche rilevanza pur in questa condizione.

A questo punto sorge il problema di determinare una metrica della rilevanza, cioè un metodo per misurare la rilevanza di un record rispetto a una certa query. A questo scopo sono state elaborate diverse tecniche, tra cui ha una notevole importanza la **regola del coseno** (*cosine rule*). I termini del documento e quelli della query vengono consideranti come vettori in uno spazio n-dimensionale (il numero delle dimensioni è uguale a quello dei termini), e la rilevanza del documento rispetto alla query viene stimata in base al coseno dell'angolo tra i due vettori: il coseno è 1 quando i vettori coincidono, e poi può assumere valori via via più bassi a misura che aumenta l'angolo, e quindi da differenza tra la query e i documenti.

Le ranked query sono molto usate nei motori di ricerca sul Web, dove spesso questo tipo di query viene proposto di default, anche se questi sistemi - in nome della semplicità d'uso - tendono a nascondere all'utente la loro infrastruttura tecnica, per cui a volte non si capisce bene cosa in realtà facciano. Le ricerche sul Web sono interrogazioni full-text su enormi database di documenti completi senza alcuna standardizzazione di linguaggio, per cui spesso le ricerche booleane producono troppo rumore oppure, se si tenta di restringerle per evitare il rumore, rischiano di produrre risultati eccessivamente ristretti o nulli. Le ranked query invece tentano di presentare una serie di documenti ordinati per rilevanza, il che tra l'altro permette di presentare prima quelli giudicati più rilevanti, in modo da aumentare la probabilità che l'utente ritrovi le informazioni di suo interesse già solo scorrendo la prima parte della lista dei documenti ritrovati. In ogni caso, come ben sa chiunque li abbia provati, i software dei motori di ricerca sono tutt'altro che infallibili e non sempre i loro giudizi di rilevanza sono quelli che ci si aspetterebbe, anche se in genere sono abbastanza soddisfacenti; spesso non centrano molto l'obiettivo di evitare il rumore, ma riescono a separare con discreta efficacia i documenti più pertinenti da quelli meno, per cui non c'è nessuna assicurazione che il documento classificato nella posizione 1 sia davvero più pertinente di quello alla posizione 3, ma è molto probabile che sia più pertinente di quello alla posizione 200. Per sfruttare bene le caratteristiche della ranked query è opportuno introdurre molti termini, in modo che il software abbia una base di confronto più ampia: non bisogna infatti trattare la ranked query come una query booleana, in cui un eccessivo numero di termini può condurre a non ritrovare niente se la query è in and o a ritrovare troppo se la query è in or. Una buona strategia può anche essere quella di fare sia una ranked query che una query booleana e poi confrontare i risultati. Notiamo infine che in genere i motori di ricerca applicano le loro tecniche di valutazione della rilevanza anche ai risultati delle query booleane, in modo da mostrarli appunto in ordine di rilevanza, cosa che può risultare di notevole utilità.

5. COMPONENTI SOFTWARE

Un qualunque RDBMS o IRS consiste di diversi componenti che svolgono funzioni distinte. Per la precisione, se le funzioni sono comunque per loro natura distinte, i componenti possono essere a loro volta programmi ben distinti che vengono eseguiti separatamente (anche se tutti insieme), oppure può trattarsi di un unico programma che svolge tutte le funzioni, cosa che avviene di solito nei software più semplici. L'utilizzatore di questi programmi non vede però direttamente tutti i componenti, ma solo alcuni di essi, appunto quelli che sono destinati all'interazione con l'utente, e questa visione parziale è fonte di numerose incomprensioni. per cui è necessario correggerla.

Possiamo dire che i componenti, intesi nel senso illustrato prima, rientrano in tre categorie:

- motore relazionale o motore di information retrieval
- front-end (interfaccia utente)
- strumenti di sviluppo

Gli strumenti di sviluppo hanno un ruolo particolare, per cui ad essi sarà dedicato un capitolo a parte, mentre qui ci occuperemo delle altre due categorie:

- il **motore relazionale** (per gli RDBMS) o il **motore di information retrieval** (per gli IRS) è quel componente che si occupa della vera e propria gestione della base dati (per cui si usa anche l'espressione **motore di database** in un senso che comprende sia RDBMS che IRS), ossia di tutte le operazioni che coinvolgono la base dati vera e propria: definizione e modifica archivi, creazione di indici, aggiunta, modifica e cancellazione di dati, gestione delle transazioni, ricerca; come facilmente si comprende, si tratta del componente più importante perché, quali che siano gli altri componenti, essi devono necessariamente ricorrere al motore di database se vogliono fare qualche operazioni sulla base dati; eventuali limiti o difetti di un motore di database non possono quindi essere superati da altri componenti; è anche il componente più difficile da progettare per cui, come si vedrà meglio nella sezione dedicata agli strumenti di sviluppo, ci sono molto meno motori di database che non applicazioni di database; per fare alcuni esempi tra i software per biblioteca, Teca, Edan e Biblio utilizzano tutti il motore IRS di CDS-ISIS, Erasmo utilizza, a scelta, i motori relazionali di Access o di SQL Server (prodotti Microsoft che sono impiegati in un gran numero di applicazioni di ogni genere), Sebina Produx utilizza il linguaggio Progress, pure esso impiegato in numerose applicazioni, Aleph utilizza un altro dei più noti e diffusi motori relazionali, quello di Oracle; molto diffuso anche il motore di information retrieval della Dataware Technologies, impiegato nel CD-ROM della BNI e in diversi altri CD bibliografici
- ma come si fa a comunicare al motore di database quali operazioni si desidera effettuare, ossia dirgli ora di cancellare un dato, ora di aggiungerne un altro, ora di avviare la tale ricerca ? bisogna che ci siano degli elementi adatti ad accettare e ad eseguire i comandi dell'utente; in particolare, devono essere possibili almeno le seguenti operazioni:
  - **operazioni sulle tabelle**: inserimento, modifica e cancellazione di dati direttamente nelle tabelle; questo gruppo di funzioni consente, come si vede, di creare nuovi database e modificarne la struttura, nonché di inserire dati direttamente nelle tabelle, cosa che di solito viene fatta solo a scopo di test, oppure in database molto semplici; espresse in termini di tabelle, queste operazioni si applicano evidentemente solo agli RDBMS, ma se invece che a tabelle in senso stretto si pensa in generale alla struttura di un archivio, è chiaro che le stesse cose si possono applicare anche agli IRS; alcuni IRS, come CDS-ISIS, non consentono però di inserire dati direttamente nell'archivio, senza utilizzare i componenti di cui parleremo nel prossimo punto
  - **amministrazione del database**: si tratta di diversi tipi di operazioni ad esempio creazione ed eliminazione di tabelle o modifica della loro struttura, creazione, modifica ed eliminazione di indici, gestione della sicurezza (abilitazione e disabilitazione di utenti ecc.), salvataggio dei dati, installazione di nuove versioni del software; l'addetto a questi compiti è detto DBA, sigla di Data Base Administrator; nei software per uso personale, queste operazioni sono di solito molto intuitive e alla portata di tutti, mentre nei prodotti professionali possono richiedere un grado anche molto elevato di competenza tecnica; nei grossi sistemi (ad esempio in polo SBN) gli utenti non possono svolgere alcuna di queste operazioni, che sono rigorosamente riservate a specialisti dotati di una approfondita conoscenza del programma utilizzato
  - **inserimento dati controllato ed aiutato**: spesso inserire dati direttamente nell'archivio è complicato, soprattutto negli RDBMS in cui i dati sono suddivisi in molte tabelle; inoltre è molto grande il pericolo di introdurre dati errati, che possono poi essere difficili da individuare e correggere; per evitare questi inconvenienti bisogna utilizzare dei "moduli" per l'inserimento dei dati (che, a seconda del programma che si usa, vengono chiamati maschere, worksheet o form, che è forse il termine più diffuso, soprattutto nei programmi per Windows), i quali, analogamente a dei moduli cartacei, presentano vari campi da compilare; ovviamente i dati inseriti vanno a finire in campi del database sottostante, ma vengono presentati non secondo la struttura di questo, ma nel modo più intuitivo e semplice per chi deve inserirli; ad esempio, nell'indirizzario che abbiamo descritto nel capitolo sui RDBMS, la form di inserimento dati non presenterebbe le varia tabelle distinte, ma presenterebbe i dati della persona, ed una lista di città (presa, ovviamente dalla corrispondente tabelle) dalla quale scegliere la voce appropriata; dopo la scelta poi il programma si occuperebbe di aggiornare la terza tabella, quella di collegamento, che resterebbe sempre invisibile all'utente

- **ricerca** (o **query**): più o meno in tutti i programmi è possibile inserire direttamente istruzioni di ricerca in SQL o in altri linguaggi di ricerca, ma poiché questo risulta difficile per molti utenti, vi sono delle interfacce di ricerca facilitate, che permettono di comporre una ricerca in modo intuitivo senza conoscere lo specifico linguaggio di interrogazione; in molti programmi le stesse tecniche vengono usate anche per determinare l'ordinamento dei dati; si noti che non tutti i programmi mettono a disposizione una ricerca facilitata: alcuni, come CDS-ISIS, permettono di usare solo il linguaggio nativo di interrogazione, mentre la ricerca facilitata deve essere realizzata separatamente con gli strumenti di sviluppo; si noti comunque che a volte le interfacce facilitate non permettono di sfruttare tutte le possibilità del linguaggio di ricerca; può anche succedere che per creare ricerche molto complesse e sofisticate sia più semplice e chiaro utilizzare direttamente il linguaggio di ricerca che non l'interfaccia facilitata; negli RDBMS spesso è possibile salvare una formulazione di ricerca (in SQL o eventualmente in un altro linguaggio) che può poi venire utilizzata esattamente come una tabella; tali query possono essere ricerche in senso stretto, con le quali si selezione l'insieme dei record che hanno soddisfano a certe condizioni. oppure possono anche essere delle union o dei join
- **presentazione e stampa**: di solito presentare i dati esattamente così come sono memorizzati nell'archivio, ad esempio in forma di tabella se si tratta di un RDBMS, non viene considerato sufficiente per tutte le esigente, perché spesso si desidera una presentazione più piacevole e facilmente leggibile, ed inoltre non sempre uguale, ma adattabile a diverse circostanze; di conseguenza, tutti i programmi consentono di personalizzare in modo più o meno sofisticato la presentazione dei dati; per quanto riguarda la presentazione a video, alcuni software utilizzano delle form analoghe a quelle che si usano per l'inserimento dei dati, mentre altri richiedono di costruire un formato con un qualche tipo di linguaggio di formattazione; per quanto riguarda invece la stampa, alcuni software utilizzano lo stesso linguaggio di formattazione che si usa per la presentazione dei dati a video, altri prevedono strumenti specifici per la creazione dei **report**, che sono appunto dei formati di stampa; questi strumenti consentono di scegliere i dati da stampare e il loro aspetto (ad esempio corsivo o neretto), e di solito anche di inserire nella stampa titoli e scritte varie, nonché particolari informazioni come il numero di pagina, il totale dei record stampati, la data e l'ora, il nome del database ecc.; per quanto riguarda l'ordinamento in stampa, alcuni programmi prevedono che venga specificato direttamente nel report, mentre in altri si specifica tramite una query a cui il report fa riferimento

Nei programmi per Windows, come Access, Paradox e Visual dBase, è in genere presente la cosiddetta **interfaccia visuale**, che permette di progettare form e report in modo intuitivo, ponendo su di una superfice presentata a video vari **oggetti**, che possono essere campi contenenti i dati (esistenti o da inserire), bottoni che attivano certe azioni (ad esempio il passaggio ad un'altra form), immagini, scritte ecc. Questi oggetti in genere vengono scelti da un insieme predefinito e messo a disposizione dal particolare software che si sta usando e posizionati tramite il mouse sulla superfice di lavoro. Attraverso appositi menu è possibile definire le **proprietà** dei vari oggetti: ad esempio, se l'oggetto è un campo per presentare o immettere dati, si potrà indicare a quale colonna di quale tabella fa riferimento, se è possibile immettere dati o solo visualizzarli, con quale font e di quale gandezza devono apparire i dati ecc. L'interfaccia visuale si rivela nella maggior parte dei casi molto vantaggiosa, perché permette anche ad utenti non professionisti di ottenere ottimi risultati (e permette anche ai professionisti di risparmiare tempo). Tuttavia sarebbe illusorio pensare di poter risolvere tutti i problemi con l'interfaccia visuale, senza conoscere la struttura del database e le possibilità di personalizzazione e configurazione degli oggetti, spesso molto sofisticate. Non di rado, inoltre, non è sufficiente conoscere la struttura del particolare database che si sta usando, ma è anche necessario avere almeno una certa conoscenza delle possibilità e delle caratteristiche del motore di database, sia esso un information retrieval o un RDBMS, poiché altrimenti si rischia di non poter sfruttare che in minima parte quelle caratteristiche[29].

[29] Ad esempio, un tipico errore di coloro che utilizzano gli RDBMS senza ben sapere che siano è quello di inserire tutti i dati in una unica tabella, invece di suddividerli in diverse tabelle e poi creare i join opportuni; naturalmente in questo modo si perdono quasi tutti i vantaggi degli RDBMS senza per questo acquisire i vantaggi degli IRS

In conclusione, quindi, possiamo riassumere quello che abbiamo detto finora osservando che nell'insieme di componenti con cui si ha a che fare quando si usa un RDBMS o un IRS si distinguono tre livelli:

- il motore di database, che gestisce i dati, e che comprende lo storage manager e il query processor
- l'archivio, ossia di dati con la loro struttura
- il front-end, ossia l'insieme di tutti quei componenti che permettono all'utente di interagire con il motore di database al fine di gestire l'archivio

Con uno stesso motore di database si possono creare e gestire molti archivi diversi. A seconda di ciò che mette a disposizione il programma, si possono avere anche front end più o meno diversi. In alcuni casi si possono solo avere molte form o report diversi, in altri si possono avere interfacce utente che nascondono completamente il motore di database sottostante, rendendo visibile solo quanto serve per l'uso di quel particolare archivio. Questo argomento risulterà più chiaro ancora dopo la lettura del capitolo sugli strumenti di sviluppo.

Chi non tiene presente questa distinzione non può avere una idea chiara del funzionamento dei programmi perché non può identificare chiaramente il ruolo dei diversi componenti cui si trova di volta in volta di fronte.

6. SICUREZZA DELLE BASI DI DATI

L'argomento della sicurezza viene spesso trascurato, ma è essenziale per l'uso professionale delle basi di dati. Si tratta infatti di ottenere i seguenti risultati:

- **regolamentare l'accesso ai dati**, impedendolo a chi non ne ha diritto, e mettendo, se necessario, dei limiti a chi ne ha diritto (per esempio, qualcuno potrebbe leggere una parte dei dati, ma non un'altra; qualcun altro potrebbe essere autorizzato a leggere i dati ma non a modificarli), ossia assegnando ad ogni utente un insieme di autorizzazioni, dette anche privilegi
- **assicurare l'autenticazione**, ossia da una parte identificare con certezza chi sta utilizzando la base dati, dall'altra rendere certo l'utente su quale applicazione sta effettivamente utilizzando; spesso non si pensa a questo secondo aspetto, che pure è del tutto sensato; si pensi ad esempio a qualcuno che sta lavorando in collegamento ad un polo SBN: come può costui essere certo di essere veramente collegato al polo, e non invece ad un server pirata che imita in tutto l'aspetto dell'applicativo SBN, ma che in realtà sia del tutto diverso, ed installato per qualche scopo occulto, operazione improbabile ma tecnicamente possibile ? per avere questa certezza è necessario che vi siano delle procedure di autenticazione, ossia tali da distinguere un applicativo "vero" da uno pirata che si spaccia per il primo; come si può facilmente intuirem l'autenticazione è un argomento collegato con il controllo degli accessi, ma che non identifica del tutto con questo; un altro aspetto dell'autenticazione è il rendere possibile la **non ricusazione**, ossia impedire a chi ha avuto parte in una certa operazione di negare di averlo fatto; si tratta, ad esempio, di impedire che chi ha immesso dei dati (magari errati) possa negare di averli immessi, oppure che chi ha ricevuto un certo messaggio o una certa richiesta possa negare la ricezione; la non ricusazione ha notevole importanza in campo commerciale, quando si tratta di ordini o pagamenti, ma può avere rilievo anche nel campo bibliotecario, in particolare nell'ambito dei servizi di prestito e riproduzione, nonché nei rapporti coi fornitori, ed eventualmente anche nell'ambito catalografico, ad esempio se si volesse avere la possibilità di identificare quali catalogatori fanno continuamente errori senza che costoro possano negare di esserne proprio loro i responsabili
- **impedire danneggiamenti e dispersioni dei dati** dovuti a guasti, errori ed incidenti

Il sistema più noto per il controllo degli accessi e per l'autenticazione è quello della **password**, ossia una parola chiave che deve essere nota solo a un certo utente, per cui chi la immette viene identificato appunto come quel certo utente, dotato di determinate autorizzazioni. La password deve quindi essere conservata gelosamente, affinché estranei non possano venirne a conoscenza. Inoltre è necessario che chi usa una certa password sappia esattamente quali sono le sue autorizzazioni, soprattutto se queste includono operazioni potenzialmente pericolose. Putroppo queste elementari precauzioni non vengono sempre rispettate: in certi casi, la password viene conservata senza alcuna cura della segretezza, e addirittura scritta sul monitor o su qualche bigliettino attaccato al computer, in modo da non doverla imparare a memoria. Le conseguenze di

questi comportamenti possono essere molto gravi: ad esempio, se la password permette di accedere alla base dati della biblioteca per la catalogazione a cade in mano ad un estraneo, costui potrebbe aggiungere, modificare e cancellare dati. Se poi la password desse anche i privilegi di data base administrator, un estraneo potrebbe addirittura cancellare l'intero catalogo ! Se invece la password desse accesso ai dati dei lettori e dei prestiti, oltre ai pericoli per l'integrtà dei dati, si potrebbe andare incontro a problemi legali: **infatti la legge 675/1996 sulla tutela dei dati personali impone a chiunque detenga archivi contenenti dati personali di adottare tutte le precauzioni necessarie perché questi non vadano in mano a persone non autorizzate**, e non si può certo dire che abbia adempiuto a questo obbligo chi scrive la password sul monitor del computer o utilizza altri sistemi di sicurezza di analoga efficacia.

Non manca poi chi non ha chiara cognizione delle autorizzazioni cui una certa password dà diritto, per cui non sa di poter anche compiere operazioni pericolose (ivi compresa, se la password è quella dell'amministratore del sistema, la cancellazione non solo del catalogo della biblioteca, ma dell'intero contenuto dei dischi del computer) e quindi rischia di non agire con la dovuta cautela.

Comunque non è difficile rimediare agli inconvenienti che abbiamo descritto. Per il secondo, è sufficiente documentarsi sulle caratteristiche del programma ed eventualmente del sistema operativo che si sta usando, in modo da comprendere come vengono gestite le autorizzazioni agli utenti. Per la gestione delle password è sufficiente (a meno che non si operi in contesti ad alto rischio) evitare di scrivere le password ovunque, e anzi scriverle su carta una sola volta e per l'uso quotidiano impararle a memoria (la cosa migliore sarebbe anzi non scriverle affatto, ma se le password sono molte e assegnate a diversi utenti questa soluzione non è praticabile). Le password scritte andranno poi conservate in modo da non essere facilmente accessibili, ad esempio dissimulandole in mezzo a documenti di altro genere. Se poi la biblioteca dispone di una cassaforte, è senz'altro consigliabile conservarle lì. Inoltre, se il software utilizzato lo consente, è bene cambiare frequentemente le password, cosa che è comunque necessaria se si ha anche solo il sospetto che qualcuno ne sia venuto a conoscenza.

Ma le password non sono il solo sistema di sicurezza. Esse infatti in certi contesti non sono applicabili, ad esempio quando si tratta di garantire la riservatezza di dati privati che però viaggiano su una rete pubblica, come Internet, o anche privata. Bisogna quindi fare in modo che certi dati non siano utilizzabili da parte di chi non ne ha diritto, anche se costui ne viene materialmente in possesso. A questo scopo si utilizza la **crittografia**, ossia i dati vengono trasformati secondo un determinato criterio, per cui non sono decifrabili se non da chi applichi all'inverso lo stesso criterio, ossia possegga la **chiave** per la decrittazione. Ad esempio, se un testo venisse crittografato sostituendo ad ogni lettera quella che viene sette posizioni dopo nell'ordine alfabetico (si noti però che i sistemi di crittografia reali prevedono elaborazioni enormemente più complesse), potrebbe venire ricostruito solo da chi sa che deve sostituire ogni lettera con quella che viene sette posizioni prima nell'ordine alfabetico. Questo esempio ci dà modo di distinguere più precisamente **l'algoritmo di crittografazione** dalla **chiave**. Infatti in questo caso l'algoritmo è la regola *sostituisci ogni lettera con quella che viene n posizioni dopo nell'ordine alfabetico*, mentre la chiave è il valore di *n*, in questo caso sette. **La sicurezza crittografica si basa sulla segretezza della chiave, non su quella dell'algoritmo**, che invece è pubblico. A prima vista, questo potrebbe sembrare strano, ma non lo è: basti pensare al fatto che un algoritmo segreto potrebbe sempre essere scoperto da qualcuno in modo autonomo, e più ancora al fatto che - in presenza di un algoritmo segreto - l'utente di un sistema di crittografia non potrebbe mai essere sicuro delle sue effettive prestazioni, né che chi detiene l'algoritmo non si sia provvisto anche dei mezzi per decrittare i dati altrui, per segreti che questi possano essere per tutti gli altri.

Ci sono due tipi di crittografia. Quello più tradizionale è detto **a chiave singola**, ed è quello descritto poco fa. La stessa chiave viene usata sia per la codifica che per la decodifica, per cui deve essere conosciuta sia da chi codifica i dati sia da chi li decodifica. Questo è il punto debole del sistema: infatti la chiave deve evidentemente essere tenuta segreta, ma almeno una volta essa deve essere comunicata dal mittente al destinatario dei dati. In questa operazione, la chiave deve essere trasmessa in chiaro, cioè non crittografata, perché altrimenti ci sarebbe bisogno di un'altra chiave per crittografarla, la quale a sua volta dovrebbe essere

trasmessa in chiaro a meno di non avere una terza chiave e così via. Risulta evidente quindi che almeno una chiave deve essere trasmessa in chiaro, indipendentemente dal fatto che venga trasmessa con mezzi informatici, a voce, per posta o coi segnali di fumo (sistema, quest'ultimo, scarsamente adatto ad assicurare una qualsiasi segretezza dei dati), e quindi priva di protezione. Naturalmente qualsiasi dato crittografato diventa completamente accessibile a chi conosce la chiave, per cui l'obiettivo è di rendere sicura quest'ultima, e proprio questo punto di vista il sistema a chiave singola presenta una certa debolezza (per contro, se la chiave è progettata bene, esso può essere assolutamente sicuro finché la chiave rimane segreta).

Per rimediare ai difetti del sistema a chiave singola è stata ideata la crittografia **a doppia chiave**. In questo sistema ogni utente dispone di una **chiave privata**, nota a lui solo, e di una **chiave pubblica**, che invece può venire resa nota a tutti. Chi deve inviare dati a qualcuno li codifica con la chiave pubblica **del destinatario**, il quale poi li decodifica con la sua chiave privata. In questo modo la chiave privata non deve mai passare di mano. Le due chiavi sono progettate in modo che sia impossibile, o quasi, ricavare la chiave privata da quella pubblica. Il sistema si può anche utilizzare in modo inverso a quello appena descritto: **il mittente** codifica i dati con la sua chiave privata, e poi il ricevente li decodifica con la chiave pubblica del mittente. Questo evidentemente non garantisce la segretezza dei dati, ma ne garantisce la provenienza: infatti con la chiave pubblica di qualcuno non è possibile decrittare dati che non siano stati crittografati con la sua chiave privata [30]. I dati crittografati in questo modo costituiscono una **firma digitale**. Tra i sistemi di crittografia basati sulla doppia chiave, oltre celebre PGP (Pretty Good Privacy), scaricabile gratuitamente da Internet (http://www.pgp.com), che però dalla versione 7 non è più interamente open source, segnaliamo l'open source GNU Privacy Guard, scsricabile da http://www.gnupg.org/.

Ci sono anche sistemi di crittografia **non reversibili**, ossia tali che i dati originali non possono più essere ricostruiti da quelli crittografati. Questi sistemi non servono a scambiare dati codificati, perché evidentemente chi li riceve non può risalire a quelli originali, ma servono in particolare per codificare le password. Infatti con le password è necessario che il sistema che esegue l'autenticazione possa confrontare la password immessa da chi tenta di accedere con quella registrata in forma crittografata, mentre non si desidera affatto che chi dovesse venire in possesso della password crittografata possa risalire a quella originale.

Che cosa vuol dire che è quasi impossibile ricavare la chiave privata da quella pubblica? Vuol dire che non è impossibile in linea di principio, ma che comporta un carico computazionale talmente elevato da essere impossibile in pratica: se ricavare la chiave pubblica fosse possibile solo a condizione di far lavorare per mille anni il compuer più potene del mondo, è evidente che la cosa non costituirebbe nessun effettivo pericolo. La potenza di calcolo dei computer però aumenta a ritmo elevatissimo, e quindi attacchi che prima erano inconcepibili possono non esserlo più. Da ciò che abbiamo detto, si comprende facilmente che nella maggior parte dei casi, gli attacchi ai sistemi di crittografia non avvengono tentando di impadronirsi delle chiavi presso chi le possiede oppure, nel caso dei sistemi a chiave singola, nel momento in cui queste vengono scambiate, ma avvengono provando, tramite appositi software, un grandissimo numero di chiavi sperando di trovare quella giusta (si tenga presente che l'algoritmo di crittografia è noto). Di conseguenza i progettisti dei sistemi di sicurezza tentano di trovare algoritmi così complessi e chiavi così lunghe da rendere necessario un tempo di calcolo enorme per l'individuazione della chiave (a questo punto si comprende perché l'algoritmo di crittografia citato nell'esempio di prima non offra la minima sicurezza: infatti il numero di possibili chiavi è uguale a quello delle lettere dell'alfabeto, e i tentativi si potrebbero fare addirittura a mano, e a maggior ragione quindi con un computer). D'altra parte i progettisti non possono neppure aumentare oltre misura la complicatezza dell'algoritmo e la lunghezza delle chiavi, perché altrimenti anche la codifica e la decodifica diventerebbero estremamente lente. D'altra parte però la maggiore capacità di elaborazione permette anche di diminuire i tempi di codifica e decodifica, rendendo proponibile l'uso di chiavi sempre più lunghe, per cui la situazione non è mai in equilibrio. Fino a poco tempo fa, le chiavi di di 40 bit erano considerate scarsamente sicure, quelle di 128 bit abbastanza accettabili almeno per ambienti non molto a rischio, mentre chiavi di oltre

[30] È appena il caso di notare che questo non vale se la chiave privata viene scoperta, perché allora chi la conosce può codificare i dati proprio come farebbe il vero proprietario della chiave

1000 bit erano considerate assolutamente sicure (per un paragone, si pensi che per la chiave nell'esempio delle lettere dell'alfabeto basterebbero soli 5 bit[31]). Secondo notizie apparse recentemente su Internet (cfr. il sito http://cryptome.com) la NSA (National Security Agency, ente governativo statunitente preposto al controspionaggio e alla sicurezza interna, considerato quello dotato delle più avanzate risorse in fatto di intelligence e crittografia, più ancora di CIA e FBI), utilizzando al massimo le sue risorse di calcolo, sarebbe in grado di violare chiavi di 1024 bit e forse anche più lunghe, mentre quelle di 4096 bit dovrebbero essere ancora inattacabili. Comunque l'uso di chiavi di 4096 bit con un PC Pentium II a 350 Mhz, e quindi abbastanza aggiornato ma non certo all'avanguardia, non comporta particolari problemi di prestazioni[32].

Per quanto riguarda le password, è stato notato in studi approfonditi che esse possono venire indovinate con una certa frequenza perché spesso sono scelte male. Ciò accade quando si scelgono password prevedibili, soprattutto da chi abbia qualche informazione sugli utenti che le hanno scelte. Esempi di password prevedibili, e quindi assolutamente da evitare sono: il nome o il cognome dell'utente, oppure della fidanzata, della moglie, dei figli e simili, la città di residenza o quella di lavoro, la data di nascita, il nome del calciatore o della squadra di calcio preferita. Sono anche da evitare tutte le password troppo brevi: ad esempio, le possibili password di tre lettere scelte da un alfabeto di 24 lettere sono 24^3, cioè 13.824. L'insieme di tutte queste password può essere facilmente generato da un apposito programma, che poi può provarle una ad una fino a trovare quella giusta[33]. Le password dovrebbero invece essere di almeno cinque o sei caratteri (le possibili password di cinque carattere utilizzando un alfabeto di 24 lettere più le 10 cifre dallo 0 al 9, per un totale di 34 simboli, sono 34^5, cioè 45.435.424), alternando maiuscole, minuscole e numeri in modo da evitare di formare parole dotate di significato (quindi vanno bene password come *Ky667u* oppure *rAT2gh7*[34]).

I sistemi di crittografia hanno anche un significato politico, perché permettono ai cittadini di comunicare liberamente senza pericolo che estranei, ivi compresa la polizia e le autorità, possano intercettare il contenuto della comunicazione. Gli USA non consentono l'esportazione di sistemi di crittografia con chiave più lunga di 40 bit, limite giudicato, come detto poc'anzi, scarsamente sicuro, perché questi sistemi sono considerati alla stregua di armi dal momento che potrebbero permettere a terroristi od altri criminali di comunicare senza che la magistratura e la polizia abbiano la possibilità di scoprirli. A ciò si obietta che la crittografia permette anche ai cittadini di difendere la loro privacy contro intrusioni arbitrarie. Per motivi simili, alcuni diffidano dei sistemi di crittografia ideati ed approvati da organismi pubblici (PGP, ad esempio, è di iniziativa totalmente privata), perché temono che essi nascondano dei punti di accesso atti a permettere alla polizia di decifrare facilmente i dati. In Italia attualmente non ci sono limitazioni all'uso dei sistemi di crittografia, per cui tutti possono usare quello che a loro pare più sicuro. Tuttavia i sistemi con chiave di lunghezza superiore ai 40 bit non possono essere scaricati da Internet né acquisiti in altri modi qualora provengano dagli Stati Uniti, appunto perché la legge USA ne vieta l'esportazione[35].

---

[31] È utile ricordare che il numero di combinazioni di n elementi ognuno dei quali può assumere t valori diversi è pari a t^n . Poiché un bit può assumere, per definizione, sono due valori, con 5 bit il numero di chiavi possibili è 32, con 40 bit è già 1.099.511.627.776, con 128 bit è 3,402823669209e+38 e con 1000 bit è un numero quasi inimmaginabile, cioè 1,071508607186e+301

[32] In questa sede si è trattato l'argomento dal punto di vista puramente tecnico. In pratica, può essere utile valutare quanto si è veramente a rischio di intercettazione, tenuto conto che la NSA certamente non utilizza giorni di tempo di calcolo dei suoi supercomputer per decrittare i dati di chiunque

[33] Per evitare questo tipo di attacco, in genere i sistemi scollegano l'utente dopo un certo numero di errati inserimenti della password, di solito tre. A volte poi il sistema, se il numero di tentativi errati è particolarmente elevato, disabilita del tutto l'utente, che quindi poi non può fare il login neppure con la password giusta. Questa tecnica è anche usata dal Bancomat per il caso di errato inserimento del codice segreto

[34] Queste password sono state formate a caso a puro titolo di esempio e, per quanto consta all'autore, non sono utilizzare su alcun sistema reale. **In ogni caso si consiglia di non usarle** perché il solo fatto che siano citate in questo documento potrebbe suggerire a qualcuno che sta tentando di forzare una password di provare proprio loro.

[35] PGP, realizzato negli USA, sarebbe appunto uno dei sistemi di cui è proibita l'esportazione; per aggirare il divieto, fu fatta uscire dagli USA una stampa del codice sorgente, che non rientra nel divieto, codice che poi in Europa qualcuno ha pazientemente digitato a computer e poi ricompilato

Osserviamo, in conclusione, che nel valutare i problemi di sicurezza in un particolare contesto occorre tenere presente non solo quali gli attacchi tecnicamente possibili, ma soprattutto quelli più probabili in quel particolare ambiente. Nella maggior parte dei casi, le biblioteche non dovrebbero essere un ambiente ad alto rischio sotto il profilo della sicurezza: infatti l'accesso non autorizzato alle basi dati delle biblioteche non riveste un particolare interesse economico, per cui gli attacchi potrebbero nella maggior parte dei casi essere effettuati a scopi dimostrativi o di puro e semplice teppismo. Non si può escludere comunque la possibilità di tentativi di violazioni della privacy dei lettori orientati a conoscere le letture di specifiche persone. Che le biblioteche non siano in generale un ambiente ad alto rischio non significa però che i problemi di sicurezza si debbano ignorare, ma solo che bisogna operare con la normale diligenza e mettere in atto le tecniche più comuni, purché efficaci, senza che sia necessario effettuare grossi investimenti per sistemi di sicurezza adatti invece ad ambienti ad alto rischio.

6.1 **Metodologie ed hardware di backup**

Gli attacchi derivanti da accessi illeciti non sono l'unica causa di perdite o alterazioni dei dati. Anzi, nella maggior parte dei casi, e anche nelle biblioteche questi inconvenienti sono dovuti a malfunzionamenti hardware e software (ad esempio un guasto irreparabile dell'hard disk su cui è memorizzato un database) oppure ad errori degli operatori (cancellazione involontaria di record o anche di interi file). L'unico modo per rimediare a queste perdite è disporre di una copia dei dati da usare per sostituire quelli andati persi o danneggiati; questa copia è detta copia di **backup** (o di salvataggio). Il ripristino dei dati fatto a partire dal backup viene detto **restore** o ripristino. **Alcuni trascurano del tutto di effettuare regolarmente i salvataggi: si tratta di un errore gravissimo, perché può comportare la perdita, senza alcuna possibilità di recupero, del risultato di anni o mesi di lavoro.**

I dischi, dischetti, nastri o altri mezzi su cui vengono immagazzinati i dati vengono detti **supporti**. Dei supporti di backup di parlerà di seguito.

Vediamo invece quale metodologia bisogna seguire per effettuare il backup, metodologia che si può riassumere in alcune regole:

- **il backup deve essere fatto tutti i giorni** (tranne, si intende, i giorni in cui non si fa alcuna modifica ai dati), e non una volta ogni tanto, in modo minimizzare così le perdite di dati (se si fa il ripristino da un backup del giorno prima si perdono solo i dati inseriti nella giornata corrente); è anche possibile, soprattutto sui server, automatizzare il backup, cioè avviarlo ogni giorno automaticamente ad un'ora determinata senza l'intervento da parte dell'operatore; in questi casi, tuttavia, è necessario verificare sempre o almeno periodicamente che il backup sia andato a buon fine, a scanso di brutte sorprese al momento del restore
- **oggetto del backup sono i dati**, e non altri elementi del database, come indici, query, form ecc.; alcuni programmi però non permettono di separare i dati dagli altri elementi, per cui in questi casi dovrà essere fatto il backup di tutto l'archivio; se invece è possibile salvare solo i dati, un backup di tutto l'archivio si può comunque fare a scadenze più lunghe oppure quando si modificano elementi diversi dai dati, come appounto forms o query; a scadenze ancora più lunghe si può fare il backup del programma completo; nel seguito, se non diversamente specificato, ci si riferisce sempre al backup dei dati
- **è meglio utilizzare le procedure di backup interne all'applicazione**, se queste esistono, e se non si rivelano difettose o inefficienti; se si fa il backup dall'esterno dell'applicazione, è consigliabile utilizzare un compressore come Pkzip o simili prodotti, perché diminuiscono di gran lunga la necessità di spazio sui supporti di backup; anche quando si fa il backup dall'interno dell'applicazione spesso è vantaggioso compattare poi i file di backup con uno di questi programmi; quando si fa il backup dall'esterno dell'applicazione, bisogna essere sicuri di aver individuato esattamente tutti i files necessario per fare poi con successo il restore; in caso di dubbio, è sempre meglio includere qualche file in più piuttosto che qualcuno in meno
- **bisogna sempre avere almeno due copie di ogni backup**, per il caso che una si riveli inutilizzabile, il che può avvenire soprattutto per danneggiamento del supporto di backup

- **è necessario avere sempre disponibili diversi backup effettuati in giorni diversi** (si intende che **ciascuno** di questi backup deve essere in almeno due copie); il motivo è che quando viene fatto il backup potrebbe già esserci qualche errore nei dati non ancora rilevato, per cui al momento del restore vengono ripristinati dati corrotti, senza alcuna possibilità di recuperare quelli sani; ci sono diversi criteri per effettuare questi backup scaglionati; un metodo è di utilizzare supporti diversi a cicli, ad esempio, settimanali, per cui ci sarà il backup del lunedì, quello del martedì ecc.; un altro metodo è quello di fare i backup giornalieri e anche un backup settimanale, ossia in pratica conservare per una settimana uno dei backup giornalieri; in aggiunta ad entrambi i metodi, potrebbe essere fatto anche un backup mensile su un supporto non riscrivibile che garantisce la migliore sicurezza contro le alterazioni dei dati
- **bisogna sostituire periodicamente i supporti** anche quando apparentemente non presentano alcun danno, poiché i difetti spesso vengono alla luce improvvisamente, magari proprio quando si deve fare un restore; evidentemente, se i supporti non presentano inconvenienti, possono essere tranquillamente utilizzati per scopi meno critici
- **i supporti vanno conservati in modo che non subiscano danni**, e quindi tenuti lontano da fonti di calore, luce solare diretta, umidità, sporcizia, campi magnetici[36]; inoltre vanno manipolati in modo che non subiscano urti, e non devono essere conservati insieme a materiale diverso, affinché non vengano continuamente mossi e spostati senza necessità; se il backup contiene dati riservati, e in particolare dati personali soggetti alla disciplina della L. 675/1996, bisogna anche prendere le precauzioni necessarie perché non cada in mano ad estranei

Una volta convinti dell'utilità del backup e compreso il metodo da seguire, bisogna scegliere il supporto adatto. Scegliere un supporto implica anche scegliere un particolare hardware.

Dal novero dei supporti utilizzabili, eliminiamo subito i **floppy disk**, che sono innanzitutto inaffidabili, avendo la tendenza a danneggiarsi, anche senza causa apparente, con estrema facilità. Questo basterebbe già, ma in aggiunta i floppy sono anche molto lenti e di scarsa capacità, per cui quando si ha una quantità considerevole di dati un backup rischia di richiedere un tempo intollerabile.

Rimangono quindi i seguenti tipi di supporto:

- **nastri**; ne esistono di molti tipi, e di capacità variabile tra qualche centinaio di Mb e parecchi Gb; ciascun tipo di nastro richiede il drive appropriato, non essendoci un formato universale, ma i nastri utilizzati per applicazioni professionali sono normalmente i DAT[37], disponibili in diverse capacità sempre dell'ordine di parecchi Gb; i nastri hanno in generale il più basso costo in rapporto alla capacità di memorizzazione; bisogna tener presente che essi sono supporti sequenziali, che cioè devono venire letti di seguito dall'inizio per potersi posizionare su di un determinato punto, a differenza dei dischi, nei quali ci si può posizionare direttamente sul punto desiderato; per quanto motivo i nastri sono in pratica indicati solo per il backup, e non si possono utilizzare per eseguire programmi o tenervi sopra altri dati
- **dischi magnetici e ottico-magnetici**; anche di questi ne esistono molti tipi, di varia capacità e costo; possono essere usati (naturalmente impiegando dischi diversi per i vari impieghi) non solo per il backup, ma anche per conservarvi programmi e dati meno utilizzati, anche se con una certa perdita di prestazioni rispetto agli hard disk; la capacità varia da 100 Mb a circa 2 Gb, con molti valori intermedi, comunque si può dire che le due categorie principali siano quelli di capacità dell'ordine dei 100 Mb e quelli con capacità dell'ordine di 1-2 Gb; pur non esistendo uno standard, e richiedendo quindi ogni tipo di disco l'apposito lettore, si può dire che gli Iomega e i Syquest sono i più diffusi; recentemente hanno avuto successo gli Imation da 120 Mb, anche perché il drive è compatibile con i floppy per cui, se viene montato all'origine, si risparmia il drive dei floppy
- **dischi ottici riscrivibili**; questa categoria è rappresentata essenzialmente dai CD-RW (CD riscrivibili), da 650 Mb di capacità, perché sono standardizzati e quindi non si è legati a un particolare modello di drive;

[36] Ovviamente i campi magnetici sono pericolosi solo per i supporti magnetici, come i nastri, e non per i supporti ottici

[37] DAT è la sigla di Digital Audio Tape (natro audio digitale), perché i DAT - apparsi verso la metà degli anni '80 - furono in origine concepiti per le registrazioni musicali digitali

molto promettenti i DVD-RAM, con capacità da 4,7 a oltre 18 Gb, anche i drive per la scrittura di DVD sono ancora piuttosto costosi; i DVD sono standard e quindi non vincolano ad un particolare produttore di drive

- **dischi ottici non riscrivibili**; sono i ben noti CD-ROM (650 Mb di capacità) e i recenti DVD-ROM (capacità da 4.7 a oltre 18 Gb), per i quali gli apparecchi per la scrittura sono già sul mercato, ma sono ancora alquanto costosi; hanno l'enorme vantaggio di essere standard, ma il fatto di essere scrivibili una volta sola non li rende adatti ai backup giornalieri; possono invece essere interessanti per i backup mensili e anche per i backup storici di cui diremo tra poco, appunto perché questi supporti non sono cancellabili neanche accidentalmente; ovviamente richiedono l'apposito drive per la scrittura dei CD, che dovrebbe aggiungersi a quello utilizzato per i backup giornalieri, aumentando quindi il costo dell'insieme; lo scrittore di CD è per utilizzabile anche per molti scopi diversi dal backup

Il mercato dei supporti per backup è in continua evoluzione, per cui prima di un eventuale acquisto è bene verificare la situazione in quel preciso momento, e anche valutare eventuali offerte speciali proposte dai fornitori.

6.2 **Conservazione dei documenti digitali**

Il discorso sui backup ci porta spontaneamente a pensare a quei backup che potremmo definire storici, ossia quelli di archivi ormai chiusi, che non vengono più aggiornati, e che ci si propone di conservare a tempo indefinito appunto come memoria storica o per precauzione nel caso possano risultare utili in futuro. Ma è opportuno generalizzare la trattazione a un punto di vista che oggi sta acquistando sempre maggiore importanza, e cioè quello della conservazione dei documenti digitali (tra i quali naturalmente i database hanno particolare importanza, il che giustifica l'inserimento dell'argomento in questa trattazione). Si intende che si parla di **conservazione a lungo termine**, perché per la conservazione a breve termine sono sufficienti i consigli dati nel paragrafo precedente.

Un aspetto da mettere in evidenza è anche il dibattito sempre crescente sul problema, se i documenti digitali diano sufficienti garanzie di durata e usufruibilità a lungo termine, o se per queste esigenze siano superiori i tradizionali documenti analogici. Un contesto in cui spesso si manifesta questo dubbio è quello della riproduzione di libri o manoscritti rari e di pregio: ci si chiede infatti se sia saggio affidarsi alla sola digitalizzazione, o se sia meglio affiancare alla digitalizzazione - idonea per assicurare la fruizione e diffusione della riproduzione nell'immediato - la microfilmatura per garantire una più sicura conservazione a lungo e lunghissimo termine.

Il problema è lungi dall'essere risolto, ma qui possiamo tentare di chiarire maggiormemente il quadro concettuale in cui ci si muove ed evidenziare alcuni presupposti che molti assumono senza esplicitarli.

Bisogna innanzitutto chiarire opportunamente l'oggetto del problema: conservazione, ma di che cosa esattamente? Possiamo affermare che vi siano almeno tre aspetti di cui tener conto, e cioè

- conservazione dei supporti
- conservazione dei formati
- conservazione degli ambienti.

Conservazione dei supporti. Spesso è l'unico aspetto di cui si tiene conto, il che è eccessivo, ma si tratta certamente di un aspetto fondamentale. Se infatti i supporti digitali, come floppy disk, CD-ROM, DVD, nastri e così via si deteriorano al punto da determinare la distruzione dei dati in essi contenuti, è evidente che tutta l'informazione va persa definitivamente. Come ovvio, non esistendo CD-ROM o nastri DAT di cento anni fa, non esistono esperienze sulla conservazione a lungo termine dei supporti, ma è evidente una caratteristica intrinseca dei supporti digitali, molto favorevole da questo punto di vista: essi possono essere duplicati indefinitamente senza alcuna perdita di qualità e, di solito, su supporti che diventano via via più capaci, con grandi vantaggi per la facilità di immagazzinamento. Per esempio, chi ha una raccolta di 100.000

floppy disk da 1,4 Mb già da anni ha la possibilità di trasferirla senza troppe difficoltà su circa 200 CD -ROM da 700 Mb, e ora può anche passarla su una ventina di DVD, agendo in tempi tali da evitare ogni rischio di degrado dei supporti.

Un altro aspetto della conservazione dei supporti è la possibile obsolescenza della loro tecnologia, che finisce per determinare la sparizione dal mercato dei prodotti necessari, per cui può accedere che se si guasta il drive adatto per leggere un certo tipo di disco rimuovibile non sia possibile ripararlo o acquistarne un altro, anche se il disco in sé fosse in perfetto stato. A questo problema però si può spesso rimediare scegliendo, dove possibile, supporti standard e non proprietari, cioè non legati a dispositivi di uno specifico produttore, che potrebbe anche sparire completamente dal mercato. Questo ci permette di introdurre un concetto che avremo modo di ritrovare in seguito: bisogna puntare sulle soluzioni pubbliche, documentate, standardizzate e non su quelle chiuse e proprietarie. Per fare un esempio, osserviamo cosa accade per i CD-ROM, che furono introdotti nel 1983 e oggi, a diciotto anni di distanza sono disponibili in ogni angolo e possono essere letti e prodotti con apparecchi di centinaia di marche diverse che certamente saranno disponibili ancora per anni. Per di più i lettori di DVD sono compatibili con i CD-ROM, e se - come probabile - avranno lo stesso successo possiamo prevedere almeno una quarantina d'anni di utilizzabilità dei CD, tempo nel quale c'è ogni possibilità di predisporre il trasferimento dei dati sui supporti che saranno in auge tra quindici o vent'anni. Si noti anche che dall'introduzione dei CD (che data dal 1982 per i CD audio, che sono supporti identici ai CD-ROM) non pare siano emersi particolari problemi di durata dei supporti (a meno che non si tratti di esemplari difettosi), per cui sembra che una stima cautelativa di 15 anni di vita per un CD sia del tutto accettabile.

Possiamo quindi arrivare ad una prima conclusione, e cioè che esiste la possibilità di conservare i dati digitali anche a tempo indeterminato purché si provveda opportunamente alla gestione dei supporti, tramite il trasferimento dei dati su supporti più aggiornati, scegliendo i supporti tra quelli più standardizzati e documentati. Per contro, pare che i supporti analogici diamo migliori garanzie nel caso in cui questa gestione manchi: è alquanto improbabile che un CD dimenticato in un magazzino sia ancora utilizzabile tra cento anni, mentre è più probabile che sia leggibile un microfilm che, anche se fosse attaccato da muffe o altri agenti di deterioramento, potrebbe ancora essere decifrabile almeno in parte, mentre anche un degrado parziale può rendere del inutilizzabile un supporto digitale.

Conservazione dei formati. Non di rado si sente lamentare il fatto che i dati realizzati oggi in formato digitale, ad esempio immagini, tra pochi anni potrebbero non essere più utilizzabili per la mancanza del software in grado di elaborarli. Espressa così, la preoccupazione si basa su un presupposto quasi mai esplicitato e piuttosto ingenuo, e cioè che ogni formato di dati possa essere letto da un unico programma. Chiunque abbia un poco di esperienza di lavoro coi computer sa benissimo che questo non è vero: per esempio, il formato RTF può essere utilizzato dai pricipali word processor, qualunque editor di immagini appena decente è in grado di aprire file GIF, TIFF o JPEG, esistono innumerevoli riproduttori di file musicali mp3, e così via. Peraltro, si deve anche riconoscere che il presupposto ha degli elementi di verità se riferito alla leggibilità a lungo termine: può darsi, e anzi è probabile, che tra venti o trent'anni almeno alcuni dei formati oggi in uso diventino tanto obsoleti che nessuno più si preoccuperà di produrre del software che possa trattarli, e non parliamo di quello che succederà tra cinquanta o cento anni.

Un primo punto centrale sta dunque nel rapporto tra dati e programmi, e da quello che abbiamo detto si comprende facilmente che una strategia orientata alla conservazione a lungo termine deve assolutamente evitare una stretta associazione tra dati e programmi, e privilegiare invece l'indipendenza dei dati dai programmi. Ma perché questo sia possibile occorre in primo luogo evitare le soluzioni proprietarie, nelle quali dati e programmi sono proprietà di un produttore che non ne rende note le specifiche e i sorgenti, per cui l'utente è completamente dipendente dalle scelte del produttore stesso. Bisogna invece privilegiare i formati pubblicamente documentati, che sono la condizione perché possano essere sviluppati molteplici software in grado di gestirli, e il software libero che, rendendo pubblici i sorgenti, dà a tutti la possibilità di adattarlo, perfezionarlo ed eventualmente trasferirne in nuovi ambienti i principi e le tecnologie.

Una strategia di conservazione può quindi prevedere: da un lato la periodica conversione dei dati in formati più aggiornati (purché sempre documentati), dall'altro - a seconda dei casi - la ricompilazione, l'adattamento dei programmi già esistenti, o addirittura la scrittura ex novo di programmi in grado di utilizzare i dati. Risulta quindi del tutto pensabile una conservazione a lungo termine. Se - avendo adottato le strategie di conservazione dei supporti di cui abbiamo parlato al punto precedente - tra cento anni ci fosse la necessità di utilizzare delle immagini registrate in un formato in disuso da decenni, e questo formato fosse ben documentato, si potrebbe pur sempre scrivere appositamente il software necessario, o addirittura, se fossero disponibili i sorgenti del software impiegato in origine, riutilizzarne algoritmi e soluzioni. Naturalmente questo non sarebbe possibile se si trattasse di sistemi proprietari, nel qual caso l'unica soluzione sarebbe richiedere al produttore le necessarie informazioni, sperando che questi esista ancora, sia disponibile a comunicarle, e le abbia conservate nei suoi archivi.

Conservazione degli ambienti. Forse a molti non è mai accaduto di osservare che in una biblioteca è più facile trovare un catalogo a volume dell'ottocento o un catalogo a schede, pure manoscritte, in formato Staderini, piuttosto che un catalogo su computer di vent'anni fa, e non solo perché vent'anni fa poche biblioteche avevano un catalogo su computer, almeno in Italia: anche in quelle che lo avevano, nella migliore delle ipotesi si saranno conservati i dati, trasferiti in un catalogo più moderno, ma non sarà più possibile consultare il catalogo originale con il suo software originale. Questo introduce il tema della conservazione degli ambienti, e cioè della conservazione non solo dei dati ma di tutto l'ambiente originale di gestione degli stessi, o almeno delle parti più significative di esso (si può infatti distinguere tra la conservazione del solo software applicativo o anche del sistema operatico, ed eventualmente anche dell'hardware originale). Risulta infatti evidente il contrasto tra la cura con cui si conservano antichi cataloghi, inventari, indici, repertori ecc., anche quando i dati sono stati trasferiti in altri sistemi informativi, e la noncuranza con cui si abbandonano vecchi sistemi informativi elettronici che pure sono una testimonianza di interesse storico e culturale. Questa noncuranza, perlatro, non è assoluta, perché non mancano esempi di interesse ai vecchi software e al vecchio hardware.

Alla luce di queste considerazioni, non sembra che si possa dubitare che una strategia globale di conservazione dei documenti digitali debba anche prendere in considerazione la conservazione degli ambienti, per la quale si possono elencare almeno alcune possibili soluzioni, la cui applicabilità andrebbe valutata caso per caso perché dipende sia dalle particolari caratteristiche dell'ambiente che si vuole conservare sia, in generale, dalla sua lontananza rispetto alle architetture hardware e software correnti:

- **conservazione completa dell'hardware e del software originali**; questa soluzione naturalmente garantisce il perfetto mantenimento dell'ambiente originale, ma a lungo termine tende a diventare sempre più difficile da sostenere perchè è quanto meno probabile che si determini una sparizione dei pezzi di ricambio e dei supporti di memorizzazione idonei a permettere riparazioni dell'hardware e salvataggi dei software; inoltre questa soluzione tende a portare all'isolamento dell'ambiente conservato rispetto a quello corrente (un po' come se si avesse un'automobile storica che non può più circolare sulle strade aperte al traffico), a meno di non progettare opportune interfacce, compito che però potrebbe diventare sempre più difficile a causa della sempre maggiore distanza che certamente verrebbe a determinarsi tra l'ambiente conservato e le tecnologie correnti
- **conservazione del software originale e ricostruzione dell'hardware**; di per sé questa soluzione garantirebbe una conservazione dell'ambiente a tempo indeterminato, visto che prevede di ricostruire ciò che manca, ma rischia di essere di difficile applicabilità a causa dei costi: richiederebbe infatti la ricostruzione in piccolissimo numero di apparecchiature elettroniche obsolete se non addirittura quasi dimenticate, ricostruzione per la quale non è sufficiente un bravo tornitore, come potrebbe avvenire se si trattasse di dispontivi meccanici, ma si richiedono comunque attrezzature industriali e competenze tecniche di alto livello
- **abbandono dell'hardware e ricompilazione del software**; garantirebbe il riutilizzo del software originale in nuovi ambienti hardware (se si ricompila sia il sistema operativo che i software applicativi) o software (se si ricompilano i soli applicativi), ma non è affatto garantito a priori che si potrà sempre ricompilare il software: può essere che ad un certo punto non siano più disponibili compilatori adatti; in

certi casi si potrebbe rimediare con alcune modifiche al software da compilare, ma non è detto che questo sia sempre possibile, e inoltre si porrebbero dubbi sull'effettiva autenticità del software

- **abbandono dell'hardware e riproduzione del software originale tramite la scrittura ex novo di programmi identici come funzionalità e come interfaccia**; si tratterebbe non tanto di conservazione quanto di riproduzione degli ambienti la quale, comunque, se effettuata con criteri rigorosi, avrebbe pure sempre un valore documentario; tuttavia potrebbero esserci degli ambienti che rendono impossibile anche questa operazione: ad esempio, un sistema operativo che preveda la sola interfaccia grafica renderebbe probabilmente impossibile scrivere un software con interfaccia a carattere
- **abbandono dell'hardware e ricorso all'emulazione software**; sembrerebbe la soluzione di impiego più generale, con riferimento non tanto all'emulazione del singolo software, ma ai sistemi operativi basati sull'idea di macchina virtuale (come il vecchio VM della IBM e il recente VMware), dove un sistema operativo host implementa via software una emulazione di un ambiente hardware che a sua volta è in grado di eseguire un sistema operativo utente (è peraltro da dire che anche in questo caso l'implementazione potrebbe essere difficile e costosa).

Se al termine di queste considerazioni volessimo tentare di arrivare a qualche conclusione, potremmo proporre le seguenti:

- la conservazione dei documenti digitali è possibile se esiste una attenta gestione del problema, mentre sembra molto più dubbia nel caso in cui si abbia un abbandono dei documenti, caso nel quale sono probabilmente migliori i supporti analogici, dove applicabili
- la conservazione dei documenti digitali è oltremodo avvantaggiata dall'uso di software libero, disponibile coi sorgenti, e di formati, standard e tecnologie sotto ogni aspetto pubblici e documentati, mentre tende ad essere compromessa dall'uso di soluzioni proprietarie.

7. SCAMBIO DI DATI

Non tutti usano gli stessi software per la gestione di archivi di dati, né, a parità di software, tutti usato archivi strutturati allo stesso modo. Di qui l'esigenza di scambiare dati tra basi dati diverse, o di accedere, utilizzando un solo programma, a diversi archivi. La possibilità di effettuare queste operazioni tra diversi programmi e/o archivi viene detta, in linea generale, **compatibilità**, termine che è però quanto mai vago e anzi quasi privo di senso se non si specifica esattamente tra che cosa vi è compatibilità e in quali limiti.

Lo scambio di dati può avvenire in due modi, ossia **online**, che in questo contesto non significa che avvenga per via telematica, ma che un software di gestione di archivi di dati accede direttamente a dati prodotti con un altro programma, oppure **in batch**, quando i dati vengono **esportati** da un archivio e poi **importati** in un altro. L'operazione online può anche non essere uno scambio di dati: un programma può operare direttamente su un archivio esterno senza trasferire i dati in un proprio archivio. Si può parlare quindi anche semplicemente di compatibilità o interoperabilità.

La compatibilità tra basi di dati può essere di due tipi:

- **compatibilità fisica** quando i dati hanno lo stesso formato a basso livello, ad esempio quando sono tutti database di Access oppure di Oracle
- **compatibilità logica** quando gli archivi di dati hanno lo stesso tracciato record; si può parlare di compatibilità anche quando il tracciato record non è esattamente lo stesso ma le differenze sono limitate a questi casi: uno dei due database ha dei campi in più, ma i campi in comune sono uguali; un campo nel database di destinazione è più lungo del campo corrispondente nel database di origine; un campo nel database di destinazione è di un tipo più ampio del campo corrispondente nel database di origine (questo vale soprattutto per i campi numerici: se il campo di origine può contenere solo numeri interi, mentre quello di destinazione tutti i numeri reali, il trasferimento dei dati potrà avvenire perché gli interi sono un sottoinsieme dei reali); un campo nel database di origine prevede dei vincoli più ristretti del campo corrispondente nel database di destinazione; inoltre, di solito non vi sono problemi di trasferimento dei dati

se tra un archivio e l'altro cambiano solo i nomi dei campi, mentre corrispondono tutte le altre caratteristiche degli stessi

Questi due generi di compatibilità possono trovarsi da soli o assieme. Vediamo quindi i diversi casi possibili:

1. **solo compatibilità fisica**; contrariamente a quello che potrebbe sembrare, è il caso meno favorevole; infatti se abbiamo parecchi database in formato, ad esempio, xBase, basta installare un qualunque programma che riconosca questo formato e potremo senza problemi operare **separatamente** su ciascuno di questi database; se però il tracciato record è diverso, sarà difficile trasferire dati perché non vi sarà corrispondenza tra i campi dei vari archivi; ad esempio, in un archivio la formulazione di collezione potrebbe essere contenuta in un unico campo destinato alla descrizione ISBD, in un altro potrebbe trovarsi in un campo a sé stante, e in un terzo addirittura in una tabella a parte; inoltre in generale query, form, procedure e altri componenti progettati per un certo tracciato record non funzioneranno su di un altro; ciò non vuol dire, comunque, che la compatibilità fisica sia in generale inutile: ci sono infatti molti casi in cui non si vuole trasferire completamente i dati di un archivio in un altro, ma solo consultare i dati in formati eterogenei, importarne solo un particolare campo ecc.; in questi casi, evidentemente, la compatibilità fisica è utilissima; questo è il motivo per cui molti programmi di gestione database sono i grado di riconoscere dall'origine un gran numero di formati diversi; una tecnica molto diffusa per generalizzare la compatibilità fisica è quella di utilizzare uno strato di software standardizzato che permetta ad un programma di gestione di database di accedere in modo trasparente a formati diversi; naturalmente ci vorrà un software specifico per ciascun formato da utilizzare, mentre è standardizzata l'interfaccia verso il programma utilizzatore; una diffusissima implementazione di questa architettura è la tecnologia ODBC (*Open Database Connectivity*), utilizzata con gli RDBMS; quando un RDBMS supporta la tecnologia ODBC è sufficiente fornirgli i driver specifici per un certo formato perché esso possa leggere tutte le tabelle registrate in quel formato, effettuare join e tutte le altre operazioni necessarie
2. **compatibilità logica e fisica**; è in linea di massima il caso più favorevole; in particolare, è possibile fondere senza difficoltà due database diversi; tuttavia talvolta questo non è sufficiente: spesso infatti può essere necessario identificare e fondere i record duplicati; altre volte possono essere gli identificatori univoci a porre dei problemi: essendo stati assegnati in modo indipendente nell'ambito di database diversi, uno stesso identificatore può essere stato impiegato in tutti i database da fondere, per cui deve essere modificato per preservare l'univocità; se l'identificatore era stato utilizzato anche per i riferimenti nella tabella intermedia di un join n a m, deve essere corretto anche lì in modo appropriato
3. **solo compatibilità logica**; in questo caso i dati sarebbero, per così dire, proprio quelli giusti, pronti per essere utilizzati (salvo i possibili problemi descritti poc'anzi), ma non si possono leggere perché il formato dei file a livello fisico non è riconosciuto dal programma di gestione dell'archivio; l'unica soluzione è quindi usare un formato intermedio riconosciuto da entrambi i programmi in gioco: il primo scriverà i dati in questo formato, e il secondo li leggerà, trasferendoli nel suo archivio [38]; ci sono molti formati utilizzabili a questo scopo: a parte Unimarc, cui viene dedicata un trattazione particolare, è molto diffuso il formato *delimitato* (detto anche CSV = *Comma Separated Values*, cioè *valori separati da virgola*), in cui ogni record corrisponde ad una riga, mentre i campi sono separati tra loro da un carattere prestabilito, di solito una virgola; il contenuto dei campi testo, in cui potrebbe facilmente presentarsi il delimitatore, viene normalmente racchiuso tra virgolette per evitare ambiguità; vi sono poi molti tipi di formati *etichettati*, in cui ogni riga corrisponde ad un campo; il contenuto del campo è preceduto da un etichetta identificativa; alcuni programmi consentono soltanto di utilizzare un certo numero di formati predefiniti, altri invece permettono di creare formati personalizzati

Da quello che abbiamo detto, dovrebbe risultare chiaro che il caso 1 e 2 permettono scambi di dati sia online che in batch, mentre il caso 3 solo in batch.

[38] Strettamente parlando, il primo programma potrebbe anche essere in grado solo di scrivere in quel formato e il secondo solo di leggere, ma normalmente - non essendoci programmi destinati per principio solo all'esportazione dei dati e altri solo all'importazione - se un programma riconosce un formato lo riconosce tanto in lettura quanto in scrittura

Come accennato, per poter condurre a buon fine uno scambio di dati possono essere necessarie parecchie operazioni aggiuntive, che spesso sono tutt'altro che semplici e possono anche richiedere lo sviluppo di programmi sofisticati (si veda anche quanto detto nel capitolo sugli strumenti di sviluppo). Le principali di queste operazioni possono essere raggruppate in due categorie:

- **controllo dei duplicati**; implica due fasi: l'identificazione dei record doppi e la scelta di quello da conservare (oppure la creazione di un nuovo record che combini dati di entrambi quelli di origine); per l'identificazione dei doppi non basta trovare i record che sono identici in ogni singolo carattere, ma bisogna anche trovare quelli che, pur riferendosi alla stessa cosa, presentano tra loro differenze più o meno grandi (si pensi a descrizioni bibliografiche della stessa edizione che differiscano solo per qualche dettaglio), e questo è un compito molto difficile, perché ci può essere il rischio di considerare doppi record che sono solo accidentalmente simili; una volta identificati i doppi, ci vuole poi un criterio per decidere quale dei due record conservare, altra cosa non facile; quando si tratta di controllo dei doppi, spesso il programma si limita a fare una prima scelta dei sospetti record duplicati, rimandando la decisione definitiva all'intervento di operatori umani
- **riformattazione**; sono quelle operazioni che modificano il tracciato record originale oppure la presentazione dei dati; ad esempio, se alcuni dati nel database di origine sono suddivisi tra due campi, mentre in quello di destinazione sono contenuti in uno solo, bisognerà concatenare i dati di origine in modo che possano essere inseriti in un unico campo; in generale, è abbastanza facile fondere dati che erano suddivisi, mentre può essere difficilissimo, o anche impossibile, separare dati che erano uniti, perché può mancare un criterio univoco per determinare dove va fatta la separazione; per quanto riguarda la presentazione dei dati, può essere necessario produrre punteggiatura o altri elementi non previsti nell'archivio di origine ma necessari in quello di destinazione (ci sono, ad esempio, programmi nei quali la punteggiatura ISBD fa parte dei dati e altri che invece la producono automaticamente solo in fase di output), oppure eliminarli nel caso contrario; bisogna tener presente che normalmente è più facile trasferire dati da un archivio di struttura complessa a uno semplice che non il contrario; la riformattazione può essere fatta in fase di esportazione, oppure sui dati esportati, oppure in fase di importazione, o ancora dopo l'importazione, elaborando i dati importanti nel programma di arrivo (in questo caso in genere si ricorre ad un database di appoggio diverso da quello definitivo); a volte, a seconda dei programmi che sono in gioco, la riformattazione viene anche fatta in diversi momenti, ad esempio sia durante l'esportazione che durante l'importazione.

Nel corso del tempo sono stati sviluppati diversi formati standard specificamente progettati per l'interscambio in batch di dati bibliografici. Di interesse ormai quasi esclusivamente storico è il formato MEKOF, sviluppato nell'URSS e utilizzato nei paesi socialisti[39] (dove peraltro l'automazione delle biblioteche era ben poco sviluppata), mentre il formato MAB ha tuttora un certo impiego in Germania. Tuttavia l'unico standard di fatto affermatosi su scala internazionale per l'interscambio di dati bibliografici è il formato Unimarc, al quale è dedicato il prossimo paragrafo.

7.1 Formato Unimarc

Il formato Unimarc è nato negli anni '60 con il nome di Marc (Machine Readable Catalog). Successivamente si svilupparono molte varianti particolari del Marc, per lo più su base nazionale[40]. Tutte queste varianti sono poi state riunificate appunto con il nome di Unimarc[41].

Scopo di Unimarc è quello di permettere lo scambio di dati bibliografici anche al massimo di livello di completezza come richiesto, ad esempio, dalle bibliografie nazionali, per cui è piuttosto complicato e prevede tutte le informazioni che possono avere una qualche rilevanza nell'ambito bibliografico, anche se alcune sono

[39] Sono comunque disponibili alcuni software di conversione di dati bibliografici sviluppati recentemente in Russia in grado di trattare il formato MEKOF

[40] La versione italiana era chiamata ANNAMarc, dove ANNA sta per Automazione Nella Nazionale, nome dato al primo progetto di automazione della Nazionale di Firenze

[41] Un utile presentazione di Unimarc, valida soprattutto per gli aspetti biblioteconomici, è costituita da *[Unimarc 2000]*

utilizzate molto raramente. Unimarc inoltre nasconde completamente le caratteristiche del programma di origine, e non pone particolari vincoli per quanto riguarda il programma di destinazione. È responsabilità dei programmi che rispettivamente scrivono e leggono l'Unimarc l'assicurare la corrispondenza appropriata tra i dati presenti nel database e quelli prodotti in formato Unimarc nel caso della scrittura, e l'appropriato inserimento nel proprio database dei dati tratti da Unimarc nel caso della lettura. Unimarc è un formato per lo scambio di dati in batch, ossia tramite una esportazione e una successiva importazione, e non è adatto come formato interno da utilizzare per il trattamento dei dati con un motore relazionale o di information retrieval.

Unimarc è basato sullo standard ISO 2709. Questo standard è denominato *Scambio di dati bibliografici su nastro magnetico* (in realtà può però essere usato su qualsiasi supporto), ma prevede solo una sintassi dei dati e non anche una semantica, per cui determina come deve essere strutturato il file, ma non che cosa deve contenere, e quindi può essere utilizzato per archivi di qualunque tipo e di qualunque struttura. Vi sono alcuni software, come CDS-ISIS e Highway che sono predisposti per esportare ed importare dati in ISO 2709. Unimarc aggiunge all'ISO 2709 una semantica, cioè determina quali dati possono essere presi in considerazione e in quali campi devono essere inseriti, per cui stabilisce, ad esempio, che l'area 1 della descrizione ISBD si trova nel campo 200, l'area 2 nel campo 201 ecc. In ogni caso, i dati Unimarc sono sempre conformi allo standard ISO 2709, mentre non vale il contrario (cioè vi possono essere dei dati ISO 2709 che non sono Unimarc, ad esempio quelli originati dalla procedura di scarico in ISO di CDS-ISIS).

Ecco le principali caratteristiche di Unimarc. Un file Unimarc è un file di testo senza fine riga (taluni programmi dividono il file in righe allo scopo di facilitarne l'ispezione con un editor; questa caratteristica di per sé è al di fuori dello standard ma è piuttosto diffusa, ed indubbiamente comoda) in cui i record sono separati da un apposito marcatore. Si tratta di un file piatto che non prevede strutture di dati simili alle tabelle od operazioni simili ai join. Ogni record è diviso in tre parti:

- la **label** che contiene informazioni generali sul record, tra cui la lunghezza dello stesso
- l'**header**, che è un indice dei campi presenti nel record e della loro posizione
- i **dati**.

I dati sono registrati in campi di lunghezza variabile, identificati da un numero, suddivisi in sottocampi. I sottocampi sono identificati da un carattere apposito seguito da un lettera che varia a seconda del sottocampo. All'inizio dei campi, prima dei dati veri e propri, vi possono essere gli **indicatori**, che sono codici alfabetici o numerici che indicano determinate caratteristiche del campo che non fanno parte dei dati (si tratta quindi di metadati). Come osservato, Unimarc prevede un tracciato molto dettagliato che contiene, in particolare, la descrizione ISBD, gli autori, i soggetti, i titoli collegati, informazioni amministrative ecc. Il tracciato viene determinato unicamente dallo standard (il quale a sua volta è basato sulle norme catalografiche) e non dalle caratteristiche del programma di origine dei dati.

Si tratta, come si nota facilmente di una struttura che rammenta molto di più quella di un IRS che quella di un RDBMS. Notevole il fatto che vi sono archivi, come il popolare Teca, basato su CDS-ISIS, che hanno un tracciato record strettamente ricalcato sull'Unimarc (Teca se ne differenzia solo per i numeri dei campi).

Unimarc è un formato piuttosto complicato da trattare, ma è l'unico formato standard di interscambio di dati bibliografici: qualunque archivio Unimarc può essere prodotto o importato da qualsiasi programma che riconosca questo formato. Negli ultimi anni è di molto aumentato il numero di programmi che supportano l'Unimarc. Tra quelli diffusi in Liguria, ricordiamo Sebina, Erasmo e Aleph, mentre Teca è un caso singolare: pur avendo, come detto, un tracciato record identico a quello Unimarc, non pare sia stato ancora realizzato un programma di esportazione[42], mentre ci sono possibilità di importazione. Degno di nota è il fatto che CDS-ISIS può importare qualunque file Unimarc ma non in quanto Unimarc, bensì in quanto ISO 2709, per cui questa importazione non tiene conto delle caratteristiche specifiche dell'Unimarc e può richiedere ulteriori elaborazioni per dare un risultato pienamente soddisfacente.

[42] Un simile programma ovviamente non potrebbe limitarsi ad esportare i dati pari pari, ma dovrebbe produrre una label e un header corretti

8. BASI DI DATI IN RETE

La condivisione di basi di dati in rete ha lo scopo di permettere a molti utenti di usufruire di una stessa copia della base, in modo da poter utilizzare i dati creati da altri. Infatti quando molti inseriscono dati su copie diverse dello stesso archivio si pone il problema di mantenere sincronizzate queste copie, evitando dati duplicati oppure inseriti in modo incoerente. Quando invece le basi di dati devono essere rese disponibili per la sola consultazione il problema è soprattutto la facilità di accesso, che a seconda delle circostanze si può ottenere più facilmente attraverso l'uso delle reti oppure duplicando la base dati, come prova il successo delle basi dati distribuite su CD-ROM.

Ci sono fondamentalmente tre tecniche per utilizzare le basi di dati in rete, che possono essere adottare indifferentemente con RDBMS e con IRS. La scelta non dipende solo dai vantaggi e svantaggi intrinseci di queste tecniche, ma anche dai programmi e dai sistemi operativi utilizzati, poiché non ogni programma rende possibile indifferentemente l'uso di queste tecniche, le quali sono:

- **file system condiviso**; un file system condiviso è un file system che si trova su una macchina collegata in rete, e che altre macchine possono utilizzare come se fosse un file system locale; in linea di principio, qualunque computer in rete può rendere condivisi i suoi file system, ma preferibilmente si usano dei computer destinati solo a questo scopo, più affidabili di un normale PC - anche se possono utilizzare la stessa architettura hardware, in particolare quella dei computer IBM compatibili - e dotati di memorie di massa molto grandi, che sono detti **server** e in particolare **file server**, perché hanno essenzialmente lo scopo di rendere dei files disponibili agli utenti; dal punto di vista della macchina con usa il file system di rete, esso viene visto secondo le convenzioni del file system locale, e quindi nelle macchine Windows viene identificato da una lettera di unità (ad esempio E:, K: e simili) mentre nelle macchine Unix viene visto come una directory tra le altre (ad esempio /usr/netfs); non è affatto necessario che il server e le stazioni di lavoro usino lo stesso sistema operativo, ma è sufficiente che usino lo stesso protocollo di condivisione di file system (di solito di usa SMB coi server Windows NT ed NFS coi server Unix, ma non è una regola assoluta[43]); è quindi possibile collocare un archivio di dati in un file system condiviso ed installare invece il programma di gestione sulle stazioni di lavoro; a questo punto sarà sufficiente istruire il programma a cercare i dati non sui file system locali, ma su quello condiviso; ad esempio, se un computer Windows l'unità di rete è K:, e i dati si trovano nel file k:\data\biblio.dbf, si dovrà indicare quel percorso al programma quando gli si chiede di aprire il file; normalmente bisogna anche indicare al programma che il database è condiviso, affinché esso metta in atto i comportamenti appropriati, ad esempio applichi il blocco dei record; in questa architettura ci saranno quindi molti esemplari del motore relazionale o information retrieval installate sui computer degli utenti, ed un sola copia dell'archivio, installata in un file system condiviso del server; questo facilita, tra l'altro, i salvataggi, che devono essere effettuati solo sul server, e non su ogni computer della rete; è possibile collocare sul file system condiviso non solo i dati, ma anche il programma: in questo caso ogni utente avvierà il programma da questo file system invece che da quello locale, ma **il programma verrà eseguito sulla stazione di lavoro e non sul server** (dove si trova fisicamente memorizzato); quest'ultima soluzione è utile se si vuole centralizzare l'installazione e la configurazione del software; nulla impedisce, infine, che alcune stazioni di lavoro eseguano il programma dal loro file system locale e altre invece da quello di rete
- **sistema time-sharing (multiutenza)**; un sistema time-sharing è quello in cui il tempo di esecuzione è suddiviso tra molti utenti, i quali utilizzano tutti lo stesso computer non solo per memorizzarvi i dati, ma anche per eseguirvi i programmi; a questo scopo, in passato si impiegavano terminali collegati al server

[43] Negli ultimi anni ha avuto grande successo, soprattutto in ambiente Linux, Samba, che è una implementazione di SBM funzionante sotto Unix e che utilizzata TCP/IP come protocollo di trasporto e di rete, per cui permette di condividere i filesystem tra macchine Unix e Windows, oltre ovviamente che tra macchine solo Unix. Esiste una implementazione per Linux anche dell'architettura di rete Macintosh Aplletalk che include anche il protocollo Apple Share per la condivisione dei file system, il che permette la condivisione tra macchine MacOS (il sistema operativo dei Macintosh) e macchine Linux.

tramite la porta seriale; questi terminali non eseguivano programmi, ma avevano solo il compito di visualizzare gli output prodotti dal sistema operativo o dai programmi applicativi e di inviare al server gli input dell'utente; essi avevano quindi lo stesso ruolo del monitor e della tastiera di un normale PC, ma potevano essere collocati anche a grande distanza dal server; oggi di solito invece di terminali si usano dei PC che possono essere collegati al server o ancora tramite porta seriale o più frequentemente in rete locale o geografica; questi PC eseguono un programma, normalmente molto piccolo, che ha lo scopo di emulare un terminale; in ambiente TCP/IP si usa a questo scopo il protocollo Telnet, per cui l'emulatore di terminale è propriamente un client telnet che dialoga con un processo server telnet in esecuzione sulla macchina server[44]; in ogni caso, l'effetto è sempre quello di eseguire, dal proprio computer, programmi su una macchina remota, utilizzando i file system e le periferiche accessibili, localmente o via rete, su tale macchina **e non su quella dove è in esecuzione l'emulatore di terminale**; anche in questo caso il server e le stazioni di lavoro possono utilizzare sistemi operativi diversi[45]; è chiaro come questa architettura può essere utilizzata per l'accesso a basi dati: l'utente si collega ad un server e lì esegue il programma di gestione della base dati, accedendo ad archivi conservati non sul suo computer, ma sul server, od eventualmente anche su un file system di rete accessibile al server, mentre il computer dell'utente esegue solo il programma di emulazione di terminale; tutti gli utenti quindi eseguono lo stesso programma, il quale però può attivare configurazioni personalizzate a seconda dell'utente che si collega (ad esempio alcuni possono essere autorizzati ad inserire dati, altri solo a ricercarli); questa tecnologia viene impiegata in tutti i poli SBN attuali nonché nel Catalogo Bibliografico Trentino

- **client-server**; nell'architettura client-server i compiti del programma vengono suddivisi in due parti, e quindi in due programmi distinti, che di solito sono in esecuzione su macchine diverse (le quali a loro volta vengono chiamate client e server); il **client** è quello che viene direttamente maneggiato dall'utente, ed invia richieste al **server**, il quale risponde alle richieste del client; nel campo dei database, questo significa che il client ha il ruolo di front-end, quindi fa da interfaccia utente; inoltre di solito gestisce localmente le stampe e può anche gestire alcuni dati non destinati all'archivio condiviso; il server è costituito essenzialmente dal motore relazionale o di information retrieval, e quindi si occupa - oltre ovviamente che del colloquio con il client - della vera e propria gestione della base dati, la quale risiede sul file system locale del server o su un filesystem di rete ad esso accessibile (ma non sul file system della macchina client); client e server sono due programmi distinti i quali comunicano attraverso un qualche opportuno protocollo, per cui con uno stesso server possono essere utilizzati, in linea di principio, diversi client e viceversa; ad esempio, qualunque client Z39.50 può interrogare qualunque server Z39.50; come ovvio, anche in questo caso client e server possono usare sistemi operativi diversi tra di loro; esistono anche architetture client server a più livelli, in cui il client comunica con un server si primo livello che a sua volta fa da client per un server di secondo livello; inoltre i compiti tra client e server possono essere distribuiti in vario modo: ci possono essere client leggeri che svolgono solo il ruolo di accettazione dell'imput e visualizzazione dell'output (e alla fine tendono a coincidere, almeno concettualmente, con gli emulatori di terminale) e client pesanti che svolgono localmente quante più funzioni possibile; su Internet il browser WWW, che è un client per il protocollo HTTP, viene sempre più impiegato come interfaccia per l'accesso a basi dati; esso però non è un client specifico per questo scopo, e di basi dati non sa nulla, poiché è progettato solo per dialogare con server HTTP e non con un server di database; per questo motivo è necessario, lato server, un ulteriore componente software, un gateway, che interfacci il server HTTP con il motore di database; **l'argomento delle basi dati su Internet è trattato in maggior dettaglio nella dispensa sulle reti**

Vediamo ora quali sono i vantaggi e gli svantaggi delle diverse soluzioni.

---

[44] Il telnet si usa tipicamente per il collegamento alle macchine Unix, emulando per lo più i terminali della serie VT (VT-52, VT-100, VT-220 ecc.), che erano terminali prodotti dalla Digital; per il collegamento ai mainframe IBM si usa invece il protocollo Tn3270, funzionalmente analogo ma progettato per emulare appunto i terminali IBM 3270

[45] Questo anzi avviene molto spesso, perché mentre il client telnet è diffuso pressoché in tutti i sistemi operativi, il server o demone telnet è presente in tutte le versioni di Unix ma non, per ora, in Windows NT server, anche sono disponibili prodotti di terze parti; la Microsoft tuttavia prevede di introdurre una tecnologia funzionalmente analoga nelle prossime versioni di NT server

File system condiviso.

- Vantaggi
  - è di semplice realizzazione, soprattutto con le reti Windows che facilitano molto la condivisione dei file system
  - è intuitivo per l'utente abituato a lavorare su PC isolati, che non percepisce in pratica alcuna differenza nel modo di lavorare
  - permette di utilizzare il software di gestione database anche per archivi sul file system locale
  - se gli utenti non sono moltissimi funziona anche con server di modesto livello o, in piccole reti, addirittura utilizzando le unità di una stazione di lavoro
  - non è necessario molto spazio disco sulle stazioni di lavoro, perché queste non devono contenere i dati
- Svantaggi
  - la gestione del database rimane affidata ai programmi delle stazioni di lavoro, che di solito sono programmi per PC, e sono versatili e anche di semplice utilizzo, ma possono non avere la potenza e l'affidabilità di programmi concepiti per grossi sistemi
  - le stazioni di lavoro devono essere in grado di eseguire il programma di gestione della base dati, per cui possono essere necessarie macchine di potenza elevata
  - in caso di guasto del server o della rete non è possibile lavorare, poiché le stazioni di lavoro dispongono del programma ma non dei dati (si può però utilizzare il programma con dati che non siano in rete)

Time sharing.

- Vantaggi
  - richiede pochissime risorse alle stazioni di lavoro, permettendo di utilizzare senza alcun inconveniente anche computer molto vecchi
  - la configurazione delle stazioni di lavoro è semplicissima
  - permette di usare programmi concepiti per grossi sistemi, che quindi sono in generale più sofisticati, potenti ed affidabili[46]
- Svantaggi
  - in caso di guasto del server o della rete non è possibile lavorare, poiché le stazioni di lavoro non dispongono né del programma né dei dati
  - richiede un server piuttosto potente, perché esso deve eseguire tutta l'elaborazione per tutti gli utenti
  - non è possibile effettuare localmente alcuna elaborazione e quindi tutte, in particolare qualunque stampa, devono passare attraverso la rete ed il server
  - a volte non sono supportati tutti i caratteri speciali (es. lettere accentate), oppure sono supportati solo tramite procedure relativamente complicate, e la mappatura della tastiera può essere diversa da quella abituale
  - normalmente si usano interfacce a carattere (perché altrimenti il server avrebbe il carico anche dell'elaborazione grafica per tutti gli utenti) che alcuni, abituati alle interfacce grafiche, potrebbero trovare meno gradevoli

- **Client-server**
- Vantaggi
  - permette di avere programmi potenti e sofisticati sia lato client che lato server
  - permette di utilizzare client eterogenei, a seconda delle necessità, mantenendo sempre lo stesso server

[46] Vi sono programmi che hanno sia versioni per Dos/Windows che possono usare unità condivise, sia versioni Unix accessibili in emulazione di terminale: è il caso di CDS-ISIS; Sebina Produx funziona invece in monoutenza sotto Dos, in emulazione terminale sotto Unix e in architettura client-server sotto NT

    - ♦ permette di gestire localmente diverse funzionalità, come la stampa e a volte anche una parte dei dati, (sempre che il client sia stato progettato in questo modo) per cui può esserci una certa capacità di lavoro anche in caso di guasto del server o della rete
    - ♦ permette di realizzare client grafici di uso piacevole ed immediato
    - ♦ permette di raggiungere un elevato grado di sicurezza e protezione dei dati, soprattutto nell'architettura three-tier (v.seguito)
- ♦ Svantaggi
    - ♦ richiede una notevole potenza di elaborazione dal lato server, e può richiederla anche dal lato client se si usano client pesanti
    - ♦ richiede operazioni di installazione e di configurazione, che possono non essere semplici, non solo sul server, ma anche su tutti i posti di lavoro client

In pratica, le unità condivise vengono utilizzate prevalentemente in piccole reti, mentre per installazioni più grosse e sofisticate si preferiscono le altre due soluzioni. L'emulazione di terminale è la soluzione tradizionale, anche perché un tempo era l'unica possibile, che ha perso terreno in favore del client-server. Tuttavia i client pesanti hanno il difetto di richiedere macchine potenti ed aggiornate, ed essendo programmi in genere molto più complessi del client telnet o tn3270 possono richiedere parecchio lavoro di installazione, configurazione, manutenzione ed assistenza agli utenti. Per questo negli ultimi anni è emersa una tendenza all'uso di client sempre più leggeri (intendendosi per client sia il computer che il software) che delegano al server tutta l'elaborazione più pesante e quindi concettualmente finiscono per assomigliare ai vecchi terminali o emulatori, anche se prevede di mantenere l'interfaccia grafica gestita localmente dal client.

Il client-server descritto finora è chiamato *a due livelli* o *two-tier*, ed è il tipo più semplice di client-server. Per applicazioni particolarmente sofisticate si usa però il client server *a tre livelli* o *three-tier*. In questo caso si hanno, in linea generale (precisazione necessaria, perché ci possono essere varianti a seconda dell'implementazione) i seguenti componenti:

- un componente client che fa da front ent e gestisce l'interfaccia utente
- un componente intermedio detto **middleware** diviso a sua volta in client e server, che sovraintende alla logica applicativa, nel senso che il lato client passa dei comandi (che naturalmente dipendono dalle azioni che l'utente effettua sul front-end) al lato server il quale provvede ad attivare le necessarie risorse sul sistema server; poiché stiamo parlando di database, queste azioni saranno tipicamente procedure attivate nell'ambito di un sistema di gestione di basi di dati, ma potrebbero essere anche altre cose, ad esempio il trasferimento di un file; il lato client del middleware può anche implementare una gestione più o meno sofisticata delle transazioni e in particolare dello scheduling, e quindi può finire per assumere le funzioni di un TP monitor (oppure il TP monitor può sussistere come componente distinto dal lato client del middleware, nel qual caso potrebbe essere legittimo parlare di client-server a quattro livelli)
- un back-end consistente in un RDBMS o in un IRS che svolge le vere e proprie operazioni sugli archivi di dati

Le applicazioni di database basate sul web sono classificabili come client-server three tier, ma con la particolarità che manca il lato client del middleware, mentre il lato server è diviso in due parti strettamente integrate, cioè server http e gateway di database: infatti sul lato client abbiamo il browser web che, tramite il server http, attiva delle procedure che a sua volta ordinano ad un sistema di gestione di basi dati di eseguire determinate operazioni, come una ricerca o un inserimento. Non mancano le possibili varianti: ad esempio, il browser web potrebbe scaricare ed eseguire un applet in Java che fa da client middleware interagendo non più con il server web ma con uno specifico software lato server.

Il discorso sulle basi di dati in rete non sarebbe però completo se non si trattasse anche delle architetture distribuite progettate per collegare, attraverso una rete (o anche su una singola macchina), diversi componenti detti oggetti, tra i quali possono esservi anche basi di dati. Si tratta di architetture client-server, nelle quali l'elemento chiave è l'**ORB** (Object Request Broker = Intermediario per le richieste di oggetti). L'ORB è a sua volta un elemento distribuito, ossia formato in generale di più componenti che funzionano su macchine

diverse che interagiscono via rete, ed ha il compito di gestire le richieste di accesso agli oggetti originate dalle varie applicazioni. In questa architettura, il client non ha bisogno di conoscere in precedenza dove si trovano le varie risorse, poiché quando ne ha bisogno formula una richiesta all'ORB, che si incarica di reperirla. L'interazione è possibile perché i vari componenti colloquiano tramite interfacce standard comuni a tutti. Si tratta evidentemente di architetture client-server almeno a tre livelli (front-end, ORB, risorsa finale). Per questo genere di architetture distribuite ci sono due standard principali: CORBA (Component Object Request Broker Architecture), promosso da un insieme di soggetti riuniti nell'Object Management Group (OMG[47]) e DCOM (Distributed Component Object Module), promosso essenzialmente dalla Microsoft.

8.1 Z39.50 e Metaopac

È utile fornire qualche notizia in più sul protocollo Z39.50 (tutte le informazioni tecniche e meno tecniche sull'argomento, compreso il testo ufficiale dello standard possono essere reperite a partire dal sito http://lcweb.loc.gov/z3950/agency/). Si tratta di un protocollo di livello applicativo, che oggi viene normalmente impiegato utilizzando TCP come protocollo di trasporto, destinato a standardizzare le operazioni di interrogazioni di basi dati, rendendole completamente indipendenti dalla struttura logica e fisica delle basi di dati che vengono interrogate. L'unica condizione è che tali strutture possano venire mappate, da parte del server Z39.50, nella semantica prevista dal protocollo. Non si deve credere che Z39.50 sia utilizzato solo nel campo bibliografico: infatti possono essere definiti degli *attribute set* (insiemi di attributi) utili per trattare qualsiasi tipo di dati, anche se quello per applicazioni bibliografiche detto BIB1, è di gran lunga il più utilizzato. Ogni client Z39.50 (detto *origin* nella terminologia del protocollo) può interrogare ogni server Z39.50 (detto *target*). Il protocollo è orientato alla connessione, e permette ricerche estremamente sofisticate. Alla potenza del protocollo fa però riscontro la sua notevole complessità, che rende alquanto complicato lo sviluppo tanto dei client quanto dei server. Su Internet vi sono molti server Z39.50, ma la maggior parte del pubblico li consulta non attraverso un client nativo, ma via http con un normale browser: questo è possibile grazie all'esistenza di gateway Z39.50/HTTP, che sono a loro volta programmi di realizzazione non banale, soprattutto perché la natura stateless di http è del tutto opposta al funzionamento orientato alla connessione tipico di Z39.50. Normalmente via http si può utilizzare sono una parte delle funzionalità che Z39.50 metterebbe a disposizione. Questa architettura può essere considerata un client-server a quattro livelli. Abbiamo infatti: browser web, server http+gateway http/Z39.50, server Z39.50, motore di database.

Poiché ogni client Z39.50 può interrogare indifferentemente ogni server, l'operazione può essere automatizzata, facendo sì che il client sottoponga automaticamente ad una serie di target l'interrogazione effettuata dall'utente. Un software di questo genere viene detto metaopac. La stessa operazione può essere fatta anche seguendo un approccio diverso, cioè sezna Z39.50 utilizzando un software che interroghi i server via http nel linguaggio nativo di ciascuno (ossia passando loro un URL che avvii l'interrogazione). Il dibattito sui vantaggi e svantaggi dei metaopac non Z39.50 rispetto a Z39.50 è tuttora aperto e molto interessante. Un metaopac bibliografico molto importante, anzi forse il più grande del mondo, è stato realizzato in Italia dal CILEA di Segrate, e viene gestito in collaborazione con l'Associazione Italiana Biblioteche (AIB). Si chiama Azalai, interroga attualmente oltre 100 OPAC bibliotecari italiani e si può raggungere a partire dal sito dell'AIB http://www.aib.it[48]

9. BASI DI DATI BIBLIOGRAFICHE

Sarebbe perfettamente inutile impiegare basi di dati nelle biblioteche se poi con queste non si potessero gestire i dati bibliografici. Ci si può dunque chiedere: quale tipo di software è più adatto per questo impiego ? gli IRS o gli RDBMS ? Uno sguardo al mercato mostra che vengono impiegati con successo entrambi i tipi di

[47] Il sito Web dell'OMG è: http://www.omg.org

[48] Su Internet ci sono diversi metaopac non Z39.50 che interrogano non database bibliografici, ma motori di ricerca, per cui piuttosto che metaopac vengono detti metamotori. Tra essi ricordiamo gli ottimi Infozoid (http://www.infozoid.com/), The Big Hub (http://www.thebighub.com/), che prima si chiamava Internet Sleuth, e Vivisimo (http://www.vivisimo.com/).

software, anche se con una certa prevalenza degli RDBMS. Ad esempio: CDS-ISIS è un information retrieval puro, Highway (utilizzato per il Catalogo delle Biblioteche Liguri) è un information retrieval, Tinlib si basa su Tinman che è un RDBMS ibrido che cerca di includere funzionalità da IRS, Sebina si basa su Progress che è un RDBMS, Erasmo si basa su Access o SQL Server, entrambi RDBMS, Aleph utilizzava all'inizio un motore di database realizzato appositamente mentre ora usa Oracle, uno dei più celebri RDBMS, ma cerca di includere anche funzionalità di information retrieval, le varie versioni di SBN usano degli RDBMS come SQL/DS, Adabas, DB2, Ingres.

Non in tutti i campi accade che si possano usare con successo entrambi i tipi di prodotto. Per esempio, difficilmente qualcuno userebbe un information retrieval per realizzare applicazioni di contabilità o gestione magazzino, né qualcuno ricorrerebbe ad un RDBMS per un applicazione di ricerche full text. Come mai allora nel campo bibliografico accade questo? Perché, anche se spesso questo non viene messo in evidenza, i dati bibliografici hanno una natura ambigua. Da un certo punto di vista si possono analizzare in termini adatti ad un relazionale: descrizione, titolo, autore, soggetto, inventario, collocazione e quant'altro si possono considerare elementi atomici o comunque con una struttura fissa, rappresentabili in tabelle tra le quali si possono enucleare delle possibili associazioni: ad esempio una associazione n a m tra titoli e autori (un autore può essere responsabile di molti titoli e un titolo può avere molti autori), tra titoli e soggetti, tra titolo proprio e titolo di collezione. Naturalmente queste associazioni verranno implementate tramite dei join. Per i dati bibliografici formalizzati in questo modo va benissimo un RDBMS che consentirà di rappresentare senza difficoltà tutti questi legami, ed inoltre faciliterà al massimo il controllo bibliografico tramite la creazione di tabelle distinte per gli elementi da sottoporre a controllo, come autori o soggetti, tabelle che possono essere anche aggiornate separatamente dai dati relativi ai titoli. Ma da un altro punto di vista tutti questi elementi sono anche testi che possono avere anche una notevole lunghezza, per cui è desiderabile effettuare anche ricerche testuali sofisticate all'interno di essi: ed ecco che allora risulta perfettamente adatto un information retrieval che gestisca testi strutturati (non andrebbe bene invece poter fare solo ricerche full-text su tutta la notizia bibliografica considerata come un testo indifferenziato).

Come si vede, entrambi questi punti di vista sono del tutto legittimi, per cui è impossibile fare una scelta definitiva tra RDBMS e IRS, mentre molto dipende dalla qualità e dalle prestazioni dei singoli prodotti. Resta comunque il fatto che gli RDBMS tendono ad avere prestazioni migliori nel controllo dei dati, e gli IRS invece nella ricerca, per cui quando si hanno esigenze particolarmente sofisticate si utilizzano due database: uno appunto per la produzione, gestito con un relazionale, e un altro per la consultazione, magari via Web, gestito con un information retrieval. I due archivi sono fisicamente distinti, e i dati vengono trasferiti dall'uno all'altro a scadenze predefinite, che possono anche essere giornaliere. In certi casi, l'information retrieval può anche essere in grado di interrogare direttamente alcune tabelle del RDBMS per acquisirne dati in tempo reale (ad esempio in campo bibliotecario questo potrebbe essere fatto per verificare la situazione del prestito di un certo documento reperito tramite l'information retrieval). Una architettura di questo genere viene utilizzata nell'Indice SBN, in cui usa DB2 per la produzione dei dati e Basis per la consultazione pubblica via Internet sia tramite WWW sia con client Z39.50.

Il rapporto tra basi dati bibliografiche e RDBMS ha alcuni aspetti interessanti. Abbiamo visto infatti poc'anzi che il dato bibliografico infatti ha una struttura molto complessa, rappresentabile per lo più tramite associazioni molti a molti, **e quindi richiede una struttura di database fortemente normalizzata**. Si ricorderà che nella trattazione delle forme normali abbiamo trovato esempi abbastanza naturali di schemi di relazione applicabili a dati bibliografici per tutte le forme normali, tranne forse la DKNF, dove l'esempio dato era alquanto artificioso (il che non esclude che non si possano trovare esempi migliori). In linea generale, si può dire che una progettazione rigorosa richiederà probabilmente l'uso della 4NF, e fors'anche della PJNF. Questo, aggiunto al gran numero di tabelle necessarie, soprattutto se si vogliono trattare anche i dati amministrativi, spiega perché sono destinati al fallimento i tentativi di progettare in poco tempo basi dati bibliografiche di struttura semplificata con tempi di sviluppo trascurabili. Queste basi dati potranno essere adatte per gestire un elenco di libri o una piccola bibliografica, ma non per un impiego bibliotecario di livello professionale, anche se qualcuno si illude del contrario.

D'altra parte anche le risorse degli information retrieval devono essere sfruttate a fondo per ottenere buoni risultati. Esempi interessanti si possono trarre da CDS-ISIS e da un archivio molto sofisticato come Teca. Le funzionalità che offre CDS-ISIS per il collegamento di record tramite la funzione `ref()` del linguaggio di formattazione possono venire impiegate per rappresentare associazioni uno a molti (ad esempio i titoli delle parti di un documento con titolo d'insieme) o anche molti a molti. Il linguaggio di formattazione permette operazioni anche più raffinate. Ad esempio un formato come `(v1,ref(mfn,(v2)))`[49] associa a ogni occorrenza del campo ripetibile 1 una occorrenza del campo ripetibile 2 (nello stesso record), operazione concettualmente simile a un prodotto cartesiano (se si assume come rispettato il requisito dell'atomicità dei dati per il campi 1 e 2, magari definendo un dominio proprio a questo scopo, si tratta di un prodotto cartesiano a tutti gli effetti).

Ricordiamo infine che un tale catalogo predisposto espressamente per la consultazione pubblica viene detto OPAC, cioè *Online Public Access Catalog*. Il termine OPAC si applica non solo ad un catalogo a sé stante, ma anche ad una procedura di consultazione facilitata che può far parte di un applicativo per biblioteca a fianco di quelle per la catalogazione, il prestito e l'amministrazione.

10. BASI DI DATI MULTIMEDIALI

Le basi dati multimediali sono quelle in cui sono presenti dati come immagini, suoni o filmati. In particolare, vi possono essere basi in cui i dati multimediali sono accessori: si pensi ad una base dati bibliografica che contiene anche la riproduzione dei frontespizi o di pagine di particolare interesse, o anche - se essa include notizie relative a registrazioni audio o video - alcuni secondi di musica associati ai record delle registrazioni audio o di video associati ai record delle registrazioni video. In altri casi invece gli elementi multimediali sono il contenuto principale della base dati, come ad esempio quando si cataloga una raccolta di fotografie e si riproduce ogni fotografia oltre a inserire dati come l'autore, l'epoca ecc. Quando si riproducono interi libri e poi si rendono consultabili tramite un software che permetta di sfogliare le pagine, andare ad una certa pagina determinata e simili operazioni a volte si utilizza una struttura di database per gestire i dati identificativi delle pagine, ma sono possibili anche altre tecniche, come ad esempio l'uso del popolare formato PDF[50].

Vediamo dunque in primo luogo come si acquisiscono i dati limitandoci a quelli più comunemente associati ai database, cioè alle **immagini**. Anche se ci sono immagini prodotte direttamente da computer, di solito quelle che interessano sono immagini di originali come libri a stampa o manoscritti, disegni, fotografie e simili. Per ottenere immagini da questi originali sono necessari degli appositi apparecchi. I più diffusi attualmente sono gli **scanner**, che esistono in varie forme. Se gli originali sono trasparenti, si tratta in genere di diapositive, per le quali si possono usare degli scanner appositi, di piccole dimensioni, nei quali in genere la diapositiva si inserisce tramite una fessura. Adattatori per diapositive si possono applicare alla maggior parte degli scanner piani di cui diremo ora. Per gli originali opachi, come carta o stampe fotografiche, gli scanner più diffusi sono quelli **piani**, simili nell'aspetto ad una fotocopiatrice e di costo moderato. Vi sono molti modelli che si distinguono tra loro sia per la qualità che per la velocità della riproduzione. Se si devono effettuare acquisizioni di grandi quantità di immagini, la velocità diventa un fattore essenziale, perché anche pochi secondi risparmiati per ogni acquisizione possono alla fine risultare in settimane o mesi di lavoro in meno. Con gli scanner piani non si possono riprodurre agevolmente tutti gli originali: infatti libri molto spessi o che non si aprono bene, soprattutto se preziosi e delicati, non possono essere appoggiati correttamente sul piano di ripresa. Un altro tipo di scanner è quello a **tamburo**, usato nell'editoria elettronica. Si tratta di apparecchi di costo molto elevato e di qualità altrettanto elevata, nei quali però gli originali devono essere fissati all'interno di un tamburo, per cui

[49] Putroppo questo interessante esempio è comprensibile in dettaglio solo a chi conosce il linguaggio di formattazione di CDS-ISIS, una esposizione del quale però sarebbe al di là degli scopi di questo documento.

[50] In realtà è possibile che il formato PDF internamente preveda una qualche struttura di database per mantenere i dati relativi alle molte pagine che possono comporre un documento, ma questa struttura, se esiste, non è accssibile come tale e quindi non ci interessa in questa sede

deve in pratica trattarsi di fogli singoli o al massimo di fascicoli sottilissimi. Quando c'è la necessità di riprodurre qualunque tipo di materiale bibliografico, anche delicato e non ben manipolabile, bisogna ricorrere a quegli scanner appositamente progettati nei quali l'originale viene collocato su di un piano come lo si collocherebbe su di un tavolo e la ripresa avviene tramite un obiettivo collocato al di sopra del piano, al cune decine di centimetri di distanza (in alcuni modelli c'è invece una testa di ripresa mobile che scorre al di sopra dell'originale). Questi apparecchi sono disponibili in pochi modelli e sono estremamente costosi, all'incirca da 40 a 130 milioni, oltre ad essere molto pesanti ed ingombranti. Essi si distinguono anche per la velocità di ripresa, per cui sono adatti per effettuare un gran numero di acquisizioni in tempi relativamente brevi.

Una possibile alternativa agli scanner sono le **macchine fotografiche digitali**. Le macchine professionali, che di solito sono appositi dorsi da applicare a normali macchine reflex 35 mm. o medio formato, hanno costi paragonabili agli scanner dell'ultimo tipo citato, soprattutto se è necessario un corredo di obiettivi, uno stativo e le luci appropriate, anche se hanno il vantaggio di essere molto più maneggevoli. C'è poi una vasta gamma di macchine comunemente classificate come amatoriali o semiprofessionali, paragonabili non alle reflex ma alle compatte, e quindi non dotate di ottiche intercambiabili, i cui prezzi vanno da meno di 500.00 lire a circa 3 milioni e le cui prestazioni migliorano continuamente. Si tratta di apparecchi progettati per competere con le macchine fotografiche tradizionali e non con gli scanner per impiego bibliografico, per cui fino a poco tempo fa non davano buoni risultati nella ripresa dei testi, che risultavano leggibili con molta fatica se non del tutto illeggibili, ma a partire dalla seconda metà del 1998 sono apparsi diversi modelli che forniscono prestazioni molto valide accettabili. Si tratta di modelli dotati di un CCD (il sensore di ripresa) da almeno 2 milioni di pixel, meglio se da 2,5 milioni di pixel. Dall'autunno 1999 poi hanno cominciato ad apparire modelli di reflex progettati dall'origine come digitali e di caratteristiche professionali, a prezzi molto inferiori rispetto ai dorsi digitali prima citati, dell'ordine del 13-14 milioni compresi alcuni obiettivi e i principali accessori[51]. Rispetto agli scanner questi apparecchi hanno il vantaggio di essere molto più leggeri e maneggevoli, oltre che più versatili, potendo essere usati non solo per riprendere libri o altri documenti, ma qualunque altro soggetto, per cui è pensabile che in breve tempo, se continueranno a progredire con il ritmo attuale, diventeranno il principale strumento di acquisizione immagini anche in campo bibliografico e documentario. Ovviamente per la riproduzione di documenti le fotocamere digitali (come del resto quelle analogiche) non possono essere usate a mano libera, ma vanno accoppiate ad uno stativo di riproduzione e ad una illuminazione artificiale tale da essere uniforme su tutto il campo inquadrato. In questo contesto, la fotocamera potrà essere mantenuta sempre collegata al PC ed alimentata da rete invece che da batteria. Inoltre è opportuno che, se disponibile, venga usato un software che consenta il completo controllo dell'apparecchio tramite il computer.

Vediamo ora le caratteristiche delle immagini ottenute con in vari strumenti di acquisizione, caratteristiche che vengono descritte da vari parametri. Molto importante è la **profondità di colore** , cioè il numero di colori che possono essere utilizzati in una certa immagine. Ogni immagine digitalizzata è composta da **pixel**, cioè elementi atomici, non ulteriormente scomponibili (si possono immaginare come tanti puntini luminosi associati a comporre l'immagine), il colore di ognuno dei quali viene rappresentato da un certo numero di bit. La profondità di colore non è altro che il numero di possibili combinazioni di bit, e quindi per n bit è 2^n . Molto importante è il caso in cui si ha un bit di profondità di colore, e quindi solo 2 colori rappresentabili, in pratica il bianco e il nero senza sfumature intermedie. Si tratta di una soluzione particolarmente adatta per rappresentare il testo, perché riproduce un testo perfettamente nero su di uno sfondo perfettamente bianco. Essa inoltre consente di generare, a parità di condizioni, dei files molto piccoli, bastando un solo bit per ogni pixel. Per contro la riproduzione in due colori può dar luogo a problemi quando il contrasto tra testo e sfondo è troppo basso (ad esempio il testo è sbiadito o lo sfondo annerito), perché - non avendo sfumature intermedie tra il bianco e il nero - rischia di riprodurre tutto come bianco o tutto come nero. Con l'acquisizione in due colori si può riprodurre il testo ma si perdono tutte le informazioni sull'aspetto reale della pagina, ad esempio il colore della carta, la presenza di macchie e simili, per cui essa non è adatta quando si vuole riprodurre non solo il testo ma l'aspetto fisico del documento. Ancor meno adatta è, come ovvio, per la riproduzione di

[51] Il primo modello di questa categoria si può considerare la Nikon D1, apparsa nel settembre 1999, seguita nel gennaio 2000 dalla Minolta Dimage RD-3000 di caratteristicheche più o meno simili, che secondo ogni verosimiglianza sarà a sua volta seguita da un gran numero di modelli di caratteristiche analoghe o superiori.

originali a colore, come disegni e miniature. Se non interessa il colore, può risultare molto indicata in questi casi l'acquisizione in scala di grigi, usualmente a 4 od 8 bit per pixel, che danno rispettivamente 16 e 256 toni di grigio mantenendo l'immagine di dimensioni ancora abbastanza ridotte. Questa soluzione è adatta per risolvere i problemi dovuti al basso contrasto tra testo e sfondo e quando vi sono disegni in bianco e nero con sfumature. Se si vuole l'acquisizione a colori, si può partire dai 4 bit (16 colori) supportati dalle vecchie schede VGA, che sono sufficienti solo per immagini semplicissime per arrivare alle immagini a 24 bit per pixel, che riproducono fino a 16.777.216 colori, ma sono evidentemente molto grandi (non considerando la compressione, 24 volte più grandi di analoghe immagini ad 1 bit). In pratica, per la riproduzione di materiale bibliografico è quasi sempre sufficiente l'acquisizione in bianco e nero, perché quella a colori è necessaria solo quando vi sono disegni o altri elementi a colori (ad esempio nei libri e manoscritti miniati) o quando si vuole riprodurre nei minimi particolari l'aspetto reale del documento e non solo rendere leggibile il testo[52].

Un altro parametro importante è la **risoluzione**, che è un indicazione sulle dimensioni minime dei particolari che l'apparecchiatura di acquisizione è in grado di discriminare (in pratica, due punti che nell'originale sono più vicini della massima risoluzione dell'apparecchio verranno riprodotti come un solo punto). La risoluzione si misura in punti per pollice (dot per inch = dpi). Nella maggior parte dei casi, per il materiale bibliografico è sufficiente una risoluzione di 300 x 300 dpi (i due numeri indicano la risoluzione verticale e quella orizzontale), ma una risoluzione di 600 x 600 dpi è molto consigliabile perché permette di riprodurre meglio i particolari molto piccoli. Per molti scanner vengono dichiarati due valori di risoluzione, quella ottica e quella interpolata. La risoluzione ottica è quella reale che si ottiene in fase di acquisizone dell'immagine, mentre quella interpolata, che è sempre superiore alla prima, è una risoluzione emulata effettuando calcoli sull'immagine, e ha come prezzo una certa perdita di qualità. In pratica, quella che interessa è la risoluzione ottica, e a questa si riferiscono i valori riportati sopra.

Una volta acquisita, l'immagine deve essere salvata in un file. A questo scopo sono disponibili molti formati grafici. Gran parte di questi non sono formati proprietari, ma sono utilizzabili con un gran numero di software, e alcuni anzi si può dire con tutti i software grafici (i formati proprietari vanno assolutamente evitati perché vincolano all'uso di specifici software, cosa che non accade con gli altri). La maggior parte dei formati grafici prevede almeno un certo grado di compressione sull'immagine al fine di diminuirne le dimensioni, che altrimenti potrebbero diventare preoccupanti (esistono però anche formati senza compressione). Si distinguono due tipi di compressione: la compressione **lossy**, che prevede una certa perdita di qualità dell'immagine rispetto a quella originale non compressa, perdita che dipende da formato e dal livello di compressione desiderato, e che può essere anche modestissima, e la compressione **lossless**, cioè senza perdita di qualità. Come si può facilmente immaginare, i formati lossy ottengono generalmente una compressione superiore. Il più diffuso formato con compressione lossy è il JPEG, mentre tra i formati lossless ricordiamo il TIFF (che esiste in numerose varianti, compresa una senza compressione) e il PNG, recentemente introdotto. La maggior parte dei programmi di elaborazione immagini sono in grado di gestire un gran numero di formati, e quindi anche di convertire un'immagine da un formato all'altro, ma evidentemente una volta che una immagine è stata salvata in un formato lossy non è più possibile eliminare la perdita di qualità anche se la si converte in un formato lossless. Sulle immagini è possibile effettuare un gran numero di elaborazioni attraverso appositi programmi, come Photoshop, Photopaint o Paint Shop Pro. Alcune di queste elaborazioni hanno lo scopo di ottenere degli effetti artistici, mentre altre servono a migliorare la leggibilità del contenuto dell'immagine. Un tipico esempio di questo secondo tipo di elaborazioni è l'aumento del contrasto che risulta utile, tra l'altro, quando il testo spicca troppo poco rispetto allo sfondo.

Veniamo infine ad esaminare i modi in cui immagini, suoni e video possono essere integrati nei database. Questi elementi, a seconda del motore relazione o di information retrieval che si usa (non ci sono, sotto questo aspetto, particolari differenze) possono essere conservati a parte come files nel loro formato originario (ad esempio TIFF, JPEG ecc.), per cui il database contiene dei riferimenti a questi files, oppure possono venire a

[52] Qualche volta il testo risulta meno leggibile nell'acquisizione a colori a 24 bit che in quella in bianco e nero a 1 bit perché la leggibilità può risulta disturbata da fattori come il colore del supporto scrittorio o la presenza di macchie

far parte direttamente dei dati che si trovano all'interno del database. In particolare, a questo scopo viene impiegato un tipo di dati denominato BLOB, cioè *binary large objects* (grandi oggetti binari).

Più interessante è però probabilmente esaminare il modo in cui questi dati vengono gestiti dal motore di database. Esistono diverse tecniche, ed in particolare le seguenti:

- **uso di un programma esterno**; in questo caso il campo del database contiene un riferimento al file, e quando, normalmente nell'ambito di qualche operazione che richiede l'output di dati come la visualizzazione dei risultati di una query, viene richiesta la visualizzazione o riproduzione del file viene attivato un programma esterno adatto allo scopo (spesso scelto dall'utente); questa soluzione ha un limite notevole, e cioè la poca integrazione tra il database e gli elementi multimediali; essendo questi gestiti da un programma esterno, può accadere che non sia possibile controllarli come sarebbe desiderabile, anche se molto dipende dal motore di database che si usa; inoltre la loro gestione può non essere prevedibile nei dettagli in sede di progettazione del database
- **funzioni di visualizzazione incorporate nel motore di database**; diversi moderni motori di database permettono di visualizzare direttamente almeno alcuni formati grafici; tali funzioni di visualizzazione possono però essere piuttosto elementari, e quindi insufficienti se si desiderano operazioni molto sofisticate; probabilmente questa soluzione è in generale adatta soprattutto quando gli elementi grafici o multimediali non costituiscono il contenuto principale del database
- **uso di un programma che utilizzi sia le funzionalità del motore di database sia routine specifiche di elaborazione di elementi multimediali**; questa descrizione può sembrare un po' oscura, ma diventerà comprensibili dopo la lettura del capitolo sugli strumenti di sviluppo; si tratta di scrivere un programma che utilizzi da una parte le librerie del motore di database, dall'altra librerie specifiche di elaborazione immagini (o suoni, video ecc.), in modo da poter ottenere i migliori risultati sotto entrambi gli aspetti; questa tecnica è però la più difficile da mettere in opera, e spesso non alla portata dell'utente non professionista

11. BASI DI DATI ORIENTATE AGLI OGGETTI

La nozione di oggetto è stata introdotta allo scopo di semplificare e rendere più efficiente la progettazione del software, ed ha avuto notevole successo: i principali linguaggi di programmazione attualmente usati, a cominciare dal C++ per andare al Delphi o al Visual Basic sono infatti in misura più o meno estesa orientati agli oggetti. La nozione di oggetto non viene utilizzata solo nell'ambito dei linguaggi di programmazione di impiego generale, ma anche nell'ambito delle architetture per la gestione delle basi di dati: si parla infatti di **sistemi di gestione di basi di dati orientati agli oggetti** (OODBMS = Object Oriented DBMS).

Un oggetto è una entità astratta che ha lo scopo di modellizzare un qualche aspetto del mondo reale che in certo contesto è importante trattare. Un oggetto è caratterizzato da:

- **OID** (Object Identifier), cioè un identificatore univoco - ad esempio un numero - che permette all'oggetto di avere una esistenza permanente
- **proprietà** o **attributi**, che sono aspetti dell'oggetto che possono assumere dei valori tratti da un certo dominio; ad esempio un oggetto *computer* potrebbe avere l'attributo *CPU* che può assumere i valori *Pentium II*, *Pentium III*, *Athlon*, *PowerPC* ecc.[53]
- **metodi**, che sono procedure associate all'oggetto che ne costituiscono parte integrante; per esempio, l'oggetto *computer* dell'esempio precedente potrebbe avere il metodo *calcola_prezzo* che fa la somma dei prezzi dei componenti per calcolare il prezzo del computer completo.

Per essere più precisi un oggetto come *computer* nell'esempio precedente non sarebbe normalmente trattato direttamente come un oggetto, ma come una **classe**, cioè come uno schema per la costruzione di oggetti

[53] Naturalmente l'oggetto non ha qualsiasi proprietà dell'oggetto reale che si intende modellizzare: si tratta di una entità astratta nella quale si prendono in considerazione gli aspetti che si ritengono utili per un determinato scopo.

aventi la stessa struttura. In base alla classe computer, il programmatore può creare gli specifici oggetti *computer* (detti instanziazioni della classe) di cui di volta in volta ha bisogno[54].

Si noti anche che gli oggetti possono avere una struttura ad albero. Ad esempio, nel caso del nostro *computer*, è un po' troppo semplice pensare che basti prevedere un attributo *CPU* che possa avere un valore semplice che identifica il tipo di CPU. Infatti una CPU possiede numerose proprietà che è utile poter trattare esplicitamente una per una, e quindi l'attributo CPU potrebbe essere composto da altri attributi, come *frequenza_di_clock*, *tensione_di_alimentazione*, *frequenza_del_bus*, *cache*, *fabbricante*, *modello*, *prezzo* ecc., ognuno dei quali a sua volta può assumere uno specifico valore.

Gli oggetti hanno tre importanti caratteristiche che sono le seguenti:

- **incapsulazione**: metodi e attributi sono incapsulati nell'oggetto, cioè costituiscono sue parti integranti e non sono accessibili separatamente dall'oggetto, per cui per attivare un metodo si una sintassi che fa innanzitutto riferimento all'oggetto, ad esempio *oggetto.metodo*; in questo modo che utilizza l'oggetto non deve preoccuparsi della sua struttura interna, ma solo della sua interfaccia, cioè del modo in cui far riferimento a metodi e attributi; se un metodo viene riscritto, purché non ne venga modificata l'interfaccia, questo interessa solo a chi progetta l'oggetto e non a chi lo utilizza, che può continuare ad utilizzarlo esattamente come prima
- **ereditarietà**: si possono definire oggetti sulla base di altri oggetti; gli oggetti così definiti ereditano proprietà e metodi degli oggetti su cui sono basati, insieme a proprietà e metodi loro propri. In questo modo qualsiasi oggetto diventa una base che si può estendere in modo indefinito
- **polimorfismo**: quando si definisce un oggetto basato su un altro oggetto è possibile ridefinire o modificare i suoi metodi, in modo tale che l'interfaccia rimanga invariata ma cambi il comportamento del metodo, per cui chi utilizza l'oggetto può fare riferimento al metodo sempre nello stesso modo, ma ottenendo un comportamento di volta in volta appropriato al contesto in cui si opera (questo, naturalmente, se l'oggetto è stato progettato bene).

Per quanto riguarda l'applicazione della tecnologia ad oggetti alle basi di dati, l'idea base è di rappresentare come oggetti sia i dati sia tutti gli altri componenti di un sistema di gestione di basi di dati. In realtà, gli OODBMS elaborati, spesso a livello sperimentale, alcuni anni fa ed in particolare negli anni '80, come GemStone, Iris, ORION, Vbase e altri non hanno avuto particolare successo, ma la tecnologia ad oggetti è stata incorporata, in misura più o meno estesa in numerosi prodotti commerciali, sia "popolari" Access, sia professionali come Oracle e PostgreSQL. Di solito non si tratta di OODBMS puri, ma di RDBMS implementati tramite il ricorso agli oggetti: ad esempio, invece di utilizzare gli oggetti per rappresentare i dati in sostituzione delle tabelle, si può concepire una tabella come un oggetto con i suoi metodi e le sue proprietà; anche il risultato di una query si può concepire come un oggetto, con la query SQL che fa parte della definizione dell'oggetto, e per di più query e tabelle possono essere viste come oggetti della stessa classe, in modo da mettere in evidenza le affinità tra di loro (è quanto succede in Access con gli oggetti *recordset*). Anche gli elementi dell'interfaccia, come form, caselle combinate, checkbox ecc. possono essere rappresentanti come oggetti. Si noti, peraltro, che in alcuni sistemi è possibile solo fare riferimento alle classi predefinite creandone instanziazioni, mentre nei sistemi più completi e sofisticati è possibile creare nuove classi e metodi, ottenendo così una libertà d'azione di gran lunga maggiore.

12. DATA WAREHOUSE

L'espressione *data warehouse* significa letteralmente *supermercato dei dati*, e si riferisce ad una tipologia di software che ha avuto un certo successo soprattutto in ambito aziendale. Si tratta, come si vedrà, di software apparentato con quello di gestione di database, tanto che spesso viene prodotto dagli stessi fornitori.

[54] Spesso gli oggetti vengono creati dichiarando una variabile di tipo <*classe*> (nel nostro caso sarebbe di tipo *computer*) oppure con una istruzione del tipo: *x:=new(computer)*.

L'idea che sta alla base del data warehouse è quella di recuperare i dati contenuti in un database (di solito relazionale), ma in una forma che agevoli non le normali operazioni di inserimento e ricerca dei dati, ma altre operazioni, come analisi statistiche e storiche. A questo scopo viene realizzato un apposito archivio, diverso da quello del database di origine dei dati, e nel quale questi ultimi vengono trasferiti quando l'utente lo desidera. Questo archivio è dotato di una sua propria struttura, che include gli elementi sui quali si intende compiere l'analisi. Appunto su tali elementi si eserciteranno poi i tipi di analisi resi possibili dal software di data warehouse. Spesso si tratta di analisi che non sarebbero in assoluto impossibili per mezzo del RDBMS di origine, ma sarebbero più complicate e anche meno raffinate.

Le operazioni tipiche del data warehouse vengono raggruppate sotto il nome di **On Line Analytical Processing (OLAP)**, per indicare un tipo di elaborazione che si distingue dall'**OLTP (On Line Transactional Processing)** tipica degli RDBMS. L'OLTP è caratterizzata dalla presenza da un gran numero di operazioni di piccola entità, come la scrittura di una tupla, mentre l'OLAP è caratterizzata dall'esistenza di operazioni che possono essere in minor numero, ma che coinvolgono grandi quantità di dati.

Il data warehouse non è stato finora molto utilizzato nell'ambito delle biblioteche, ma possiamo tentare di delinearne un possibile impiego. Supponiamo di avere il solito database che comprende: dati bibliografici, dati amministrativi (archivio dei fornitori, ordini, pagamenti, proposte di acquisto), dati su lettori e prestiti. Attraverso un software di data warehouse si potrebbero fare, ad esempio, analisi di questo genere:

- serie storiche di vari parametri, come prestiti, acquisti, catalogazioni, ricerche bibliografiche
- statistiche
- correlazioni tra dati che di per sé sarebbe separati, ad esempio per mettere in evidenza le eventuali relazioni tra stanziamenti per la biblioteca e numero degli iscritti al prestito, oppure tra acquisizioni nell'ambito di una certa materia e letture

Il data warehouse può apparire, ed essere, una cosa molto attraente. Tuttavia prima di implementare un sistema di questo genere bisogna valutarne attentamente il rapporto costi-benefici, tenendo conto che allo scopo sono necessari software anche costosi, hardware adeguato e risorse umane.

È interessante notare peraltro che anche gli OPAC bibliografici potrebbero essere classificati nell'area dell'OLAP, perché sono orientati ad operazioni di estrazione di dati più raffinate e complesse di quelle che tipicamente vengono effettuate nell'ambito dell'OLTP.

13. STRUMENTI DI SVILUPPO

In un certo senso, tutti i programmi di gestione database, siano essi RDBMS o IRS mettono a disposizione degli strumenti di sviluppo, nel senso di creazione di qualcosa che prima non c'era. Infatti, qualunque programma di questo genere dovrà come minimo permette di creare dei nuovi database, definendone la struttura, ma in pratica quasi tutti permettono di fare molto di più, ed in particolare definire query, form e report più o meno sofisticati. Si tratta di operazioni che possono essere tutt'altro che banali, perché i programmi più evoluti hanno possibilità che per essere sfruttate a fondo richiedono una buona comprensione dei principi di funzionamento del programma. Si tratta inoltre di operazioni molto importanti; anzi, la definizione della struttura del database e, per gli RDBMS, delle query sono assolutamente fondamentali. Tutte queste cose però si possono considerare rientranti nell'ambito delle attività ordinarie che si fanno con un programma di gestione di database, mentre per sviluppo intendiamo qui ciò che va ancora oltre, ad esempio: creazione di funzionalità evolute e personalizzate di controllo sull'immissione dei dati e in generale di tutto quello che riguarda l'interazione con l'utente, elaborazione di programmi per implementare funzionalità non disponibili nell'ambito del programma di base, come scrittura o lettura di particolari formati di scambio di dati, di interfacce di ricerca, di programmi che utilizzano un certo motore RDBMS o IRS ma nascondendo completamente l'aspetto del programma standard.

Per realizzare queste cose esistono varie tecniche, ed in particolare quelle elencate qui di seguito (a seconda del programma che si usa se ne può trovare disponibile una o anche più):

- **niente**; alcuni programmi non prevedono alcuno strumento di sviluppo; si tratta quasi sempre di programmi molto semplici, utilizzabili solo per impieghi senza eccessive pretese, anche se a volte sono invece programmi che prevedono ugualmente funzionalità sofisticate ed esempio per la creazione di query o di report
- **macro;** con il termine *macro* si intende in generale una successione di comandi registrati in modo da poter essere eseguiti automaticamente uno di seguito all'altro[55]; esistono le *macro di tastiera*, che sono la registrazione di una serie di tasti premuti, che permettono di automatizzare le operazioni che si effettuano premendo questi tasti, ma non sono un vero e proprio strumento di sviluppo, perché con queste macro non si fa nulla di più di quello che si farebbe premendo manualmente i tasti registrati; in altri casi le macro permettono sviluppi maggiori, e talvolta sono anche di uso non del tutto semplice; esempi di RDBMS che dispongono di un evoluto linguaggio di macro sono Access e Approach
- **linguaggio di programmazione interno**; molti programmi mettono a disposizione un vero e proprio linguaggio di programmazione specializzato, i programmi creati col quale vengono eseguiti solo **all'interno** del programma principale (il RDBMS o IRS); è il caso, ad esempio di Access, Isis, Paradox, tutti i dBase; questi linguaggi sono particolarmente ricchi di comandi per la gestione del database, per cui, ad esempio basta un solo comando per aprire una tabella e un altro per visualizzare i dati, laddove con un linguaggio di impiego generale bisognerebbe scrivere complesse procedure per effettuare le stesse operazioni; per contro, di solito sono più limitati per quanto riguarda le operazioni di input e output a basso livello; questi linguaggi sono normalmente piuttosto semplici da utilizzare, e quindi alla portata non solo dei professionisti, ma anche degli utenti evoluti: anzi, pur non essendo stati concepiti a scopo didattico sono molto indicati per iniziare a programmare, sia appunto perché abbastanza semplici, sia perché permettono di realizzare presto programmi non del tutto banali, il che riesce di incoraggiamento all'aspirante programmatore[56]; è interessante notare che nella maggior parte dei programmi moderni dotati di interfaccia grafica (come il solito Access) i programmi vengono attivati in risposta ad azioni effettuate sugli elementi dell'interfaccia, ad esempio il click su un bottone, l'imissione o la modifica di un dato, la selezione di un form, per cui invece di un unico grosso programma si scrive un insieme di procedure che vengono attivate a seconda delle operazioni dell'utente
- **linguaggio di programmazione specializzato**; vi sono dei linguaggi di programmazione adatti a creare dei programmi autonomi (stand-alone) che incorporano le funzioni del motore di database; questi linguaggi, come Clipper e Progress, mettono a disposizione istruzioni apposite per la gestione dei database, simili a quelle dei linguaggi di cui al punto precedente, ma di solito sono più ricchi di istruzioni di impiego generale; in questi casi il motore di database non è un singolo programma autonomo, ma un insieme di procedure che vengono di volta in volta incorporate nei vari programmi; si tratta di linguaggi più o meno simili a quelli di cui al punto precedenti, ma spesso più complessi e flessibili e perciò più difficili da usare; questi linguaggi, permettendo la realizzazione di programma stand-alone, facilitano la distribuzione delle applicazioni
- **linguaggio di programmazione di impiego generale + librerie specializzate**; i linguaggio di programmazione di impiego generale sono quelli che non sono stati progettati con riguardo ad uno specifico tipo di programma; esempi di questi linguaggi sono C, Pascal, Delphi (derivato dal Pascal), Basic, Cobol, Power Builder e altri; le librerie sono invece raccolte di procedure che possono essere richiamate da un programma scritto in uno di questi linguaggi[57]; vi sono librerie contenenti procedure adatte a molteplici

[55] Nella terminologia di alcuni software di database però il termine indica un piccolo programma scritto in un vero e proprio linguaggio di programmazione, e che quindi non rientra nelle macro come sono definite qui

[56] Come avviamento alla programmazione è particolarmente adatto l'Isis-Pascal perché conserva la chiarezza e il rigore del Pascal standard, e quindi abitua ad un buono stile di programmazione

[57] Alcuni motori RDBMS per grossi sistemi, come DB2 della IBM prevedono una soluzione un po' diversa: il programma viene scritto in linguaggio C o in altri linguaggi inserendovi delle speciali istruzioni per l'accesso al database; successivamente viene sottoposto al *precompilatore*, fornito con il motore relazionale, che automaticamente traduce queste speciali istruzioni in procedure standard del linguaggio (molto più complicate); si ha così un programma scritto completamente con il linguaggio standard che può quindi venire elaborato da un normale compilatore

scopi, tra cui anche la gestione dei database; in particolare, vi possono essere motori di database disponibili sotto forma di librerie, come il motore relazionale Microsoft Jet, che è lo stesso utilizzato da Access, oppure il motore di information retrieval di Isis per Windows e per Unix[58] (in altri casi, le librerie potrebbero contenere slo le procedure lato client necessarie per il colloquio con un server di database); con questa soluzione, si ha a disposizione tutta la potenza di un linguaggio di impiego generale e contemporaneamente un insieme di istruzioni specializzate per la gestione dei database: si tratta quindi di una tecnica molto valida, che è quella normalmente usata per realizzare applicazioni di alta qualità, tuttavia il suo uso non è sempre alla portata di tutti perché richiede una buona conoscenza di un linguaggio di programmazione, conoscenza alla quale peraltro possono arrivare non solo programmatori professionisti ma entro certi limiti anche utenti evoluti; tra i linguaggi citati, Pascal e Delphi di distinguono per chiarezza ed eleganza, mentre il C è il più potente ma anche più difficile degli altri

- **linguaggio di programmazione di impiego generale + librerie di base**; per *librerie di base* intendiamo qui librerie che non corrispondono ad un completo motore di database, ma implementano solo alcune funzionalità specifiche, ad esempio la gestione degli indici, per cui il motore di database non è già pronto, ma va costruito a partire però non da zero ma da costituenti più elementari; si tratta di una tecnica molto più complessa di quelle descritte finora, e che più difficilmente può essere alla portata dei non professionisti, ma può anche garantire risultati di alta qualità; questi risultati, peraltro, non sono certo garantiti, ma dipendono dalla bravura del programmatore e dalla qualità delle librerie utilizzate; possono essere utilizzati più o meno tutti i linguaggi citati nel punto precedente, con una certa preferenza per il C
- **linguaggio di programmazione di impiego generale (da solo)**; utilizzare solo un linguaggio di impiego generare significa scrivere da zero il motore relazionale o di information retrieval; scrivere un motore relazionale o IRS non specifico per una certa applicazione, come Oracle o DB2 è una impresa di enorme complessità, che richiede una preparazione di livello professionistico, ed è quindi una cosa che viene fatta raramente, anche perché esistono già ottimi e collaudati prodotti di questo genere; più frequentemente invece vengono realizzati ex novo piccoli motori progettati appositamente per specifici database, sui quali magari è previsto solo un numero limitato di operazioni: anche questa, comunque, è una cosa tutt'altro che banale; per attività di questo genere è quasi obbligatorio l'impiego del linguaggio C

Concludiamo ricordando che numerosi RDBMS o IRS comprendono il cosiddetto **modulo runtime**: si tratta di una versione del programma che permette di utilizzare un archivio già esistente con tutti i suoi componenti, come query, form, report ed eventuali procedure ad essi collegate, ma non modificarlo o di crearne di nuovi, e viene utilizzata allo scopo di distribuire archivi già pronti.

14. HARDWARE

Molti forse non sanno che esiste dell'hardware specializzato per la gestione dei database: si tratta delle cosiddette **database machine**, che sono computer specializzati, progettati ed ottimizzati appunto per gestire database. Tra le caratteristiche principali che hanno di solito le database machine vi sono il **parallelismo** e l'uso delle **memorie associative**. Il parallelismo significa che le operazioni di calcolo vengono distribuite, dal sistema operativo, tra molti processori che operano contemporaneamente, in modo da ottenere prestazioni complessive superiori. Le memorie associative sono memorie il cui contenuto è costituito da una serie di coppie <*chiave, valore*>, in cui il valore è un insieme di dati, mentre la chiave è un identificatore univoco analogo ad una chiave di indice. In fase di ricerca il dato da ricercare viene confrontato con le chiavi: se si trova una corrispondenza, viene restituito il valore associato alla chiave. Fin qui non ci sarebbe niente di speciale, ma la particolarità delle memorie associative è che il dato in ingresso viene confrontato **contemporaneamente** con tutte le chiavi, tramite un hardware speciale. In questo modo le prestazioni sono di gran lunga superiori a quelle che si ottengono con gli indici convenzionali. In ogni caso, parallelismo e memorie associative richiedono dell'hardware molto complesso e costoso, per cui le database machine sono

[58] Si noti che questa tecnica, come descritta qui, ricomprende sia l'embedded SQL che la SQL/CLI, descritti nel capitolo su SQL; le librerie di Isis, pur non implementando SQL ma un motore di information retrieval, adottano sostanzialmente la tecnica CLI

usate raramente, e solo quando vi è necessità di prestazioni elevatissime con grossi database e numerosi utenti (anche in questi casi, comunque, si usa più comunemente l'hardware convenzionale).

Qui però non vogliamo parlare delle database machine, ma dei normali computer che si usano comunemente nelle biblioteche e in quasi tutti gli altri contesti. Ci si potrà chiedere quale hardware sia più adatto per applicazioni di database. La risposta dettagliata non si può dare qui perché dipende da molti fattori, come la quantità di dati da gestire, il numero di utenti, il tipo di operazioni da effettuare sui dati, il grado di affidabilità richiesto. Si possono però illustrare alcuni principi generali e fare alcuni esempi.

Osserviamo innanzitutto che i programmi possono venire distinti in programmi **CPU bound** (limitati dalla CPU) e programmi **I/O bound** (limitati dall'I/O). I primi sono programmi che richiedono un gran numero di calcoli nella CPU, per cui sono condizionati soprattutto dalle prestazioni di quest'ultima, mentre i secondo sono programmi che devono compiere un gran numero di operazioni di I/O, quindi accessi alla RAM, alle memorie di massa, soprattutto l'hard disk, ed eventualmente alle risorse di rete, per cui sono condizionati soprattutto dalle prestazioni dei vari sottosistemi di I/O, quindi RAM, dischi e rete. Tra i programmi del primo tipo ci sono quelli di grafica, CAD, elaborazione immagini, elaborazione suono e video, calcolo tecnico e scientifico, giochi, tra i secondi ci sono appunto i programmi di gestione di database, che devono continuamente leggere e scrivere dalle e sulle varie memorie. Quindi dovendo stabilire la configurazione di un computer destinato a far girare questi programmi non si dovrà scegliere la CPU più potente e appena uscita, che è per lo più molto costosa, e poi risparmiare sugli altri componenti, ma fare esattamente il contrario, quindi scegliere una CPU di livello medio, e prevedere abbondante RAM e un sottosistema dischi molto veloce. Queste indicazioni si devono intendere in senso relativo: se il computer deve servire molti utenti gestendo grandissimi database ci vorrà anche una o più CPU di grande potenza, ma bisognerà sempre investire in misura ancora maggiore in RAM e dischi.

Cerchiamo di chiarire questi concetti ipotizzando alcuni scenari. Gli esempi fatti sono puramente indicativi, anche perché non tengono conto delle caratteristiche di specifici programmi, e non coprono certo tutte le situazioni possibili (si noti, tra l'altro, che vi possono essere innumerevoli situazioni intermedie tra gli scenari descritti nel seguito). Inoltre, poiché fanno riferimento a determinate configurazioni hardware, saranno certamente la parte di questo manuale che diventerà superata in più breve tempo (sono comunque previsti aggiornamenti).

Scenario 1. ***Piccola biblioteca con al massimo 4-5 posti di lavoro in rete locale, modesto traffico e non più di 15.000 registrazioni bibliografiche. I dati risiedono su un file server mentre il software di catalogazione, prestito e ricerca risiede su stazioni di lavoro e gira sotto DOS, Windows o Linux***
Considerate le prestazioni dei computer odierni, non è neppure indispensabile un vero e proprio server, ma è sufficiente un normale computer di buon livello con CPU Pentium III o Athlon 1 GHz, 384 o 512 Mb di RAM e HD UltraDMA 66 da 40 Gb. Come sistema operativo è consigliabile almeno Windows 2000 in versione workstation o, meglio ancora, Linux. La configurazione dei posti di lavoro dipende dal software che si usa, ma in molti casi saranno sufficienti anche dei PC preesistenti con CPU Pentium e 32 o 64 Mb di RAM (se il software è un vecchio programma Windows possono bastare anche 16 Mb, o se si tratta di programmi DOS anche 8).

Scenario 2. ***Biblioteca media o medio-piccola con al massimo 10-15 posti di lavoro in rete locale, fino a circa 30-40.000 registrazioni bibliografiche. I dati risiedono su un file server mentre il software di catalogazione, prestito e ricerca risiede su stazioni di lavoro e gira sotto DOS, Windows o Linux***
In questo caso sarà utile investire un po' di più sul file server, acquistando un vero server con controller Ultra SCSI 160 e sistema operativo Windows 2000 server o il solito Linux, mentre per il resto è sufficiente quanto indicato per lo scenario 1.

Scenario 3. ***Come lo scenario 2, ma con architettura client-server o time sharing.***

Dovrebbe essere sufficiente anche la configurazione indicata al punto precedente, ma poiché la RAM costa poco sarà bene abbondare un po' di più ed installarne senz'altro almeno 512 Mb.

Scenario 4. ***Biblioteca grande o medio-grande con al massimo 40-50 posti di lavoro in rete locale e fino a 80-100.000 registrazioni bibliografiche. Architettura client-server o time-sharing.***
Si può ritenere appropriato un server con processore Pentium 4 o Athlon 1,3 GHz, 768 Mb di RAM, controller Ultra SCSI 160 o IEEE-1394, 50-80 Gb di disco. Il sistema operativo dipende dall'applicazione prescelta, ma sarà più probabilmente della famiglia Unix.

Scenario 5. ***Sistema bibliotecario con biblioteca centrale come nello scenario 4 ed in più biblioteche decentrate collegate in rete geografica ed accessi per sola consultazione anche da strutture non bibliotecarie (es. uffici pubblici, istituzioni culturali), per un totale di 80-100 posti di lavoro e fino a 250.000-300.000 notizie bibliografiche. Architettura client-server o time-sharing***
Si può ipotizzare un server con due processori come nello scenario 4, 768 Mb-1 Gb di RAM e 60-100 Gb di disco. Per il sistema operativo vale quanto detto per lo scenario 4. Si può inoltre prendere in considerazione anche un server con architettura non-Intel, come gli IBM RISC, Sun Sparc ecc. (i cosiddetti *mini*, detti anche *di fascia intermedia*, nel senso di intermedia tra i personal e i mainframe), con uno o due processori, mentre per il resto la configurazione può essere uguale a quella indicata sopra (i costi delle macchine di questa categoria sono in generale superiori a quella dei computer Intel-based, anche se l'effettiva differenza dipende molto dalla configurazione che si sceglie)

Scenario 6. ***Grandissimo sistema bibliotecario che raggruppa numerose biblioteche decentrate collegate in rete geografica, tra cui alcune grandi e medio-grandi, e prevede accessi per sola consultazione anche da strutture non bibliotecarie (es. uffici pubblici, istituzioni culturali), per un totale di almeno 300 posti di lavoro e milioni di notizie bibliografiche. Architettura client-server o time-sharing***
Per uno scenario di questo genere si può ormai considerare non necessario l'impiego di un mainframe, essendo oggi sufficiente ricorrere ad una macchina di fascia intermedia con 2-4 CPU, 2-4 Gb di RAM ed un sottosistema disco altamente sofisticato dotato di abbondante cache e con 100-150 Gb di spazio complessivo. Per il sistema operativo vale sempre quanto detto per lo scenario 4.

15. ESEMPI REALI

In questo capitolo verranno illustrati alcuni esempi reali di applicazioni delle basi di dati ai servizi bibliotecari. L'esposizione sarà incentrata sull'aspetto tecnico e, dove occorre, sulle sue interazioni coi fattori organizzativi, mentre non verranno trattati gli aspetti normativi, amministrativi ed istituzionali.

15.1 SBN

SBN è la sigla di *Servizio Bibliotecario Nazionale*. Si tratta del maggior progetto di cooperazione tra biblioteche gestito con mezzi informatici esistenti in Italia. SBN fu ideato e progettato nella prima metà degli anni '80, mentre l'attività effettiva iniziò nel 1986.

Scopo di SBN è di permettere la cooperazione tra biblioteche attraverso la condivisione dei dati, ed in particolare di quelli catalografici, affinché ogni dato sia immesso una sola volta e poi non più duplicato, ma riutilizzato da chi ne ha bisogno. SBN prevede la partecipazione di biblioteche di ogni tipo, dalle nazionali alle civiche a quelle delle università a quelle di enti privati, ed è un sistema ugualitario perché ogni biblioteca usufruisce degli stessi servizi.

Premessa di SBN fu un lavoro di analisi della notizia bibliografica, che è alla base di tutte le diverse implementazioni software. Questa analisi mise in luce in modo esplicito le diverse entità a cui fanno riferimento, talvolta in modo meno esplicito, le norme catalografiche, e le loro correlazioni. Si tratta di un'analisi assai lucida e accurata che di per sé costituisce un eccellente risultato, e non fa riferimento a particolari software[59], anche se si presta particolarmente bene alle implementazioni basate su RDBMS, essendo in buona misura basata sul modello entità/associazioni, pur senza dichiararlo espressamente [60].

Quando si progettò l'infrastruttura informatica di SBN si fecero diverse scelte, non tutte ovvie e alcune, anzi, opinabili, che vanno individuate in dettaglio perché determinano tuttora l'architettura del sistema, e che sono principalmente le seguenti:

- **tutto il lavoro viene svolto online**; questo comporta una fortissima dipendenza dalle prestazioni e dall'affidabilità delle reti (peraltro oggi di gran lunga migliori rispetto agli esordi di SBN), ma è necessario per realizzare la condivisione dei dati: il lavoro online permette di gestire una sola copia fisica dei dati condivisi (per esempio, un titolo viene inserito una volta sola, mentre vengono moltiplicate solo le localizzazioni)
- **i server di database sono mainframe**; all'epoca dell'avvio di SBN era l'unica scelta possibile, anche se comportava costi molto alti, perché gli altri computer non avevano prestazioni sufficienti, ma vedremo che nel corso dell'evoluzione di SBN questo sta cambiando
- **il sistema è distribuito e non centralizzato**; gli utenti non lavorano su un unico database nazionale, ma su una serie di **poli**, ossia di database locali che normalmente servono un gruppo di biblioteche, anche se esistono poli monobiblioteca; per ogni polo vi sono quindi molti posti di lavoro che sono sempre terminali o PC in emulazione di terminale; i poli sono collegati su scala nazionale tramite l'**indice**, che è un database centrale sul quale i dati dei poli sono automaticamente replicati (tranne alcuni di interesse esclusivamente locale); pertanto ogni polo conosce i dati degli altri poli non direttamente, ma tramite l'indice; sarebbe stato tecnicamente possibile impiantare invece un unico database nazionale, che avrebbe semplificato di molto la realizzazione e la gestione del sistema; all'autore di questo testo non è del tutto chiaro perché sia stata scelta la soluzione decentrata, ma probabilmente la scelta fu il risultato di diversi fattori, come: non disponibilità delle risorse necessarie a livello centrale; politica volta a privilegiare la gestione decentrata, ed in particolare a valorizzare il ruolo delle regioni; timore di prestazioni insufficienti, se tutto il carico fosse stato concentrato su di un unico sistema; timori per l'affidabilità, poiché un guasto del sistema centrale avrebbe reso impossibile il lavoro in tutta Italia, mentre ora sul polo è possibile fare almeno alcune operazioni, tra cui il prestito
- **vengono utilizzati diversi software (tutti basati sulla stessa analisi) e non uno unico**; questa scelta, in sé altamente opinabile, discende in realtà dalla precedente; a causa degli alti costi, non si poteva obbligare tutti i poli a dotarsi di un apposito mainframe per SBN, per cui bisognava permettere di utilizzare le macchina già disponibili in molti centri di calcolo; i mainframe però non sono normalmente compatibili tra loro, per cui diventava necessario realizzare programmi funzionanti in vari ambienti; di fatto, si trattò dei quattro software SBN della prima generazione, funzionanti in ambiente Bull, Unisys, IBM con il DBMS Adabas e IBM con il DBMS SQL/DS
- **i software sono realizzati appositamente**; con le premesse appena enunciate era difficile fare diversamente: si sarebbero dovuti trovare software già esistenti funzionanti in diversi ambienti ma tutti esattamente conformi alle stesse specifiche; inoltre, vista la portata strategica del progetto, probabilmente i promotori avevano il desiderio di gestire direttamente lo sviluppo dei prodotti invece di essere condizionati da strategie commerciali magari decise con riguardo al mercato americano; naturalmente tutto ha un prezzo, per cui l'uso di software diversi progettati ex novo comportò costi elevati e tempi di sviluppo molto lunghi

[59] Per questo essa può risultare utile anche a chi non lavora in SBN. Particolarmente interessante è la trattazione delle descrizioni a più livelli, molto più approfondita rispetto a quella che compare negli ISBD

[60] Si può notare, peraltro, che l'applicazione del modello E/R non è assolutamente rigorosa, perché non tutte le associazioni possibili sono trattate; ad esempio, l'editore è solo un elemento della descrizione, ma non esiste una entità *editore* alla quale riferire eventualmente diverse denominazioni e da mettere in collegamento con la descrizione.

In aggiunta, i software SBN della prima generazione condividevano una caratteristica che probabilmente non fu espressamente voluta come tale: erano molto più orientati al bibliotecario che all'utente, perché ad evolute funzionalità di controllo dei dati si contrapponeva la mancanza di una interfaccia di ricerca flessibile e facilmente utilizzabile.

Nel contesto determinato da queste premesse presero avvio, nel 1986, i primi poli. La situazione però aveva una caratteristica piuttosto anomala: l'indice non fu avviato contemporaneamente ai poli, né subito dopo, poiché mancavano a quell'epoca le risorse a livello centrale per la sua realizzazione. La mancanza dell'indice continuò per diversi anni, fino alla fine del 1992. Nel frattempo SBN si era venuto a trovare in una situazione piuttosto difficile. Non era da attendersi che un sistema così complesso entrasse immediatamente a regime non appena avviato, ed una fase di sperimentazione piuttosto lunga e complessa era del tutto prevedibile: tuttavia questa fase si prolungò in modo eccessivo, ed in particolare nel periodo 1990-1992 il rapporto costi-benefici di SBN era piuttosto negativo, perché i costi risultavano elevati, né poteva essere altrimenti con l'architettura utilizzata, ma i dati immessi crescevano lentamente e per di più, data la mancanza dell'indice, erano condivisi solo a livello di polo. Con l'avvio dell'indice, si rese necessario innanzitutto trasferirvi i dati in precedenza nei poli, con le cosiddette *migrazioni*. Non si poteva però semplicemente cumulare questi dati, perché si sarebbe dato luogo ad una esorbitante quantità di dati duplicati, con grave peggioramento della qualità del catalogo: le migrazioni comportarono quindi una serie di complesse operazioni di ricerca dei duplicati e loro eliminazione. L'indice utilizza un applicativo scritto appositamente, diverso da quelli dei poli, basato sul RDBMS IBM DB2 in ambiente OS/390 su mainframe IBM. A partire dal 1993, comunque, SBN ha decisamente preso quota, soprattutto per i benefici apportati dall'indice e anche perché ormai le quattro applicazioni della prima generazione hanno ormai raggiunto un buon livello di affidabilità[61]. Anche il ritmo di catalogazione è decisamente aumentato, per cui attualmente l'indice contiene oltre tre milioni di titoli e un milione di autori. La percentuale di catture, cioè di notizie bibliografiche recuperate dall'indice in fase di catalogazione si è rivelata in genere superiore al 90 % per la catalogazione di fondi correnti, mentre ovviamente è minore per fondi rari o specializzati. La qualità dei dati dell'indice, peraltro, è inferiore a quanto si potrebbe pensare considerato il rigore e la precisione degli standard catalografici adottati in SBN. Il problema di gran lunga più importante è senza dubbio la duplicazione di notizie, soprattutto titoli: molti titoli infatti sono stati inseriti due o più volte[62], aumentando inutilmente la quantità di dati e provocando incertezza e confusione nei catalogatori e negli utenti. Ci sono inoltre notizie contenenti errori di vario genere, a volte sottili ma altre volte alquanto grossolani. Di questi difetti, difficilmente si potrebbe dare la colpa a SBN nel suo complesso, ed in particolare ai suoi aspetti informatici: i software infatti mettono a disposizione tutti i mezzi per creare notizie corrette e per evitare i duplicati, ma alcuni catalogatori non li usano e inseriscono i dati senza fare le dovute verifiche.

L'aumentata potenza di calcolo dei computer portò abbastanza presto a prendere in considerazione l'idea di utilizzare per SBN non solo mainframe, ma anche macchine di fascia media in ambiente Unix, molto più economiche nell'acquisto e nella gestione, per cui all'incirca dal 1990 iniziò lo sviluppo di un software SBN per questo ambiente. Questo software fu commissionato direttamente dall'ICCU al CSI Piemonte e al Lombardia Informatica (le ditte che già avevano sviluppato la versione Adabas) in unica versione e non in molteplici versioni, ma ha avuto una storia assai travagliata. Consegnato nel 1992, non fu rilasciato ufficialmente che all'inizio del 1995. Messo alla prova, si rivelò ben presto pieno di difetti e scarsamente affidabile, per cui l'incarico di manutenzione del prodotto nel frattempo affidato alla Finsiel comportò interventi molto profondi (secondo i tecnici che se occuparono si trattò quasi di una riscrittura) sul software, che nel 1999 fu reso finalmente utilizzabile, anche se non ha caratteristiche particolarmente entusiasmanti, mentre permette effettivamente di abbattere i costi di impianto e gestione di un polo. Questo prodotto è analogo a quelli per mainframe sia per le funzionalità che per l'interfaccia, sempre a carattere (in emulazione VT100, come è tipico dei sistemi Unix), ed è basato sul RDBMS Ingres e sul TP Monitor X-Totem. Funziona

[61] Tra queste applicazioni, nel lungo periodo si è probabilmente rivelata come la migliore quella funzionante su Adabas in ambiente MVS su mainframe IBM

[62] Non mancano neppure i casi di titoli inseriti più volte dalla stessa biblioteca ...

su macchine IBM con sistema operativo AIX o Hewlett-Packard con HP-UX[63]: non si tratta di prodotti diversi, ma di porting dello stesso prodotto su diverse architetture hardware. Viene utilizzato da tre poli monobiblioteca, quello dell'ISTAT di Roma (che in seguito ha però adottato Sebina Indice, del quale si parlerà dopo), della Biblioteca Universitaria di Pisa e del Ministero della Sanità, e da qualche polo multibiblioteca, tra cui quello ligure, comprendente le seguenti biblioteche: Servizio Programmi e Strutture Culturali della Regione, Giunta Regionale, Consiglio Regionale, Universitaria di Genova, Civica di Imperia, Centro Sistema Bibliotecario Provinciale di La Spezia, Civica di Lerici, Civica di Castelnuovo Magra[64]. I primi due poli sono già entrati in produzione, mentre il Polo ligure stava concludendo i test quando è subentrata la necessità di adeguare al 2000 il software di base, che ha indotto a considerare la possibilità di utilizzare altri software nel frattempo divenuti disponibili, come poi in effetti è stato fatto, abbandonando quindi X-Totem.

SBN Unix/X-Totem non è l'unico software SBN per Unix disponibile: infatti l'applicativo basato su Adabas in ambiente mianframe IBM fu portato in ambiente Unix, per diminuire i costi di impianto e di gestione, e attualmente viene utilizzato in due Poli molto importanti come quelli della Regione Piemonte e della Regione Lombardia e in altri. Si tratta di un prodotto antiquato come interfaccia, ma efficiente e collaudato, e certamente superiore alla versione X-Totem.

SBN Unix/X-Totem, anche dopo la correzione della gran maggioranza degli errori, è, come detto, un prodotto dalle caratteristiche alquanto modeste, soprattutto come rapidità e flessibilità d'uso, destinato ad essere sostituito da una versione completamente nuova, che prevede una architettura client server a tre livelli, con client grafico per Windows e server Unix. Questa versione, dopo una elaborazione durata parecchi anni, è stata resa disponibile per i test all'inzio del 2000, rivelando, nei test condotti da diversi Poli tra cui quello ligure, notevoli potenzialità ma anche parecchi difetti di messa a punto. A partire dall'autunno del 2000 si sono però registrati notevoli progressi nello sviluppo, **per cui nei primi mesi del 2001 il software è stato adottato dal Polo SBN Ligure, che è entrato in produzione il 5.7.2001, primo polo ad impiegare client/server in ambiente di esercizio**. L'interfaccia utente di SBN client/server è molto ben progettata e, rispetto ai precedenti software SBN, segna un notevolissmo passo avanti in termini di flessibilità e chiarezza. L'architettura del programma è un esempio di client server a tre livelli: si ha infatti un front end Windows scritto in Power Builder che gestisce l'interfaccia utente, il middleware XSBN della Finsiel, scritto in Visual Basic sul lato client e in C sul lato server che gestisce la logica applicativa attivando, sul server, programmi in Cobol intepretato con SQL embedded che interagiscono con server di database che è il RDBMS Informxi Dynamic Server. Il back end prevede un database di polo e uno più database di biblioteca (che possono riferirsi a una singola biblioteca o un qualunque sottoinsieme di biblioteche del polo), che possono trovarsi anche su macchine diverse da quella del server di polo. Il database di biblioteca contiene dati gestionali e - quando è collocato su una macchina autonoma da quella di polo - permette alle biblioteche una limitata operatività anche quando il polo non è attivo[65]. L'architettura di SBN client/server può destare qualche perplessità di principio: infatti sul lato client prevede l'uso di un client "pesante", cioè di un grosso programma specializzato, soluzione normale alcuni anni fa quando iniziò lo sviluppo, ma che ora gode meno favore rispetto soprattutto all'uso dei client WWW (si ripropone il problema degli eccessivi tempi di sviluppo dei software SBN), anche se il client è in grado di girare su macchine ormai piuttosto superate[66] e non ha dato particolari problemi di installazione e configurazione, mentre sul lato server prevede l'impiego di numerosi componenti tra cui in particolare il RDBMS Informix. In precedenza era richiesto anche il TP monitor Tuxedo, che però nelle versioni rilasciate a partire dall'inizio del 2001 è stato eliminato, con notevole vantaggio economico per gli utenti, essendo Tuxedo un prodotto piuttosto costoso. Per quanto riguarda l'hardware lato server, non sono ammesse le macchine

[63] Esistena anche una versione per Digital Unix poi abbandonata perché il s.o. non è più supportato dalla Digital.
[64] Altre biblioteche si sono aggiunte dopo l'abbandono di questo software
[65] Nel polo ligure viene usato un solo database di biblioteca e un solo server. La configurazione multiserver ha l'inconveniente di non essere molto economica, poiché ogni server richiede la stessa configurazione software del server centrale di polo. Si può risparmiare sull'hardware, ma poiché in molti casi anche per il server di polo sono sufficienti macchine di fascia bassa relativamente alla rispettiva gamma, il costo effettivo del server di biblioteca finisce per non essere molto inferiore
[66] Si può affermare che un Pentium 233 MHz con 64 Mb di RAM sia una configurazione del tutto sufficiente

basate su Intel, per cui bisogna impiegare server di fascia media, in particolare IBM con AIX (adottato nel Polo Ligure), Hewlett-Packard con HP-UX e Sun con Solaris.

SBN ha avuto sviluppi anche per quanto riguarda l'accesso pubblico. Già da parecchi anni funziona un OPAC dell'indice a carattere, in emulazione 3270, semplice ma efficace. Nel 1996 fu attivato un OPAC molto più sofisticato, che utilizza un archivio diverso da quello di produzione, gestito in ambiente Aix tramite Basis. Questo è conforme allo standard Z39.50 ed è consultabile, sempre tramite Internet, sia via WWW (http://opac.sbn.it) sia con un client Z39.50 (l'host è sempre opac.sbn.it), ed in particolare con il client SBN-Z sviluppato appositamente per l'uso in SBN e dalle buone caratteristiche (si può comunque utilizzare qualunque client). L'OPAC di polo ha avuto invece una storia molto più travagliata (nel 1996 sembrava di imminente rilascio, ma poi si perse nelle nebbie), per cui non esiste ancora un OPAC SBN di polo ufficiale, e nei vari poli vengono utilizzati soluzioni diverse, che in genere prevedono lo scarico dei dati dal RDBMS di produzione ad un information retrieval.

Un'altra novità di grande importanza è la decisione di consentire l'uso in SBN di prodotti non nativi, ossia di altri software per gestione bibliotecaria che vengano messi in grado di colloquiare con l'indice tramite l'aggiunta di apposite procedure. Finora vi sono tre produttori che stanno seguendo questa strada, l'Akros con Sebina Produx, la Nexus con Easycat, un originale sistema di catalogazione con interfaccia WWW che usa Isis in ambiente Unix come motore di database e la Atlantis con Aleph, prodotto molto utilizzato nelle università che non aderiscono ad SBN. Il 26 novembre 1998 è stato presentato il prodotto dell'Akros, reso poi disponibile nella seconda metà del 1999, che risulta dotato di buone qualità, e viene usato con molto successo anche il Poli di grandi dimensioni, come il Polo Unificato di Bologna e il Polo di Romagna. Per quanto riguarda gli altri due prodotti, non si hanno attualmente notizie precise sull'epoca di rilascio. Sebina Indice, rispetto a SBN client/server prevede una architettura più semplice (accesso via telnet, e software di base limitato al solo runtime del Progress. il RDBMS sul quale Sebina si basa) e requisiti hardware più bassi rispetto a SBN client/server (sono ammessi anche i server basati su Intel), ma comporta costi non trascurabili per la licenza del software e la manutenzione. Nel febbraio 2000 l'Akros ha annunciato ufficialmente il supporto per Linux, in seguito all'introduzione del supporto Linux per il Progress: è quindi per la prima volta possibile gestire un polo SBN con un sistema Linux[67].

Non si può neanche escludere che a lungo termine i software adattati per il colloquio con l'indice finiscano per sostituire quelli nativi SBN, cosa che difficilmente potrebbe essere considerata desiderabile perché i secondi - e questo vale in particolare per il client/server - hanno il vantaggio di essere programmi di proprietà pubblica e non derivati da prodotti commerciali. Naturalmente però i prodotti nativi SBN non potranno far valere questo vantaggio se non dimostreranno anche adeguate qualità tecniche.

Per quanto riguarda l'unico prodotto nativo SBN attualmente proponibile (considerato che i sw per mainframe sono in fase di dismissione[68], e quello per Unix/X-Totem costituisce al più una soluzione di transizione, come del resto quello per Adabas/Unix, ancorché di migliore qualità), cioè SBN client/server, il suo uso in produzione nel Polo Ligure dà finalmente conforto a coloro che desiderano che continui la disponibilità di prodotti SBN interamente di proprietà pubblica e che sperano che questo software continui a dare buoni risultati anche nel medio e lungo periodo. Sul piano strategico, un fallimento di SBN client/server, sarebbe stato infatti molto negativo, perché - a differenza di quanto accaduto finora - avrebbe determinato la completa mancanza di un prodotto SBN di proprietà pubblica, ma per fortuna questa prospettiva sembrerebbe ora scongiurata.

15.2 CBL

[67] In realtà da tempo alcuni utenti impiegavano Sebina Produx (non quindi la versione SBN) sotto Linux, utilizzando gli eseguibili per SCO Unix e l'emulatore IBCS, ma questa soluzione non era ufficialmente supportato dall'Akros.

[68] Per la versione su mainframe Unisys non è neppure più in commercio l'hardware necessario, e i poli che hanno bisogno di sostituire dei componenti devono acquistarli sul mercato dell'usato.

CBL è la sigla di *Catalogo delle Biblioteche Liguri*. CBL nasce in un'epoca e in un contesto completamente diversi da quelli di SBN. Nell'autunno 1994 la Regione Liguria cercava di elaborare un progetto di informatizzazione che potesse coinvolgere tutte le biblioteche del territorio regionale[69], le quali però negli anni precedenti, in assenza di un tale progetto, avevano adottato autonomamente diversi sistemi di automazione. Risultò chiaro quindi che questi sistemi non si potevano totalmente unificare, mentre si poteva cercare di arrestarne la proliferazione e di rendere i dati disponibili almeno per la consultazione da parte del pubblico, realizzando quindi un OPAC. La produzione dei dati avrebbe continuato ad essere effettuata localmente. Come mezzo per la consultazione si individuò Internet, mentre tra i vari programmi utilizzati sul territorio ligure si rilevò che solo quattro erano di livello professionale, cioè Sebina Produx, Tinlib, Erasmo e Isis/Teca, per cui solo di questi si decise di garantire l'importazione nel CBL[70]. Come formato di scambio di dati si scelse Unimarc, salva la possibilità di impiegare formati diversi per programmi che non disponessero dello scarico in Unimarc.

Per la realizzazione del CBL si pensò in primo tempo di usufruire di un servizio del CILEA di Segrate (il CILEA è un centro di calcolo delle università lombarde), che gestiva il *Catalogo Collettivo delle Università Padane*, poi denominato *delle Università Lombarde*, che era all'epoca una vecchia applicazione consultabile da Internet con una interfaccia a carattere[71]. Il CILEA si dichiarò disponibile a mettere online un altro database di struttura analoga contenente il CBL. Questa ipotesi non si poté realizzare per difficoltà amministrative, per cui nel corso del 1995 prese corpo il progetto di una realizzazione e gestione diretta del CBL da parte della Regione. Nel dicembre 1995 fu affidato alla Datasiel di Genova, società di informatica a partecipazione regionale, l'incarico di realizzare il software per il CBL, utilizzando il motore di information retrieval Highway. I lavori furono poi sospesi tra l'aprile 1996 e l'aprile 1997 per l'annunciata disponibilità dell'OPAC di polo SBN, che avrebbe potuto rendere superfluo lo sviluppo di una applicazione apposita per il CBL. Non essendo poi stato reso disponibile l'OPAC di Polo SBN, ripresero i lavori per il CBL, che iniziò il servizio al pubblico il 24 febbraio 1998, ed è consultabile all'URL **http://opac.regione.liguria.it/cgi-win/hiweb.exe/a3** oppure **http://www.regione.liguria.it/cbl/**. Nel maggio 2001 il database contava circa 226.000 record. Nell'autunno del 2000 è stata completata la seconda versione del software, caratterizzata da numerosi miglioramenti, che è stata messa online nel maggio 2001.

Il CBL, come detto, è basato su Highway e permette di importare dati in formato Unimarc. Attualmente ospita dati prodotti con Isis/Teca ed importati però non tramite Unimarc, ma tramite un Iso2709 opportunamente formattato in fase di scarico da Isis, e dati Unimarc originati da Sebina Produx. L'importazione di dati Unimarc prodotti da Erasmo e Tinlib non è ancora iniziata perché nel primo caso sono stati riscontrati problemi di correttezza del formato, mentre nel secondo il produttore non ha ancora rilasciato la procedura di conversione, pure annunciata da lungo.

Esiste anche la versione del CBL su CD-ROM ha completato i test nell'ottobre 2000 per cui se ne prevede l'uscita nel 2001.

---

[69] Per la verità, nel 1993 era stato elaborato un altro progetto, denominato CCL, cioè *Catalogo Collettivo Ligure*, che, sottoposto all'esame delle principali biblioteche della regione e della sezione ligure dell'Associazione Italiana Biblioteche fu sostanzialmente rifiutato, senza peraltro che contro il progetto fosse stata avanzata alcuna critica pertinente

[70] A questi quattro si dovrebbe aggiungere, come programma di livello professionale, Aleph, utilizzato dall'Università; anche se non è tra i programmi di cui viene garantita comunque l'importazione, quasi certamente questa sarebbe tecnicamente possibile con poche difficoltà

[71] Attualmente il *Catalogo delle Università Lombarde* è consultabile su Internet via Web, partendo dall'URL http://www.cilea.it/Virtual_Library/

16. BIBLIOGRAFIA

Le edizioni sono elencate in ordine alfabetico di titolo.

[Advanced 1997] Advanced database systems / Carlo Zaniolo [et al.]. - San Francisco : Morgan Kaufmann, c1997. - ISBN 1-55860-443-X.

[Basi 1992] Basi di dati : strutture e algoritmi / A. Albano. - Milano : Addison Wesley Masson, 1992. - ISBN 88-7192-046-5.

[Basi 1999] Basi di dati / Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone. - 2. ed. - Milano : McGraw-Hill, 1999. - ISBN 88-386-0824-5.

[Crittologia 1996] Crittologia : come proteggere le informazioni riservate / Luigia Berardi, Albert Beutelspacher. - Milano : Angeli, c1996. - ISBN 88-204-9898-7.

[Database 1997] Database system concepts / Abraham Silberschatz, Henryk F. Korth, S. Sudarshan. - 3. ed. - Boston [etc.] : WCB/McGraw-Hill, c1997. - ISBN 0-07-044756-X.

[Datenbanken 1998] Datenbanken mit Linux / Bernard Röhrig. - Vaterstetten : C&L, c1998. - ISBN 3-932311-32-9.

[Datenbanken 2000] Datenbanken unter Linux : Oracle 8i, MySQL, Adabas, Informix, Sybase, DB2, PostreSQL, MiniSQL, Empress / Hans Dicken, Gunther Hipper, Peter Müßig-Trapp. - Bonn : MITP, c2000. - ISBN 3-8266-0555-1.

[Dati 1996] Dati e basi di dati : il modello relazionale / Vittorio Belski, Roberto Breschi, Fortunato Pigni, Maria T. Zocchi. - Milano : Angeli, c1996. - ISBN 88-204-7406-9.

[Digital 1991] Digital image processing / William K. Pratt. - 2. ed. - New York [etc.] : John Wiley, c1991. - ISBN 0-471-85766-1.

[Digital 2000] Digital media : the future / John Vince and Rae Ernshaw (eds.). - London : Springer, c2000. - ISBN 1-85233-246-8.

[Informatica 1983] Informatica e biblioteche : automazione dei sistemi informativi bibliotecari / Maria Bruna Baldacci, Renzo Sprugnoli. - Roma : La Nuova Italia Scientifica, 1983.

[Inside 1998] Inside CORBA : distributed object standards and applications / Thomas A. Mowbray, William A. Ruh. - Reading, Mass. [etc.] : Addison Wesley, c1998. - ISBN 0-201-89540-4.

[Managing 1999] Managing gigabytes : compressing and indexing documents and images / Ian H. Witten, Alistair Moffat, Timothy C. Bell. - 2. ed. - San Francisco : Morgan Kaufmann, c1999. - ISBN 1-55860-570-3.

[Multidimensionale 1999] Multidimensionale Datenbanksysteme : Modellierung und Verarbeitung / Wolfgang Lehner. - Stuttgart ; Leipzig : Teubner, 1999. - ISBN 3-519-00310-4.

[Preserving 2000] Preserving digital informazione : a how-to-do-it manual / Gregory S. Hunter. - New York ; London : Neal-Schuman, c2000. - ISBN 1-55570-353-4.

[Rappresentazione 1988] Rappresentazione e ricerca delle informazioni : come comunicare attraverso i sistemi informativi automatizzati / Maria Bruna Baldacci. - Roma : La Nuova Italia Scientifica, 1988.

[Sistemi 1992] Sistemi di basi di dati orientate agli oggetti : concetti e architetture / Elisa Bertino, Lorenzo Dario Martino. - Milano : Addison Wesley Masson, 1992. - ISBN 88-7192-036-8.

[Sistemi 1991] Sistemi informativi pittorici : principi e progettazione / Shi-Kuo Chang. - Milano : Jackson ; Hemel Hempstead : Prentice Hall, 1991. - ISBN 8825602634

[SQL 1991] SQL / Jean-Louis Marx. - Milano : Jackson, c1991. - ISBN 88-256-0299-5.

[SQL-99 1999] SQL-99 complete : really / Peter Gulutzan and Trudy Pelzer. - Lawrence (Kansas) : R&D, c1999. - ISBN 0-87930-568-1.

[Sviluppo 2000] Sviluppo di database per il Web passo per passo / Jim Buyens. - [Milano?] : Mondadori Informatica, 2000. - ISBN 88-8331-112-4.

[Tecniche 1989] Tecniche di organizzazione delle informazioni : [strutture di dati e tecniche di accesso alle memorie di massa] / Leonardo Felician. - Milano : Mondadori Informatica, 1989. - ISBN 88-7131-010-1

[Teoria 1980] Teoria degli insiemi / K. Kuratowski, A. Mostowski ; traduzione e cura di Palladino Dario. - Roma : Abete, 1980.

[Unimarc 2000] Unimarc / Antonio Scolari. - Roma : AIB, 2000. - ISBN 88-7812-061-8.

APPENDICI

Le appendici trattano di argomenti che non si sono inseriti nella trattazione principale per non appesantire l'esposizione, ma possono ugualmente essere lette con profitto soprattutto da coloro che sono interessati ad approfondire alcune questioni tecniche.

I. Esempi di codice.

Daremo qui alcuni esempi di codice di programmi per l'accesso a file di database. Si tratta di codice scritto dall'autore di questo documento, allo scopo di accedere a basso livello a file di database CDS-ISIS e dBase. Accesso a basso livello significa che il codice accede ai dati direttamente attraverso la struttura fisica dei file, e non attraverso i servizi di un motore di database come i linguaggi di sviluppo contenuti in molti software di database come CDS-ISIS, tutta la famiglia dBase, Paradox ecc.

ATTENZIONE: questo codice viene riportato solo per illustrare, a scopo didattico, come si possano gestire database a basso livello e non come modello di buona programmazione, per cui non si esclude che contenga difetti, imperfezioni o errori veri e propri. Non si garantisce in alcun modo il suo perfetto funzionamento.

Vediamo innanzitutto il codice in Turbo Pascal per DOS[72] per accedere ad un archivio di CDS-ISIS. Il lavoro viene fatto dalla funzione `cercarecord()` il cui valore di ritorno qui non ci interessa in dettaglio perché viene usato da altre funzioni e procedure. Alcune parti di `cercarecord()` sono state omesse perché non essenziali alla comprensione. Si noterà in nanzitutto la funzione `fai_vedere` che ha lo scopo di mostrare i dati inseriti dal corpo principale della funzione nei due array `dati` e `listacampi`, che contengono rispettivamente la struttura del record e i dati. Non commentiamo la procedura `Iso2709`, che produce dati in formato ISO 2709 (e peraltro è notevolmente più lenta di quella del programma originale). Passando al corpo principale, si noterà che il codice è alquanto complesso perché, partendo dal file .xrf e passando poi al file .mst, deve leggere tutti i diversi livelli di indice necessari per identificare la posizione dei campi a lunghezza variabile. La funzione viene eseguita una volta per ogni record, per cui viene normalmente richiamata in un ciclo controllato dal suo valore di ritorno. Puntiamo l'attenzione sulle due istruzioni `seek(mfile, puntogiusto + base); e blockread(mfile,dati,(mfrl - base));`: la prima, in base ai dati acquisiti dagli indici, si posiziona all'inizio di un record, mentre la seconda legge tutto il record, la cui lunghezza è `(mfrl - base)` in un array di caratteri. La funzione è in grado di sapere dove iniziano i singoli campi in questo array perché, attraverso gli indici, ha acquisto tutte le informazioni sulla struttura del record., che viene letta in un array di record, dove ogni record contiene le informazioni che descrivono un campo. Dovrebbe essere evidente la notevole complessità del codice necessario per accedere ai dati, complessità che deriva dalla struttura a sua volta complessa degli indici, la quale poi è resa necessaria dalla presenza di campi a lunghezza variabile e dall'orientamento di CDS-ISIS alla gestione di testi strutturati e non di tabelle: la struttura del record, infatti, viene memorizzata per ogni record e non una volta sola per tutto il database, perché può essere diversa per ogni record. Il file .fdt, che descrive la struttura dell'archivio, serve soprattutto come documentazione per l'utente, per indicare quello che il progettista ha inteso essere il tracciato record, e in base al quale sono stati creati i formati di output, le maschere di immissione dati e gli altri elementi dell'archivio, ma non determina affatto la struttura fisica dell'archivio, tanto che il programma non ne fa uso allorché deve accedere ai dati.

```
function cercarecord(numero: longint; switch: integer): char;
```

[72] Anche fosse per Windows o per Unix il codice però non sarebbe però molto diverso, perché è guidato più dalla struttura dell'archivio che dalle caratteristiche del sistema operativo, al quale si richiede essenzialmente solo la possibilità di effettuare l'accesso casuale ad un file, qui implementato tramite la funzione procedura standard del Turbo Pascal `seek()`.

```
type elemento = record
                      tag: integer;
                      pos: integer;
                      len: integer;
                end;

var  xrfmfb: longint;
     xrfmfp: longint;
     puntogiusto: longint;
     dove: longint;
     punta: longint;
     conta: integer;
     car: byte;
     mfrl: integer;
     base: integer;
     nvf: integer;
     x: char;
     dati: array[0..8000] of char;
     listacampi: array[1..5000] of elemento;
     info: integer;                 {elemento dell'indice del record}
     conta2: integer;
     cancellato: boolean;
     differenza: integer;


{parti omesse}


function fai_vedere: char;         {mostra i dati a video}

var conta: word;
    conta2: longint;

begin
writeln('  >>>  MFN ',numero:12);
if cancellato then
  writeln('LOGICALLY DELETED RECORD ');
writeln;
for conta:=1 to nvf do begin                  {lo scrive usando l'indice}
   write(listacampi[conta].tag:4,' '+chr(16)+' ');
   for conta2:=listacampi[conta].pos to
       (listacampi[conta].pos + (listacampi[conta].len - 1)) do
           write(dati[conta2]);          {scrive l'array}
   writeln;
   if wherey >= 45 then begin            {fine pagina}
               gotoxy(10,47);
               write('Any key to continue ...  ');
               x:=readkey;
               clrscr;
               writeln('  >>>  MFN ',numero:12);
               writeln;
   end;
end;                                    {for conta:= to nvf}
repeat
       x:=readkey;
until (ord(x) = 0) or (x in [chr(27), chr(9)]);                  {tasto speciale}
if ord(x) <> 0 then
   fai_vedere:=x
else
   fai_vedere:=readkey;            {legge il codice tasto speciale}
end;


{parti omesse}
```

```
procedure iso2709;                    {scrive in formato ISO 2709}

var data: pchar;
    indice: pchar;
    intestazione: pchar;
    fld_delim: pchar;
    rec_delim: pchar;
    c: pchar;                         {carattere in elaborazione}
    conta: word;                      {contatore dei campi}
    conta2: word;                     {contatore dei caratteri}
    conta3: word;                     {usato per dividere in righe}
    tag: pchar;
    lungh: pchar;
    posizione: pchar;
    l_totale: pchar;    {lunghezza del record ISO}
    inizio_dati: pchar; {indirizzo di inizio dati}
    temp: pchar;                      {usato per la divisione in righe}
    tutto: pchar;                     {usato per la divisione in righe}

begin
fld_delim:='#';
rec_delim:='##';
getmem(intestazione,25);
getmem(indice,2500);
getmem(data,8500);
getmem(c,2);
getmem(tag,4);
getmem(lungh,5);
getmem(posizione,6);
getmem(l_totale,6);
getmem(inizio_dati,6);
getmem(temp,81);
getmem(tutto,11026);
if not cancellato then begin
    fillchar(intestazione^,25,0);
    fillchar(indice^,2500,0);
    fillchar(data^,8500,0);
    fillchar(l_totale^,6,0);
    fillchar(inizio_dati^,6,0);
    fillchar(tutto^,11026,0);
    for conta:=1 to nvf do begin
          strcat(data,fld_delim);            {** PARTE DATI **}
          if (listacampi[conta].len -1) > -1 then
          for conta2:=listacampi[conta].pos to
            (listacampi[conta].pos + (listacampi[conta].len - 1)) do begin
                strpcopy(c,dati[conta2]);
                strcat(data,c);
          end;
          strpcopy(tag,zeri(int_to_string(listacampi[conta].tag),3));  {** INDICE **}
          strpcopy(lungh,zeri(int_to_string(listacampi[conta].len+1),4));
          strpcopy(posizione,zeri(int_to_string(listacampi[conta].pos+(conta-1)),5));
          strcat(indice,tag);
          strcat(indice,lungh);
          strcat(indice,posizione);
    end;
    strcat(data,rec_delim);
    strpcopy(l_totale,zeri(int_to_string(24+strlen(indice)+strlen(data)),5));
    strcat(intestazione,l_totale);
    strcat(intestazione,'0000002');
    strpcopy(inizio_dati,zeri(int_to_string(24+strlen(indice)+1),5));
    strcat(intestazione,inizio_dati);
    strcat(intestazione,'0004500');
    strcat(tutto,intestazione);        {crea un unico array che contiene}
    strcat(tutto,indice);              {tutto il record}
    strcat(tutto,data);
    case switch of
     9: writeln(bersaglio,tutto);
```

```
    10: begin                       {scrive dividendo in righe}
        fillchar(temp^,81,0);
        for conta3:=0 to (strlen(tutto)-1) do begin
           strpcopy(c,tutto[conta3]);
           strcat(temp,c);
           if ((conta3 + 1) mod 80 = 0) then begin    {fine riga}
              writeln(bersaglio,temp);
              fillchar(temp^,81,0);
           end;
        end;
        writeln(bersaglio,temp);   {ultima riga del record}
        end;
    end;
    writeln('MFN ',numero,' exported');
end;
freemem(intestazione,25);
freemem(indice,2500);
freemem(data,8500);
freemem(c,2);
freemem(tag,4);
freemem(lungh,5);
freemem(posizione,6);
freemem(l_totale,6);
freemem(inizio_dati,6);
freemem(temp,81);
freemem(tutto,11026);
end;

{parti omesse}

begin                                {corpo principale di cercarecord}
     if switch = 99 then             {switch 99 per seq_read}
        if seq_read then
           cercarecord:=chr(255)
        else
           cercarecord:=chr(254)
     else begin
     cancellato:=false;
     differenza:=numero div 128;
     dove:=numero + differenza;    {calcola il record dell'xrf}
     if (dove div 128) <> differenza then  {caso in cui Š multiplo di 128}
        dove:=numero + (dove div 128);
     seek(xrf, dove);                    {va al punto giusto}
     read(xrf,punta);
     xrfmfb:=(punta and $FFFFF800) div 2048;
     xrfmfp:=(punta and 2047);
     if xrfmfp > 512 then                {record non aggiornato nell'IF}
        xrfmfp:=xrfmfp - 512;
     if not (xrfmfb < 0) then begin   {non è cancellato logicamente o fisicamente}
          reset(mfile,1);                     {assegna 1 come valore di record}
          reset(mfile2);
          puntogiusto:=((xrfmfb -1) * 512) + xrfmfp;  {inizio del record}
          seek(mfile2, (puntogiusto div 2) + 2);    {va sul mfrl}
          if not tutto_pronto then                  {I/O IMPOSSIBILE}
             exit;
          read(mfile2, mfrl);                       {legge la lunghezza del record}
          mfrl:=abs(mfrl);
          if mfrl_errato  then begin
             writeln(chr(7),'WRONG DATA ! IMPOSSIBLE TO CONTINUE   Any key to exit');
             readkey;
             exit;
          end;
          seek(mfile2, filepos(mfile2) + 4);        {va su mvf}
          read(mfile2, nvf);                        {legge nvf}
          if nvf_errato then begin
             writeln(chr(7),'WRONG DATA !  IMPOSSIBLE TO CONTINUE   Any key to exit');
             readkey;
             exit;
```

```
          end;
          base:=18 + (6 * nvf);                          {calcola base}
          fillchar(listacampi,sizeof(listacampi),0);     {lo riempie di zeri}
          seek(mfile2,filepos(mfile2) + 1);              {va avanti di un interi}
          for conta:=1 to nvf do begin                   {legge l'indice}
              read(mfile2,listacampi[conta].tag);
              read(mfile2,listacampi[conta].pos);
              read(mfile2,listacampi[conta].len);
          end;                                    {for conta:=1 to nvf}
          seek(mfile, puntogiusto + base);               {va a inizio dati}
          blockread(mfile,dati,(mfrl - base));           {legge il record}
          if (switch = 1) then
                 cercarecord:=fai_vedere
          else begin                                    {non scrive su video}
                 case switch of
                      2: scrittura;
                      3: expo;
                      4: lista;
                      5: lista2;
                      6: delim;
                      7: ricerca;
                      8: delim2;
                      9,10: iso2709;
                      11: expo2;
                 end;
                 if switch <> 7 then
                    cercarecord:=chr(0);
          end;
      end
      else begin
          writeln('RECORD MFN ',numero:10,' deleted !');
          cercarecord:=chr(9);
      end;
      end;                 {if switch = 99}
end;
```

Vediamo ora un esempio di codice, sempre in Turbo Pascal per DOS, per leggere un file in formato dBase. Poiché questo codice non prende in considerazione i campi memo, dovrebbe funzionare con qualunque file dBase (dbase III, IV, Fox ecc.), ma è stato provato solo con file dBase III e IV. Si noterà subito che il codice è molto più semplice di quello usato per CDS-ISIS: il motivo è che la struttura dell'archivio è registrata una volta sola, è uguale per tutti i record, e i campi sono a lunghezza fissa per cui, una volta letta la struttura dell'archivio (anche qui in un array di record) è sufficiente un ciclo che si posizioni all'interno del file per effettuare successive letture nei punti facilmente calcolabili in base alla struttura dell'archivio e alla lunghezza dei campi. Le tre procedure riportate svolgono le seguenti funzioni: `open_dbf` legge la struttura dell'archivio nella variabile `lista_campi` che è un puntatore ad un array di record che contengono la descrizione dei vari campi; `read_data` legge un record in un array di caratteri, dopo essersi posizionata al punto giusto dopo un semplice calcolo (numero del record da leggere moltiplicato per la lunghezza del record - che è fissa - più la lunghezza dell'intestazione del file); infine `show_data` manda in output i dati formattando l'array di caratteri in cui ha letto i dati in base alle informazioni sulla struttura del record contenute in `lista_campi`.

```
{-------------------------------------------------------------------------}
procedure show_data(hdr_len,rec_len,num_campi: integer; reccount: longint;
                    campi: field_list; var  dati: array of char);

var contarecord: longint;
    cnt: word;
    contacampi: byte;
    campo: string;
    lista_campi: ^field_list;

begin
```

```
lista_campi^:=campi;
seek(f4,hdr_len);                 {va a inizio dati}
for contarecord:=1 to reccount do begin
    fillchar(dati,4000,0);
    blockread(f4,dati,rec_len);           {legge il record}
    write(contarecord,' ');
    write(dati[0],' ');                       {il primo carattere Š a parte}
    cnt:=1;
    for contacampi:=1 to num_campi do begin        {fa passare i campi}
         campo:=arr_to_string(dati,cnt,cnt+lista_campi^[contacampi].len-1);
         write(campo);
         gotoxy(wherey+3,wherex);
         cnt:=cnt+lista_campi^[contacampi].len;  {inizio campo successivo}
    end;
    writeln;
end;
end;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
procedure read_data(nr: longint; hdr_len,rec_len: integer;              {legge un singolo
record}
                    var dati: array of char);

begin
seek(f4,hdr_len+((nr - 1) * rec_len));                     {va al record specificato}
fillchar(dati,4000,0);
blockread(f4,dati,rec_len);           {legge il record}
end;
{-------------------------------------------------------------------------}


{-------------------------------------------------------------------------}
procedure open_dbf(nome: string; var struttura: field_list;
                   var totale: longint; var quanti_campi: byte;
                   var anno: byte; var mese: byte; var giorno: byte;
                   var hdr_len: integer; var rec_len: integer);


var descr_len: integer;
    reccount: longint;
    num_campi: integer;
    field_count: integer;
    lista_campi: ^field_list;


begin
if nome <> '' then begin
   assign(f1,nome);
   assign(f2,nome);
   assign(f3,nome);
   assign(f4,nome);
   assign(f5,nome);
   reset(f1);
   if not tutto_pronto then
      exit
   else begin
      reset(f2);
      reset(f3,1);
      reset(f4,1);
      reset(f5);
      seek(f1,1);
      read(f1,reccount);            {legge il numero di record}
      close(f1);
      seek(f2,4);
      read(f2,hdr_len);             {legge la lunghezza dell'intestazione}
      read(f2,rec_len);             {legge la lunghezza del record}
```

```
      close(f2);
      descr_len:=hdr_len - 33;      {lunghezza parte descrizione campi}
      num_campi:=descr_len div 32;      {calcola il numero di campi}
      seek(f3,32);
      getmem(lista_campi,sizeof(field_list));
      fillchar(lista_campi^,sizeof(field_list),0);
      for field_count:=1 to num_campi do
          blockread(f3,lista_campi^[field_count],32);
      close(f3);
      seek(f5,1);
      read(f5,anno);                  {legge data ultimo aggiornamento}
      read(f5,mese);
      read(f5,giorno);
      close(f5);
      struttura:=lista_campi^;
      totale:=reccount;
      quanti_campi:=num_campi;
      close(f4);
      freemem(lista_campi,sizeof(field_list));
   end;                 {if not tutto_pronto}
end;                    {if nome <> ''}
end;
{-------------------------------------------------------------------------}
```

Ecco del codice in C che svolge più o meno le stesse funzioni di quello in Turbo Pascal visto poc'anzi per accedere a file dBase. Questo codice è stato compilato sotto Linux con il GNU C Compiler, ma dovrebbe essere compilabile sulla maggior parte dei sistemi Unix e anche DOS/Windows. Viene riportato innanzitutto un file di intestazione che contiene una `struct` (equivalente al record in Pascal) che descrive la struttura dell'intestazione del file. Segue il programma vero e proprio, che funziona in modo analogo a quello in Pascal, anche se è meno completo e testato **(attenzione: potrebbe contenere errori!)**.

Questo è l'header file.

```
/* header file for xbase data file processing */

char __TERMINATOR__ = 13;

struct xbase_str             /*xbase file structure*/
{
char ident;                  /* 0     */
char date[3];                /* 1-3   */
long int recno;              /* 4-7   */
short int head;              /* 8-9   */
short int rec_length;        /* 10-11 */
char res1[2];                /* 12-13 */
char trans;                  /* 14    */
char crypt;                  /* 15    */
char lan[12];                /* 16-27 */
char mdx;                    /* 28    */
char res2[3];                /* 29-31 */
};
```

Questo è il programma vero e proprio.

```
#include <stdio.h>
#include "xbase.h"
#include "beppe.h"
#include "beppe.c"

/*function prototypes*/

char *ftype(unsigned char t);
```

```
void table(FILE *f, struct xbase_str fstr);
void explanation();

/*end of function  prototypes*/


main(int argc, char *argv[])

{

struct xbase_str    fstruct;
struct xbase_fields ffields;
struct xbase_fields fields_list[255];
int fcount;
int total_fields;
char fname[60];
FILE *datafile;
char choice[1];


printf("\n");
printf("%s \n",__RELEASE_XBASE__);
printf("\n");
info();
printf("\n");
if (argc < 2) explanation();
        else
{
       sscanf(argv[1],"%s",fname);
       datafile = fopen(fname,"rb");
       fread(&fstruct, sizeof(fstruct), 1, datafile);
       total_fields = (fstruct.head-32)/32;
       for (fcount = 0; fcount <= total_fields-1; fcount++)
              {
             fread(&fields_list[fcount], 32, 1, datafile);
             }
       printf("\n");
       printf("DISPLAYING INFORMATION ABOUT FILE %s \n",fname);
       printf("\n");
       printf("Last updated %u-%u-%u \n",fstruct.date[0],fstruct.date[1],fstruct.date[2]);
       printf("Number of fields: %u \n",total_fields);
       printf("Number of records: %u \n",fstruct.recno);
       printf("Header length: %i \n",fstruct.head);
       printf("Record length: %i \n",fstruct.rec_length);
       printf("\n");
       printf("Displaying structure of %s \n",fname);
       printf("\n");
       printf("NAME           TYPE           LENGTH \n");
       printf("-------------------------------------\n");
       for (fcount = 0; fcount <= total_fields-1; fcount++)
             {
              printf("%-14s ",fields_list[fcount].name);
                 printf("%-14s ",ftype(fields_list[fcount].type));
                 printf("%-9u \n",fields_list[fcount].field_length);
               }
       printf("\n");
        if (argc == 3)
             {
                    sscanf(argv[2],"%s",choice);
                    switch(choice[0])
                           {
                            case 116:    /*116 = t*/
                                  table(datafile, fstruct);
                                  break;
                           }
             }                                  /*if argc == 3*/
       fclose(datafile);
}
```

```
}

/*-------------------------------------------------------------*/
/* Depending on the parameter, returns a string corresponding to the */
/* data type of a dbase file field */

char *ftype(unsigned char t)

{
switch(t)
       {
          case 67:             /*67 = C*/
              return "Character";
              break;
          case 68:              /*68 = D*/
               return "Date";
              break;
          case      70:              /*70 = F*/
              return "Float";
              break;
          case 76:                 /*76 = L*/
               return "Boolean";
              break;
          case 77:                 /*77 = M*/
               return "Memo";
              break;
          case 78:             /*78 = N*/
               return "Integer";
              break;
          default:
               return "Unknown";
              break;

       }
}
/*-------------------------------------------------------------*/


/*-------------------------------------------------------------*/
/* Prints a dbase file in table form                           */

void table(FILE *f, struct xbase_str fstr)

{
char data[20000] = "";
int counter;
long reccount;

printf("Displaying data \n\n");
printf("-----------------------------------------------\n");
fseek(f,2,SEEK_CUR);      /*skips two bytes to go to beginning of data*/
for (reccount = 1; reccount <= fstr.recno; reccount++)
     {
      for (counter = 0; counter <= fstr.rec_length; counter++)
       data[counter] ='\0';
      fread(&data,fstr.rec_length,1,f);
      printf("| %8u | ",reccount);
      for (counter = 0; counter <= fstr.rec_length; counter++)
        printf("%c",data[counter]);
     }
```

HISTORY

Versione preliminare: dicembre 1998
Versione 1: marzo 1999
Versione 2: febbraio 2000
Versione 2.1: 6.8.2000
Versione 2.2: 18.10.2000
Versione 2.2.1: 12.1.2001
Versione 3: 18.7.2001