**4 - Database**

**1. Database: Generalità**

**1.1 Cos’è un database**

Con il termine database (base di dati) si indica l’insieme dei dati utilizzati in uno specifico sistema informativo, di tipo aziendale, scientifico, amministrativo o altro.

Un database è composto da due diversi tipi di informazione, appartenenti a distinti livelli di astrazione:

I dati, che rappresentano le entità del sistema da modellare. Le proprietà di tali entità sono descritte in termini di valori (di tipo numerico, alfanumerico ecc.). I dati, inoltre, sono raggruppati o classificati in categorie in base alla loro struttura comune (per esempio Libri, Autori);

Le strutture (metadati), che descrivono le caratteristiche comuni delle varie categorie di dati, quali i nomi e i tipi dei valori delle proprietà. Per esempio, il database utilizzato in una biblioteca può contenere, tra gli altri, i dati di Figura 1.1.

La struttura di tali dati può, in uno specifico database, essere descritta nel modo seguente:

**Libri:**

Codice-Libro: numerico

Titolo: testo(50)

Collocazione: testo(10)

**Libri**

|  |  |  |
| --- | --- | --- |
| **CODICE\_LIBRO** | **TITOLO** | **COLLOCAZIONE** |
| 1 | LE RETI DI COMPUTER | 1/C/2 |
| 2 | IL MANUALE DI SQL | 3/H/8 |
| 3 | TECNICHE DI PROGRAMMAZIONE | 3/P/2 |
| 4 | C++ MANUALE DL RIFERIMENTO | 1/H/9 |

**Autori:**

Codice\_Autore: numerico

Cognome: testo(30)

Nome: testo(30)

**Autori**

|  |  |  |
| --- | --- | --- |
| **CODICE\_AUTORE** | **COGNOME** | **NOME** |
| 1 | PAOLO | RETI |
| 2 | GIOVANNI | MARIOTTI |
| 3 | MARIO | BIANCHI |

1.1 Esempio di dati contenuti in un database utilizzato in una biblioteca

* Un database deve rappresentare i diversi aspetti della realtà, e in particolare, oltre ai dati veri e propri, anche le relazioni tra i dati, ovvero le connessioni logiche presenti tra le varie categorie. Per esempio, deve essere rappresentata l’associazione che lega ciascun autore ai propri libri e viceversa.

Il database deve inoltre rispondere ai seguenti requisiti.

* I dati devono essere organizzati con ridondanza minima, ossia non essere inutilmente duplicati. Questa condizione deriva dalla esigenza di evitare non solo l’utilizzo non necessario di risorse di memorizzazione, ma anche e soprattutto l’onere di gestione di copie multiple; inoltre, se le informazioni relative a una categoria di dati vengono duplicate, si corre il rischio che un aggiornamento effettuato su una delle copie e non riportato sulle altre abbia conseguenze negative sulla consistenza e sull’affidabilità dell’insieme dei dati.
* I dati devono essere utilizzabili contemporaneamente da più utenti. Questa esigenza deriva dal punto precedente: è da evitare la situazione in cui ogni utente (o categoria di utenti) lavora su una propria copia dei dati, ed è necessario che esista un’unica versione dei dati, cui tutti gli utenti possano accedere; ciò implica la necessità che ciascun tipo di utenza possa avere una specifica visione dei dati e specifici diritti di accesso ai dati stessi. Inoltre, sono necessarie delle tecniche che consentano di evitare che l’attività dei vari utenti generi conflitti per l’uso contemporaneo degli stessi dati.
* I dati devono essere permanenti. Ciò implica non solo l’utilizzo di memorie di massa, ma anche l’applicazione di tecniche che preservino l’insieme dei dati in caso di malfunzionamento di un qualsiasi componente del sistema.

**1.2 Dagli archivi ai DBMS**

Il DataBase Management System (DBMS), o sistema per la gestione di basi di dati, è il componente del software di base che consente la gestione di uno o più database secondo i requisiti indicati nel paragrafo precedente e quindi un DBMS è un insieme di programmi il cui scopo è quello di gestire efficientemente grosse raccolte di informazioni (DATA BASE), garantendo un accesso semplice, rapidità nel fornire le risposte, sicurezza sia fisica (guasti, calamità naturali, azioni dolose) che logica (accesso a chi ne ha diritto). Il tutto consentendo anche un accesso concorrente da parte di più utenti; più sinteticamente il DBMS gestisce il DATA BASE e ne controlla l’accesso da parte di utenti e programmi applicativi.

I primi DBMS furono sviluppati alla fine degli anni Sessanta per i calcolatori di fascia alta (mainframe) di grandi organizzazioni. Precedentemente, l’approccio tradizionalmente applicato al problema dell’archiviazione dei dati era quello dell’utilizzo diretto delle strutture del file system (Figura 1.2.1). La caratteristica saliente che differenzia un sistema per la gestione di basi di dati rispetto all’approccio file system è la presenza di un componente specializzato in tale compito di gestione (Figura 1.2.2).

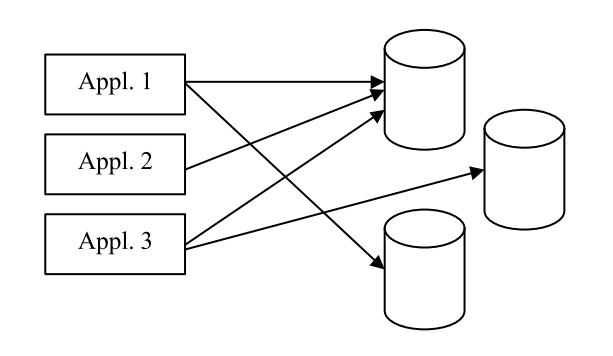


Fig. 1.2.1 Approccio file system

Nell’approccio file system le applicazioni Appl. 1, Appl. 2... accedono direttamente agli archivi dei dati. Nello sviluppo di tali applicazioni deve essere conosciuta la struttura interna degli archivi; inoltre sono a carico delle applicazioni stesse la rappresentazione delle relazioni tra i dati, le tecniche che permettono l’utilizzo contemporaneo degli archivi da parte di più applicazioni e che garantiscono la permanenza dei dati. Questi ultimi aspetti spesso vengono delegati a strati sottostanti di software non specializzato, quale il sistema operativo.

Schematizzando i maggiori problemi derivanti da questo approccio sono dovuti a:

### Inconsistenza e ridondanza dei dati

Vi possono essere differenze tra i valori relativi ad una stessa entità ma riportati in archivi diversi. La duplicazione di dati crea spreco di memoria

### Riservatezza dei dati

### Integrità dei dati

L'integrità dei dati viene assicurata dai 'vincoli di consistenza'. Ad esempio un campo non può assumere valore negativo. Con archivi separati, l'integrità dei dati viene affidata a programmi applicativi (soggetti ad errori)

### Concorrenza

Gestire gli accessi contemporanei alla stessa informazione.

### Dipendenza tra dati e codice

Cambia il codice ogni volta che si modificano i dati degli archivi.

Mentre i principali vantaggi sono:

* i requisiti hardware/software (microprocessore, RAM, spazio su hard disk) richiesti dai DBMS sono decisamente elevati: in più di una situazione non sono semplicemente disponibili e l’unica alternativa (spesso perfettamente adeguata) è rappresentata dalle tecniche ‘tradizionali’;
* certe elaborazioni sono molto difficili da realizzare con i linguaggi specifici per i data base: questi linguaggi ‘pagano’ la loro indubbia semplicità che li rende alla portata anche del non programmatore con una minore ‘potenza’; il problema è così sentito che in molti linguaggi per data base sono stati reintrodotti costrutti di programmazione tipici dei linguaggi tradizionali
* per certe operazioni la velocità delle tecniche tradizionali è impareggiabile: nessun DBMS supera in velocità linguaggi come il Pascal o il C per scrivere e leggere file di testo ( e non dimenticate che le pagine WEB in HTML sono file di testo …)

La Figura 1.2.2 mostra come nell’approccio DBMS le applicazioni rivolgano le proprie richieste di accesso al database (DB) al DBMS, il quale gestisce i dati, svincolando le applicazioni da questo onere

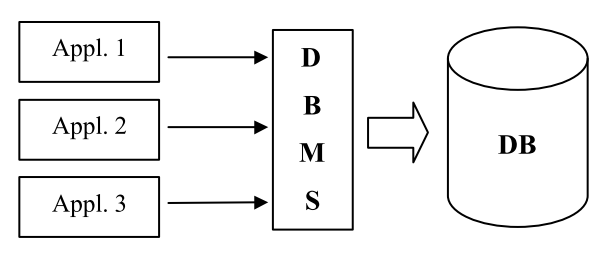


Fig. 1.2.2 Approccio DBMS

La transizione dall’approccio file system all’approccio DBMS deriva dall’evoluzione verso una visione modulare del software: un modulo - il DBMS – è specializzato nella gestione del database; tutti gli altri si rivolgono a questo per la gestione dei dati. Si ottiene così un duplice scopo:

* le funzionalità di gestione del database sono raggruppate in un unico insieme realizzato come componente autonomo, rendendo così più agevole lo sviluppo delle applicazioni;
* nessuna applicazione può effettuare operazioni scorrette sul database.

Sebbene i DBMS siano stati progettati originariamente per i grandi sistemi centralizzati, sono oggi disponibili su tutte le principali piattaforme e architetture, quali personal computer e workstation, sia standalone sia interconnessi in reti di diversa estensione, da quelle locali a Internet

**1.3 Vantaggi offerti dai DBMS**

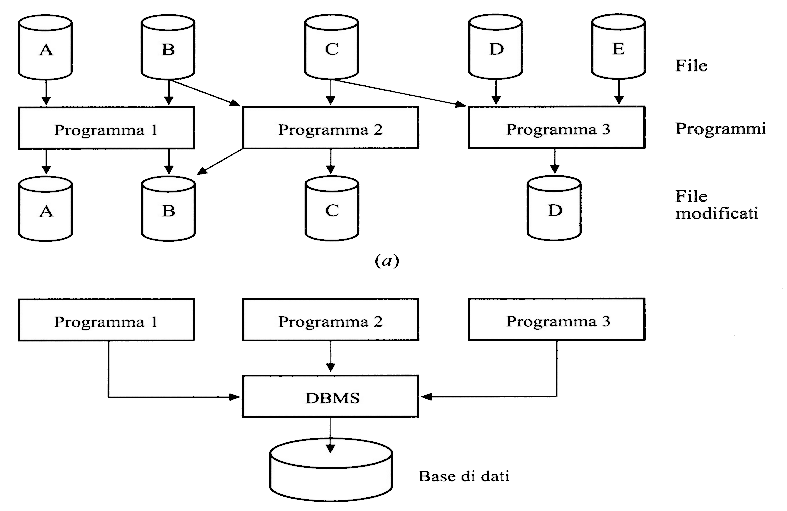
Più in dettaglio, un DBMS :

1. Garantisce l’indipendenza logica: la possibilità di cambiare la struttura logica del data base (aggiungere una tabella, aggiungere/togliere un campo ad una tabella ecc.) senza che si sia costretti a modificare le applicazioni già scritte (se non usavano la tabella/il campo eliminato e non devono usare la tabella/il campo aggiunto, ovviamente). Questo perché il DBMS fornisce ad ogni applicazione una ‘vista’ personalizzata delle informazioni (cioè, banalizzando, la lista dei campi usati). Se cambia la struttura logica ma la vista non ne è influenzata, le applicazioni non si ‘accorgono’ della modifica.  
   Pensate invece, ad esempio, ai file tipizzati del Pascal: se si deve modificare la struttura del suo record sarà necessario modificare TUTTI i programmi che usavano la vecchia struttura.
2. Garantisce indipendenza fisica: se il modo con cui il DBMS organizza i dati sul disco cambia (ad esempio migliora il sistema di indicizzazione) la struttura logica dei dati non cambia e quindi neppure le applicazioni necessitano di modifica.
3. Assicura un utilizzo più flessibile degli archivi : con i vecchi metodi, una volta fatte, alcune scelte non potevano più essere cambiata se non a prezzo di modifiche dei programmi. Con un DBMS, operazioni come il passare da un elenco mantenuto in ordine alfabetico ad uno in ordine di data è invece una questione da pochi secondi.
4. Elimina o comunque usa in modo perfettamente controllato la duplicazione dei dati: lo schema unitario del DATA BASE rende possibile inserire una volta sola le informazioni ed usarle in tanti modi diversi (lo stesso indirizzo potrà essere usato da tutti i reparti, ad esempio).
5. Elimina, grazie a quanto affermato nel punto precedente, l’inconsistenza e l’incoerenza delle informazioni: tutti i dati sono tenuti sincronizzati e non è possibile eliminarne uno che è in relazione con qualcun altro. Ecco due esempi :
6. Se cambia un numero di fattura, esso verrà modificato automaticamente in tutti i punti in cui è usato ;
7. Non è possibile cancellare dagli archivi un cliente se esistono ancora delle sue fatture memorizzate (renderebbe le fatture inconsistenti, cioè non utilizzabili, in quanto farebbero riferimento ad un cliente di cui non si possiedono più, ad esempio, i dati fiscali).
8. Utilizza in modo efficiente lo spazio a disposizione sui supporti di memorizzazione di massa: sofisticate tecniche di compressione dei dati ottimizzano lo spazio usato. Pensate a quante persone si chiamano ‘Alessandro’ o ‘Alessandra’ : invece di ripetere tutte le volte questi caratteri è possibile usare un codice numerico, che si sa corrispondere a quei nomi, usando meno spazio. Naturalmente quando poi si vorrà passare dal codice numerico al nome, sarà necessario consultare (lo farà il computer) una tabella dove, per ogni codice, è indicato il nome che corrisponde. Questa è una semplificazione ma penso sia sufficiente a rendere l’idea che sta dietro alle tecniche di compressione. I testi mediamente di riducono del 50%. Le immagini anche del 60% o più.
9. Permette un accesso sicuro, anche a più utenti o applicazioni contemporaneamente, alle informazioni. Innanzitutto è previsto un meccanismo di identificazione (chi sei ?) e di autenticazione (dimostramelo : inserisci la tua ‘password’, cioè il codice di accesso) : si parla di fase di ‘login’ (ingresso nel sistema). Ogni utente ha l’autorizzazione, grazie al tipo di login che riesce a superare, a portare a termine solo un ben determinato tipo di operazioni (chiaro che un impiegato allo sportello non potrà accedere ad informazioni riservate ai manager o cancellare quello che vuole). Ogni utente ha, per così dire, una sua vista dei dati.
10. Gestisce automaticamente ed efficientemente le connessioni in rete (LAN o Internet): il software del DBMS dovrà interagire al meglio con i moduli del sistema operativo preposti alla gestione della rete ; sarà così possibile interrogare il DBMS anche da stazioni remote. Una tipologia di utilizzo ormai molto usata è quella chiamata elaborazione Client - Server. Sulle stazioni di lavoro sono usati programmi (detti di tipo ‘client’) di interrogazione degli archivi ; i client, in base a ciò che ha impostato l’utente, inviano sulla rete la richiesta di informazioni ; questa perviene alla macchina collegata in rete su cui risiedono gli archivi (e di solito anche il DBMS) ; qui un programma (indicato come applicazione ‘server’) elabora gli archivi ed estrae le informazioni ; solo i risultati della richiesta sono inviati al ‘client’. Una volta era l’applicazione client che elaborava direttamente gli archivi, con una quantità di dati che transitava sulla rete molto maggiore, con conseguente sovraccarico e rallentamento generale delle prestazioni.
11. Garantisce la sicurezza fisica dei dati. Si serve di meccanismi hardware e software per assicurare la sicurezza fisica delle informazioni : si vogliono evitare perdite di dati a causa di rotture accidentali delle apparecchiature o dovute a fenomeni catastrofici (inondazioni, incendi, terremoti ecc.) o dolose (attentati terroristici, azioni di squilibrati, di dipendenti vendicativi, di dipendenti corrotti ecc.). Naturalmente tutte le contromisure possono essere presenti anche senza DBMS : diciamo che il DBMS può essere predisposto per sfruttare al meglio queste linee difensive e che le stesse sono realizzate sulle macchine più importanti, di solito sono proprio quelle su cui risiede il DBMS ed i preziosissimi archivi aziendali. Vediamo qualche esempio di queste misure di sicurezza:
12. (esempio di soluzione hardware) Usare in parallelo più dischi fissi (se se ne rompe uno c’è sempre l’altro); si parla di mirroring (raddoppio speculare, da ‘mirror’, specchio); se viene duplicata anche l’elettronica di controllo dei dischi (controller) si parla di duplexing. In realtà questa linea difensiva, se presente, è automatica e non controllabile dal DBMS. Quest’ultimo, però, potrebbe decidere di non mettere in atto altre tecniche di protezione (backup automatico) che appesantirebbero il suo lavoro, visto l’alto grado di protezione già garantito da mirroring e duplexing. Oggi esistono alcuni dischi fissi con software integrato che controlla periodicamente lo ‘stato di salute’ del disco (S.M.A.R.T.) e, in base a sofisticati calcoli statistici, è in grado di predire quanto sta per avvenire una rottura dello stesso, dando il tempo di prendere le necessarie contromisure.  
    Il mirroring/duplexing è solo uno dei livelli (livello 1) di una tecnologia più ampia chiamata RAID (Redundant Array of Inexpensive Disks: set ridondante di dischi economici) e massimizza la sicurezza. Per massimizzare invece le prestazioni esiste lo *striping****.*** Il file viene suddiviso e distribuito su più dischi che vengono letti/scritti in parallelo (se però un disco si rompe il file risulta irrimediabilmente privo di una porzione). Un compromesso tra sicurezza e velocità è dato invece dal livello 5 del raid: un disco è dedicato ad una porzione di bit di parità (oppure i dati di parità sono distribuiti tra i dischi) che consente di ricostruire la parte di file mancante in caso di malfunzionamento di uno dei dischi.
13. (esempio di soluzione hardware) L’uso di più microprocessori sullo stesso sistema di elaborazione, oltre ad assicurare migliori prestazioni può mettere al riparo da avarie di uno di essi (gli altri possono continuare a lavorare). Anche questa caratteristica, se presente, non è controllabile dal DBMS, ma è automatica.
14. (esempio di soluzione hardware) Un gruppo di continuità è in pratica un generatore di energia elettrica che usa una batteria che normalmente è alimentata dalla rete elettrica. Se viene a mancare l’alimentazione la batteria è in grado di fornire energia per alcune decine di minuti al sistema di elaborazione : il tempo necessario a registrare i dati in sospeso ed a portare il sistema in uno stato sicuro per il suo spegnimento. E’ anche possibile avvalersi di veri e propri sistemi di alimentazione a lunga autonomia che evitano qualsiasi periodo, anche lungo, di interruzione delle attività.
15. (esempio di soluzione software) Una delle difese più efficaci contro ogni accidente è anche la più semplice : fare delle copie di sicurezza dei dati aziendali, il più frequentemente possibile (operazione di backup). Si va dal semplice ‘copy’ degli archivi sui floppy disk (scomodo, lento ed insicuro), all’uso dei nastri magnetici o supporti ottici. I DBMS possono offrire sofisticati programmi per la gestione automatica e periodica di queste operazioni. Non dimentichiamoci che in commercio esistono ottimi programmi per il backup da usarsi anche senza un DBMS. Se è disponibile un collegamento in rete una soluzione contro le catastrofi è inviare le copie di sicurezza in posti lontani : un incendio potrà anche distruggere i locali di una filiale, ma nulla potrà contro le copie di sicurezza inviate, prudentemente, alla sede centrale o ad un’altra filiale ! Naturalmente è sempre possibile prevedere anche misure fisiche specifiche: antincendio, contro le inondazioni ecc.
16. (esempio di soluzione software) Oltre ai controlli sull’identità imposti dal sistema operativo, il DBMS potrebbe adottare altre tecniche per assicurare che le informazioni non cadano in mani sbagliate : la crittografia le può rendere incomprensibili ad occhi indesiderati (le posso rubare ma non le posso capire !).
17. (esempio di soluzione software) Un moderno DBMS può lavorare con una logica degli aggiornamenti sui dischi basata sulle cosiddette transazioni. Una transazione è l’insieme di tutte le modifiche che il DBMS deve apportare al Data Base per soddisfare una richiesta di aggiornamento. Ad esempio, se un operatore comanda la cancellazione di un cliente, se confermata, l’operazione deve cancellare, oltre ai dati del cliente, anche tutte le sue fatture ed altri dati correlati. Se il DBMS cominciasse senza altre precauzioni ad apportare questa sequenza di modifiche e capitasse un inconveniente, con conseguente blocco del sistema, gli archivi rimarrebbero in una situazione inconsistente : potrebbero, ad esempio, esserci delle fatture dopo che il cliente è già stato cancellato, perché il sistema è andato in avaria prima del termine delle cancellazioni. A peggiorare la situazione, chi ha comandato l’operazione potrebbe anche non sapere che non è andata a buon fine : come ripristinare gli archivi ? In una logica che usa le transazioni, le operazioni da apportare sono prima descritte (non eseguite) registrandole su un archivio : questo contiene allora le istruzioni da seguire per soddisfare la richiesta dell’utente (cancella il ciente X, poi la fattura 121, poi la fattura 127 ecc.). In seguito il DBMS, leggendo le istruzioni da questo archivio, inizia ad eseguirle, tenendone traccia in un altro file (chiamato file di log, ‘traccia’). Se per qualche motivo il sistema si blocca, alla successiva ripartenza il DBMS va ad esaminare eventuali file di log : dall’esame di questi capisce quali transazioni non sono state completate. Allora può, sempre grazie al file di log, risalire alla situazione prima dell’inizio della transazione (roll back=tornare indietro) : a questo punto tenta di nuovo di eseguire la transazione. E’ un po’ come se, prima di apportare le modifiche (transazione) ad una lettera usando un programma di videoscrittura, facessimo un elenco delle variazioni che faremo. Poi su un foglio a parte (log) tenessimo traccia delle modifiche via via apportate : in un qualunque momento potremmo tornare al documento originale ripercorrendo al contrario (roll back) le variazioni.
18. Garantisce la sicurezza logica dei dati: significa garantire l’accesso solo a chi ne ha diritto. Il meccanismo di base prevedete l’autenticazione degli utenti (username/password, impronta digitale o retinica, riconoscimento vocale o dei visi ecc.). Ogni utente ha quindi una sua vista ‘confezionata’ dal DBMS in base ai diritti dell’utente stesso. La cifratura (trasformazione tramite sofisticati algoritmi che rendono incomprensibili i dati senza una ‘chiave’, cioè una sequenza di bit, necessaria a decifrarli) protegge i dati da eventuali furti.
19. E’ in grado di scambiare facilmente ed efficientemente informazioni con altri DBMS: è infatti indispensabile poter colloquiare con altri tipi di DBMS che potrebbero essere in uso in altri reparti, in altre ditte con cui si collabora, con enti statali ecc. L’esperanto (lingua da tutti riconosciuta) dei dbms è SQL, Structured Query Language (linguaggio strutturato per le interrogazioni, query in inglese) : è un linguaggio progettato per rendere semplici ed allo stesso tempo potenti le interrogazioni inviate ad un DBMS. Molti DBMS non costringono a scrivere le interrogazioni in SQL (è pur sempre un’operazione simile alla difficile arte della programmazione ...) ma offrono strumenti visuali per specificare una interrogazione : le istruzioni SQL corrispondenti saranno generate automaticamente (è un po’ come se, per scrivere un racconto, potessimo scegliere da elenchi, argomenti e frasi già pronte, ed il tema fosse poi composto automaticamente).

# Database management system

### Da Wikipedia, l'enciclopedia libera.

In [informatica](http://it.wikipedia.org/wiki/Informatica), un **Database Management System** (abbreviato in **DBMS**) è un [sistema software](http://it.wikipedia.org/w/index.php?title=Sistema_software&action=edit&redlink=1) progettato per consentire la creazione e manipolazione efficiente di [database](http://it.wikipedia.org/wiki/Database) (ovvero di collezioni di [dati](http://it.wikipedia.org/wiki/Dato) strutturati) solitamente da parte di più utenti. I DBMS svolgono un ruolo fondamentale in numerose applicazioni informatiche, dalla [contabilità](http://it.wikipedia.org/wiki/Contabilit%C3%A0), alla gestione delle [risorse umane](http://it.wikipedia.org/wiki/Risorse_umane_%28economia%29) e alla [finanza](http://it.wikipedia.org/wiki/Finanza) fino a contesti tecnici come la [gestione di rete](http://it.wikipedia.org/wiki/Gestione_di_rete) o la [telefonia](http://it.wikipedia.org/wiki/Telefonia).  
Se in passato i DBMS erano diffusi principalmente presso le grandi [aziende](http://it.wikipedia.org/wiki/Azienda) e istituzioni (che potevano permettersi l'impegno economico derivante dall'acquisto delle grandi infrastrutture [hardware](http://it.wikipedia.org/wiki/Hardware) necessarie per realizzare un sistema di database efficiente), oggi il loro utilizzo è diffuso praticamente in ogni contesto. L'espressione [applicazione enterprise](http://it.wikipedia.org/w/index.php?title=Applicazione_enterprise&action=edit&redlink=1), che nel gergo informatico si riferisce ad applicazioni legate al *business* delle aziende che le utilizzano, implica quasi "per definizione" la presenza di una o più basi di dati amministrate da uno o più DBMS.



La teoria dei database, e dei DBMS, rappresenta da sempre uno dei filoni più solidi e importanti dell'[informatica](http://it.wikipedia.org/wiki/Informatica).

Un DBMS è differente dal concetto generale di [applicazione](http://it.wikipedia.org/wiki/Applicazione) sulle [banche dati](http://it.wikipedia.org/wiki/Banche_dati), in quanto è progettato per sistemi multi-utente. A tale scopo, i DBMS si appoggiano a [kernel](http://it.wikipedia.org/wiki/Kernel) che supportano nativamente il [multitasking](http://it.wikipedia.org/wiki/Multitasking) e il collegamento in [rete](http://it.wikipedia.org/wiki/Reti_di_calcolatori). Una tipica applicazione per la gestione dei database non includerebbe, infatti, tali funzionalità, ma si appoggerebbe al [sistema operativo](http://it.wikipedia.org/wiki/Sistema_operativo) per consentire all'utente di fruirne dei vantaggi.

## DBMS

* [1 Descrizione](http://it.wikipedia.org/wiki/DBMS#Descrizione)
  + [1.1 Autorizzazioni](http://it.wikipedia.org/wiki/DBMS#Autorizzazioni)
  + [1.2 Integrità](http://it.wikipedia.org/wiki/DBMS#Integrit.C3.A0)
  + [1.3 Interrogazioni e modifiche](http://it.wikipedia.org/wiki/DBMS#Interrogazioni_e_modifiche)
  + [1.4 DBMS e sistemi informativi](http://it.wikipedia.org/wiki/DBMS#DBMS_e_sistemi_informativi)
  + [1.5 Architettura e organizzazione](http://it.wikipedia.org/wiki/DBMS#Architettura_e_organizzazione)
* [2 Storia](http://it.wikipedia.org/wiki/DBMS#Storia)
  + [2.1 Database Navigazionali](http://it.wikipedia.org/wiki/DBMS#Database_Navigazionali)
  + [2.2 Database Relazionali](http://it.wikipedia.org/wiki/DBMS#Database_Relazionali)
* [3 Architettura di un DBMS](http://it.wikipedia.org/wiki/DBMS#Architettura_di_un_DBMS)

## 1. Descrizione

Un DBMS può essere costituito da un insieme assai complesso di programmi [software](http://it.wikipedia.org/wiki/Software) che controllano l'organizzazione, la memorizzazione e il reperimento dei dati ([campi](http://it.wikipedia.org/wiki/Campo_%28informatica%29), [record](http://it.wikipedia.org/wiki/Record_%28informatica%29) e archivi) in un database. Un DBMS controlla anche la sicurezza e l'integrità del database. Il DBMS accetta richieste di dati da parte del programma applicativo e "istruisce" il [sistema operativo](http://it.wikipedia.org/wiki/Sistema_operativo) per il trasferimento dei dati appropriati.

### 1.1 Autorizzazioni

Il sistema di sicurezza dei dati impedisce agli utenti non autorizzati di visualizzare o aggiornare il database. Mediante l'uso di *password* (parole d'ordine) agli utenti è permesso l'accesso all'intero database o ad un suo sottoinsieme: in questo secondo caso si parla di *subschema*. Per esempio, un database di impiegati può contenere tutti i dati riguardanti un singolo soggetto, ma un gruppo di utenti può essere autorizzato a vedere solamente i dati riguardanti lo stipendio, mentre altri utenti possono essere autorizzati a vedere solamente le informazioni che riguardano la sua storia lavorativa e la situazione sanitaria.

### 1.2 Integrità

Il DBMS può mantenere l'integrità del database non consentendo a più utenti di modificare lo stesso record contemporaneamente (blocco del record). Il database può impedire l'immissione di due record duplicati; per esempio può essere impedita l'immissione nel database di due clienti con lo stesso numero identificativo (*campi chiave*). L'insieme di regole che determinano l'integrità e la consistenza di una [base di dati](http://it.wikipedia.org/wiki/Base_di_dati) prendono il nome di [Vincoli di integrità](http://it.wikipedia.org/wiki/Vincoli_di_integrit%C3%A0) referenziale. A tale proposito si vedano le cosiddette proprietà "[ACID](http://it.wikipedia.org/wiki/ACID)".

### 1.3 Interrogazioni e modifiche

I linguaggi di interrogazione del database mediante *query* (interrogazioni) e i generatori di *report* permettono agli utenti di interrogare in maniera interattiva il database e di analizzarne i dati.

Se il DBMS fornisce un modo per aggiornare ed immettere nuovi dati nel database, oltre che per interrogarlo, questa capacità permette di gestire database personali. Comunque queste funzionalità non danno la possibilità di mantenere traccia delle revisioni e non forniscono gli strumenti necessari alla gestione di una organizzazione multi-utente. Questi controlli sono disponibili solamente quando un insieme di programmi applicativi sono appositamente costruiti per gestire e coordinare ciascuna funzione di immissione o modifica dei dati.

### 1.4 DBMS e sistemi informativi

Un sistema informativo commerciale è costituito da soggetti (clienti, impiegati, venditori) e attività (ordini, pagamenti, acquisti, ecc.). La progettazione del [database](http://it.wikipedia.org/wiki/Database) ([*database design*](http://it.wikipedia.org/w/index.php?title=Database_design&action=edit&redlink=1)) è il processo decisionale su come organizzare questi dati in tipi di record e su come ciascun tipo di record si relaziona con gli altri. Il DBMS dovrebbe rispecchiare la struttura dei dati dell'organizzazione e gestire in maniera efficiente le varie transazioni.

Quando si usa un DBMS i sistemi informativi possono essere adeguati molto facilmente al cambiamento delle richieste informative dell'organizzazione. Possono essere aggiunte al database nuove categorie di dati senza dover stravolgere il sistema esistente.

### 1.5 Architettura e organizzazione

Le organizzazioni possono usare un DBMS per gestire il normale processo quotidiano delle transazioni e in un secondo tempo spostare il dettaglio in un altro computer che usa un altro DBMS più adatto per gestire interrogazioni casuali e l'attività di analisi. Le decisioni globali circa l'architettura dei sistemi informativi, sono gestite dagli analisti di sistema e dagli amministratori dei dati. La progettazione di dettaglio del database è demandata agli amministratori del database stesso.

I tre tipi di organizzazione più comuni sono il [modello gerarchico](http://it.wikipedia.org/wiki/Modello_gerarchico), il [modello reticolare](http://it.wikipedia.org/wiki/Modello_reticolare) e il [modello relazionale](http://it.wikipedia.org/wiki/Modello_relazionale): il modello dominante oggi è quello relazionale, normalmente utilizzato con il linguaggio di interrogazione [SQL](http://it.wikipedia.org/wiki/SQL). Molti DBMS supportano le API ([Application programming interface](http://it.wikipedia.org/wiki/Application_programming_interface)) dell'[Open Database Connectivity](http://it.wikipedia.org/wiki/ODBC) (ODBC) o [Java Database Connectivity](http://it.wikipedia.org/wiki/JDBC) (JDBC, lo standard per [Java](http://it.wikipedia.org/wiki/Java_%28linguaggio%29)), che forniscono ai programmatori strumenti standardizzati per l'accesso ai database.

I *database server* sono computer ottimizzati per ospitare i programmi che costituiscono il database reale e sui quali girano solo il DBMS e il software ad esso correlato (nelle situazioni reali spesso questi computer svolgono anche altre funzioni non correlate con la gestione del database). Di solito si tratta di macchine multiprocessore e con dischi fissi configurati in modalità [RAID](http://it.wikipedia.org/wiki/RAID) per una memorizzazione stabile ed affidabile dei dati che garantisca la continuità del servizio anche in caso di guasto ad un componente (sistemi *fault tolerant*). In ambienti dove vengono processate transazioni con moli di dati particolarmente elevate vengono utilizzati anche componenti *hardware* che hanno la funzione specifica di acceleratori di database e che sono collegati ad uno o più server attraverso canali preferenziali ad alta velocità.

Sempre più frequentemente si assiste alla integrazione delle [basi di dati](http://it.wikipedia.org/wiki/Basi_di_dati) e di [Internet](http://it.wikipedia.org/wiki/Internet): una vasta classe di applicazioni della rete fa uso di informazioni presenti su basi di dati; esempi di questo tipo di applicazioni vanno dai cataloghi delle imprese, disponibili per il pubblico, alle edizioni *on-line* dei giornali e dei quotidiani. Per garantire un linguaggio di modellizzazione che consenta di passare dalla visualizzazione dei dati in un formato compatibile con le basi di dati, ad una "vista" concettuale del futuro sito web esiste un linguaggio specifico chiamato [WebML](http://it.wikipedia.org/wiki/WebML).

## 2. Storia

I [database](http://it.wikipedia.org/wiki/Database) sono stati utilizzati fin dall'inizio della storia dell'[informatica](http://it.wikipedia.org/wiki/Informatica), ma la grande maggioranza di questi erano programmi specializzati per l'accesso ad un singolo database. Oggi, invece, i moderni sistemi possono essere utilizzati per compiere operazioni su un gran numero di basi di dati differenti. Questa "specializzazione" era dovuta alla necessità di guadagnare in velocità di esecuzione pur perdendo in flessibilità.

### 2.1 Database Navigazionali

Con la crescita della capacità elaborativa dei calcolatori questo contrasto con la flessibilità andò attenuandosi, con la creazione negli [anni sessanta](http://it.wikipedia.org/wiki/Anni_1960) di una serie di database utilizzabili per diverse applicazioni. L'interesse nel fissare uno standard crebbe, e [Charles W. Bachman](http://it.wikipedia.org/w/index.php?title=Charles_W._Bachman&action=edit&redlink=1), creatore di uno di questi prodotti, (**"IDS"**), fondò il **"Database Task Group"**, all'interno del **"Codasyl"**, il gruppo di lavoro dedicato alla creazione e standardizzazione del [linguaggio di programmazione](http://it.wikipedia.org/wiki/Linguaggio_di_programmazione) [COBOL](http://it.wikipedia.org/wiki/COBOL). Nel [1971](http://it.wikipedia.org/wiki/1971) tale standard fu prodotto e prese il nome di **"Approccio Codasyl"**; presto furono disponibili sul [mercato](http://it.wikipedia.org/wiki/Mercato) una serie di prodotti basati su tale approccio.

Questo approccio era basato sulla navigazione manuale in un insieme di dati disposti sotto forma di rete. Alla prima apertura del programma, il programma si apriva sul primo dato disponibile, contenente, tra le altre cose, un [puntatore](http://it.wikipedia.org/w/index.php?title=Puntatore_%28input/output%29&action=edit&redlink=1) ai dati successivi. Per trovare un dato il programma attraversava la serie di puntatori fino a trovare il dato corretto. Delle semplici [query](http://it.wikipedia.org/wiki/Query) come "Trova tutte le persone nate in Svezia" richiedevano l'attraversamento dell'intero set di dati.  
Non esisteva, dunque, alcuna funzione di ricerca; oggi, questo potrebbe sembrare una limitazione, ma all'epoca, essendo i dati archiviati su [nastro magnetico](http://it.wikipedia.org/wiki/Nastro_magnetico), operazioni come quelle evidenziate sopra non erano particolarmente costose in termini di tempo.

Nel [1968](http://it.wikipedia.org/wiki/1968), la [IBM](http://it.wikipedia.org/wiki/IBM) sviluppò un proprio sistema [DBMS](http://it.wikipedia.org/wiki/DBMS), chiamato [**IMS**](http://it.wikipedia.org/wiki/IMS). IMS era uno sviluppo di un programma utilizzato nelle [missioni Apollo](http://it.wikipedia.org/wiki/Programma_Apollo) sui Sistemi /360 e utilizzava un sistema simile all'approccio *Codasyl*, con l'unica differenza di avere un sistema gerarchico anziché a rete.

Ambedue le soluzioni presero poi il nome di "database navigazionali" a causa del metodo di consultazione che era stato previsto. Inoltre, [Charles Bachman](http://it.wikipedia.org/w/index.php?title=Charles_Bachman&action=edit&redlink=1), in occasione della premiazione nel [1973](http://it.wikipedia.org/wiki/1973) in cui gli venne conferito il [Premio Turing](http://it.wikipedia.org/wiki/Premio_Turing), presentò un lavoro intitolato "Il programmatore come navigatore". IMS è abitualmente classificato come un database [gerarchico](http://it.wikipedia.org/wiki/Modello_gerarchico), mentre IDS e IDMS (ambedue database *CODASYL*), CINCOMs e [TOTAL](http://it.wikipedia.org/w/index.php?title=TOTAL&action=edit&redlink=1) sono classificati come database a rete (o [reticolari](http://it.wikipedia.org/wiki/Modello_reticolare)).

### 2.2 Database Relazionali

I DBMS relazionali sono detti anche RDBMS (Relational DBMS).

[Edgar F. Codd](http://it.wikipedia.org/wiki/Edgar_F._Codd) lavorava alla sede [californiana](http://it.wikipedia.org/wiki/California) della [IBM](http://it.wikipedia.org/wiki/IBM) come ricercatore sulla nascente tecnologia degli [hard disk](http://it.wikipedia.org/wiki/Hard_disk) quando osservò l'inefficienza dell'approccio *Codasyl* con la nuova modalità di memorizzazione dei dati, inefficienza principalmente dovuta all'assenza di una funzione di ricerca. Nel [1970](http://it.wikipedia.org/wiki/1970) cominciò a produrre diversi documenti schematizzanti un nuovo approccio alla costruzione delle basi di dati, culminati nel "[Modello relazionale](http://it.wikipedia.org/wiki/Modello_relazionale) per Basi di dati condivise" (**"A Relational Model of Data for Large Shared Data Banks"**).

In questo articolo, descrisse un nuovo sistema per archiviare e modificare grandi quantità di dati. Invece di utilizzare delle "righe" (in [inglese](http://it.wikipedia.org/wiki/Lingua_inglese), ma anche molto usato in [italiano](http://it.wikipedia.org/wiki/Lingua_italiana): *"*[*record*](http://it.wikipedia.org/wiki/Record)*"* o anche "[tuple](http://it.wikipedia.org/wiki/Tupla)") collegate tra di loro attraverso un qualche tipo di struttura "ad albero", come in *Codasyl*, ritenne di utilizzare una "tabella" di righe a lunghezza fissa. Questo sistema sarebbe stato molto inefficiente nell'archiviazione di dati "sparsi", in cui la tabella avrebbe potuto avere diverse "celle" vuote; tale errore di impostazione fu corretto dividendo i dati in diverse tabelle, in cui gli elementi opzionali venivano spostati, anziché sprecare spazio nella tabella principale.

Ad esempio, un utilizzo comune delle [basi di dati](http://it.wikipedia.org/wiki/Database) è quello di registrare delle informazioni sugli utenti: il loro nome, informazioni di accesso, indirizzo e numeri di telefono. In un database navigazionale tutti questi dati sarebbero stati memorizzati in un unico *"*[*record*](http://it.wikipedia.org/wiki/Record)*"*, e gli elementi non presenti (ad esempio un utente di cui non sia noto l'indirizzo) sarebbero stati semplicemente omessi. Al contrario, in un database relazionale, le informazioni vengono divise, ad esempio, nelle tabelle "utente", "indirizzi", "numeri di telefono" e solo se i dati sono presenti viene creata, nella rispettiva tabella, una [*tupla*](http://it.wikipedia.org/wiki/Tupla).

Uno degli aspetti interessanti introdotti nei database relazionali sta nel collegamento delle tabelle: nel modello relazionale, per ogni *"*[*record*](http://it.wikipedia.org/wiki/Record)*"* viene definita una "chiave", ovvero un identificatore univoco della *tupla*. Nella ricostruzione delle relazioni, l'elemento di riferimento, che distingue una riga da un'altra è proprio questa "chiave" e viene richiamata nella definizione della relazione. La chiave può essere uno dei dati stessi che vengono memorizzati (ad esempio, per la tabella utenti, il "Codice Fiscale" della persona), o un campo che viene aggiunto specificatamente per questo scopo (spesso chiamato *"OID"* - *"Object IDentifier"*), o una combinazione di più campi (chiave composta).

Questa operazione di "riunificazione" dei dati non è prevista nei linguaggi di programmazione tradizionali: mentre l'approccio navigazionale richiede semplicemente di *"ciclare"* per raccogliere i diversi *"*[*record*](http://it.wikipedia.org/wiki/Record)*"*, l'approccio relazionale richiede al programma di "ciclare" per raccogliere le informazioni riguardanti ogni record. Codd, propose, come soluzione, la creazione di un linguaggio dedicato a questo problema. Tale linguaggio, più tardi, si è sviluppato nella codifica che oggi è universalmente adottata e che è il mattone fondamentale delle basi di dati: [**SQL**](http://it.wikipedia.org/wiki/SQL).

Utilizzando una branca della [matematica](http://it.wikipedia.org/wiki/Matematica) chiamata "calcolo delle [*tuple*](http://it.wikipedia.org/wiki/Tupla)", dimostrò che questo sistema era in grado di compiere tutte le normali operazioni di amministrazione dei database (inserimento, cancellazione, etc.) e che inoltre consentiva di disporre di uno strumento semplice per trovare e visualizzare gruppi di dati tramite un'unica operazione.

LA [IBM](http://it.wikipedia.org/wiki/IBM) cominciò a implementare questa teoria in alcuni prototipi all'inizio degli [anni settanta](http://it.wikipedia.org/wiki/Anni_1970), come nel *"System R"*. La prima versione fu realizzata nel [1974](http://it.wikipedia.org/wiki/1974)/[75](http://it.wikipedia.org/wiki/1975) con uno strumento "monotabella"; negli anni successivi furono studiati i primi sistemi che potessero supportare la suddivisione dei dati in tabelle separate, utile, come abbiamo visto, per la separazione dei dati opzionali in tabelle diverse da quella principale. Versioni "multiutente" furono realizzate nel [1978](http://it.wikipedia.org/wiki/1978) e nel [1979](http://it.wikipedia.org/wiki/1979); negli stessi anni fu standardizzato il linguaggio [SQL](http://it.wikipedia.org/wiki/SQL). La superiorità di questo sistema rispetto a *Codasyl* fu quindi evidente e la IBM passò a sviluppare una versione commerciale di "System R", che prese il nome di "SQL/DS" prima e di "Database 2" (DB2) infine.

Il lavoro di Codd venne proseguito presso l'[Università di Berkeley](http://it.wikipedia.org/wiki/Universit%C3%A0_di_Berkeley) da [Eugene Wong](http://it.wikipedia.org/w/index.php?title=Eugene_Wong&action=edit&redlink=1) e [Michael Stonebraker](http://it.wikipedia.org/w/index.php?title=Michael_Stonebraker&action=edit&redlink=1). Il loro progetto, chiamato [INGRES](http://it.wikipedia.org/w/index.php?title=INGRES&action=edit&redlink=1) e finanziato con fondi destinati alla creazione di un database [geografico](http://it.wikipedia.org/wiki/Geografia), vide la luce nel [1973](http://it.wikipedia.org/wiki/1973) e produsse i primi risultati nel [1974](http://it.wikipedia.org/wiki/1974) anche grazie all'opera di numerosi [studenti](http://it.wikipedia.org/wiki/Studente) che si prestarono quali programmatori (quasi 30 persone lavorarono al progetto). [INGRES](http://it.wikipedia.org/w/index.php?title=INGRES&action=edit&redlink=1) era assai simile a "System R" e prevedeva un linguaggio alternativo a [SQL](http://it.wikipedia.org/wiki/SQL), chiamato [QUEL](http://it.wikipedia.org/w/index.php?title=QUEL&action=edit&redlink=1).

Molte delle persone coinvolte nel progetto si convinsero della fattibilità commerciale dello stesso e fondarono imprese per entrare nel mercato con questo prodotto. Sybase, Informix, NonStop SQL e alla fine Ingres stessa nacquero quali *"spin-off"* per la diffusione di [INGRES](http://it.wikipedia.org/w/index.php?title=INGRES&action=edit&redlink=1) all'inizio degli [anni ottanta](http://it.wikipedia.org/wiki/Anni_1980). Perfino [Microsoft](http://it.wikipedia.org/wiki/Microsoft) SQL Server è, per certi versi, una derivazione di "Sybase" e, quindi, di [INGRES](http://it.wikipedia.org/w/index.php?title=INGRES&action=edit&redlink=1). Solamente la [Oracle](http://it.wikipedia.org/wiki/Oracle) di [Larry Ellison](http://it.wikipedia.org/wiki/Larry_Ellison) partì utilizzando un approccio diverso, basato sul "System R" della IBM, e alla fine prevalse sulle altre compagnie con il suo prodotto, lanciato nel [1978](http://it.wikipedia.org/wiki/1978). In [Svezia](http://it.wikipedia.org/wiki/Svezia) il lavoro di Codd venne sviluppato nella [Università](http://it.wikipedia.org/wiki/Universit%C3%A0) di [Uppsala](http://it.wikipedia.org/wiki/Uppsala) che sviluppò un diverso prodotto, "[Mimer SQL](http://it.wikipedia.org/w/index.php?title=Mimer_SQL&action=edit&redlink=1)", commercializzato nel [1984](http://it.wikipedia.org/wiki/1984). Una particolarità di questa soluzione sta nell'introduzione del concetto di [transazione](http://it.wikipedia.org/wiki/Transazione), successivamente importata in quasi tutti i DBMS.

## 3. Architettura di un DBMS

Un DBMS è uno strumento per la creazione e la gestione efficiente di grandi quantità di dati che consente di conservarli in modo sicuro per lunghi periodi di tempo. Un DBMS fornisce agli utenti questi servizi:

* **Persistent storage**: come un file system, un DBMS permette la memorizzazione di grandi quantità di dati, ma garantisce una flessibilità molto più elevata
* **Programming interface**: permette agli utenti di accedere e modificare i dati attraverso un potente linguaggio di interrogazione
* **Transaction management**: supporta l'accesso concorrente ai dati evitando conseguenze indesiderate dovute a crash del sistema o dell'applicazione

Si considerano due diversi tipi di utenti:

* utenti convenzionali/applicazioni che modificano dati e formulano interrogazioni
* l'amministratore della base di dati (database administrator - DBA) responsabile per la struttura, lo schema e la gestione della base di dati

Nell'architettura di un DBMS abbiamo le seguenti sezioni:

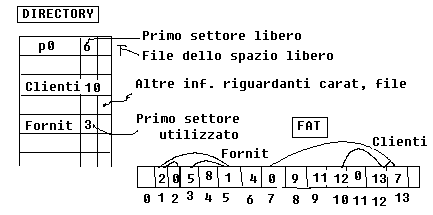
1. Dischi e file
2. Storage manager
3. Buffer manager
4. Index/file/record manager
5. Execution engine
6. Query compiler
7. Concurreny control
8. Logging/recovery
9. Transaction manager

**I Files**

**Il File è un insieme d’informazioni registrate su un supporto magnetico. Possono anche essere considerati File le informazioni stampate su un tabulato, visualizzate su video o acquisite da tastiera. Un File è formato da più componenti, normalmente i record , che per essere registrate devono essere prima impostate in memoria centrale. Per leggere le informazioni da un File si utilizzano le istruzioni di lettura che portano in memoria centrale le singole componenti in aree predisposte al loro ricevimento. Per lavorare con un File si deve, come prima operazione , utilizzare una particolare istruzione d’apertura (Open) che predispone il sistema ad operare con quel File. A fine lavoro si deve utilizzare l'istruzione di chiusura (Close) che indica al sistema di liberare le risorse utilizzate per l'elaborazione del File. Il numero di File che possono essere elaborati in contemporanea è limitato e dipende dal tipo di Sistema e dalla configurazione utilizzata .**

**I File su disco sono sempre associati ad un nome in modo che il Sistema Operativo li possa distinguere. Esiste sempre un File particolare, gestito dal Sistema, che prende il nome di Directory del disco, ove sono memorizzati tutti i nomi degli altri File con le loro caratteristiche fondamentali (Tipo, Indirizzo della prima registrazione, Spazio occupato ..).**

**Nei sistemi MS-DOS, Windows e in molti sistemi moderni lo spazio su disco è gestito utilizzando i settori (512 byte) non necessariamente in adiacenza fisica. Esiste, per questo, un File di Sistema che prende il nome di FAT (File Allocation Table) contenente una tabella in cui ogni casella rappresenta un settore del disco (casella 1 settore 1, casella 2 settore 2 ..) e contiene il numero del settore successivamente utilizzato per un determinato File (La Directory fa riferimento al primo settore utilizzato per ogni File). In questo modo l'ordine logico con cui si susseguono le informazioni nei File non corrisponde necessariamente all'ordine fisico (posizione assoluta) con cui sono state registrate. Quando le informazioni contenute in un File sono cancellate i settori occupati sono ceduti in blocco (tramite la FAT) alla lista dei settori disponibili. Si ottiene così un migliore sfruttamento dello spazio su disco anche se a discapito della velocità di lettura sequenziale. Con questo tipo di organizzazione non si ha, inoltre, la necessità di riservare a priori lo spazio occupato da ogni File.**



**Il File Clienti in questo caso occupa i settori 10,12,13,7**   
**I settori liberi in questo caso sono: 6,4,8,9,11**

**I File possono essere fondamentalmente di due categorie: i File di testo e i File binari.**

**I File di testo contengono informazioni esclusivamente in formato ASCII (o in altri codici equivalenti). In questi File le informazioni sono direttamente stampabili o visualizzabili.**

**I File binari possono contenere anche informazioni in formato binario (ad esempio informazioni numeriche rappresentate in aritmetica binaria proprio come sono rappresentate in memoria centrale). Le componenti di questi File, in genere, sono i record, mentre le componenti di un File di testo possono essere una qualsiasi informazione semplice (carattere, numero intero o frazionario, sequenza di caratteri) espressa in codice ASCII o equivalente.**

**I File** Sequenziali

**Sono quei File che possono essere elaborati solo sequenzialmente. I File sequenziali si usano per memorizzare informazioni che devono essere elaborate globalmente (Dati statistici o storici oppure informazioni che sono la copia di salvataggio d’altre informazioni). Utili per la memorizzazione di informazioni testuali, si tratta di strutture organizzate per righe e non per record.**

**Non sono adatte per grandi quantità di dati per il dilatarsi dei tempi d’accesso alle singole informazioni.**

Le caratteristiche principali di un FILE SEQUENZIALE sono:

* Semplicità di gestione del file.
* Trattamento dei dati rigorosamente sequenziale (da cui il nome).
* Relativa lentezza nell’elaborazione
* Difficoltà nel reperimento dei dati cercati
* Impossibilità di inserire o eliminare o modificare dati all’interno del file, mentre è possibile aggiungere dei dati in coda a quelli già esistenti.

Le modalità di utilizzazione di un file sequenziale sono 3:

* OUTPUT: per memorizzare dati in un file eliminando quelli eventualmente già presenti.
* APPEND: per memorizzare dati in un file sequenziale aggiungendoli in coda a quelli già presenti.
* INPUT: per leggere i dati dal file sequenziale

**I File ad Accesso Diretto**

**Sono quei File dove ogni componente (di solito le componenti sono dei record) può essere trattata direttamente indipendentemente dalle componenti che la precedono. L’accesso diretto può avvenire tramite R.R.N (Relative Record Number) e vale a dire attraverso la posizione relativa del record all’interno del File, quando questa sia nota, oppure attraverso al contenuto di un campo (semplice o composto), del record, che identifica il medesimo. Questo campo prende il nome di chiave d’accesso o Chiave Logica.**

**L'accesso attraverso R.R.N.**

**Con questa tipo di accesso ogni record del File è individuabile attraverso la sua posizione relativa R.R.N. (Relative Record Number) all'interno del File. Le posizioni relative vanno normalmente da 1 a N . Se il File è nuovo e si registra il primo record senza indicarne la posizione relativa questo andrà ad occupare la posizione 1, il secondo record scritto andrà ad occupare la posizione relativa 2 e così via. In questo modo si avrà la sensazione di avere a che fare con un File sequenziale con la particolarità però che i singoli record potranno essere elaborati direttamente attraverso la loro posizione relativa. Si possono leggere i record anche sequenzialmente e cioè uno dopo l'altro senza indicare la posizione relativa. In in questo caso, per individuare la fine del File, si deve utilizzare la funzione EOF.**

**Nel caso in cui i vari record sono in ordine su un particolare campo, per ricercarne uno, si potrà utilizzare la Ricerca Dicotomica. In questo caso si effettuerà un numero di letture massimo pari al logaritmo in base due del numero di record presenti nel File contro N/2 letture, in media, per ricercarlo con la Ricerca Sequenziale. L'organizzazione Relative è realizzata in modi diversi a seconda del linguaggio e dei sistemi utilizzati.**

|  |  |
| --- | --- |
|  | **Nei sistemi in cui lo spazio su disco occupabile deve essere riservato in dimensione per ogni singolo File, i vari record sono collocati in adiacenza fisica. In questo caso, per accedere ad un determinato record, il sistema applicherà la seguente formula per il calcolo dell'indirizzo:** |

**N = Indirizzo primo record + (N – 1) \* (lunghezza in byte del record)**

**La formula precedente è valida se il primo record ha posizione relativa 0.**

|  |  |
| --- | --- |
|  | **Se lo spazio su disco è gestito tramite la FAT (MS-DOS, Windows) l'individuazione di un record da leggere sarà effettuata, non tramite un calcolo matematico, ma utilizzando la FAT. Si passa da una casella all'altra per un numero di caselle pari al R.R.N. del record da cercare più uno (Se il numero relativo del primo record è 0 e ogni record occupa un intero settore).** |

**In generale se si vuole gestire un File Relative come una tabella composta da N record (anche registrabili in modo diretto) sarà possibile farlo registrando, precedentemente, tutti i record e indicando in un campo (campo di stato) la condizione di record contenente informazioni o record vuoto.**

**L'organizzazione Relative, oltre a poter essere utilizzata direttamente da tutti i linguaggi di programmazione, è anche quella che utilizzano, come base, le organizzazioni che permettono l'accesso tramite chiave logica.**

**L'Accesso tramite Chiave Logica**

**La Chiave logica è il contenuto di un campo (semplice o composto) che identifica un record specifico all'interno del File. (Chiave logica primaria se ne identifica uno solo, secondaria se ne identifica più di uno).**  
**Esistono diverse organizzazioni che permettono l'accesso tramite chiave logica.**

**Le Organizzazioni Index-Sequential**

**Questo tipo di organizzazione prevede che l'accesso al File Dati possa essere fatto attraverso un altro File di appoggio detto File Indice il quale contiene le chiavi logiche dei record presenti nel File Dati e il loro collegamento (una chiave un collegamento al record che contiene quella chiave). A seconda di come è organizzato l'Indice si hanno alcuni tipi di Index-Sequential .**

**Il termine Sequential indica la possibilità, presente in tutte le organizzazioni di questo tipo, di poter elaborare il File Dati in modo sequenziale e cioè un record dopo l'altro con chiave logica sempre più alta nell'ordine alfabetico.**

**L'accesso diretto al File Dati è fatto utilizzando la chiave logica del record che deve essere elaborato. Questa chiave è cercata, dal Sistema, nel File Indice e tramite il collegamento associato alla chiave (è l'indirizzo o meglio il R.R.N. del record con quella chiave sul File Dati) il Sistema accederà al record interessato sul File Dati.**

**Normalmente è permesso, per uno stesso File Dati, che sia costruito più di un Index in modo che sia possibile accedere attraverso più chiavi logiche (Multi- Index).**

**E’ possibile che una chiave logica possa essere contenuta in più record (Chiave logica secondaria) e in questo caso l'accesso diretto, con quella chiave, porterà ad uno dei record; sarà poi possibile, se quel record non è quello desiderato, procedere a leggere in modo sequenziale per accedere agli altri record con quella chiave fino a rottura di chiave o a EOF. E' anche possibile accedere a un record fornendo solo una parte (i primi caratteri) della chiave logica. In questo caso si dice che la lettura avviene per chiave parziale e il Sistema fornirà, se esiste, il primo record con quella parte di chiave. Si potrà, quindi, procedere in modo sequenziale per individuare, fra tutti i record, con quella parte di chiave, quello che ci interessa.**

**Storicamente l'organizzazione Index-Sequential si è sviluppata con l'evoluzione tecnologica delle unità a disco (dagli anni 60) e diverse sono state le versioni implementate dalle varie case. Possiamo affermare che le realizzazioni più vecchie utilizzavano, normalmente, un File Dati ordinato fisicamente sulla chiave logica e permettevano, per ogni File Dati, un unico indice, mentre le organizzazioni più moderne utilizzano un File Dati non necessariamente ordinato su alcun campo e permettono il Multi-Iindex.**

**L’organizzazione ISAM (Index Sequential Access Metod) della IBM oggi assolutamente non più in uso, è stata per anni (anni 60 e 70) la più importante ed usata organizzazione per i computer IBM; nel seguito verranno presentati i concetti di base delle organizzazioni più moderne che utilizzano i B-Tree per la costruzione dell'indice e tecniche di hash.**

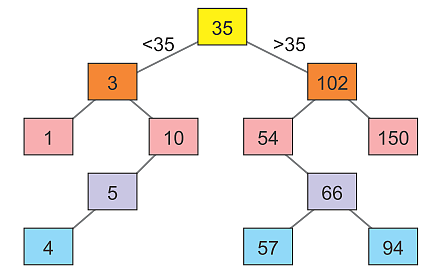
**Alberi binari e tecniche di hashing per la gestione delle chiavi**

**Alberi binari**

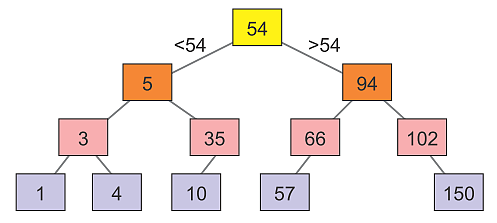
Gli indici di un file possono essere trattati in modo efficiente con tecniche che si basano sull’uso di alberi binari per le chiavi. La ricerca infatti diventa più veloce se le chiavi vengono mantenute in ordine, in modo da poter utilizzare il metodo della ricerca binaria.

ORDINE DI INSERIMENTO DELLE CHIAVI:

35 102 3 54 66 10 94 1 5 4 57 150

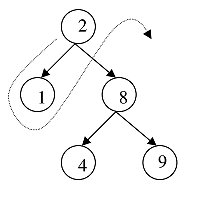


Per ottimizzare la ricerca è opportuno che l’albero si mantenga bilanciato: il bilanciamento consiste nel trasformare l’albero in modo da ottenerne uno equivalente, contenente cioè gli stessi dati, che abbia profondità minima, cioè con un numero basso di livelli gerarchici.

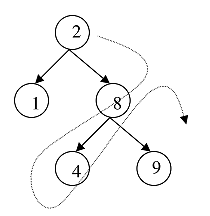


**Alberi binari di ricerca**

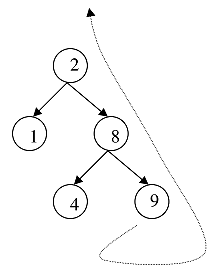
**Visualizzazione**



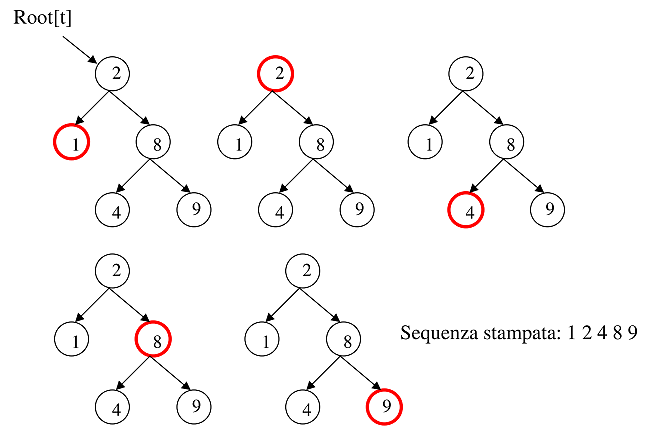
* si parte in 2
* viene chiamata la funzione sul figlio sinistro
* siamo in 1
* viene chiamata la funzione sul figlio sinistro
* non esiste figlio sinistro e la ricorsione termina
* torniamo in 1
* stampiamo 1
* viene chiamata la funzione sul figlio destro
* non esiste figlio destro e la ricorsione termina
* torniamo in 1
* la funzione termina
* torniamo in 2
* stampiamo 2



* viene chiamata la funzione sul figlio destro
* siamo in 8
* viene chiamata la funzione sul figlio sinistro
* siamo in 4
* viene chiamata la funzione sul figlio sinistro
* non esiste figlio sinistro e la ricorsione termina
* torniamo in 4
* stampiamo 4
* viene chiamata la funzione sul figlio destro
* non esiste figlio destro e la ricorsione termina
* la funzione termina
* torniamo in 8
* stampiamo 8
* viene chiamata la funzione sul figlio destro
* siamo in 9



* viene chiamata la funzione sul figlio sinistro
* non esiste figlio sinistro e la ricorsione termina
* torniamo in 9
* stampiamo 9
* viene chiamata la funzione sul figlio destro
* non esiste figlio destro e la ricorsione termina
* torniamo in 9
* la funzione termina
* torniamo in 8
* la funzione termina
* torniamo in 2
* la funzione termina



In un albero binario di ricerca l’operazione di ordinamento viene eseguita semplicemente attraversando i nodi dell’albero in modo ricorsivo

Molte applicazioni richiedono un insieme dinamico che fornisca solo operazioni di:

* Ricerca
* Inserimento
* Cancellazione

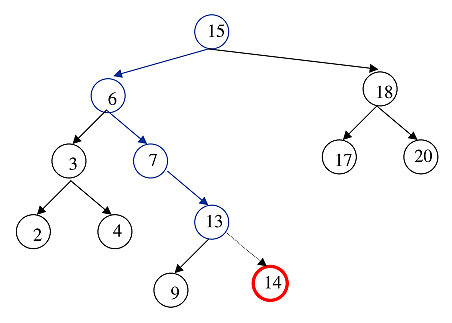
**Ricerca**

* L’idea è di confrontare la chiave di un nodo x con la chiave cercata
* nel caso che non coincidano si cerca solo nel sottoalbero in cui potrà trovarsi
* è possibile sapere quale sia il sottoalbero perché tutti i nodi del sottoalbero destro contengono chiavi maggiori della chiave di x (e nel sottoalbero sinistro chiavi minori)

**Inserimento**

1. Per trovare la posizione giusta ci si muove a partire dalla radice spostandosi sul sottoalbero destro o sinistro come in una ricerca
2. si prosegue però fino ad arrivare ad un punto in cui fallirebbe la ricerca
3. a questo punto si inserisce il nuovo nodo

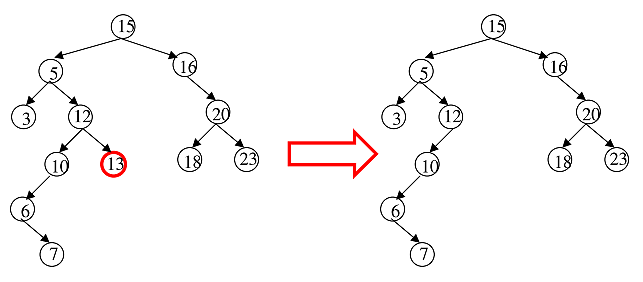
* Inserzione di key 14



**Cancellazione**

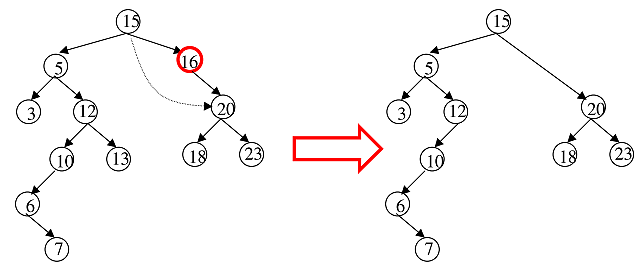
* La procedura di cancellazione è più laboriosa in quanto si deve tenere conto di tre casi possibili
* dato un nodo z i casi sono:
  + z non ha figli
  + z ha un unico figlio
  + z ha due figli
* Caso 1:

Nel caso in cui z non abbia figli si elimina direttamente il nodo z



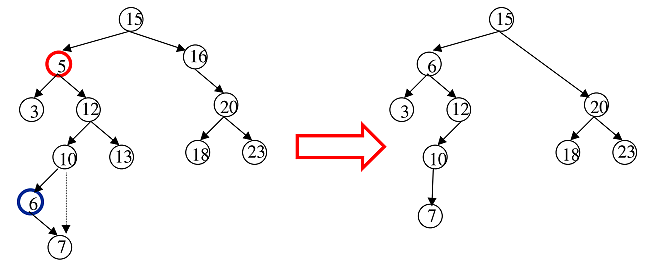
* Caso 2:

Nel caso in cui z abbia un unico figlio si rimuove z e si collega il figlio al posto di z



* Caso 3:

1. Nel caso in cui z abbia 2 figli si determina il successore x di z
2. si copia il contenuto di x al posto di quello di z
3. infine si elimina il vecchio nodo x (caso 1 o 2)

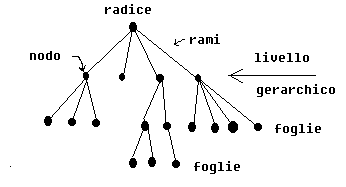


**B-tree (o Balanced tree, albero bilanciato)**

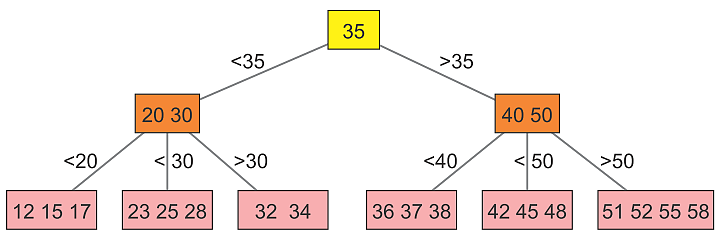
Per diminuire il numero di accessi nella struttura ad albero, nella ricerca delle chiavi, si usano **alberi generali** suddividendoli in sottoalberi, detti pagine, in modo tale che ogni accesso a una pagina richieda un solo accesso al disco. Con l’inserimento di nuove chiavi la struttura ad albero si espande e la crescita deve essere mantenuta sotto controllo per garantirne il bilanciamento.

A queste esigenze risponde la struttura detta **B-tree** (o Balanced tree, albero bilanciato), che può essere considerata una **generalizzazione** degli alberi binari bilanciati.

**Occorre, prima, spiegare alcuni concetti relativi ai B-tree. Bilanced Tree e cioè Albero Bilanciato è il termine esteso. La struttura ad albero è una struttura di tipo gerarchico composta da elementi, detti nodi, collegati fra di loro mediante i così detti rami. Un primo elemento (radice) è collegato ad altri (padre -> figli) ed ognuno di questi può ,a sua volta, essere collegato ad altri in ordine gerarchico (non sono possibili collegamenti con elementi in ordine gerarchico più alto ed inoltre un figlio può avere un unico padre). Bilanciato sta ad indicare che il numero di figli, per ogni elemento, è uguale, eccezion fatta per gli ultimi (foglie) che non hanno discendenti.**



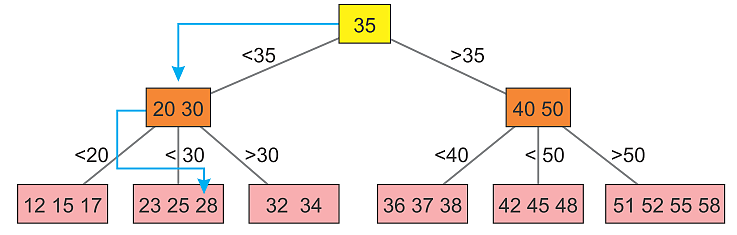
Per esempio un albero B-tree viene illustrato nella figura seguente:



Si consideri per esempio la pagina contenente le due chiavi che hanno i valori 20 e 30: essa ha tre pagine come discendenti che contengono rispettivamente le chiavi minori di 20, comprese tra 20 e 30, e maggiori di 30.

La ricerca di una chiave nella struttura B-tree avviene come per gli alberi binari: a partire dalla radice vengono attraversati i nodi nei sottoalberi finché la chiave viene trovata oppure finché la ricerca termina senza successo. Se la chiave cercata è compresa tra quelle della pagina considerata, la ricerca si conclude in modo positivo (nella pagina oltre al valore della chiave è memorizzato un puntatore al record), altrimenti si va a esaminare tra le pagine discendenti quella corrispondente all’intervallo entro il quale è compresa la chiave richiesta.

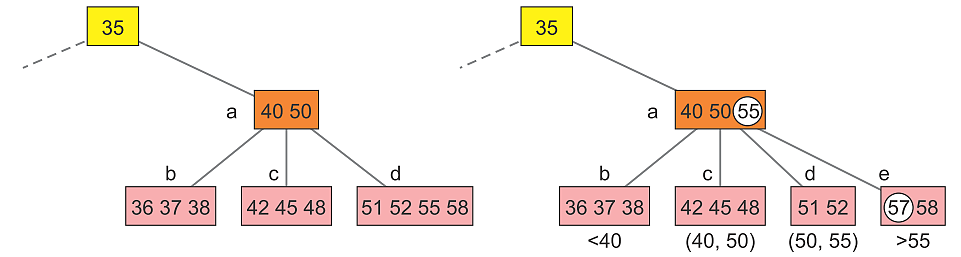
Con riferimento all’esempio precedente, se la chiave da cercare ha valore 28, il percorso è il seguente:



Nella struttura B-tree l’inserimento di una nuova chiave è un’operazione che deve avere l’obiettivo di mantenere l’albero bilanciato. Può accadere che l’inserimento richieda l’allocazione di una nuova pagina, nel caso in cui la pagina adatta a contenere la nuova chiave sia piena perché contiene già 2n elementi.

Per esempio, se nell’albero-B precedente si deve inserire la nuova chiave 57, si procede nel seguente modo:

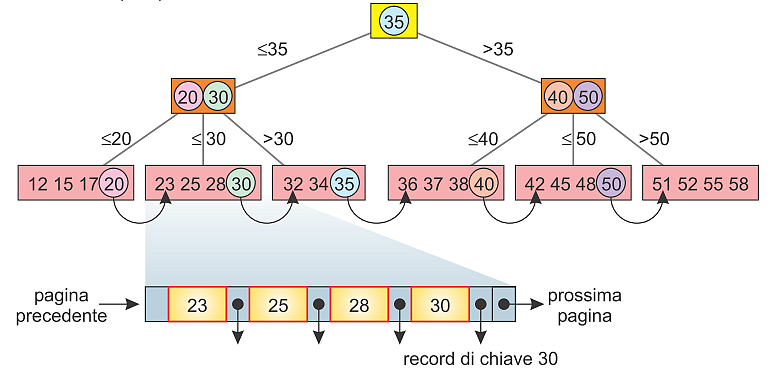
* la chiave 57 non è presente, ma non può essere messa nella pagina contrassegnata dalla lettera d perché è piena;
* la pagina d viene divisa in due pagine, creando una nuova pagina e;
* le chiavi vengono suddivise in parti uguali tra le due pagine e l’elemento centrale (55) viene spostato alla pagina di livello superiore.



In questo modo vengono conservate le caratteristiche della struttura B-tree. Successivi inserimenti possono causare il riempimento anche della pagina di livello superiore, secondo un meccanismo di suddivisione delle pagine che può arrivare anche alla radice dell’albero: la struttura si espande così in modo dinamico, e allo stesso tempo controllato, al crescere del numero dei record memorizzati nel file.

La struttura B-tree dell’esempio precedente permette l’individuazione della posizione di un record accedendo a non più di tre pagine dell’albero: per esempio il record con chiave di valore 35 viene individuato al primo tentativo, quello con chiave di valore 40 al secondo, mentre il record con chiave 45 richiede l’accesso a tre pagine. È piuttosto complicato però accedere sequenzialmente ai record ordinati secondo il valore della chiave per rispondere, per esempio, alla seguente richiesta: “elencare tutti i record con chiave maggiore di 30”.

L’organizzazione **B+-tree** permette di ovviare a questo inconveniente al prezzo di una certa ridondanza delle informazioni. In un albero B+ solo i nodi foglia contengono puntatori ai record: pertanto tutti i valori delle chiavi nei nodi intermedi devono essere riportati negli opportuni nodi foglia. Inoltre le foglie dell’albero sono tutte collegate tra di loro per permettere la lettura sequenziale dei record. La figura mostra la struttura dei nodi foglia e l’albero B + corrispondente al B-tree dell’esempio precedente.



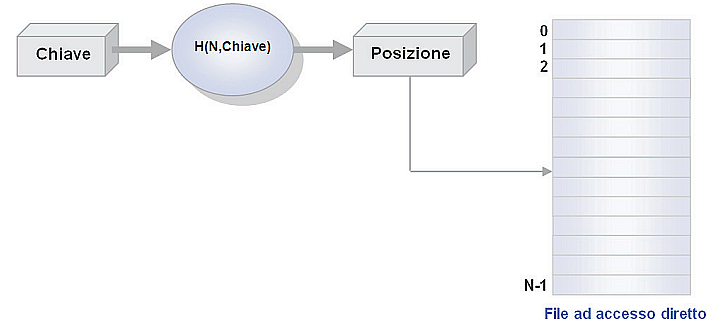
**Le organizzazioni che adoperano i B-Tree hanno le seguenti caratteristiche:**

|  |  |
| --- | --- |
|  | * **Velocità d’accesso ad ogni singolo record elevata. Il numero di nodi da leggere sull'Indice è al massimo il logaritmo in base [m/2] (parte intera per eccesso del rapporto) del numero di record presenti sul File Dati. Infatti, ogni volta che si cambia nodo si riduce l'insieme di ricerca di [m/2] volte, al minimo, essendo le strade che si dipartono da ogni nodo di tale numero. Supponendo, quindi, che il numero di record sul File Dati sia un milione e l'ordine del B-Tree sia uguale a dieci (nove chiavi presenti al massimo in ogni nodo) il logaritmo in base cinque di un milione è circa otto -> 1.000.000./ (5\*5\*5\*5\*5\*5\*5\*5) = 1 (il logaritmo in base cinque di un milione corrisponde al numero di volte che si deve dividere un milione per cinque per ottenere uno)** |
|  | * **L'Indice non necessita di riorganizzazioni.** |
|  | * **Attraverso l'Indice è possibile accedere (mediante un opportuno algoritmo) al File Dati con i record in ordine crescente di chiave. Si può, quindi, passare da un record all'altro del File Dati in modo sequenziale in ordine per chiave logica.** |
|  | * **E' permesso il Multi-Index e mediante un B-Tree per ogni tipo di chiave logica possiamo avere più indici associati al File Dati.** |
|  | * **I record del File Dati sono tutti identificabili attraverso il loro R.R.N. e non cambiano mai la loro posizione relativa.** |
|  | * **Si può operare in ricerca anche per chiave parziale.** |

**Le tecniche di hashing**

I file con organizzazione random, come abbiamo visto, contengono record che vengono identificati attraverso la posizione che essi occupano all’interno del file.

Volendo associare a ciascun record un valore chiave, di norma alfanumerico, occorre determinare una funzione che possa trasformare la chiave in un numero. Questa tecnica di indicizzazione si chiama metodo di trasformazione della chiave o metodo hash: una funzione hash, avente come argomento una chiave, deve restituire la posizione del record corrispondente, con l’obiettivo di distribuire nel modo più uniforme possibile i record all’interno del file.



La funzione hash calcola un numero di record sulla base del valore della chiave e del massimo numero di record previsti per il file, cioè trasforma l’intervallo dei valori della chiave (che può

essere anche molto vasto) in un numero compreso tra 0 e NR-1, dove NR è il numero di record nel file.

Poiché il calcolo dipende dal numero massimo di record per il file, in fase di creazione del file, occorre specificare qual è il numero massimo dei record.

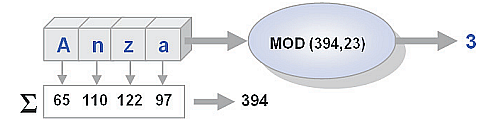
Una semplice funzione hash utilizza un numero intero ricavato dalla chiave, divide questo intero per il numero massimo di record e usa il resto della divisione come valore restituito dalla funzione.

Per esempio i valori delle chiavi possono essere codici numerici compresi tra 1000 e 3000, anche se non tutti i numeri vengono utilizzati. Supponendo che i record debbano essere memorizzati in un file di 100 record, la funzione hash divide il codice numerico per 100, ottenendo come resto un numero compreso tra 0 e 99 che identifica la posizione nel file:

**posizione = codice mod 100.**

Nel caso di chiavi alfanumeriche si costruiscono funzioni hash basate sul valore dei codici ASCII (o di un’altra codifica) dei caratteri he compongono la chiave. Per esempio, supponiamo che

la chiave sia composta da 4 caratteri alfabetici; si considera allora la somma dei codici ASCII dei caratteri componenti la chiave e si divide per il numero massimo di record presenti nel file ottenendo un numero che fornisce la posizione del record nel file, in fase di scrittura e di ricerca. Applichiamo questa tecnica ai record del file anagrafe usato in precedenza e consideriamo la prima delle 15 chiavi dei record che compongono il file: “Anza”. Supponiamo di voler distribuire i record in un hash file con 23 record:



Eseguiamo il calcolo per la chiave “Lore”

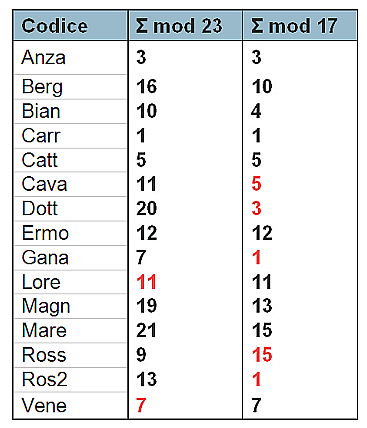


Quando un record viene aggiunto in un file gestito con una tecnica hashing, il software di gestione dei file tenta di mettere il record nella posizione calcolata dalla funzione hash . Se quella posizione è già occupata da un record, accade una **collisione** e il record deve essere messo da qualche altra parte.

Calcoliamo, per esempio, la posizione di inserimento del record di chiave “Cava”:



In questo caso, come si vede, si ha una collisione con il record di chiave “Lore”. Estendiamo il procedimento a tutti i record del file. I risultati dell’esercizio sono riportati nella tabella sottostante che mostra, accanto alle posizioni calcolate con l’algoritmo illustrato (colonna Σmod 23 ), anche il caso di posizioni calcolate con il medesimo algoritmo, ma dimensione del file ridotta a 17 record (colonna Σ mod 17). Le collisioni sono evidenziate in rosso nella tabella.



Se i 15 record del file sono collocati all’interno di un file di 17 record si hanno 5 collisioni. Le collisioni scendono però a due se si usa un file di 23 record. Il risultato, decisamente intuitivo, ha carattere di generalità ed è confermato dalle raccomandazioni dei manuali che suggeriscono, nella creazione di un file indicizzato in modo hashing, di incrementare il numero dei record previsti del 30% e di rrotondare in eccesso il valore ottenuto a un numero primo. Nel nostro

caso, con un file di 15 record:

15 x 1,3 = 19,5 arrotondato a 23

si ottiene 23 come dimensione suggerita per il file. Il suggerimento, che ha lo scopo di ridurre il numero di possibili collisioni, è frutto di considerazioni di tipo statistico sulle prestazioni delle funzioni di hashing e, conseguentemente, non esclude che per particolari istribuzioni di valori

delle chiavi si manifestino molte collisioni. Per esempio: modificando di poco l’algoritmo che abbiamo utilizzato nell’esercizio e applicandolo ai 15 record del file si evidenziano addirittura 8 collisioni.

Ci sono diverse tecniche per la **gestione delle collisioni**.

Con il metodo della **scansione lineare**, il record viene collocato nella prima posizione libera successiva a quella calcolata. Se viene raggiunta la fine del file, la ricerca continua ripartendo dall’inizio del file sino a trovare un posto libero. Il procedimento ha fine nell’ipotesi di inserire un numero di record inferiore alla dimensione massima del file creato.

Per esempio supponiamo di inserire i record del file in ordine di chiave. Iniziamo dal record di chiave “Anza” inserito in posizione 3, seguito da “Berg” in posizione 16, “Bian” nel posto 10 e così via.

Va tutto bene sino a “Lore” che trova il posto 11 occupato. Con il metodo della scansione lineare si cerca di collocare “Lore” in posizione 12 (già occupata) e poi nel posto 13 che, essendo libero, viene occupato. Risolto un problema se ne presenta poco dopo un altro, non previsto, quando si cerca di allocare il record di chiave “Ros2” a cui competerebbe proprio il posto 13. Avevamo previsto due collisioni e invece ne troviamo tre.

Quindi con il metodo della scansione lineare è possibile che si creino lunghe catene di record adiacenti. Questo può rallentare l’accesso ai record in modo significativo, dal momento che sarebbe necessario leggere ciascun record nella catena per trovare quello desiderato.

Un metodo alternativo è rappresentato dalla **scansione quadratica**: viene calcolato il numero di record, se il posto è già occupato si va a controllate il primo posto successivo, poi il quarto, il nono e così via sino a un opportuno criterio di arresto. Questo metodo tende a distribuire le collisioni prodotte dalla funzione hash attraverso tutto il file, piuttosto che creare accumuli attorno a certe posizioni.

Una terza soluzione, molto diffusa nelle implementazioni di indicizzazione con funzione di hashing, consiste nel creare un’area separata, detta anche **area di overflow**, adatta a contenere i record che hanno valori di chiave corrispondenti a situazioni di collisione con le chiavi già esistenti.

Applicandola all’esempio precedente si giungerebbe alla situazione illustrata nella figura.

