Introduzione

Definizioni e tipologie di sistemi informativi

Sistemi informativi: insiemi di **persone**, **risorse** e **strumenti** utilizzati da un'organizzazione in modo coordinato, con l'obiettivo di **acquisire**, **selezionare**, **ridistribuire informazioni** utili ad individuare strategie di gestione efficienti ed efficaci in tempi utili e nei giusti livelli di sintesi.

Prevedono la raccolta e classificazione delle informazioni tramite l'utilizzo di **procedure integrate ed idonee**.

Correlano informazioni a decisioni e controllo.

Tipologia		Supporto ai processi…	Supporto alle decisioni…	Elaborazione (input → elaborazione → output)
Strategic system	**E** xecutive **S** upport **S** ystem	…direzionali	…non strutturate	Dati aggregati, esterni e interni → grafici, simulazioni, interattività → proiezioni
Management system	**M** anagement **I** nformation **S** ystem	…gestionali	…semi-strutturate	Grandi volumi di dati, sintetici → rapporti, semplici modelli, analisi di dettaglio → sommari e rapporti di errori
	D ecision **S** upport **S** ystem	…direzionali	…non strutturate	Piccoli volumi di dati, modelli analitici → simulazioni e analisi interattive → rapporti di analisi, decisioni
Kwnoledge system	**K** nowledge **W** ork **S** ystem			Specifiche di progetto, basi di conoscenza → modellazione, simulazioni → modello, grafici
	O ffice **A** utomation **S** ystem			Documenti, piani di lavoro → documenti, gestione, pianificazione, comunicazione → documenti, piani di lavoro, posta
Operational system	**T** ransaction **P** rocessing **S** ystem	…operativi	…strutturate	Transazioni, eventi → ordinamenti, liste, aggiornamenti → rapporti di dettaglio, elenchi, sommari

Le informazioni (divario percettivo, risorse, valore, processi decisionali)

Divario percettivo: "gap" tra la percezione della realtà da parte dell'organizzazione e la realtà stessa.

Le informazioni che "catturano" la realtà, presenti nei database, rappresentano un valore (sempre crescente) richiesto da tutta l'organizzazione per pianificare, gestire, controllare.

I sistemi informativi trasformano dati e informazioni in conoscenza.

Qualità dell'informazione	
Soggettività	*il valore dell'informazione è diverso per individui e decisioni diversi*
Rilevanza	*l'informazione deve essere pertinente alla decisione da prendere*
Tempestività	*l'informazione deve essere disponibili al momento decisionale*
Accuratezza	*l'informazione deve essere corretta e precisa*
Presentazione	*l'informazione deve essere utilizzabile senza ulteriori elaborazioni*
Accessibilità	*l'informazione deve essere disponibile a chi la richiede quando sono necessarie*
Completezza	*l'informazione deve essere completa per permettere una decisione corretta*

Impatti della tecnologia dell'informazione (ICT) sull'organizzazione

Global networks	Enterprise networks	Distributed computing	Portable computing	Graphical user interfaces
↓	↓	↓	↓	↓
Indipendenza dal luogo	Lavoro cooperativo e di gruppo	informazioni disponibili e affidabili	Organizzazione virtuale	Accessibilità alle conoscenze

Ciclo di vita e processo incrementale dei sistemi informativi

I sistemi informativi sono realizzati necessariamente attraverso un processo incrementale che si sviluppa lungo l'intero ciclo di vita del software e tiene conto di tutti gli aspetti necessari al corretto funzionamento del sistema.

Ciclo di vita
Definizione strategica
Pianificazione
Analisi dell'organizzazione
Progettazione del sistema
Progettazione esecutiva
Realizzazione e collaudo in fabbrica
Installazione
Collaudo del sistema installato
Esercizio
Evoluzione
Messa fuori servizio
Post mortem

Fattori del processo incrementale

Visione globale del sistema e del suo ruolo nell'ente
Quadro concettuale di riferimento
Adeguate metodologie di progetto e controllo qualità
Appropriate metodologie per il monitoraggio in esercizio
Esigenze di interoperabilità e scalabilità
Flessibilità e possibilità "illimitate" di accesso ai dati
Architettura ad elevate prestazioni
Ridondanza, Sicurezza, Protezione
Disponibilità continua dei dati
Interfaccia utente adattativa

ERP: Enterprise Resource Planning, applicazioni software modulari che permettono di realizzare sistemi integrati e che possono essere personalizzate sulle esigenze e sui processi dell'ente

Il sistema informativo *utilizza* il sistema informatico per svolgere gran parte dell'attività di gestione ed elaborazione dei dati, tuttavia è qualcosa di più ampio rispetto al sistema informatico in quanto comprende processi aziendali, risorse umane e fisiche, metodologie…

Data Base Management System (DBMS)

DBMS: sistema software in grado di **gestire efficacemente le informazioni** necessarie ad un sistema informativo.

Garantisce la persistenza dei dati e la loro rappresentazione in forma integrata.

RDBMS: DBMS che offre, come modello logico dei dati, il modello relazionale

Base dati: collezione di dati (database) gestito tramite un DBMS.

Vedere anche: Distributed DBMS, Multidimensional DBMS, KDBMS (?)

Essenza del corso

Punti di vista
- ✓ Utente: come utilizzare un Data Base (modelli dei dati, linguaggi, SQL)
- ✓ Progettista: come progettare un DB e relative applicazioni (modello E/R, teoria relazionale, strumenti)
- ✓ Sistemista: come gestire un DBMS (architettura, indici, esecuzione di interrogazioni, transazioni)

Aspetti logici da modellare	
Conoscenza concreta	*oggetti e fatti specifici*
Conoscenza astratta	*fatti generali che descrivono la conoscenza concreta*
Conoscenza procedurale	*modalità per modificare la conoscenza concreta*
Dinamica	*modalità per l'evoluzione nel tempo della conoscenza*
Comunicazione	*modalità per accedere alla conoscenza*

Informazioni strutturate nei DBMS	
Campo (o attributo)	*dato elementare: unità minima di informazione dotata di significato*
Record	*raggruppamento di campi relativi ad un medesimo oggetto*
Chiave (primaria)	*campo utilizzato per identificare un record e l'oggetto da esso rappresentato*
Archivio	*insieme omogeneo di registrazioni memorizzato su memoria permanente*

Caratteristiche dei dati	
Struttura dei record	*campi dei record e relazioni logiche tra i campi*
Tipi di associazioni	*relazioni esistenti tra i diversi archivi*
Volatilità	*più un archivio è soggetto a modifiche più è volatile*
Espandibilità	*gli archivi possono dover essere riorganizzati e riprogettati nel tempo*
Dimensioni	*archivi di grandi dimensioni richiedono organizzazioni più sofisticate ed efficienti*
Periodo di vita	*archivi a "breve durata" richiedono minori manutenzioni*

Modalità d'uso dei dati	
Tipi di operazioni eseguite	*modifiche, ricerche (tipi), ordinamenti, interattività, …*
Frequenza di accesso ai dati	*regola (empirica) del "80-20": l'80% degli accessi riguarda il 20% dei dati*
Tempi di risposta richiesti	*presenza di vincoli sui tempi di risposta*
Affidabilità	*modi e misure della garanzia di integrità dei dati a fronte di malfunzionamenti*
Sicurezza	*modi e misure di regolamentazione delle modalità d'uso dei dati da parte degli utenti*

Biografie e riferimenti

Esercizi di Progettazione di basi di dati (Esculapio Ed.) – D. Maio, S. Rizzi, A. Franco

Basi di Dati: modelli e linguaggi di interrogazione (McGraw-Hill Italia) – P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone

Basi di Dati. Architetture e linee di evoluzione (McGraw-Hill Italia) – P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone

Lezioni di Basi di Dati (Esculapio Ed.) – P. Ciaccia, D. Maio

SQL the Complete Reference (McGraw-Hill/Osborne Media) – James R. Groff, Paul N. Weinberg

Transaction processing: concepts and techniques (Morgan Kaufmann) – J. Gray, A. Reuter

Database Management Systems (McGraw-Hill) – R. Ramakrishnan, J. Gehrke

Sistemi di basi di dati – I fondamenti (Pearson – Addison Wesley) – R.A. Elmasri, S.B. Navathe

Funzionalità DBMS

Sistemi informatici settoriali

I sistemi informatici settoriali (o dipartimentali) sono sviluppati per singoli settori aziendali, quindi:

- Gestiscono processi e informazioni di volumi contenuti
- Implementano la gestione dei dati secondo la visione di un singolo settore
- Determinano la realizzazione di differenti sistemi per gli stessi dati, creando ridondanze, spreco di memoria e risorse, ma soprattutto difficoltà (impossibilità) di manutenzione dei dati continuativa e completa
- Determinano la mancanza di standard e assenza di identificazione di vincoli a livello globale di organizzazione
- Realizzano i comportamenti infra-settore (più numerosi e frequenti) ma non sono in grado di gestire i comportamente inter-settore (flussi meno frequenti ma esistenti)

I DBMS sono uno strumento per superare i difetti dei sistemi settoriali.

L'utilizzo di file system per gestire grandi quantità di dati in modo persistente e condiviso sarebbe possibile ma presenta diversi inconvenienti:

- × Povertà di astrazione per modellare i dati
- × Limitazione dei meccanismi di condivisione
- × Limitazione dei meccanismi di protezione da guasti
- × L'accesso ai dati è determinato da una descrizione degli stessi all'interno del codice delle applicazioni (rischio di inconsistenza)
- × Non sono disponibili servizi aggiuntivi offerti da un DBMS
- ✓ Attenzione: la gestione dei dati può risultare più efficiente tramite fle system.

DBMS

Modello dei dati: collezione di concetti utilizzati per descrivere i dati, le associazioni e i vincoli.

L'astrazione logica con cui i dati vengono resi disponibili definisce un **modello dei dati**.

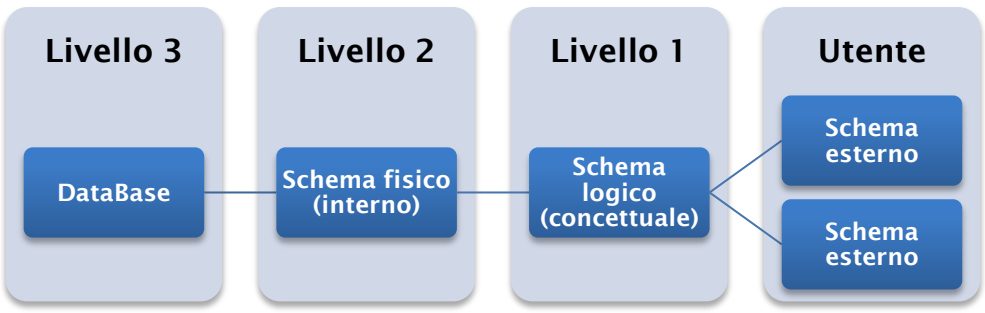
Esempi: modello gerarchico, modello reticolare, modello relazionale.

Nei Data Base, gli **schemi** definiscono la struttura dei dati (parte intensionale del DB), le **istanze** sono invece i dati veri e propri (parte estensionale del DB). Lo schema permette di interpretare i dati dell'istanza.

Un obiettivo dei DBMS consiste nel fornire indipendenza fisica e logica.

Indipendenza fisica: l'organizzazione fisica dei dati dipende da considerazioni legate all'efficienza, la riorganizzazione fisica dei dati non deve comportare effetti collaterali sulle applicazioni

Indipendenza logica: le modifiche allo schema logico non devono comportare modifiche degli schemi e delle applicazioni



Il **livello fisico** consiste di una serie di file residenti su memorie di massa, contengono dati, indici, altro.

Lo schema fisico descrive come il DataBase logico è rappresentato a livello fisico (in quale/i file è memorizzata una relazione).

La gestione del DataBase fisico è a carico dell'amministratore del DBMS.

Il **livello esterno** è descrive (tramite viste) parte dello schema logico a seconda delle esigenze dei diversi utenti, può combinare i dati di diverse relazioni o ridurre (filtrare) i dati esposti. Le viste possono essere usate anche per rendere trasparente agli utenti (e alle applicazioni) una ristrutturazione dello schema logico, realizzare un controllo degli accessi, calcolare dati dinamicamente senza introdurre ridondanze.

Linguaggi del DBMS

Data Definition Language: linguaggio per la definizione degli schemi (logici, esterni, interni)

Data Manipulation Language: linguaggio per l'interrogazione e la modificare delle istanze nei DataBase

Data Control Language: linguaggio per il controllo dei DataBase (controllo accessi, …)

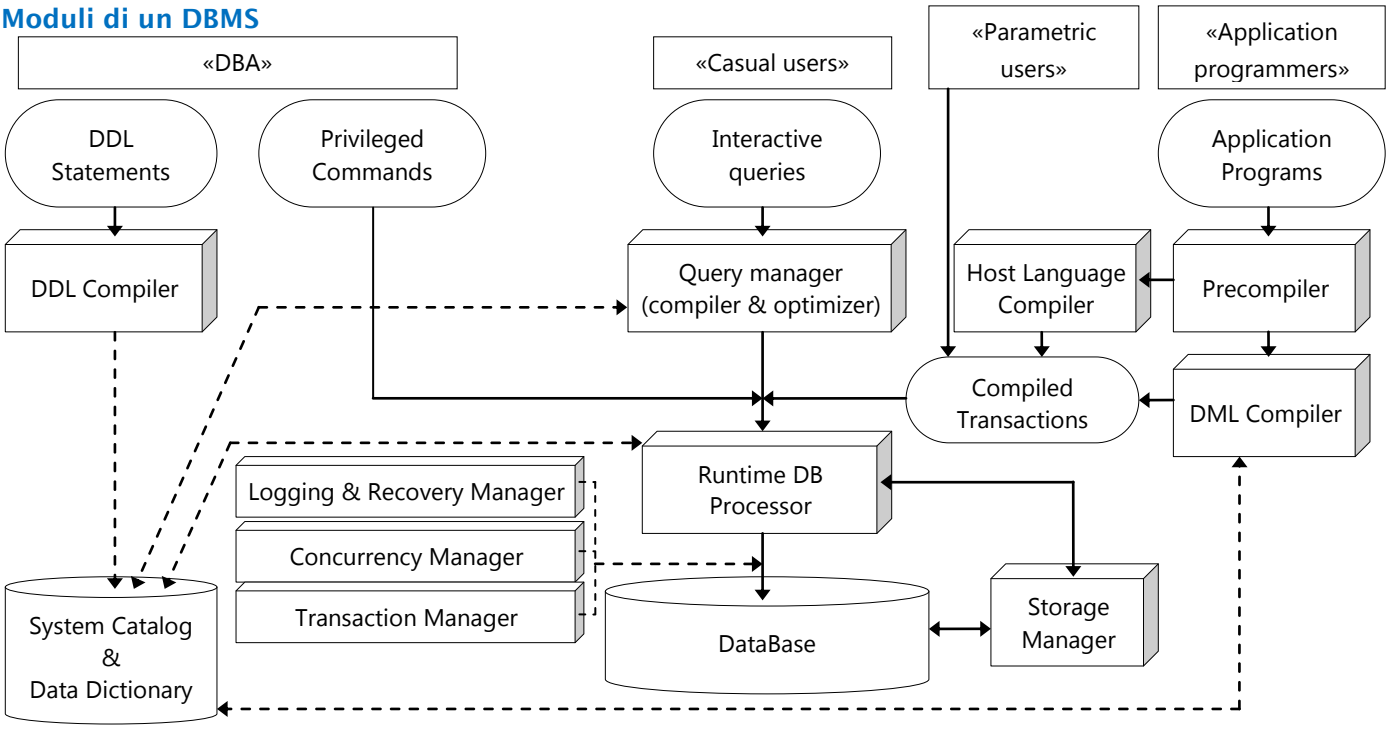
SQL: Standard Query Language è il linguaggio standardizzato per i DataBase basati sul modello relazionale, include tutte le tipologie di linguaggi (DDL, DML, DCL).

Esistono inoltre linguaggi nativi Turing-completi del DBMS, linguaggi ospiti per manipolare i dati con procedure esterne (ODCB, JDBC), linguaggi ospiti con precompilatore (es: pro*C per SQL in ambiente ORACLE).

La gestione delle autorizzazioni in un DBMS è affidata di Data Base Administrator (DBA) che conferiscono ai singoli utenti i giusti privilegi.

I DBMS devono garantire che gli accessi ai dati, da parte di diverse applicazioni, non interferiscano tra loro; pertanto devono gestire richieste e aggiornamenti concorrenti sugli stessi dati. Devono inoltre garantire l'integrità dei dati a fronte di guasti.

Moduli di un DBMS



Progettazione Basi di dati

Meccanismi di astrazione

Classificazione: meccanismo di raggruppamento di oggetti, funzioni o stati in classi (insiemi), in base alle loro proprietà

Aggregazione: meccanismo di definizione delle relazioni tra classi che rappresentano parti o componenti di altre classi; cattura le relazioni di tipo "è parte di"

Generalizzazione: meccanismo di astrazione delle caratteristiche comuni a diverse classi (definizione di superclassi); cattura le relazioni di tipo "è un". Il meccanismo inverso è detto specializzazione. Le generalizzazioni possono avere copertura totale o parziale, ed essere specializzate in classi disgiunte o sovrapposte.

Proiezione: meccanismo di cattura della vista delle relazioni strutturali fra oggetti, funzioni e stati; distingue il *punto di vista* dei diversi soggetti

Le associazioni modellano le relazioni tra classi, è importante modellare i vincoli di cardinalità esistenti tra classi (uno a uno; uno a molti; molti a molti). Solitamente le aggregazioni sono tra due classi, possono esistere relazioni ternarie ma sono rare.

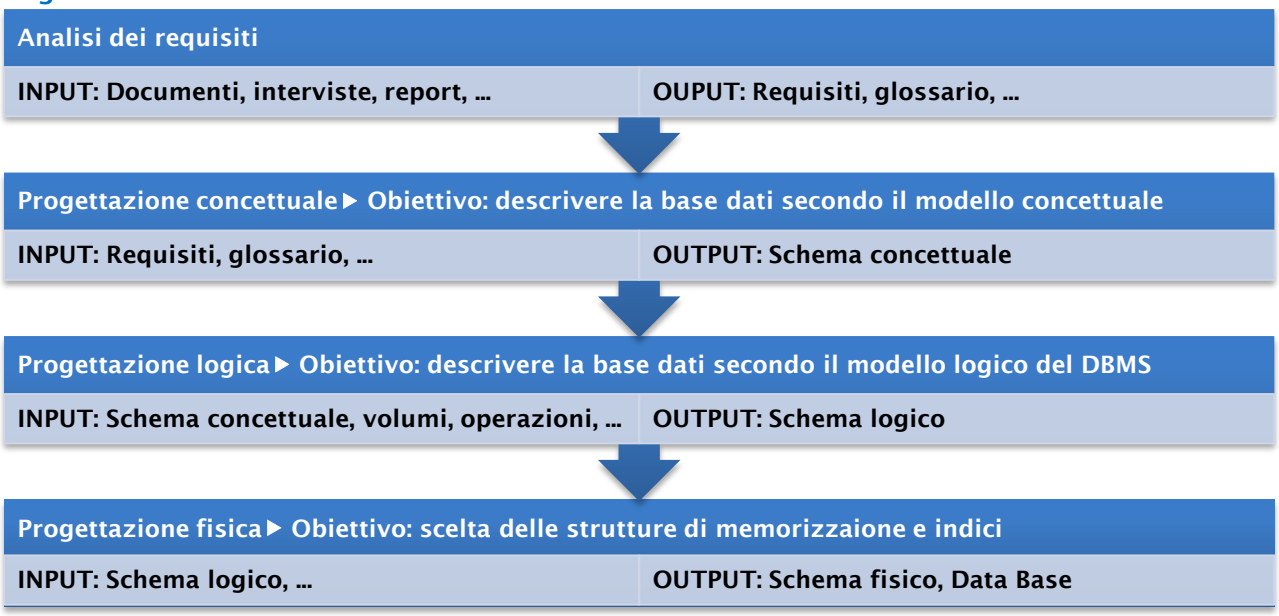
Analisi

Analisi orientata alle funzioni: ha l'obiettivo di rappresentare un sistema come una rete di processi e un insieme di flussi informativi tra processi; questo tipo di analisi produce una gerarchia funzionale; utilizza i Data Flow Diagram

Analisi orientata agli oggetti: pone l'enfasi sull'identificazione degli oggetti e sulle interrelazioni tra essi esistenti; si basa sul concetto che le proprietà strutturali degli oggetti sono stabili nel tempo mentre l'uso degli oggetti è sensibilmente mutevole

Analisi orientata agli stati: ha l'obiettivo di definire gli stati operativi in cui può trovarsi il sistema (le sue classi, funzioni, relazioni) in momenti diversi, definisce inoltre le transizioni di stato.

Progettazione delle basi di dati



Modelli logici e concettuali

I modelli logici sono utilizzati nei DBMS per l'organizzazione dei dati, sono indipendenti dalle strutture fisiche di memorizzazione e sono utilizzati dalle applicazioni.

I modelli concettuali permettono di rappresentare i dati indipendentemente dal particolare sistema: descrivono i concetti del mondo reale ad un livello di astrazione alto.

Modello Entity Relationship

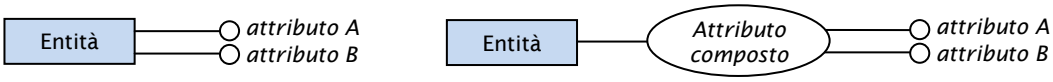
Entità: insieme (classe) di oggetti della realtà (dominio applicativo) che possiedono caratteristiche comuni e hanno esistenza autonoma; uno specifico elemento di un'entità è detto **istanza**.

Associazione: legame logico tra entità, un'istanza di associazione è una combinazione di istanze delle entità correlate: non possono esistere due coppie uguali di istanze delle classi che fanno parte dell'associazione

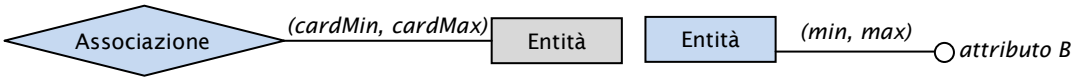
Il **grado** delle associazioni rappresenta il numero di entità coinvolte.

Le **associazioni ad anello** coinvolgono più volte la stessa entità, nel caso di associazioni ad anello è necessario specificare se si tratti di associazioni *simmetriche*, *riflessive*, *transitive*.

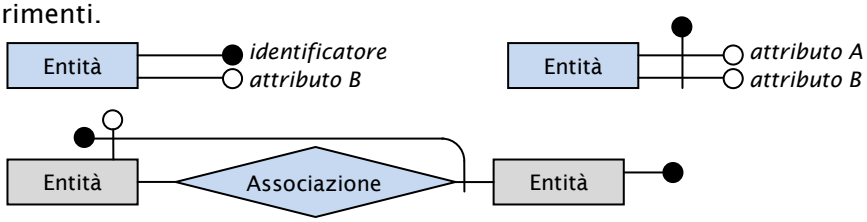
Attributo: proprietà elementare di un'entità o di un'associazione definita su un dominio di valori, possono essere semplici o composti.



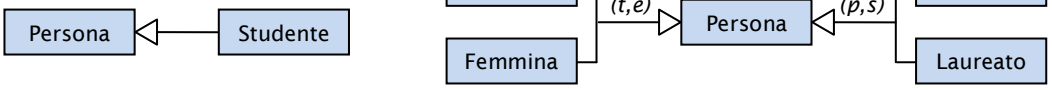
Vincolo di cardinalità: coppie di valori (min, max) associate ad ogni entità che partecipa ad un'associazione che specificano il numero minimo e massimo di istanze dell'associazione a cui un'istanza dell'entità può partecipare



Identificatore: definisce gli attributi che individuano unicamente le istanze delle entità; deve valere la proprietà di **minimalità** (nessun sottoinsieme proprio dell'identificatore deve a sua volta essere un identificatore); può essere **interno** (costituito da attributi dell'entità) o **esterno** (costituito da attributi di altre entità collegate e da eventuali attributi dell'entità); si dice **semplice** se composto da un elemento, **composto** altrimenti.

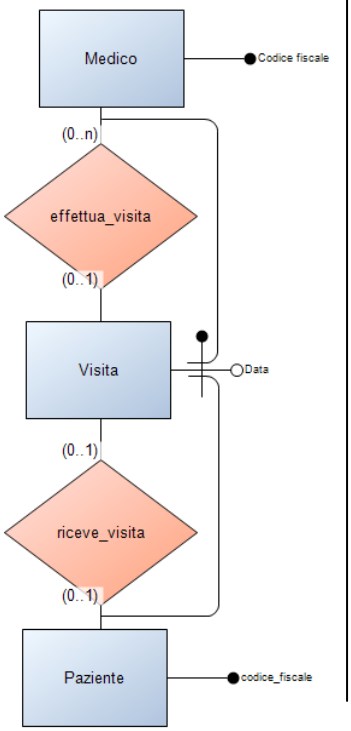


Gerarchia: generalizzazione di un gruppo di entità (dette *specializzazioni*), richiede che sia specificato il tipo di copertura (totale o parziale, disgiunta (esclusiva) o sovrapposta); le proprietà dell'entità "madre" sono ereditate dalle entità "figlie" e non devono essere indicate in esse. I **subset** sono gerarchie nelle quali si evidenzia una sola classe specializzata.

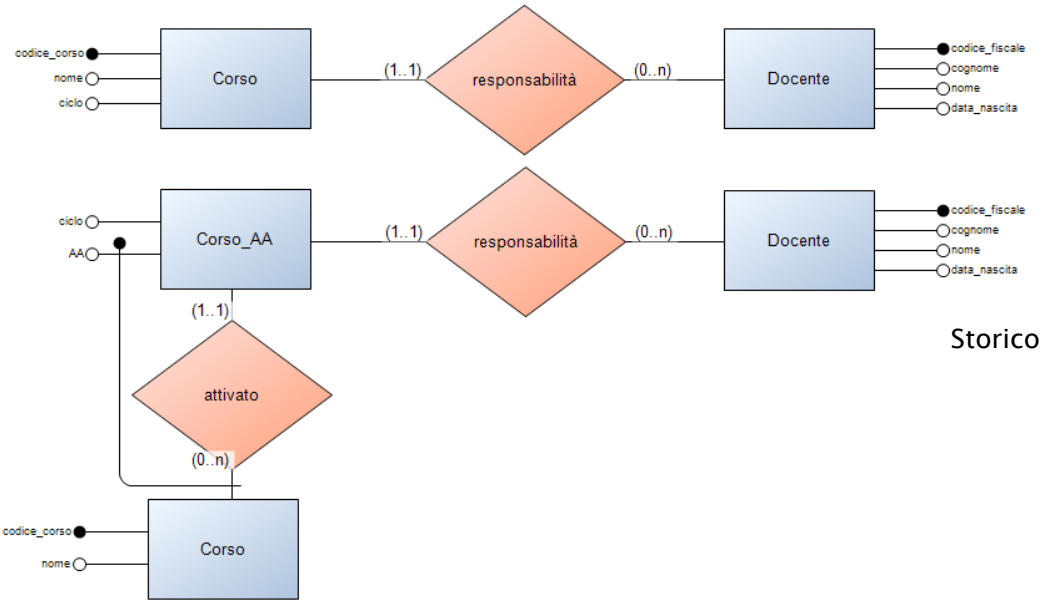


Patterns

Coppie di istanze multiple in date differenti

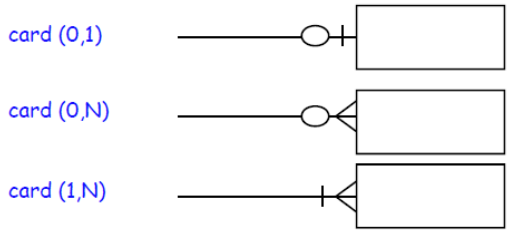


Fotografia

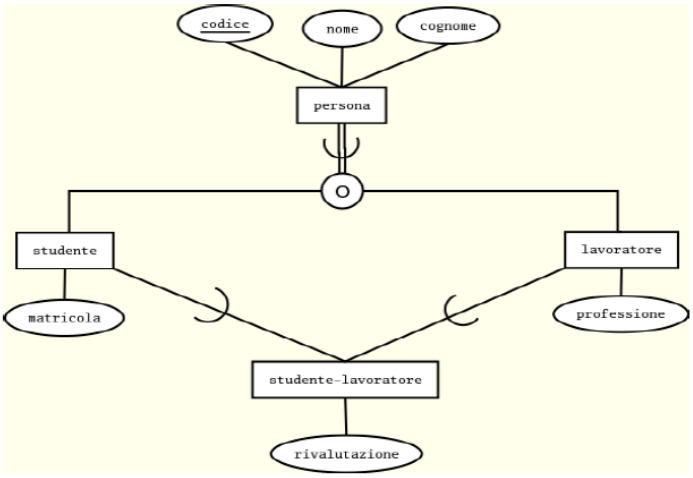


Storico

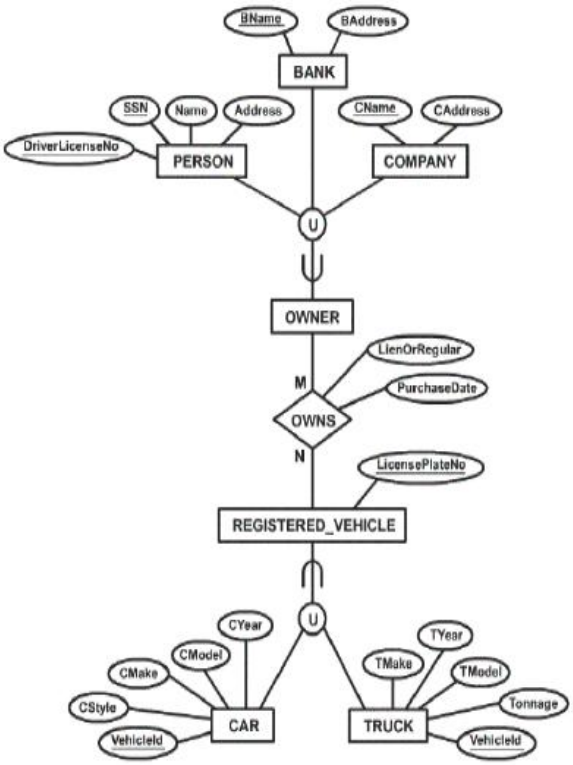
Varianti, estensioni



Vincoli di cardinalità, varianti



Gestione di ereditarietà multipla



Categoria, tipo unione

Utilità e limiti

- ✓ Documentazione
- ✓ Reverse engineering
- ✓ Integrazione di sistemi
- ✘ Utilizzo di nomi non sempre completamente esplicativi
- ✘ Esistenza di vincoli non esprimibili

Modello relazionale

I modelli gerarchici e reticolari fanno uso di puntatori per gestire le relazioni, nel modello relazionale si utilizzano unicamente i **valori**. Una relazione denota una **classe di legami fra entità**.

Relazione matematica: dati due insiemi A e B non vuoti e non necessariamente distinti, ogni sottoinsieme non vuoto del prodotto cartesiano A x B è detto **relazione da A a B**. Un elemento a di A è in relazione con un elemento b di B se la coppia (a, b) appartiene alla relazione.

Una relazione è un insieme di n-ple tutte distinte tra loro, sulle quali non è definito un ordinamento; ogni n-pla è costituta da elementi dei domini secondo un ordinamento specifico.

Definizione formale di relazione
Si indichi con **dom(A)** il dominio dell'attributo A e si consideri un insieme di attributi $X = \{ A_1, A_2, \ldots, A_n \}$.
Una **tupla t** su X è una **funzione** che associa ad ogni $A_i \in X$ un valore di dom(A).
L'istanza di una relazione su X è un insieme di tuple su X.
Lo schema di una relazione su X è dato da un nome (della relazione) R e dall'insieme di attributi X, scritto R(X).

Uno schema di un DataBase relazionale è un insieme di relazioni con nomi distinti. L'istanza di un DataBase è un insieme di istanze di relazioni.

Una **tabella** rappresenta una relazione se:
- ✓ I valori di ciascuna colonna sono tra loro omogenei (definiti sullo stesso dominio)
- ✓ Le righe sono tra loro diverse
- ✓ Le intestazioni di colonne sono tra loro diverse
- ✓ L'ordinamento delle righe è irrilevante
- ✓ L'ordinamento delle colonne è irrilevante.

Il modello relazionale, basato sui valori e non sui puntatori:
- ✓ è **indipendente dalle strutture fisiche** che possono modificarsi dinamicamente
- ✓ rappresenta solo ciò che risulta rilevante dal punto di vista dell'applicazione utente
- ✓ permette una maggiore portabilità dei dati da un sistema ad un altro
- ✓ l'uso di eventuali puntatori a livello fisico è invisibile agli utenti
- ✓ i riferimenti tra i dati nelle relazioni diverse sono rappresentati per mezzo dei valori che compaiono nelle tuple.

Valori, vincoli, chiavi, superchiavi

Il modello relazionale prevede l'utilizzo del valore **null** che denota l'assenza di un valore nel dominio, esso non fornisce alcuna informazione sull'attributo e non è possibile applicarvi operatori di confronto. I valori null sono considerati uguali tra loro. Non è possibile attribuire un valore null ad un campo identificatore (primario).

Vincolo di integrità: proprietà che deve essere soddisfatta da ogni istanza della relazione, è rappresentato da una funzione che associa ad ogni istanza il valore VERO o FALSO.

Vincolo di dominio: definisce i valori ammissibili per un singolo attributo.

Vincolo di tupla: condizione espressa su ciascuna tupla, indipendentemente dalle altre (es. condizioni tra valori di attributi differenti)

Vincolo di chiave: definisce gli attributi che identificano univocamente le tuple, vieta la presenza di tuple distinte con lo stesso valore sugli attributi del vincolo.

Dato uno schema R(X), un **insieme di attributi K ⊆ X** è:

superchiave se e solo se in ogni istanza ammissibile r di R(X) non esistono due tuple distinte t1 e t2 tali che t1[K] = t2[K]

chiave se e solo se è una superchiave minimale, ovvero non esiste K' ⊆ K con K' superchiave.

Una chiave è un identificatore minimale per ogni r su R(X).

Dimostrazione dell'esistenza di chiavi
Poiché ogni istanza r su R(X) è un insieme, ne consegue che l'insieme X di tutti gli attributi dello schema è senza dubbio una superchiave per R(X). Essendo il numero di attributi n finito, è sempre possibile individuare almeno una chiave K ⊆ X; infatti:

```
K = X
for i = 1 to n
{
      if K – { Ai } è superchiave then K = K – { Ai }
}
```

L'esistenza delle chiavi garantisce l'accessibilità ad ogni dato del DataBase: ogni singolo valore è identificato da:

- Nome della relazione
- Valore della chiave
- Nome dell'attributo

Tra le possibili chiavi di una relazione se ne sceglie una detta **chiave primaria**, per convenzione si sottolineano gli attributi che ne fanno parte.

Vincolo di integrità referenziale: vincolo "inter-relazione" che impone che in ogni istanza di una relazione r, l'insieme dei valori Y di un certo attributo sia un sottoinsieme dei valori della chiave primaria di un'altra relazione. L'insieme Y è detto **foreign key** (**chiave importata**).

Livello fisico

Architettura di un DBMS

Unità e abbreviazioni

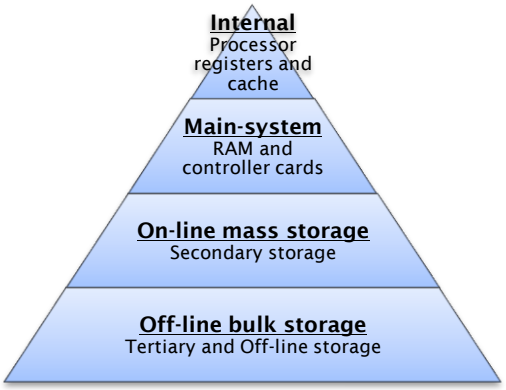
nome	abbreviazione	grandezza
yotta	y,Y	$10^{24} \cong 2^{80}$
zetta	z,Z	$10^{21} \cong 2^{70}$
exa	e,E	$10^{18} \cong 2^{60}$
peta	p,P	$10^{15} \cong 2^{50}$
tera	t,T	$10^{12} \cong 2^{40}$
giga,bilion	g,b,G,B	$10^{9} \cong 2^{30}$
mega	m,M	$10^{6} \cong 2^{20}$
kilo	k,K	$10^{3} \cong 2^{10}$
		$10^{0} \cong 2^{0}$
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}
femto	f	10^{-15}

unità	abbreviazione
bit	b
byte (8 bit)	B
bits per second	bps
Bytes per second	Bps
instructions per second	ibs
I/O operations per second	I/Ops
transactions per second	tps
bits per inch	bpi
rounds per minute	rpm

Memorie e prestazioni

Le prestazioni di una memoria si misurano in termini di **tempo di accesso**:

$$tempo\ di\ accesso = latenza + \frac{dimensione\ dati\ da\ trasferire}{velocità\ di\ trasferimento}$$



I DataBase risiedono solitamente su dischi (a causa delle dimensioni), i dati devono essere trasferiti in memoria centrale per essere elaborati.

Il trasferimento avviene per *blocchi* o *pagine*; blocchi piccoli determinano maggiori operazioni di I/O, blocchi grandi aumentano la frammentazione interna e richiedono maggiore spazio in memoria.

Le operazioni di I/O sono il collo di bottiglia del sistema, l'implementazione fisica del DB deve:

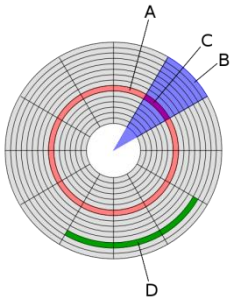
- ✓ Organizzare **efficientemente** le tuple sui dischi
- ✓ Utilizzare strutture di accesso **efficienti**
- ✓ Gestire **efficientemente** i buffer di memoria
- ✓ Utilizzare strategie di esecuzione **efficienti** per le query.

A livello di **applicazione** si opera su record.
A livello di **sistema di archiviazione** di opera su blocchi di byte dimensionati in base al file system.
A livello di **dispositivo** si opera su blocchi di byte di dimensioni scelte dall'utente o fissate dai dispositivi.

Modalità di accesso ai dispositivi:

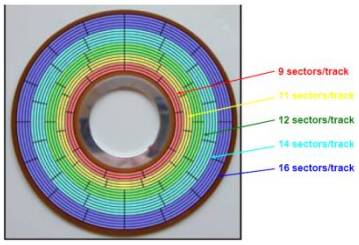
- Sequenziale: i record sono localizzabili solo in sequenza
- Diretto: i record sono localizzabili in base alla loro posizione all'interno dell'archivio
- Associativo: i record sono localizzabili in base al valore di un campo chiave.

Hard Disk



A – Traccia
B – Settore
C – Settore di una traccia
D – Cluster, insieme di settori contigui

La velocità di trasferimento è più elevata nelle zone più esterne , la densità presenta valori più elevati al centro.



Le prestazioni interne sono influenzate da: caratteristiche meccaniche, tecniche di memorizzazione e codifica dei dati, caratteristiche del controller e cache.

Access time = Command Overhead Time + Seek Time + Settle Time + Latency

Command Overhead Time: tempo necessario per impartire comandi al drive **O(0.5ms)**

Seek Time: tempo necessario per spostare le testine sul cilindro desiderato
composto da: speedup (accelerazione), coast (velocità costante), slowdown (rallentamento)

Average seek high-end server drives	**O(3ms)**
Average seek mobile drives	**O(12ms)**
Track-to-track	**O(1ms)**
Full-stroke	**O(15-20ms)**

Average seek distance

$$E[d] = \sum_{d=0}^{n-1} d \times \frac{2 \times (n-d)}{n^2} = \frac{2}{n}\sum_{d=1}^{n-1} d - \frac{2}{n^2}\sum_{d=1}^{n-1} d^2 = \frac{2}{n}\frac{(n-1)\times n}{2} - \frac{2}{n^2}\frac{(n-1)\times\left(n-\frac{1}{2}\right)\times n}{3} =$$

$$= (n-1) - \frac{(n-1)\times(2\times n-1)}{3\times n} = (n-1)\left(\frac{3\times n-(2\times n-1)}{3\times n}\right) = \frac{n^2-1}{3\times n} \approx \boldsymbol{\frac{n}{3}}$$

Modelo per il seek time: $\boldsymbol{t_s(d) = a \times \sqrt{d-1} + b \times (d-1) + c} \quad \boldsymbol{0 < d < N\_Cyl}$

$a :=$ *accelerazione* $\quad b :=$ *fase a velocità costante* $\quad c :=$ *minimo tempo di seek*

Settle Time: tempo necessario per stabilizzare le testine **O(0.1ms)**

Internal Media Transfer Rate = (bytes / sector) x (sectors / track) / rotation time

L'Internal Media Transfer Rate rappresenta la velocità massima alla quale il drive può leggere o scrivere bit (trasferimento dai piatti alla cache o viceversa), tipicamente nell'ordine di qualche centinaio di Mb/s.

Le prestazioni esterne sono influenzate da: tipi di interfacce, architetture del sottosistema I/O, file system.

Sustained Transfer Rate include anche l'overhead per head switch time e cylinder switch time.

Read-ahead: il controller *anticipa* le richieste di lettura caricando nella propria cache il contenuto di una o più tracce.

Command queuing: i driver SCSI permettono la gestione di più richieste in contemporanea da parte del controller e valuta il miglior ordine per servirle.

Write-back: il controller segnala che la richiesta di scrittura è soddisfatta quando i dati sono scritti nella sua cache.

Write-through: il controller segnala che la richiesta di scrittura è soddisfatta quando i dati sono scritti effettivamente su disco.

Per accertarsi che la scrittura sia effettivamente avvenuta utilizzando la tecnica write-back, è possibile effettuare controlli specifici: il controller attende una rotazione (dopo aver scritto un blocco) e rilegge il blocco, se riscontra discordanze ripete la scrittura, eventualmente su un altro disco (read after write).

A livello fisico, un DataBase consiste in un insieme di file, ognuno dei quali è visto come una collezione di pagine di dimensione fissa. I record sono memorizzati nelle pagine; ogni record, consistente di più campi rappresentanti gli attributi, rappresenta una tupla logica. Ogni DBMS adotta soluzioni specifiche a livello fisico.

Modello di memorizzazione di DB2

DB2 organizza lo spazio fisico in **tablespace** ognuno dei quali è una collezione di **container**. I container (tipicamente memorizzati su dischi differenti) sono divisi in **extent** che rappresentano l'unità minima di allocazione su disco (insiemi contigui di pagine di 4 KB).

Ogni relazione è memorizzata su un singolo tablespace. Un extent contiene dati di una singola relazione.

Nei tablespace di tipo **System Managed Space (SMS)** la gestione dello spazio su disco è demandata al sistema operativo. Nei tablespace di tipo **Database Managed Space (DMS)**, la gestione è a carico del DBMS.

All'atto della creazione di un tablespace è possibile specificare una serie di parametri tra cui:

- **EXTENTSIZE**: numero di blocchi dell'extent
- **BUFFERPOOL**: nome del pool di buffer associato al tablespace
- **PREFETCHSIZE**: numero di pagine da trasferire in memoria prima che vengano effettivamente richieste
- **OVERHEAD**: stimal del tempo medio di latenza per un'operazione di I/O
- **TRANSFERRATE**: stima del tempo medio per il trasferimento di una pagine.

Modello di memorizzazione di ORACLE

ORACLE organizza lo spazio in unità logiche di memorizzazione dette **tabelspace** ognuna delle quali consiste di uno o più **datafile** (strutture fisiche, conformi al sistema operativo).

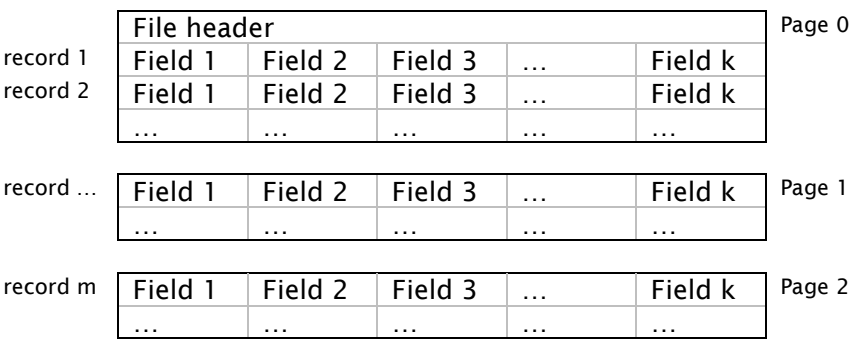
Per espandere gli spazi in ORACLE è possibile:

- *Aggiungere datafile*
- *Aggiungere tablespace*
- *Modificare la dimensione di* un datafile.

Organizzazione dei dati nei file

Le prestazione di un DBMS dipendono *fortemente* dall'organizzazione fisica dei dati su disco (è opportuno conoscere come i dati dovranno essere elaborati e quali sono le correlazioni logiche tra essi). **Queste informazioni non possono essere note al file system**.

Schema di riferimento semplificato



Per ogni record del DataBase deve essere definito uno *schema fisico* che permetta di interpretare il sisgnificato dei byte che costituiscono il record. Nel caso in cui tutti i record abbiano lunghezza fissa, evidentemente, la situazione è più semplice. Nel caso in cui, invece, vi siano record di lunghezza variabile, è necessario adottare strategie per interpretarne correttamente i corrispondenti bytes: una soluzione consolidata consiste nel memorizzare prima tutti i campi a lunghezza fissa poi quelli a lunghezza variabile aggiungendo un campo "*prefix pointer*" che riporta l'indirizzo del primo byte del campo.

I record possono includere un **header** che, oltre alla lunghezza del campo, può contenere:
- ✓ L'identificatore della relazione cui appartiene il record
- ✓ L'identificatore univoco del record nel DataBase
- ✓ Un *timestamp* che indica quando il record è stato inserito o modificato l'ultima volta.

Solitamente un record ha dimensioni molto inferiori a quelle delle pagine. Ogni pagine è dotata di un **page header** che mantiene informazioni specifiche della pagina (ID, timestamp ultima modifica, relazione cui le tuple appartengono, …). Si può avere spreco di spazio (frammentazione interna).

Lunghezza record	Lunghezza pagina	Lunghezza header pagina	Nr. pagine utilizzate	Nr. record memorizzabili	Spazio non utilizzato
296 byte	4096 byte	12 byte	1	13	236 byte
296 byte	4096 byte	12 byte	770	10.000	181.184 byte (su 769 pagine)

Tipicamente una pagina in un DBMS contiene una directory con un puntatore ad ogni record: l'identificazione di un record è effettuata tramite una coppia di valori: PID (page identifier) e Slot (posizione nella pagina).

Lettura e scrittura di pagine

- La lettura di una tupla richiede che la pagina che la contiene sia trasferita in memoria (*buffer pool*)
- Ogni buffer può ospitare una copia di pagina su disco
- La gestione del buffer pool è demandata ad un modulo specifico del DBMS: il *Buffer Manager (BM)*
- BM è utilizzato anche per scrivere su disco pagine modificate ed ha un ruolo fondamentale nella gestione delle transazioni, per garantire integrità a fronte di guasti.

A fronte di una richiesta di pagina:
- ➢ Se la pagina è presente nel buffer, viene restituito al programma chiamante l'indirizzo del buffer.
- ➢ Se la pagina non è in memoria, il *Buffer Manager* individua un buffer ed ne rimpiazza l'eventuale contenuto con la pagina.

Buffer Manager

Il BM presenta, agli altri moduli del DBMS, un'interfaccia composta da quattro metodi:
- ➢ **getAndPinPage**: richiede la pagina al BM la "segna" come *in uso* (apponendovi un *pin*)
- ➢ **unPinPage**: rilascia la pagina e rimuove un *pin*
- ➢ **setDirty**: indica che la pagina è stata modificata
- ➢ **flushPage**: forza la scrittura su disco della pagina.

Le pagine sono rimpiazzate non secondo politiche **LRU** molto usate nei sistemi operativi perché non tiene conto di "pattern di accesso ai dati".

HitRatio: frazione di richieste che non provocano una operazione di I/O, indica quanto è buona una politica di rimpiazza mento.

Cataloghi

Ogni DBMS mantiene **cataloghi**: relazioni che descrivono il DataBase sia a livello logico che fisico e riportano informazioni statistiche sulle relazioni.

Organizzazione dati

Tipi di organizzazione dati

Un'organizzazione primaria impone un criterio di allocazione dei dati, al contrario di un'organizzazione secondaria.

Un'organizzazione dinamica si adatta alla mole effettiva dei dati; un'organizzazione statica prevede fasi periodiche di *riorganizzazione* a fronte di variazioni, più o meno consistenti, dei dati.

Il valore di una chiave primaria identifica un unico record in un'organizzazione dati, mentre quello di una chiave secondaria, identifica più record.

Tipi di operazioni

Le operazioni, in termini di I/O, sono riconducibili ad alcune *primitive di base*:

- ricerca
  - effettuata per *chiave primaria* restituisce al più un solo record
  - effettuate per *chiave secondaria* restituisce 0 o più record
  - effettuata per *intervallo* restituisce 0 o più record
- inserimento di uno o più record
- cancellazione di uno o più record
- modifica di uno o più record.

Una **transazione** è un'insieme di operazioni elementari che devono essere eseguite per soddisfare una richiesta.

Clustering e indexing

Clustering: indica la presenza di *addensamenti* di dati nei blocchi dei file (l'ordinamento è un caso particolare di clustering); è importante per ricerche su chiave secondaria e può essere indotto da dipendenze tra gli attributi.

Indexing: riguarda aspetti relativi all'accesso ai dati, ovvero la possibilità di risolvere efficacemente operazioni di ricerca e aggiornamento.

I file sono organizzazioni di dati che può essere *strutturata* (rappresentando collezioni di record appartenenti) oppure *non strutturata* (sequenza di byte, stream, su cui è possibile operare tramite primitive orientate alla manipolazione di byte o blocchi.

Organizzazione sequenziale

Disponibile su nastri e dischi, l'organizzazione sequenziale prevede che i record sono scritti in ordine in base ad un campo chiave o, più semplicemente, in base all'ordine temporale di registrazione. La lettura deve essere fatta seguendo lo stesso ordine della scrittura.

Le organizzazioni sequenziali sono utili se:

- si gestiscono piccoli volumi di dati
- si effettuano operazioni che riguardano tutti (o gran parte) dei record
- si effettuano aggiornamenti raramente.

Ricerca per chiave primaria

Se il file non è ordinato, in presenza di NP blocchi, ogni blocco ha probabilità 1/NP di ospitare il record desiderato.

Caso medio:	Caso peggiore:	Caso di non esistenza:
$\sum_{j=1}^{NP}\left(j \times \frac{1}{NP}\right) = \frac{1+2+\ldots+NP}{NP} = \frac{NP \times (NP+1)}{2\,NP} = \frac{NP+1}{2}$	NP	NP

Se il file è ordinato, in presenza di NP blocchi, ogni blocco ha probabilità 1/NP di ospitare il record desiderato.

Caso medio:	Caso peggiore:	Caso di non esistenza:
$\sum_{j=1}^{NP}\left(j \times \frac{1}{NP}\right) = \frac{1+2+\ldots+NP}{NP} = \frac{NP \times (NP+1)}{2\,NP} = \frac{NP+1}{2}$	NP	$\frac{NP+1}{2}$

Organizzazione ad accesso diretto

Consente di indirizzare ogni record tramite un numero (da 0 a N-1, con N record). Il tipo di dati astratto corrispondente è l'array. Le primitive di accesso disponibili sono:

- **seek(i)**: per posizionarsi sul record di indirizzo logico i
- **getNext:** per proseguire nella sequenza logica degli indirizzi.

Il file system opera una mappatura tra indirizzi logici di record e indirizzi di blocco.

Se il file non è ordinato, in presenza di NP blocchi, ogni blocco ha probabilità 1/NP di ospitare il record desiderato.

Caso medio:	Caso peggiore:	Caso di non esistenza:
$\sum_{j=1}^{NP}\left(j \times \frac{1}{NP}\right) = \frac{1+2+\ldots+NP}{NP} = \frac{NP \times (NP+1)}{2\,NP} = \frac{NP+1}{2}$	NP	NP

Se il file è ordinato, in presenza di NP blocchi, ogni blocco ha probabilità 1/NP di ospitare il record desiderato, è possibile operare ricerca dicotomica.

Caso medio:	Caso peggiore:	Caso di non esistenza:
$\lfloor log_2 NP \rfloor$	$\lfloor log_2 NP \rfloor + 1$	$\lfloor log_2 NP \rfloor + 1$

Fusione di archivi ordinati

L'operazione di *fusione* di due o più archivi presuppone che gli archivi di input e quello di output siano ordinati secondo un criterio comune.

Algoritmo di fusione

1. si legge il primo record da ogni archivio e lo si inserisce nell'insieme dei record correnti RC
2. do
    a. si scrive in output il record R con il più piccolo valore di chiave tra quelli presenti in RC
    b. si elimina R da RC
    c. si legge un record dall'archivio da cui è stato scelto il record R e lo si inserisce in RC
3. while RC != ∅

L'ordinamento esterno richiede l'utilizzo di algoritmi di ordinamento interno e fusione: infatti, se si vuole ordinare un archivio composto da N record, avendo a disposizione una memoria centrale capace di ospitare M < N record, si può procedere

1. ordinando internamente i record a gruppi di M, producendo [N/M] *sottoarchivi* (detti *sequenze*, *run*)
2. fondendo gli [N/M] *sottoarchivi* archivio.

Sort-merge orientato ai record

Caso: $M < N \leq M^2$	Caso: $N > M^2$
Sort interno: $\lceil N/M \rceil \leq M$ *run* *Merge*: le *run* sono $\leq M$ è sufficiente un unico passo (ciclo) per effettuare il merge	*Sort interno*: $\lceil N/M \rceil > M$ *run* *Merge*: le *run* sono $> M$, sono necessari più passi (cicli) per effettuare il merge
Costo: $N + 3 \times NP$	Costo: $2 \times NP + (N + NP) \times \left\lceil \log_M \frac{N}{M} \right\rceil$ *Nota: l'algoritmo non sfrutta adeguatamente l'organizzazione a blocchi dei record.*

L'algoritmo sort-merge a Z vie, organizza la memoria centrale in Z pagine logiche (**frame**) di capacità M/Z record; ogni frame corrisponde a FS≥1blocchi del file. La fusione è effettuata considerando Z *run* alla volta.

Costo sort-merge a Z vie: $2 \times NP \times \left(1 + \left\lceil \log_Z \frac{NP}{Z \times FS} \right\rceil\right) = 2 \times NP \times \left\lceil \log_Z \frac{NP}{FS} \right\rceil$

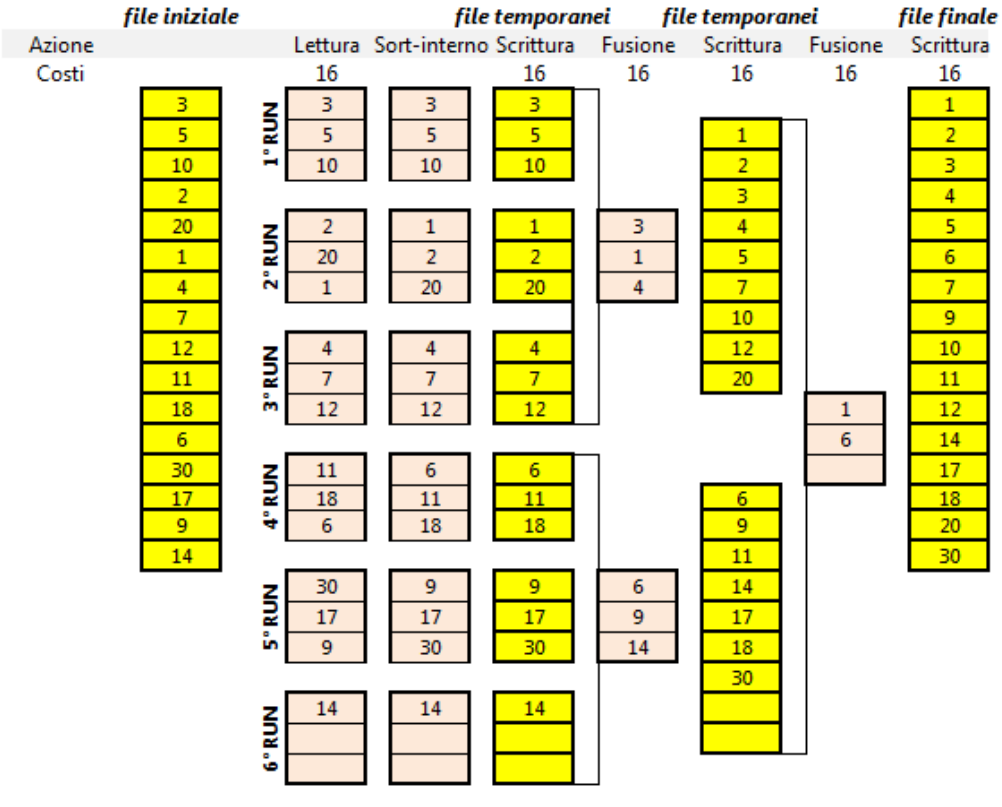
Passi di fusione: $PF = \left\lceil \log_Z \frac{N}{M} \right\rceil$

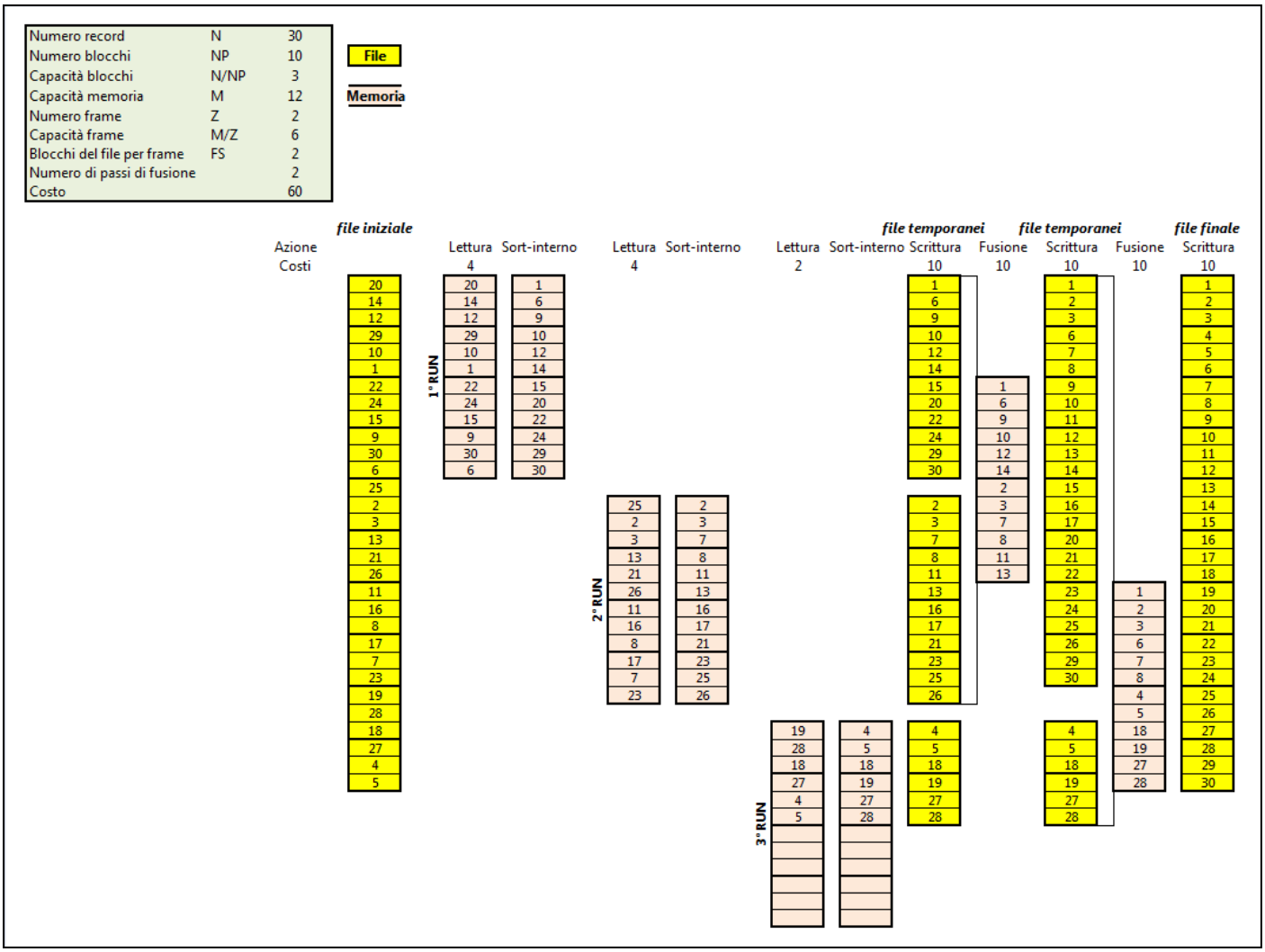
A parità di M è conveniente scegliere Z massimo, con FS = 1: $2 \times NP \times \left\lceil \log_Z \frac{NP}{FS} \right\rceil = 2 \times NP \times \lceil \log_Z NP \rceil$

Numero record	N	16
Numero blocchi	NP	16
Capacità blocchi	N/NP	1
Capacità memoria	M	3
Numero frame	Z	3
Capacità frame	M/Z	1
Blocchi del file per frame	FS	1
Numero di passi di fusione		2
Costo		96

File

Memoria





Cammini di accesso, indici

Per accedere rapidamente ai record di un file, tramite ricerca dicotomica su una chiave di ricerca, sarebbe necessario mantenere ordinato il file secondo i valori della chiave. In questo caso, comunque, i costi di ricerca sarebbero comunque elevati per file di dimensioni elevati ed estremamente inefficienti sarebbero le ricerche su campi diversi dalla chiave.

Per risolvere questi problemi si fa uso di:

- **organizzazioni primarie**: organizzazioni ad albero o basate su tecniche hash dei file dati
- **organizzazioni secondarie**: indici (separati dai file dati) organizzati ad albero (o hash).

Cammino di accesso: una struttura che rende efficiente le ricerca di particolari record della base di dati.

Indice: cammino di accesso che garantisce l'accesso diretto ai dati attraverso un termine *indice* o una parola chiave. L'idea di base di un indice consiste nella realizzazione di una *tabella* in cui sono memorizzate coppie di valori k,p che identificano (k) un valore del campo su cui l'indice è costruito e (p) un *riferimento* al record con quel valore chiave.

L'accesso tramite indice consiste in:

1. accesso all'indice
2. ricerca della coppia di valori (k,p) con chiave k = valore dato
3. conversione del riferimento p in un indirizzo assoluto
4. accesso al blocco con l'indirizzo assoluto ottenuto.

Accedere ad un record, facendo uso di ricerca binaria sull'indice anziché sul file dati, determina un risparmio, infatti siano:

- NP i blocchi in cui è memorizzato il file dati
- C il numero di record per ciascun blocco
- IP i blocchi in cui è memorizzato il file indice
- IC il numero di chiavi memorizzate per ciascun blocco

Da cui: **NP x C = IP x IC**

Costo ricerca binaria sul file ordinato: $\log_2 NP$

Costo ricerca binaria sull'indice ordinato: $\log_2 IP + 1 \ \ (accesso\ al\ blocco\ dati)$

Risparmio:

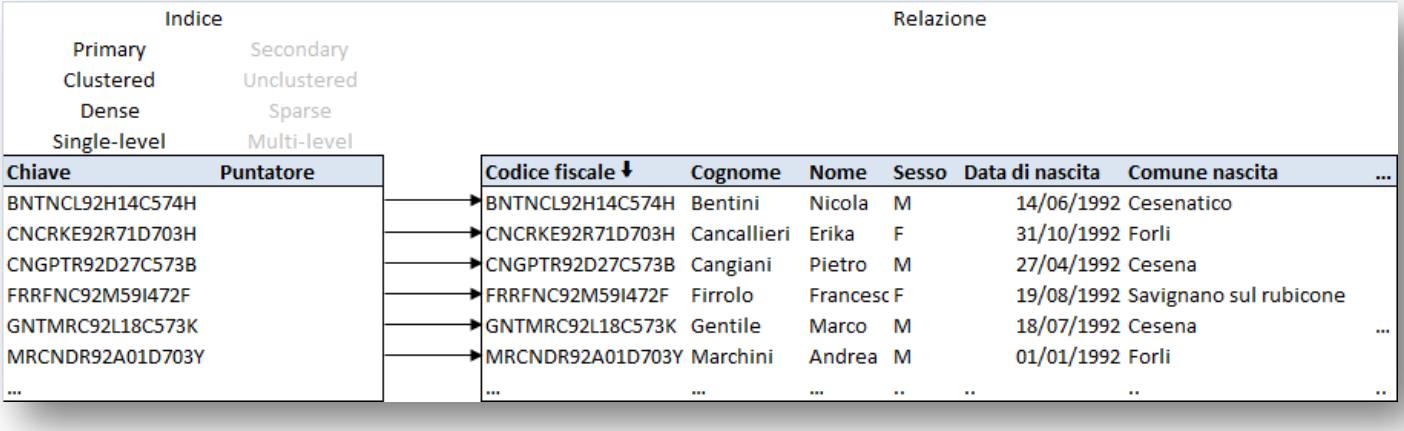
$$\log_2 NP - (\log_2 IP + 1) = \log_2 NP - \log_2 \frac{NP \times C}{IC} - 1 = \log_2 NP - \log_2 NP - \log_2 \frac{C}{IC} - 1 = -\log_2 \frac{C}{IC} - 1 = \log_2 \frac{IC}{C} - 1$$

Classificazione di indici

Criterio di classificazione	Classificazione	Descrizione
Unicità dei valori di chiave	Primary index	*indice su un attributo che assume valori unici*
	Secondary index	*indice su un attributo che può assumere valori ripetuti*
Ordinamento del file dati	Clustered index	*indice su un attributo secondo cui il file dati è mantenuto ordinato*
	Unclustered index	*indice su un attributo secondo cui il file dati non è mantenuto ordinato*
Numero di coppie dell'indice	Dense index	*indice in cui il numero di coppie (k,p) è pari al numero di record dati*
	Sparse index	*indice in cui il numero di coppie (k,p) è minore al numero di record dati*
Numero di livelli dell'indice	Single-level index	*indice organizzato in modo "flat"*
	Multi-level index	*indice organizzato in più livelli (albero)*

Possono esistere tutte le combinazioni delle 8 classificazioni di indice, anche se una soluzione **sparse & unclustered** non ha solitamente senso (esiste *partial index* per indice secondario, utile se si escludono dall'indice i valori chiave molti ripetuti per i quali l'uso dell'indice stesso può rivelarsi inutile o controproducente).

Di seguito alcuni esempi



Chiave	Puntatore
BNTNCL92H14C574H	
CNCRKE92R71D703H	
CNGPTR92D27C573B	
FRRFNC92M59I472F	
GNTMRC92L18C573K	
MRCNDR92A01D703Y	
...	

Codice fiscale	Cognome	Nome	Sesso	Data di nascita	Comune nascita	...
GNTMRC92L18C573K	Gentile	Marco	M	18/07/1992	Cesena	...
FRRFNC92M59I472F	Firrolo	Francesc	F	19/08/1992	Savignano sul rubicone	
MRCNDR92A01D703Y	Marchini	Andrea	M	01/01/1992	Forli	
CNCRKE92R71D703H	Cancallieri	Erika	F	31/10/1992	Forli	
CNGPTR92D27C573B	Cangiani	Pietro	M	27/04/1992	Cesena	
BNTNCL92H14C574H	Bentini	Nicola	M	14/06/1992	Cesenatico	
...	...	...	..	..	..	..

Chiave	Puntatore
Cesena	
Cesenatico	
Forli	
Savignano sul rubicone	
...	
...	
..	

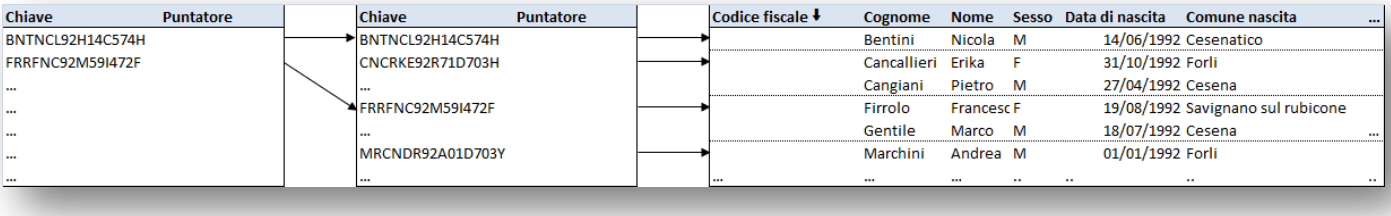
Codice fiscale	Cognome	Nome	Sesso	Data di nascita	Comune nascita	...
GNTMRC92L18C573K	Gentile	Marco	M	18/07/1992	Cesena	...
FRRFNC92M59I472F	Firrolo	Francesc	F	19/08/1992	Savignano sul rubicone	
MRCNDR92A01D703Y	Marchini	Andrea	M	01/01/1992	Forli	
CNCRKE92R71D703H	Cancallieri	Erika	F	31/10/1992	Forli	
CNGPTR92D27C573B	Cangiani	Pietro	M	27/04/1992	Cesena	
BNTNCL92H14C574H	Bentini	Nicola	M	14/06/1992	Cesenatico	
...	...	...	..	..	..	..

Il valore "p" dell'indice può far riferimento al numero del blocco o al numero del record nel file dati: il puntatore a blocchi determina un indice più piccolo (quindi più rapido da "navigare") ma implica una successiva ricerca all'interno del blocco (o dei blocchi, e la valutazione di esistenza del record), il puntatore al record determina indici più grandi ma identifica il record esatto.

Organizzazioni notevoli

PISM (Pure Indexed Sequential Method; heap+indice), **ISAM** (Indexed Sequential Access Method; IBM), **VSAM** (Virtual Storage Access Method; IBM, evoluzione di ISAM) e **UFAS** (Regular Indexed Sequential; Bull) sono alcune organizzazioni notevoli per indici *primary, clistere, sparse, multi-level*; in esse l'indice può essere parte integrante del file dati (i blocchi indici sono allocati in maniera dipendente dai blocchi dati).

ISAM è un'organizzazione statica soggetta a costose riorganizzazioni periodiche, in fase di inserimento record lascia spazi liberi per ulteriori inserimenti e, in caso di spazio non disponibile fa uso di aree di overflow. In ogni coppia (k, p) dell'indice k è il valore di chiave più basso individuato tra quelli appartenenti al sottoalbero individuato da p.



Indici multilivello

In memoria secondaria, gli indici sono solitamente organizzati in più livelli per motivi di efficienza. Essi devono soddisfare i requisiti di:

- ✓ **Bilanciamento**: l'indice deve essere bilanciato considerando i blocchi, in quanto è l'accesso ai blocchi a determinare i costi di I/O
- ✓ **Occupazione minima**: è necessario individuare un limite inferiore di utilizzo dei blocchi per evitare eccessivo spreco di memoria
- ✓ **Efficienza di aggiornamento**: deve essere garantito un costo limitato per le operazioni di aggiornamento.

Indici multilivello: B-tree (Bayer, McCreight 1972)

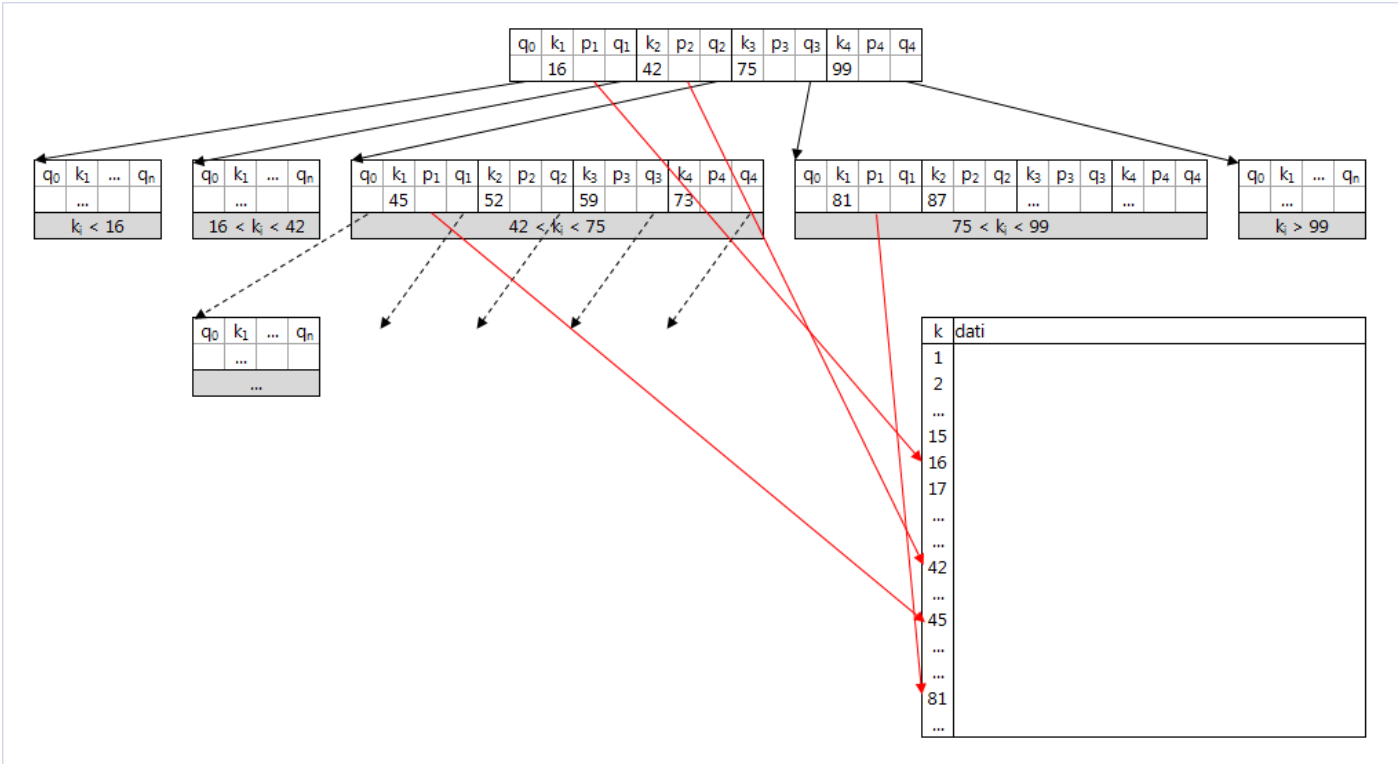
B-tree: famiglia di indici multilivello che soddisfa i tre requisiti ("B" indica Balanced, Bayer o Boeing – la compagni per la quale gli autori lavoravano?). Appartengono a tale famiglia: B-tree, B*-tree, B^+-tree

Indice B-tree: albero (direzionato) a più vie *perfettamente bilanciato organizzato a nodi* che corrispondono a blocchi su disco.

Siano **g**, **h** > 0 due numeri naturali. **g** è detto *ordine*. **h** è detta *altezza*.
Un B-tree della classe **τ(g,h)** rispetta le seguenti proprietà:

- ✓ Ogni percorso dalla *radice* ad una *foglia* ha sempre la stessa lunghezza **h**
- ✓ Ogni nodo, escluse radice e foglie, ha un numero di figli compresi tra **g+1** e **2g+1**
- ✓ La radice ha almeno 2 figli (oppure è una foglia: **h** = 1)

- ✓ La radice memorizza tra **1** e **2g** chiavi
- ✓ Ogni nodo memorizza tra **g** e **2g** chiavi
- ✓ Ogni nodo (non foglia) che memorizza **L** chiavi (**g** ≤ **L** ≤ **2g**) ha **L+1** puntatori ad altrettanti nodi figli:
- ✓ Il primo puntatore indirizza il nodo figlio che contiene valori di chiave inferiori alla prima chiave presente nel nodo, per ogni chiave k_i è presente:
  - un puntatore al record con chiave k_i (eventualmente deve essere gestita la molteplicità per chiavi ripetute)
  - un puntatore al blocco contenente le chiavi maggiori di k_i e inferiori di k_{i+1}
- ✓ In ogni nodo le chiavi sono memorizzate in ordine crescente.



Algoritmo di ricerca in un B-tree

```
function searchBtree(y, out found, out q, out s) {
  q = root
  s = nil                // s: puntatore per l'inserimento
  found = false
  while ( q ≠ nil ) and ( !found ) do
    s = q
    if y < k1 then q = q0
    else if ∀ i ( y = ki ) then found = true  //trova la chiave
    else if ∀ i ( ki < y < ki+1 ) then q = qi   //si sposta al sottoalbero
    else q = ql                                 // l: ultima chiave
}
```

Modifiche in un B-tree

Le modifiche partono sempre dalle foglie e si dice che i B-tree "crescono verso l'alto".

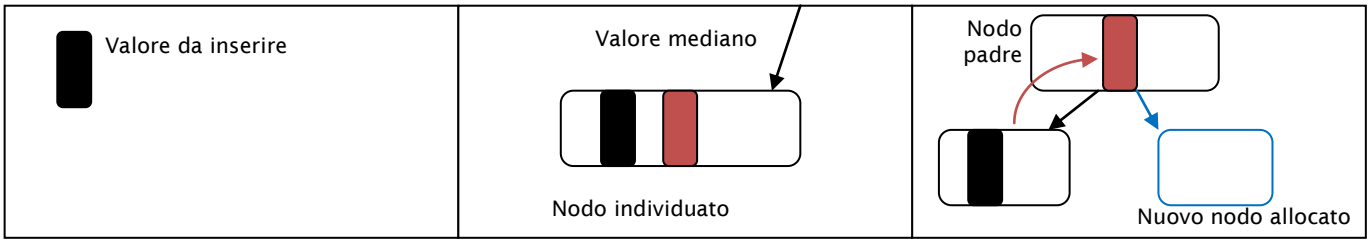
Inserimento

1. si effettua una ricerca per verificare la presenza della chiave nell'albero
2. in caso di assenza (con vincolo di valori non duplicati) si inserisce la chiave in una foglia
   2.1. se la foglia non è piena si aggiorna la foglia
   2.2. se la foglia è piena si attiva un processo di *splitting* che può essere ricorsivo e può propagarsi, verso l'alto, anche fino alla radice

```
function insertBtree(y) {
  searchBtree(y, out found, out q, out s);
  if (!found) then
    if ( s = nil ) then crea radice con y
    else if ( P(s) è pieno ) then splitting()
         else inserisci(y, p, nil) in P(s)
  else {...} // caso di ammissione duplicati non espresso
}
```

Algoritmo di *splitting*

1. Se il nodo è pieno
   1.1. Si individua il valore di chiave mediano tra tutti i valori del nodo individuato incluso il valore che si sta inserendo
   1.2. Si inserisce il valore mediano nel nodo padre del nodo individuato
   1.3. Si alloca un nuovo nodo e si ripartiscono i valori delle chiavi tra il nodo individuato e il nodo allocato
   1.4. Si aggiornano i puntatori del nodo padre
   1.5. Si applica l'algoritmo di *splitting* al nodo padre.



Un B-tree che gestisce l'overflow adotta una strategia che tende a generare alberi con nodi più pieni a fronte di maggiori costi di inserimento: nel caso in cui una foglia sia piena, anziché effettuare uno splitting in ogni caso, si ridistribuiscono i puntatori tra il nodo e i suoi adiacenti.

Eliminazione

Se la chiave da cancellare si trova in una foglia la si rimuove, altrimenti la si rimpiazza con il valore di chiave più piccolo del suo sottoalbero di destra.

L'eliminazione può richiedere una fase di *concatenation* quando due nodi adiacenti hanno complessivamente meno di **2g** chiavi. Come lo *splitting* anche la *concatenantion* può propagarsi fino alla radice.

Se, prima della cancellazione, la somma del numero delle chiavi nei due nodi adiacenti è maggiore di **2g** allora le chiavi possono essere ridistribuite tra i due nodi e il nodo padre. Tale processo di *under flow* non si propaga in quando il numero di chiavi del nodo padre non si propaga.

Prestazioni di un B-tree

Le prestazioni dei B-tree dipendono da diversi fattori: altezza, ordine, gestione della cache, caratteristiche del dispositivo, gestione della concorrenza, …

Gestione della cache: un buon uso della cache determina un minor numero di letture/scritture da/verso il dispositivo; è auspicabile che un nodo esaminato durante un'operazione sia letto/scritto una sola volta. Con altezza **h** è sufficiente prevedere in memoria centrale un buffer per **h+1** nodi.

NR	numero di record
NP	numero di blocchi del file dati
NK	numero dei valori distinti di chiave dell'attributo utilizzato (se l'attributo è chiave primaria allora NK = NP)
IP	numero di nodi dell'indice
NL	numero di nodi foglia

B-tree complexity in big O notation		
Invented	1972 by Rudolf Bayer, Edward M. McCreight	
	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(log *n*)
Insert	O(log *n*)	O(log *n*)
Delete	O(log *n*)	O(log *n*)

Conteggi

Numero minimo di nodi

Radice + 2 nodi figli della radice + (g+1) nodi figli di ogni nodo non radice e non foglia.
Al livello 0 c'è solo la radice
Al livello 1 ci sono 2 *nodi*
Al livello 2 ci sono $2 \times (g+1)$ *nodi*
Al livello 3 ci sono $2 \times (g+1)^2$ *nodi*
…
In generale al livello $i > 0$ *ci sono* $2x(g+1)^{i-2}$ *nodi.*
Quindi, il numero minimo di nodi di un B-tree di altezza h e ordine g è:

$$IP_{min} = 1 + 2\sum_{i=0}^{h-2}(g+1)^{i-2} = 1 + 2\frac{(g+1)^{h-1}-1}{(g+1)-1} = 1 + \frac{2}{g}((g+1)^{h-1}-1)$$

Numero massimo di nodi

$$IP_{max} = 1 + 2\sum_{i=0}^{h-1}(2g+1)^{i-2} = \frac{1}{2g}((2g+1)^h - 1)$$

Numero minimo di chiavi

$$NK_{min} = 1 + g \times (IP_{min} - 1) = 1 + g \times \left(\frac{2}{g}((g+1)^{h-1}-1)\right) = 1 + 2((g+1)^{h-1}-1) = 2(g+1)^{h-1} - 1$$

Numero massimo di chiavi

$$NK_{max} = 2g \times IP_{max} = 2g \times \frac{1}{2g}((2g+1)^h - 1) = (2g+1)^h - 1$$

Altezza

$$\left\lceil \boldsymbol{log_{2g+1}(NK+1)} \right\rceil \leq \boldsymbol{h} \leq \left\lfloor \boldsymbol{1 + log_{g+1}\left(\frac{NK+1}{2}\right)} \right\rfloor$$

Costi

Operazione	Migliore	Peggiore	Medio
Inserimento	$h+1$	$3h+1$	$< h+1+\frac{2}{g}$
	h letture + 1 scrittura	h letture + (2h+1) scritture	
Eliminazione	$h+1$	$3h$	$< h+5+\frac{3}{g}$
	h letture + 1 scritture	(2h-1) letture + h+1 scritture	
Gestione overflow			$h+5+\frac{4}{g}$

	Re-trieval	Insertion in index without deletions and without overflows	Deletion in index without insertions, with or without overflows	Insertion in index without deletions, but with overflow	Insertion in index with deletions, without overflow	Deletion in index with insertions, with or without overflows	Insertion in index with deletion, with overflow
min	$f=1$ $w=0$	$f=h$ $w=1$	$f=h$ $w=1$	$f=h$ $w=1$	$f=h$ $w=1$	$f=h$ $w=1$	$f=h$ $w=1$
Average as derived in paper	$f \leqq h$ $w=0$	$f=h$ $w<1+\frac{2}{k}$	$f<h+1+\frac{1}{k}$ $w<4+\frac{2}{k}$	$f \leqq h+2+\frac{2}{k}$ $w \leqq 3+\frac{2}{k}$	$f=h$ $w \leqq 2h+1$	$f \leqq 2h-1$ $h-1 \leqq u \leqq h+1$	$f \leqq 3h-2$ $w \leqq 2h+1$
max	$f=h$ $w=0$	$f=h$ $w=2h+1$	$f=2h-1$ $w=h+1$	$f=3h-2$ $w=2h+1$	$f=h$ $w=2h+1$	$f=2h-1$ $w=h+1$	$f=3h-2$ $w=2h+1$

Scelta dell'ordine

il numero di nodi e l'altezza (h) tendono a diminuire con conseguente riduzione dei costi delle varie operazioni; aumenta però la dimensione di un nodo e quindi il relativo costo di trasferimento.

Un semplice modello che valuta solo il tempo di I/O

$$\boldsymbol{T_{I/O} \approx (t_s + t_r) + \frac{2g \times t_b}{2g_1}} \quad con \quad \begin{array}{l} t_s + t_r : latenza \\ t_b : tempo\ di\ trasferimento\ di\ un\ blocco \\ g_1 : ordine\ nel\ caso\ nodo = blocco \\ \frac{2g}{2g_1} : nr.blocchi/nodo\ in\ un\ B-tree\ di\ ordine\ g \end{array}$$

Il numero medio di pagine lette o scritte per ogni operazione è proporzionale ad h, il tempo totale per una operazione ($T(op)$) può essere espresso come:

$$\boldsymbol{T(op) \propto h \times T_{I/O} \approx h \times (t_s + t_r) + \frac{2g \times t_b}{2g_1}}$$

Approssimando l'altezza h con $\log_{2\beta g+1}(NK+1)$ con β fattore di utilizzo d un nodo ($0.5 \leq \beta \leq 1$), si ha:

$$\boldsymbol{T(op) \propto \log_{2\beta g+1}(NK+1) \times (t_s + t_r) + \frac{2g \times t_b}{2g_1}}$$

L'ordine ottimale dipende allora dalle caratteristiche del dispositivo di memoria secondaria e delle chiavi; il tempo minimo si ottiene scegliendo g tale che:

$$\boldsymbol{\frac{t_s + t_r}{t_b} 2g_1 = \frac{(2\beta g + 1) \times \ln(2\beta g + 1)}{\beta} - 2g = f(g, \beta)}$$

Esempio
Dato un disco con $t_s = 11\ msec$ $t_r = 6\ msec$ $t_b = 0.5\ msec$ $(blocchi\ da\ 1KB)$ e supponendo $g_1 = 20$ si ottiene:

$$\boldsymbol{\frac{11+6}{0.5} \times 2 \times 20 = 1360}$$

Da cui: g ≈ 160 e dimensione dei blocchi = 8 nodi.

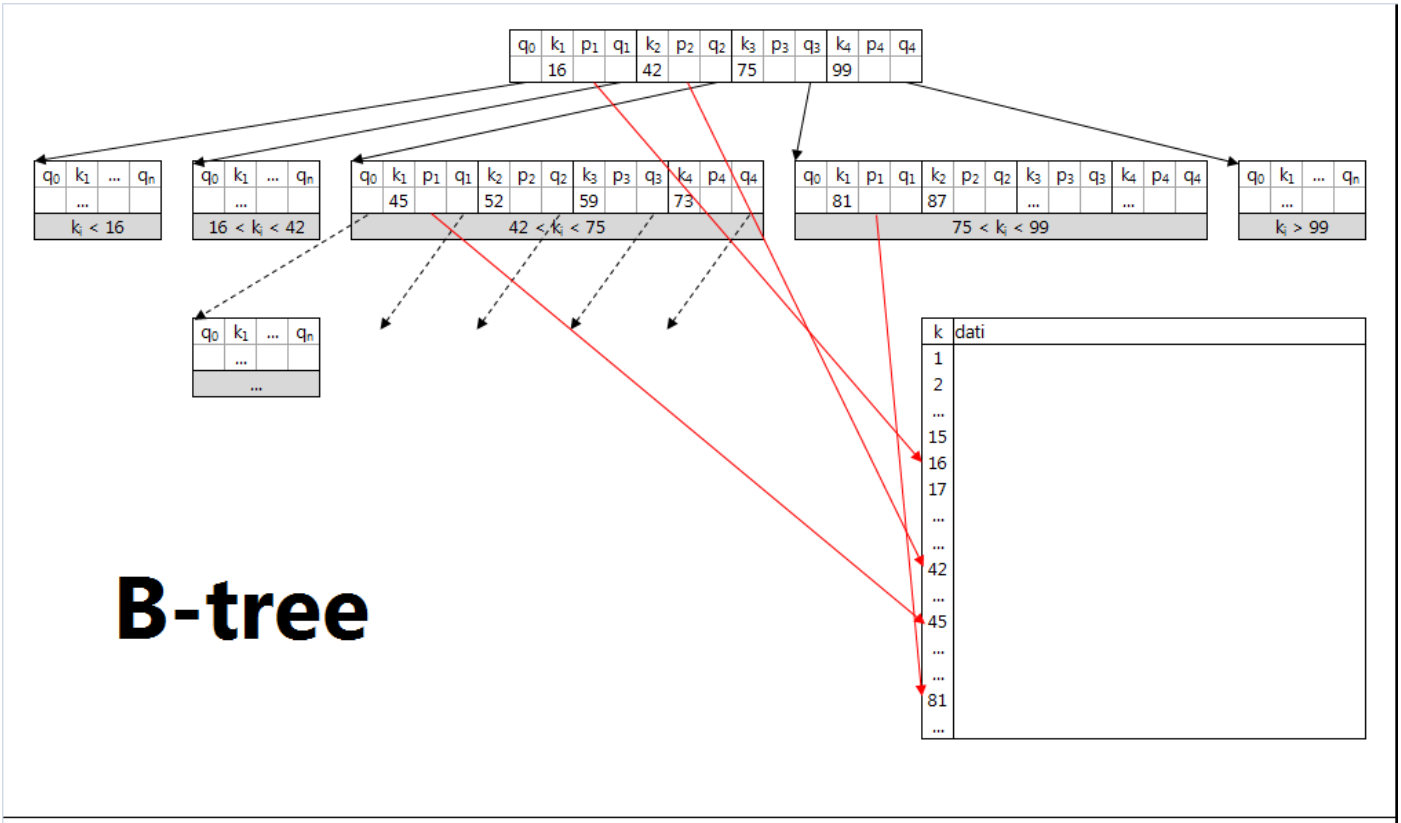
- Un B-tree è molto efficiente per ricerche e modifiche di singoli record.
- Esiste un limite inferiore di utilizzo della memoria (50%) con un utilizzo medio del 69%.
- Un B-tree non è adatto per elaborazione di tipo sequenziale nell'ordine dei valori di chiave e nel reperimento di valori di chiave in un intervallo dato.
- La ricerca del successore può comportare la scansione di molti nodi.
- La ricerca del valore più piccolo (nella foglia più a sinistra) comporta l'accesso a tutti i nodi del percorso tra la radice e la figlia.

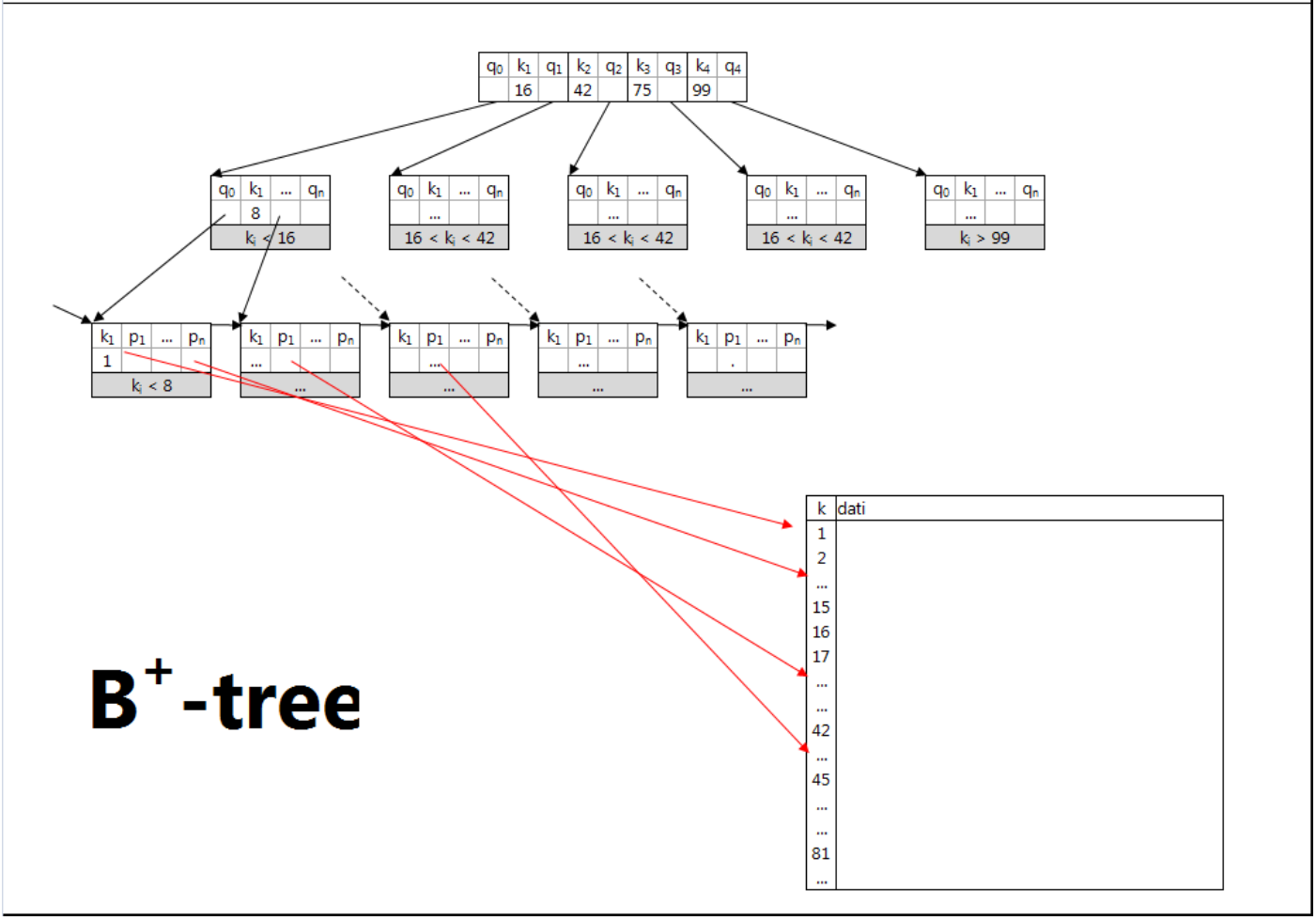
Un B*-tree è una variante dei B-tree in cui l'utilizzo dei nodi è pari almeno a 2/3 anziché 1/2; utilizzano uno schema di ridistribuzione locale allo scopo di ritardare lo *splitting*.

B⁺-tree

A differenza dei B-tree che utilizzano i valori di chiave sia come *separatori* (permettendo di individuare il cammino da seguire in fase di ricerca) che come *puntatori* (permettendo di accedere ai dati), nel B⁺-tree tali funzioni sono mantenute separate:

- Le foglie contengono tutti i valori di chiave
- I nodi interni, organizzati come un B-tree, costituiscono solo la mappa per consentire la rapida localizzazione delle chiavi
- Inoltre, al fine di facilitare elaborazioni sequenziali (e su intervalli), le foglie sono tra loro concatenate in una lista ed è presente un puntatore alla testa della lista.


q0 k1 p1 q1 k2 p2 q2 k3 p3 q3 k4 p4 q4
16
42
75
99
q0 k1 ... qn
ki < 16
16 < ki < 42
45
52
59
73
42 < ki < 75
81
87
75 < ki < 99
ki > 99
k
dati
1
2
15
16
17
42
45
81
B-tree


q0 k1 q1 k2 q2 k3 q3 k4 q4
16
42
75
99
q0 k1 ... qn
8
ki < 16
16 < ki < 42
ki > 99
k1 p1 ... pn
1
ki < 8
k
dati
1
2
15
16
17
42
45
81
B+-tree

I nodi interni sono formati da una sequenza di puntatori a sottoalberi (q_r) alternati a separatori (s_r); $r \leq 2g$.

I nodi foglia sono formati da una sequenza di valori di chiave (k_t) alternati a puntatori ai record (p_t); $t \leq 2g$. Ogni nodo foglia contiene anche un puntatore al nodo foglia successivo.

La funzione dei *separatori* è di determinare il giusto cammino quando si cerca un valore di chiave (non assumono necessariamente valori di chiave presenti nei dati). In caso di chiavi alfanumeriche, la scelta dei separatori è molto importante i quanto può ridurre l'altezza dell'albero (scegliendo ad esempio separatori di lunghezza ridotta; cfr Simple Prefix B⁺-tree, Prefix B⁺-tree.

L'**ordine** nei B⁺-tree è un concetto significativo solo se si fa uso di separatori a lunghezza fissa, negli altri casi si può effettuare un'approssimazione alla lunghezza media dei separatori. Nel primo caso valgono le considerazioni fatte per i B-tree. Invece: se la dimensione di un nodo è fissata in **D** byte, se ogni puntatore **q** richiede **len(q)** byte e se i separatori sono i valori di chiave di lunghezza **len(k)** byte, allora, poiché

$$\boldsymbol{2g \times len(k) + (2g + 1) \times len(q) \leq D}$$

si deriva che l'ordine di un B+-tree è

$$\boldsymbol{g = \left\lfloor \frac{D - len(q)}{2\left(len(k) + len(q)\right)} \right\rfloor}$$

Esempio
D = 4096 byte; len(q) = 4 byte; len(k) = 10 byte → $g = \left\lfloor \frac{4096-4}{2(10+4)} \right\rfloor = \left\lfloor \frac{4092}{28} \right\rfloor = \lfloor 146.14 \dots \rfloor = 146$
Per ogni blocco si ha uno *spreco* di 4 byte: $2 \times 146 \times 10 + (2 \times 146 + 1) \times 4 = 2920 + 1172 = 4092 \leq 4096$

Esempio
D = 4096 byte; len(q) = 4 byte; len(k) = 40 byte → $g = \left\lfloor \frac{4096-4}{2(40+4)} \right\rfloor = \left\lfloor \frac{4092}{88} \right\rfloor = \lfloor 46.5 \rfloor = 46$
Per ogni blocco si ha uno *spreco* di 44 byte: $2 \times 46 \times 40 + (2 \times 46 + 1) \times 4 = 3680 + 372 = 4052 \leq 4096$

In un B⁺-tree il numero di foglie dipende dal numero di record (NR) presenti nel file, dalla dimensione dei nodi (D) e dall'utilizzo delle foglie stesse (sperimentalmente l'utilizzo medio è $\ln 2 \approx 0.69$).

Trascurando, per semplicità, il puntatore alla foglia successiva, considerando chiavi di lunghezza len(k) e puntatori ai dati (TID) di lunghezza len(p), il numero di foglie (NL) può essere calcolato come segue:

$$\boldsymbol{NL = \left\lceil \frac{NR \times \left(len(k) + len(p)\right)}{D \times u} \right\rceil}$$

NR	Numero record	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000	1.000.000
D	D	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096
len(k)	Lunghezza valori di chiave	10	10	10	10	10	10	10	40	40	40	40	40	40	40
len(q)	Lunghezza puntatori ai nodi	1	2	3	4	8	16	32	1	2	3	4	8	16	32
len(p)	Lunghezza puntatori ai record	3	3	3	3	3	3	3	3	3	3	3	3	3	3
u	Fattore utilizzo foglie (ln 2)	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69
g	**Ordine del B⁺-tree**	**186**	**170**	**157**	**146**	**113**	**78**	**48**	**49**	**48**	**47**	**46**	**42**	**36**	**28**
	Spazio per le chiavi in ogni nodo	3.720	3.400	3.140	2.920	2.260	1.560	960	3.920	3.840	3.760	3.680	3.360	2.880	2.240
	Spazio per i puntatori in ogni nodo	373	682	945	1.172	1.816	2.512	3.104	99	194	285	372	680	1.168	1.824
	Spazio utilizzato in ogni nodo	4.093	4.082	4.085	4.092	4.076	4.072	4.064	4.019	4.034	4.045	4.052	4.040	4.048	4.064
	Spazio non utilizzato in ogni nodo	**3**	**14**	**11**	**4**	**20**	**24**	**32**	**77**	**62**	**51**	**44**	**56**	**48**	**32**
NL	**Numero foglie**	**4.579**	**4.579**	**4.579**	**4.579**	**4.579**	**4.579**	**4.579**	**15.146**	**15.146**	**15.146**	**15.146**	**15.146**	**15.146**	**15.146**
h_{min}	**Altezza minima**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
h_{max}	**Altezza massima**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**4**	**4**

NR	Numero record	2	16	32	1.024	2.048	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576
D	D	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024
len(k)	Lunghezza valori di chiave	10	10	10	10	10	10	10	10	10	10	10	10	10	10
len(q)	Lunghezza puntatori ai nodi	4	4	4	4	4	4	4	4	4	4	4	4	4	4
len(p)	Lunghezza puntatori ai record	4	4	4	4	4	4	4	4	4	4	4	4	4	4
u	Fattore utilizzo foglie (ln 2)	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69
g	**Ordine del B⁺-tree**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**	**36**
	Spazio per le chiavi in ogni nodo	720	720	720	720	720	720	720	720	720	720	720	720	720	720
	Spazio per i puntatori in ogni nodo	292	292	292	292	292	292	292	292	292	292	292	292	292	292
	Spazio utilizzato in ogni nodo	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012
	Spazio non utilizzato in ogni nodo	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
NL	**Numero foglie**	**1**	**1**	**1**	**21**	**41**	**81**	**162**	**324**	**647**	**1.293**	**2.586**	**5.171**	**10.342**	**20.683**
h_{min}	**Altezza minima**	**1**	**1**	**1**	**2**	**2**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**4**	**4**
h_{max}	**Altezza massima**	**1**	**1**	**1**	**2**	**2**	**3**	**3**	**3**	**3**	**3**	**3**	**4**	**4**	**4**

NR	Numero record	2	16	32	1.024	2.048	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576
D	D	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024
len(k)	Lunghezza valori di chiave	20	20	20	20	20	20	20	20	20	20	20	20	20	20
len(q)	Lunghezza puntatori ai nodi	4	4	4	4	4	4	4	4	4	4	4	4	4	4
len(p)	Lunghezza puntatori ai record	4	4	4	4	4	4	4	4	4	4	4	4	4	4
u	Fattore utilizzo foglie (ln 2)	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69	0,69
g	**Ordine del B⁺-tree**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**
	Spazio per le chiavi in ogni nodo	840	840	840	840	840	840	840	840	840	840	840	840	840	840
	Spazio per i puntatori in ogni nodo	172	172	172	172	172	172	172	172	172	172	172	172	172	172
	Spazio utilizzato in ogni nodo	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012	1.012
	Spazio non utilizzato in ogni nodo	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
NL	**Numero foglie**	**1**	**1**	**2**	**35**	**70**	**139**	**277**	**554**	**1.108**	**2.216**	**4.432**	**8.864**	**17.728**	**35.456**
h_{min}	**Altezza minima**	**1**	**1**	**2**	**2**	**3**	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**
h_{max}	**Altezza massima**	**1**	**1**	**2**	**2**	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**5**

I puntatori ai dati, nei (secondary) B⁺-tree, possono essere di due tipi:

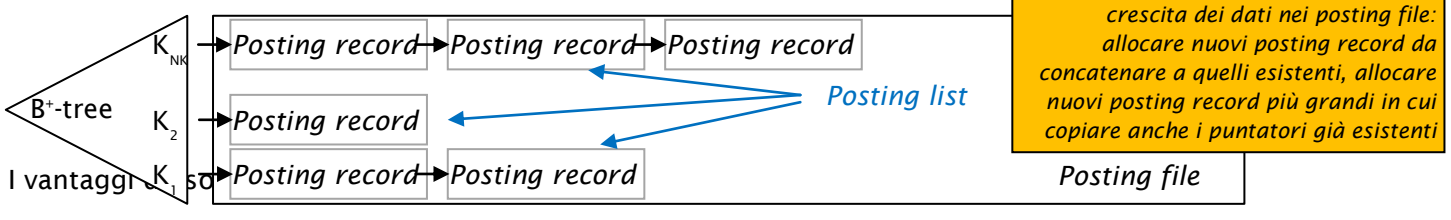
1. **Puntatori a TID** (Tuple IDentifier): ogni valore di chiave presente nelle foglie corrisponde ad una lista di puntatori ai record con quel valore di chiave (*inverted index*); la lista di TID è solitamente mantenuta ordinata per valori crescenti
2. **Puntatori a PID** (Page IDentifier): ogni valore di chiave presente nelle foglie corrisponde ad una lista di puntatori alle pagine del file di dati che contengono almeno un record con quel valore di chiave (è conveniente per indici *clustered* in cui i valori di chiavi uguali sono aggregati).

In un B⁺-tree con puntatori a TID, il numero di foglie (NL) dipende dal numero di chiavi distinte (NK), e l'altezza dipende dal minimo tra il numero di chiavi distinte e il numero di foglie:

$$NL = \left\lceil \frac{NK \times \left(len(k) + NR \times len(p)\right)}{D \times u} \right\rceil$$

$$N := min\{NK, NL\} \quad \left\lceil log_{2g+1}(N+1) \right\rceil \leq h \leq \left\lfloor 1 + log_{g+1}\left(\frac{N+1}{2}\right) \right\rfloor$$

Allo scopo di mantenere un formato a lunghezza fissa per i nodi foglia e facilitare la gestione dell'evoluzione dell'albero, si adotta una tecnica che fa uso di un area separata detta *posting file* per cui le foglie del B⁺-tree contengono, per ciascun valore di chiave distinto, un puntatore alla testa della relativa lista di puntatori (a TID o a PID) sono memorizzati nel *posting file*. I *posting file* sono costituiti da *posting record* concatenati a formare *posting list* dedicate ai singoli valori di chiave.



Esistono strategie diverse per gestire la crescita dei dati nei posting file: allocare nuovi posting record da concatenare a quelli esistenti, allocare nuovi posting record più grandi in cui copiare anche i puntatori già esistenti

I vantaggi d so

clustering: il fatto che il file sia ordinato sui valori della chiave facilita l'elaborazione secondo valori crescenti di chiave
dinamicità: le procedure di gestione dei B-tree applicate ai file permettono di evitare problemi legati alla riorganizzazione delle strutture dati (ISAM)

Nelle organizzazioni primarie con B^+-tree, il file dati è organizzato per valori crescenti del campo chiave, all'inserimento di record può essere necessario provvedere a modificare i puntatori nell'indice.

Virtual **S**torage **A**ccess **M**ethod è un'organizzazione dinamica, utilizzata nei sistemi IBM OS/VS che si compone di un file ordinato sul valore chiave e di un indice *sparso*, organizzato come un B^+-tree distribuito nello stesso file: le pagine dati sono gestite come foglie di un B^+-tree, nelle pagine indice sono riportate le coppie valore di chiave/indirizzo pagina. Il file VSAM sono divisi in *regioni* (costituite da un insieme di tracce) costituite da *intervalli* (parti di traccia o tracce contigue accessibili con una sola operazione I/O).

Convenienza d'uso di un indice

L'*ottimizzazione delle query* ha lo scopo di determinare la *strategia di esecuzione* a costo minimo (miglior *piano di accesso*). L'utilizzo di un indice piuttosto che una scansione sequenziale possono essere l'uno più conveniente dell'altro in situazioni differenti. La bontà di una strategia di esecuzione può essere valutata secondo diverse metriche di cui, solitamente, la *minimizzazione del numero di operazioni di I/O* è quella più utilizzata.

Un possibile modello di stima dei costi può basarsi sui seguenti criteri:
1. Ordinamento dei dati (richiesto o non)
2. Ordinamento preventivo dei TID
   2.1. Di un valore di chiave
   2.2. Di più valori di chiave
3. Distribuzione dei valori di chiave uniforme
4. Distribuzione dei record con un certo valore di chiave sulle pagine del file di dati (casuale)

Nell'ipotesi che:
1. Non sia richiesto alcun ordinamento dei dati in output
2. La lista di TID di un valore di chiave sia ordinata
3. La distribuzione dei valori di chiave sia uniforme
4. La distribuzione di record con lo stesso valore chiave sia uniforme sulle pagine del file di dati

è possibile utilizzare la *Formula di Cardenas* per fornire una stima del numero medio di pagine su un totale di NP che contengono almeno uno degli ER (expected) record.

Si consideri che:
1. 1/NP è la probabilità che una pagina contenga uno degli ER record
2. 1-1/NP è la probabilità che una pagina non lo contenga
3. $(1-1/NP)^{ER}$ è la probabilità che non contenga nessuno degli ER
4. $1-(1-1/NP)^{ER}$ è la probabilità che ne contenga almeno uno

allora, moltiplicando per il numero delle pagine, si ottiene:

$$\Phi(ER, NP) = NP \times \left(1 - \left(1 - \frac{1}{NP}\right)^{ER}\right) \leq \min\{ER, NP\}$$

Costo di accesso con scansione sequenziale: $C_a(seq) = NP$

Costo di accesso con indice unclustered:

$C_I = h - 1 + \left\lceil \frac{EK}{NK} NL \right\rceil$ *costo di accesso all'indice*

$C_D = EK \times \Phi\left(\frac{NR}{NK}, NP\right)$ *costo di accesso alle pagine dati*

$C_a(uncl) = C_I + C_D$ *costo totale di accesso con indice unclustered*

Note:

- L'altezza effettiva dell'albero è ininfluente, solitamente si assume h = 4
- La formula fornisce un valor medio che può non essere intero
- È necessario arrotondare il valore EK / NK × EL per il numero medio delle foglie.
- Il modello di Cardenas assume pagine a *capacità infinita* e sottostima il valore corretto nel caso di pagine con meno di circa 10 record.

La *formula di Yao* considera la capacità effettiva delle pagine: C = NR / NP.

$\binom{NR}{ER}$ *numero di combinazioni*

$\binom{NR-C}{ER}$ *numero di combinazioni escludendo una pagina*

$\binom{NR}{ER} - \binom{NR-C}{ER}$ *numero di combinazioni che interessano una pagina*

$\frac{\binom{NR}{ER} - \binom{NR-C}{ER}}{\binom{NR}{ER}}$ *probabilità di accedere ad una pagina*

$$\Phi_Y(ER, NR, C) = NP \times \left(1 - \frac{\binom{NR-C}{ER}}{\binom{NR}{ER}}\right)$$

Note:

- Nel caso di pagine con numero variabile di record, si può dimostrare che la formula di Yao sovrastima
- Nel caso di allocazione non casuale dei record, entrambi i modelli sovrastimano
- Se ER è grande, la formula di Yao può richiedere elevati tempi di calcolo.

Utilizzo di più indici, di indici su combinazioni di valori

Per risolvere alcune interrogazioni complesse è possibile utilizzare più indici tramite algoritmi di *intersezione* e di *unione*: nel caso di TID si ottiene una lista di puntatori ai record che soddisfano entrambi i predicati, nel caso di PID si ottiene una lista di puntatori a pagine che contengono almeno un record che soddisfa il primo predicato e un record che soddisfa il secondo (*attenzione: potrebbe non esistere alcun record che soddisfi entrambi i predicati*. In certi casi è possibile creare indici *multi-attributo* che memorizzano come valore di chiave tutte le combinazioni dei valori degli attributi scelti; questi hanno i vantaggi di ridurre il numero di TID, rendere efficienti le elaborazioni con condizioni di tipo "i primi j valori…", rendere efficienti gli aggiornamenti.

Bit-mapped indexing: metodo di accesso applicabile nel caso di attributi con valori ripetuti e in presenza di più indici; crea una *mappa di bit* per ogni indice ed effettua un AND logico sulle tuple per verificare quali record soddisfano tutti i criteri.

Organizzazioni Hash

Le organizzazioni hash utilizzano funzioni hash che, a differenze delle tecniche *tabellari*, trasformano ogni valore di chiave in un indirizzo; è necessario gestire le collisioni in quanto le funzioni hash non sono, solitamente, iniettive.

Ogni indirizzo generato dalla funzione hash individua una *pagina logica* (*bucket*); la capacità del bucket è determinata dal numero di elementi che possono esservi allocati.

Le funzioni hash indirizzano bucket che costituiscono l'area di memoria detta *primaria*. La presenza di overflow determina l'utilizzo di un'area di memoria separata detta *area di overflow*.

Una funzione hash deve essere *suriettiva* generando NP indirizzi: tanti quanti sono i bucket dell'area primaria.

Alcuni aspetti comuni a tutte le tipologie di organizzazioni hash:

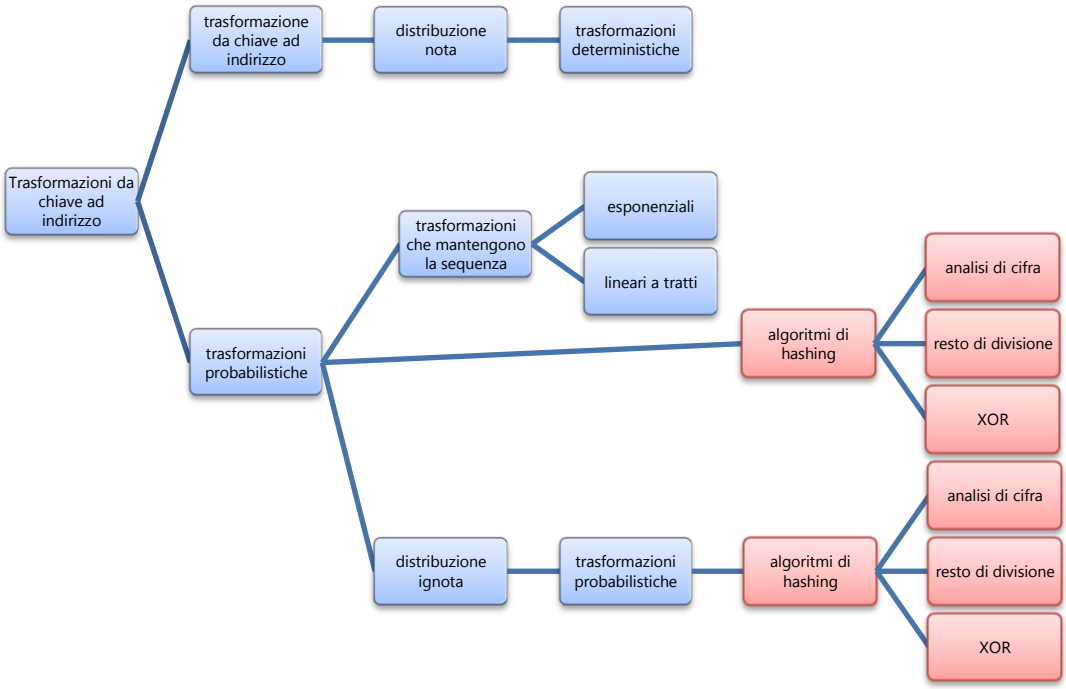
- Scelta della funzione hash
- Politica di gestione degli overflow
- Capacità dei bucket dell'area primaria
- Capacità dei bucket dell'eventuale area di overflow
- Utilizzo della memoria allocata.

Solitamente le organizzazioni hash sono primarie. Siccome gli indici hash non preservano l'ordine, è sconsigliato l'utilizzo di essi nel caso di possibili interrogazioni per intervallo.

Organizzazioni hash statiche

Definiscono un dimensionamento dell'area primaria direttamente sulla base del valore di NP.

Una tipica funzione hash statica è il *modulo*.



Funzioni hash perfette (PHF): si tratta di una classe di funzioni hash che mappano un insieme statico di chiavi in bucket (di capacità) unitaria senza generare collisioni.

Tali funzioni sono rare. Su tabelle di dimensioni minima, pari al numero delle chiavi, si parla di *funzioni hash perfette minimali (MPHF);* se esse mantengono l'ordine delle chiavi si parla di *order preserving MPHF (OMPHF).*

Esempio: date NK = 31 chiavi e NP = 41 bucket, si hanno:

$\binom{41}{31} = 0.112 \times 10^{10}$ *modi di scegliere* 31 *bucket*

$31! = 0.822 \times 10^{34}$ *modi di assegnare le* 31 *chiavi ai* 31 *bucket*

$41^{31} = 0.992 \times 10^{50}$ *funzioni da* $[1..31]$ *a* $[1..41]$

$\frac{\binom{41}{31} \times 31!}{41^{31}} = 0.930 \times 10^{-7}$ *funzioni perfette (iniettive)*

È possibile costruire funzioni hash anche per chiavi multiple: $(a_1, a_2, \dots, a_n) \xrightarrow[hash]{} (H_1(a_1), H_2(a_2), \dots, H_n(a_n))$

Esempio: $A_1 = int(5), A_2 = int(9), A_3 = string(10)$ $\quad$ *area primaria di* $2^9 = 512$ *bucket* (9 *bit necessari:* 4 *per* A_1, 3 *per* A_2, 2 *per* A_3)
Scelte le funzioni hash: $H_1 = A_1 \ mod\ 16 \quad H_2 = A_2 \ mod\ 8 \quad H_3 = (numero\ di\ caratteri\ blank\ di\ A_3) \ mod\ 4$
Il record: $a = (58651, 130326734, \text{"mamma"}) \xrightarrow[hash]{} (58651 \ mod\ 16, 130326734 \ mod\ 8, 5 \ mod\ 4) = (11,6,1) \xrightarrow[binary]{} (1011,110,1) = bucket\ \#377$

In caso di interrogazioni parzialmente specificate, sarà necessario visitare un numero di bucket in funzione del numero di bit non individuate per via dell'attributo mancante.

Esistono casi in cui la distribuzione dei valori di chiave è nota a priori e si possono utilizzare funzioni hash ad hoc basate sull'idea di suddividere l'intervallo di chiavi possibili in NP sottointervalli e utilizzare una funzione che associa ogni chiave al sottointervallo relativo:

$$H(k_i) = \left\lceil \frac{k_1 - K_{min} + 1}{K_{max} - K_{min} + 1} \times NP \right\rceil - 1$$

Tuttavia le funzioni hash note a priori non si presentano comunemente. In generale le funzioni hash devono *comportarsi bene* per distribuzioni arbitrarie dei valori di chiave nel dominio di definizione.

Nel caso di distribuzione omogenea, ogni bucket ha la stessa probabilità (1/NP) di essere indirizzato da un valore di chiave; la probabilità che un certo insieme di chiavi (X_j) sia assegnato allo stesso bucket (j) è descritta da una distribuzione binomiale:

$$P(X_j = x_j) = \binom{NR}{x_j} \left(\frac{1}{NP}\right)^{x_j} \left(1 - \frac{1}{NP}\right)^{NR - x_j} \quad con\ valor\ medio\ \mu = \frac{NR}{NP} \quad e\ varianza\ \sigma^2 = \frac{NR}{NP} \times \left(1 - \frac{1}{NP}\right)$$

Valore medio e varianza non dipendono dal bucket.

Le *prestazioni* delle funzioni hash variano al variare dello specifico set di chiavi: ogni funzione può dar luogo a prestazioni disastrose nel caso peggiore. Un criterio adeguato per la valutazione delle funzioni hash è dato dall'analisi della sua *degenerazione* che è data dal rapporto

$$\frac{\sigma_H}{\sqrt{\mu_H}} \quad con\ \mu_H = \sum_{j=0}^{NP-1} \frac{x_j}{NP} = \frac{NR}{NP} \quad e \quad \sigma_H^2 = \sum_{j=0}^{NP-1} \frac{(x_j - \mu_H)^2}{NP}$$

Si hanno prestazioni migliori per funzioni con degenerazione più bassa.

Alcune funzioni hash

k = valore di chiave *NP = numero di bucket* *H = funzione hash* *B = bucket*

Mid square: si eleva la chiave al quadrato k^2, si estrae un numero di cifre *centrali* pari a quelle del valore NP-1 e si normalizza tale numero a NP.

Esempio: k = 145142 ; NP = 8000 → k^2 = 21066620164 → ⌊6620⌋ × 0.8 = 5296

Shifting: la chiave è suddivisa in un certo numero di parti costituite da un numero di cifre pari al numero di cifre di NP meno una. Si sommano le parti e si normalizza il risultato.

Esempio: k = 21 066 200 164; NP = 8000 → 164 + 200 + 66 + 21 = 451 → ⌊6994 × 0.8 ⌋ = 360

Folding: la chiave è suddivisa come nello *shifting* ma le somme sono fatte sulle cifre delle singole parti "ripiegate".

Esempio: k = 21 066 200 164; NP = 8000 → 461 + 2 + 660 + 21 = 1144 → ⌊6994 × 0.8 ⌋ = 915

Divisione: la chiave viene divisa per un numero P e l'indirizzo è ottenuto considerando il resto.
H(k) = h mod P Per la scelta di P si hanno indicazioni pratiche:

1. P è il più grande numero primo minore o uguale a NP
2. P è non primo, minore o uguale a NP, con nessun fattore primo minore 20
   Se P < NP, si deve porre NP = P per non perdere la suriettività della funzione hash.

Un confronto sperimentale delle prestazioni dei diversi metodi, sulla base di un *fattore di caricamento* definito (rapporto tra numero di chiavi allocate e capacità dell'area primaria), mostra che il metodo della divisione è quello più affidabile. Naturalmente a capacità dell'area primaria più ridotte corrisponde una maggiore percentuale di record in overflow.

Funzioni hash 2-universali

Dato un insieme K *di chiavi* {0, 1, …, |K| - 1} *e* NP *bucket, una classe* H = {H_1, H_2, …} *di funzioni hash è 2-universale se, per ogni coppia di interi* K*, il numero totale di collisioni in* H *è minore o uguale a* |H|/NP*.*

Quindi, scegliendo a caso una funzione da H, la probabilità che due chiavi collidano è minore o uguale a 1/NP.
La classe H = { ((A × k + D) mod P) mod NP) | A ≥ 1, D, P ≥ 0 } è 2-universale.
ponendo A=1; D=0, P=∞ si ottiene: ((1 × k + 0) mod ∞) mod NP) = k mod NP (metodo della divisione).

Chiavi alfanumeriche

Un metodo comune per gestire chiavi alfanumeriche utilizza un *alfabeto* (A) a cui appartengono i caratteri delle stringhe, una *funzione biiettiva* (ord) che associa ogni elemento ad un numero intero nel range [1,|A|], una *base di conversione* (b).ù

Una stringa S = s_{n-1}, …, s_1, s_0 è convertita in una chiave numerica: $k(S) = \sum_{i=0}^{n-1} ord(s_i) \times b^i$

Esempio
A = {a, b, …, z} ord() → {1,…,26} b = 32 data la chiave S = "indice"

$$k(S) = 9 \times 32^5 + 14 \times 32^4 + 4 \times 32^3 + 9 \times 32^2 + 3 \times 32^1 + 5 \times 32^0 = 316.810.341$$

Nel caso non si facesse uso della base (b = 1), si otterrebbero chiavi uguali per anagrammi di parole in quanto la posizione dei singoli caratteri non influenzerebbe la funzione hash.

La **scelta della base** nel metodo della divisione è un fattore importante: per b e NP hanno fattori comuni, si possono riscontrare problemi. Infatti se b e NP hanno un fattore comune i valori che vengono generati dalla conversione di stringhe in numeri tramite alfabeto, vengono *eliminati* dal modulo per tutti i valori di base superiori al fattore comune.

$$H\big(k(S)\big) = \left\{\sum_{i=0}^{n-1}(ord(s_i) \times b^i)\right\} mod(b^\alpha) = \left\{\sum_{i=0}^{n-1}(ord(s_i) \times b^i)\, mod(b^\alpha)\right\} mod(b^\alpha)$$

Fattore di caricamento

Data una stima del numero NR di record da gestire e fissata la capacità C dei bucket, la scelta di un determinato fattore di caricamento d, determina il numero di bucket NP, in area primaria.

Valori tipici, che rappresentano un compromesso tra utilizzo della memoria e costi di esecuzione delle operazioni, si hanno nell'intervallo [0.7,0.8].

Capacità dei bucket

Poiché in generale, all'aumentare della capacità dei bucket (a parità di fattore di caricamento) la percentuale di record in overflow diminuisce, è conveniente scegliere capacità massima purché la lettura di un bucket comporti una sola operazion di I/O e il trasferimento di un bucket di capacità C avvenga in un tempo minore del trasferimento di due bucket di capacità minore di c.

Nel caso di funzioni hash ideali, il numero di volte che un indirizzo j è generato corrisponde ad una variabile aleatoria che segue una distribuzioni binomiale:

$$\boldsymbol{P(x) = \binom{NR}{x}\left(\frac{1}{NP}\right)^x\left(1-\frac{1}{NP}\right)^{NR-x} \approx \left(\frac{NR}{NP}\right)^x \times \frac{e^{-\frac{NR}{NP}}}{x!}}$$

Gestione dell'overflow

Allo scopo di ridurre al minimo il numero di accessi a bucket necessari per reperire il record cercato, si utilizzano diversi metodi:

- Metodi di concatenamento (*chaining*): utilizzano puntatori e possono memorizzare i record in overwflow sia in area primaria che in un'area separata

  1. Liste separate: se un bucket j è saturo, i record aventi j come *home bucket* sono allocati nel primo bucket non pieno trovato eseguendo una ricerca a partire dal bucket j+1; i record che collidono sono collegati a lista anche quelli non in overflow; ogni record deve includere un campo puntatore al successivo record
  2. Liste confluenti: si utilizza un solo puntatore per bucket, se il bucket j è saturo, il record viene inserito nel primo bucket non pieno j+h, si attiva un collegamento da j a j+h

     Determinano un minore utilizzo dei puntatori ma prestazioni peggiori.
  3. Chaining in area separata: si memorizzano gli overflow in un'area di memoria distinta da quella primaria, non indirizzata dalla funzione hash; i bucket di overflow possono avere capacità minore per evitare spreco di spazio.

- Metodi di indirizzamento aperto (*open addressing*): non utilizzano puntatori, i record in overflow sono allocati in area primaria tramite una *legge di scansione*. Richiedono, in caso di necessità di gestire un numero di record superiore a quello previsto, una fase di riorganizzazione completa. Consistono nell'inserire la chiave nel bucket definito dalla funzione di hash, quando questo è saturo, si applica la funzione di hash al risultato precedente modificato. La ricerca segue lo stesso metodo. La cancellazione deve essere gestita per permettere la ricerca sempre.

  1. Linear probing: incrementa, ad ogni passo, il valore da passare alla funzione di hash, di un valore costante s
     STEP (H_{j-1})(k_i)) = (H_{j-1} (k_i) + s) mod NP
     s non deve avere fattori in comune con NP.

     Linear probing genera il fenomeno del *clustering primario* per cui i record tendono ad addensarsi in alcuni bucket.
  2. Scansione quadratica: permette di superare il problema del *clustering primario* modificando il valore secondo una legge di scansione quadratica
     STEP (H_{j-1})(k_i)) = (H_{j-1} (k_i) + a + b (2 × j -1)) mod NP
     ovvero **STEP (H_j)(k_i)) = (H_0 (k_i) + a × j + b × j^2) mod NP**

     Non risolve però il problema di *clustering secondario*.

3. Double hashing: ha l'obbiettivo di eliminare il problema del *clustering secondario* facendo uso di due funzioni hash H' e H"; le sequenze di indirizzi sono date da:
   $H_0(k_i) = H'(k_i)$
   $H_j(k_i) = (H_{j-1}(k_i) + H''(k_i)) \bmod NP$

   Questo metodo approssima un caso ideale di hash uniforme, tuttavia genera una grande variabilità degli indirizzi generati con conseguente possibilità di appesantimento delle operazioni di I/O.

Dal punto di vista delle prestazioni i metodi con liste separate risultano quelli con costi medi di ricerca inferiori.

Organizzazioni hash dinamiche

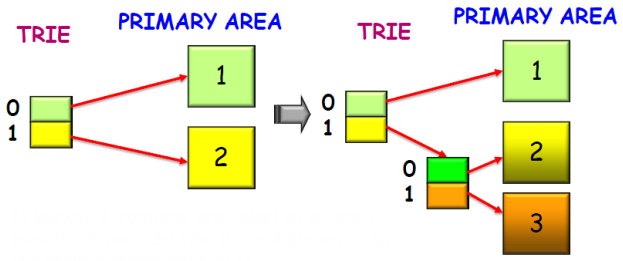
Definiscono un dimensionamento dinamico dell'area primaria per adattarsi meglio al volume effettivo dei dati da gestire (più funzioni hash).

Nel caso di archivi fortemente dinamici, un'allocazione statica è inadeguata a causa dell'eccessivo spreco di memoria o del deterioramento delle prestazioni.

Esistono due grandi categorie di organizzazioni hash dinamiche:

- Con direttorio
  1. Virtual hasing (Litwin 1978), si basa sull'idea di raddoppiare l'area primaria quando si verifica un overflow in un bucket e ridistribuire i record tra il bucket saturo e il suo *buddy*. Utilizza una struttura ausiliaria per determinare se occorre utilizzare funzione hash iniziale o quella nuova per ogni singolo bucket.

     (per approfondire: slide 5-13 organizzazioniDati_5)
  2. Dynamic hashing (Larson 1978), evita l'uso di tecniche di raddoppio che causano appesantimenti facendo uso di una struttura ausiliaria organizzata come un *trie binario*. Adotta una funzione hash che genera una stringa binaria detta *pseudo-chiave*; il trie serve per organizzare la ricerca: il cammino è definito dalla *pseudo-chiave*: il trie indirizza i bucket in modo che il nodo 0 indirizza il bucket contenente i valori per i quali la *pseudo-chiave* inizia con il simbolo 0, se tale bucket è saturo, viene allocato un nuovo bucket e i valori sono ridistribuiti in modo che uno contenga quelli con pseudo-chiave di tipo 00 e l'altro quelli con *pseudo-chiave* di tipo 01; al trie si aggiunge un nodo "figlio" del nodo 0 che punta rispettivamente ai due nuovi bucket.
     Se il trie è in memoria centrale è sufficiente un singolo accesso per recuperare un record. Viceversa, nel caso peggiore non si hanno buone prestazioni in manutenzione perché possono generarsi diverse fusioni e split dei bucket a seguito di cancellazioni e inserimenti.
     Esiste una variante del dynamic hashing che alloca inizialmente NP bucket tramite una funzione statica H, gli overflow generare trie differenti per i diveri bucket inziali.

  3. Extendible hashing (Fagin, Nievergelt, et al. 1979), è un'organizzazione simile al dynamic hashing, si differenzia per la gestione del direttorio che è un insieme di 2^p celle (p è detta *profondità del direttorio*), si utilizzano i p bit meno significativi della *pseudo-chiave* per accedere direttamente ad una cella che contiene un puntatore a un bucket. Garantisce il reperimento di un record con non più di due accessi alla memoria secondaria.

- Senza direttorio
  1. Linear hashing (Liwtin 1980), l'idea di base è "*non si esegue lo split del bucket in cui si è verificato overflow, ma si suddivide un altro bucket scelto secondo un criterio prefissato*", come conseguenza si ha che non è necessario un direttorio, occorre gestire l'overflow e l'area primaria cresce linearmente (senza raddoppi).

     (per approfondire: slide 27-34 organizzazioniDati_5)

     Pregi: l'assenza di un direttorio e la politica di gestione degli split rendono semplice la realizzazione della struttura; la gestione dell'area primaria è immediata in quanto i bucket vengono sempre aggiunti o rimossi in coda.
     Difetti: l'utilizzo della memoria allocata è basso, la gestione dell'area di overflow presenta problemi simili a quelli di un'area primaria statica, le catene di overflow ai bucket di indirizzo maggiore, non ancora suddivisi possono diventare molto lunghe.

  2. Spiral hashing (Martin 1979), cerca di risolvere il problema di catene di overflow lunghe impiegando una funzione di tipo esponenziale che consente di memorizzare i record più densamente all'estremo iniziale dell'area primaria.

Riepilogo costi e formule

NP:	numero pagine
NT:	numero record / tuple
M:	capacità memoria centrale (numero record)
FS:	numero di blocchi per pagina logica
IP:	numero di blocchi del file indice
C:	numero di record per blocco
IC:	numero di chiavi per blocco
	NP x C = IP x IC
$NK_{attributo}$:	numero di valori distinti di chiave dell'attributo
NL:	numero di nodi foglia dell'indice
C_I:	costo di accesso all'indice (numero blocchi letti)
C_D:	costo di accesso ai dati (numero blocchi letti)
C_{sort}:	costo di accesso per ordinamento (numero blocchi letti/scritti)
EL:	numero di foglie *attese*
EK:	numero di valori di chiavi distinti *attesi*
EP:	numero di pagine *attese*

Costo di accesso su file sequenziale ordinato:	$(NP + 1)/2$
Costo di accesso su file sequenziale non ordinato:	$(NP + 1)/2$
Costo di accesso su file ad accesso diretto ordinato:	$\lfloor \log_2 NP \rfloor$
Costo di accesso su file ad accesso diretto non ordinato:	$(NP + 1)/2$
Costo sort merge con M < N ≤ M²:	$N + 3 \times NP$
Costo sort merge con N > M²:	$2 \times NP + (N + NP) \times \left\lceil \log_M \frac{N}{M} \right\rceil$
Costo sort merge z-vie:	$2 \times NP \times \left(1 + \left\lceil \log_Z \frac{NP}{Z \times FS} \right\rceil\right) = 2 \times NP \times \left\lceil \log_Z \frac{NP}{FS} \right\rceil$
Costo ricerca binaria su file ordinato:	$\log_2 NP$
Costo ricerca binaria su indice ordinato:	$\log_2 IP + 1$ $(accesso\ al\ blocco\ dati)$
Costo piano d'accesso (generale):	$C_I + C_D + C_{sort}$
Costo di accesso scansione sequenziale:	$NP + C_{sort}$
Costo di accesso con indice clustered:	$EL + EP + C_{sort} = \left\lceil \frac{EK}{NK} NL \right\rceil + \left\lceil \frac{EK}{NK} NP \right\rceil$
Costo di accesso con indice unclustered:	$EL + EP + C_{sort} = \left\lceil \frac{EK}{NK} NL \right\rceil + EK \times \Phi\left(\frac{NR}{NK}, NP\right)$

Metodi per l'esecuzione di join

Algoritmi di join

La più semplice implementazione di un algoritmo di join di due relazioni (r e s) prevede il confronto di ogni tupla di r con ogni tupla di s, con complessità $O(NT_r \times NT_s)$.

Alcuni punti di vista per affrontare il problema dell'esecuzione di join:

- *Access path selection*: come viene influenzata l'esecuzione di un join dal cammino di accesso alle relazioni
- *Optimal nesting*: qual è l'ordine migliore di esecuzione di n join
- *Clustering*: in quale modo l'ordinamento fisico dei dati incide sui costi del join
- *Buffer*: come devono essere allocate le pagine di buffer per favorire l'esecuzione di un join
- *Hardware support*: qual è un buon supporto hardware per i join
- *Parallel processing*: come si sfrutta la presenza di più processori e/o dischi
- *Physical database design*: come si determinano gli indici utili all'esecuzione di join

Gli *two-way join* coinvolgono due relazioni, gli *multi-way join* coinvolgono più relazioni (si possono risolvere come n-1 two-way join indipendenti).

Tree join: albero di esecuzione di un join; rappresenta la modalità con cui si effettuano i collegamenti tra relazioni e si estraggono le tuple.

In caso di multi-way join non è sempre necessario che l'esecuzione di un join termini prima dell'inizio di un altro, si possono attivare allora esecuzioni in *pipeline*.

Nested loops join

È il metodo di join più semplice e deriva direttamente dalla definizione dell'operazione. Una delle due relazioni coinvolte è designata come *esterna*, l'altra come *interna*.

Per ogni tupla della relazione esterna che verifica i *predicati locali*, ricerca tutte le tuple della relazione interna che possono concatenarsi e che soddisfano i *predicati locali*.

Il costo di esecuzione si esprime come: $\mathbf{C_a(r) + ET_r \times C_a(s)}$

$C_a(r)$: costo di accesso a r
ET_r : numero atteso di tuple residue di r
$C_a(s)$: costo di accesso ad s

In assenza di predicati sulla relazione esterna:	$\mathbf{C_a(r) + NT_r \times C_a(s)}$	(NT numero tuple di r)
Facendo uso di scan sequenziali:	$\mathbf{NP_r + ET_r \times NP_s}$	(NP numero pagine)
Se valgono entrambe le condizioni:	$\mathbf{NP_r + NT_r \times NP_s}$	

A seconda che si parta dalla relazione r o da quella s si ha:
$NP_r + NT_r \times NP_s = NP_r + NP_r \times TP_r \times NP_s$ e $NP_s + NT_s \times NP_r = NP_s + NP_s \times TP_s \times NP_r$

Pertanto: per relazioni *grandi* contiene scegliere, come esterna, quella con il minor numero di tuple per pagina, nel caso in cui il numero di tuple per pagina sia lo stesso, si sceglie come relazione esterna, quella con il minor numero di pagine.

Ovviamente, la scelta della relazione esterna, determina l'ordine *primario* con cui sono generate le tuple.

Sulla relazione esterna è possibile effettuare accessi tramite:

- Scansione sequenziale
- Indice/indici su uno o più attributi che compaiono nei predicati locali della relazione esterna

Sulla relazione interna, è possibile effettuare accessi tramite:

- Scansione sequenziale
- Indice/indici su uno o più attributi che compaiono nei predicati locali della relazione interna
- Indice/indici su uno o più attributi di join.

Alcune varianti:

- Zig-zag: prevede che la relazione interna sia letta alternativamente dall'inizio alla fine e viceversa; in questo modo, ad ogni passo tranne il primo è possibile risparmiare la lettura di una pagina della relazione interna.
  $C_a(r) + (ET_r - 1) \times (C_a(s) - 1)$

- Uso di relazioni temporanee: vengono effettuate *restrizione* e *proiezione* delle due relazioni prima di verificare la condizione di join; questa variante preclude l'uso di un indice join sulla relazione interna.

- Nested-loops parallelo: l'algoritmo di nested-loops è parallelizzato

- Nested-block join: riduce il numero di scan sulla relazione interna facendo uso di un buffer di BP pagine di cui BP-1 riservate per la lettura della relazione esterna e 1 per la relazione interna; per ogni gruppo di BP-1 pagine della relazione esterna caricate nel buffer si apre una scansione sulla relazione interna.
  Costo: $NP_r + \lceil NP_r / (BP - 1) \rceil \times NP_s$
  Facendo uso di indici: $C_a(r) + \lceil EP_{p(R.Ai)} / (BP - 1) \rceil \times C_{a(s)}$
  L'ordine delle tuple è diverso da quello ottenuto con nested-loops.

- Sort-Merge join: è eseguito in due passi, il primo passo (*sort*) ordina le due relazioni sugli attributi di join, il secondo passo effettua una scansione su entrambe le relazioni nell'ordine degli attributi di join e le tuple che soddisfano la condizione di join sono fuse per costruire il risultato.
  $C(\text{Sort } r) + C(\text{Sort } s) + NP_r + NP_s$

- Sort-Merge con placeholder: nel caso di valori duplicati di indice per entrambi gli attributi di join, è necessario che l'algoritmo sia in grado di effettuare *backtracking* per poter individuare la prima tupla della relazione interna che verifica il predicato di join con la corrente tupla esterna. Rispetto al nested-loops, il numero di tuple confrontate è notevolmente inferiore infatti non è necessario effettuare la scansione di tutte le tuple della relazione interna per ogni tupla esterna in quanto si riparte dalla posizione indicata dal puntatore di gruppo.

- Merging-scans join: si tratta di una generalizzazione dell'algoritmo di sort-merge che richiede di poter accedere alle tuple secondo l'ordine stabilito dai valori degli attributi di join.

- Simple hash join:

  i metodi che utilizzano tecniche hash hanno l'obiettivo di ridurre il numero di confronti tra valori evitando però l'ordinamento e senza richiedere l'uso di indici sugli attributi di join. L'algoritmo di *simple-hash join*, il più semplice di questa famiglia, consta di due passi

  1. *Build*: si applica una funzione hash ai valori degli attributi di join di una delle relazioni generando una hash table, si usa un vettore di bit per tener traccia di quali bucket sono vuoti.
  2. *Probe*: si effettua una scansione sulla seconda relazione e, per ogni tupla, si accede alla hash table generata al passo di *build* solo se il bucket relativo non è vuoto.

  Il passo di build è effettuato solitamente sulla relazione con cardinalità minore (o con il minor numero di tuple residue, in presenza di predicati locali). L'efficienza decresce all'aumentare delle collisioni (cresce il numero di accessi e confronti inutili); con semplici modifiche il metodo permette di effettuare left e right outer join; esistono unità hardware che permettono l'implementazione hardware di questo metodo.

Selettività dei predicati

Fattore di selettività

Fattore di selettività: rapporto tra numero di tuple che soddisfano un certo predicato e numero di tuple della relazione alla quale è applicato il predicato.

Predicato	p
Predicato su un attributo A	$p(A)$
Numero di tuple della relazione R	NT_R
numero di valori distinti di un attributo A	NK_A
Numero di chiavi residue	EK_A
Fattore di selettività	$f_{p(A)} = EK_A/NK_A$
Numero di tuple residue	$ET_R = f_p \times NT_R$

Predicati notevoli

Predicato " = " → $f_{(A=v)} = \frac{1}{NK_A}$

Predicato "IN" → $f_{(A\in set)} = \frac{|set|}{NK_A}$

Predicato " < " → $f_{(A<v)} = \frac{v - \min(A)}{\max(A) - \min(A)} \times \frac{NK_A - 1}{NK_A}$

per attributi con molti valori il termine $\frac{NK_A - 1}{NK_A}$ *può essere omesso*

Predicato "between" → $f_{(A\in[v1,v2])} = \frac{v_2 - v_1}{\max(A) - \min(A)} \times \frac{NK_A - 1}{NK_A} + \frac{1}{NK_A}$

per attributi con molti valori il termine $\frac{NK_A - 1}{NK_A} + \frac{1}{NK_A}$ *può essere omesso*

predicati in "and" → $f_{p1} \times f_{p2}$

predicati in "or" → $f_{p1} + f_{p2} - (f_{p1} \times f_{p2})$

Esempi

Calcolo di selettività

Informazioni

$NT_{employee}$	=	20.000
NK_{deptNo}	=	100
NK_{job}	=	10
NK_{sex}	=	2
NK_{salary}	=	10
$\min(salary)$	=	5.000
$\max(salary)$	=	50.000

Predicato	***Selettività***	***Tuple residue***
$deptNo = 51$	$\frac{1}{100}$	$20.000 \times \frac{1}{100} = 200$
$salary > 10.000$	$\frac{50.000 - 10.000}{50.000} \times \frac{9}{10} = \frac{18}{25}$	$20.000 \times \frac{18}{25} = 14.400$
$job = 'clerk'$	$\frac{1}{10}$	$20.000 \times \frac{1}{10} = 2.000$
$sex = 'female'$	$\frac{1}{2}$	$20.000 \times \frac{1}{2} = 10.000$
$deptNo = 51\ and\ salary > 10.000$	$\frac{1}{100} \times \frac{18}{25} = \frac{18}{2500}$	$20.000 \times \frac{18}{2500} = 144$
$job = 'clerk'\ or\ sex = 'female'$	$\frac{1}{10} + \frac{1}{2} - \left(\frac{1}{10} \times \frac{1}{2}\right) = \frac{6}{10} - \frac{1}{20} = \frac{11}{20}$	$20.000 \times \frac{11}{20} = 11.000$

Piani di accesso

Scansione sequenziale

$$C(seq\ R) = NP_R + \alpha \times NT_R$$

Accesso con indice clustered

$$C(IX(R.Ai)) = NI_{R.Ai} + EL_{p(R.Ai)} + EP_{p(R.Ai)} = NI_{R.Ai} + \lceil f_{p(R.Ai)} \times NL_{(R.Ai)} \rceil + \lceil f_{p(R.Ai)} \times NP_R \rceil$$

Accesso con indice unclustered

$$C(IX(R.Ai)) = NI_{R.Ai} + EL_{p(R.Ai)} + EP_{p(R.Ai)} = NI_{R.Ai} + \lceil f_{p(R.Ai)} \times NL_{(R.Ai)} \rceil + EK_{p(R.Ai)} \times \Phi\left(\frac{NT_R}{NK_{R.Ai}}, NP_R\right)$$

Esempio

Informazioni

$NP_{employee}$	=	**2.000**	*alias E per employee*
D	=	**4096 byte**	*capacità pagine dati e indice*
u	=	**0.69**	*fattore di riempimento delle foglie*
$L(deptNo)$	=	**2 byte**	*alias D per deptNo*
$L(salary)$	=	**4 byte**	*alias S per salary*
$L(job)$	=	**10 byte**	*alias J per job*
$L(TID)$	=	**4 byte**	*lunghezza di un TID*
NI	=	**2**	*per ogni indice*

$NT_{employee}$	=	**20.000**
NK_{deptNo}	=	**100**
NK_{job}	=	**10**
NK_{sex}	=	**2**
NK_{salary}	=	**10**
$\min(salary)$	=	**5.000**
$\max(salary)$	=	**50.000**

Numero di foglie: $NL = \left\lceil \frac{NK_{R.Ai} \times L(R.Ai) + NT_R \times L(TID)}{D \times u} \right\rceil$

$NL_{E.D} = \left\lceil \frac{100 \times 2 + 20.000 \times 4}{4.096 \times 0.69} \right\rceil = 29$ ⋮ $NL_{E.S} = \left\lceil \frac{10 \times 4 + 20.000 \times 4}{4.096 \times 0.69} \right\rceil = 29$ ⋮ $NL_{E.J} = \left\lceil \frac{10 \times 10 + 20.000 \times 4}{4.096 \times 0.69} \right\rceil = 29$

Scansione sequenziale

$$C(seq\ E) = NP_R + \alpha \times NT_R = 2.000 + \alpha \times 20.000$$

Indice E.D unclustered e indice E.S clustered

$$EK_{p(E.D)} = \left\lceil \frac{1}{100} \times 100 \right\rceil = 1$$

$$C(IX(E.D)\ uncl) = NI_{E.D} + EL_{p(E.D)} + EP_{p(E.D)} = 2 + \lceil f_{p(E.D)} \times NL_{(E.D)} \rceil + EK_{p(E.D)} \times \Phi\left(\frac{NT_E}{NK_{E.D}}, NP_E\right) =$$

$$= 2 + \left\lceil \frac{1}{100} \times 29 \right\rceil + \frac{1}{100} \times \Phi\left(\frac{20.000}{100}, 2.000\right) = 2 + 1 + \left\lceil 1 \times 2.000 \times \left(1 - \left(1 - \frac{1}{2.000}\right)^{200}\right) \right\rceil = 2 + 1 + 191 = 194$$

$$C(IX(E.S)\ clus) = NI_{E.S} + EL_{p(E.S)} + EP_{p(E.S)} = NI_{E.S} + \lceil f_{p(E.S)} \times NL_{(E.S)} \rceil + \lceil f_{p(E.S)} \times NP_E \rceil = 2 + \left\lceil \frac{18}{25} \times 29 \right\rceil + \left\lceil \frac{18}{25} \times 2.000 \right\rceil =$$

$$= 2 + 21 + 1.440 = 1.463$$

$$C(IX(E.D)\ uncl) + C(IX(E.S)\ clus) = 194 + 1.463 = 1.657$$

Indice E.D clustered e indice E.S unclustered

$$C(IX(E.D)\ clus) = NI_{E.D} + EL_{p(E.D)} + EP_{p(E.D)} = NI_{E.D} + \lceil f_{p(E.D)} \times NL_{(E.D)} \rceil + \lceil f_{p(E.D)} \times NP_E \rceil = 2 + \left\lceil \frac{1}{100} \times 29 \right\rceil + \left\lceil \frac{1}{100} \times 2.000 \right\rceil =$$

$$= 2 + 1 + 20 = 23$$

$$EK_{p(E.S)} = \left\lceil \frac{18}{25} \times 10 \right\rceil = 8$$

$$C(IX(E.S)\ uncl) = NI_{E.S} + EL_{p(E.S)} + EP_{p(E.S)} = 2 + \lceil f_{p(E.S)} \times NL_{(E.S)} \rceil + EK_{p(E.S)} \times \Phi\left(\frac{NT_E}{NK_{E.S}}, NP_E\right) =$$

$$= 2 + \left\lceil \frac{18}{25} \times 29 \right\rceil + 8 \times \Phi\left(\frac{20.000}{10}, 2.000\right) = 2 + 21 + \left\lceil 8 \times 2.000 \times \left(1 - \left(1 - \frac{1}{2.000}\right)^{2.000}\right) \right\rceil = 2 + 21 + 8 \times 1.265 =$$

$$= 10.143$$

$$C(IX(E.D)\ clus) + C(IX(E.S)\ uncl) = 23 + 10.143 = 10.166$$

Normalizzazione

Forma normale: proprietà di uno schema relazionale che ne definisce la *qualità* ovvero l'assenza di determinati difetti. Uno schema non normalizzato presenza ridondanza e si presta a comportamenti poco desiderabili durante gli aggiornamenti.

Normalizzazione: attività che trasforma schemi non normalizzati in schemi che normalizzati; utilizzata come tecnica di verifica dei risultati della progettazione di una base di dati.

Ridondanze e anomalie

In uno schema non normalizzato possono presentarsi:

- *Ridondanze*: valori uguali ripetuti in tuple differenti per errato accorpamento di attributi di entità differenti
- *Anomalie di aggiornamento*: la presenza di ridondanze determina la necessità di aggiornamento di tutte le tuple che contengono il valore da modificare
- *Anomalie di cancellazione*: la presenza di accorpamento di attributi di entità differenti determina che la cancellazione dei valori di una delle entità implica la cancellazione dei valori delle altre entità coinvolte
- *Anomalia di inserimento*: la presenza di accorpamento di attributi di entità differenti determina che i valori degli attributi di un'entità non possono essere inseriti se non esistono anche i valori per gli attributi di tutte le altre entità coinvolte

Esempio

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	58	Venere	15	direttore
Neri	58	Giove	15	consulente
Neri	58	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

- *Ridondanze*: lo stipendio è ripetuto in ogni tupla
- *Anomalie di aggiornamento*: se varia lo stipendio di un impiegato deve essere aggiornato in ogni tupla
- *Anomalie di cancellazione*: se un impiegato non lavora su nessun progetto, deve essere cancellato completamente dallo schema (non esiste più l'impiegato)
- *Anomalie di inserimento*: analogamente, non è possibile inserire un impiegato senza progetto

Dipendenza funzionale

Si consideri un'istanza r di uno schema R(X) e due sottoinsiemi (non vuoti) di attributi Y e Z di X.

Si dice che in r vale la dipendenza funzionale (FD) Y → Z (*si dice Y determina funzionalmente Z*) se

$$\forall t1, t2 \in r : t1[Y] = t2[Y] \Rightarrow t1[Z] = t2[Z]$$

Ovvero, per ogni coppia di tuple t1 e t2 di r con gli stessi valori su Y, t1 e t2 hanno gli stessi valori su Z.

Una dipendenza funzionale è una caratteristica dello schema (*aspetto intensionale*) e non di una particolare istanza dello schema (*aspetto estensionale*); essa è dettata dalla semantica degli attributi di una relazione e non può essere inferita da una o più particolari istanze dello schema.

Ogni istanza che rispetta una data dipendenza funzionale è detta *istanza legale* dello schema rispetto alla dipendenza funzionale.

Se X è una *chiave* in uno schema R, allora ogni altro attributo di R dipende funzionalmente da X.

È possibile esprimere il concetto di *superchiave* facendo uso di dipendenze funzionali:

K ⊆ T è superchiave di R(T) ⇔ K → T

Dimostrazione

- Se K → T allora, per ogni istanza legale r si ha che ∀ t1, t2 ∈ r : t1[K] = t2[K] ⇒ t1[T] = t2[T], ovvero t1 = t2, quindi non esistono due tuple distinte con lo stesso valore di K
- Viceversa, se K è superchiave di R(T) allora se t1[K] = t2[K] ⇒ t1[T] = t2[T]

1° Forma normale

Una relazione è in Prima Forma Normale (1NF) se e solo se il dominio di ciascun attributo comprende solo valori atomici e il valore di ciascun attributo in una tupla è un valore singolo del dominio di quell'attributo.

Una tabella è in 1NF se e solo se è isomorfa ad una quale relazione:

1. L'ordine delle righe non è rilevante
2. L'ordine delle colonne non è rilevante
3. Non ci sono righe duplicate
4. Ogni *intersezione* tra una riga e una colonna contiene esattamente un valore del dominio applicativo
    a. NULL non fa parte del dominio, tuttavia si tratta di un'eccezione tollerata
5. Ogni colonna è *regolare*: non devono esiste colonne nascoste

2° Forma normale

Attributo primo: dato uno schema R(T), un attributo A ∈ R(T) è *primo* se e solo se fa parte di almeno una chiave dello schema; in caso contrario è *non primo*.

Una relazione R(T) con vincoli F è in Seconda Forma Normale (2NF) se e solo se ogni attributo non primo dipende completamente da ogni chiave candidata dello schema (non deve esserci dipendenza parziale di un attributo non primo da una chiave).

Uno schema in 1NF in cui le chiavi sono tutte formate da un singolo attributo (semplici) è anche in 2NF.

3° Forma normale

Dipendenza transitiva: dato uno schema R(T), X ⊆ T, A ∈ T; si dice che A *dipende transitivamente* da X se esiste Y ⊂ T tale che:

1. X → Y
2. ! (Y → X)
3. Y → A
4. A ∉ Y

Una relazione R(T) con vincoli F è in Terza Forma Normale (3NF) se e solo se ogni attributo non-primo non dipende transitivamente da nessuna chiave (non deve esserci dipendenza transitiva di un attributo non-primo da una chiave).

Forma normale di Boyce-Codd

Uno schema R(T) è in Forma Normale di Boyce e Codd (BCNF) se, per ogni dipendenza funzionale (non banale) X → A definita su di esso, X è una superchiave di R(T).

Decomposizioni

La decomposizione delle relazioni, in generale, potrebbe generare perdita di informazioni.

Decomposizione senza perdita: uno schema R(X) si decompone senza perdita negli schemi R1(X1) e R2(X2) se, per ogni istanza legale r su R(X), il join naturale delle proiezioni di r su X1 e X2 è uguale a r stessa: $\boldsymbol{\pi_{X1}(r) \bowtie \pi_{X2}(r) = r}$

Per decomporre senza perdita è sufficiente e necessario che il join naturale sia eseguito su una superchiave di uno dei due sottoschemi:
X1 ∩ X2 → X1 oppure **X1 ∩ X2 → X2**

Si dice che una decomposizione *preserva le dipendenze* se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti. Se una decomposizione non preserva le dipendenze è necessario effettuare query di verifica prima di effettuare una modifica.

Una decomposizione *deve* essere senza perdita per garantire la ricostruzione delle informazioni originarie e *dovrebbe* preservare le dipendenze per semplificare il mantenimento dei vincoli di integrità originari.

Considerazioni

Non è sempre detto che sia necessario effettuare la normalizzazione, infatti:

- La normalizzazione elimina le anomalie ma può appesantire l'esecuzione di certe operazioni
- Relazioni con frequenza di aggiornamento bassa danno un minor numero di problemi se non sono normalizzate
- La ridondanza presente nelle relazioni non normalizzate deve essere quantificata allo scopo di capire quanto possa incidere sull'occupazione di memoria e sui costi per il mantenimento dei valori duplicati

Riepilogo forme normali

Condizione	1NF	2NF	3NF	BCNF
Valori atomici di dominio degli attributi	✓	✓	✓	✓
Solo valori singoli per attributo	✓	✓	✓	✓
Ogni attributo non primo dipende completamente da ogni chiave candidata dello schema		✓	✓	✓
Ogni attributo non primo non dipende transitivamente da nessuna chiave			✓	✓
Per ogni dipendenza funzionale non banale X → A, X è superchiave				✓

Progettazione concettuale

Progettazione logica

Algebra relazionale

SQL

Basi

Gruppi

Subquery

Viste

LINQ

ADONET

Appendici

Probabilità e statistica

Generazione di distribuzioni

Linear counting

Sommario