Complementi di Basi di Dati

Ilaria Battiston

Anno scolastico 2018-2019

Contents

1 Introduzione

I database non consistono solo in dati e tabelle: sono modelli centralizzati o distribuiti che permettono di gestire un carico anche significativo di utenti. La sicurezza è un aspetto importante, insieme all'affidabilità: i sistemi relazionali hanno grandi utilizzi nei settori bancari, e le operazioni devono giungere a termine senza guasti.

I DBMS sono quindi sistemi che devono garantire la gestione di dati di grandi dimensioni, persistenti, affidabili e condivisi. Oltre ai modelli relazionali esistono i NoSQL (Not Only SQL) e RDF, i quali hanno vantaggi e svantaggi a seconda dell'utilizzo.

Per garantire le precedenti caratteristiche, l'architettura di un DBMS deve avere una serie di funzionalità cooperanti, come un gestore delle transazioni, un query compiler e un gestore della memoria secondaria.

2 Strutture fisiche di accesso

La più piccola struttura di memorizzazione dei dati a cui gli utenti possano accedere è il file: così come le tabelle, ha un'intestazione fissa e un numero di righe.

I dati non devono solo essere memorizzati in modo persistente: devono anche essere facilmente recuperabili, e gli accessi da gestire sono sia in lettura che in scrittura. Un obiettivo fondamentale è la minnimizzazione del tempo di accesso e trasferimento da CPU alla memoria secondaria.

I dati sono memorizzati nei dischi magnetici, con blocchi da 4-32kbyte e un tempo di accesso di circa 10^{-8} millisecondi, con un tasso di trasferimento di 300 Mbit/secondo. Il problema è la grande differenza di ordini di grandezza tra queste operazioni.

L'organizzazione ottimale è un compromesso tra il tempo e lo spazio, di cui il tempo è la priorità considerando il costo ridotto della memoria.

2.1 Campi in SQL

Ogni campo in una tabella è memorizzato in una struttura fisica, ma lo spazio occupato cambia a seconda del tipo del dato. Le tuple sono pertanto record fisici organizzati in collezioni all'interno dei blocchi di memoria, e bisogna gestire anche modifica e cancellazione.

Esempi di tipi di dati:

- VARCHAR, che alloca $n+1$ bytes secondo un bit di carattere separatore o numero di caratteri;
- BLOB e GLOB, destinati a larghi file il cui spazio viene allocato solo al momento dell'inserimento.

I record possono avere formato e lunghezza fissi o variabili, con eventuali campi straordinari. Il record layout include informazioni supplementari con schemi o puntatori, lunghezza e timestamp di ultima lettura e scrittura.

I record sono organizzati in blocchi, unità di memoria trasferite dal disco alla memoria principale. La dimensione è generalmente fissa a 2^n, di cui solitamente alcuni byte non vengono utilizzati.

Nel caso in cui la lunghezza sia variabile, lo header contiene anche la lunghezza del record e l'offset (distanza rispetto al byte iniziale) dei campi. I campi fissi sono allocati prima di quelli variabili.

Alcuni record sono rappresentati in più blocchi, cioè spanned: anche questa informazione è contenuta nell'header, e l'ordinamento è gestito tramite la contiguità fisica o collegamento tra record (modello a grafo, sistemi scalabili).

3 Strutture di files

I metodi di accesso sono algoritmi o moduli che forniscono primitive CRUD per ciascuna delle strutture.

Esempi di metodi di accesso sono:

- Strutture sequenziali;
- Strutture con accesso calcolato (hash);
- Strutture ad albero (indice).

L'accesso può essere sequenziale, binario o con indice. Il costo è determinato dallo spazio e dal tempo, ed è solitamente approssimato come il numero di blocchi acceduti. Per sceglere la struttura ottima è necessario distinguere se i file sono statici o dinamici e la frequenza delle operazioni CRUD.

3.1 Sequenziali

Le strutture sequenziali hanno in comune il mantenimento di un ordinamento fisico in memoria. Esistono organizzazioni per righe o per colonne.

Non ordinata: i nuovi record sono appesi in fondo al file nell'ordine in cui arrivano al DB, e l'ordine di visualizzazione è arbitrario. Il dato non è persistente finchè non viene scritto sul FS, quindi il blocco dev'essere copiato nel buffer e il record aggiunto.
La scansione è necessariamente sequenziale, quindi lettura e aggiornamento sono lente, e i record vengono periodicamente riorganizzata per sovrascrivere le cancellazioni.

Ordinata: la posizione fisica è determinata dal campo chiave, e l'ordinamento fisico è l'ordinamento delle chiavi.
La ricerca (anche per range) è semplice, ma la creazione impiega più tempo. La cancellazione dipende dallo schema di allocazione, ma richiede riorganizzazione periodica.

Alcuni metodi per rimediare al tempo di inserimento sono l'append in coda con riordinamento periodico, (accesso logaritmico) oppure l'allocazione al posto di record cancellati. Può essere lasciato spazio libero per uso futuro, o possono esserci file di overflow con linked list.

3.2 OLAP

OLAP (On Line Analytical Processing) è la raccolta di dati il cui uso primario è per le decisioni organizzative. Un data warehouse è un carico di lavoro subject-oriented, integrato, non volatile e variabile con il tempo.

Le informazioni sono rappresentate tramite un cubo dimensionale di cui gli assi indicano il tempo, i prodotti e i punti vendita. L'organizzazione è gerarchica, in categorie.

Non esiste un metodo generale per la modellazione di un DW: il modello ER dev'essere trasformato ed esteso, i dati devono essere permanenti e sono memorizzati in blocchi contenenti righe vicine tra loro.

I dati sono salvati colonna per colonna per evitare scansioni multiple in query come COUNT, ma l'inserimento viene effettuato comunque con la riga (strutture row-oriented) che poi viene spostata.

3.3 Hash

Le tabelle hash permettono l'accesso diretto sulla base del campo chiave, in casi in cui una struttura ad array sarebbe inefficiente perché i possibili valori della chiave sono molti di più di quelli utilizzati.

I record sono organizzati in bucket (blocchi), e la funzione hash, generalmente il modulo, associa a ogni valore della chiave un indirizzo. Lo spazio delle chiavi è più grande dello spazio degli indirizzi (funzione non iniettiva) e ci sono possibilità di collisioni.

Le collisioni vengono gestite tramite tabella di overflow o extensible hashing. La tabella di overflow funziona come bucket aggiuntivo per gli hash duplicati, e l'eventuale spostamento con la cancellazione dei record.

Questo comporta la decisione della grandezza dei bucket e dei file: bisogna tenere in considerazione il numero medio di accessi. La probabilità di overflow cresce con il numero di record e decresce con la dimensione del blocco.

Con T records e F record per bucket in media, il numero dei bucket B dovrebbe essere $B = T/(0.8 \cdot F)$ per utilizzare il 50-80% dello spazio disponibile.

Le strutture hash sono efficienti per accesso diretto con il valore della chiave, ma non funzionano per ricerche basate su range o campi diversi. La dimensione dei file non deve subire variazioni significative nel tempo.

L'extensible hashing consente la crescita indefinita della struttura ed elimina l'overflow: essa aumenta dinamicamente con accesso tramite directory, la quale definisce la funzione hash.

Lo spreco di spazio è limitato, le riorganizzazioni sono solo a livello locale e il tempo di accesso è veloce. La directory va tenuta nella memoria principale.

3.4 Indicizzazione

Un indice è una struttura ausiliaria, il cui scopo è l'accesso efficiente ai record di un file sulla base dei valori di un campo o un insieme di campi (chiave o pseudochiave), non necessariamente identificanti.

Un indice è un altro file con record contenenti chiave e indirizzo, ordinato secondo i valori della chiave (indice analitico di un libro). Viene rappresentato come una struttura a cui è collegato il file principale.

3.4.1 Indice primario

L'indice primario è costituito da due campi, di cui il primo è dello stesso tipo della chiave di ordinamento del file dati e il secondo è un puntatore al blocco dei file dati. Generalmente il campo dev'essere anche chiave primaria, altrimenti l'indice si definisce di cluster.

Non è necessario che tutti i valori delle chiavi siano rappresentati, perciò lo spazio occupato è minimo. Questo concetto rende l'indice sparso (talvolta). Ogni file può avere al più un indice primario.

3.4.2 Indice secondario

L'indice secondario è definito su un campo chiave diverso da quello di ordinamento. I puntatori puntano ai record (o ai blocchi) che corrispondono al valore della chiave.

L'indice secondaria è per sua natura denso, cioè tutti i puntatori puntano necessariamente a tutti i record. Il numero di indici secondari è variabile.

3.4.3 Caratteristiche degli indici

La dimensione del file indice dipende dal numero di blocchi occupati. Si definiscono L record nel file dati, blcchi di dimensione B, R lunghezza dei record, K lunghezza del campo chiave e P lunghezza dei puntatori.

Si ha:
Numero di blocchi per file dati $\rightarrow N_F = \lceil L/(\lfloor B/R \rfloor) \rceil$
Numero di blocchi per un file di indice denso $\rightarrow N_D = \lceil L/(\lfloor B/(K+P) \rfloor) \rceil$
Numero di blocchi per un file di indice sparso $\rightarrow N_F = \lceil N_F/(\lfloor B/(K+P) \rfloor) \rceil$

Gli indici sono efficienti perché permettono accesso diretto efficiente sulla chiave, sia puntuale sia per intervalli. Questo significa che le operazioni CRUD sono lente, quindi sono soggetti a riorganizzazione periodiche o flag per le eliminazioni.

Una possibilità per evitare le ricerche tra blocchi diversi è la costruzione di indici sugli indici, quindi multilivello. Possono esistere livelli fino ad avere quello più alto con un solo blocco, ma solitamente sono pochi essendo i record piccoli e ordinati.
Numero di blocchi al livello j dell'indice $\rightarrow N_j = \lceil N_{j-1}/(\lfloor B/(K+P) \rfloor) \rceil$

3.5 Alberi

Gli indici sono strutture ordinate sequenziali, quindi poco flessibili e scarsamente adattabili alla dinamicità e ai cambiamenti. Gli alberi rimediano a questo problema.

3.5.1 Alberi di ricerca di ordine P

Un esempio è l'albero di ricerca: ogni nodo contiene un numero fisso F di chiavi e $F+1$ puntatori, con valori delle chiavi ordinate all'interno di un nodo. La rappresentazione sequenziale affianca puntatori e nodi.

Ogni chiave K_j con $0 \leq j \leq F$ è seguita da un puntatore P_j e preceduta dal puntatore P_{j-1}. Il primo puntatore è P_0. Ognuno di essi punta a un sottoalbero:

- P_0 punta al sottoalbero con valori di chiave $< K_1$;
- P_F punta al sottoalbero con valori di chiave $>= F$;
- P_j con $0 \leq j \leq F$ punta al sottoalbero con valori di chiave comprese nell'intervallo $K_j \leq K < K_{j+1}$.

Il valore $F + 1$ è il fan-out dell'albero, cioè il numero di input.

Ci sono due tipi di alberi utilizzati:

- Key-sequenced trees, con i record dati contenuti nelle foglie, tipici quando la chiave corrisponde al campo identificante;
- Indirect trees, con puntatori al record contenuti nelle foglie, qualsiasi altro meccanismo di allocazione delle tuple e tipici quando a un valore corrispondono più record (indice secondario, chiave non primaria).

Data la chiave V, la ricerca viene effettuata similmente alla ricerca binaria:

1. Se $V < K$ si procede da P_0;
2. Se $V \geq K_F$ si procede da P_F;
3. Altrimenti viene seguito il puntatore P_j tale che $K_j \leq V \leq K_{j+1}$.

L'accesso è pertanto associativo per chiave, dove il numero di blocchi per accesso singolo rappresenta la profondità dell'albero. Questo tipo di struttura permette accesso efficiente sia per valore che per intervallo, quindi è la più comune nei DBMS.

3.5.2 Alberi bilanciati

Si ricorda che un albero bilanciato ha un numero variabile di foglie tale che la lunghezza dei cammini verso ognuna di esse sia la stessa. Ciò è ottenuto tramite operazioni di split e merge, e permette un tempo quasi costante (logaritmico) per gli accessi. Il numero di blocchi corrisponde alla profondità dell'albero.

Le strutture ad albero bilanciate si basano sull'utilizzo di un indice multilivello, struttura ausiliaria per l'accesso efficiente ai record tramite un campo chiave (non necessariamente identificante).

Ci sono due tipi di alberi bilanciati:

- B+ trees, con foglie concatenate secondo l'ordine della chiave. I valori possono essere ripetuti, e la ricerca è ottimizzata.
  Potenziale maggiore occupazione di memoria, e supporto efficiente di range queries;
- B trees, senza concatenamento sequenziale delle foglie. I nodi interni usano due puntatori per ogni valore K_k della chiave, di cui uno punta al blocco che contiene il record e uno punta al sottoalbero con chiavi comprese tra K_j e K_{j+1}.
  Inserimento e cancellazione sono preceduti da una ricerca fino a una foglia, e se non c'è spazio libero il nodo va diviso.

Minore occupazione di memoria, ma le ricerche relative a chiavi che puntano direttamente al record sono più efficienti.

L'occupazione di memoria, attraverso l'esecuzione delle operazioni di aggiornamento, è mantenuta tra il 50% e il 100%. L'efficienza è alta, perché le pagine ai primi livelli (accedute spesso) sono spesso nel buffer: ciò può comportare racing e deadlock, ma è poco probabile dato che le transazioni sono generalmente letture.

4 Architettura DBMS e MySQL

MySQL è un esempio di DBMS che utilizza le strutture fisiche e i metodi di accesso menzionati precedentemente. L'architettura è composta da elementi come query compiler, gestore delle transazoni, gestore del buffer e gestore dei metodi di accesso.

MySql Pluggable Storage Engine Architecture consente ai DBA di selezionare sistemi di storage diversi, di cui ognuno è specializzato per particolari applicazioni.

Storage engine disponibili:

1. MyISAM, motore default, utilizzato per applicazioni web;
2. InnoDB, implementa sistemi transazionali e garantisce alcune proprietà ACID;
3. Memory, memorizza i dati in memoria centrale per migliorare l'accesso;
4. Merge, supporta l'integrazione in formato MyISAM per considerare più tabelle come un oggetto;
5. Archive, per archivi di grandi dimensioni poco acceduti;
6. Federated, per dati distribuiti con un unico schema logico;
7. Cluster/NDB, per high performances e alta disponibilità.

MyISAM è stato il sistema di storage di default fino alla versione 3.2, non è transazionale: ogni database era una directory e ogni tabella un file. Gestisce record in formato fisso e dinamico (dati variabili), con meccanismo di accesso a matrice. Gli indici sono BTREE, RTREE e FULLTEXT, e la concorrenza è solo a livello di tabella.

InnoDB usa il concetto di tablespace per cui la struttura, la tabella e gli indici sono memorizzati insieme. Sono implementati gli indici BRTEE, e le interrogazioni più frequenti hanno tabelle di hash corrispondenti. Il controllo di concorrenza ha caratteristiche multi-versioning, low-level locking e foreign key constraints.

Memory non prevede memorizzazione in memoria persistente, solo centrale. Supporta indici hash e tree-based.

Il DBA può decidere quale storage engine usare per ogni tabella tramite CREATE TABLE. I vari storage differiscono per funzionalità e per velocità di accesso ai dati.