OTTIMIZZAZIONE DELLE INTERROGAZIONI

Angelo Montanari
Dipartimento di Matematica e Informatica
Università di Udine

INTRODUZIONE

Il modulo OTTIMIZZATORE decide le strategie di accesso ai dati per rispondere alle interrogazioni.

Attività preliminari:

L'ottimizzatore è un modulo importante e classico della architettura di una base di dati. Esso riceve in ingresso una interrogazione scritta in SQL, passata tramite un qualunque meccanismo della interfaccia applicativa (ad esempio, tramite cursori o procedure). La interrogazione viene inizialmente analizzata per determinare eventuali suoi errori lessicali, sintattici o semantici, che vengono indicati all'utente per la correzione dell'interrogazione. Durante questa fase, il sistema accede al dizionario dei dati per leggere l'informazione ivi contenuta e consentire così i controlli; dal dizionario dei dati vengono lette anche informazioni statistiche relative alle dimensioni delle tabelle. Una volta accettata, l'interrogazione viene tradotta in una forma interna di tipo algebrico.

Attività di ottimizzazione:

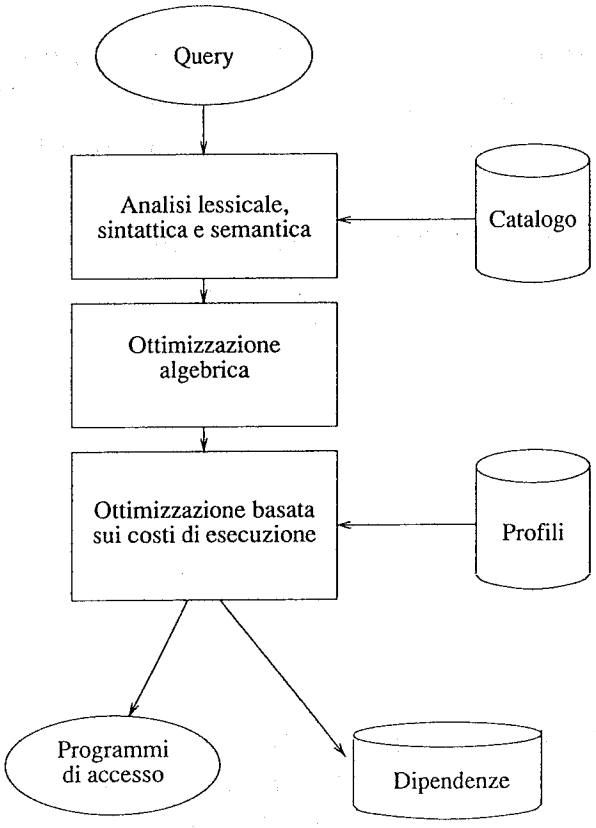
- Innanzitutto viene svolta una ottimizzazione di tipo algebrico, che consiste nell'effettuare tutte le trasformazioni algebriche che sono sempre convenienti, quali ad esempio il "push" delle selezioni e delle proiezioni, come descritto nel capitolo 3. Questa ottimizzazione, di tipo logico, avviene indipendentemente dal modello dei costi assunto nel sistema.

- Successivamente, viene svolta una ottimizzazione che dipende sia dalla tipologia dei metodi di accesso ai dati supportati dal sottostante livello, sia dal modello dei costi assunto. Per questa fase, pur essendo l'ottimizzazione delle interrogazioni assai ben definita e frutto di tecniche consolidate (specie nel contesto dei sistemi relazionali), ciascun sistema presenta di fatto caratteristiche peculiari.

- Infine, avviene la generazione del codice, che utilizza i metodi di accesso ai dati. Si ottengono cioè dei *programmi di accesso* in formato "oggetto" o "interno" che richiedono l'uso delle strutture dati fornite dal sistema (tra cui, ad esempio, gli indici).

COMPILAZIONE DI UNA INTERROGAZIONE

Il processo di ottimizzazione di un'interrogazione:



Approccio compile-and-store vs compile-and-go:

a differenza di tutti gli altri moduli di sistema descritti in questo capitolo, l'ottimizzatore è un modulo che agisce a tempo di compilazione. Nel caso comune in cui la interrogazione venga compilata una volta ed eseguita molteplici volte (approccio "compile-and-store"), il codice viene prodotto e memorizzato nella base di dati, assieme ad una indicazione delle *dipendenze* del codice dalle particolari versioni di tabelle e indici della base di dati, lette nel dizionario dati. In tal modo, se la base di dati cambia in modo significativo per la interrogazione (ad esempio perché viene aggiunto un indice), la compilazione dell'interrogazione viene invalidata e ripetuta. Talvolta, invece, una interrogazione viene compilata ed immediatamente eseguita (approccio "compile-and-go"), senza che essa venga memorizzata.

LE DUE FASI DEL PROCESSO DI OTTIMIZZAZIONE

FASE 1.

Viene eseguita un'ottimizzazione algebrica che produce una descrizione ottimizzata dell'interrogazione, in cui sono state effettuate tutte le possibili trasformazioni algebriche.

Il risultato di tale lavoro rappresenta ogni interrogazione (SQL) in strutture ad albero, i cui nodi foglia rappresentano tabelle ed i cui nodi intermedi rappresentano operazioni dell'algebra relazionale.

FASE 2.

Viene eseguita un'ottimizzazione basata sui costi. Essa dipende fortemente dalle specifiche strutture di memorizzazione utilizzate e dal modello di costi prescelto. Nel seguito forniremo una descrizione "qualitativa" di tale fase.

PROFILI DELLE RELAZIONI

Ogni DBMS commerciale possiede informazioni quantitative relative alle caratteristiche delle tabelle, organizzate in strutture dati dette profili delle relazioni, che vengono memorizzate nel dizionario dei dati.

I profili contengono (parte del) le seguenti informazioni:

- La *cardinalità* CARD(T_i) (numero di tuple) di ciascuna tabella T_i.
- La *dimensione* in byte, SIZE(T_i), di ciascuna tupla di T_i.
- La *dimensione* in byte, SIZE(A_j), di ciascun attributo A_j.
- Il *numero di valori distinti*, VAL(A_j), di ciascun attributo A_j.
- I valori *minimo* MIN(A_j) e *massimo* MAX(A_j) di ciascun attributo A_j.

Come (e quando) vengono calcolati i profili?

I profili vengono calcolati in base ai dati effettivamente memorizzati nelle tabelle, richiamando opportune primitive di sistema (ad esempio, la primitiva `update statistics`); è compito dell'amministratore della base di dati richiamare periodicamente queste primitive. Normalmente viene esclusa la possibilità di tenere aggiornati i profili durante la normale esecuzione delle transazioni, perché questa opzione è troppo costosa. In genere, è sufficiente che i profili contengano valori approssimati, in quanto comunque i modelli statistici che si applicano ad essi sono anch'essi approssimati.

PROFILI DELLE OPERAZIONI - 1

Non basta calcolare i profili delle relazioni (tabelle) memorizzate nella base di dati....

La ottimizzazione dipendente dai costi richiede di formulare ipotesi sulle dimensioni dei risultati intermedi prodotti dalla valutazione di operazioni algebriche con un approccio di tipo statistico. Ad esempio, vediamo i profili relativi alle operazioni algebriche più classiche: selezione, proiezione e join.

Vediamo i profili relativi alle operazioni algebriche più ``classiche'': selezione, proiezione e join.

Formule dei profili relativi alle selezioni Il profilo di una tabella T prodotta da una *selezione* $T = \sigma_{A_i=v}R$ è ottenuto tramite le seguenti formule, la cui giustificazione è lasciata come esercizio:

1. $\text{CARD}(T) = 1/\text{VAL}(A_i) \times \text{CARD}(R)$;
2. $\text{SIZE}(T) = \text{SIZE}(R)$;
3. $\text{VAL}(A_i) = 1$;
4. $\text{VAL}(A_j) = col(\text{CARD}(R), \text{VAL}(A_j), \text{CARD}(T))$, $j \neq i$; [2]
5. $\text{MAX}(A_i) = \text{MIN}(A_i) = v$;
6. $\text{MAX}(A_j)$ e $\text{MIN}(A_j)$ mantengono gli stessi valori.

---

[2] La formula $col(n,m,k)$, usata nella formula relativa a $\text{VAL}(A_j)$, calcola il numero di valori distinti presenti in k oggetti estratti a partire da n oggetti di m colori distinti, distribuiti omogeneamente; ciascun colore rappresenta uno dei diversi valori presenti nell'attributo A_j. Tale formula ammette la seguente approssimazione:

(a) $col(n,m,k) = k$ se $k \leq m/2$

(b) $col(n,m,k) = (k+m)/3$ se $m/2 \leq k \leq 2m$

(c) $col(n,m,k) = m$ se $k \geq 2m$.

PROFILI DELLE OPERAZIONI - 2

Formule dei profili relativi alle proiezioni Il profilo di una tabella T prodotta da una *proiezione* $T = \pi_L R$, ove L è l'insieme di attributi $A_1, A_2, \ldots A_n$, è ottenuto tramite le seguenti formule:

1. $\text{CARD}(T) = \text{MIN}(\text{CARD}(R), \prod_{i=1}^{n} \text{VAL}(A_i))$;
2. $\text{SIZE}(T) = \sum_{i=1}^{n} \text{SIZE}(A_i(R))$;
3. $\text{VAL}(A_i)$, $\text{MAX}(A_i)$, $\text{MIN}(A_i)$ mantengono gli stessi valori.

Formule dei profili relativi ai join Il profilo di una tabella T prodotta da un *equi-join* $T = R \bowtie_{A=B} S$, assumendo che A e B abbiano domini identici ed in particolare $\text{VAL}(A) = \text{VAL}(B)$.

1. $\text{CARD}(T) = (1/\text{VAL}(A)) \times \text{CARD}(R) \times \text{CARD}(S)$;
2. $\text{SIZE}(T) = \text{SIZE}(R) + \text{SIZE}(S)$;
3. $\text{VAL}(A_i)$, $\text{MAX}(A_i)$, $\text{MIN}(A_i)$ mantengono gli stessi valori.

Le formule precedenti mostrano i limiti di questo tipo di analisi statistica. Ad esempio, tutte le formule assumono una uniforme distribuzione dei dati nelle tabelle e una assenza di correlazione tra le varie condizioni presenti in una interrogazione. Si noti che spesso le formule assegnano al risultato di una operazione parametri identici a quelli dei loro operandi (ad esempio, per quanto riguarda i valori minimi e massimi di un certo attributo) in quanto non è possibile prevedere meglio gli effetti dell'operazione stessa. Comunque, questa analisi statistica è in grado di definire, anche approssimativamente, l'ordine di grandezza delle dimensioni dei risultati intermedi (ad esempio, in numero di pagine occupate); questi dati quantitativi sono sufficienti a svolgere l'ottimizzazione.

RAPPRESENTAZIONE INTERNA DELLE INTERROGAZIONI

La rappresentazione interna di un'interrogazione prodotta dall'ottimizzatore tiene conto di:

- la struttura fisica utilizzata per implementare le tabelle;

- gli indici disponibili su di esse.

Il primo passo consiste nella trasformazione dei nodi foglia in nodi che tengano conto delle strutture fisiche utilizzate.

Il secondo passo consiste nella trasformazione dei nodi intermedi in operazioni di accesso ai dati (memorizzati in opportune strutture fisiche) supportate dal sistema.

Le tipiche operazioni supportate dai DBMS relazionali sono le scnasioni sequenziali, gli ordinamenti, gli accessi tramite indici e varie modalità di join.

LE OPERAZIONI DI SCANSIONE

Operazioni di scansione Una operazione di *scansione* (*scan*) opera contestualmente varie operazioni di tipo algebrico ed extra-algebrico:

- Proiezione su di una lista L_p di attributi;
- Selezione su di un predicato semplice (del tipo: $A_i = v$);
- Ordinamento (sort) in base al valore assunto dagli attributi presenti in una lista L_s di attributi;
- Inserzioni, cancellazioni e modifiche delle tuple quando esse vengono accedute durante la scansione.

Durante una scansione, viene mantenuto sempre un puntatore alla tupla corrente; la scansione viene realizzata tramite l'uso delle seguenti primitive:

- La primitiva *open* inizializza la scansione.
- La primitiva *next* fa procedere la scansione, avanzando il puntatore alla tupla corrente.
- La primitiva *read* legge la tupla corrente.
- Le primitive *modify* e *delete* agiscono sulla tupla corrente, modificandone il contenuto oppure cancellandola.
- La primitiva *insert* inserisce una nuova tupla nella posizione corrente.
- La primitiva *close* conclude la scansione.

GLI ORDINAMENTI

Ordinamenti Il problema di ordinare strutture dati è un problema classico della teoria degli algoritmi; vari metodi, descritti in letteratura, consentono di ottenere prestazioni ottimali per ordinare dati contenuti in memoria centrale, tipicamente rappresentati tramite array di record. Le tecniche di ordinamento dei dati utilizzate dai DBMS sfruttano questi algoritmi, che non sono descritti ulteriormente. Un DBMS deve però risolvere un secondo problema, relativo al caricamento dei dati nei buffer. A volte, infatti, non è possibile caricare tutti i dati nei buffer, per la eccessiva dimensione dei dati o l'impossibilità di assegnare a questa operazione un ampio numero di buffer.

Supponendo di voler ordinare un file che consta di N blocchi avendo a disposizione soltanto M pagine nel buffer, con $M < N$, si può procedere operando innanzitutto con K ordinamenti separati, ove K è l'intero immediatamente superiore al rapporto (N/M); si ottengono così K file ordinati. Dopodiché, sono necessarie $K - 1$ operazioni di *merge*, in cui cioè a partire da due file ordinati se ne ottiene un terzo ordinato. L'operazione di merge richiede di aprire due scansioni sui due file, confrontare il valore della chiave di ordinamento nelle due tuple correnti, e trasferire in un terzo file la tupla col valore più piccolo, avanzando contemporaneamente la scansione del corrispondente file. Data la necessità di tenere aperte tre scansioni, è necessario avere disponibili perlomeno tre pagine nel buffer.

GLI ACCESSI TRAMITE INDICI

Accesso tramite indici Gli indici, realizzati tramite strutture ad albero, vengono predisposti dall'amministratore della base di dati per favorire l'accesso associativo di interrogazioni che comprendono predicati semplici (del tipo $A_i = V$) oppure di intervallo (del tipo $V_1 \leq A_i \leq V_2$). Si dice in tal caso che un predicato della interrogazione è *valutabile* tramite l'indice.

In genere, se la interrogazione presenta un solo predicato valutabile (anche in presenza di altri predicati in congiunzione) c'e convenienza ad usare l'indice. Quando una interrogazione presenta una *congiunzione* di predicati valutabili tramite indice, il DBMS sceglie il più selettivo dei due per l'accesso primario via indice; il secondo predicato viene valutato una volta caricate nel buffer le pagine che soddisfano il primo predicato. Quando invece la interrogazione presenta una *disgiunzione* di predicati, basta che uno di loro sia non valutabile per imporre l'uso di una scansione completa. Se infine si ha una disgiunzione di predicati tutti valutabili, è possibile utilizzare gli indici oppure una scansione; nel caso di uso di indici, però, bisogna stare attenti a eliminare i duplicati di quelle tuple che vengono ritrovate tramite più indici.

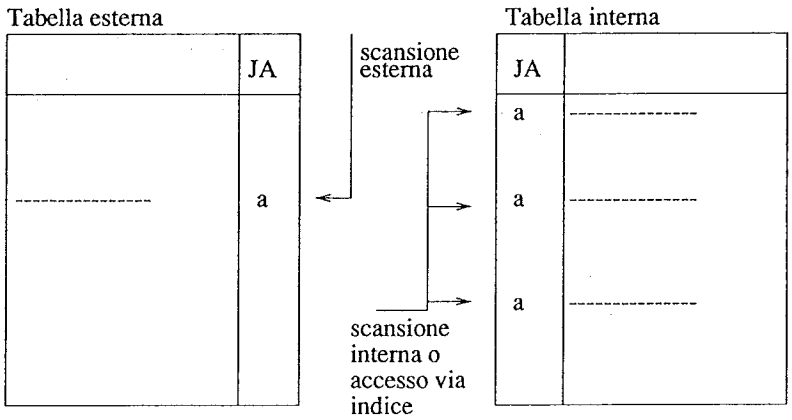
Infine, occorre tener presente che l'uso dell'indice richiede molteplici accessi per ogni tupla individuata; quando la interrogazione è poco selettiva, l'uso dell'indice può essere dominato dall'uso di una semplice scansione.

METODI DI JOIN -1

Metodi di join Il join è ritenuto l'operazione più gravosa per un DBMS, in quanto è presente il rischio di una esplosione del numero di tuple del risultato. Pertanto, non stupisce che la tecnologia dei DBMS abbia prodotto vari algoritmi per la realizzazione dei join, e che definire l'ordine e la modalità di realizzazione delle operazioni di join abbia un ruolo centrale nell'ottimizzazione globale delle interrogazioni. Solo in tempi recenti, con il crescere dell'interesse per le operazioni aggregate, analoghi algoritmi e approcci quantitativi sono stati dedicati alle operazioni aggregate e al raggruppamento. Nel seguito, vediamo tre tecniche per la realizzazione dei join, denominate *nested-loop*, *merge-scan*, e *hash-based*.

Nested loop:

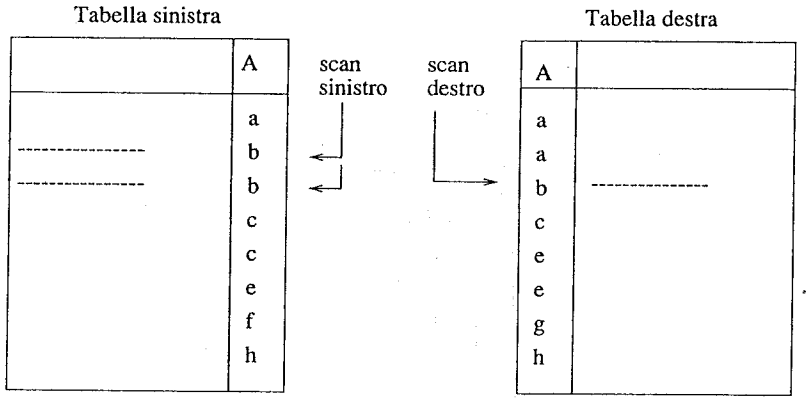
- *Nested loop.* Nel nested-loop una tabella viene definita come *esterna* e una come *interna*. Si apre una scansione sulla tabella esterna; per ogni tupla ritrovata dalla scansione, si preleva il valore dell'attributo di join, e si cercano quindi le tuple della tabella interna che hanno lo stesso valore. Per questa seconda parte dell'algoritmo è opportuno che esista un indice sull'attributo di join della tabella interna, che può essere creato ad-hoc; altrimenti, è necessario aprire uno scan sulla tabella interna per ogni valore di join della tabella esterna; ciò è possibile se l'intera tabella interna può essere allocata nel buffer. Il nome "nested loop" viene dato a questa tecnica proprio perché propone una scansione "nidificata" nell'altra. Si noti che questa tecnica ha costi diversi a seconda della attribuzione alle tabelle operandi dei ruoli di interna e esterna.



METODI DI JOIN - 2

Merge scan:

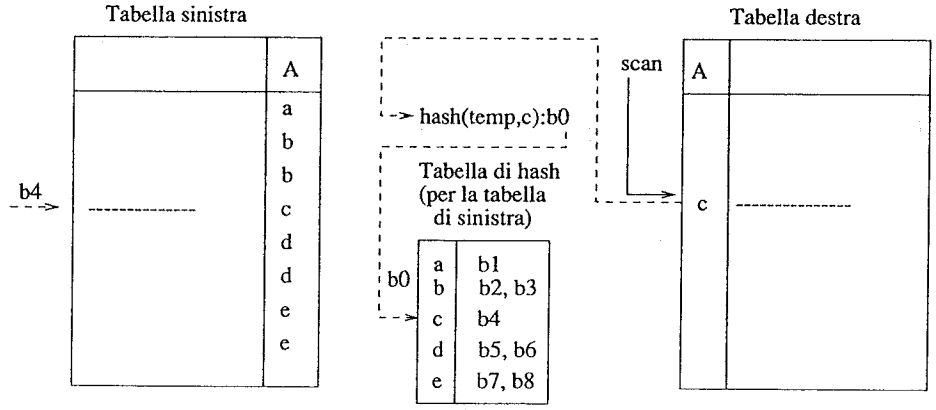
- *Merge-scan*: Questa tecnica richiede che entrambe le tabelle vengano ordinate in base agli attributi di join. Successivamente, vengono aperte due scansioni su di esse, che a questo punto scorrono le tuple in parallelo. Le scansioni possono così ritrovare nelle tuple valori ordinati degli attributi di join; quando coincidono, vengono generate ordinatamente tuple del risultato. Le scansioni vengono eseguite in modo da garantire che tutte le coppie di tuple con valori identici degli attributi di join diano luogo ad una tupla del risultato.



METODI DI JOIN - 3

Hash join:

- *Hash join*: Questo metodo richiede di costruire una *tabella di hash* per gli attributi di join di una delle tabelle, cioè una tabella ausiliaria che, ad ogni valore dell'attributo di join, restituisce identificatori di tupla (o puntatori) alle tuple della tabella con questo valore. A questo punto, la seconda tabella può essere acceduta con una tecnica qualunque (ad esempio una scansione), ritrovando il valore dell'attributo di join; questo valore viene utilizzato per un accesso "via hashing" alla tabella precedentemente costruita, e da essa alle tuple che soddisfano la condizione di join. Si noti che anche questa tecnica ha costi diversi a seconda della scelta dell'operando del join su cui costruire la tabella di hash.



bi: puntatori ai blocchi

Confronto fra le tre tecniche:

Le tre tecniche descritte si basano sull'uso combinato di tecniche descritte a monte, quali la scansione, l'accesso via indice e l'ordinamento, utilizzate come blocchi elementari nell'ambito della strategia. È chiaro che ciascuna strategia ha un costo che dipende dalla "situazione iniziale" cui si applica (ad esempio, la presenza di indici o la possibilità di caricare un operando completamente nei buffer di memoria centrale date le sue ridotte dimensioni) e produce una "condizione finale" (ad esempio, il fatto che il risultato si presenti in modo ordinato). Pertanto, il costo di realizzare una qualunque di queste tecniche non può essere valutato in astratto, ma deve essere valutato in funzione delle scelte che la precedono o seguono.

OTTIMIZZAZIONE BASATA SUI COSTI - 1

Infine, vediamo come opera un ottimizzatore globale. Il problema si presenta assai difficile su un piano computazionale, in quanto sono presenti varie dimensioni di ottimizzazione.

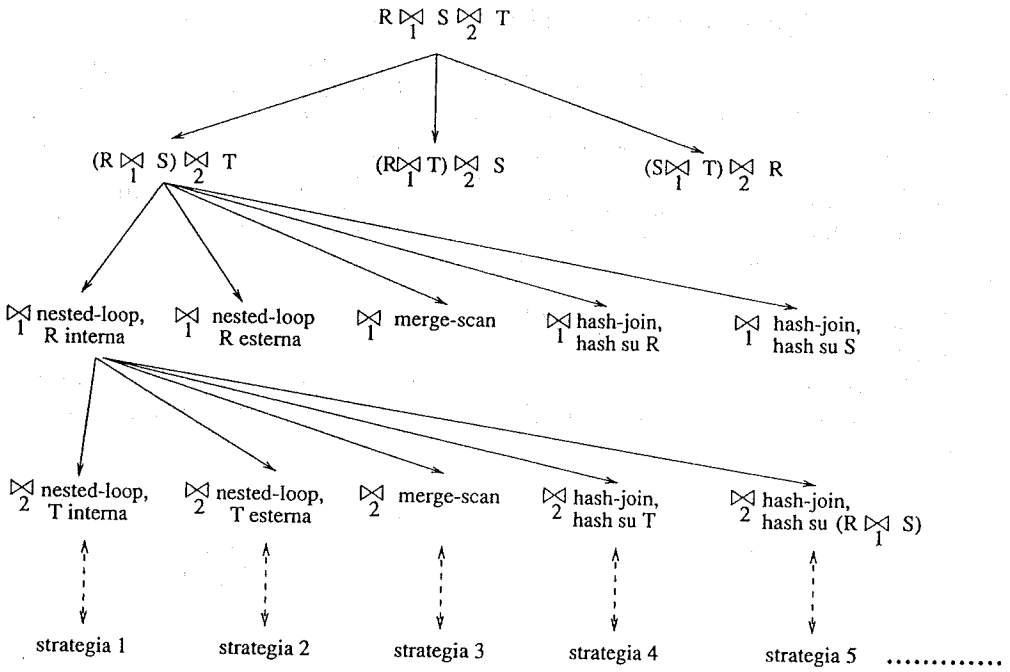
- Occorre scegliere, in presenza di alternative, quali operazioni di accesso ai dati svolgere. In particolare, per quanto concerne il primo accesso ai dati, occorre talvolta scegliere fra uno scan e un accesso tramite indici.
- Occorre scegliere l'ordine delle operazioni da compiere (ad esempio, l'ordine fra i vari join presenti in una interrogazione).
- Quando un sistema offre varie alternative per la realizzazione di una operazione, occorre scegliere quale alternativa associare a ciascuna operazione (ad esempio, scegliere il metodo di join).
- Quando la interrogazione oppure il metodo di realizzazione di una operazione richiedono un ordinamento, occorre definire a quale livello della strategia svolgere l'operazione di ordinamento.

Ulteriori opzioni si presentano all'atto di scegliere una strategia in un contesto distribuito.

Di fronte ad un problema così complesso, gli ottimizzatori dispongono in genere di formule di costo approssimate. Essi costruiscono un *albero delle alternative*, in cui ogni nodo corrisponde a fissare una particolare opzione fra quelle citate in precedenza. Ovviamente, le dimensioni di tale albero crescono esponenzialmente in funzione del numero di alternative presenti. Ogni nodo foglia dell'albero corrisponde ad una specifica *strategia di esecuzione* della interrogazione, descritta dalle scelte che si trovano percorrendo il cammino che va dalla radice al nodo foglia. Quindi, il problema di ottimizzazione è riformulato nella ricerca del nodo-foglia cui corrisponde il costo minore.

OTTIMIZZAZIONE BASATA SUI COSTI - 2

In figura 9.26 mostriamo le alternative di esecuzione di una interrogazione congiuntiva (cioè con sole selezioni, proiezioni e join) con tre tabelle e due metodi di join, in cui l'ottimizzatore deve decidere solo l'ordinamento e il metodo di join da utilizzare. Sono presenti 3 possibili ordinamenti dei join e 5 possibili modi per svolgere operazioni di join, dando luogo a 75 alternative. Questo semplice esempio è indicativo della complessità del problema nei suoi termini più generali.



OTTIMIZZAZIONE BASATA SUI COSTI - 3

Il problema viene tipicamente risolto tramite formule di costo che associano a ciascuna operazione intermedia un costo in termini di operazioni di ingresso/uscita e di istruzioni necessarie per valutare il risultato della interrogazione. In questo modo, è possibile associare ad un nodo foglia un costo:

$$C_{total} = C_{I/O} \times n_{I/O} + C_{cpu} \times n_{cpu}$$

Ove $C_{I/O}$,C_{cpu} sono parametri noti e $n_{I/O}$, n_{cpu} sono valori globali che indicano il numero di operazioni di ingresso/uscita e il numero di istruzioni necessarie per valutare il risultato della interrogazione. Tale costo è dato dalla somma di tutti i costi accumulati ai nodi intermedi per svolgere le operazioni che caratterizzano la strategia.

Talvolta, la strategia ottimale richiede la realizzazione ad-hoc di strutture temporanee, ad esempio indici o ordinamenti via hash, e quindi il costo della strategia include il costo di tali strutture. I risultati intermedi prodotti vengono spesso allocati nei buffer e consumati subito dopo la loro produzione, sfruttando il *pipelining* delle operazioni, cioè la possibilità di percorrere l'intero albero delle operazioni per una parte delle tuple estratte, invece che procedere a realizzare interamente ciascuna operazione. Talvolta, invece, è necessario riscrivere i risultati delle operazioni intermedie in memoria di massa; in tal caso, fa parte dei costi di una strategia anche il costo di riscrittura dei risultati intermedi.

Gli ottimizzatori si accontentano tipicamente di ottenere soluzioni "buone", cioè che rientrino nello stesso "ordine di grandezza prestazionale" rispetto alla soluzione ottima del problema. Essi possono escludere interi sotto-alberi quando il loro costo parziale è peggiore del costo di una strategia globale; seguono quindi tecniche di ricerca operativa, denominate *branch and bound*, per la eliminazione esatta o approssimata di sotto-alberi.

Con un approccio "compile-and-go", gli ottimizzatori sono spesso vincolati ad operare in tempi ristretti; infatti, i tempi di risposta non possono essere penalizzati dal dedicare troppo tempo alla ottimizzazione. Non ha senso trovare la soluzione ottima in un tempo molto elevato, quando è possibile trovare una soluzione "buona" in un tempo limitato e realizzare questa strategia in un tempo globale (che cioè include il tempo dedicato alla ottimizzazione) inferiore a quello della soluzione ottima. In conclusione, gli ottimizzatori sono moduli assai complessi, da cui dipende gran parte dell'efficienza del DBMS nel valutare interrogazioni.

PROGETTAZIONE FISICA DI UNA BASE DI DATI - 1

Nell'ambito del processo di progettazione di una base di dati, come lo abbiamo esaminato nella terza parte del libro, la fase finale è quella della *progettazione fisica*, che, ricevendo in ingresso lo schema logico della base di dati, le caratteristiche del sistema scelto e le previsioni sul carico applicativo, produce in uscita lo schema fisico della base di dati, costituito dalle effettive definizioni delle relazioni e soprattutto delle strutture fisiche utilizzate, con i relativi parametri.

L'attività di progettazione fisica di una base di dati relazionale può essere molto complessa, perché oltre alle scelte relative alle strutture fisiche (che possono essere compiute in una rosa più o meno ampia, a seconda dei sistemi, come abbiamo visto nella sezione precedente) può essere necessario definire molti parametri, che vanno dalle dimensioni iniziali dei file alle possibilità di espansione, dalla contiguità di allocazione alla quantità e alle dimensioni delle aree di transito per scambio di informazioni tra memoria principale e memoria secondaria. Alcuni sistemi mettono a disposizione varie decine di parametri i cui valori possono risultare significativi ai fini delle prestazioni delle applicazioni. Peraltro, esistono praticamente sempre valori standard per questi parametri, che vengono assunti dal sistema se non altrimenti specificati in modo esplicito.

La maggior parte delle scelte da effettuare nel corso della progettazione fisica dipende in effetti dallo specifico sistema di gestione utilizzato, e quindi risulta difficile fornire, nel poco spazio che possiamo dedicarle, una panoramica completa. Indicheremo solo le linee principali, che possono però essere considerate sufficienti in presenza di basi di dati di dimensioni non enormi o con carichi di lavoro non particolarmente complessi. Assumiamo che il DBMS a nostra disposizione preveda solo file non ordinati, con possibilità di definizione di indici. In questo contesto, ignorando gli altri parametri, la progettazione fisica può essere ricondotta ad una attività di individuazione degli indici da definire su ciascuna relazione (a parte la attività materiale ma semplice di specifica degli schemi delle relazioni nello specifico DDL del sistema utilizzato, con i tipi dei dati ammessi per i vari attributi).

Per orientarci nella scelta degli indici, è opportuno ricordare che, come detto nel paragrafo 9.6, le operazioni più delicate in una base di dati relazionale sono quelle di selezione (che corrisponde all'accesso ad uno o più record sulla base dei valori di uno o più attributi) e di join (che richiede di combinare ennuple di relazioni diverse sulla base dei valori di uno o più attributi di ciascuna di tali relazioni).

Ciascuna delle due operazioni può essere eseguita in modo molto più efficiente se sui campi interessati è definito un indice, che permette un accesso diretto.

PROGETTAZIONE FISICA DI UNA BASE DI DATI - 2

Consideriamo ad esempio una base di dati su due relazioni, la prima, IMPIEGATI, sugli attribuiti Matricola (che costituisce la chiave) Cognome, Nome e Dipartimento (che indica il codice del Dipartimento di afferenza) e la seconda, DIPARTIMENTI, sugli attributi Codice (che costituisce la chiave), Nome e Direttore.

Volendo effettuare sulla relazione IMPIEGATI una selezione sull'attributo Matricola (ricerca di un impiegato dato il numero di matricola), se sulla relazione è presente un indice su tale attributo si può procedere con un accesso diretto, molto efficiente, altrimenti si deve effettuare un accesso sequenziale, con un costo proporzionale alla dimensione del file. Lo stesso vale per una ricerca basata sul cognome dell'impiegato; vale la pena notare che se è definito un indice su un attributo, solo le ricerche basate su tale attributo possono trarne beneficio: se la relazione ha un indice su Matricola e non ha un indice su Cognome, le selezioni su Matricola potranno essere eseguite in modo efficiente mentre quelle su Cognome rimarranno inefficienti.

Un equi-join fra le due relazioni volto a collegare ciascun impiegato con il corrispondente dipartimento, in presenza di un indice sulla chiave Codice della relazione DIPARTIMENTI, può essere effettuato in modo molto efficiente tramite il metodo di *nested loop*: si scandisce sequenzialmente la relazione IMPIEGATI (e questo non è un problema, perché tutte le sue ennuple contribuiscono al risultato) e, per ciascuna di esse, si effettua un accesso diretto alla relazione DIPARTIMENTI sulla base dell'indice. Se l'indice non è definito, l'accesso alla relazione DIPARTIMENTI risulta inefficiente, e tutto il join risulta molto più costoso (sono possibili leggeri miglioramenti con altri algoritmi, ma senza raggiungere le prestazioni che si hanno con l'indice).

PROGETTAZIONE FISICA DI UNA BASE DI DATI - 3

È importante ricordare come molti dei join che si presentano nelle nostre applicazioni sono equi-join e per almeno una delle due relazioni i campi coinvolti formano una chiave, come nell'esempio appena mostrato. Al tempo stesso, possiamo notare come quasi sempre la chiave di una relazione sia coinvolta in operazioni di selezione o di join (o entrambe). Pertanto, possiamo dire che è ragionevole definire, su ciascuna relazione, un indice in corrispondenza della relativa chiave. Inoltre, possono essere definiti ulteriori indici su altri campi su cui vengono effettuate operazioni di selezione oppure su cui è richiesto un ordinamento in uscita (perché un indice ordina logicamente i record di un file, rendendo nullo il costo di un ordinamento). Queste osservazioni costituiscono la base per una semplice strategia di progettazione fisica.

Definiti in questo modo gli indici, si può sperimentare sul campo il comportamento della nostra applicazione: se le prestazioni risultano insoddisfacenti, si possono aggiungere altri indici, procedendo però con grande attenzione, in quanto l'aggiunta di un indice comporta un aggravio del carico per far fronte alle operazioni di modifica. Talvolta, inoltre, il comportamento del sistema è imprevedibile, e l'aggiunta di indici non altera la strategia di ottimizzazione delle interrogazioni principali, risultando del tutto inefficace. È buona norma, dopo l'aggiunta di un indice, verificare che le interrogazioni ne facciano uso (in genere, esiste un comando `show plan` che descrive la strategia di accesso scelta dal DBMS). Per questo motivo, l'attività di scelta degli indici nell'ambito del progetto fisico delle basi di dati relazionali è svolta spesso in modo empirico, con un approccio per tentativi; più in generale, l'attività di *regolazione* (*tuning*) del progetto fisico consente spesso di migliorare le prestazioni della base di dati.