UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Scienze dell'Informazione

Progetto e Sviluppo di un'Applicazione Web per il Calcolo e la Visualizzazione di Piani di Accesso a Basi di Dati

Daniel Stoilov

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2007/2008

PAROLE CHIAVE

Base di Dati

Ottimizzatore

Interrogazione

Indici

HTML

PHP

Indice

Elenco delle figure

Introduzione

La continua evoluzione di Internet, la programmazione di siti web dinamici e l'interesse in questo ambito hanno determinato la scelta di un progetto, lo scopo del quale è creare un query-optimizer e valutare quale sia la migliore strategia di accesso (access path) per le interrogazioni SQL degli utenti.

Il query-optimizer creato è principalmente uno strumento didattico, cioè è in grado di eseguire un sottoinsieme dei calcoli effettuati da un vero query optimizer ma soprattutto è in grado di aiutare chi intende studiare o capire meglio questi argomenti, grazie alla semplice interfaccia grafica e alla spiegazione dettagliata passo passo fornita in output, analoga a quella che si trova nelle soluzioni dei relativi esercizi sui libri di testo. E' ovvio che i moduli di query optimization dei DBMS hanno una complessità molto maggiore ma invece non presentano questi aspetti "educativi", perché hanno ovviamente altre finalità.

Il progetto si basa su un sito web, realizzato mediante una serie di script in linguaggi come HTML, CSS e PHP che consentono una facile implementazione delle formule per il calcolo della migliore strategia di accesso per le interrogazioni SQL. Per il suo funzionamento il programma necessita l'installazione di un web server Apache.

Il query-optimizer richiede l'inserimento di una query dall'utente ed elabora i dati inseriti, dopo di che fornisce un elenco di tutte le relazioni della query e da la possibilità di specificare/configurare il numero di tuple del file, il numero di pagine del file, il numero di valori distinti della chiava, il numero di foglie dell'indice, inoltre

l'utente può specificare il tipo dell'indice da utilizzare come segue: indice clustered, unclustered ordinato e disordinato. Il passo finale è il calcolo del costo di accesso seguito da alcuni analisi dei risultati ottenuti.

La struttura di questa tesi è organizzata in quattro capitoli il cui contenuto verrà di seguito illustrato.

Nel capitolo uno, vengono analizzate e descritte le tecniche per la valutazione ed il calcolo del costo di accesso per le interrogazioni SQL. Inoltre è stato proposto un esempio per la migliore comprensione del problema.

Il capitolo due comprende gli studi fatti sulla tecnologia PHP. Sono state studiate le caratteristiche principali del linguaggio.

Il capitolo tre comprende gli analisi dettagliata dei requisiti che hanno portato alla realizzazione del programma.

Nel ultimo capitolo viene analizzato una parte del codice HTML e PHP

Parte I

Il caso di studio

Capitolo 1

Ottimizzazione di Query

Molto spesso l'efficienza di una applicazione, sia essa una pagina ASP/JSP/PHP o un vero programma, dipende dall'efficienza del sottostante database. Ma un database efficiente non serve a nulla se le query che facciamo sono il massimo dell'inefficienza. Come si puo' migliorare l'efficienza di una Query?

1.1 Trasformazione delle Query

Quando una query SQL e' inviata al "motore" di un database (RDBMS), questo effettua diversi passaggi per trasformare la query in una sequenza di record che rispondano a determinate caratteristiche.

I passaggi possono essere diversi per query che non ritornano risultati (query di comando) e query che ritornano dei risultati (query di interrogazione). Più interessanti sono le query di interrogazione, dato che sono queste che più spesso richiedono ottimizzazione.

1.1.1 Parsing

Una volta che la query viene ricevuta dal motore del database, il primo passaggio è il "parsing", cioè la separazione della query nelle sue componenti. Questo processo ha due funzioni principali:

verificare la correttezza della query

identificare tutte le parti che compongono la query

Ogni singolo "pezzo" della query è identificato e memorizzato in una struttura interna al motore di database, solitamente nella forma di un albero (query tree).

Un albero è una rappresentazione che può essere facilmente manipolata dal sistema interno, aggiungendo, rimuovendo o spostando le sue componenti.

1.1.2 Standardizzazione

Uno dei punti di forza di un database relazionale è l'abilità di accettare query da utenti

che non hanno una elevata comprensione della sottostante struttura del database, come risultato, una query può essere molto complessa, il sistema deve essere in grado di "risolvere" svariate combinazioni di istruzioni ed identificare i risultati corretti.

Il processo di "standardizzazione" è di trasformare la query in un formato più comprensibile al motore stesso. Questo si effettua mediante una serie di manipolazioni sull'albero di query costruito precedentemente. Durante questo processo, vengono rimosse tutte le clausole eventualmente ridondanti e l'intero albero viene riarrangiato. L'albero risultante viene passato all'ottimizzazione.

1.1.3 Ottimizzazione della query

Lo scopo del processo di ottimizzazione della query è produrre un "piano di esecuzione" il più efficiente possibile, basandosi su quanto è specificato dall'albero della query. Un "ottimizzatore" teoricamente può produrre un piano di esecuzione "ottimale" per ogni query, in realtà questo finirà per produrre un piano solamente accettabile per la maggioranza delle query. Questo perché il numero di combinazioni possibili in una Join aumenta geometricamente e nello stesso modo aumenta la complessità della query.

Senza l'utilizzo di tecniche di "pruning" o altri metodi euristici per limitare il numero di combinazioni valutate, il tempo richiesto per ottenere una reale ottimizzazione della query risulta assolutamente inaccettabile. In molti casi l'ottimizzatore sceglie una query meno efficiente (o totalmente inefficiente) perché la selezione di una query più efficiente richiede più tempo che l'esecuzione della query inefficiente.

I vari database utilizzano di solito differenti tecniche di ottimizzazione per ottenere una certa efficienza nel piano di esecuzione.

Ottimizzazione-euristica

Questo e' un meccanismo basato su regole specifiche per produrre un piano di esecuzione efficiente. Dato che la query ricevuta e' una struttura definita, ogni nodo dell'albero viene mappato direttamente in una espressione algebrica-relazionale. La funzione euristica e' quindi applicata per ridurre l'espressione ai suoi termini di base, ottenendo quindi una rappresentazione più efficiente.

Utilizzando un'espressione algebrica si assicura anche che nessuna delle necessarie informazioni richieste per esaminare i dati verrà persa durante il processo.

Ottimizzazione-sintattica

Questo tipo di ottimizzazione si appoggia pesantemente sulla comprensione dell'utente sia del database sottostante, sia della distribuzione dei dati tra le varie tabelle. Si fa' affidamento sul fatto che l'utente ha già fatto delle scelte in base alle proprie conoscenze. L'ottimizzatone cerca di migliorare l'efficienza della query scegliendo gli indici appositi tra quelli disponibili per ogni singola tabella.

Questo tipo di ottimizzazione è estremamente efficiente quando si accede a dati in un'ambiente sostanzialmente statico e quando l'utente sa quello che sta facendo. Ovviamente, se il database ed i suoi contenuti cambiano in maniera molto varia o se l'utente non sa esattamente cosa richiedere (la query non è già ottimizzata di suo), i risultati possono essere pessimi.

Ottimizzazione-"Cost-Based"

Per eseguire questo tipo di ottimizzazione, l'ottimizzatore richiede informazioni specifiche relative alle informazioni del database stesso. Queste informazioni sono strettamente dipendenti dal sistema e possono includere cose come la dimensione dei files, la struttura degli stessi, la disponibilità di indici, la percentuale di record da recuperare da ogni tabella etc.

Dato che lo scopo di ogni ottimizzazione e' quello di ridurre al minimo il numero di record estratti ed il tempo di estrazione, l'ottimizzazione cost-based utilizza le informazioni sulla struttura del database e la distribuzione dei dati per assegnare un "costo" stimato, in termini di tempo e numero di record da estrarre da ogni tabella, numero di accessi etc. per ogni operazione.

Valutando la somma totale di questi "costi" è possibile selezionare la sequenza piu' efficiente di estrazione dei dati.

Ovviamente, i "costi" assegnati saranno più o meno validi a seconda delle informazioni che il sistema ha/mantiene sulla composizione delle tabelle, dei files etc. Mantenere aggiornate queste informazioni occupa tempo e risorse, quindi ogni sistema memorizza un blocco di informazioni e poi lo aggiorna (ricostruendolo) di tanto in tanto. Se sul database vengono effettuate molte operazioni che coinvolgono la distruzione totale e la ricostruzione da zero di svariate tabelle, l'efficienza di questo metodo di ottimizzazione e' seriamente compromessa.

Ottimizzazione semantica

Questo tipo di ottimizzazione non e' ancora entrata nel novero delle ottimizzazioni "standard", ma e' oggetto di ricerche. Questo metodo si basa sulla conoscenza della struttura del sottostante database per ignorare o eliminare parti della query che non ritornerebbero risultato o non ritornerebbero risultati utili.

1.2 Selezione degli indici

Per ottimizzare una query, la maggior parte degli ottimizzatori verifica se nel database sono presenti degli indici utili per migliorare l'efficienza di accesso ai dati. Un indice è considerato "utile" solo se inizia con le stesse colonne (campi) che sono contenute nella

query. Questa deve essere una corrispondenza esatta.

1.3 Selezione delle Join

Quando gli indici sono stati scelti e tutte le clausole sono state associate ad un "costo di processo", l'ottimizzatore esegue la selezione delle Join. Questo è un tentativo di scegliere il migliore ordine per combinare le varie clausole.

L'ottimizzatore confronta vari ordinamenti delle clausole, quindi seleziona quello con il minor tempo di processo stimato.

La maggior parte dei database, utilizza una ottimizzazione cost-based. Perché? Perché è molto più semplice da implementare piuttosto che una Euristica.

1.4 Come ottimizzare la query?

1.4.1 Indici

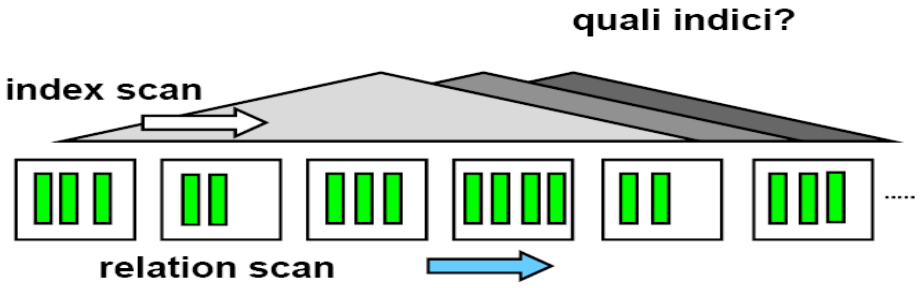

Fig. 1.1 Indici

Un indice può essere utilizzato per eseguire una interrogazione SQL se l'attributo su cui

è costruito:

- compare nella clausola WHERE
- è contenuto in un FATTORE BOOLEANO
- il fattore booleano è ARGOMENTO DI RICERCA attraverso indice
- compare in un ORDER BY o GROUP BY

Un indice è utile per una query solo se il costo di accesso con l'indice è minore del costo dell'accesso sequenziale cioè minore del numero di pagine per file (numero di pagine per file / 2 se attributo unique).

Selettività

Un predicato è selettivo ci si aspetta che non tutte le tuple lo soddisfino. Si può capire se un predicato è selettivo utilizzando il fattore di selettività F (detto anche fattore di filtro).

La selettività (F) è uguale a 1 fratto il numero di valori distinti della chiave (NK)

$$\text{se A = valore} \quad F = \frac{1}{NK_A}$$

con valore di default 1/10.

$$\text{se A invece ha più valori ad esempio (v1, v2, v3, v4....),} \quad F = \frac{4}{NK_A}$$

la regola generale dice: $F = \frac{numero\,valori}{NK}$

Il fattore di selettività si può sfruttare con A < o > di un certo valore, la cui cardinalità non è controllabile:

esempio maggiore:

$$F = \frac{max_A - val_A}{max_A - min_A}$$

con valore di default 1/3

Analogamente per BETWEEN con valori come esempio 100 e 60

$$F_A = \frac{100 - 60}{max_A - min_A}$$

valore di default ¼

Predicati su attributi diversi (Pred1 OR Pred2):

$$F = \frac{F_{Pred1} + F_{Pred2}}{F_{Pred1} * F_{Pred2}}$$

Attributo A = attributo B:

$$F = \frac{1}{max(NK_A, NK_B)}$$

se i due domini sono sovrapposti. F=0 altrimenti

Negazione: A = not valore

$$F = \frac{1 - 1}{NK_A}$$

Numero di tuple del risultato dove NT è il numero di tuple del file:

$$E = \frac{NT}{F_{Pred1}}$$

e nell'ipotesi di assenza di correlazione tra i valori degli attributi, per più predicati:

$$E = \frac{NT}{\prod i_{Pred1}}$$

- l'ipotesi non è sempre verificata, bisognerebbe rilevare un fattore di correlazione o di clustering relativo tra valori di attributi differenti:
esempi:
tipo di lavoro, data di nascita: *incorrelati*
qualifica, dipartimento: *correlati*

Costo di accesso

Nella mia tesi i casi esaminati si differenziano a seconda che:

indice clustered e indice unclustered (non-clustered) ordinato e disordinato

1.1.4.1 Indici clustered

La creazione di un indice cluster comporta il riordinamento dell'intera tabella. In effetti non esiste fisicamente un indice in quant'è la stessa tabella che viene ordinata in base al campo presente nell'indice, è evidente che questo è il tipo di indice più performante dal punto di vista delle richieste ma il più pesante durante l'aggiornamento e l'inserimento, non a caso il più delle volte coincide con il campo identificatore del record che è immodificabile e viene incrementato ad ogni inserimento di un record. Quindi la scelta di quale campo inserire in questo tipo di indice è strategica, anche perché è evidente il motivo per il quale non possono esserci due indici cluster per una tabella perché i dati

fisicamente o sono ordinati rispetto ad un campo o sono ordinati rispetto ad un altro.

1.1.4.2 indice non-clustered

In un indice non-clustered i nodi dello stesso contengono i riferimenti alle righe della tabella valorizzate con una determinata *n-pla* di valori in corrispondenza dei campi indicizzati; per poter effettuare una selezione il database accede all'indice, di solito di tipo binario, con i valori desiderati per le righe da selezionare, e utilizza i riferimenti memorizzati nello stesso per leggere da disco le righe corrispondenti al criterio di ricerca utilizzato.

Per un predicato di uguaglianza il costo di accesso tramite indice clustered o unclustered rimane invariato.

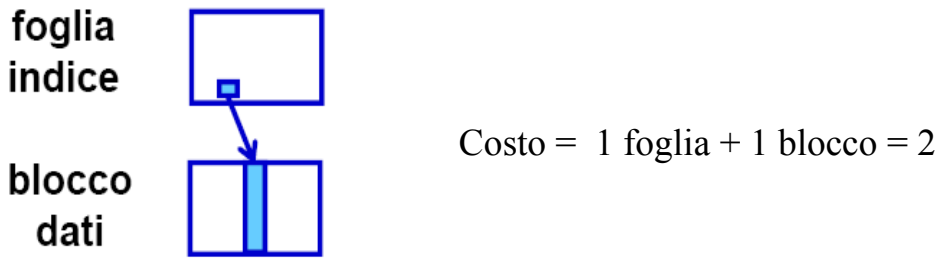

Fig. 1.2 Costo di accesso 1

Ad esempio: facoltà = M.F.N.

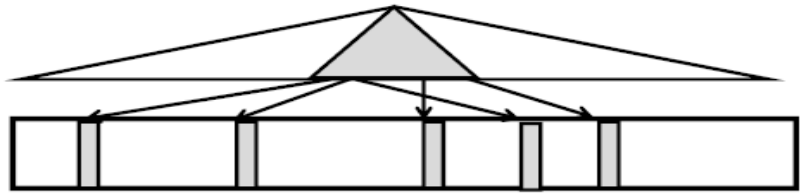

Fig. 1.3 Costo di accesso 2

Il costo per l'indice unclustered si ricava tramite la formula:

$$Costo = ([F * NF] + [F * NT])$$

analogamente per l'indice clustered:

$$Costo = ([F * NF] + [F * NB])$$

la formula del costo può essere migliorata nel caso di ordinamento dei TIDs. Riferimento figura 1.4.

– trovare le etichette che vanno bene

– ordinare i tid dei record di dati da recuperare

- recuperare i record in ordine. Questo assicura che ogni pagina di dati sia letta una ed una sola volta

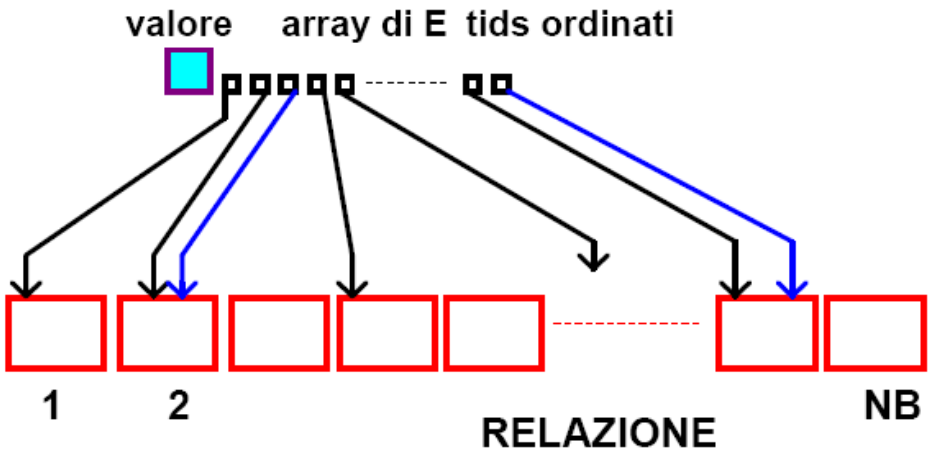

Fig. 1.4 Costo di accesso con TIDs ordinati

In generale si ha $Costo\,della\,relazione \leqslant min(E, NB)$

E – numerdo delle tuple e

NB – numero di pagine del file

Il costo della relazione può essere calcolato in modo più efficace con l'aiuto della formula di Cardenas:

$$Costo\,relazione = \Phi(E, NB) = [NB * (1 - (1 - 1/NB)^{E})]$$

La formula e' valida sotto le seguenti ipotesi:

a) tutti i record sono equiprobabili,

b) tutti i blocchi contengono lo stesso numero di tuple,

c) come conseguenza di a) e b) tutti i blocchi sono equiprobabili.

$NB * (1 - (1 - 1/NB)^{E})$ è il numero di blocchi che ci si aspetta contengano almeno una tupla

1.4.2 Costo di join

Qui sotto propongo un esercizio lo scopo del quale è ottimizzare l'esecuzione della seguente interrogazione:

SELECT *pj, nome*

FROM *prodotti, progetti*
WHERE *dno = 136*
AND *tipo = 'AA'*
AND *prodotti.pj = progetti.pj*

individuando la sequenza di accesso e gli indici più convenienti sulle seguenti relazioni e indici:

NT - numero di tuple del file

NB - numero di pagine del file

NK - numero di valori distinti della chiave

NF - numero di foglie dell'indice

prodotti (pn, pnome, tipo, pj),

NT = 2000, **NB** = 400

tipo con **NK** = 100 , **NF** = 10, indice unclustered TID disordinati

prodotti.pj con **NK** = 200 , **NF** = 15, indice unclustered TID disordinati

progetti (pj, nome, dno),
NT = 200, **NB**= 100

dno con **NK** = 20 , **NF** = 5, indice unclustered TID disordinati

progetti.pj con **NK** = 200 , **NF** = 10, indice unclustered TID disordinati

Calcolo il costo del join:

A) Sequenza prodotti => progetti

1) accesso a prodotti:

- costo di accesso sequenziale:

$C_{seq} = 400,$

-calcolo fattore di filtro

$F_{tipo} = 1 / 100,$

$E_{tipo} = 2000/100 = 20$

$C_{tipo} = [10 / 100] + [2000 / 100] = 21$

l'indice su pj non si può usare su prodotti

2) accesso a progetti:

- costo di accesso sequenziale:

$C_{seq} = 100,$

-calcolo fattore di filtro

$F_{dno} = 1 / 20,$

$E_{dno} = 200 / 20 = 10$

$C_{dno} = [5 / 20] + 10 = 11$ (indice su dno)

$C_{progetti.pj} = 1 + 1 = 2$ (indice su pj)

Con l'accesso a prodotti si ottengono $E_{tipo} = 20$ tuple che soddisfano tipo = 'AA' quindi per 20 volte si fa l'accesso a progetti per trovare le tuple che soddisfino dno = 136 ed il predicato di join prodotti.pj = progetti.pj

In conclusione:

$C_{join} = C_{prodotti} + E \times C_{progetti}$

$C_{join} = C_{tipo} + E \times C_{pj} = 21 + 20 \times 2 = 61$

$C_{join} = C_{seq\text{-}prodotti} + E \times C_{seq\text{-}progetti} = 400 + 20 \times 100 = 2400$

B) Sequenza progetti => prodotti

1) accesso a progetti:

$E_{dno} = 200 / 20 = 10$

$C_{dno} = 11$ (già calcolato)

l'indice su pj non si può usare su progetti

2) accesso a prodotti:

C_{tipo}= 21 (già calcolato)

$C_{prodotti.pj}$ = [15 / 200] + 10 = 11 (indice su pj)

$C_{join} = C_{dno} + E_{dno} \times C_{progetti.pj} = 11 + 10 \times 11 = 121$

Il mio programma si basa su un sito web, realizzato mediante una serie di script in linguaggi come HTML, CSS e PHP attraverso i quali valutare quale sia la migliore strategia di accesso per interrogazioni SQL nel caso di join. Lo scopo del software è di eseguire dei calcoli e creare un output simile a quello nell'esempio appena proposto. I criteri di valutazione servono a prendere decisioni sull'ordinamento delle relazioni e quali indici costruire. Un predicato di join è utilizzabile come argomento di ricerca solo se è possibile sostituire un valore al posto di uno dei due attributi.

Gli ottimizzatori generalmente scartano a priori le sequenze che contengono prodotti cartesiani perché potrebbero dare luogo a E intermedie troppo elevate, anche se questo non sempre è vero. Essi valutano tutte le sequenze dove due relazioni consecutive sono collegate da predicati di join e scelgono la migliore.
In conclusione si può dire che senza indici il DBMS relazionale ha prestazioni scadenti. L'ordinamento e gli indici migliorano di molto le prestazioni. Un eccesso di "indicizzazione" è nocivo per DB con elevato carico di modifiche, per questo bisogna trovare un compromesso sensato.

Capitolo 2

Tecnologia PHP

2.1 Introduzione e cenni storici

Nasce nel 1994, ad opera di Rasmus Lerdorf, come una serie di macro la cui funzione era quella di facilitare ai programmatori l'amministrazione delle homepage personali: da qui trae origine il suo nome, che allora significava appunto Personal Home Page. In

seguito, queste macro furono riscritte ed ampliate fino a comprendere un pacchetto chiamato Form Interpreter (PHP/FI).

Essendo un progetto di tipo open source , ben presto si formò una ricca comunità di sviluppatori che portò alla creazione di PHP 3: la versione del linguaggio che diede il via alla crescita esponenziale della sua popolarità. Tale popolarità era dovuta anche alla forte integrazione di PHP con il Web server Apache, e con il database MySQL. Tale combinazione di prodotti, integralmente ispirata alla filosofia del free software, diventò ben presto vincente in un mondo in continua evoluzione come quello di Internet.

Alla fine del 1998 erano circa 250.000 i server Web che supportavano PHP: un anno dopo superavano il milione. I 2 milioni furono toccati in aprile del 2000, e alla fine dello stesso anno erano addirittura 4.800.000. Il 2000 è stato sicuramente l'anno di maggiore crescita del PHP, coincisa anche con il rilascio della versione 4, con un nuovo motore (Zend) molto più veloce del precedente ed una lunga serie di nuove funzioni, fra cui quelle importantissime per la gestione delle sessioni. La crescita di PHP, nonostante sia rimasta bloccata fra luglio e ottobre del 2001, è poi proseguita toccando quota 7.300.000 server alla fine del 2001, per superare i 10 milioni alla fine del 2002, quando è stata rilasciata la versione 4.3.0. La continua evoluzione dei linguaggi di programmazione concorrenti e l'incremento notevole dell'utilizzo del linguaggio anche in applicazioni enterprise ha portato la Zend a sviluppare una nuova versione del motore per supportare una struttura ad oggetti molto più rigida e potente.

Nasce così PHP 5, che si propone come innovazione nell'ambito dello sviluppo web open source soprattutto grazie agli strumenti di supporto professionali forniti con la distribuzione standard.

Oggi PHP è conosciuto come PHP: Hypertext Preprocessor, ed è un linguaggio completo di scripting, sofisticato e flessibile, che può girare praticamente su qualsiasi server Web, su qualsiasi sistema operativo (Windows o Unix/Linux, ma anche Mac, AS/400, Novell, OS/2 e altri), e consente di interagire praticamente con qualsiasi tipo di database (SQLite, MySQL, PostgreSQL, SQL Server, Oracle, SyBase, Access e altri). Si

può utilizzare per i più svariati tipi di progetti, dalla semplice home page dinamica fino al grande portale o al sito di e-commerce.

2.2 Linguaggi di scripting

In informatica un linguaggio di scripting è un linguaggio di programmazione interpretato (cioè che non viene compilato) destinato in genere a compiti di automazione del sistema (batch) o delle applicazioni (macro), o ad essere usato all'interno delle pagine web.

Esempi di linguaggi di scripting sono JavaScript, Perl, Python, Ruby ecc.

Nei linguaggi di scripting il programmatore generalmente si disinteressa delle *risorse di sistema* che il programma finito dovrà consumare, demandando il tutto al sistema stesso. Per risorse si intendono, per esempio, la gestione della allocazione e deallocazione della memoria, la conversione tra tipi, l'inizializzazione e la chiusura dell'applicazione.

In questo modo si evitano molti problemi tipici della programmazione tradizionale, che risulta essere soggetta ad errori non facilmente individuabili e pericolosi, e inoltre costringe il programmatore ad occuparsi di problematiche non strettamente connesse con l'obiettivo del software che deve creare. L'utilizzo di un linguaggio di scripting permette di concentrarsi direttamente sulla soluzione del problema.

2.3 Struttura sintattica di PHP

PHP necessita di una coppia di tag per l'apertura e la chiusura del codice contenuto in un file richiesto da un Web Server. Si tratta dei tag

<?php

.....

?>

Un'altra possibilità è quella di usare i tag brevi, che devono essere abilitati manualmente modificando le impostazioni del file di configurazione php.ini:

<?

....

?>

I tag delimitano il codice PHP, ed il codice contenuto al loro interno non sarà inviato al browser, ma compilato e eseguito. Da questo potremmo dedurre che tutto ciò che sta fuori da questi tag non verrà toccato da PHP, che si limiterà a passarlo al browser così com'è, eventualmente ripetendolo in base a situazioni particolari

2.4 PHP e HTML

Ho deciso di utilizzare PHP per lo sviluppo della mia tesi perchè è un linguaggio la cui funzione fondamentale è quella di produrre codice HTML, che è quello dal quale sono

formate le pagine web. Ma, poichè PHP è un linguaggio di programmazione, ho la possibilità di analizzare diverse situazioni (l'input degli utenti, i dati contenuti in un database) e di decidere, di conseguenza, di produrre codice HTML condizionato ai

risultati dell'elaborazione. Questo è, in parole povere, il Web dinamico. Quando il server riceve una richiesta per una pagina PHP, la fa analizzare dall'interprete del linguaggio, il quale restituisce un file contenente solo il codice che deve essere inviato al browser.

Come avviene la produzione di codice HTML? La prima cosa da sapere è come fa l'interprete PHP a discernere quale porzione di un file contiene codice da elaborare e quale codice da restituire solamente all'utente. Questa fase di riconoscimento è molto importante, dato che permette a PHP di essere incluso all'interno di normale codice HTML in modo da renderne dinamica la creazione. Il codice PHP deve essere compreso fra appositi tag di apertura e di chiusura, che sono i seguenti:

```
<?php (tag di apertura)
        echo “Query optimizer”;
?> (tag di chiusura)
```

Tutto ciò che è contenuto fra questi tag deve corrispondere alle regole sintattiche del PHP, ed è codice che sarà eseguito dall'interprete e non sarà inviato direttamente al browser. Per generare l'output da inviare al browser attraverso codice PHP viene normalmente utilizzato il costrutto `echo`. Nel caso di sopra l'echo visualizza “Query otpimizer”

Esempio:

```
<html>
  <head>
    <title>
      <?php  echo "Tesi in PHP"; ?>
    </title>
  </head>
  <body>
    <?php echo "Buongiorno questo è il Query optimizer"; ?>
    </body>
</html>
```

Questo codice produrrà un file HTML e l'utente vedrà sul suo browser la riga "Buongiorno questo è il Query optimizer". Il dato da inviare al browser che segue il comando `echo` può essere racchiuso tra parentesi e che al comando possono essere date in input più stringhe.

2.5 Variabile POST

Nel mio programma l'utilizzo di queste variabili da la possibilità di variare i contenuti delle pagine in base alle richieste degli utenti. Questa possibilità si materializza attraverso i meccanismi che permettono agli utenti, oltre che di richiedere una pagina ad un web server, anche di specificare determinati parametri che saranno utilizzati dallo script PHP per determinare quali contenuti la pagina dovrà mostrare. Come esempio, possiamo immaginare la pagina il cui scopo è quello di visualizzare la query inserita. Nel momento in cui si richiama la pagina, si dovrà specificare ad esempio NB -numero di pagine del file, per consentire allo script di elaborare i dati attraverso le formule e

mostrar i risultati all'utente.

In alcuni casi, i dati che devono essere trasmessi allo script sono piuttosto numerosi: pensiamo ad esempio ad un modulo di registrazione per utenti, nel quale vengono indicati nome, cognome, indirizzo, telefono, casella e-mail ed altri dati personali. In questo caso lo script, dopo averli ricevuti, andrà a salvarli nel database.

2.5.1. Il metodo POST

Il metodo POST viene utilizzato con i moduli: quando una pagina HTML contiene un tag `<form>`, uno dei suoi attributi è `method`, che può valere GET o POST. Se il metodo è GET, i dati vengono passati nella query string. Se il metodo è POST, i dati vengono invece inviati in maniera da non essere direttamente visibili per l'utente, attraverso la richiesta HTTP che il browser invia al server.

Nel mio programma questa variabile viene usata ogni volta quando voglio recuperare dei dati passati dall'utente. I dati che vengono passati attraverso il metodo POST sono memorizzati nell'array $_POST che è un array superglobale. Quindi, per fare un esempio attraverso un piccolo modulo:

```
<form action="tesi.php" method="post">
  <input type="text" name="nome">
   <input type="checkbox" name="nuovo" value="si">
   <input type="submit" name="submit" value="invia">
</form>
```

Questo modulo contiene semplicemente una casella di testo che si chiama 'nome' e una

checkbox che si chiama 'nuovo', il cui valore è definito come 'sì'. Poi c'è il tasto che invia i dati, attraverso il metodo POST, alla pagina tesi.php.

Questa pagina si troverà a disposizione la variabile `$_POST['nome']`, contenente il valore che l'utente ha digitato nel campo di testo; inoltre, se è stata selezionata la checkbox, riceverà la variabile `$_POST['nuovo']` con valore 'si'. Attenzione però: se la checkbox non viene selezionata dall'utente, la variabile corrispondente risulterà non definita.

2.6 Funzioni per la gestione delle stringhe

Esamino alcune funzioni che operano sulle stringhe e che ho usato per lo sviluppo del progetto. Funzioni come **explode, split** ecc, mi permettono di effettuare dei controlli per il corretto funzionamento del ptogramma. (l'eventuale indicazione di un parametro tra parentesi quadre indica che quel parametro è facoltativo, quindi può non essere indicato nel momento in cui si chiama la funzione):

- **strlen(stringa)**: verifica la lunghezza della stringa, cioè il numero di caratteri che la compongono. Restituisce un numero intero.

- **Strstr(stringa, stringa)**: cerca la seconda stringa all'interno della prima, e restituisce la prima stringa a partire dal punto in cui ha trovato la seconda. `strstr('programma', 'gr')` restituisce 'gramma'. Restituisce una stringa se la ricerca va a buon fine, altrimenti il valore booleano FALSE. La funzione stristr() funziona allo stesso modo ma non tiene conto della differenza fra maiuscole e minuscole.

- **explode(stringa, stringa [, intero])**: trasforma la seconda stringa in un array, usando la prima per separare gli elementi. Il terzo parametro può servire ad indicare il numero massimo di elementi che l'array può contenere (se la suddivisione della stringa portasse ad un numero maggiore, la parte finale della stringa sarà interamente contenuta nell'ultimo elemento). Ad esempio: `explode(' ', 'Basi dati')` restituisce un array di due elementi in cui il primo è 'Basi' e il secondo 'dati'.

  Esempio utilizzo la funzione **explode():**

```
<?php
$pizza  = "prezzo1 prezzo2 prezzo3";
$prezzi= explode(" ", $pizza);
echo $prezzi[0]; // prezzo1
echo $prezzi[1]; // prezzo2
?>
```

- **ucfirst(stringa)**: trasforma in maiuscolo il primo carattere della stringa. Restituisce la stringa modificata.

- **ucwords(stringa)**: trasforma in maiuscolo il primo carattere di ogni parola della stringa, intendendo come parola una serie di caratteri che segue uno spazio. Restituisce la stringa modificata.

- **split():** serve per suddividere una stringa (il secondo argomento della funzione) in più parti in base al carattere passato come primo argomento; svolge quindi lo stesso lavoro della funzione explode() ma, a differenza di questa, accetta un'Espressione Regolare come primo argomento.

  È possibile quindi "splittare" la stringa non sulla base di un singolo carattere, ma di un'espressione. Importante rilevare come split() restituisca un array contenente tanti elementi quanti sono le parti della stringa risultanti dalla

divisione; la funzione accetta anche un terzo parametro facoltativo (un numero intero) che, se passato, limita il numero di elementi dell'array.

Esempio utilizzo la funzione **split():**

```
<?php
$date = "04/30/1973";
list($mese, $giorno, $anno) = split('[/.-]', $data);
echo "Mese: $mese;Giorno: $giorno; Anno: $anno<br />\n";
?>
```

- **strtoupper(): r**estituisce la stringa *string* con i caratteri alfabetici convertiti in maiuscolo.

  Nota: i caratteri 'alfabetici' sono determinati in base alle impostazioni locali. Ciò significa, ad esempio, che nelle impostazioni locali di default del "C", il c arattere umlaut-A (à„) non sarà convertito.

  Esempio utilizzo la funzione **strtoupper():**

```
<?php
$str = "tesi ottimizzatore di query";
$str = strtoupper($str);
echo $str; // Stampa TESI OTTIMIZZATORE DI QUERY
?>
```

Parte II

Progetto e realizzazione di un ottimizzatore di query

Capitolo 3

Progetto

La progettazione del software è passata attraverso una serie di fasi. Per prima cosa sono stati raccolti i requisiti da sviluppare.
Prima di procedere con la scrittura del codice PHP sono state valutate una serie di modi per ottimizzare le performance di accesso ai dati.

3.1 Requisiti funzionali

Di seguito sono elencati i requisiti principali che hanno portato alla realizzazione del software.

RF01

Autore: inserimento dati pagina iniziale.

Introduzione: ogni autore ha la possibilità di inserire il numero di righe per le clausole And , Or, Between

Input: i dati inseriti nel form.

Processing: il numero di righe specificato dall'autore sarranno aggiunte nel form successivo

Output: operazione effettuata e redirezione alla pagina successiva.

RF02

Autore: inserimento dati pagina inserimento query.

Introduzione: ogni autore ha la possibilità di inserire una query nel form inserimento query.

Input: i dati inseriti nel form.

Processing: i dati inseriti dell'autore saranno controllati; in caso di inserimento errato non sarà possibile proseguire avanti.

Output: operazione effettuata e redirezione alla pagina successiva.

RF03

Autore: eliminazione dati inseriti

Introduzione: inserita la query, l'autore ha la possibilità di cancellare i dettagli inseriti.

Input: conferma dell'eliminazione dei dati nel form.

Processing: i dati inseriti nella form saranno cancellati.

Output: operazione effettuata.

RF04

Autore: inserimento dati per il calcolo del costo di accesso.

Introduzione: una volta inserita la query nella fase uno, sarà possibile effettuare l'inserimento dei dati necessari per il calcolo del costo di accesso. Per ogni relazione l'autore può specificare che tipo di indice utilizzare (clustered, unclustered ordinati o unclustered disordinati).

Input: i dati inseriti nel form.

Processing: i dati inseriti saranno controllati; in caso di inserimento errato non sarà possibile effettuare il calcolo del costo di accesso.

Output: operazione effettuata e redirezione alla pagina successiva.

RF05

Autore: elaborazione dati

Introduzione: una volta inseriti i datti correttamente saranno processati dal programma.

Input: i dati inseriti.

Processing: elaborazione dei dati inseriti

Output: operazione effettuata.

RF06

Autore: visualizzazione risultati.

Introduzione: vengono stampati i risultati con i costi di accesso, più la sequenza di accesso più conveniente.

Input: i dati inserite nel form.

Processing: i dati elaborati saranno visualizzati.

Output: la stampa dei risultati.

3.2 Scenario: ottimizzatore di query

Aprendo il programma, l'autore ha la possibilità di specificare quante righe vorrebbe per le clausole And, Or e Between. Dopo aver scelto il numero di righe si passa alla pagina successiva dove l'utente può inserire la query nel apposito form. In caso di inserimento scorretto l'autore può annullare l'inserzione o in caso contrario proseguire inviando i dati al form successivo.

Al passo successivo, l'autore specifica per ogni relazione: il numero di tuple del file, il numedo di pagine del file, il numero dei valori distinti della chiava, il numero di foglie dell'indice, il tipo del indice (clustered, unclustered ordinato oppure unclustered disordinato).

Una volta inseriti i dati nel form il sistema, attraverso una serie di script realizzati in PHP, controlla la correttezza dei vari campi, dopo di che elabora i dati inseriti.

Al termine di questa operazione, vengono visualizzati i risultati.

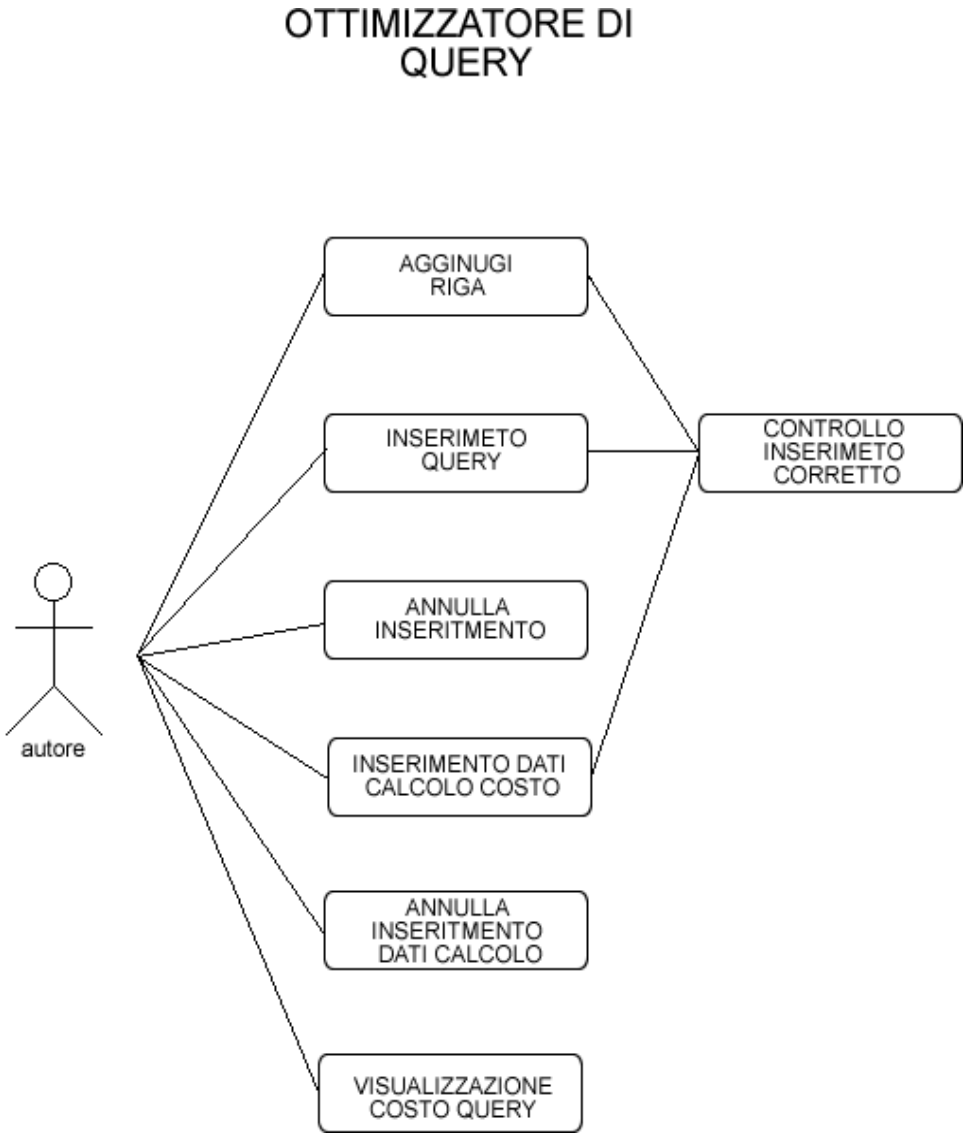

Fig. 3.1 Scenario: Ottimizzatore di query

3.3 Casi d'uso

Caso d'uso: Inserimento query

UC: UC01
Nome: inserimento query
Attori: Autore
Precondizioni: ➢ L'autore ha specificato le dimensioni del form e ha inserito la query.
Sequenza eventi: ➢ L'autore specifica le dimensioni del form. ➢ Clicca su 'Aggiungi righe.' ➢ L'autore inserisce la query. ➢ Clicca su 'Inserisce il campo SELECT'. ➢ Clicca su 'Inserisce il campo FROM'. ➢ Clicca su 'Inserisce il campo WHERE'. ➢ Clicca su 'Selezione tra AND, OR e BETWEEN e inserisce il campo a destra'. ➢ Ripete l'operazione precedente per i campi successivi. ➢ Clicca su 'Calcola'. ➢ Il sistema controlla la sintassi della query inserita : 1. Se la sintassi non è corretta il sistema da messaggio di errore e chiede l'inserimento di una query corretta. 2. In caso contrario si passa alla pagina successiva. ➢ I dati inseriti sul form possono essere cancellati cliccando sul pulsante 'ANNULLA'
Estensioni: ➢ Segnalazioni di mancata compilazione del form. ➢ Segnalazioni di inserimenti errati nel form.

Postcondizioni: ➢ Al termine della procedura sarà possibile inserire i dati sul form per il calcolo dei costi di accesso.

Caso d'uso: Inserimento dati per il calcolo dei costi

UC: UC02
Nome: Calcola costo di accesso
Attori: Autore
Precondizioni: ➢ L'autore ha inserito la query nel form.
Sequenza eventi: ➢ L'autore compila il form per il costo della query ➢ Inserisce valori nel campo 'NT' per ogni tabella. ➢ Inserisce valori nel campo 'NB' per ogni tabella. ➢ Seleziona un attributo. ➢ Per ogni attributo selezionato specifica 'NK' ➢ Per ogni attributo selezionato specifica 'NF' ➢ L'autore specifica un indice tra: clustered, unclustered ordinato o unclustered disordinato. Un indice clustered non può essere specificato più di una volta per relazione. ➢ Clicca su 'Calcola costo'. ➢ Il sistema elabora i dati inseriti. ➢ Il sistema controlla se i dati sono inseriti correttamente: 1. Se i campi obbligatori non sono inseriti il sistema da un messaggio di errore. 2. In caso contrario viene visualizzata la pagina con i costi di accesso. ➢ I dati inseriti sul form possono essere cancellati cliccando sul pulsante 'ANNULLA'

Estensioni: ➢ Segnalazioni di mancata compilazione del form. ➢ Segnalazioni di inserimenti errati nel form.
Postcondizioni: ➢ Al termine della procedura saranno visualizzati i costi di accesso.

3.4 Activity Diagram

1) Activity Diagram #1: Inserimento query
2) Activity Diagram #2: Calcolo costo di accesso

Activity Diagram #1 – figura 3.4.1

Per prima cosa l'utente sceglie le dimensioni del form, dopo di che inserisce una query nel form appena specificato come segue: select, from, where ecc. L'autore conferma la query e solo dopo che tutti i dati sono stati inseriti in modo corretto è possibile passare al livello successivo.
E' presente inoltre la possibilità per l'annullamento dei dati inseriti in caso l'utente trovi qualche errore di compilazione.

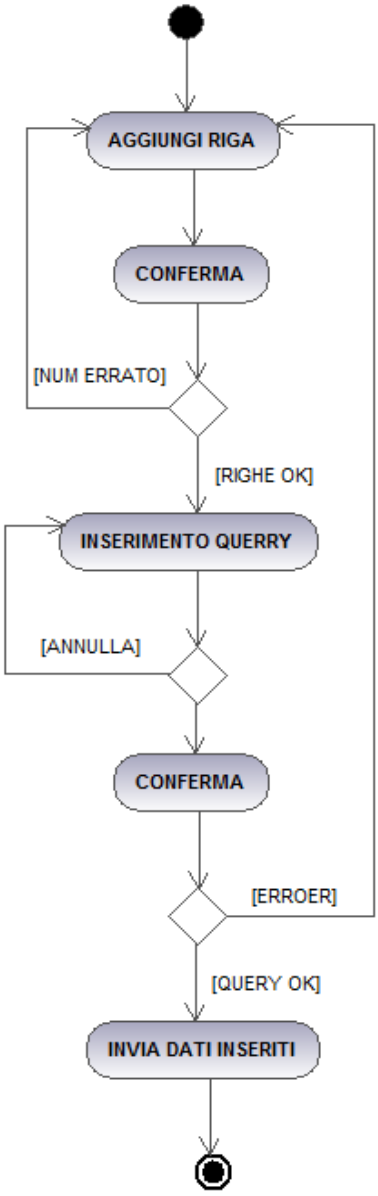

Fig. 3.2 Activity Dagram: inserimento query

Activity Diagram #2 Calcolo costo di accesso

Activity Diagram #2– figura 3.4.2

Si effettua l'inserimento di tutti i campi partendo dal valore di NT (numero di tuple del file) per la prima relazione, dopo di che si specifica il valore di NB (numero di pagine del file) della medesima. Per ogni relazione è possibile specificare degli attributi. Per essi è richiesta la compilazione dei campi NK (numero di valori distinti della chiave) e NF (numero di foglie dell'indice), inoltre si sceglie un indice tra clustered, unclustered ordinato oppure unclustered disordinato. Un indice clustered può essere specificato al più una volta per relazione.
Al termine delle compilazione per l sistema sarà possibile calcolare il costo della query inserita e visualizzare i risultati nella pagina successiva.

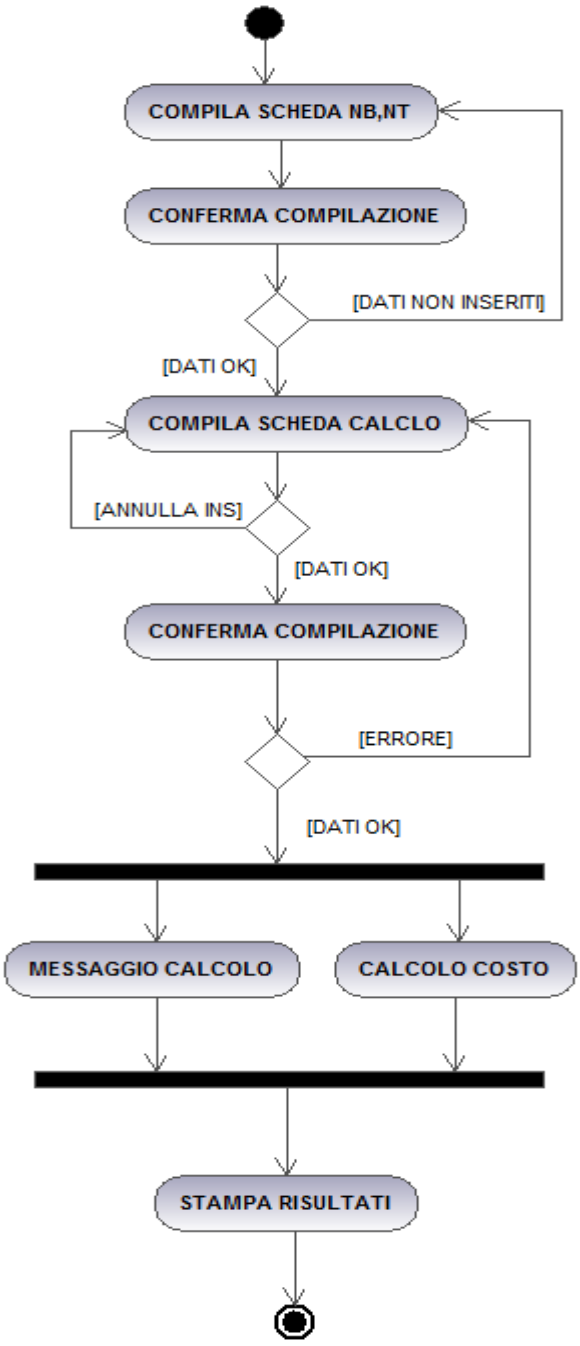
COMPILA SCHEDA NB,NT
CONFERMA COMPILAZIONE
[DATI NON INSERITI]
[DATI OK]
COMPILA SCHEDA CALCLO
[ANNULLA INS]
[DATI OK]
CONFERMA COMPILAZIONE
[ERRORE]
[DATI OK]
MESSAGGIO CALCOLO
CALCOLO COSTO
STAMPA RISULTATI

Fig. 3.3

Capitolo 4

Implementazione

In questo capitolo viene analizzato il progetto dal punto di vista implementativo.
Vengono descritti nel dettaglio gli script PHP lato server che interessano la maggior parte del software. Inoltre viene analizzato il codice HTML e CSS.
Il progetto è suddiviso in vari file come segue:

4.1 Script PHP e HTML

- AggiungiRiga.php: richiede la specifica delle righe del form.
- InserimentoQuery.php: richiede l'inserimento di una query dall'utente. Controlla il corretto inserimento e in caso di sintassi sbagliata redireziona l'autore alla

pagina AggiungiRiga.php.

- InserimentoFrom.php: righiede la specifica dei campi NT (numero di tuple del file) e NB (numero di pagine del file) per ogni tabella. Controlla se i dati richiesti sono specificati e in caso contrario da messaggio di errore.
- InserimentoCalcolo.php: permette di selezionare per ogni attributo il valore di NK (numero di valori distinti della chiava), NF (numero di foglie dell'indice) ed il tipo di indice utilizzato: clustered, unclustered ordinato o unclustered disordinato. Controlla se i dati sono inseriti, controlla se l'indice clustered è utilizzato al più una volta per relazione.
- StampaRisultati.php: visualizza i risultati del calcolo.

4.1.1 AggiungiRiga

Lo script A*ggiungiRiga* realizza un form contenente il campo *Aggiungi riga*

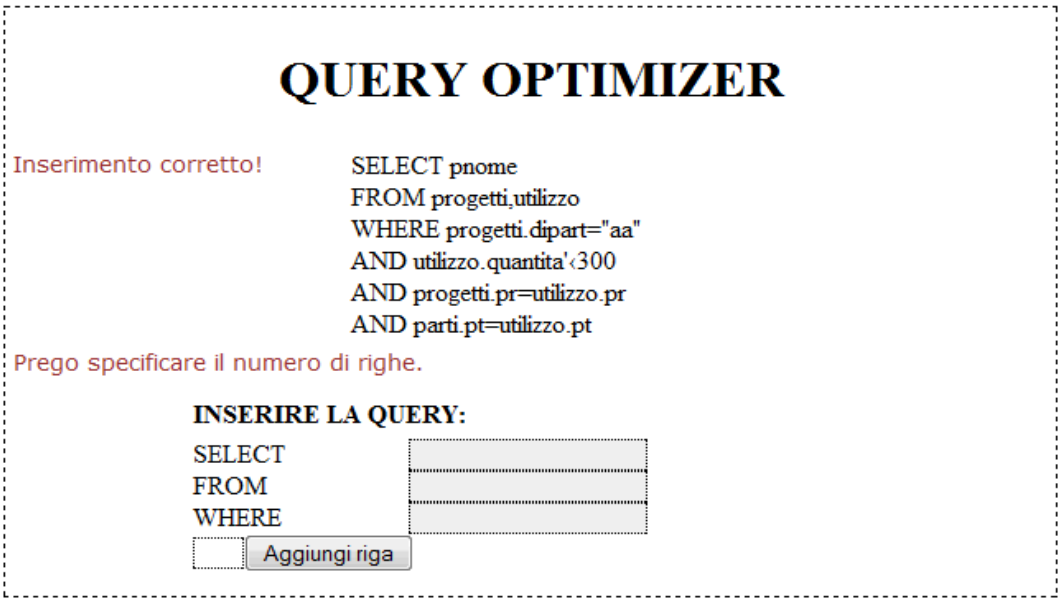

Fig. 4.1 Schermata di aggiungi riga

```
<form action="InserimentoQuery.php"  method="post">
...
<tr>
    <td colspan="2" align="left">
        <input type="text" name="add" size="1" class="text">
        <input type="submit" value="Aggiungi riga">
    </td>
</tr>
...
</form>
```

Al momento di aggiungere le righe viene controllato se si inserisce cifra tra uno e tre. In caso affermativo, con la pressione del pulsante "Aggiungi righa" il processing dei dati inseriti sarà passato, mediante il metodo post, allo script InserimentoQuery e compariranno nel form tra una , due o tre righe. Se invece viene inserito un numero minore di uno oppure maggiore di tre, il programma segnala anomalia, dopo di che bisogna tornare indietro e specificare un numero corretto.

Fig. 4.2 Schermata di aggiungi riga con numero righe > 1 e < 3

```
<?php
$add=$_POST['add'];
if($add<1 || $add>3)
{
echo "<td align='left' width='125' class='testo' colspan='2' id='ind'>Prego tontare
<a href='ok.php'>indietro</a>! Inserire numero di righe tra 1 e 3!</td>";
}
....
?>
```

4.1.2 InserimentoQuery

Con questo script viene realizzato il form per l'inserimento della query. Esso richiede la compilazione dei campi: *Select, From, Where, And, Or e Between.* Inoltre viene mostrato un esempio per il corretto inserimento di una query.

```
<form action="InserimentoFrom.php" name="queryOut" method="post">
<table border="0" cellpadding="0" cellspacing="5" width="420">
...
<td width="100" align="left">SELECT</td>
<td align="left" width="125"><input type="text" name="select" size="20"
class="text"></td>
</tr>
<tr>
<td width="100" align="left">FROM</td>
<td align="left" width="125"><input type="text" name="from" size="20"
class="text"></td>
</tr>
<tr>
<td width="100" align="left">WHERE</td>
<td align="left" width="125"><input type="text" name="where" size="20"
class="text"></td>

<?php
if((isset($add)) && ($add=="1" || $add=="2" || $add=="3") )
for($i=0;$i<$add;$i++)
```

```
echo "<table><tr><td aligin='left'>
  <select class='sel' name=a". $i ." style='display:none;'>
  </select><select class='sel' name=a". $i .">
  <option selected>   </option><option>AND</option>
  <option>OR</option>
  <option>BETWEEN</option>
  </select>
  </td><td aligin='left'><input class='text' type='hidden' name=addriga". $i ." >
  <input class='text' type='text' name=addriga". $i ." >
  </td></tr>
  </table>";
echo "<tr><td  align='left'>
<input type='reset' value='Annulla' name='annulla'>
<input type='submit' value='Calcola'>
</td></tr>";
?>
```

Lo script effettua un controllo se i numeri inseriti sono compresi tra uno e tre. Il ciclo for crea un menu select e un input type text per ogni riga.

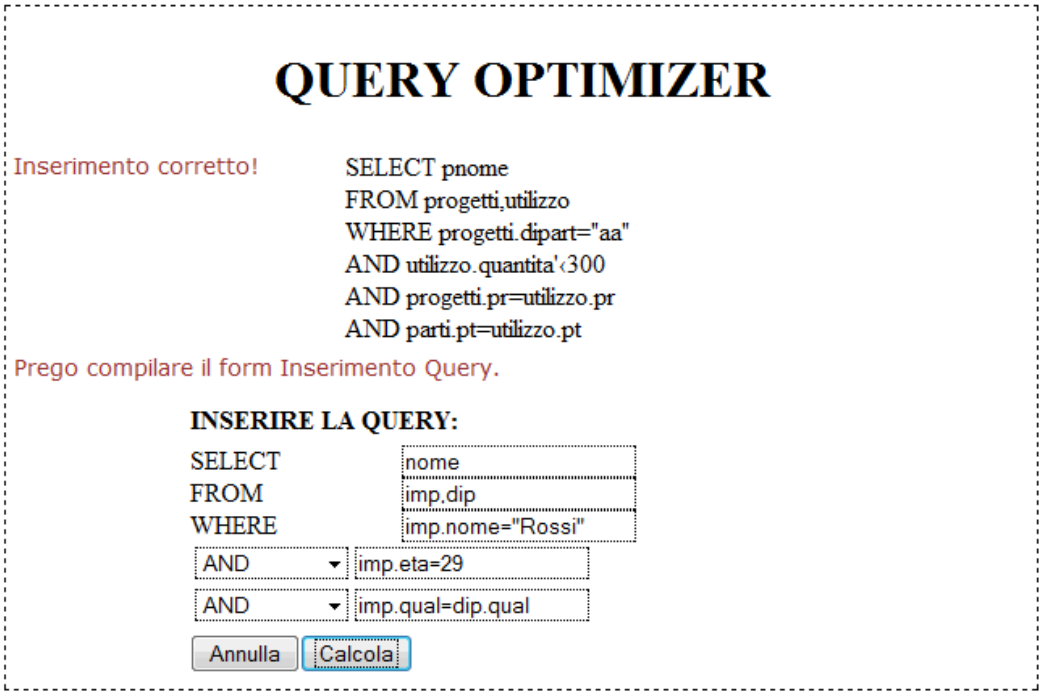

Fig. 4.3 Schermata di inserimento query

4.1.3 InserimentoFrom

Mediante il metodo *get_defined_vars();* e possibile richamare tutte le variabili di sessione creati nel form precedente.

```
<?php
//Richiamo tutte le variabili
$vars = get_defined_vars();
//Array contenente tutte le variabili
$my_array = array();
//Popolo l' array
foreach($vars['_POST'] as $key => $val)
{
        array_push($my_array, $val);
}
```

Con lo script InserimentoFrom è possibile specificare il numero di tuple per file ed il numero di pagine del file per ogni tabella necessari per il calcolo del costo di accesso. Tale pagina consente inoltre la visualizzazione della query inserita al passaggio precedente.

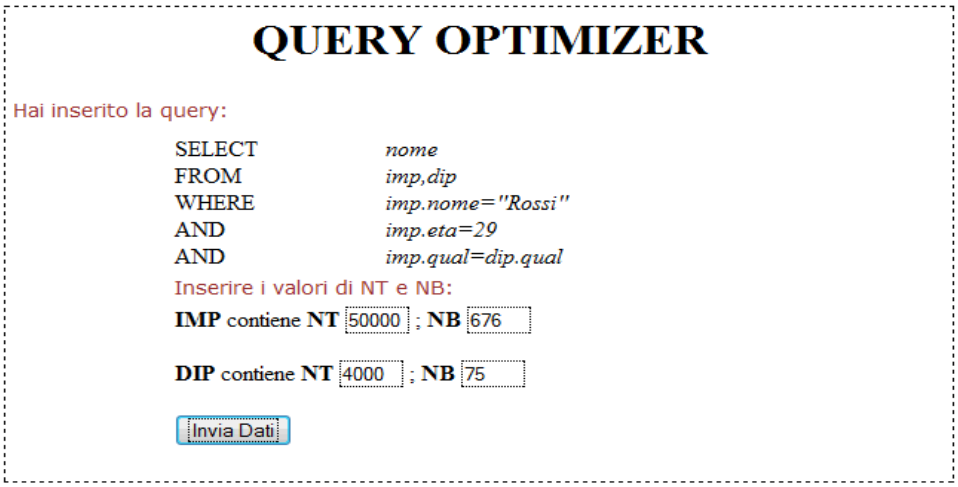

Fig. 4.4 Schermata di inserimento from

```
...
<tr>
<td width="100" align="left">SELECT</td>
<td align="left" width="125"><?php echo "<i>" . $my_array[0] . "</i>"?></td>
</tr>
<tr>
<td width="100" align="left">FROM</td>
<td align="left" width="125"><?php echo "<i>" . stripslashes($my_array[1]) ."</i>"; ?
></td>
</tr>
<tr>
<td width="100" align="left">WHERE</td>
<td align="left" width="125"><?php echo "<i>" . stripslashes($my_array[2]) ."</i>"; ?
</td>
</tr>
<?php
if(isset($my_array[3]))
echo "<tr>
<td width='100' align='left'>". $my_array[3]."</td>
<td align='left' width='125'><i>" . stripslashes($my_array[4]) ."</i></td>
 </tr>";
if(isset($my_array[5]))
echo "<tr>
<td width='100' align='left'>". $my_array[5]."</td>
<td align='left' width='125'><i>" . stripslashes($my_array[6]) ."</i></td>
 </tr>";
if(isset($my_array[7]))
echo "<tr>
<td width='100' align='left'>". $my_array[5]."</td>
<td align='left' width='125'><i>" . stripslashes($my_array[8]) ."</i></td>
 </tr>";
...
```

Per poter recuperare tutti i valori del campo FROM uso il metodo explode() il quale mi divide la frase all'occorrenza di una virgola. Ad esempio se l'utente inserisce: *impiegati,dipartamenti,città* nella variabile $risultati[0] mi troverò *impiegati*, nella $risultati[1] mi troverò *dipartimenti* e cosi via.

```
$from=$_POST['from'];
$risultati=explode(',', $from);
```

Dopo la divisione utilizzo la variabile $risultati in un ciclo for per realizzare e visualizzare un text type per il valore di NT e un text type per il valore di NB per ogni tabella.

```
<?php
...
for($i=0;$i<sizeof($risultati);$i++)
{
echo "<b>" . strtoupper($risultati[$i]) ."</b> contiene <b>NT</b>
<input type='hidden' name=nt" . $i ." size='3' class='text'>
<input type='text' name=nt" . $i ." size='3' class='text'> ;
<b>NB</b> <input type='hidden' name=nb" . $i . " size='3' class='text'>
<input type='text' name=nb" . $i . " size='3' class='text'><br><br>";
}
echo "<input type='submit' value='Invia Dati'>";
...
?>
```

4.1.4 InserimentoCalcolo

InserimentoCalcolo realizza il form di inserimento dati per il calcolo del costo di accesso.
Per ogni tabella tale form comprende i seguenti campi:

- viene riportato il nome di ogni tabella con i rispettivi valori di NT e NB
- per ogni tabella si possono scegliere quali attributi selezionare mediante un apposito checkbox.
- per attributo selezionato si specificano i valori di NK e NF.
- scelta indice tra: clustered, unclustered ordinato o unclustered disordinato. Un

indice clustered non può essere specificato più di una volta per relazione

I dati correttamente inseriti, selezionando il pulsante 'Calcola', saranno passati, mediante il metodo *post*, allo script *stampaRisultati* per la visualizzazione dei costi di accesso.

Con il tasto 'Annulla' sarà possibile cancellare i campi.

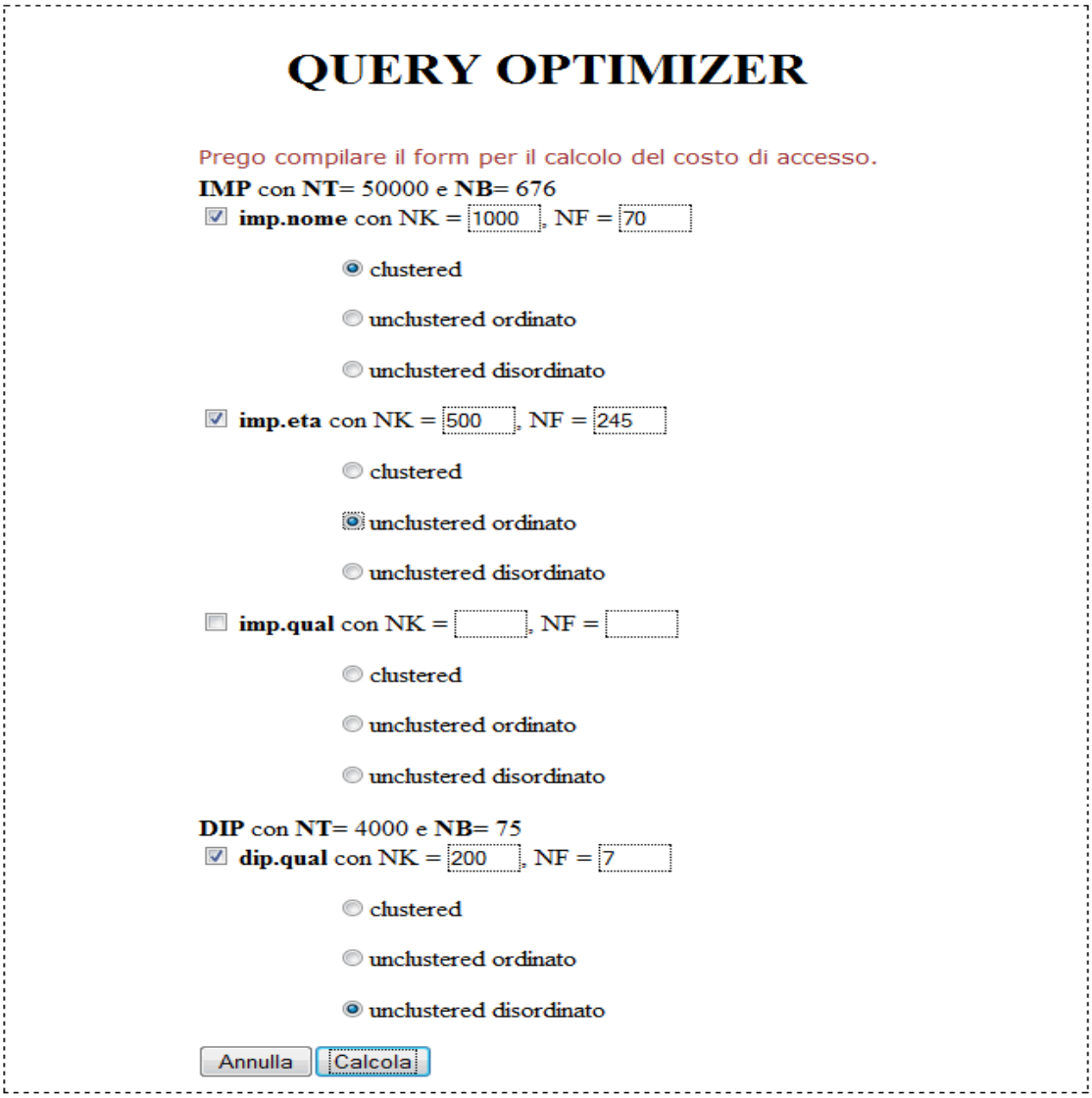

Fig. 4.5 Schermata di calcolo costo di accesso

Attraverso il metodo split(), che serve per suddividere una stringa in più parti in base al carattere passato come primo argomento recupero i dati inseriti nei campi Where, Andor ecc. Siccome split() restituisce un array contenente tanti elementi quanti sono le parti della stringa, recuperare i dati che mi servono per il calcolo del costo della query risulta essere molto semplice. Il metodo viene usato ad esempio quando nel campo Where è stata inserita la stringa impiegati.nome=”Rossi” e mi serve ricavare soltanto la parte impiegati.nome.

```
<?php
...
$rfrom=explode(',', $my_arrayQuery[0]);
$rwhere=split('[=<>]', $my_arrayQuery[1]);
$where=split('[.]', $my_arrayQuery[1]);
$andone=split('[.]',$my_arrayQuery[2]);
$randone=split('[=<>]',$my_arrayQuery[2]);
$ctrlwhere=explode('.',$rwhere[1]);
$ctrlandone=explode('.',$randone[1]);
$randonesecondo=split('[=<>]',$my_arrayQuery[3]);
...
?>
```

Per realizzare il form inserimentoCalcolo occorre innanzitutto recuperare i dati NT e NB relativi ad ogni tabella. Tali dati sono ricavati attraverso il metodo post. Ogni tabella viene inoltre visualizzata in maiuscole con l'aiuto del metodo strtoupper(), che restituisce la stringa *“nome tabella”* con i caratteri alfabetici convertiti in maiuscolo. Con appositi controlli lo script individua la corretta posizione degli attributi. Ogni attributo può essere selezionato attraverso un apposito checkbox. Per ogni attributo vanno specificati i valori di NK e NF realizzati mediante il tag HTML <input type=”text”>. Il calcolo del costo di accesso necessita anche la specifica di uno tra i tre tipi di indice: clustered, unclustered ordinato e unclustered disordinato. La soluzione immediata è stata quella di creare dei bottoni radio legati sempre agli attributi in modo che si possa selezionare al più un indice per attributo. Inoltre un indice clustered può

essere utilizzato una ed una sola volta per tabella.

```
<?php
...
if(isset($rfrom[1]))
{
echo "<b>".strtoupper($rfrom[0]). "</b> con <b>NT</b>= ". $my_arrayQuery[5]. " e
<b>NB</b>= " . $my_arrayQuery[6]."<br>";
if($where[0]==$rfrom[0])
{
echo "<input name='box' type='checkbox' value='cl' style='display:none;>
<input name='box' type='checkbox' value='cl'>". "<b>". $rwhere[0] ."</b>" ." con NK
=
<input type='hidden' name='indexNK' size='3' class=\"text\">
<input type='text' name='indexNK' size='3' class=\"text\">, NF =
<input type='hidden' name='indexNF' size='3' class=\"text\"><input type='text'
name='indexNF' size='3' class=\"text\"><br>";
echo "<blockquote><blockquote>
<input name='bottone' type='radio' value='clustered' style='display:none;>
<input name='bottone' type='radio' value='clustered'>clustered
</blockquote></blockquote>";
echo "<blockquote><blockquote>
<input name='bottone' type='radio' value='unclusteredord' style='display:none;>
<input name='bottone' type='radio' value='unclusteredord'>unclustered
ordinato</blockquote></blockquote>";
echo "<blockquote><blockquote>
<input name='bottone' type='radio' value='unclustereddis' style='display:none;>
<input name='bottone' type='radio' value='unclustereddis'>unclustered disordinato
</blockquote></blockquote>";
}
echo "<input type='reset' value='Annulla' name='annulla'>
<input type='submit' value='Calcola'>";
?>
```

4.1.5 StampaRisultati

Questo script effettua i calcoli e visualizza i risultati. Mediante il metodo *post[]* vengono recuperati i valori dei vari campi inseriti al livello precedente, tali valori saranno necessari per il calcolo del costo di accesso.
Come prima cosa viene mostrata la sequenza di accesso seguita dalla prima relazione. Viene inoltre visualizzato il costo dello scansione sequenziale. Prima di mostrare i risultati il programma controlla se i campi obbligatori per il calcolo sono compilati.

```
echo "<tr><td>RELAZIONE: <b>".
        strtoupper($my_arrayCosto[0]) .
       "</b></td></tr>";
if(isset( $my_arrayCosto[13]))
{
         echo "<tr><td>Costo C<span class='textdown'>(scansione sequenziale)</span>
= NB =  " .  $nb1."</td></tr>";//NB
}
```

Vengono implementate le formule attraverso le quali il programma calcola i costi di accesso. Come prima cosa viene controllato se un attributo è stato selezionato. I controlli sono realizzati con l'aiuto delle funzioni *empty*() e *isset*() per ogni record coinvolto nel calcolo.
Per l'implementazione delle formule per il calcolo degli indici clustered, unclustered ordinati oppure unclustered disordinati viene utilizzata la funzione ceil(), che arrotonda un numero per eccesso, cioè al numero intero superiore e va usata poiché il numero calcolato deve essere un numero intero.
Per prima cosa si esegue il calcolo da sinistra verso destra da sinistra verso destra (esempio: A) sequenza IMP => DIP) , dopo di che si esegue da destra verso sinistra (esempio: B) sequenza: DIP => IMP). Se un costo è stato calcolato precedentemente il programma segnala il risultato come già calcolato come mostrato in figura 4.6.

Costo Join (sequenza IMP => DIP) = $C_{imp.nome} + E_{imp.nome} * C_{dip.qual} = 52$
B) sequenza: DIP => IMP
RELAZIONE: **DIP**
Costo $C_{(scansione\ sequenziale)}$ = NB = 75
Accesso tramite indice su: **dip.qual** (Gia' calcolato)
C = 1
$E_{dip.qual}$ =[F*NT]= 1

Fig. 4.6 Schermata di Già calcolato

```
<?php
....

if(isset($set1))
{
        if(!empty($nk1))
        {
                $F1=1/$nk1;
        }
        if(!empty($nt0))
        {
                $E1=($F1*$nt0);
        }
        if(isset($set1)&&isset($radioCL1))
        {
                if($radioCL1=='clustered1')
                {
                //costo indice clustered
                $C1=(ceil($F1*$nf1)+ceil($F1*$nb1));
                $cprint1="C<spanclass='textdown'>clustered</span>
                        =[F*NF]+[F*NB]=";
                }
                else if($radioCL1=='unclusteredord1')
                {
                //costo indice unclustered ordinato
               $cardenas1=ceil($nb1*(1-pow(1-(1/$nb1),$E1)));
                $C1=ceil($F1*$nf1)+$cardenas1;
                $cprint1="C<span class='textdown'>
```

```
                unclustered TID ordinati</span>
                        =[F*NF]+[?(E,NB)]=";
                }
                else if($radioCL1=='unclustereddis1')
                {
                //costo indice unclustered disordinato
                $C1=ceil(($F1*$nf1)+($F1*$nt1));
                $cprint1="C<span class='textdown'>
                        unclustered TID disordinati</span>
                    =[F*NF]+[F*NT]";
                }
                }
                echo "<tr><td>Accesso tramite indice su: <b>"
                                . $hidind1 ."</b></td></tr>";
                    echo "<tr>";
                        echo "<td>";
                        echo "<tr><td><blockquote>Fattore di filtro F
                            <span class='textdown'>". $hidind1."
                            </span>=1/NK=  " . $F1 ."</div></td></tr>";
                        echo "<tr><td><blockquote>"
                            . $cprint1 . " " .  $C1 ."
                             </div></td></tr>";
                        echo "<tr><td><blockquote>E
                           <span class='textdown'>".$hidind1."
                           </span>=[F*NT]= " . $E1 . "</div></td></tr>";
                        echo "</td>";
                echo "</tr>";
}
```

In fine il programma sceglie quale indice costruire in base al minor costo calcolato.

In figura 4.7 è mostrato un esempio di calcolo della sequenza A

RISULTATI

Legenda:

NT - numero di tuple del file
NB - numero di pagine del file
NK - numero di valori distinti della chiave
NF - numero di foglie dell'indice

Risultari calcolo.

A) sequenza: IMP => DIP

RELAZIONE: **IMP**

Costo $C_{(scansione\ sequenziale)}$ = NB = 676

Accesso tramite indice su: **imp.nome**

Fattore di filtro $F_{imp.nome}$ =1/NK= 0.001

$C_{clustered}$=[F*NF]+[F*NB]= 2

$E_{imp.nome}$ =[F*NT]= 50

Accesso tramite indice su: **imp.eta**

Fattore di filtro $F_{imp.eta}$ =1/NK= 0.002

$C_{unclustered\ TID\ ordinati}$=[F*NF]+[Fi(E,NB)]= 95

$E_{imp.eta}$ =[F*NT]= 100

RELAZIONE: **DIP**

Costo $C_{(scansione\ sequenziale)}$ = NB = 75

Accesso tramite indice su: **dip.qual**

Fig. 4.7 Schermata di calcolo sequenza A

Conclusione e sviluppi futuri

Obiettivi raggiunti:

- L'obiettivo principale di questa tesi è stato di creare un'interfaccia web in grado di eseguire dei calcoli per il costo di accesso.

- Creare un programma in grado di prendere decisioni sull'ordinamento delle relazioni e quali indici costruire. Tutto questo è stato possibile grazie all'utilizzo di una serie di script in linguaggi come HTML, CSS e PHP che consentono una facile implementazione.

- E' stato creato uno strumento didattico in grado di aiutare chi intende studiare o capire meglio questi argomenti, grazie alla semplice interfaccia grafica e alla spiegazione dettagliata passo passo fornita in output, analoga a quella che si trova nelle soluzioni dei relativi esercizi sui libri di testo. E' ovvio che i moduli di query optimization dei DBMS hanno una complessità molto maggiore ma invece non presentano questi aspetti "educativi", perché hanno ovviamente altre finalità.

Sviluppi futuri:

- creare script PHP per estendere i calcoli di join per più di tre tabelle.
- specificare i valori degli attributi nei casi di minore o maggiore
- estendere gli script al fine di calcolare i conti di Proiezione o Modifica

Bibliografia

1 PHP:

1.1 PHP 5 and MySQL Bible ;

1.2 http://php.net/ ;

1.3 http://www.w3schools.com/php/default.asp

2 HTML:

2.1 http://www.w3schools.com/html/default.asp ;

2.2 http://xhtml.html.it/guide/leggi/51/guida-html/

3 CSS :

3.1 CSS - O'Reilly - Cascading Style Sheets The Definitive Guide ;

3.2 http://www.w3schools.com/css/

4 GUIDA OTTIMIZZAZIONE:

4.1 Lucidi Professor Martogla ;

4.2 Esercizi di basi di dati – Fabio Grandi ;

4.3 http://en.wikipedia.org/wiki/Query_optimizer;