PHYSICAL DATABASES

Es. 2 A table STUDENT(RegNo, Name, City) has 100K tuples in 7K blocks with an entry-sequenced organization. There are B +-tree indexes on Name and City, both of depth 3, with the ability to reach 5 tuples in average for each value of Name, and 40 tuples in average for each value of City. Consider the following SQL query:

```
select * from Student
where Name in (Name1, Name2, ..., Namen)
  and City in (City1, City2, ..., Cityc)
```

Determine the optimal strategy for query execution based on the number of values (**n**,**c**) of Name and City listed in the where clause query, considering these four cases:

1) (**n**,**c**) = (10, 10)
2) (**n**,**c**) = (1000, 10)
3) (**n**,**c**) = (10, 1000)
4) (**n**,**c**) = (1000, 1000)

One search based on 1 Name value costs : 3 (tree nodes) + 5 (primary blocks) = 8 i/o operations
One search based on 1 City value costs : 3 (tree nodes) + 40 (primary blocks) = 43 i/o op.s

Scenarios / Used Index	Name	City	Seq. Scan
1)	10 x 8 = **80**	10 x 43 = 430	7.000
2)	1.000 x 8 = 8.000	10 x 43 = **430**	7.000
3)	10 x 8 = **80**	1.000 x 43 = 43.000	7.000
4)	1.000 x 8 = 8.000	1.000 x 43 = 43.000	**7.000**

Es. 3 A table USER(Email, Password, LastName, FirstName) contains 128K users and is stored on 25K blocks, with a primary hash organization on the primary key.

A table REVIEW(Email, ISBN, Date, Rating, ReviewText) has instead 4M tuples and is organized with a primary B +-tree (ie, the tuples are entirely contained in the leaves of the tree) with the email as a key; the average fan-out is equal to 35 and leaf nodes occupy 1M blocks. Estimate the cost of implementing the join between the two tables using the most efficient technique.

Strategy 1

A strategy with Reviews scanned in email order, with a lookup for each such email:

4 + 1M = 1M i/o to scan the external table

128K i/o to lookup all the users (only once per user, as the reviews are in email order)

Overall: 1M + 128K = 1.13 M i/o

Strategy 1

A strategy with Reviews scanned in email order, with a lookup for each such email:

4 + 1M = 1M i/o to scan the external table

128K i/o to lookup all the users (only once per user, as the reviews are in email order)

Overall: 1M + 128K = 1.13 M i/o

Strategy 2

Scanning the Users and looking up the reviews by means of the B+ would cost more:

25 K to scan the users

Cost of 1 lookup : 4 intermediate nodes ($\log_{35}$ 1M ~= 4)
+ 9 leaf nodes to collect the 32 (4M / 128K) reviews by each user
(4M / 1M = 4 reviews per block, 32/4 = 8, we read one additional block because ranges will not start at the beginning of the block)

Overall: 25K + 128K x (4 + 9) = 1.7 M i/o

A tale of two Joins (*31 Jan 2012*)

STUDENT (Matricola, FirstName, LastName, BirthDate, …)

contains 150K students on 10K blocks in a primary hash over attribute Matricola.

EXAM (Matricola, CourseId, Date, Grade)

contains 2M tuples on 8K blocks in a entry-sequenced structure
we also have a secondary B+ index over Matricola, with fanout 200.

Estimate and compare the cost (in terms of number of disk accesses) of executing the query

```
select *
from Student S join Exam E on S.Matricola = E.Matricola
```

a) with a **hash join** in which Exam is re-structured in a suitable primary hash storage
b) with a **nested loop join** with Student as **external** table.
c) with a **nested loop join** with Student as **internal** table.

(Ignore collisions and caching)

STUDENT has 15 tuples per block, each hash-based access costs 1 i/o.

EXAM has 256 tuples per block, and has no hash-based access.

We need to build a hash structure (in order to ba able to use it!). This requires scanning the whole table, extracting each tuple and inserting it into the right «bucket». The number of buckets is of course the same, as the hash funcion is the same (e.g., *Matricola mod 10.000*): these buckets however will be rather «empty» w.r.t. the blocks in the previous representation (the storage grows overall from 8K to 10K blocks).

We therefore need to perform **8K i/o ops** in order to ***read*** and as much as **4M i/o ops to *update***, as the right bucket must be found for each exam, and the corresponding block needs to be read (moved to main mamory), updated with the new exam, and then re-written to disk.

The **hash-join** per se then costs **2 * 10K = 20K i/o**, to an overall cost of **4M + 28K i/o**

(*a remarkable cost, that should be payed off by the efficient future executions of the same join)*

Calcoliamo ora il costo del nested loop con ESAME come tabella **interna** (cioè nel caso in cui leggiamo un blocco della tabella STUDENTE (iterazione esterna) e per ogni studente cerchiamo in ESAME (iterazione interna) tutti gli esami sostenuti da quel particolare studente. Leggiamo un nuovo blocco di STUDENTE solo dopo aver esaurito tutti gli studenti nel blocco corrente).

L'albero ha profondità media pari a circa 3 ed è relativamente sparso (ha spazio per circa 8M valori di chiave). La **ricerca degli esami di un dato studente** costa quindi **3 accessi ai nodi dell'albero**, **più 1** accesso ai blocchi della struttura primaria **entry-sequenced** per recuperare tutti i campi dell'esame (trascuriamo il caso in cui i puntatori agli esami di uno stesso studente siano ripartiti su più di un nodo foglia dell'albero). In totale pagheremo 3 accessi per ogni studente e uno per ogni esame, cioè **3*150K + 2M accessi**.

Il caricamento di tutti i blocchi di STUDENTE nel**l'iterazione esterna costa inoltre 10K** accessi, per cui in totale avremo = **2M + 460K accessi**.

È un costo inferiore a quello di costruirsi la struttura ad-hoc (poco più di metà), il quale però può ancora considerarsi un buon investimento se, conservando la struttura creata, velocizza poi sostanzialmente le esecuzioni dello stesso join (*20K contro 2,5M*).

Calcoliamo infine il costo del nested loop con ESAME come tabella **esterna** (che a prima vista sembra l'opzione più sensata, dato che gli accessi random basati su matricola alla struttura primaria a hash dello STUDENTE costano molto meno di quelli a ESAME basati sull'indice B+ (1 contro 4)).

Possiamo **scandire tutta la tabella ESAME pagando solo 8K** accessi per la sua lettura, sfruttando la struttura entry-sequenced (non avrebbe senso, infatti, scandire in sequenza le foglie dell'indice B+ e accedere ai blocchi primari "uno studente alla volta", dato che avere gli esami in ordine di matricola non ci aiuta), e dobbiamo poi **accedere 2M volte alla struttura hashed**, una volta per ogni esame, a recuperare i dati dello studente (più volte lo stesso studente, ogni volta che ci imbattiamo in un suo esame – ma stiamo trascurando gli effetti della cache), per un totale di **2M+8K accessi**.

Il costo è minore, ma comunque dello stesso ordine di grandezza di quelli precedentemente calcolati. Un ottimizzatore "miope" potrebbe continuare a scegliere questa ultima opzione senza mai "imbarcarsi" nell'impresa di costruire la struttura hash-based, che invece si rivela strategicamente la più conveniente se il join viene ripetuto frequentemente (come è probabile).

B+-primary and Hash-secondary structures (21 Feb. 2012)

A table STUDENT (Matricola,FirstName,LastName,BirthDate,…) has 200K tuples in a ***primary B+ tree*** on attribute *Matricola*, on 10K blocks with a maximum fanout of 100;

there is also a ***secondary hash index*** on *LastName* that takes 2K blocks (val(LastName) = 50K).

Estimate the cost of the following queries, ignoring collisions and caching:

1) select * from Student where Matricola = '623883'

2) select * from Student
where LastName in ('Braga','Campi','Comai','Paraboschi') **and** Matricola > '575478'

3) select * from Student where Lastname < 'B'

Ogni nodo/blocco *interno* all'albero contiene fino a 99 matricole e 100 puntatori fisici, e in ogni nodo/blocco *foglia* dell'albero ci sono invece 20 studenti (le intere tuple) e 1 solo puntatore fisico (al blocco successivo nella "catena delle foglie").

Immaginando molto densa la struttura ad albero (nodi sostanzialmente tutti pieni e albero perfettamente bilanciato), oltre alla radice abbiamo un livello intermedio di 100 blocchi interni e 10K blocchi "foglia", il che corrisponde alla dimensione indicata della struttura. L'albero avrà quindi una profondità pari a 3.

Nei blocchi dell'indice hash (la cui chiave di accesso non è una chiave primaria) ci sono puntatori fisici ai blocchi che contengono i dati di circa (in media) 200K / 2K = 100 puntatori fisici, corrispondenti a meno di 100 valori di chiave (sarebbero esattamente 100 se non esistessero studenti con lo stesso cognome, in realtà se ne avranno in media sensibilmente di meno). I blocchi avranno un fattore di riempimento che dipende dal rapporto di dimensione tra cognomi e puntatori fisici, che i dati non lasciano stimare.

Il costo è quindi

(**query 1**). 3 accessi tramite albero (2 blocchi interni e una foglia), e ho subito l'intera tupla cercata.

Il costo è quindi

(**query 1**). 3 accessi tramite albero (2 blocchi interni e una foglia), e ho subito l'intera tupla cercata.

(**query 2**). Se uso l'indice sul Cognome pago 4 accessi per il lookup nello hash e poi seguo 4 x 200K/50K = 4 x 4 = 16 puntatori, in totale 20 accessi.
avendo 4 condizioni in OR e 4 puntatori ai blocchi fisici da seguire.

È ragionevole ritenere che la condizione sulla matricola sia inservibile (non riteniamo probabile che esistano molto meno di 20x20 = 400 studenti con matricola superiore a 575478) e una strategia di interval-query fallimentare.

Il costo è quindi

(**query 1**). 3 accessi tramite albero (2 blocchi interni e una foglia), e ho subito l'intera tupla cercata.

(**query 2**). Se uso l'indice sul Cognome pago 4 accessi per il lookup nello hash e poi seguo 4 x 200K/50K = 4 x 4 = 16 puntatori, in totale 20 accessi.
avendo 4 condizioni in OR e 4 puntatori ai blocchi fisici da seguire.

È ragionevole ritenere che la condizione sulla matricola sia inservibile (non riteniamo probabile che esistano molto meno di 20x20 = 400 studenti con matricola superiore a 575478) e una strategia di interval-query fallimentare.

(**query 3**). Siccome in base alla traccia non possiamo garantire che la funzione di hash dia valori correlati all'ordinamento alfabetico, nessuna delle due strutture di accesso ci aiuta, e dobbiamo scandire l'intera tabella (10K accessi)