CALCOLO DEL COSTO DI JOIN

scopo della lezione

- **scopo:**
  - **valutare quale sia la migliore strategia di accesso per interrogazioni SQL nel caso di join**
  - **i criteri di valutazione servono anche a prendere decisioni sull'ordinamento delle relazioni e quali indici costruire**
  - **I criteri che vedremo sono in linea con i metodi e le scelte utilizzati dai query-optimizer dei DBMS relazionali**

utilizzo degli indici

CASO DEL JOIN

```
SELECT cognome, salario
FROM impiegati as IMP, dipartimenti as DIP
WHERE lavoro = 'fattorino'
AND sede= 'milano'
AND imp.dno = dip.dno
```

con: dipartimenti (dno, nome, sede,..)

impiegati.dno = dipartimenti.dno

(o dipartimenti.dno = impiegati.dno)

è argomento di ricerca ?

utilizzo degli indici

Esecuzione del JOIN con il metodo nested loops:

```
{loop 1}:per ogni tupla della relazione IMP
                se soddisfa ai predicati su IMP allora
{loop 2}:  per ogni tupla della relazione DIP
                se soddisfa ai predicati su DIP e
                se soddisfa al predicato di jOIN
                        allora la giunzione delle due
                        tuple fa parte del risultato
        fine
fine
```

sono possibili due sequenze:
IMP $\Rightarrow$ DIP, DIP $\Rightarrow$ IMP

costo di esecuzione

Costi di esecuzione del JOIN con il metodo nested loops:

(con scan sequenziale delle due relazioni)

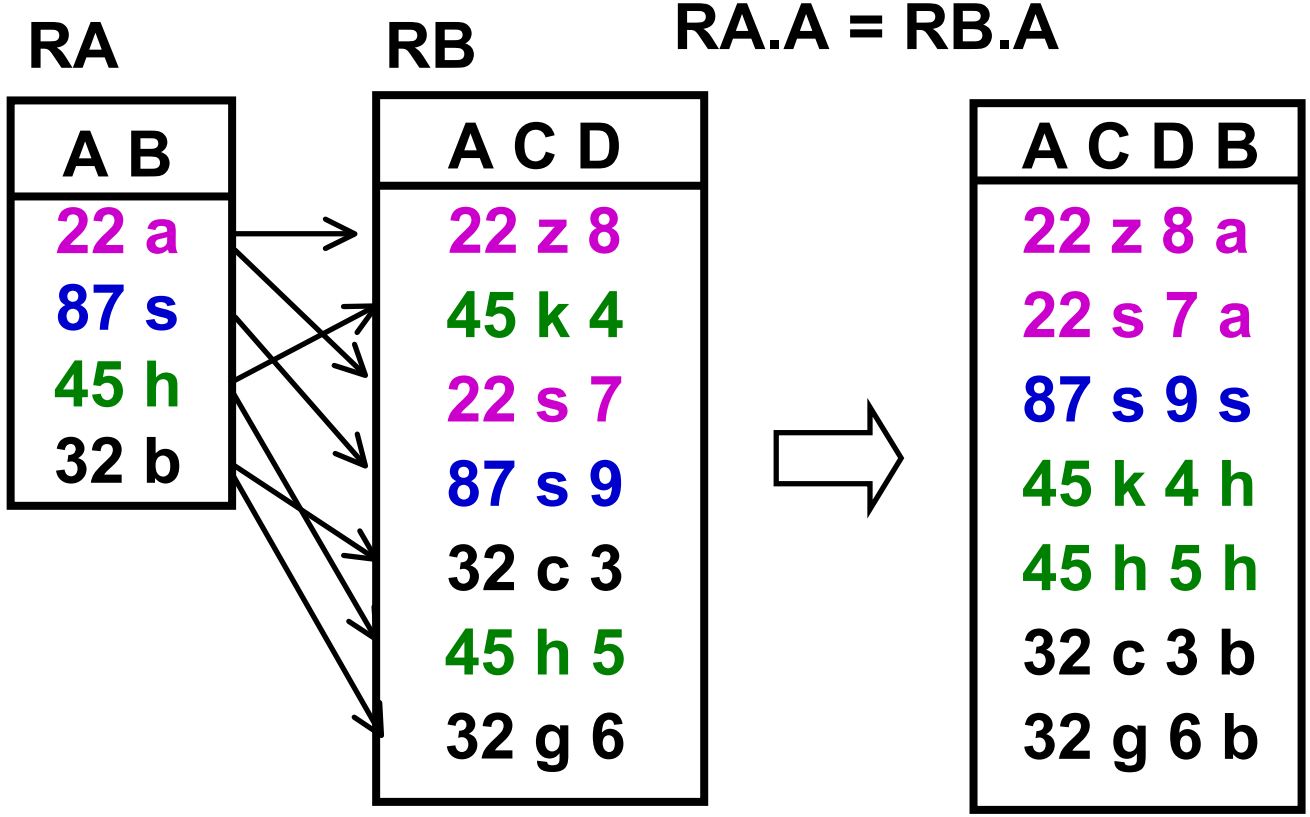
$$C_{join} = NB_{ext} + ET_{ext} \times NB_{int}$$

(con metodi di accesso più economici, es. indici)

$$C_{join} = Ca\,(R_{ext}) + ET_{ext} \times Ca(R_{int})$$

utilizzo degli indici

Esecuzione del JOIN con il metodo nested loops:



utilizzo degli indici

CASO DEL JOIN:

- **un predicato di join è utilizzabile come argomento di ricerca (ed è utilissimo!) solo se è possibile sostituire un valore al posto di uno dei due attributi.**

– **se nell'esecuzione del join si visita prima impiegati allora per l'accesso a dipartimenti**

  valore = dipartimenti.dno **SI**

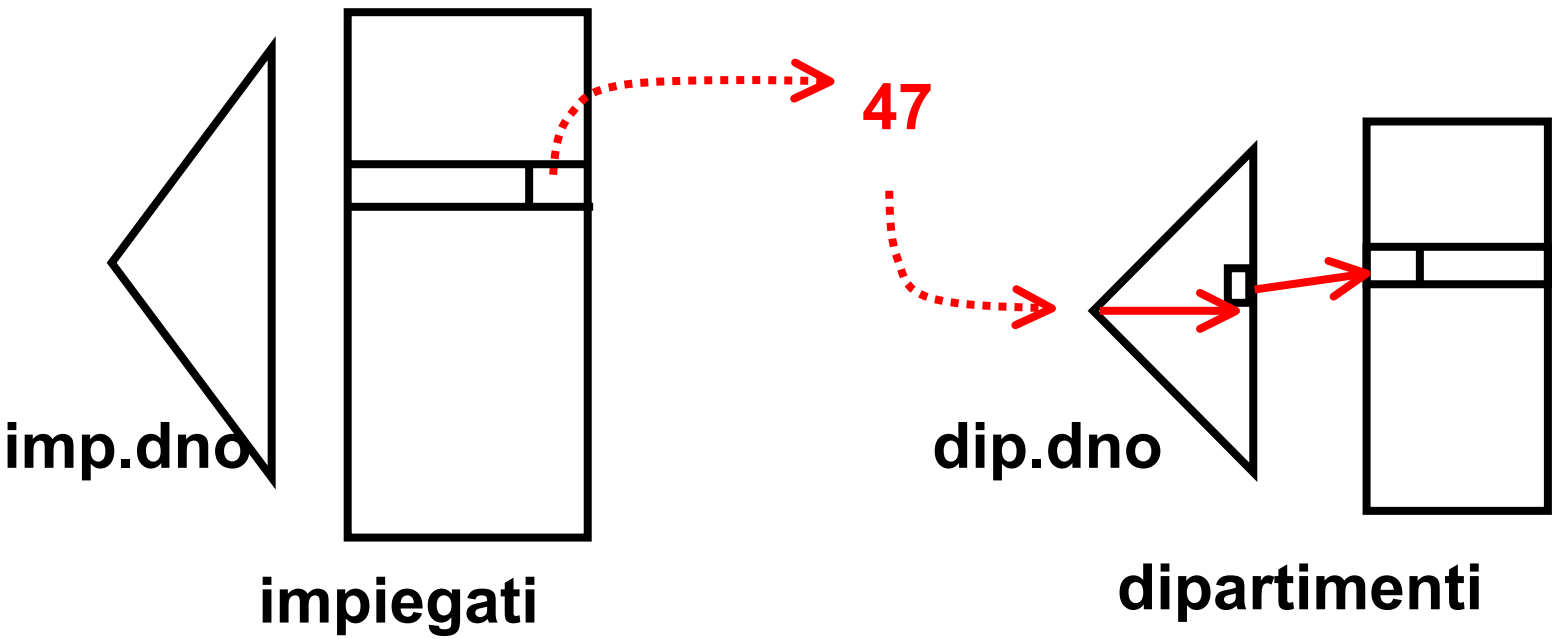
  (il valore è quello trovato in una delle tuple di impiegati)

  per impiegati sarebbe: impiegati.dno = ? **NO**

utilizzo degli indici

nella sequenza impiegati ⇒ **dipartimenti**

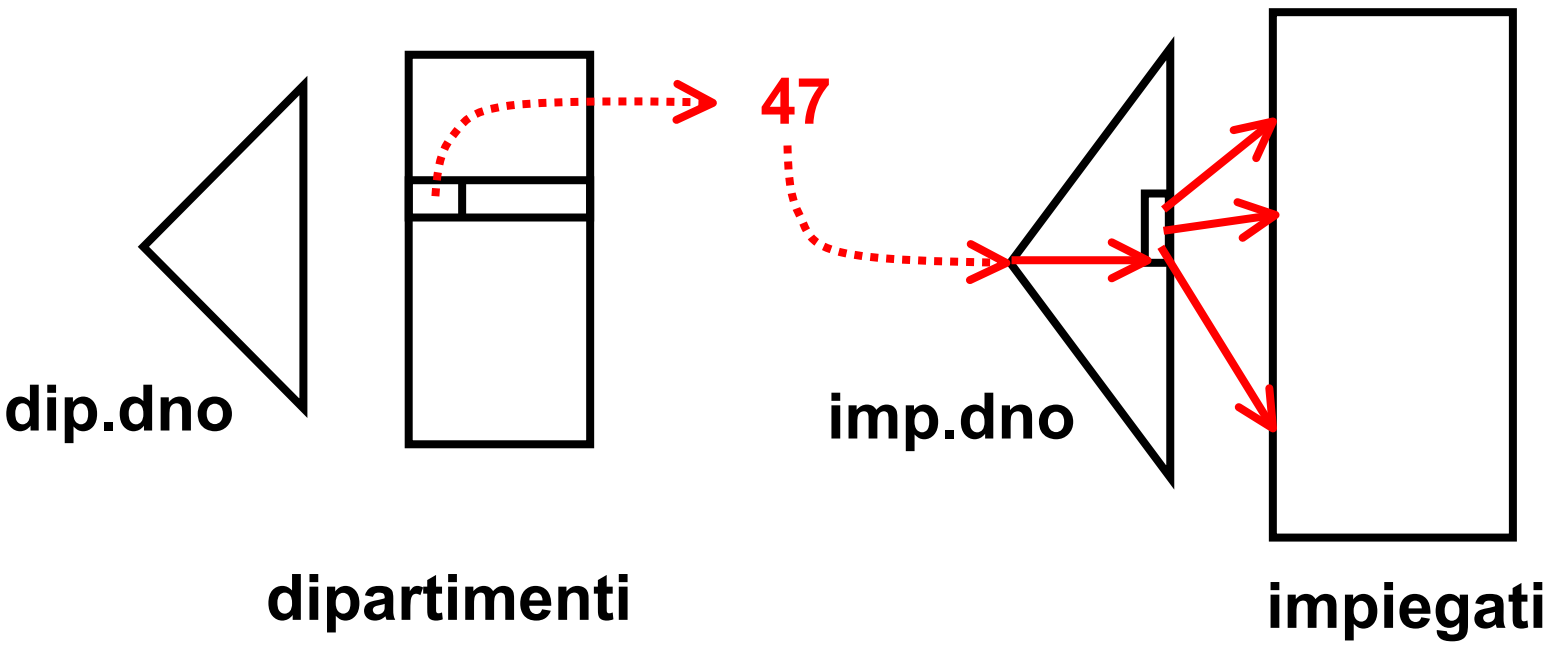
imp.dno = ? **e** **dip.dno = valore**



utilizzo degli indici

nella sequenza dipartimenti ⇒ impiegati

dip.dno = ? e imp.dno = valore



costo di accesso

Esercizio, con le relazioni:

prodotti (pn, pnome, tipo, pj),

NT = 2000, NB= 400

progetti (pj, nome, dno),

NT = 200, NB= 100

gli indici su:

dno	**con NK = 20 , NL = 5**
tipo	**con NK = 100 , NL = 10**
prodotti.pj	**con NK = 200 , NL = 15**
progetti.pj	**con NK = 200 , NL = 10**

costo di accesso

Ed il JOIN :

```
SELECT pj, nome
FROM prodotti, progetti
WHERE      dno = 136
AND        tipo = 'AA'
AND        prodotti.pj = progetti.pj
```

calcoliamo il costo nel caso

prodotti $\Rightarrow$ progetti e

nessun indice è clustered

costo di accesso

prodotti $\Rightarrow$ progetti

accesso a prodotti:

C_{seq} = 400,

F_{tipo} = 1 / 100, E_{tipo} = 2000/100 = 20

$C_{tipo} = \lceil 10 / 100 \rceil + \lceil 2000 / 100 \rceil = 21$

l'indice su pj non si può usare su prodotti

accesso a progetti:

C_{seq} = 100, F_{dno} = 1 / 20, E_{dno} = 200 / 20 = 10

$C_{dno} = \lceil 5 / 20 \rceil + 10 = 11$ (indice su dno)

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

costo di accesso

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:

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

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

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

$= 400 + 20 \times 100 = 2400$

costo di accesso

progetti $\Rightarrow$ prodotti

accesso a progetti:

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

$C_{dno} = 11$ (già calcolato)

l'indice su pj non si può usare su progetti

accesso a prodotti:

$C_{tipo} = 21$ (già calcolato)

$C_{prodotti.pj} = \lceil 15 / 200 \rceil + 10 = 11$ (indice su pj)

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

$11 + 10 \times 11 = 121$

costo di accesso

Sono date le seguenti relazioni:

- **BUS (CB, MODELLO, TARGA, ANNO,...)**
  con 100 n-ple in 50 pagine, CB è la chiave e MODELLO ha 20 valori diversi;
- **PERCORSI (CP, LUOGO_P, LUOGO_A,....)**
  con 500 n-ple in 100 pagine, CP è la chiave
  LUOGO_P e LUOGO_A hanno entrambi 100 valori diversi;
- **CORSE (CC, CB, CP, T_IN, T_OUT,...)**
  con 10000 n-ple in 1000 pagine, CC è la chiave ,
  il numero di valori in T_IN e T_OUT è ignoto.

costo di accesso

Scrivere un join che risolva l'interrogazione:

“selezionare tutti i bus di modello =Z partiti prima di un tempo T_OUT > TX e rientrati prima di un tempo T_IN > TY, dove LUOGO_P=A oppure LUOGO_A=P”:

```
SELECT  *, --, --
FROM BUS  B, CORSE  C, PERCORSI P
WHERE       B.MODELLO = 'Z'
AND   C.T_OUT> TX  AND  C.T_IN >TY
AND   (P.LUOGO_P = 'A' OR P.LUOGO_A ='B')
AND   B.CB = C.CB   AND    P.CP = C.CP
```

costo di accesso

Per le sequenze: **BUS ⇒ CORSE ⇒ PERCORSI** **e**
PERCORSI⇒CORSE ⇒BUS

- **si stabilisca la migliore disposizione di indici per il metodo nested-loops considerando le relazioni non ordinate e gli indici unclustered con tids ordinati.**
- **Si proponga un solo indice per relazione .**
- **Per la migliore delle due soluzioni individuare quale degli indici proposti potrebbe essere il migliore indice di ordinamento.**
- **Si assuma il numero di pagine di ogni indice uguale al 10% del numero di pagine della relazione.**

CASO: BUS ⇒ CORSE ⇒PERCORSI

a) <u>BUS ESTERNA</u>

$C_{BUS}(seq) = 50$ $f(modello) = 1/20$

$$C_{BUS}(modello) = \lceil 5 \times 1/20 \rceil + \lceil \Phi(100/20,50) \rceil =$$
$$= 1 + \lceil \Phi(5,50) \rceil = 6$$

$$E_{BUS} = \lceil 100 \times 1/20 \rceil = 5$$

CASO: BUS ⇒ CORSE ⇒PERCORSI

b) CORSE INTERNA

$C_{CORSE}(seq) = 1000$

$f(t\_out) = 1/3 \quad f(t\_in) = 1/3 \quad f(cb) = 1/100$

$$C_{CORSE}(cb) = \lceil 100 \times 1/100 \rceil + \lceil \Phi(10000/100,1000) \rceil$$
$$= 1 + \lceil \Phi(100,1000) \rceil = 1 + 96 = 97$$

$$C_{BUS\text{-}CORSE} = C_{BUS}(modello) + E_1 \times C_{CORSE}(cb) =$$
$$= 6 + 5 \times 97 = 491$$

$$E_{BUS\text{-}CORSE} = \lceil 100 \times 10000 \times 1/20 \times 1/3 \times 1/3 \times 1/100 \rceil$$
$$= 56$$

CASO: BUS ⇒ CORSE ⇒PERCORSI

C) **PERCORSI**

$C_{PERCORSI}(seq) = 100 \qquad f(cp) = 1/500$

$C_{PERCORSI}(cp) = \lceil 1/500 \times 10 \rceil + \lceil \Phi(500/500,100) \rceil = 2$

$C_{B\text{-}C\text{-}P} = C_{BUS\text{-}CORSE} + E_{BUS\text{-}CORSE} \times C_{PERCORSI}(cp) =$

$= 491 + 56 \times 2 = 603$

CASO : PERCORSI ⇒CORSE⇒BUS

d) PERCORSI ESTERNA

$C_{PERCORSI}$ **(seq)=100**

f(luogo_p)= 1/100 **f(luogo_a)= 1/100**

f(luogo_p = <val> or luogo_a = <val>) =

= f(luogo_p) + f(luogo_a) - f(luogo_p) × f(luogo_a) =

= 1/100 + 1/100 - 1/100 × 1/100 = 199/10000

$E_{PERCORSI} = \lceil 500 \times 199/10000 \rceil = 10$

CASO : PERCORSI ⇒CORSE⇒BUS

e) **CORSE INTERNA**

$C_{CORSE}(seq) = 1000$

$f(t\_out) = 1/3 \qquad f(t\_in) = 1/3 \qquad f(cp) = 1/500$

$C_{CORSE}(cp) = \lceil 1/500 \times 100 \rceil + \lceil \Phi(10000/500, 1000) \rceil =$

$= 1 + \lceil \Phi(20,1000) \rceil = 21$

$C_{PERCORSI\text{-}CORSE} = C_{PERCORSI}(seq) + E_{PERCORSI} \times C_{CORSE}(cp) =$

$= 100 + 10 \times 21 = 310$

$E_{PERCORSI\text{-}CORSE} =$

$= \lceil 10000 \times 500 \times 199/10000 \times 1/3 \times 1/3 \times 1/500 \rceil = 23$

CASO : PERCORSI ⇒CORSE⇒BUS

f) BUS

C_{BUS}(seq) = 50,

f(modello) = 1/20 , C_{BUS} (modello) = 6

f(cb)=1/100 $C_{BUS}(cb) = \lceil 1/100 \times 5 \rceil + \lceil \Phi(100/100,500) \rceil = 2$

$C_{P\text{-}C\text{-}B} = C_{PERCORSI\text{-}CORSE} + E_{PERCORSI\text{-}CORSE} \times C_{BUS}$ (cb) =

=310 + 23 × 2 = 356

Risulta migliore PERCORSI⇒ CORSE⇒ BUS :

PER LA RELAZIONE PERCORSI: SCAN. SEQ.

PER LA RELAZIONE CORSE: INDICE SU CP,

PER LA RELAZIONE BUS: INDICE SU CB,

CASO : PERCORSI ⇒CORSE⇒BUS

Senza indici avremmo avuto

nel caso PERCORSI⇒ CORSE⇒ BUS :

$$C_{P\text{-}C\text{-}B} = C_{PERCORSI}(seq) + E_{PERCORSI} \times C_{CORSE}(seq) + E_{PERCORSI\text{-}CORSE} \times C_{BUS}(seq) =$$

$$= 100 + 10 \times 1000 + 23 \times 50 = 11250$$

nel caso BUS ⇒ CORSE⇒ PERCORSI :

$$C_{B\text{-}C\text{-}P} = C_{BUS}(seq) + E_{BUS} \times C_{CORSE}(seq) + E_{BUS\text{-}CORSE} \times C_{PERCORSI}(seq) =$$

$$= 50 + 5 \times 1000 + 56 \times 100 = 10650$$

INDICI DI ORDINAMENTO

Supponiamo di avere nella relazione CORSE.CP ordinato:

$f(cp) = 1/500$

$C'_{CORSE}(cp) = \lceil 1/500 \times 100 \rceil + \lceil 1/500 \times 1000 \rceil = 3$

e quindi: $C'_{PERCORSI\text{-}CORSE} =$

$= C_{PERCORSI}(seq) + E_{PERCORSI} \times C'_{CORSE}(cp) =$

$= 100 + 10 \times 3 = 130$

per poi ottenere un costo complessivo:

$C'_{P\text{-}C\text{-}B} = C'_{PERCORSI\text{-}CORSE} + E_{PERCORSI\text{-}CORSE} \times C_{BUS}(cb) =$

$= 130 + 23 \times 2 = 176$

quindi poco più della metà del costo precedente.

INDICI DI ORDINAMENTO

Per la relazione BUS se CB fosse ordinato:

f(cb) = 1/100

$$C'_{BUS}(cb) = \lceil 1/100 \times 5 \rceil + \lceil 1/100 \times 50 \rceil = 2$$

Il costo rimane invariato come se l'indice fosse disordinato con tids ordinate, quindi l'indice da proporre per l'ordinamento potrebbe essere CP sulla relazione CORSE.

costo di accesso

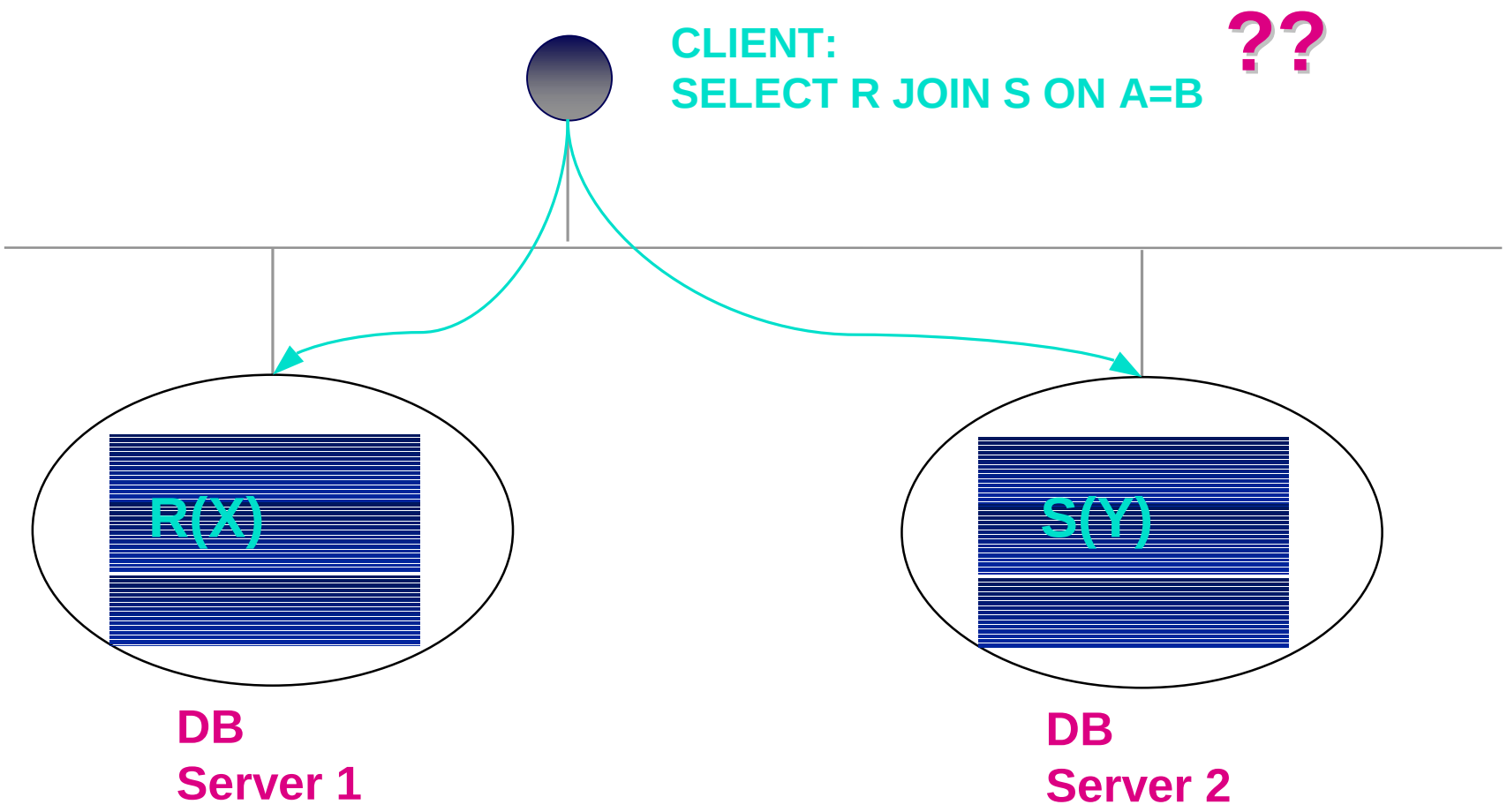
- **Gli ottimizzatori generalmente scartano a priori le sequenze che contengono prodotti cartesiani (ad es. BUS $\Rightarrow$ PERCORSI $\Rightarrow$ CORSE) perché potrebbero dare luogo a E intermedie troppo elevate, anche se questo non sempre è vero.**
- **Gli ottimizzatori valutano tutte le sequenze dove due relazioni consecutive sono collegate da predicati di join e scelgono la migliore.**
- **Esistono molti altri algoritmi di join oltre al nested loops che comunque è uno dei più adoperati e risulta in linea di massima molto efficace in presenza degli indici opportuni.**

costo di accesso

Conclusioni:

- **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**
- **bisogna trovare un compromesso sensato**

Ottimizzazione query distribuite



Ottimizzazione query distribuite

- **Il Join coinvolge relazioni (o frammenti) che risiedono su siti diversi**
- **Come (e dove?) eseguire il Join?**
- **Occorre trasferire dei dati: i modelli di costo per l'ottimizzazione devono tenere conto dei costi di trasmissione:**

  C_0 = costo di avvio di una trasmissione

  C_1 = costo di trasmissione di una tupla

  $C_t(R) = C_0 + C_1 \times NT_R$
  = costo di trasmissione della relazione R

Join distribuito

- **Anche in ambiente distribuito l'esecuzione del join è l'operazione più onerosa**

- **Alcuni metodi molto usati per l'esecuzione del join distribuito si basano sull'operazione di semijoin e sulle sue proprietà algebriche**

- **Tali metodi consentono di minimizzare la quantità di dati che devono essere trasferiti (e quindi di ridurre i costi di trasmissione)**

Semi-Join: definizioni

- **Partendo dal join naturale r1 ▷◁ r2 fra due relazioni r1 e r2, istanze rispettivamente di R1(X1) e R2(X2), si possono definire due operazioni di semi-join come segue:**

$$r1 \triangleright\!< r2 = \pi_{X1}\,(r1 \triangleright\!\triangleleft r2)$$
$$r1 >\!\triangleleft r2 = \pi_{X2}\,(r1 \triangleright\!\triangleleft r2)$$

- **Il join "intero" può essere ricostruito come:**

$$r1 \triangleright\!\triangleleft r2 = (r1 \triangleright\!< r2) \triangleright\!\triangleleft r2 = (r1 >\!\triangleleft r2) \triangleright\!\triangleleft r1$$

Semijoin: esempi

Voli

Codice	Data	Comandante
AZ427	21/07/2001	Bianchi
LH427	23/07/2001	Rossi
TW056	21/07/2001	Smith

Linee

Codice	Partenza	Arrivo
AZ427	FCO	JFK
AF235	CDG	MPX
TW056	LAX	FCO

Voli ▷◁ Linee

Codice	Data	Comandante	Partenza	Arrivo
AZ427	21/07/2001	Bianchi	FCO	JFK
TW056	21/07/2001	Smith	LAX	FCO

Voli ▷< Linee

Codice	Data	Comandante
AZ427	21/07/2001	Bianchi
TW056	21/07/2001	Smith

Voli >◁ Linee

Codice	Partenza	Arrivo
AZ427	FCO	JFK
TW056	LAX	FCO

Metodo del semijoin

- **Esecuzione distribuita del Join $r \rhd\lhd s$**
- **Sfrutta l'equivalenza algebrica con $(s \rhd< r) \rhd\lhd r$**
- **L'algoritmo consta dei seguenti passi:**

1. **Sul sito di r: calcola $r1 := \pi_{X \cap Y}(r)$**
2. **Trasferisci r1 nel sito di s**
3. **Sul sito di s: calcola $s1 := s \rhd\lhd r1$**
   (NB: s1 non è altro che il semijoin $s \rhd< r$)
4. **Trasferisci s1 nel sito di r**
5. **Sul sito di r: calcola $q := s1 \rhd\lhd r$**

Convenienza del metodo

- **Con metodo "naïve"**

  (trasferimento dell'intera s sul sito di r

  dove viene eseguito il Join r ▷◁ s)

  ho costi di trasmissione:

  $Ct' = C_0 + C_1 \times NT_S$

- **Col metodo del semijoin**

  ho costi di trasmissione:

  $Ct'' = 2\,C_0 + C_1 \times (NT_{R1} + NT_{S1})$

  ➔ **Risulta + conveniente ($Ct'' < Ct'$) se:**

  $C_0 + C_1 \times (NT_{R1} + NT_{S1}) < C_1 \times NT_S$

Conclusioni

- **Il metodo può essere facilmente esteso al caso di θ-join**
- **Presuppone che i costi di trasmissione siano più elevati rispetto a quelli di elaborazione locale (non si usa in sistemi basati su LAN con velocità paragonabile al transfer rate da disco a memoria centrale)**
- **Se sono molti gli attributi coinvolti dal join può essere comunque più vantaggioso trasferire l'intera relazione piuttosto che procedere al computo della proiezione**
- **Ci sono anche metodi più avanzati (es. Two-way semijoin)**