

TUNING: LIVELLO FISICO

TUNING FISICO

- Dato un carico di lavoro, l'attività di tuning fisico si preoccupare di progettare uno schema fisico, in termini di insieme di indici creati, che permetta di rendere il più possibile efficiente l'esecuzione delle operazioni contenute nel workload
- Nello specifico
 - Su quali tabelle e attributi creare indici?
 - Che tipi di indici creare?
 - albero/hash
 - clusterizzato/non clusterizzato
 - chiave di ricerca: singolo attributo/multi-attributo

ATTENZIONE

- Una scelta impropria degli indici può portare a
 - indici che vengono mantenuti dal sistema ma mai usati (e quindi hanno solo un impatto negativo sugli aggiornamenti)
 - scansioni sequenziali per recuperare uno o pochi record
 - join multipli con tempi di esecuzione elevatissimi (ore o giorni)
 - spreco di spazio disco
- La scelta degli indici è guidata dai piani che ci aspettiamo che l'ottimizzatore del DBMS utilizzato considererà per l'interrogazione
- E' opportuno quindi cercare di scegliere indici che siano utilizzabili nel maggior numero possibile di interrogazioni

TUNING FISICO

- Il progetto fisico può essere visto come un problema di ottimizzazione in cui il costo di ogni query/transazione viene modificato dalla presenza/assenza di un indice
- È però evidente che lo spazio delle possibili soluzioni è enorme
- Rispetto all'adozione di strumenti (semi-)automatici che richiedono il dettaglio di un workload, l'alternativa è considerare un approccio euristico di tipo incrementale
 - Prima si ragiona su quali indici potrebbero migliorare le prestazioni delle query più importanti, analizzando una query alla volta
 - Quindi si considera se l'aggiunta di ulteriori indici può migliorare la soluzione

SCELTA DEGLI INDICI - APPROCCIO GENERALE

- Si considerano una alla volta le interrogazioni, partendo da quelle ritenute più importanti (= usate più di frequente)
 - Per ogni interrogazione Q_i
 - Gli attributi che appaiono in una clausola WHERE sono candidati come chiavi di ricerca per un indice
 - Si prova a ipotizzare un potenziale piano di esecuzione ottimale per l'interrogazione
 - Si individuano gli indici che permettono al sistema di prendere in considerazione il piano individuato nello spazio dei piani, tenendo presente gli algoritmi di realizzazione messi a disposizione dal sistema
 - Si valuta empiricamente l'efficacia degli indici creati sull'interrogazione (cioè si esegue l'interrogazione in presenza e in assenza degli indici e si confrontano le prestazioni)
 - Se gli indici non sono efficaci, cercare di comprenderne la motivazione ed eventualmente rimuoverli dal livello fisico

SCELTA DEGLI INDICI - APPROCCIO GENERALE

- Si valuta se gli indici identificati al passo i (per la query Q_i) sono compatibili con gli indici identificati fino al passo $i-1$ (per la query Q_{i-1}), ed eventualmente si apportano correttivi
 - Solo un indice per relazione può essere clusterizzato
 - Impatto di indici distinti su insiemi di attributi non disgiunti (ad esempio, $I_R(A)$, $I_R(A,B)$, $I_R(B,A)$)

SCELTA DEGLI INDICI – APPROCCIO GENERALE

APPROCCIO ITERATIVO ALLA PROGETTAZIONE FISICA

INPUT: carico di lavoro C espresso come lista di interrogazioni, ordinate rispetto a frequenza e importanza, $C = \langle Q_1, \dots, Q_n \rangle$

OUTPUT: schema fisico per C

- a. Partire con uno schema $S_0 = \{\}$
- b. Per ogni i , $1 \leq i \leq n$:
 - i. Considerare l'interrogazione Q_i e lo schema fisico ΔS_i ritenuto ottimale per Q_i (identificato al punto A dell'esercizio)
 - ii. $S_i \leftarrow S_{i-1} \otimes \Delta S_i$
dove \otimes rappresenta un operatore di fusione che tiene conto di eventuali conflitti tra S_{i-1} e ΔS_i (ad esempio dovuti alla presenza di più di un indice clusterizzato per una stessa relazione).
- c. Restituire S_n come schema fisico finale.

NEL SEGUITO

- Discuteremo alcune situazioni particolari e formuleremo alcuni principi guida per la scelta degli indici in presenza di particolari tipi di interrogazioni
- Ci concentreremo su
 - Indici per condizioni di selezione
 - Indici per tabelle piccole
 - Indici ad albero/hash
 - Scelta di indici clusterizzati e non clusterizzati
 - Indici multi-attributo
 - Indici per join

SCELTA DEGLI INDICI: TABELLE PICCOLE

- **Tabella piccola** = tabella memorizzata su pochi (eventualmente 1) blocchi

E' opportuno evitare di creare indici su tabelle piccole

- Se i dati di una tabella risiedono in una o poche pagine, la scansione sequenziale è sempre una strategia efficiente
 - Vengono accedute tutte le pagine (poche) su cui la relazione è memorizzata
- Non serve creare indici → tuning fisico non necessario
- Nel seguito consideriamo solo interrogazioni che coinvolgono tabelle non piccole

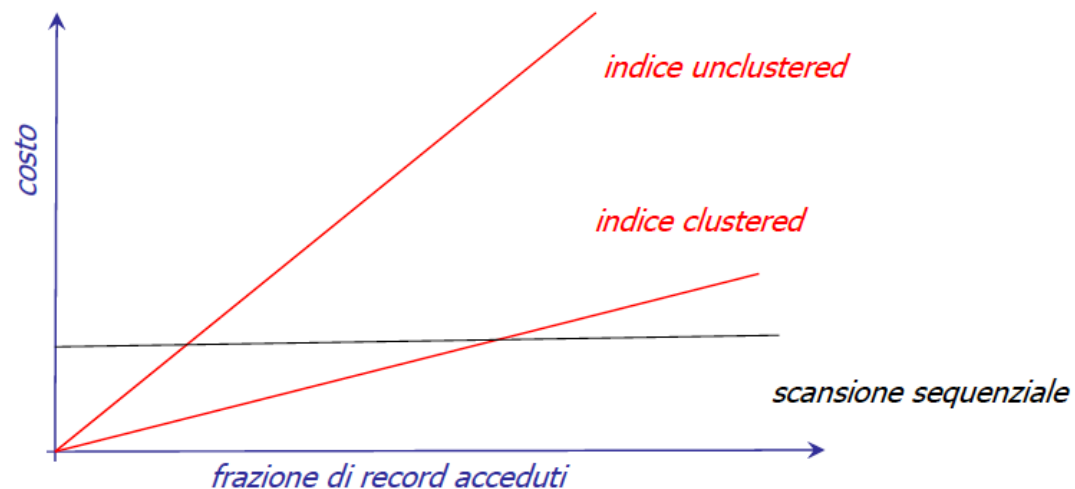
SCELTA DEGLI INDICI: INDICI ORDINATI E HASH

- Interrogazioni supportate da indici ordinati
 - Selezioni (e join) con condizioni di uguaglianza o intervallo
 - Ordinamento
 - Raggruppamento
- Interrogazioni supportate da indici hash
 - Selezioni (e join) con condizioni di uguaglianza
 - Raggruppamento

- Per query con condizioni di selezione e join di uguaglianza, un indice hash può garantire migliori prestazioni, ma la presenza di overflow può però limitare il vantaggio
 - Nei casi dubbi, meglio indice ad albero
- Per query con condizioni di selezione e join di tipo intervallo, solo un indice ordinato permette l'accesso ai dati

SCELTA DEGLI INDICI: INDICI ORDINATI CLUSTERIZZATI E NON CLUSTERIZZATI

- Scansione sequenziale: $B(R)$ accessi
- Accesso con indice ad albero: h accessi a disco + accesso a k blocchi contenenti le tuple risultato
 - In generale k è maggiore per indici non clusterizzati
 - Se indice non è clusterizzato, k nel caso peggiore equivale al numero di tuple restituite dall'interrogazione (RIVEDERE COSTI OPERATORI FISICI SELEZIONE)



- L'indice è preferibile alla scansione sequenziale se $B(R) > h + k$
- k è piccolo per interrogazioni ad alta selettività (quindi con una bassa frazione di record acceduti)

ESEMPIO 1: SCANSIONE SEQUENZIALE E ACCESSO CON INDICI NON CLUSTERIZZATI

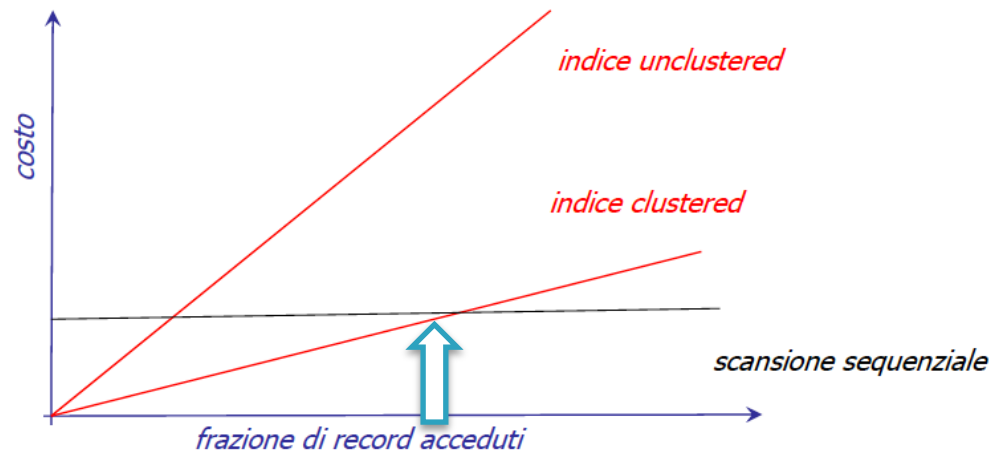
- Dim. record: 50B
- Dim pagina: 4kB
 - $4k/50 = 80$ Record/pagina
- N tuple/record
- $V(A,R) = 20$
- Data una condizione $A = v$, un indice non clusterizzato aiutare?
- Valutazione
 - $NR = N/20$ numero record restituiti (selettività condizione)
 - $NP = N/80$ numero pagine per R
 - Poiche' $NR > NP$ l'indice non aiuta

ESEMPIO 2: SCANSIONE SEQUENZIALE E ACCESSO CON INDICI NON CLUSTERIZZATI

- Dim. record: 2kB
- Dim pagina: 4kB
 - $4k/2k = 2$ Record/pagina
- N tuple/record
- $V(A,R) = 20$
- Data una condizione $A = v$, un indice non clusterizzato aiutare?
- Valutazione
 - $NR = N/20$
 - $NP = N/2$
 - Poiche' $NR \ll NP$, l'indice puo' essere utile

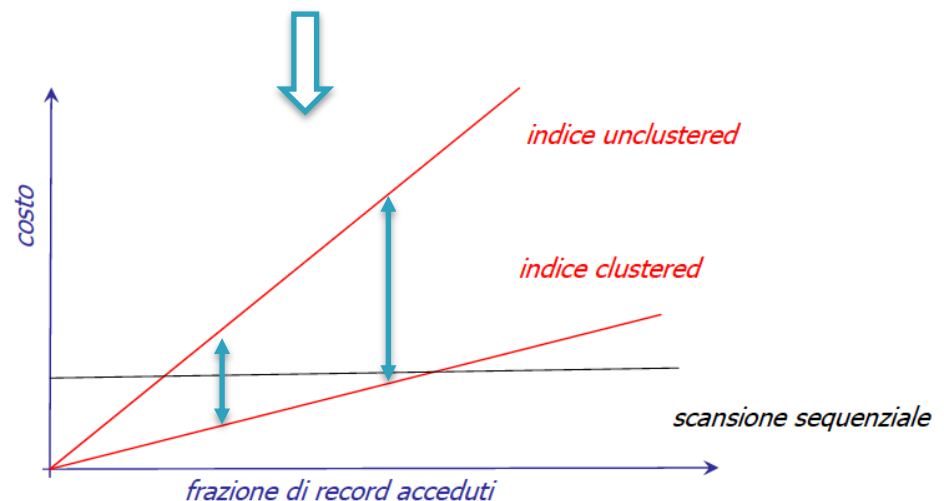
SCELTA DEGLI INDICI: INDICI ORDINATI CLUSTERIZZATI E NON CLUSTERIZZATI

Se su una tabella è necessario creare un solo indice, conviene crearlo clusterizzato (sempre più conveniente di indice non clusterizzato)



SCELTA DEGLI INDICI: INDICI ORDINATI CLUSTERIZZATI E NON CLUSTERIZZATI

- Se su una tabella è necessario creare più di un indice, è necessario ricordare che **un solo indice per relazione può essere clusterizzato**
- In questo caso è opportuno
 - clusterizzare l'indice con chiave di ricerca coinvolta in una condizione della interrogazione a selettività più bassa (frazione record restituiti più alta)
 - favorire l'indice che permette di ottimizzare più di una interrogazione



SCELTA DEGLI INDICI: INDICI MULTI-ATTRIBUTO

- Gli indici multi-attributo richiedono uno spazio maggiore per la loro memorizzazione e vengono aggiornati più spesso
- Permettono di utilizzare l'indice in un maggior numero di interrogazioni

ESEMPIO

- `Film(titolo, regista, anno genere valutaz)`
- `IFilm(titolo)`: permette di eseguire selezioni di uguaglianza (o range se è ordinato e se sono significative per l'attributo) rispetto a `titolo`
 - da aggiornare ad ogni inserimento/cancellazione in `Film` e modifica di `titolo`
- `IFilm(regista)` ordinato: permette di eseguire selezioni di uguaglianza (o range se è ordinato e se sono significative per l'attributo) rispetto a `regista`
 - da aggiornare ad ogni inserimento/cancellazione in `Film` e modifica di `regista`
- `IFilm(titolo, regista)` ordinato: permette di eseguire selezioni di uguaglianza (o range se è ordinato e se sono significative per l'attributo) rispetto a `(titolo, regista)`
 - da aggiornare ad ogni inserimento/cancellazione in `Film` e modifica di `titolo o regista`
- `IFilm(regista, titolo)` ordinato: permette di eseguire selezioni di uguaglianza (o range se è ordinato e se sono significative per l'attributo) rispetto a `(regista, titolo)`
 - da aggiornare ad ogni inserimento/cancellazione in `Film` e modifica di `titolo o regista`
 - diverso da `IFilm(titolo, regista)`: se ordinato, il livello foglia è organizzato secondo un diverso ordinamento

SCELTA DEGLI INDICI: JOIN

- **Equijoin**: $R \text{ join } S \text{ on } R.A = S.B$, con S relazione inner
 - Indice (ordinato o hash) $I_S(B)$ permette al sistema di considerare l'index nested loop
 - Indici ordinati $I_R(A)$ e $I_S(B)$ clusterizzati sono utili per la strategia merge join
 - In assenza di indici, spesso il Sistema sceglie l'hash join (spesso più efficiente di nested loop)
- **Non equijoin**: $R \text{ join } S \text{ on } R.A \theta S.B$
 - Indice ordinato $I_S(B)$ permette al sistema di considerare l'index nested loop
 - Merge join e hash join non indicate (mai scelti dal Sistema)

TUNING DELLE INTERROGAZIONI

TUNING DELLE INTERROGAZIONI

- L'obiettivo è riscrivere una query contenuta nel carico di lavoro in modo da permettere al sistema di scegliere piani di esecuzione fisici più efficienti
- Il tuning a livello fisico e logico hanno side effect
 - Modifica dello schema fisico (es. aggiungiamo un indice)
 - Modifica dello schema logico
- Il tuning delle interrogazioni non comporta side-effect
 - Solo benefici
 - Prima attività da svolgere se le prestazioni di una interrogazione non sono soddisfacenti
- Perché non bastano gli ottimizzatori?
 - Lo spazio di ricerca dei possibili piani di esecuzione non viene analizzato in modo esaustivo
 - Il costo di ciascun piano di esecuzione viene solo stimato

TUNING DELLE INTERROGAZIONI: PRINCIPIO

- Riscrivi le query “troppo lente”
 - Troppi accessi a disco
 - Scansione totale di una tabella
 - Indici non utilizzati

TIPOLOGIE DI QUERY SU CUI FARE TUNING

- Condizioni su espressioni
- Uso di viste e viste materializzate
- Clausola DISTINCT
- Sottointerrogazioni

CONDIZIONI SU ESPRESSIONI

- Molti ottimizzatori non usano piani di esecuzione indicizzati in presenza di espressioni
- In questi casi, può convenire cercare di riscrivere l'interrogazione, aggiungendo filtri o modificando le condizioni ed eliminare le espressioni

CONDIZIONI SU ESPRESSIONI - ESEMPIO


```
SELECT * FROM Film  
WHERE 100*valutaz/5 >= 30;
```



$I_{\text{Film}}(\text{valutaz})$ potrebbe non poter essere utilizzato dal DBMS

```
SELECT * FROM Film  
WHERE salary >= 1,5;
```

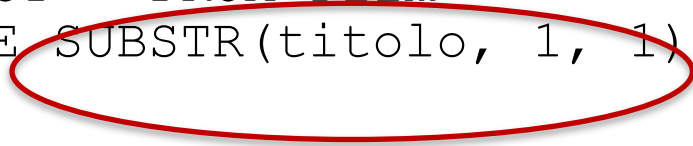
$I_{\text{Film}}(\text{valutaz})$ può essere utilizzato dal DBMS, se ritenuto utile



$I_{\text{Film}}(\text{valutaz})$ potrebbe non poter essere utilizzato dal DBMS

$I_{\text{Film}}(\text{valutaz})$ può essere utilizzato dal DBMS, se ritenuto utile

```
SELECT * FROM Film  
WHERE SUBSTR(titolo, 1, 1) = 'T';
```



```
SELECT * FROM Film  
WHERE titolo LIKE "T%"
```



VISTE

- L'uso delle viste semplifica la specifica delle query
- Le query su viste non sono mai più veloci di quelle senza uso di viste, usarle con prudenza

VISTE - ESEMPIO

Creazione vista

```
CREATE VIEW Noleggi_Commedie
AS
SELECT dataNol, codCli
FROM Noleggi NATURAL JOIN Video NATURAL JOIN Film
WHERE genere = 'commedia'
```

Interrogazione su vista

```
SELECT dataNol
FROM Noleggi_Commedie
WHERE codCli = 6635
```

Esecuzione: il sistema espande la vista ed esegue

```
SELECT dataNol
FROM Noleggi NATURAL JOIN Video NATURAL JOIN Film
WHERE genere = 'commedia' AND codCli = 6635
```

MA SE POTESSIMO SALVARE (=MATERIALIZZARE) IL RISULTATO DELLA VISTA SU DISCO?

Creazione vista

```
CREATE MATERIALIZED VIEW Noleggi_Commedie
AS
SELECT dataNol, codCli
FROM Noleggi NATURAL JOIN Video NATURAL JOIN Film
WHERE genere = 'commedia'
```

Interrogazione su vista

```
SELECT dataNol
FROM Noleggi_Commedie
WHERE codCli = 6635
```

Vantaggio durante la specifica e l'esecuzione delle interrogazioni

- nessuna espansione della query, l'interrogazione può essere direttamente eseguita sul contenuto materializzato della vista
- abbiamo **precalcolato il join** e abbiamo «salvato» il risultato nella vista materializzata → introduzione di un livello fisico anche per la vista

Il contenuto materializzato della vista deve essere mantenuto «allineato» con il contenuto delle tabelle di base

- se aggiungiamo un impiegato, il contenuto della vista cambia
- deve quindi cambiare la sua materializzazione su disco

VISTE MATERIALIZZATE

- Le viste materializzate (non presenti in tutti i sistemi) sono viste il cui risultato viene calcolato, memorizzato su disco e mantenuto aggiornato dal sistema

```
CREATE MATERIALIZED VIEW Name  
AS  
<Query>
```

VISTE MATERIALIZZATE NELLO STANDARD SQL

```
CREATE MATERIALIZED VIEW Name  
  [BUILD {IMMEDIATE | DEFERRED}]  
  [REFRESH {COMPLETE | FAST | NEVER}]  
  [ENABLE QUERY REWRITE]  
AS  
<Query>
```

BUILD: specifica quando il contenuto della vista deve essere determinato per la prima volta

- **IMMEDIATE**: calcola e memorizza i dati al momento della definizione della vista
- **DEFERRED**: il calcolo e la memorizzazione avviene al primo utilizzo

VISTE MATERIALIZZATE NELLO STANDARD SQL

```
CREATE MATERIALIZED VIEW Name  
[BUILD {IMMEDIATE | DEFERRED}]  
[REFRESH {COMPLETE | FAST | NEVER}]  
[ENABLE QUERY REWRITE]  
AS  
<Query>
```

REFRESH

- **COMPLETE**: ad ogni aggiornamento, si ricalcola completamente il contenuto della vista
- **FAST**: aggiornamento incrementale
- **NEVER**: nessun aggiornamento automatico

VISTE MATERIALIZZATE NELLO STANDARD SQL

```
CREATE MATERIALIZED VIEW Name  
  [BUILD {IMMEDIATE | DEFERRED}]  
  [REFRESH {COMPLETE | FAST | NEVER}]  
  [ENABLE QUERY REWRITE]  
AS  
<Query>
```

ENABLE QUERY REWRITE

L'ottimizzatore può riscrivere le interrogazioni utilizzando eventuali viste materializzate presenti nello schema esterno (v. esempio successivo)

ESEMPIO

```
CREATE MATERIALIZED VIEW Noleggi_Commedie
AS
SELECT dataNol, codCli
FROM Noleggi NATURAL JOIN Video NATURAL JOIN Film
WHERE genere = 'commedia'
```

Determinare i clienti che hanno noleggiato commedie

Non usando la vista materializzata Usando la vista materializzata

```
SELECT codCli
FROM Noleggi NATURAL JOIN
Video NATURAL JOIN Film
WHERE genere = 'commedia'
```

```
SELECT codCli
FROM Noleggi_Commedie
```


ESEMPIO

- Ma se l'utente non è consapevole dell'esistenza della vista materializzata `Noleggi_Commedie`?

- In questo caso possiamo solo scrivere

```
SELECT codCli  
FROM Noleggi NATURAL JOIN Video NATURAL JOIN Film  
WHERE genere = 'commedia'
```

- Il sistema è però sempre consapevole dell'esistenza delle viste e, se la vista è definita con la clausola `ENABLE QUERY REWRITE`, proverà però a riscrivere l'interrogazione usando `Noleggi_Commedie`

ESEMPIO

```
SELECT codCli  
FROM Noleggi NATURAL JOIN Video  
NATURAL JOIN Film  
WHERE genere = 'commedia'
```

Esiste nello schema (precedentemente creata)

```
CREATE MATERIALIZED VIEW Noleggi_Commedie  
AS  
SELECT dataNol, codCli  
FROM Noleggi NATURAL JOIN Video NATURAL JOIN Film  
WHERE genere = 'commedia'  
ENABLE QUERY REWRITE
```



```
SELECT codCli  
FROM Noleggi_Commedie
```

Questa query non richiede l'esecuzione di join, può essere quindi eseguita più efficientemente della query iniziale

VISTE MATERIALIZZATE IN POSTGRESQL

- Dalla versione 9.4, anche in PostgreSQL sono presenti le viste materializzate

```
CREATE MATERIALIZED VIEW Name  
AS <Query>
```

```
REFRESH MATERIALIZED VIEW Name;
```

- Rispetto allo standard, le viste materializzate in PostgreSQL vengono gestite con le seguenti opzioni:
 - BUILD: immediate
 - REFRESH: never (nessun aggiornamento automatico)
 - QUERY REWRITE: non attivato

DISTINCT

- La presenza della clausole `DISTINCT` comporta l'eliminazione dei duplicati dal risultato e quindi richiede una operazione di ordinamento (costosa)

Se non è necessario usare
`DISTINCT`, non usarlo

- Ma quando `DISTINCT` è necessario?
 - Esistono condizioni sufficienti per stabilirlo, non le vedremo nel dettaglio ma discuteremo il problema su alcuni esempi

DISTINCT - ESEMPIO

- Determinare i film di quentin tarantino

```
SELECT DISTINCT titolo, regista  
FROM Film  
WHERE regista = 'quentin tarantino'
```

- DISTINCT non è necessario
 - titolo, regista è chiave per Film, quindi è anche chiave per un sottoinsieme dei Film
- Interrogazione riscritta

```
SELECT titolo, regista  
FROM Film  
WHERE regista = 'quentin tarantino'
```

DISTINCT - ESEMPIO

- Determinare il codice dei video contenenti film con valutazione superiore a 3

```
SELECT DISTINCT colloc  
FROM Video NATURAL JOIN Film  
WHERE valutaz > 3
```

- DISTINCT non è necessario
 - colloc è chiave per Video
 - il join con Film non genera duplicati (per ogni video, esiste esattamente un film)

- Interrogazione riscritta

```
SELECT colloc  
FROM Video NATURAL JOIN Film  
WHERE valutaz > 3
```

DISTINCT - ESEMPIO

- Determinare il titolo e il regista dei film contenuti in almeno un video

```
SELECT DISTINCT titolo, regista  
FROM Video NATURAL JOIN Film
```

- DISTINCT è necessario
 - titolo, regista è chiave per Film
 - il join con Video genera duplicati (si associa un film a ciascuno dei video che lo contengono)
- Interrogazione non può essere riscritta

DISTINCT - ESEMPIO

- Determinare il titolo e il regista dei film contenuti in almeno un video, **insieme al codice dei video**

```
SELECT DISTINCT titolo, regista, colloc
FROM Video NATURAL JOIN Film
```

- DISTINCT **non è necessario**
 - titolo, regista è chiave per Film
 - il join con Video non genera duplicati: si associa un film a ciascuno dei video che lo contengono ma oltre a titolo, regista si restituisce anche colloc chiave per Video
 - quindi l'interrogazione non potrà generare duplicati
- Interrogazione può essere riscritta

```
SELECT titolo, regista, colloc
FROM Video NATURAL JOIN Film
```


SOTTOINTERROGAZIONI

- Durante la fase di ottimizzazione fisica, se possibile le interrogazioni che contengono sottointerrogazioni vengono trasformate in interrogazioni equivalenti senza sottointerrogazioni
- Se non e' possibile eliminare la sottointerrogazione (ad es. sottointerrogazioni scalari), la sottointerrogazione viene valutata e il valore ottenuto dalla sua interrogazione sostituito nell'interrogazione principale
- Le sottointerrogazioni correlate devono essere invece valutate in riferimento al valore della tupla candidata esaminata nell'interrogazione principale
 - L'ottimizzatore ha molti meno margini di ottimizzazione

SOTTOINTERROGAZIONI

- In generale, non tutti gli ottimizzatori trattano in maniera efficace le sottointerrogazioni (e sono in grado di riconoscere quando una interrogazione può essere riscritta senza utilizzo di sottointerrogazioni)
- La presenza di sottointerrogazioni e l'eventuale incapacità del sistema ad eliminarle tramite riscrittura può portare a non utilizzare indici esistenti nel piano fisico individuato

Quindi, se è possibile, è preferibile riscrivere una interrogazione con sottointerrogazioni in una equivalente ma senza sottointerrogazioni

SOTTO-INTERROGAZIONI

- La strategia di riscrittura dipende dal tipo di sotto-interrogazione
 - correlata/non correlata
 - scalare/non scalare
- Nel seguito: solo sotto-interrogazioni non correlate, tramite esempi

SOTTO-INTERROGAZIONI NON CORRELATE SCALARI - ESEMPIO

```
SELECT snum  
FROM Employee  
WHERE salary >  
      (SELECT AVG(salary) FROM Employee)
```

- Non problematica
- La maggior parte dei sistemi ottimizza indipendentemente query interna (che restituisce un singolo valore) e query esterna)
- Nessun problema di riscrittura

SOTTO-INTERROGAZIONI NON CORRELATE NON SCALARI - ESEMPIO

- Query con sottointerrogazione

```
SELECT titolo, regista  
FROM Film  
WHERE (titolo, regista) IN  
      (SELECT titolo, regista FROM Video)
```

- Può generare valutazione **inefficiente**
 - Per ogni Film si verifica se esiste almeno un video
 - $I_{\text{Video}}(\text{titolo}, \text{regista})$ se presente potrebbe non essere considerato

- Query equivalente con join

```
SELECT DISTINCT titolo, regista  
FROM Film NATURAL JOIN Video
```

- Valutazione **efficiente**
 - $I_{\text{Video}}(\text{titolo}, \text{regista})$ se presente viene preso in considerazione per piani con index nested loop join

- Attenzione:**

- la riscrittura può generare duplicati, quindi è necessario eliminarli (clausola **DISTINCT**) per garantire l'equivalenza con l'interrogazione di partenza
- in questi casi, non sempre l'interrogazione riscritta garantisce migliori prestazioni

SOTTO-INTERROGAZIONI NON CORRELATE NON SCALARI- STRATEGIA

- Combina gli argomenti delle due clausole di FROM
 - Metti in congiunzione le clausole di WHERE
 - Rimpiazza la condizione “`outer.attr1 IN (SELECT inner.attr2 ...)`” con “`outer.attr1 = inner.attr2`” nella clausola di WHERE
 - Mantieni la clausola SELECT della query esterna
-
- La strategia funziona per qualunque livello di annidamento ma potrebbe introdurre duplicati non restituiti dalla query iniziale
 - In questo caso è necessario aggiungere una nuova clausola DISTINCT che potrebbe limitare il beneficio della riscrittura