Parte III

11. Descrizione dei Piani di Esecuzione

Descriviamo i piani di esecuzione delle tre interrogazioni del carico di lavoro, prima e dopo il tuning fisico effettuato al fine di ottimizzarle.

Andiamo quindi ad analizzarle una alla volta;

1. Determinare l'identificatore dei giochi che coinvolgono al più quattro squadre e richiedono l'uso di due dadi.

   EXPLAIN ANALYZE SELECT CodGioco

   FROM Gioco

   WHERE maxSquadre = 4 AND dadiRichiesti = 2;

Osserviamo il piano di esecuzione **senza l'indice:**

```
"Seq Scan on gioco  (cost=0.00..232.47 rows=6758 width=4) (actual
time=0.025..3.719 rows=6761 loops=1)"

"  Filter: ((maxsquadre <= '4'::numeric) AND (dadirichiesti =
'2'::numeric))"

"  Rows Removed by Filter: 4"

"Planning time: 0.188 ms"

"Execution time: 4.111 ms"
```

Viene eseguita una semplice scansione sequenziale, filtrando rispetto alla condizione presente nel WHERE.

Creiamo quindi un indice multiattributo ordinato su maxSquadre e dadiRichiesti:

CREATE INDEX gioco_multi

ON Gioco(maxSquadre, dadiRichiesti);

Osserviamo il piano di esecuzione con l'indice:

```
"Index Scan using gioco_multi on gioco  (cost=0.28..8.30 rows=1 width=4)
(actual time=0.029..0.029 rows=1 loops=1)"

"  Index Cond: ((maxsquadre = '4'::numeric) AND (dadirichiesti =
'2'::numeric))"

"Planning time: 0.155 ms"

"Execution time: 0.041 ms"
```

Viene eseguita dal sistema una scansione mediante l'indice creato, che risulta molto efficiente; il tempo di esecuzione passa infatti da 4.111 ms a 0.041 ms, riducendosi del **99**%.

2. Determinare l'identificatore delle sfide relative a un gioco A di vostra scelta = OCA che, in alternativa:
   - hanno avuto luogo a gennaio 2021 e durata massima superiore a 2 ore, o
   - hanno avuto luogo a marzo 2021 e durata massima pari a 30 minuti.

   ```
   SELECT CodSfida
   FROM Sfida
   WHERE    codGioco = 'OCA' AND
            EXTRACT(YEAR FROM Sfida.dataOra) = 2021 AND
            ((EXTRACT(MONTH FROM Sfida.dataOra) = 1 AND Sfida.durataMax > 120)
            OR (EXTRACT(MONTH FROM Sfida.dataOra) = 3 AND Sfida.durataMax = 30));
   ```

Osserviamo il piano di esecuzione senza l'indice:

```
"Seq Scan on sfida  (cost=0.00..398.94 rows=1 width=5) (actual
time=0.041..9.092 rows=3972 loops=1)"
"  Filter: (((codgioco)::text = 'OCA'::text) AND (date_part('year'::text,
dataora) = '2021'::double precision) AND (((date_part('month'::text,
dataora) = '1'::double precision) AND (duratamax > 120)) OR
((date_part('month'::text, dataora) = '3'::double precision) AND
(duratamax = 30))))"
"  Rows Removed by Filter: 6026"
"Planning time: 0.739 ms"
"Execution time: 9.305 ms"
```

Creiamo degli indici su codGioco, dataOra e durataMax:

```
CREATE INDEX sfida_codGioco
ON Sfida(codGioco);
CLUSTER Sfida USING sfida_codGioco;

CREATE INDEX sfida_dataOra
ON Sfida(dataOra);
```

CREATE INDEX sfida_durataMax

ON Sfida(durataMax);

Osserviamo il piano di esecuzione con gli indici:

```
"Seq Scan on sfida  (cost=0.00..398.94 rows=1 width=5) (actual
time=0.044..9.440 rows=3972 loops=1)"

"  Filter: (((codgioco)::text = 'OCA'::text) AND (date_part('year'::text,
dataora) = '2021'::double precision) AND (((date_part('month'::text,
dataora) = '1'::double precision) AND (duratamax > 120)) OR
((date_part('month'::text, dataora) = '3'::double precision) AND
(duratamax = 30))))"

"  Rows Removed by Filter: 6026"

"Planning time: 1.603 ms"

"Execution time: 8.015 ms"
```

Purtroppo, in questo caso il sistema non ha considerato altri piani oltre la scansione sequenziale e quindi l'aggiunta di indici non ha portato a nessun miglioramento a livello prestazionale.

Se fossimo ancora in fase di progettazione, potremmo riconsiderare questa interrogazione per provare qualche altro piano di esecuzione che la renda più efficiente; se nuovamente questi tentativi non portassero a nessun risultato potremmo quindi considerare l'eliminazione di questi indici, in quanto essi occupano memoria e rendono più costoso l'aggiornamento.

3. Determinare le sfide, di durata massima superiore a 2 ore, dei giochi che richiedono almeno due dadi. Restituire sia l'identificatore della sfida sia l'identificatore del gioco.

   SELECT Sfida.CodSfida, Gioco.CodGioco

   FROM Dado NATURAL JOIN Gioco JOIN Sfida ON Gioco.codGioco = Sfida.codGioco

   WHERE Sfida.durataMax > 120

   GROUP BY Sfida.CodSfida, Gioco.CodGioco

   HAVING COUNT(Gioco.CodGioco) >= 2;

Creiamo due indici ordinati su Dado.codGioco e Gioco.codGioco per il **merge join**; per la selezione abbiamo già l'indice ad albero su durataMax creato nel punto precedente.

CREATE INDEX gioco_codGioco

ON Gioco(codGioco);

CLUSTER Gioco USING gioco_codGioco;

CREATE INDEX dado_codGioco

ON Dado(codGioco);

CLUSTER Dado USING dado_codGioco;

Confrontiamo i piani di esecuzione con e senza indice:

Prima della creazione degli indici:

```
"GroupAggregate  (cost=2400.15..2635.13 rows=11749 width=9) (actual time=59.850..70.356 rows=7937 loops=1)"
"  Group Key: sfida.codsfida, gioco.codgioco"
"  Filter: (count(gioco.codgioco) >= 2)"
"  Rows Removed by Filter: 1"
"  ->  Sort  (cost=2400.15..2429.52 rows=11749 width=9) (actual time=59.835..64.701 rows=15875 loops=1)"
"        Sort Key: sfida.codsfida, gioco.codgioco"
"        Sort Method: external merge  Disk: 304kB"
"        ->  Hash Join  (cost=751.94..1199.90 rows=11749 width=9) (actual time=27.933..48.663 rows=15875 loops=1)"
"              Hash Cond: ((dado.codgioco)::text = (gioco.codgioco)::text)"
"              ->  Seq Scan on dado  (cost=0.00..173.98 rows=9998 width=4) (actual time=0.013..1.379 rows=9998 loops=1)"
"              ->  Hash  (cost=613.56..613.56 rows=7950 width=13) (actual time=12.635..12.635 rows=7938 loops=1)"
"                    Buckets: 2048 (originally 2048)  Batches: 32 (originally 8)  Memory Usage: 349kB"
"                    ->  Merge Join  (cost=27.55..613.56 rows=7950 width=13) (actual time=0.548..6.082 rows=7938 loops=1)"
"                          Merge Cond: ((gioco.codgioco)::text = (sfida.codgioco)::text)"
"                          ->  Index Only Scan using gioco_pkey on gioco  (cost=0.28..185.76 rows=6765 width=4) (actual time=0.135..0.648 rows=4615 loops=1)"
"                                Heap Fetches: 0"
"                          ->  Index Scan using sfida_codgioco on sfida  (cost=0.29..372.25 rows=7950 width=9) (actual time=0.009..2.600 rows=7938 loops=1)"
"                                Filter: (duratamax > 120)"
"                                Rows Removed by Filter: 2060"
"Planning time: 1.293 ms"
"Execution time: 71.519 ms"
```

Dopo la creazione degli indici:

```
"GroupAggregate  (cost=2214.99..2449.97 rows=11749 width=9) (actual time=29.337..36.820 rows=7937 loops=1)"
"  Group Key: sfida.codsfida, gioco.codgioco"
"  Filter: (count(gioco.codgioco) >= 2)"
"  Rows Removed by Filter: 1"
"  ->  Sort  (cost=2214.99..2244.37 rows=11749 width=9) (actual time=29.323..31.278 rows=15875 loops=1)"
"        Sort Key: sfida.codsfida, gioco.codgioco"
"        Sort Method: external merge  Disk: 296kB"
"        ->  Merge Join  (cost=86.94..1014.75 rows=11749 width=9) (actual time=1.956..19.057 rows=15875 loops=1)"
"              Merge Cond: ((sfida.codgioco)::text = (dado.codgioco)::text)"
"              ->  Index Scan using sfida_codgioco on sfida  (cost=0.29..372.25 rows=7950 width=9) (actual time=0.009..2.918 rows=7938 loops=1)"
"                    Filter: (duratamax > 120)"
"                    Rows Removed by Filter: 2060"
"              ->  Materialize  (cost=0.57..622.90 rows=9998 width=8) (actual time=0.022..7.466 rows=22692 loops=1)"
"                    ->  Merge Join  (cost=0.57..597.90 rows=9998 width=8) (actual time=0.019..5.404 rows=6822 loops=1)"
"                          Merge Cond: ((gioco.codgioco)::text = (dado.codgioco)::text)"
"                          ->  Index Only Scan using gioco_codgioco on gioco  (cost=0.28..185.76 rows=6765 width=4) (actual time=0.008..0.546 rows=4615 loops=1)"
"                                Heap Fetches: 0"
"                          ->  Index Only Scan using dado_codgioco on dado  (cost=0.29..270.26 rows=9998 width=4) (actual time=0.006..0.824 rows=6822 loops=1)"
"                                Heap Fetches: 0"
"Planning time: 1.830 ms"
"Execution time: 38.144 ms"
```

Notiamo come nel primo caso venga eseguito un **Hash Join** mentre nel secondo esso venga sostituito da un **Merge Join**.

Questo rende l'interrogazione più efficiente, infatti il tempo di esecuzione passa da **71.519 ms** a **38.144 ms**, con un miglioramento del **46.67**%.