Progetto Finale (12 CFU)
Online Challenge Activity

Gruppo 18 (Enrico Pezzano, Daniele Scala)

Parte 3 - Deliverable

Piani di Esecuzione scelti secondo i principi del Tuning Fisico

Per quanto riguarda il tuning fisico, abbiamo deciso di ottimizzare le query ed il carico di lavoro (ove possibile) in modo tale che le operazioni costose fossero ridotte, limitando quindi join, clausole DISTINCT e sotto-interrogazioni che non permettono al sistema di scegliere piani di esecuzione fisici efficienti.

QUERY MENO FREQUENTI

Le interrogazioni non incluse nel carico di lavoro presentano invece clausole e collegamenti costosi, che ci hanno portato alla creazione di una vista materializzata (per le prime due), in quanto essa ci permette di eseguire le interrogazioni direttamente sul proprio contenuto, anche in seguito ad eventuali modifiche.

Quindi abbiamo:

```
CREATE MATERIALIZED VIEW caselle_con_task AS
select distinct gioco
from casella_percorso
right join task on task.casella = casella_percorso.idCas;
```

QUERY A:

```
select distinct gioco
from casella_percorso
right join task on task.casella = casella_percorso.idCas;

--Execution Time:      0.074 (query su tabella)
--Numero nodi: 5

select gioco
from caselle_con_task;

--Execution Time:      0.023 (query su vista materializzata)
--numero nodi: 1
```

Purtroppo non è possibile rimuovere la clausola distinct dalla query, in quanto stiamo selezionando gli elementi di un attributo che non è unico e potremmo quindi avere molte ripetizioni superflue.

QUERY B

```
select idG from gioco
except
select distinct gioco
from casella_percorso
right join task on task.casella = casella_percorso.idCas;

--Execution Time:      9.818 (query su tabella)
--Numero nodi: 10

select idG from gioco
except
select gioco from caselle_con_task;
```

--Execution Time: 7.991(query su vista materializzata)
--Numero nodi: 6

Abbiamo lo stesso schema dell'interrogazione precedente, ma in aggiunta abbiamo l'except che rende l'operazione più costosa e lunga. Come possiamo notare, l'utilizzo della vista materializzata può essere utile anche per questa interrogazione, ma i tempi di esecuzione variano di poche unità (sebbene l'analisi di questo piano fisico coinvolge 6 nodi, mentre l'interrogazione senza utilizzo della vista coinvolge 10 nodi).

Nota bene: non essendo richiesto di eseguire inserimenti massivi su tabelle coinvolte nelle precedenti query non abbiamo testato precisamente l'efficacia eventuali indici per le prime due interrogazioni, in quanto abbiamo ritenuto opportuno evitare di crearne su tabelle piccole.

QUERY C

```
select ids
from sfida s
where s.durata > (select avg(sfida.durata)
                  from sfida
                  where s.gioco=sfida.gioco
                  group by sfida.gioco);
```

Per quanto riguarda quest'ultima interrogazione, possiamo subito notare che il suo costo è compromesso dalla presenza di una subquery. Quest'ultima però non è possibile rimuoverla, bensì essendo scalare viene valutata, ed il valore ottenuto dall'interrogazione viene poi inserito al suo posto nella query principale.

In quest'ultima query, avendo a disposizione molte tuple della tabella sfida, abbiamo sfruttato la possibilità di verificare che, con l'utilizzo di indici, il piano fisico scelto è più efficace, difatti l'uso di un indice ordinato non clusterizzato ha ridotto di molto il tempo di esecuzione della query. Si può anche utilizzare un indice hash ma è meno efficiente:

--Execution Time: 15165.659 senza indici
--Numero nodi: 3

```
create index I_sfida_g_hash on sfida using hash(gioco);
```

--Execution Time: 1293.158 con indice hash
--numero nodi: 3
```
drop index I_sfida_g_hash;
```

create index I_sfida_g on sfida(gioco);

--Execution Time: 948.473 con indice ordinato
--numero nodi: 3

Carico di Lavoro

Per quanto riguarda il carico di lavoro, abbiamo cercato di ottimizzare al meglio le query evitando di inserire operazioni costose ed utilizzando indici solamente dove opportuno:

Q1

```
select dado.gioco
from gioco
join dado on dado.gioco=gioco.idG
where Max_n_Squadre<=4
group by dado.gioco having count(dado.gioco) >= 2;
```

--Execution Time: 3.487 (senza utilizzo di indici)
--Numero nodi: 5

Abbiamo provato ad utilizzare diversi indici, ma abbiamo riscontrato leggerissimi, se non nulli, cambiamenti circa il numero di nodi dello schema fisico e il tempo di esecuzione.
L'unico indice che abbiamo tenuto in considerazione è un indice ordinato non clusterizzato sull'attributo max_n_squadre della tabella gioco.

```
create index I_n_squadre on gioco(Max_n_Squadre);
```

--Execution Time: 1.789 (con l'utilizzo dell'indice)
--Numero nodi: 5

Q2

```
SELECT idS
from sfida
where sfida.gioco = 111
and (((EXTRACT(MONTH FROM data_ora) = 01) and (EXTRACT(YEAR FROM data_ora) = 2021)
and durata_max > '02:00:00')
or ((EXTRACT(MONTH FROM data_ora) = 03) and (EXTRACT(YEAR FROM data_ora) = 2021) and
durata_max = '00:30:00'));
```

questa query è molto meno costosa delle precedenti, in quanto, selezionando solo le partite di un unico gioco, abbiamo rimosso la quasi totalità delle tuple da filtrare. Per agevolare la selezione, il DBMS sfrutta l'indice I_sfida_g creato precedentemente per la query c. Già in questo modo il tempo di esecuzione è molto basso.

--Execution Time: 0.093 (con l'uso dell'indice I_sfida_g)
--Numero nodi: 1

Nota bene: in assenza dell'indice *I_sfida_g* il sistema utilizza un indice implicito (*sfida_data_ora_gioco_key*) che determina anch'esso un piano fisico ottimo.

Q3

```
select idS, dado.gioco
from sfida
join gioco on sfida.gioco = gioco.idG
join dado on gioco.idG = dado.gioco
where durata_max > '02:00:00'
group by idS, dado.gioco
having count(dado.gioco) >= 2;
```

Abbiamo provato ad implementare diversi indici che limitassero gli scan e che quindi agissero sulle condizioni di ricerca, però abbiamo notato che il piano fisico ottimale non considerava opportuno il loro utilizzo
--Execution Time: 12.075 (dopo implementazione indici)
--Numero nodi: 8

abbiamo deciso quindi di creare nuovamente una vista materializzata, in grado di limitare i costi in termini di accesso su più tabelle dettati dai join:

```
CREATE MATERIALIZED VIEW sfide_2h_2dadi AS
select idS, dado.gioco
from sfida
join gioco on sfida.gioco = gioco.idG
join dado on gioco.idG = dado.gioco
where durata_max > '02:00:00'
group by idS, dado.gioco
having count(dado.gioco) >= 2;

select *
from sfide_2h_2dadi;
```

--Execution Time: 0.052(query su vista materializzata)
--Numero nodi: 1

Descrizione della Politica di controllo

Per la politica di controllo degli accessi abbiamo scelto di definire i seguenti ruoli: *utente*, *giocatore*, *gameadmin* e *gamecreator*, individuando la relativa gerarchia.

L'*utente* è colui che è registrato nella base di dati, ma non gioca partite, perciò riteniamo che sia utile permettergli solo di vedere gli altri utenti registrati, le squadre (per potervisi eventualmente unire) ed i giochi disponibili.

Un *giocatore* può essere in grado di vedere gli altri utenti registrati, le squadre, le sfide ed il loro relativo procedimento.
Inoltre deve poter inserire le risposte dei quiz e dei task (e quindi anche vederne i testi); nota bene che non concediamo la selezione su *casella_podio*, poiché il giocatore può vedere la situazione della classifica (circa le sfide che gli interessano) attraverso attributo podio delle squadre.

Il *gameadmin* è colui che è in grado di attivare le sfide, quindi di monitorarle e gestirle, ed approva i task, pertanto c'è bisogno che egli sia in grado di modificare anch'essi;
Inoltre è necessario che sia in grado di visualizzare lo schema per intero, in modo da tener monitorate le varie sfide e tutti gli elementi che interagiscono con esse.

Il *gamecreator* crea i giochi, pertanto, su di loro e su ogni cosa che ne deriva, egli dev'essere in grado di poter vedere modificare ciò che ritiene opportuno; inoltre ha la possibilità di cedere privilegi ad altri ruoli in base a determinate esigenze.

ALCUNE NOTE FINALI:
Abbiamo preferito mantenere esplicitamente la gerarchia:
gamecreator > *gameadmin* > *giocatore* > *utente*.

Solo il *gamecreator* ha la possibilità di cedere ulteriori privilegi agli altri ruoli, in quanto il *gameadmin* gestisce le sfide, ma non gli è permesso cedere i suoi privilegi a giocatori o utenti senza autorizzazione del *gamecreator*.

Codici SQL

Nell'archivio della consegna è allegato anche lo script SQL richiesto:

- "utenti e ruoli.sql", contenente l'implementazione della politica di controllo dell'accesso.