BD - Progetto finale Tuning

Team 13

Descrizione piani di esecuzione

Query 2a

Per la prima operazione di manipolazione il sistema come prima operazione per il piano di esecuzione della query inizia con il join Caselle su Task e sceglie correttamente un join di tipo hash, al passaggio sucessivo il sistema procede con un hash inner join dovuto alla clausola del where dove confrontiamo l'attriuto Caselle.IdCasdelle con Task.IdCaselle, per poi procedere con l'aggregate nel distinct e ottenere il risultato finale.

```
SELECT DISTINCT IdGioco
FROM Caselle NATURAL JOIN Task
WHERE Task.IdCaselle = Caselle.IdCaselle;
```

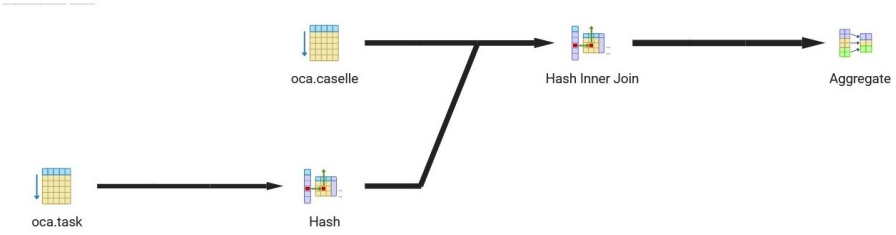

Figure 1: Query1 schema

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Aggregate (cost=188.77..189.57 rows=80 width=11) (actual=1.407..1.41...	0.156 ms	1.41 ms	↑ 1.78	45	80	1
2.	→ Hash Inner Join (cost=41.69..186.36 rows=964 width=11) (actual=... **Hash Cond**: ((caselle.idcaselle)::text = (task.idcaselle)::text)	0.647 ms	1.254 ms	↑ 1	964	964	1
3.	→ Seq Scan on oca.caselle as caselle (cost=0..116.2 rows=5020 ...	0.349 ms	0.349 ms	↑ 1	5020	5020	1
4.	→ Hash (cost=29.64..29.64 rows=964 width=11) (actual=0.258..... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 50 kB	0.133 ms	0.258 ms	↑ 1	964	964	1
5.	→ Seq Scan on oca.task as task (cost=0..29.64 rows=964 wi...	0.125 ms	0.125 ms	↑ 1	964	964	1

Figure 2: Query1 costi

Nella fase di tuning delle interrogazioni e‘ stato scelto di utililizzare una materialized view contenente sia il JOIN che la clausola distinct in quanto entrambe sono causa del forte rallentamento che subisce la query. Si e‘ scelto di mantenere il DISTINCT all'interno della vista materializzata in quanto esso si rivela necessario sia nella prima che nella seconda query. Quindi la nostra interrogazione e‘ diventata una select sulla vista. Ovviamente i tempi si sono ridotti nella query, poiche le operazioni richieste non sono fatte direttamente da lei, e il piano di esecuzione viene semplificato ad una sola operazione.

```
CREATE MATERIALIZED VIEW Caselle_Task
AS SELECT DISTINCT IdGioco
FROM Caselle NATURAL JOIN Task
WHERE Task.IdCaselle = Caselle.IdCaselle;

SELECT IdGioco
FROM Caselle_Task;
```

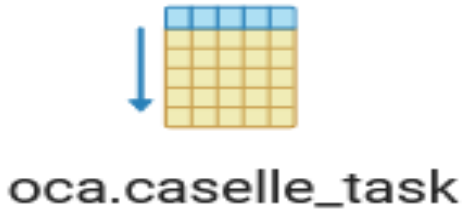

Figure 3: Query1 Tuning schema

			Timings		Rows			
	#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
	1.	→ Seq Scan on oca.caselle_task as caselle_task (cost=0..19.4 rows=940 wi...	0.016 ms	0.016 ms	↑ 20.89	45	940	1

Figure 4: Query1 Tuning costi

Query 2b

La query dopo l'EXCEPT utillizza lo stesso piano di esecuzione della query spiegata al punto precedente prima del tuning. Entrambe le query procedono con una subquery scan che scansiona sequenzialmente le tabelle, e passano ad un append necessario per l'operazione di EXCEPT sucessiva. I tempi rispetto a prima sono ulteriormente rallentati a causa della presenza di EXCEPT.

```
SELECT IdGioco
FROM Gioco
EXCEPT
SELECT DISTINCT IdGioco
FROM Caselle NATURAL JOIN Task
WHERE Task.IdCaselle = Caselle.IdCaselle;
```

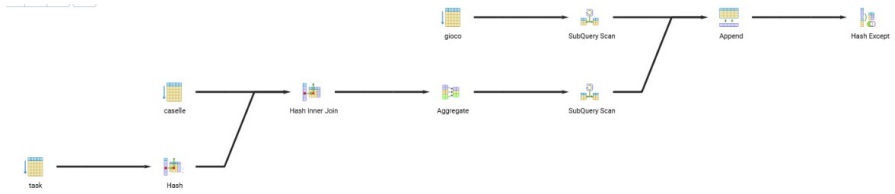

Figure 5: Query2 schema

#	Node	Timings		Rows		
		Exclusive	Inclusive	Rows X	Actual	Plan
1.	→ Hash Except (cost=0..369.01 rows=3620 width=62) (actual=5.368..5.7...	1.497 ms	5.735 ms	↑ 1.02	3575	3620
2.	→ Append (cost=0..359.76 rows=3700 width=62) (actual=0.013..4.2...	0.274 ms	4.239 ms	↑ 1.01	3665	3700
3.	→ Subquery Scan (cost=0..169.4 rows=3620 width=15) (actual...	0.444 ms	1.184 ms	↑ 1	3620	3620
4.	→ Seq Scan on oca.gioco as gioco (cost=0..133.2 rows=3...	0.74 ms	0.74 ms	↑ 1	3620	3620
5.	→ Subquery Scan (cost=188.77..190.37 rows=80 width=15) (ac...	0.008 ms	2.781 ms	↑ 1.78	45	80
6.	→ Aggregate (cost=188.77..189.57 rows=80 width=11) (a...	0.225 ms	2.774 ms	↑ 1.78	45	80
7.	→ Hash Inner Join (cost=41.69..186.36 rows=964 wi... **Hash Cond**: ((caselle.idcaselle)::text = (task.idcaselle)::text)	1.3 ms	2.55 ms	↑ 1	964	964
8.	→ Seq Scan on oca.caselle as caselle (cost=0..1...	0.583 ms	0.583 ms	↑ 1	5020	5020
9.	→ Hash (cost=29.64..29.64 rows=964 width=11)... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 50 kB	0.451 ms	0.667 ms	↑ 1	964	964
10.	→ Seq Scan on oca.task as task (cost=0..2...	0.217 ms	0.217 ms	↑ 1	964	964

Figure 6: Query2 costi

Utilizzando la vista creata precedentemente, abbassiamo di molto il tempo, e lo schema che effettua la nostra interrogazione rimane, rispetto a prima, solo il procedimento dell'except. Questo e' ottimale dal momento che entrambe le query utilizzano la stessa vista materializzata e non e' necessario crearne una nuova. Questo riduce il carico di lavoro da parte del sistema che deve solamente mantenere una singola vista materializzata per due interrogazioni.

```
SELECT IdGioco
FROM Gioco
EXCEPT
SELECT IdGioco
FROM Caselle_Task
```

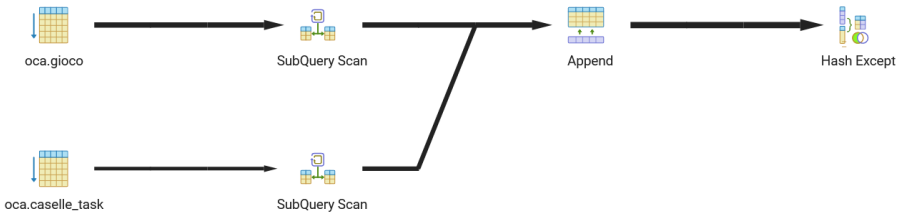

Figure 7: Query2 schema

#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Hash Except (cost=0..209.6 rows=3620 width=62) (actual=1.543..1.727 ...	0.874 ms	1.727 ms	↑ 1.02	3575	3620	1
2.	→ Append (cost=0..198.2 rows=4560 width=62) (actual=0.013..0.854 ...	0.18 ms	0.854 ms	↑ 1.25	3665	4560	1
3.	→ Subquery Scan (cost=0..169.4 rows=3620 width=15) (actual=0...	0.275 ms	0.66 ms	↑ 1	3620	3620	1
4.	→ Seq Scan on oca.gioco as gioco (cost=0..133.2 rows=362...	0.385 ms	0.385 ms	↑ 1	3620	3620	1
5.	→ Subquery Scan (cost=0..28.8 rows=940 width=62) (actual=0.0...	0.004 ms	0.014 ms	↑ 20.89	45	940	1
6.	→ Seq Scan on oca.caselle_task as caselle_task (cost=0..19...	0.011 ms	0.011 ms	↑ 20.89	45	940	1

Figure 8: Query2 costi

Query 2c

In questo caso si procede con la costruzione di una subplan che appartiene alla elaborazione della tabella per il valore medio di durata delle sfide, si procede con aggregate per l'AVG e si passsa con un altro aggregate per HAVING.

```
SELECT IdSfida, DurataMax, IdGioco
FROM Sfida S
GROUP BY IdSfida
HAVING DurataMax >= (SELECT AVG(DurataMax)
                     FROM Sfida
                     WHERE Sfida.IdGioco = S.IdGioco);
```

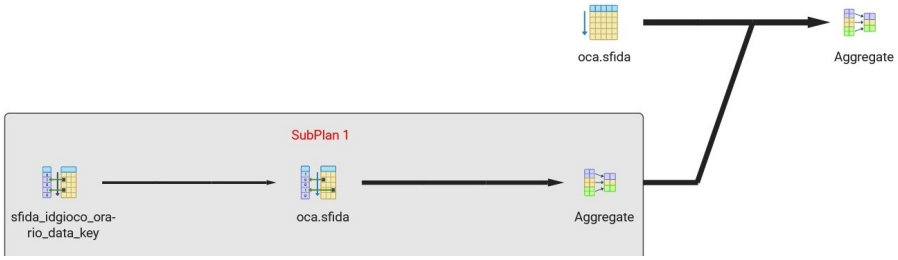

Figure 9: Query3 schema

#	Node	Timings		Rows	
		Exclusive	Inclusive	Rows X	Actual
1.	→ Aggregate (cost=193.72..243.91 rows=5018 width=30) (actual=3.40... **Filter**: ((s.duratamax)::interval >= (SubPlan 1)) **Rows Removed by Filter**: 2398	10.488 ms	513.035 ms	↑ 1.92	2620
2.	→ Seq Scan on oca.sfida as s (cost=0..181.18 rows=5018 width=3...	0.747 ms	0.747 ms	↑ 1	5018
3.	→ Aggregate (cost=33.44..33.45 rows=1 width=16) (actual=0.1..0....	230.829 ms	501.8 ms	↑ 1	1
4.	→ Bitmap Heap Scan on oca.sfida as sfida (cost=4.35..33.39 ... **Recheck Cond**: ((sfida.idgioco)::text = (s.idgioco)::text) **Heap Blocks**: exact=236048	180.648 ms	270.972 ms	↓ 14	126
5.	→ Bitmap Index Scan using sfida_idgioco_orario_data_k... **Index Cond**: ((sfida.idgioco)::text = (s.idgioco)::text)	90.324 ms	90.324 ms	↓ 14	126

Figure 10: Query3 costi

Nella fase tuning abbiamo valutato, essendo questa un interrogazione scalare, di non poterla ottimizzare eliminandola. Visto che questa sottointerrogazione e' anche correlata, abbiamo tentato con degli indici di ottimizzarla ma non abbiamo ottenuto un effettivo miglioramento dei tempi di elaborazione. Pertanto abbiamo deciso di non modificarla.

Workload 1 - 2

Abbiamo scelto di non ottimizzare ulteriormente le prime due query dal momento che gli indici creati durante la fase di creazione del progetto fisico sono gia‘ ottimali.

Workload 3

Come possiamo osservare dal piano di esecuzione scelto dal sistema la nostra query effettua prima un join di tipo hash tra presente e sfida sull'attributo idGioco di entrambe le relazioni, successivamente confronta tramite hash inner join la clausola WHERE e infine aggregate valuta la clausola DISTINCT. Osserviamo dalle immagini che la query e‘ molto costosa.

```
SELECT DISTINCT IdSfida, Sfida.IdGioco
FROM Sfida JOIN Presente ON Sfida.IdGioco = Presente.IdGioco
WHERE Presente.IdGioco=Sfida.IdGioco
AND Presente.NumDadi >=2 AND DurataMax > '02:00:00';
```

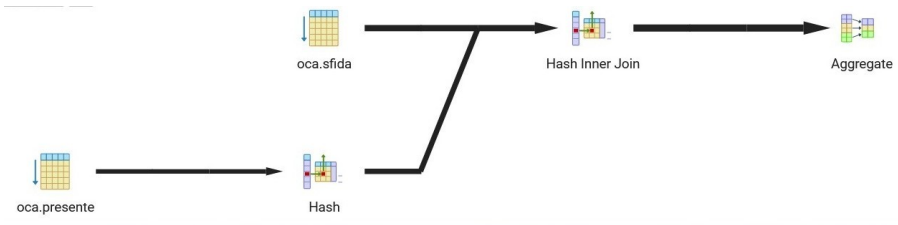

Figure 11: Workload 3 schema

#	Node	Timings		Rows		
		Exclusive	Inclusive	Rows X	Actual	Plan
1.	→ Aggregate (cost=1172.04..1213.18 rows=4114 width=22) (actual=21....	13.58 ms	22.156 ms	↑ 1.21	3401	4114
2.	→ Hash Inner Join (cost=55.2..905.48 rows=53313 width=22) (actu... **Hash Cond**: ((sfida.idgioco)::text = (presente.idgioco)::text)	7.162 ms	8.576 ms	↑ 1.03	52125	53313
3.	→ Seq Scan on oca.sfida as sfida (cost=0..193.72 rows=4114 ... **Filter**: (sfida.duratamax > '02:00:00'::time without time zone) **Rows Removed by Filter**: 904	1.058 ms	1.058 ms	↑ 1	4114	4114
4.	→ Hash (cost=43.23..43.23 rows=958 width=11) (actual=0.357... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 49 kB	0.15 ms	0.357 ms	↑ 1	958	958
5.	→ Seq Scan on oca.presente as presente (cost=0..43.23 r... **Filter**: (presente.numdadi >= '2'::numeric) **Rows Removed by Filter**: 100	0.207 ms	0.207 ms	↑ 1	958	958

Figure 12: Workload 3 costi

Abbiamo scelto di ottimizzare la seguente query attraverso l'utilizzo di una vista meterializzata poiche‘ con il solo uso degli indici creati durante la fase di progettazione fisica, i tempi di esecuzione del piano scelto dal sistema non venivano ottimizzati abbastanza. Possiamo notare dalle immagini che attraverso l'utilizzo della vista materializzata i tempi si riducono notevolmente. E il piano di esecuzione viene ovviamente semplificato.

```
CREATE MATERIALIZED VIEW SfidaPresente AS
SELECT DISTINCT IdSfida, Sfida.IdGioco
FROM Sfida JOIN Presente ON Sfida.IdGioco = Presente.IdGioco
WHERE Presente.IdGioco=Sfida.IdGioco AND Presente.NumDadi >=2 AND DurataMax > '02:00:00';

SELECT IdSfida, IdGioco
FROM SfidaPresente;
```

Figure 13: Workload 3 schema

		Timings		Rows			
#	Node	Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Seq Scan on oca.sfidapresente as sfidapresente (cost=0..58.01 rows=3...	0.221 ms	0.221 ms	↑ 1	3401	3401	1

Figure 14: Workload 3 costi

Abbiamo deciso di fare i piani di esecuzione per le query del workload e per le altre controllando i risultati prima e dopo l'applicazione del tuning.

BD - Scelte Ruoli

Team 13

Ruoli

Tabella dei ruoli

	GameAdmin	GameCreator	Giocatore	Utente
Caselle	ALL	ALL	S	
Composta	ALL	S	S	
Contiene	ALL	ALL	S	
Dadi	ALL	ALL	S	
Gioco	ALL	ALL	S	S
Icone	ALL	ALL	S	
Lancio	ALL	S	SI	
Posizione	ALL	S	S	
Presente	ALL	ALL	S	
Quiz	ALL	ALL	S	
RispostaData	ALL	S	SI	
RisposteQuiz	ALL	ALL	S	
Sfida	ALL	S	S	S
Squadra	ALL	S	S	
Task	ALL	ALL	S	
Turno	ALL	S	S	
Utente	ALL	S	SI	S

Scelta dei ruoli e gerarchia

La nostra gerarchia si articola con il gameadmin che e‘ il proprietario assoluto del gioco e contiene al suo interno tutti gli altri ruoli, per questo abbiamo deciso di assegnarli tutti i privilegi su tutta la base di dati.
Il game creator e‘ colui che crea il gioco e quindi ha il privilegio di insert update e delete solo sulle parti strutturali del gioco come ad esempio la creazione delle icone, delle caselle e dei quiz associati; mentre invece ha il privilegio di select su tutto per poter controllare come viene utilizzato effettivamente un gioco e poter cosi creare dei giochi piu‘ accattivanti.
Il giocatore invece ha il permesso di select su tutto per poter avere una visione complessiva del gioco e ha il privilegio di inserimento solo sulle risposte sul lancio e sull'utente poiche‘ le prime due sono necessarie per un corretto svolgimento del gioco mentre l'ultima per potersi iscrivere.
Non ha i privilegi di delete e update perche‘ potrebbe creare confusione all'interno della base di dati non essendone il proprietario.
L'utente infine ha soltanto il privilegio di lettura per quanto riguarda i giochi le sfide e gli utenti, cosi da poter scegliere a quale gioco giocare e a quale sfida partecipare.
Il gameadmin nel nostro caso non si occupa solamente della convalida dei task e dell'attivazione delle sfide, ma anche della creazione delle squadre e dell'eliminazione di quest'ultime e degli utenti nel caso violassero alcune regole del gioco.