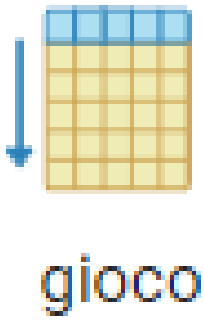
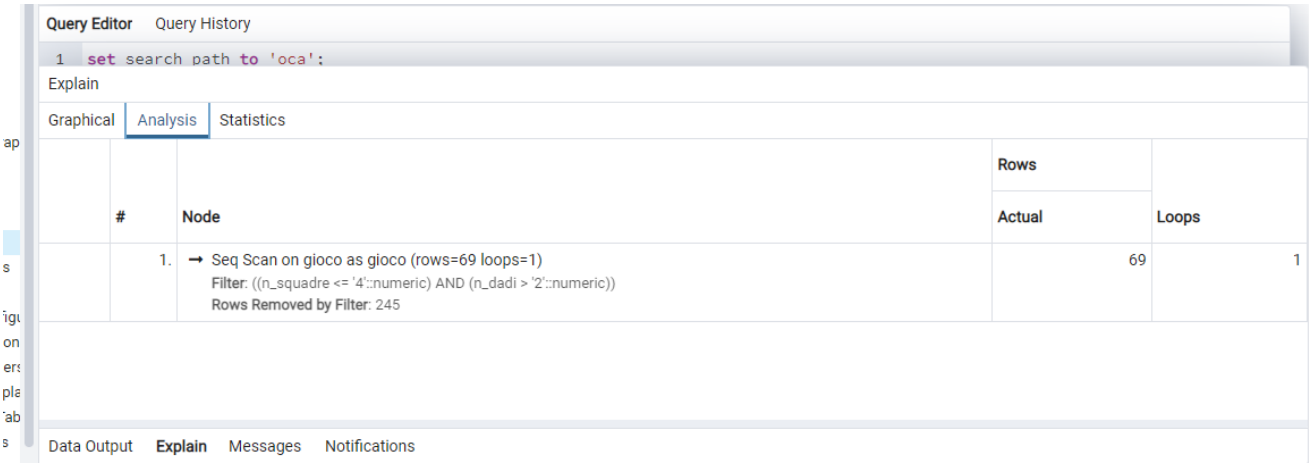
# PIANI DI ESECUZIONE

PRIMA QUERY

```
SELECT id_g
FROM GIOCO
WHERE n_squadre <= 4 AND n_dadi > 2;
```

Utilizzando il comando analyze su Pgadmin possiamo vedere come chiaramente l'unica relazione utilizzata è GIOCO e come l'unico cammino di accesso individuato è un SEQ_SCAN.




Questo approccio è piuttosto costoso quindi come abbiamo visto nella parte 1 con il tuning aggiungiamo due indici.

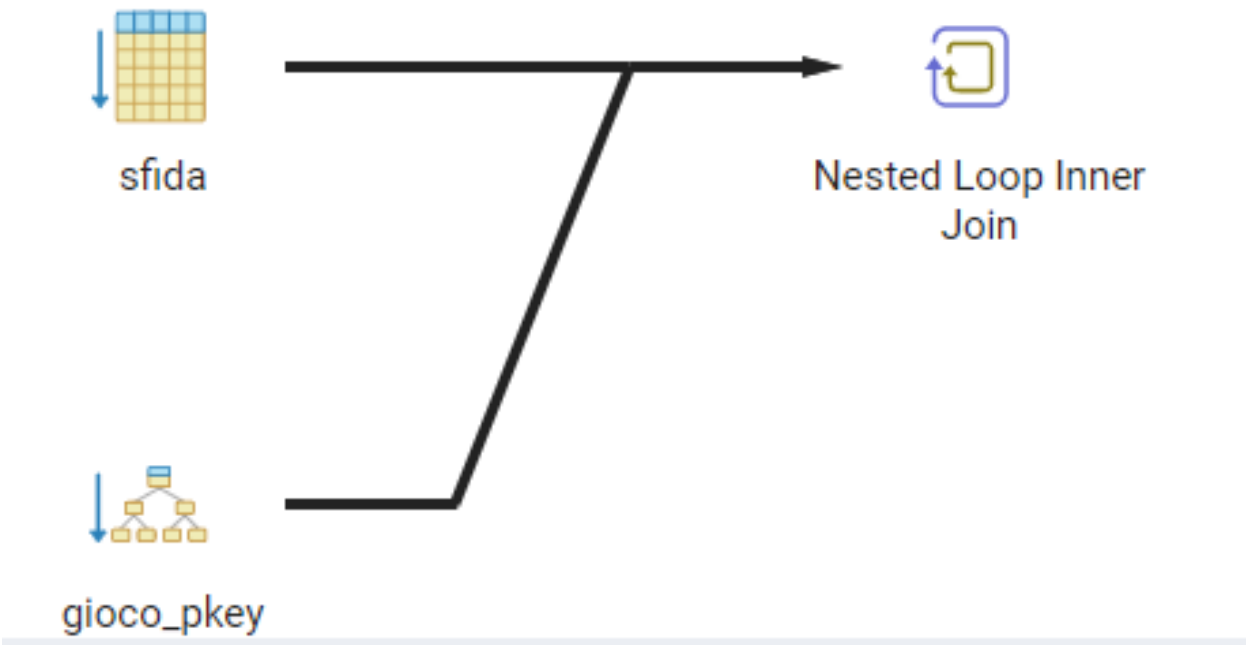
SECONDA QUERY

La nostra seconda query del workload è la seguente:

```
SELECT cod_Sfida
FROM SFIDA NATURAL JOIN GIOCO
WHERE SFIDA.id_g = 'ab0'
AND ((Sfida.data BETWEEN '01/01/2021' AND '31/01/2021'
AND SFIDA.durata_max > '02:00:00')
OR
(SFIDA.data BETWEEN '01/03/2021' AND '31/03/2021'
AND SFIDA.durata_max = '00:30:00'));
```

In questo caso si può già notare come le operazione sono aumentate molto.

Analizzando anche questa query il risultato è:




| | Node | Rows | |
|---|---|---|---|
| | | Actual | Loops |
| 1. | → Nested Loop Inner Join (rows=0 loops=1) | 0 | 1 |
| 2. | → Seq Scan on sfida as sfida (rows=0 loops=1)<br>**Filter**: (((id_g)::text = 'ab01'::text) AND (((data >= '2020-01-01'::date) AND (data <= '2020-01-31'::date) AND (durata_max > '02:00:00'::time without time zone)) OR ((data >= '2020-01-03'::date) AND (data <= '2020-03-31'::date) AND (durata_max = '00:30:00'::time without time zone))))<br>**Rows Removed by Filter**: 312 | 0 | 1 |
| 3. | → Index Only Scan using gioco_pkey on gioco as gioco (rows=0 loops=0)<br>**Index Cond**: (id_g = 'ab01'::text) | 0 | 0 |

Come possiamo vedere qua è stato utilizzato una seq_scan su sfida assieme ad un indice sulla chiave di gioco per fare un nested loop join.

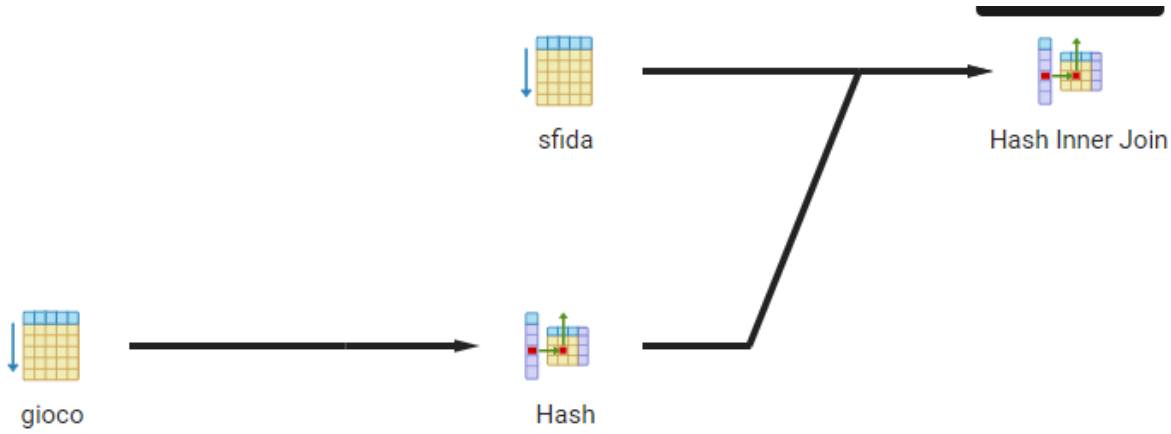
TERZA INTERROGAZIONE:

La terza interrogazione è :

SELECT id_g , cod_Sfida

FROM GIOCO NATURAL JOIN SFIDA

WHERE GIOCO.n_dadi > 2 AND SFIDA.durata_max > '02:00:00';




| # | Node | Rows | |
|---|---|---|---|
| | | Actual | Loops |
| 1. | → Hash Inner Join (rows=221 loops=1)<br>**Hash Cond**: ((sfida.id_g)::text = (gioco.id_g)::text) | 221 | 1 |
| 2. | → Seq Scan on sfida as sfida (rows=241 loops=1)<br>**Filter**: (durata_max > '02:00:00'::time without time zone)<br>**Rows Removed by Filter**: 71 | 241 | 1 |
| 3. | → Hash (rows=280 loops=1)<br>**Buckets**: 1024 **Batches**: 1 **Memory Usage**: 20 kB | 280 | 1 |
| 4. | → Seq Scan on gioco as gioco (rows=280 loops=1)<br>**Filter**: (n_dadi > '2'::numeric)<br>**Rows Removed by Filter**: 34 | 280 | 1 |

Viene utilizzato un hash join sui campi id_g di Sfida e Gioco per permettere l'hash inner join.

Viene compiuta anche un'operazione di  hash sulla tabella gioco che necessita di una seq_scan in ingresso.

L’unico modo utile per ottimizzare queste interrogazioni è la creazione di indici utili perché la creazione di viste materializzate non ha molto senso su interrogazioni che fanno uso così esiguo di join.

CREATE INDEX SquadreInGioco ON GIOCO(n_squadre);
CREATE INDEX DadiInGioco ON GIOCO(n_dadi);

CREATE INDEX DataInSfida on SFIDA(data);
CREATE INDEX DurataInSfida on SFIDA(durata_max);

CREATE INDEX IDgioco on GIOCO(id_g);
CREATE INDEX IDsfida on SFIDA(id_g);

CLUSTER GIOCO USING IDgioco;
CLUSTER SFIDA USING IDsfida;

DOPO LA CREAZIONE DEGLI INDICI LA SITUAZIONE NELLA BASE DI DATI E’ LA SEGUENTE:

Per quanto riguarda la prima interrogazione dal momento che la quantità di tuple in gioco è piuttosto elevata il sistema decide di usare l’indice da noi creato.

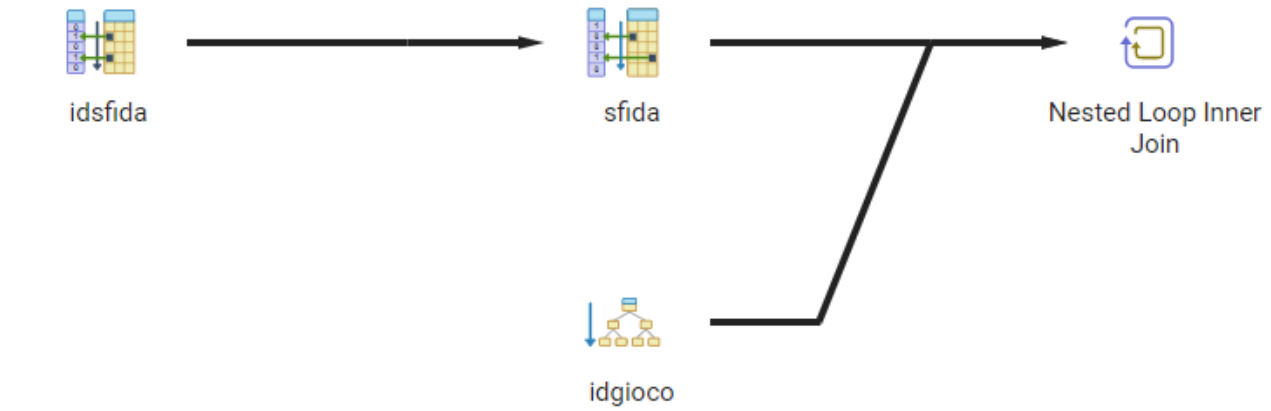

Il sistema se ci sono troppi pochi dati preferisce continuare ad utilizzare una SEQ_SCAN ma in questo caso con l’utilizzo di datanamic il numero di tuple è abbastanza elevato da utilizzare l’indice.

| # | Node | Rows Actual | Loops |
|---|---|---|---|
| 1. | → Bitmap Heap Scan on gioco as gioco (rows=2161 loops=1)<br>**Filter**: (n_dadi > '2'::numeric)<br>**Rows Removed by Filter**: 254<br>**Recheck Cond**: (n_squadre <= '4'::numeric)<br>**Heap Blocks**: exact=241 | 2161 | 1 |
| 2. | → Bitmap Index Scan using squadreingioco (rows=2415 loops=1)<br>**Index Cond**: (n_squadre <= '4'::numeric) | 2415 | 1 |

Anche i costi dovrebbero essere sensibilmente migliorati.

Nel caso della seconda interrogazione gli indici vengono utilizzati permettendo una scansione più rapida per il join:

| # | Node | Actual | Loops |
|---|---|---|---|
| 1. | → Nested Loop Inner Join (rows=0 loops=1) | 0 | 1 |
| 2. | → Bitmap Heap Scan on sfida as sfida (rows=0 loops=1)<br>**Filter**: (((data >= '2021-01-01'::date) AND (data <= '2021-01-31'::date) AND (durata_max > '02:00:00'::time without time zone)) OR ((data >= '2021-03-01'::date) AND (data <= '2021-03-31'::date) AND (durata_max = '00:30:00'::time without time zone)))<br>**Rows Removed by Filter**: 0<br>**Recheck Cond**: ((id_g)::text = 'ab0'::text)<br>**Heap Blocks**: exact=0 | 0 | 1 |
| 3. | → Bitmap Index Scan using idsfida (rows=0 loops=1)<br>**Index Cond**: ((id_g)::text = 'ab0'::text) | 0 | 1 |
| 4. | → Index Only Scan using idgioco on gioco as gioco (rows=0 loops=0)<br>**Index Cond**: (id_g = 'ab0'::text) | 0 | 0 |




In questo modo la velocità della query e quindi del workload intero aumenta motevolmente.

Infine la terza interrogazione preferisce nuovamente utilizzare l'hash join creato automaticamente dal sistema.

La scelta degli indici quindi non è stata utile ( a causa dello schema ancora troppo piccolo).

In generale guardando le altre interrogazioni richieste abbiamo provato a dare delle ottimizzazione che hanno velocizzato fortemente la basi di dati.

Per esempio la query per determinare i giochi con task associati faceva uso di un molteplice join molto costoso.

Abbiamo pensato quindi di creare una vista materializzata così da non dover ripetere più volte il join.

CREATE MATERIALIZED VIEW game_with_task

AS SELECT GIOCO.id_g

FROM ((GIOCO NATURAL JOIN CASELLA) NATURAL JOIN DOMANDA) NATURAL JOIN TASK

WHERE DOMANDA.cod_domanda = TASK.cod_domanda;