Progetto di Base Dati 2023 - "Orti Scolastici" - 12 CFU

Componenti del Gruppo

- **Andrea Franceschetti** - 4357070
- **William Chen** - 4827847
- **Alessio De Vincenzi**s - 4878315

Progettazione Fisica

Al fine di verificare il corretto funzionamento delle interrogazioni a pieno carico, si è deciso di creare un database di test con un elevato numero di tuple, in modo da valutare le prestazioni delle interrogazioni in situazioni di carico massimo.

Per fare ciò, abbiamo creato un Notebook in Python che genera un elevato numero di dati in modo dinamico e casuale, inserendoli direttamente nel database.

Lo script, generando valori di dati in modo altamente casuale durante la creazione del database, permette di ottenere un database di test diverso ad ogni esecuzione.

Nota: Lo script è stato creato per essere eseguito su un database vuoto, quindi se si vuole eseguirlo nuovamente, è necessario eliminare il database e ricrearlo utilizzando il file SQL '[Parte II.A.A] BaseDati.sql'.

Interrogazioni a pieno carico

Interrogazione 1 - JOIN

Questa query restituisce il nome del docente, la sezione della classe e il nome dell'orto per tutte le scuole che hanno il finanziamento abilitato.

```
SELECT P.Nome, C.Sezione, O.NomeOrto
FROM Persona P
JOIN Classe C ON P.Email = C.Docente
```

```
JOIN Scuola S ON C.Scuola = S.Cod_Meccanografico
JOIN Orto O ON O.Scuola = S.Cod_Meccanografico
WHERE S.Finanziamento IS NOT NULL
GROUP BY P.Nome, C.Sezione, O.NomeOrto
ORDER BY O.NomeOrto;
```

Interrogazione 2 - Condizione Complessa

Questa query restituisce l'ID della rilevazione, la temperatura e l'umidità dai dati per tutte le rilevazioni in cui la temperatura è superiore a 25 e l'umidità è superiore a 80, oppure per tutte le rilevazioni effettuate a partire dal 1º gennaio 2023.

```
SELECT R.IdRilevazione, D.Temperatura, D.Umidita
FROM Rilevazione R
JOIN Dati D ON R.IdRilevazione = D.Rilevazione
WHERE (D.Temperatura > 25 AND D.Umidita > 80) OR R.DataOraRilevazione >= '2023-01-
01';
```

Interrogazione 3 - Funzione Generica

Questa query restituisce il nome della scuola e il numero di piante coltivate in ciascuna scuola, considerando solo le scuole che hanno più di 10 piante coltivate.

```
SELECT S.NomeScuola, COUNT(*) AS NumeroPiante
FROM Scuola S
JOIN Classe C ON S.Cod_Meccanografico = C.Scuola
JOIN Pianta P ON C.IdClasse = P.Classe
GROUP BY S.NomeScuola
HAVING COUNT(*) > 10;
```

Indici per le interrogazioni a pieno carico

L'idea che abbiamo avuto è di realizzare indici clusterizzati di tipo B-Tree i quali organizzano fisicamente i dati sulla base delle colonne specificate, migliorando le prestazioni delle operazioni di lettura che coinvolgono tali colonne.

Tuttavia, abbiamo tenuto presente che l'utilizzo di indici clusterizzati può comportare una penalità sulle operazioni di inserimento, aggiornamento e cancellazione, poiché i dati devono essere riorganizzati per mantenere l'ordine clusterizzato.

Indice 1 - JOIN

Questo indice è stato creato per velocizzare la query di JOIN.

```
CREATE INDEX idx_Classe_Docente ON Classe USING btree(Docente);
CLUSTER Classe USING idx_Classe_Docente;

CREATE INDEX idx_Scuola_Cod_Meccanografico ON Scuola USING
btree(Cod_Meccanografico);
CLUSTER Scuola USING idx_Scuola_Cod_Meccanografico;

CREATE INDEX idx_Scuola_Finanziamento ON Scuola USING btree(Finanziamento);
CLUSTER Scuola USING idx_Scuola_Finanziamento;
```

La creazione di indici clusterizzati sulla colonna "Docente" nella tabella "Classe" e sulle colonne "Cod_Meccanografico" e "Finanziamento" nella tabella "Scuola", ci permette di accedere direttamente ai dati di nostro interesse, accelerando conseguentemente i le operazioni di JOIN tra le tabelle.

Indice 2 - Condizione Complessa

Questo indice è stato creato per velocizzare la query di Condizione Complessa.

```
CREATE INDEX idx_Rilevazione_DataOraRilevazione ON Rilevazione USING
btree(DataOraRilevazione);
CLUSTER Rilevazione USING idx_Rilevazione_DataOraRilevazione;

CREATE INDEX idx_Dati_Rilevazione ON Dati USING btree(Rilevazione);
CLUSTER Dati USING idx_Dati_Rilevazione;

CREATE INDEX idx_Dati_Temperatura_Umidita ON Dati USING btree(Temperatura,
Umidita);
CLUSTER Dati USING idx_Dati_Temperatura_Umidita;
```

L'indice clusterizzato sulla colonna "DataOraRilevazione" nella tabella "Rilevazione" organizza fisicamente le righe di dati in base ai valori di questa colonna. Gli indici clusterizzati sulla colonna "Rilevazione" nella tabella "Dati" e sulle colonne "Temperatura" e "Umidita" nella tabella "Dati" hanno lo stesso effetto, organizzando fisicamente le righe di dati in base ai valori di queste colonne.

Indice 3 - Funzione Generica

Questo indice è stato creato per velocizzare la query di Funzione Generica.

```
CREATE INDEX idx_Classe_Scuola ON Classe USING btree(Scuola);
CLUSTER Classe USING idx_Classe_Scuola;

CREATE INDEX idx_Pianta_Classe ON Pianta USING btree(Classe);
CLUSTER Pianta USING idx_Pianta_Classe;

CREATE INDEX idx_Scuola_NomeScuola ON Scuola USING btree(NomeScuola);
CLUSTER Scuola USING idx_Scuola_NomeScuola;

CREATE INDEX idx_Pianta_Specie ON Pianta USING btree(Specie);
CLUSTER Pianta USING idx_Pianta_Specie;
```

Gli indici clusterizzati sulla colonna "Scuola" nella tabella "Classe", sulla colonna "Classe" nella tabella "Pianta", sulla colonna "NomeScuola" nella tabella "Scuola" e sulla colonna "Specie" nella tabella "Pianta" organizzano fisicamente le righe di dati in base ai valori di queste colonne, migliorando le prestazioni delle query che coinvolgono tali colonne.

Tabella riassuntivo del carico di lavoro

Dati ottenuti tramite la seguente query:

Nota: La query estrae i valori richiesti per tutte le tabelle del database

```
SELECT
  t.relname AS Tabella,
  t.n_tup_ins AS Numero_Tuple_Inserite,
  pg_size_pretty(pg_relation_size(t.relid)) AS Dimensione_Blocchi,
  pg_size_pretty(pg_total_relation_size(t.relid)) AS Dimensione_Totale
FROM
  pg_stat_user_tables t
  JOIN pg_class c ON t.relname = c.relname
WHERE
  c.relkind = 'r' -- Considera solo le tabelle
ORDER BY
  t.relname;
```

Tabella	Numero Tuple	Dimensione Blocchi Singolo	Dimensione Totale
Classe	19	8192 byte	24 kb
Scuola	10	8192 byte	32 kb
Pianta	100	16 kb	56 kb

Rilevazione	100	8192 byte	24 kb
Dati	100	8192 byte	24 kb

Piani di esecuzione delle interrogazioni a pieno carico scelti dal Sistema

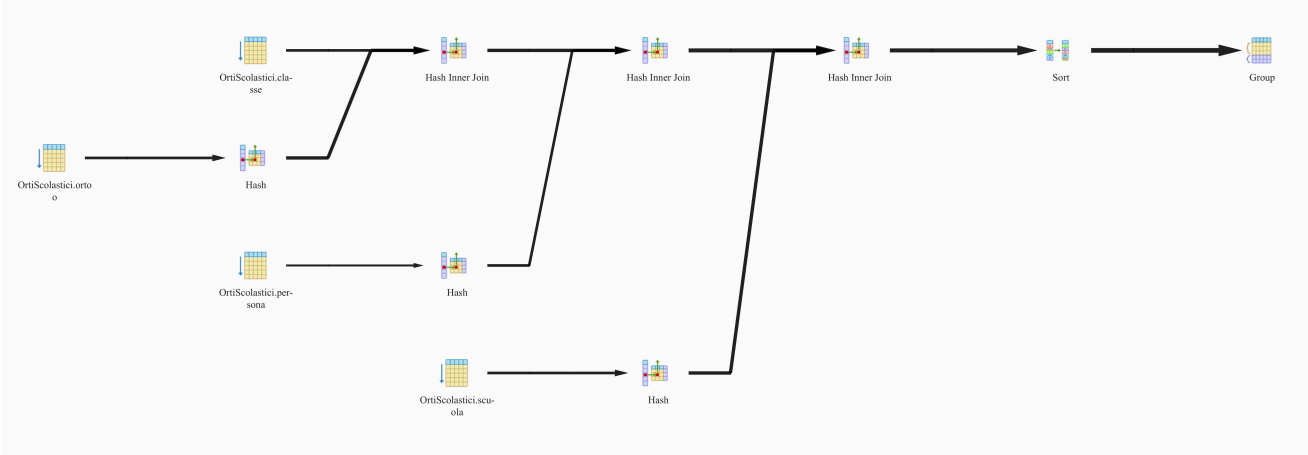
Interrogazione 1 - JOIN

Piano esecuzione prima dell'indice

- Analisi

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Group (cost=53.58..56.58 rows=300 width=249) (actual=0.311..0.331 rows=66 loops...	0.018 ms	0.331 ms	↑ 4.55	66	300	1
2.	→ Sort (cost=53.58..54.33 rows=300 width=249) (actual=0.31..0.314 rows=66 loo...	0.147 ms	0.314 ms	↑ 4.55	66	300	1
3.	→ Hash Inner Join (cost=27.63..41.23 rows=300 width=249) (actual=0.116..0... **Hash Cond**: ((c.scuola)::text = (s.cod_meccanografico)::text)	0.02 ms	0.167 ms	↑ 4.55	66	300	1
4.	→ Hash Inner Join (cost=16.5..29.84 rows=100 width=325) (actual=0.08... **Hash Cond**: ((c.docente)::text = (p.email)::text)	0.023 ms	0.121 ms	↑ 1.52	66	100	1
5.	→ Hash Inner Join (cost=12.25..25.31 rows=100 width=536) (actual... **Hash Cond**: ((c.scuola)::text = (o.scuola)::text)	0.027 ms	0.056 ms	↑ 1.52	66	100	1
6.	→ Seq Scan on OrtiScolastici.classe as c (cost=0..11.5 rows=1...	0.009 ms	0.009 ms	↑ 7.9	19	150	1
7.	→ Hash (cost=11..11 rows=100 width=256) (actual=0.019..0.02... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	0.008 ms	0.02 ms	↑ 3.34	30	100	1
8.	→ Seq Scan on OrtiScolastici.orto as o (cost=0..11 rows=1...	0.012 ms	0.012 ms	↑ 3.34	30	100	1
9.	→ Hash (cost=3..3 rows=100 width=31) (actual=0.043..0.043 rows=... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 15 kB	0.023 ms	0.043 ms	↑ 1	100	100	1
10.	→ Seq Scan on OrtiScolastici.persona as p (cost=0..3 rows=10...	0.02 ms	0.02 ms	↑ 1	100	100	1
11.	→ Hash (cost=10.5..10.5 rows=50 width=38) (actual=0.027..0.027 rows=... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.011 ms	0.027 ms	↑ 5	10	50	1
12.	→ Seq Scan on OrtiScolastici.scuola as s (cost=0..10.5 rows=50 wid... **Filter**: (s.finanziamento IS NOT NULL) **Rows Removed by Filter**: 0	0.016 ms	0.016 ms	↑ 5	10	50	1

- Grafico

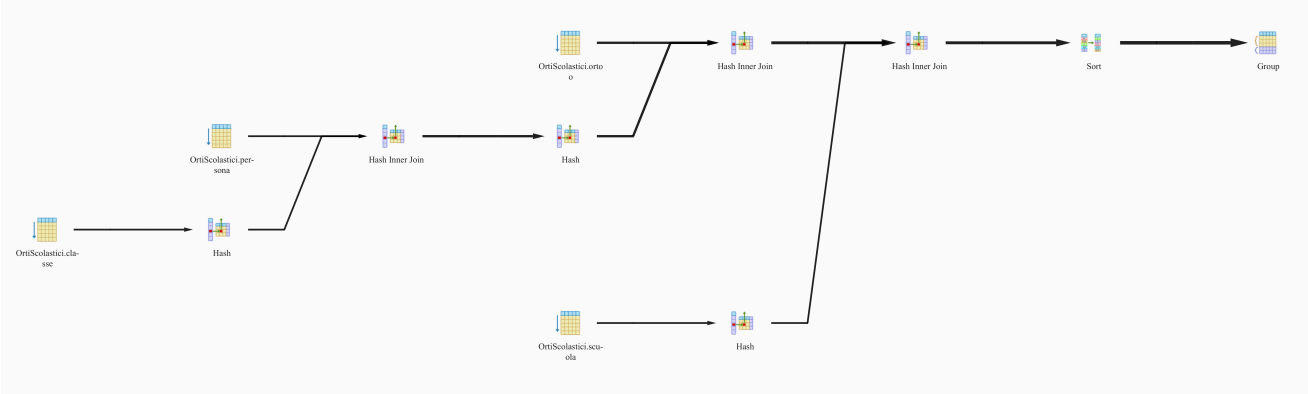


Piano esecuzione dopo l'indice

- Analisi

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Group (cost=25.27..27.17 rows=190 width=249) (actual=0.388..0.414 rows=66 loops...	0.022 ms	0.414 ms	↑ 2.88	66	190	1
2.	→ Sort (cost=25.27..25.75 rows=190 width=249) (actual=0.387..0.392 rows=66 lo...	0.192 ms	0.392 ms	↑ 2.88	66	190	1
3.	→ Hash Inner Join (cost=6.46..18.08 rows=190 width=249) (actual=0.157..0.... **Hash Cond**: ((c.scuola)::text = (s.cod_meccanografico)::text)	0.037 ms	0.2 ms	↑ 2.88	66	190	1
4.	→ Hash Inner Join (cost=5.23..16.79 rows=19 width=325) (actual=0.107.... **Hash Cond**: ((o.scuola)::text = (c.scuola)::text)	0.034 ms	0.131 ms	↓ 3.48	66	19	1
5.	→ Seq Scan on OrtiScolastici.orto as o (cost=0..11 rows=100 width...	0.013 ms	0.013 ms	↑ 3.34	30	100	1
6.	→ Hash (cost=4.99..4.99 rows=19 width=69) (actual=0.083..0.084 r... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.009 ms	0.084 ms	↑ 1	19	19	1
7.	→ Hash Inner Join (cost=1.43..4.99 rows=19 width=69) (actual... **Hash Cond**: ((p.email)::text = (c.docente)::text)	0.033 ms	0.076 ms	↑ 1	19	19	1
8.	→ Seq Scan on OrtiScolastici.persona as p (cost=0..3 rows...	0.019 ms	0.019 ms	↑ 1	100	100	1
9.	→ Hash (cost=1.19..1.19 rows=19 width=280) (actual=0.0... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	0.012 ms	0.024 ms	↑ 1	19	19	1
10.	→ Seq Scan on OrtiScolastici.classe as c (cost=0..1.1...	0.013 ms	0.013 ms	↑ 1	19	19	1
11.	→ Hash (cost=1.1..1.1 rows=10 width=38) (actual=0.033..0.033 rows=10... **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.011 ms	0.033 ms	↑ 1	10	10	1
12.	→ Seq Scan on OrtiScolastici.scuola as s (cost=0..1.1 rows=10 widt... **Filter**: (s.finanziamento IS NOT NULL) **Rows Removed by Filter**: 0	0.023 ms	0.023 ms	↑ 1	10	10	1

- Grafico



Conclusioni

Con l'introduzione degli indici clusterizzati, il piano di esecuzione fisico cambia di poco, ma le prestazioni peggiorano leggermente. Infatti, il tempo di esecuzione della query è di 0.3 ms senza indici e di 0.4 ms con indici.

Questo perchè essendo una query di JOIN, il sistema deve comunque eseguire un'operazione di JOIN tra le tabelle, quindi l'indice clusterizzato non ha un impatto significativo sulle prestazioni.

Il piano di esecuzione fisico per eseguire il JOIN sceglie di eseguire un'operazione di Hash Join tra le tabelle "Classe" e "Persona", utilizzando l'indice clusterizzato sulla colonna "Docente" nella tabella "Classe" e l'indice clusterizzato sulla colonna "Email" nella tabella "Persona".

Dopo aver eseguito l'operazione di Hash Join, il sistema esegue un'operazione di Hash Join tra le tabelle "Scuola" e "Classe", utilizzando l'indice clusterizzato sulla colonna "Cod_Meccanografico" nella tabella "Scuola" e l'indice clusterizzato sulla colonna "Scuola" nella tabella "Classe".

Infine, il sistema esegue un'operazione di Hash Join tra le tabelle "Orto" e "Scuola", utilizzando l'indice clusterizzato sulla colonna "Cod_Meccanografico" nella tabella "Scuola" e l'indice clusterizzato sulla colonna "Scuola" nella tabella "Orto".

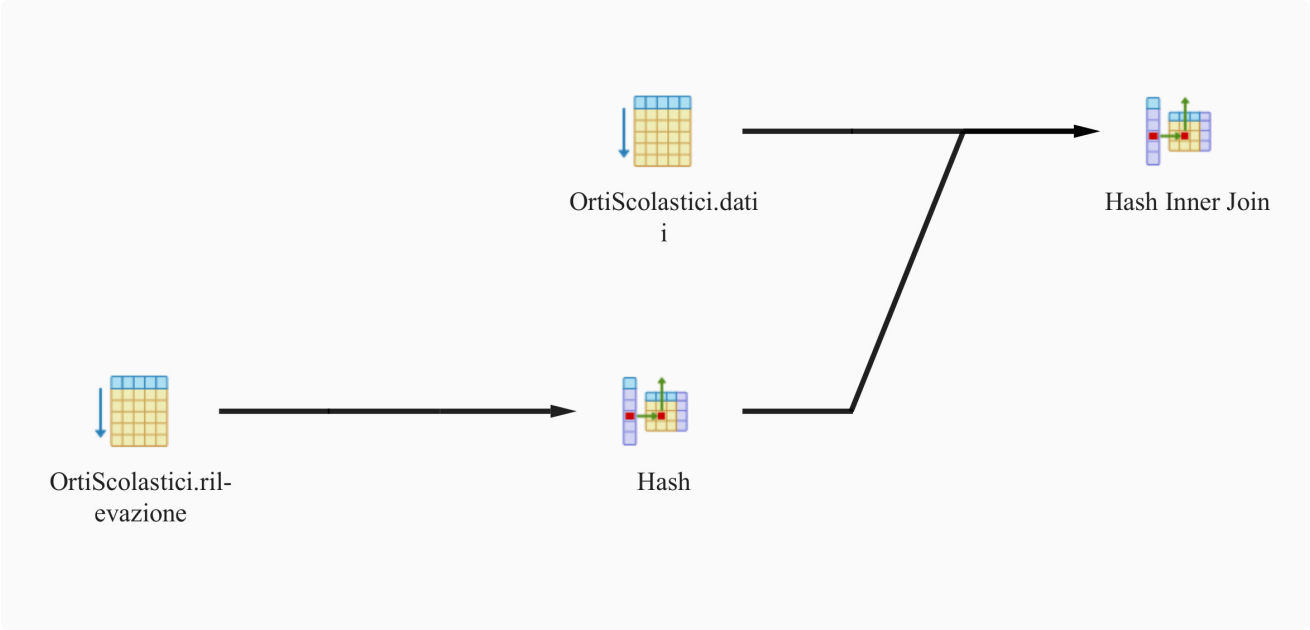
Interrogazione 2 - Condizione Complessa

Piano esecuzione prima dell'indice

- Analisi

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Hash Inner Join (cost=3.25..6.51 rows=10 width=24) (actual=0.068..0.114 rows=9 loo… **Join Filter**: (((d.temperatura > '25'::double precision) AND (d.umidita > '80'::double precision)) O **Hash Cond**: (d.rilevazione = r.idrilevazione)	0.054 ms	**0.114 ms**	↑ 1.12	9	10	1
2.	→ Seq Scan on OrtiScolastici.dati as d (cost=0..3 rows=100 width=24) (actual=0.01…	0.023 ms	0.023 ms	↑ 1	100	100	1
3.	→ Hash (cost=2..2 rows=100 width=16) (actual=0.037..0.037 rows=100 loops=1) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 13 kB	0.017 ms	0.037 ms	↑ 1	100	100	1
4.	→ Seq Scan on OrtiScolastici.rilevazione as r (cost=0..2 rows=100 width=16) (…	0.02 ms	0.02 ms	↑ 1	100	100	1

- Grafico

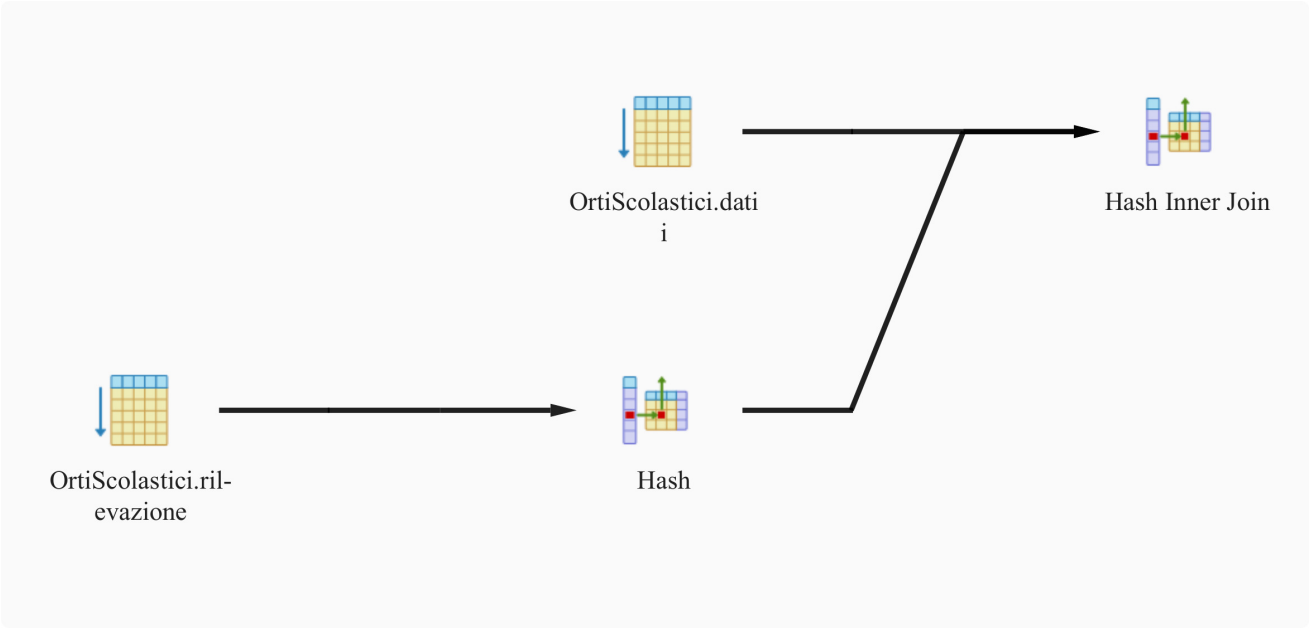


Piano esecuzione dopo l'indice

- Analisi

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Hash Inner Join (cost=3.25..6.51 rows=10 width=24) (actual=0.073..0.107 rows=9 loo… **Join Filter**: (((d.temperatura > '25'::double precision) AND (d.umidita > '80'::double precision)) O **Hash Cond**: (d.rilevazione = r.idrilevazione)	0.043 ms	**0.107 ms**	↑ 1.12	9	10	1
2.	→ Seq Scan on OrtiScolastici.dati as d (cost=0..3 rows=100 width=24) (actual=0.01…	0.023 ms	0.023 ms	↑ 1	100	100	1
3.	→ Hash (cost=2..2 rows=100 width=16) (actual=0.041..0.041 rows=100 loops=1) **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 13 kB	0.02 ms	0.041 ms	↑ 1	100	100	1
4.	→ Seq Scan on OrtiScolastici.rilevazione as r (cost=0..2 rows=100 width=16) (…	0.022 ms	0.022 ms	↑ 1	100	100	1

- Grafico



Conclusioni

Con l'introduzione degli indici clusterizzati, il piano di esecuzione fisico non cambia e le prestazioni rimangono le stesse. Infatti, il tempo di esecuzione della query è di 0.1 ms sia senza indici che con indici.

Questo perchè essendo una query di JOIN, il sistema deve comunque eseguire un'operazione di JOIN tra le tabelle, quindi l'indice clusterizzato non ha un impatto significativo sulle prestazioni.

Il piano di esecuzione fisico per eseguire il JOIN sceglie di eseguire un'operazione di Hash Join tra le tabelle "Rilevazione" e "Dati", utilizzando l'indice clusterizzato sulla colonna "Rilevazione" nella tabella "Dati".

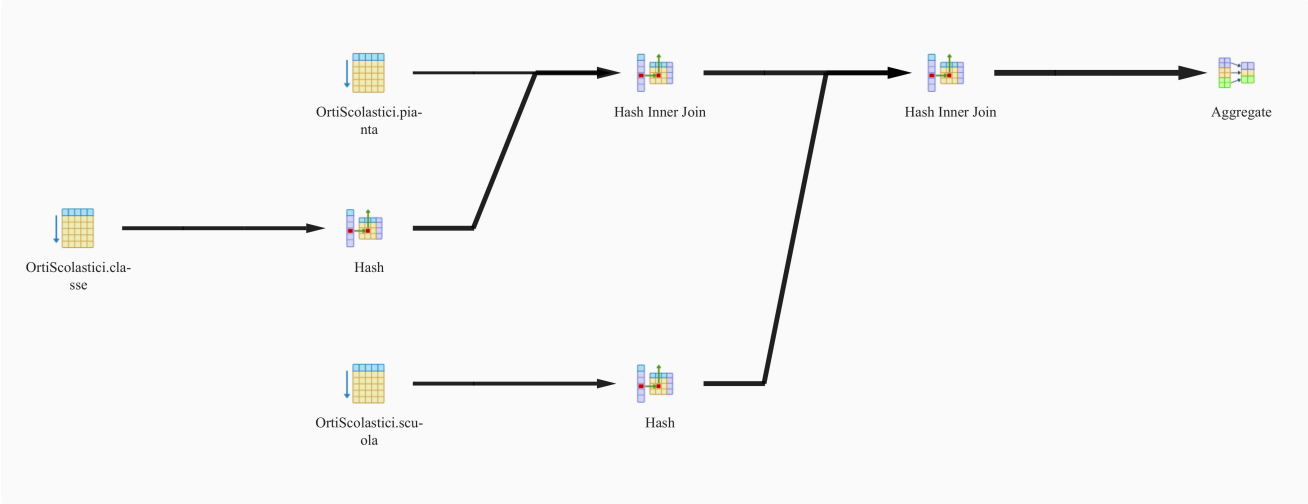
Interrogazione 3 - Funzione Generica

Piano esecuzione prima dell'indice

- Analisi

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Aggregate (cost=28.55..29.17 rows=17 width=226) (actual=0.224..0.228 rows=4 loop… **Filter**: (count(*) > 10) **Rows Removed by Filter**: 2 **Buckets**: **Batches**: **Memory Usage**: 24 kB	0.043 ms	**0.228 ms**	↑ 4.25	4	17	1
2.	→ Hash Inner Join (cost=24.5..28.05 rows=100 width=218) (actual=0.11..0.186 row… **Hash Cond**: ((c.scuola)::text = (s.cod_meccanografico)::text)	0.036 ms	0.186 ms	↑ 1	100	100	1
3.	→ Hash Inner Join (cost=13.38..16.65 rows=100 width=38) (actual=0.05..0.09… **Hash Cond**: (p.classe = c.idclasse)	0.049 ms	0.095 ms	↑ 1	100	100	1
4.	→ Seq Scan on OrtiScolastici.pianta as p (cost=0..3 rows=100 width=8) (a…	0.026 ms	0.026 ms	↑ 1	100	100	1
5.	→ Hash (cost=11.5..11.5 rows=150 width=46) (actual=0.02..0.02 rows=19… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	0.006 ms	0.02 ms	↑ 7.9	19	150	1
6.	→ Seq Scan on OrtiScolastici.classe as c (cost=0..11.5 rows=150 wid…	0.014 ms	0.014 ms	↑ 7.9	19	150	1
7.	→ Hash (cost=10.5..10.5 rows=50 width=256) (actual=0.055..0.055 rows=10 lo… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.03 ms	0.055 ms	↑ 5	10	50	1
8.	→ Seq Scan on OrtiScolastici.scuola as s (cost=0..10.5 rows=50 width=25…	0.025 ms	0.025 ms	↑ 5	10	50	1

- Grafico

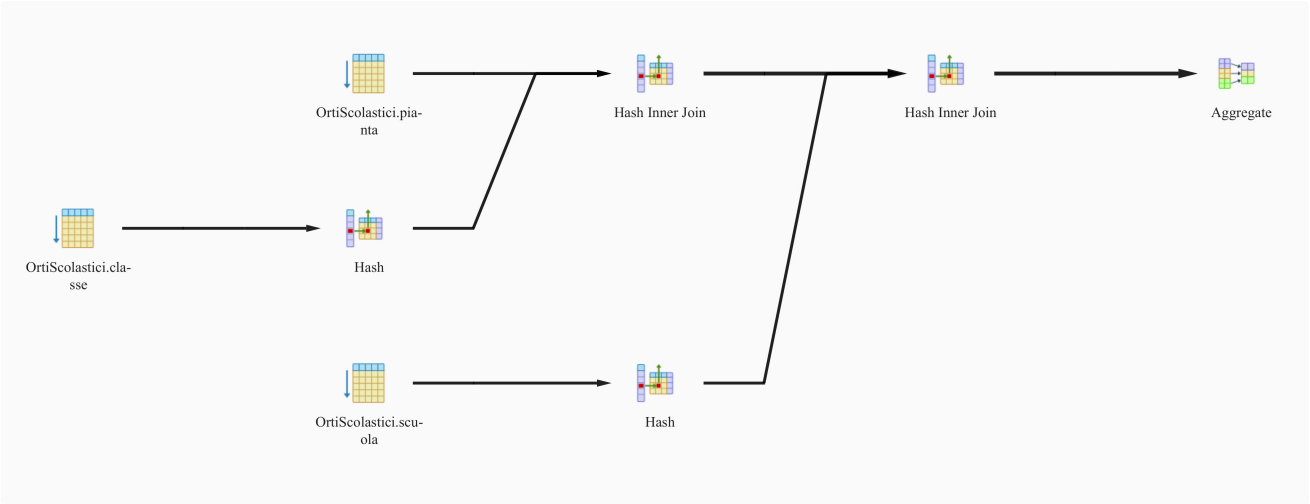


Piano esecuzione dopo l'indice

- Analisi

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Aggregate (cost=6.79..6.91 rows=3 width=226) (actual=0.181..0.184 rows=4 loops=1) **Filter**: (count(*) > 10) **Rows Removed by Filter**: 2 **Buckets**: **Batches**: **Memory Usage**: 24 kB	0.037 ms	**0.184 ms**	↓ 1.34	4	3	1
2.	→ Hash Inner Join (cost=2.65..6.29 rows=100 width=218) (actual=0.065..0.148 row… **Hash Cond**: ((c.scuola)::text = (s.cod_meccanografico)::text)	0.048 ms	0.148 ms	↑ 1	100	100	1
3.	→ Hash Inner Join (cost=1.43..4.75 rows=100 width=38) (actual=0.033..0.072 … **Hash Cond**: (p.classe = c.idclasse)	0.033 ms	0.072 ms	↑ 1	100	100	1
4.	→ Seq Scan on OrtiScolastici.pianta as p (cost=0..3 rows=100 width=8) (a…	0.022 ms	0.022 ms	↑ 1	100	100	1
5.	→ Hash (cost=1.19..1.19 rows=19 width=46) (actual=0.017..0.017 rows=1… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 10 kB	0.005 ms	0.017 ms	↑ 1	19	19	1
6.	→ Seq Scan on OrtiScolastici.classe as c (cost=0..1.19 rows=19 widt…	0.013 ms	0.013 ms	↑ 1	19	19	1
7.	→ Hash (cost=1.1..1.1 rows=10 width=256) (actual=0.028..0.028 rows=10 loop… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.007 ms	0.028 ms	↑ 1	10	10	1
8.	→ Seq Scan on OrtiScolastici.scuola as s (cost=0..1.1 rows=10 width=256…	0.022 ms	0.022 ms	↑ 1	10	10	1

- Grafico



Conclusioni

Con l'introduzione degli indici clusterizzati, il piano di esecuzione fisico cambia di poco, ma le prestazioni peggiorano leggermente. Infatti, il tempo di esecuzione della query è di 0.2 ms senza indici e di 0.1 ms con indici.

Questo perchè essendo una query di JOIN, il sistema deve comunque eseguire un'operazione di JOIN tra le tabelle, quindi l'indice clusterizzato non ha un impatto significativo sulle prestazioni.

Il piano di esecuzione fisico per eseguire il JOIN sceglie di eseguire un'operazione di Hash Join tra le tabelle "Classe" e "Scuola", utilizzando l'indice clusterizzato sulla colonna "Scuola" nella tabella "Classe" e l'indice clusterizzato sulla colonna "Cod_Meccanografico" nella tabella "Scuola".

Dopo aver eseguito l'operazione di Hash Join, il sistema esegue un'operazione di Hash Join tra le tabelle "Pianta" e "Classe", utilizzando l'indice clusterizzato sulla colonna "Classe" nella tabella "Pianta" e l'indice clusterizzato sulla colonna "IdClasse" nella tabella "Classe".

Infine, il sistema esegue un'operazione di Hash Join tra le tabelle "Pianta" e "Specie", utilizzando l'indice clusterizzato sulla colonna "Specie" nella tabella "Pianta" e l'indice clusterizzato sulla colonna "NomeSpecie" nella tabella "Specie".

Progettazione Fisica - Sicurezza

Controllo degli Accessi Gerarchia dei Ruoli

Il successivo elenco mostra la gerarchia dei ruoli in ordine decrescente di privilegi:

1. **Amministratore** - Ruolo che ha tutti i privilegi (Gestore globale del Progetto).
2. **Referente** - Ruolo che ha i privilegi di gestione dei dati di una Scuola.
3. **Insegnante** - Ruolo che ha i privilegi di gestione dei dati di una Classe.
4. **Studente** - Ruolo che ha i privilegi di gestione dati di una Classe ma minori di **Insegnante**.

Tabella Privilegi

#	Amministratore	Referente	Insegnante	Studente
Persona	ALL	SELECT, INSERT, UPDATE	SELECT	SELECT
Scuola	ALL	SELECT, INSERT, UPDATE	SELECT	SELECT
Classe	ALL	SELECT, INSERT, UPDATE	SELECT	SELECT
Studente	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT
Specie	ALL	SELECT, INSERT, UPDATE	SELECT, INSERT, UPDATE	SELECT
Orto	ALL	SELECT, INSERT, UPDATE	SELECT	SELECT
Pianta	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT

Esposizione	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT
Gruppo	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT
Sensore	ALL	SELECT, INSERT, UPDATE	SELECT, INSERT, UPDATE	SELECT
Rilevazione	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE
Dati	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE
Responsabile	ALL	SELECT, INSERT, UPDATE, DELETE	SELECT, INSERT, UPDATE, DELETE	SELECT

Nota: ALL = SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER.

Politica dei Privilegi

Si è deciso di applicare la filosofia del minimo privilegio, ovvero assegnare ad ogni ruolo solo i privilegi necessari per svolgere il proprio lavoro.

1. **Amministratore -** Ha accesso a tutti i dati del database e può eseguire tutte le operazioni.

2. **Referente** - Ha accesso ai dati della Scuola di cui è Referente e può eseguire tutte le operazioni su di essi.
    - Scritture: Inserimento, Modifica -> Su tutti i dati che riguardano la Scuola di cui è Referente.
    - Scritture: Cancellazione -> Solo su dati che riguardano le Piante e le Rilevazioni fatte dalla Scuola di cui è Referente.
    - Letture: Visualizzazione -> Su tutti i dati che riguardano la Scuola di cui è Referente.
    - Non può eseguire operazioni su dati di altre Scuole.
3. **Insegnante** - Ha accesso ai dati della Classe di cui è Insegnante e può eseguire tutte le operazioni su di essi.
    - Scritture: Inserimento, Modifica -> Su tutti i dati che riguardano la Classe di cui è Insegnante.
    - Scritture: Cancellazione -> Solo su dati che riguardano le Piante e le Rilevazioni fatte dalla Classe di cui è Insegnante.
    - Letture: Visualizzazione -> Su tutti i dati che riguardano la Classe di cui è Insegnante.
    - Può eseguire operazioni su dati di altre Classi della stessa Scuola ammesso che ne sia un Insegnante.
4. **Studente** - Ha accesso ai dati della Classe di cui è Studente e può eseguire tutte le operazioni su di essi.
    - Scritture: Inserimento, Modifica -> Su tutti i dati che riguardano la Classe di cui è Studente.
    - Scritture: Cancellazione -> Solo su dati che riguardano le Rilevazioni fatte dalla Classe di cui è Studente.
    - Letture: Visualizzazione -> Su tutti i dati che riguardano la Classe di cui è Studente.
    - Non può eseguire operazioni su dati di altre Classi.