Relazione Progetto Finale

Confronto delle prestazioni di SQL, NoSQL e NewSQL in diversi scenari applicativi: un'analisi sperimentale

Corso di Big Data – AA 2022/2023

Federico Tocci – n.matr. 533449

https://github.com/FTocci/BigData_ProgettoFinale

Introduzione

Nell'era dell'informazione digitale, la gestione efficiente dei dati si configura come un aspetto non solo importante ma addirittura cruciale per il successo e la sopravvivenza di aziende e organizzazioni di ogni dimensione. L'esplosione della quantità di dati generati e raccolti quotidianamente ha sottolineato in maniera ancor più incisiva l'importanza della scelta del sistema di gestione più adeguato. Tale scelta riveste un ruolo fondamentale poiché incide direttamente sull'efficacia e sulla scalabilità delle operazioni aziendali, influenzando aspetti come l'accesso tempestivo alle informazioni, la flessibilità nel loro utilizzo e l'abilità nel trarre conoscenza dai dati stessi.

In questo complesso scenario, emergono tre categorie principali di sistemi di gestione dati, ciascuna con il proprio approccio distintivo nell'organizzazione, nell'archiviazione e nell'interrogazione dei dati: sistemi SQL, NoSQL e NewSQL.

I sistemi SQL, che si basano sul linguaggio strutturato SQL (Structured Query Language), offrono un approccio consolidato e altamente strutturato per la gestione dei dati, ideale soprattutto per applicazioni che richiedono un rigoroso controllo delle transazioni e una struttura ben definita. Dall'altro lato, i sistemi NoSQL, che abbracciano una vasta gamma di tecnologie come database document-oriented, key-value store e graph database, si sono affermati come soluzioni flessibili, scalabili e adattabili a carichi di lavoro diversificati, spesso caratterizzati da un'enorme mole di dati non strutturati.

Nel mezzo si collocano i sistemi NewSQL, che rappresentano un compromesso tra le tradizionali proprietà dei sistemi SQL e la scalabilità offerta da quelli NoSQL. Tali sistemi cercano di coniugare l'affidabilità delle transazioni SQL con la distribuzione e la scalabilità orizzontale tipiche dei sistemi NoSQL.

L'obiettivo di questa relazione è quello di offrire una panoramica delle differenze fondamentali tra le diverse tipologie di sistemi, analizzando in modo dettagliato sia le prestazioni offerte in diversi scenari applicativi che la facilità d'utilizzo per gli utenti.

Tecnologie utilizzate

In questo paragrafo saranno presentate le varie tecnologie oggetto di confronto all'interno del progetto. Si desidera focalizzare l'attenzione sulle qualità distintive di ciascuna scelta, fornendo così un'ampia panoramica delle risorse messe a disposizione da essa.

Sistema SQL – PostgreSQL

All'interno del panorama dei sistemi SQL, PostgreSQL emerge come una delle soluzioni più avanzate e versatili. Questo sistema di gestione di database relazionale si distingue per la sua capacità di offrire un'ampia gamma di funzionalità e caratteristiche avanzate, rendendolo una scelta popolare per molteplici applicazioni.

PostgreSQL è noto per la sua conformità agli standard SQL, assicurando che le query e le operazioni siano portabili tra diversi database. Ciò lo rende una scelta affidabile per progetti che richiedono coerenza e affidabilità nella gestione dei dati. Inoltre, PostgreSQL supporta estensioni e linguaggi di programmazione, permettendo agli sviluppatori di creare funzionalità personalizzate e complesse direttamente all'interno del database. La sua architettura modulare e la flessibilità nella definizione dei tipi di dati consentono una personalizzazione precisa per adattarsi a varie esigenze di business. Con capacità di gestione delle transazioni e di concorrenza avanzate, PostgreSQL si distingue anche per la sua scalabilità e affidabilità, rendendolo una soluzione apprezzata in scenari in cui la gestione di dati critici è fondamentale.

Sistema NoSQL – Cassandra

Tra le opzioni offerte dai sistemi NoSQL, Apache Cassandra emerge come una scelta di spicco per gestire dati su larga scala e distribuiti. La sua architettura decentralizzata consente di scalare orizzontalmente su cluster di macchine, garantendo prestazioni affidabili anche in presenza di grandi volumi di dati e carichi di lavoro intensi.

Cassandra si adatta perfettamente a scenari in cui la disponibilità dei dati è essenziale, grazie al suo modello di consistenza diagnostica, che consente la distribuzione dei dati su più nodi senza compromettere l'integrità delle informazioni. I dati vengono organizzati in famiglie di colonne, consentendo una flessibilità nella modellazione dei dati che si adatta a molteplici casi d'uso. La sua capacità di gestire dati in modo rapido ed efficiente è particolarmente preziosa per applicazioni web e mobile ad alto traffico, e la sua tolleranza ai guasti lo rende adatto anche per scenari in cui l'affidabilità è di primaria importanza.

Sistema NewSQL – CockRoachDB

CockroachDB si posiziona come una soluzione all'avanguardia nell'ambito dei sistemi di database distribuiti. Questo sistema offre un'architettura altamente scalabile e resiliente che si estende su cluster di nodi distribuiti geograficamente. Ciò consente di garantire non solo una maggiore capacità di gestione dei dati, ma anche una ridondanza globale che assicura la disponibilità e la tolleranza ai guasti. Un aspetto

fondamentale di CockroachDB è il suo impegno per la consistenza dei dati, anche in scenari di distribuzione su larga scala.

Utilizzando il concetto di "consistenza seriale", CockroachDB assicura che le transazioni vengano gestite in modo coerente e affidabile, indipendentemente dalla posizione geografica dei nodi. Questo lo rende particolarmente adatto per applicazioni che richiedono un alto grado di precisione e sicurezza nella gestione dei dati, come applicazioni finanziarie e aziendali. Inoltre, CockroachDB supporta il linguaggio SQL, semplificando la migrazione dei sistemi esistenti verso questa piattaforma avanzata. Con una combinazione di scalabilità globale, consistenza dei dati e supporto SQL, CockroachDB si dimostra un candidato solido per soddisfare le sfide dell'era moderna dei dati su vasta scala.

Differenze architetturali e di modellazione dei dati

Le differenze architetturali e di modellazione dei dati tra Cassandra, PostgreSQL e CockroachDB delineano i profili distintivi di queste piattaforme di gestione dati. Cassandra, orientato alle colonne, è progettato per affrontare scenari di elevata scalabilità e disponibilità. La sua struttura distribuita consente l'espansione su cluster di nodi, ottimizzando le prestazioni su volumi di dati enormi e consentendo la ridondanza per la tolleranza ai guasti. Allo stesso modo, CockroachDB si concentra sulla scalabilità globale, ma con un'attenzione particolare alla consistenza dei dati. Utilizzando il modello di "consistenza seriale", CockroachDB garantisce la precisione delle transazioni su una vasta rete di nodi distribuiti. D'altra parte, PostgreSQL mantiene la tradizionale struttura relazionale, risultando una scelta solida per scenari in cui la stabilità, l'integrità dei dati e la coerenza sono fondamentali. La sua flessibilità nell'utilizzo di SQL e la supporto per le transazioni ACID lo rendono una scelta ideale per applicazioni in cui la precisione è prioritaria. In sintesi, mentre Cassandra e CockroachDB enfatizzano la scalabilità e la distribuzione dei dati su larga scala, PostgreSQL si distingue per la sua affidabilità e coerenza nella gestione dei dati strutturati.

Tipi di progetti adatti a ciascun sistema

I progetti destinati a sistemi SQL, NoSQL e NewSQL coprono un vasto spettro di applicazioni e necessità nell'ambito della gestione dei dati. Per quanto riguarda i sistemi SQL, progetti legati a gestionali aziendali, sistemi di contabilità e software per la gestione delle risorse umane traggono beneficio dalla struttura e dalla coerenza dei database relazionali. D'altra parte, i progetti NoSQL trovano la loro forza in scenari ad alta scalabilità e flessibilità strutturale, adatti ad esempio per applicazioni social. Infine, i sistemi NewSQL si dimostrano ideali per progetti che richiedono una scalabilità orizzontale avanzata e alte prestazioni in ambienti distribuiti, come le applicazioni finanziarie o le piattaforme di analisi in tempo reale, con esempi di progetti come Google Spanner e CockroachDB. In ultima analisi, la scelta del tipo di sistema dipenderà dalle esigenze specifiche del progetto, dalla scalabilità richiesta e dalla complessità delle interazioni dei dati.

Scenari applicativi

Per comprendere appieno l'ambito e l'impatto del progetto in esame, è essenziale esaminare da vicino i dataset utilizzati. Ogni dataset rappresenta un'entità distintiva, offrendo un insieme unico di dati che riflette specifici scenari e contesti. Nel prosieguo di questo paragrafo, verranno esplorati in dettaglio i differenti dataset impiegati, analizzando le fonti, le caratteristiche e la rilevanza di ciascuno di essi all'interno del quadro generale del progetto.

Tutti i dataset sono reperiti da Kaggle, una piattaforma che offre una vasta gamma di dataset adattabili a diverse esigenze. Di seguito, verranno elencati in ordine crescente di dimensione.

Formula 1

Il dataset "Formula 1" rappresenta una fonte ricca di dati nel contesto delle corse automobilistiche Formula 1. Suddiviso in tre tabelle, offre una panoramica completa delle informazioni legate a piloti, gare e risultati.

La tabella "drivers" dettaglia i dati dei piloti, fornendo informazioni come il loro identificatore, il numero di gara, il nome, il cognome, la data di nascita, la nazionalità e un URL di riferimento.

La tabella "races" presenta dettagli sulle gare, compresi l'identificatore della gara, l'anno, l'identificatore del circuito, la data, l'orario e ulteriori dettagli temporali delle sessioni di pratica, qualifiche e gara.

Infine, la tabella "results" fornisce dati rilevanti relativi ai risultati, inclusi l'identificatore del risultato, l'identificatore della gara e del pilota, l'identificatore del costruttore, la posizione di partenza e di arrivo, i punti ottenuti, il tempo di gara, il giro più veloce, e ulteriori dettagli sullo stato.

Questo dataset offre un'opportunità di esplorare e analizzare i dati relativi al mondo della Formula 1, consentendo di tracciare prestazioni, tendenze e sviluppi nel corso delle stagioni. È il primo dataset in ordine di dimensioni, attestandosi sulle 25mila righe nella tabella “results”.

	resultid [PK] integer	raceid integer	driverid integer	constructorid integer	number character varying (3)	grid integer	position character varying (3)
1	1	18	1	1	22	1	1
2	2	18	2	2	3	5	2
3	3	18	3	3	7	7	3
4	4	18	4	4	5	11	4
5	5	18	5	1	23	3	5

Cities, States e Countries

Il dataset "Cities, States e Countries" rappresenta un compendio di informazioni geografiche e demografiche che riguardano città, stati e nazioni in tutto il mondo. Esso è organizzato in tre tabelle principali: "cities", "states" e "countries".

La tabella "cities" comprende attributi come l'identificativo univoco della città, il nome, l'identificativo dello stato, il codice dello stato, l'identificativo del paese, il codice del paese, le coordinate di latitudine e longitudine e l'ID wiki associato.

La tabella "states" contiene informazioni sulle “regioni”, inclusi il loro identificativo, nome, identificativo del paese, codice del paese, codice dello stato, tipo di stato, coordinate geografiche e altro.

Infine, la tabella "countries" riporta dati relativi ai paesi, tra cui l'identificativo, il nome, i codici ISO3 e ISO2, il codice numerico, il prefisso telefonico, la capitale, la valuta, il nome della valuta, il simbolo della valuta.

Questo dataset offre una visione dettagliata e strutturata delle relazioni tra città, stati e paesi, fornendo una base di dati completa per analisi geografiche e studi comparativi. La tabella con il maggior numero di righe è “cities”, con un totale di 150mila righe.

	id [PK] integer	name character varying (59)	state_id integer	state_code character varying (5)	country_id integer	country_code character varying (2)	latitude character varying
1	1	Andorra la Vella	488	07	6	AD	4250779000
2	2	Arinsal	493	04	6	AD	4257205000
3	3	Canillo	489	02	6	AD	4256760000
4	4	El Tarter	489	02	6	AD	4257952000
5	5	Encamp	487	03	6	AD	4253474000

Boston Crimes

Il dataset "Boston Crimes" rappresenta un'importante risorsa per l'analisi e la comprensione delle attività criminali all'interno della città di Boston. Esso è costituito da due tabelle fondamentali: "crimes" e "offense_codes". La tabella "crimes" comprende una serie di attributi chiave, tra cui "INCIDENT_NUMBER" per l'identificazione degli incidenti, "OFFENSE_CODE" e "OFFENSE_DESCRIPTION" per categorizzare e descrivere gli illeciti commessi. Inoltre, vengono forniti dettagli temporali come "YEAR", "MONTH", "DAY_OF_WEEK" e "HOUR", insieme alle coordinate di geolocalizzazione "Lat" e "Long".

La tabella "offense_codes" associa codici specifici a nomi di reato, fornendo una mappatura esaustiva delle categorie criminali coinvolte. Con questo set di dati, è possibile condurre analisi approfondite sulla distribuzione temporale e spaziale dei reati, nonché esaminare i pattern delle attività criminali in relazione ai vari fattori.

Le informazioni presenti offrono un'opportunità unica per valutare strategie di prevenzione e controllo dei reati all'interno della comunità di Boston. La tabella con più righe all’interno dell’intero progetto è “crimes”, con circa 320mila righe.

	incident_number character varying (13)	offense_code integer	offense_code_group character varying (41)	offense_description character varying (58)
1	I182070945	619	Larceny	LARCENY ALL OTHERS
2	I182070943	1402	Vandalism	VANDALISM
3	I182070941	3410	Towed	TOWED MOTOR VEHICLE
4	I182070940	3114	Investigate Property	INVESTIGATE PROPERTY
5	I182070938	3114	Investigate Property	INVESTIGATE PROPERTY

Caso di studio: Applicazione pratica

Il progresso del progetto si basa sulla valutazione dei comportamenti dei vari sistemi precedentemente introdotti, in relazione a differenti scenari applicativi. L'obiettivo primario consiste nell'analizzare le prestazioni offerte da ciascun sistema in condizioni specifiche.

Per raggiungere questo scopo, sono stati selezionati tre casi d'uso per ciascun dataset, ognuno dei quali richiede l'esecuzione di una query specifica. Questi casi d'uso sono strutturati in modo da aumentare gradualmente la complessità, introducendo costrutti che pongono un maggior carico computazionale sul sistema al fine di valutarne il comportamento.

Di seguito vengono presentate le query impiegate in questo progetto, seguite da relative spiegazioni(la stessa notazione Q1, Q2, Q3 è utilizzata anche nei grafici):

Dataset Formula 1

Q1:
```
SELECT DISTINCT  d.forename, d.surname
FROM  drivers d
JOIN results r ON d.driverId = r.driverId
WHERE
        r.position = 1;
```

➔ *Ottieni l'elenco dei nomi completi dei piloti che hanno ottenuto almeno una vittoria.*

Q2:
```
SELECT ra.name AS race_name, ra.year, CONCAT(d.forename, ' ', d.surname) AS driver_name,
        r.fastestLapTime, r.fastestLapSpeed
 FROM  races ra
 JOIN results r ON ra.raceId = r.raceId
 JOIN drivers d ON r.driverId = d.driverId;
```

➔ *Ottieni i dettagli delle gare, mostrando il pilota che ha ottenuto il giro più veloce e la sua velocità massima in ogni gara.*

Q3:
```
SELECT  d.driverRef, ra.year, AVG(r.points) AS average_points
FROM  drivers d
JOIN results r ON d.driverId = r.driverId
JOIN races ra ON r.raceId = ra.raceId
GROUP BY d.driverRef, ra.year;
```

➔ *Ottieni la media dei punti ottenuti da ciascun pilota in ciascun anno.*

Dataset Cities, States & Countries

Q1:
```
SELECT c.name AS country_name, c.capital
FROM countries c
WHERE c.region = 'Europe';
```

➔ *Ottieni il nome e la capitale dei paesi che si trovano nella regione "Europe".*

Q2:
```
SELECT c.name AS country_name, c.capital
FROM countries c
WHERE c.region = 'Europe'
AND EXISTS (
        SELECT 1
        FROM cities ci
        WHERE ci.country_id = c.id AND ci.longitude < '0'
);
```

➔ *Ottieni il nome e la capitale dei paesi nell'Europa che hanno almeno una città con longitudine negativa.*

Q3:
```
SELECT cities.name AS city_name, states.name AS state_name,
        countries.name AS country_name, countries.currency
FROM cities
JOIN states ON cities.state_id = states.id
JOIN countries ON states.country_id = countries.id;
```

➔ *Seleziona il nome delle città e il nome dello stato in cui si trovano, insieme al nome del paese e la valuta utilizzata.*

Dataset Boston Crimes

Q1:
```
SELECT INCIDENT_NUMBER, OFFENSE_CODE, OFFENSE_DESCRIPTION FROM crimes;
```

➔ *Seleziona il numero di incidente, il codice di reato e la descrizione del reato dalla tabella "crimes".*

Q2:
```
SELECT DAY_OF_WEEK, COUNT(*) AS num_crimes
FROM crimes
GROUP BY DAY_OF_WEEK
ORDER BY num_crimes DESC;
```

➔ *Trova il numero di reati per ciascun giorno della settimana, ordinati dal più alto al più basso.*

Q3:
```
SELECT c1.INCIDENT_NUMBER AS incident1, c2.INCIDENT_NUMBER AS incident2
FROM crimes AS c1
JOIN crimes AS c2 ON c1.OCCURRED_ON_DATE = c2.OCCURRED_ON_DATE
AND c1.DISTRICT = c2.DISTRICT
WHERE c1.INCIDENT_NUMBER <> c2.INCIDENT_NUMBER;
```

➔ *Trova tutte le coppie di reati che sono stati commessi nella stessa data e nello stesso distretto.*

Implementazione in sistemi SQL, NoSQL e NewSQL - Grafici e risultati

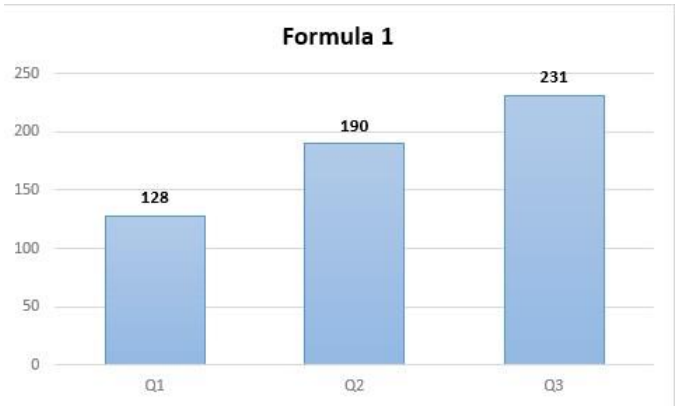
Questo paragrafo si concentra sulla presentazione dei grafici e dei risultati emersi dall'analisi introdotta nei paragrafi precedenti. Inizialmente, l'attenzione è dedicata all'analisi dei tempi di esecuzione ottenuti singolarmente da ciascun sistema. Successivamente, l'attenzione si sposta verso una visione panoramica, considerando tutti e tre i sistemi esaminati, al fine di stabilire in quali circostanze ognuno di essi garantisce le migliori prestazioni.

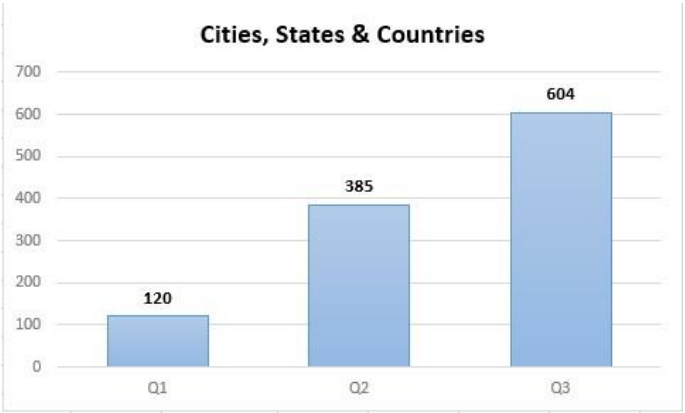
PostgreSQL

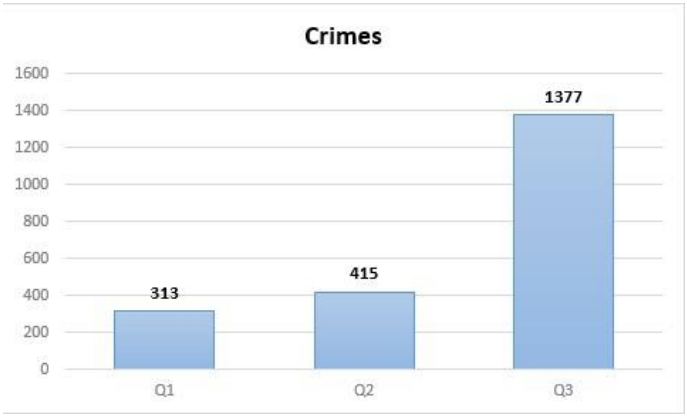
Risultati

	Formula 1	**Cities**	**Crimes**
Query1	128	120	313
Query2	190	385	415
Query3	231	604	1377

(Tempi espressi in millisecondi)







Interazione con l'utente

PostgreSQL si distingue per la sua notevole facilità di utilizzo. La sua interfaccia intuitiva e ben progettata semplifica le operazioni di gestione e interrogazione dei dati. L'adozione di un linguaggio SQL standard rende la creazione e l'esecuzione di query un processo fluido. Gli strumenti di amministrazione e monitoraggio integrati semplificano il controllo delle prestazioni e la gestione delle configurazioni. Inoltre, la flessibilità di PostgreSQL nell'adattarsi a diverse esigenze è evidente nell'ampia gamma di estensioni e plugin disponibili per personalizzare il sistema in base alle necessità specifiche.

Cassandra

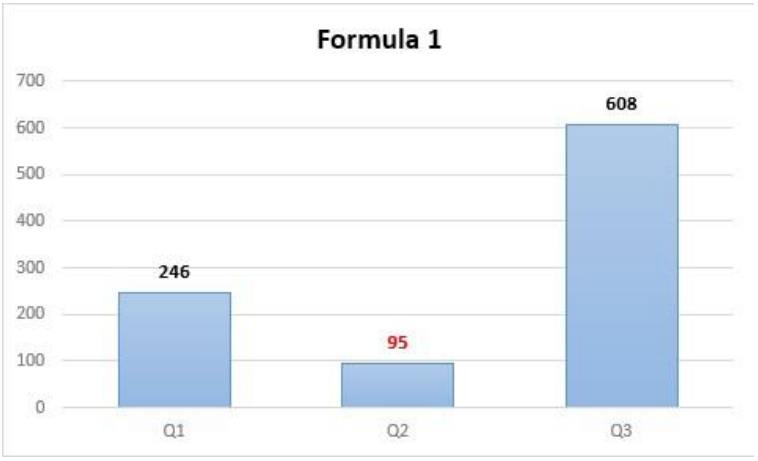
Risultati

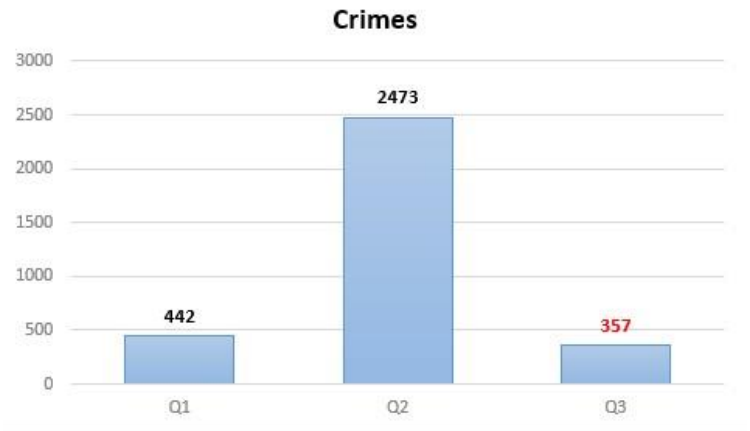
	Formula 1	**Cities**	**Crimes**
Query1	246	374	442
Query2	95	462	2473
Query3	608	126	357

(Tempi espressi in millisecondi)

Le celle evidenziate con colorazioni distintive indicano una notevole convenienza in termini di tempi di esecuzione, poiché è stato necessario adattare le query per permetterne l'esecuzione.

Come illustrato nel paragrafo "Tecnologie Utilizzate", va tenuto presente che Cassandra, in quanto sistema NoSQL, non supporta operazioni di JOIN. Pertanto, è stato indispensabile creare tabelle contenenti tutte le informazioni richieste per ottenere il risultato desiderato. Questo approccio consente al sistema di generare risultati in tempi estremamente brevi. Tali aspetti devono essere considerati attentamente durante l'analisi generale dei risultati (questi valori sono indicati in rosso all'interno dei grafici).





Interazione con l'utente

Cassandra, noto per la sua scalabilità e tolleranza ai guasti, offre indubbiamente vantaggi significativi nel campo delle basi di dati distribuite. Tuttavia, quando si tratta di valutare la sua facilità d'uso, emergono alcune sfide degne di nota. La modellazione dei dati, che differisce dalle tradizionali basi di dati relazionali, potrebbe risultare controintuitiva per coloro che sono abituati a schemi più rigidi. Inoltre, le query in Cassandra richiedono un approccio specifico, in quanto le operazioni di join e alcune interrogazioni complesse(come il GROUP BY effettuato su un campo non chiave) non sono nativamente supportate.

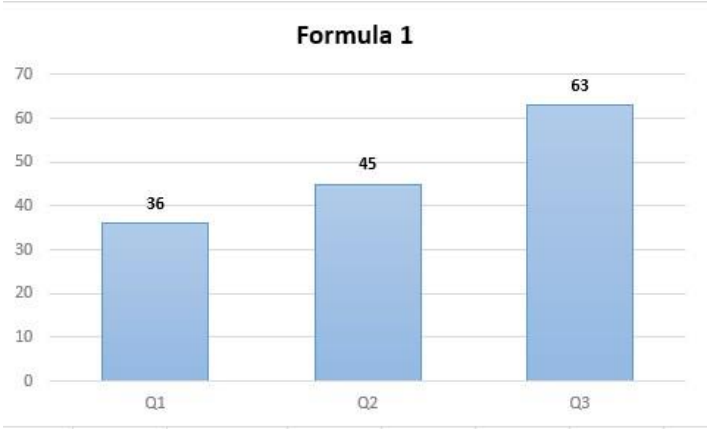
Questa limitazione potrebbe comportare la necessità di ridisegnare le query e adottare strategie alternative per raggiungere gli stessi risultati.

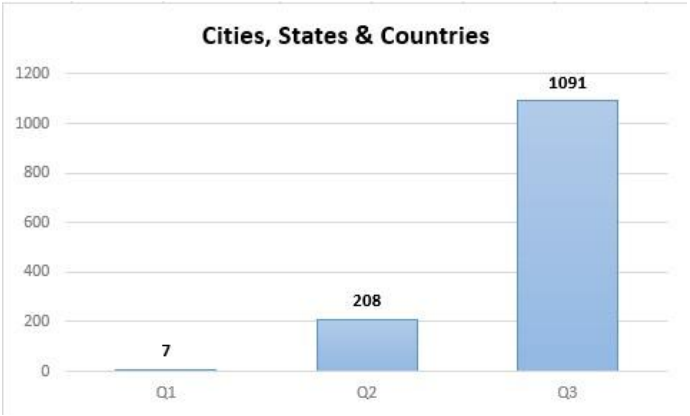
CockroachDB

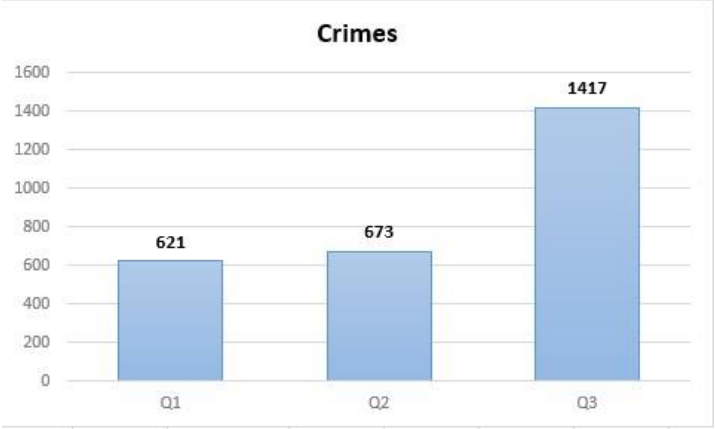
Risultati

	Formula 1	Cities	Crimes
Query1	36	7	621
Query2	45	208	673
Query3	63	1091	1417

(Tempi espressi in millisecondi)







Interazione con l’utente

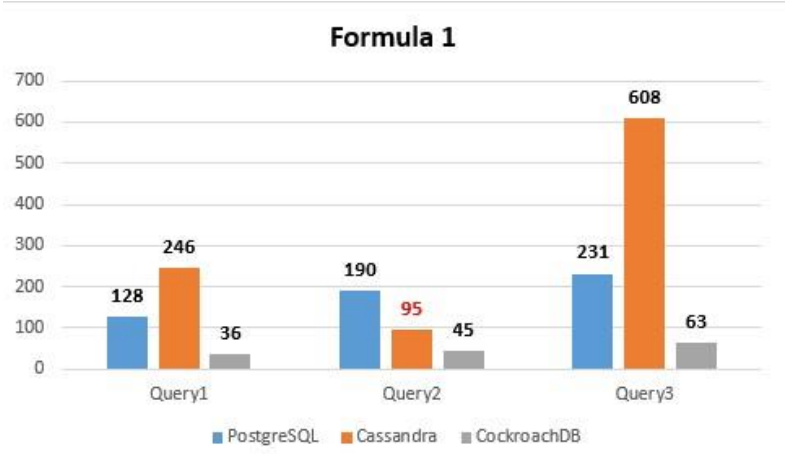
CockroachDB, analogamente a PostgreSQL, offre la possibilità di utilizzare il linguaggio SQL all'interno delle query, consentendo l'uso di istruzioni JOIN e altre istruzioni.

La configurazione di un cluster è agevolata da un'interfaccia web, attraverso la quale è possibile avviare il processo. Successivamente, è possibile collegarsi al cluster tramite una shell utilizzando istruzioni dettagliate fornite direttamente dalla piattaforma. Sebbene CockroachDB non presenti un'interfaccia grafica altrettanto intuitiva come quella di PostgreSQL, offre comunque una documentazione online molto esaustiva, superando in questo senso la documentazione disponibile per Cassandra.

Confronto Generale

In questo paragrafo, vengono condotti confronti approfonditi tra i tre distinti sistemi in termini di tempi di esecuzione delle query precedentemente delineate. Ogni dominio di applicazione è illustrato attraverso un grafico comparativo, che evidenzia le performance. Successivamente, i risultati ottenuti vengono analizzati per identificare tendenze e differenze significative nei tempi di esecuzione. L'obiettivo è presentare un'analisi oggettiva delle prestazioni di ciascun sistema e delle loro implicazioni nei diversi contesti applicativi.

Formula 1



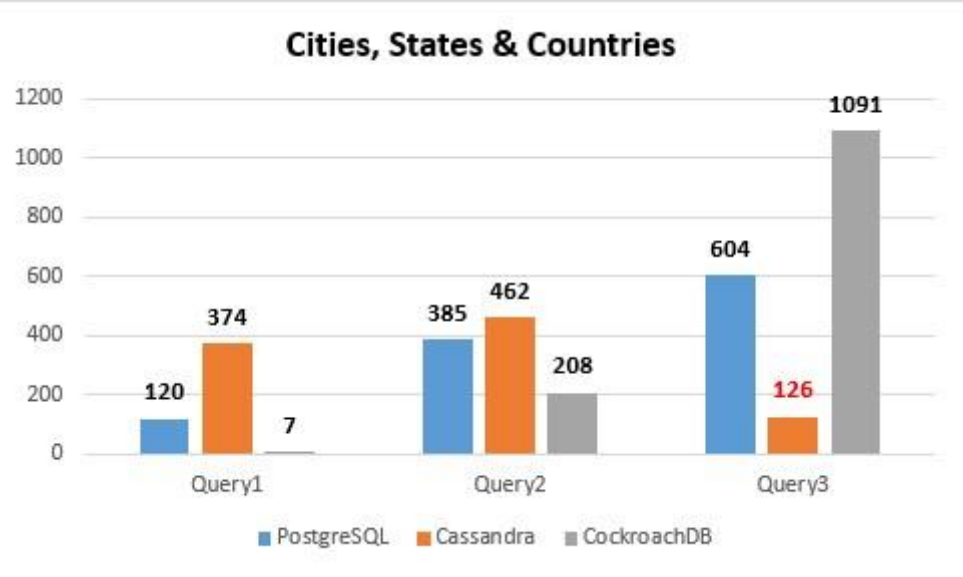
Nella prima query, si nota che CockroachDB ha il tempo di esecuzione più breve, seguito da PostgreSQL e infine da Cassandra. CockroachDB sembra eccellere in questa tipologia di query, garantendo una performance notevolmente migliorata rispetto agli altri due sistemi.

Nella seconda va notato che Cassandra presenta un tempo di esecuzione eccezionalmente basso rispetto ai suoi standard. Tuttavia, bisogna considerare che il tempo è stato ottenuto da una query adattata al fine di poter essere eseguita. CockroachDB mantiene ancora una performance competitiva, mentre PostgreSQL è leggermente più lento.

Anche nella terza query, CockroachDB dimostra tempi di esecuzione notevolmente più brevi rispetto agli altri due sistemi. PostgreSQL e Cassandra presentano tempi più lunghi, con Cassandra che risulta essere il più lento tra i tre.

È evidente che i tempi di esecuzione variano considerevolmente tra i sistemi analizzati e anche tra le diverse query. CockroachDB sembra emergere come un contendente forte, spesso registrando i tempi di esecuzione più brevi.

Cities, States & Countries



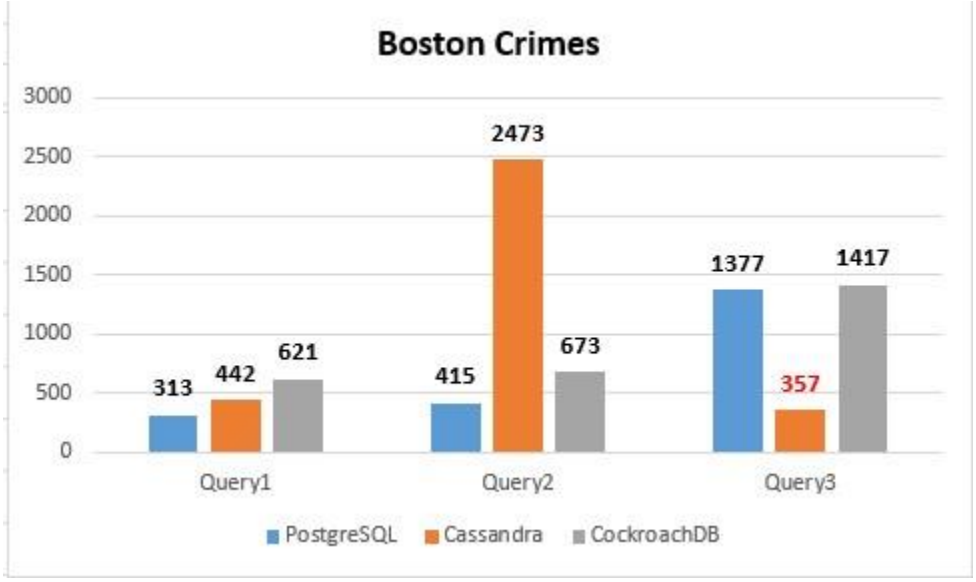
Nella prima query, CockroachDB si distingue per la performance migliore con un tempo di esecuzione di soli 7 ms. PostgreSQL segue con 120 ms, mentre Cassandra richiede 374 ms. CockroachDB sembra eccellere in questo caso specifico, mentre anche PostgreSQL garantisce una buona performance.

Nella seconda, CockroachDB presenta tempi di esecuzione notevolmente inferiori rispetto agli altri due sistemi, richiedendo 208 ms. PostgreSQL richiede 385 ms, mentre Cassandra è leggermente più lento con 462 ms.

La terza query mostra un risultato particolarmente interessante. Il risultato di Cassandra è segnalato in rosso a causa della necessità di dover eseguire una query modificata. Va notato che la sua esecuzione molto rapida (126 ms) è notevole rispetto a PostgreSQL (604 ms) e CockroachDB (1091 ms). Ciò suggerisce che Cassandra potrebbe avere un vantaggio in questa query specifica.

In generale, CockroachDB sembra essere il sistema con i tempi di esecuzione più bassi in diverse query, seguito da PostgreSQL.

Boston Crimes



In termini di prestazioni per la prima query, PostgreSQL dimostra il tempo di esecuzione più veloce, con un valore di 313 ms. CockroachDB segue con 621 ms, mentre Cassandra mostra un risultato intermedio di 442 ms. Questo suggerisce che PostgreSQL è in grado di gestire questa query in modo più efficiente rispetto agli altri due sistemi.

Nella seconda, CockroachDB registra il tempo di esecuzione intermedio, con 673 ms, mentre PostgreSQL e Cassandra presentano valori rispettivamente di 415 ms e 2473 ms. PostgreSQL sembra essere ottimizzato per questa query specifica, mentre Cassandra evidenzia un tempo significativamente più lungo.

Anche nel contesto della terza query, il tempo associato a Cassandra è evidenziato in rosso. Questa specifica query coinvolge un SELF JOIN di una tabella, un'operazione che non è realizzabile in Cassandra. Di conseguenza, il tempo di esecuzione risulta significativamente più basso rispetto agli altri due sistemi. Al contrario, sia PostgreSQL che CockroachDB presentano tempi molto simili per questo tipo di query, suggerendo che gestiscono questa operazione complessa in modo comparabile.

In generale, i dati suggeriscono che le prestazioni variano a seconda della query e del sistema di gestione dei dati utilizzato. PostgreSQL sembra essere il più coerente in termini di prestazioni veloci, ma CockroachDB dimostra di poter eccellere in alcune query specifiche.

Conclusioni

In base ai dati analizzati, emerge che le prestazioni dei sistemi di gestione dei database variano notevolmente a seconda delle query e dei contesti d'uso. Tuttavia, è possibile trarre alcune considerazioni conclusive per determinare quando potrebbe essere più opportuno utilizzare ciascun sistema.

- **CockroachDB**: Questo sistema si è dimostrato particolarmente efficiente in termini di tempi di esecuzione nelle diverse query analizzate. Se l'obiettivo principale fosse ottenere prestazioni rapide e consistenti in un'ampia gamma di query, CockroachDB potrebbe essere una scelta solida.
- **PostgreSQL**: PostgreSQL ha dimostrato prestazioni rapide in alcune query e una buona adattabilità ad altre. È una scelta affidabile quando è richiesta una gestione versatile dei dati, soprattutto in scenari in cui la performance è critica ma potrebbe non essere l'aspetto dominante.
- **Cassandra**: Nonostante alcuni tempi di esecuzione eccezionalmente bassi siano stati ottenuti con query adattate, bisogna considerare che Cassandra ha limitazioni in alcune tipologie di operazioni, come i JOIN complessi. È particolarmente adatto per scenari in cui la scalabilità orizzontale e la disponibilità dei dati sono prioritari rispetto alle prestazioni in singole query.

In conclusione, la scelta del sistema di gestione dei database dovrebbe essere basata sulle esigenze specifiche dell'applicazione. Se si mirasse a prestazioni consistenti e veloci in una varietà di situazioni, CockroachDB potrebbe essere la migliore opzione. PostgreSQL è ideale quando la flessibilità e l'adattabilità sono cruciali. Cassandra, d'altro canto, brilla in scenari di grande scalabilità e distribuzione dei dati, se si può lavorare attorno alle sue limitazioni. La decisione finale dovrebbe derivare da una valutazione approfondita delle esigenze, delle capacità e delle limitazioni di ciascun sistema nel contesto dell'applicazione specifica.