Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Migrazione di un data warehouse in ambiente cloud multi-tenant: benchmarking e proof-of-concept

Elaborato in: Business Intelligence

Relatore:
Chiar.mo Prof.
Stefano Rizzi

Presentata da:
Eddie Barzi

Sessione IV
Anno Accademico 2023-2024

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Nell'era della digitalizzazione, le aziende si ritrovano a produrre e a dover gestire una quantità sempre crescente di dati, che devono essere analizzati e trasformati in informazioni utili per supportare il processo decisionale. La Business Intelligence gioca un ruolo chiave in questo contesto, consentendo di estrarre conoscenza dai dati attraverso strumenti e metodologie avanzate.

Per aziende che forniscono servizi in ambito Business Intelligence a più clienti, l'adozione di un'architettura cloud multi-tenant rappresenta una scelta strategica vantaggiosa sotto molteplici aspetti. Questo approccio consente di ottimizzare l'utilizzo delle risorse, ridurre i costi operativi, migliorare la scalabilità e garantire una maggiore efficienza nella gestione dei dati.

Il lavoro di tesi, svolto nell'ambito di un tirocinio presso un'azienda del territorio, ha l'obiettivo di analizzare il processo di migrazione di un data warehouse da un ambiente single-tenant a un'architettura cloud multi-tenant, valutando le prestazioni di diversi DBMS attraverso dei benchmarking e sviluppando una Proof of Concept per validare l'efficacia della soluzione implementata.

La struttura di questo documento è la seguente:

Nel **capitolo 1** viene fornita una panoramica sulla Business Intelligence, illustrando le principali metodologie, tecnologie, concetti e aspetti da considerare in ambito cloud computing e multi-tenant.

Nel **capitolo 2** vengono presi in esame quattro DBMS: *Microsoft SQL Server*, *Amazon Redshift*, *Citus PostgreSQL* e *Apache Doris*. La scelta del sistema più adatto non deve solo risolvere le problematiche riscontrate nella

soluzione single-tenant, ma anche rispettare i requisiti richiesti per la migrazione verso un ambiente cloud multi-tenant. Per questo motivo, viene condotto un confronto tra le diverse soluzioni, evidenziandone le caratteristiche principali. In particolare, tutti e quattro i DBMS vengono inizialmente sottoposti a un benchmark per valutarne le prestazioni nell'esecuzione di una singola query OLAP in assenza di concorrenza. Successivamente, viene eseguito un test di carico per analizzare le prestazioni in presenza di più tenant, considerando scalabilità e gestione della concorrenza al crescere del numero di utenti.

Nel **capitolo 3** ci si concentra su Apache Doris, il DBMS che ha registrato le migliori prestazioni nei benchmark. Dopo un approfondimento sulle sue caratteristiche e funzionalità, viene descritto il processo di modellazione dello schema a stella, adattato alle specifiche di Doris. Viene studiato anche il processo di ETL, il quale si evolve in ELT. Infine, vengono illustrate alcune dashboard di esempio per dimostrare la corretta operatività del sistema.

Capitolo 1

Background

L'obiettivo di ogni azienda privata è il profitto, mentre quello di ogni ente pubblico è offrire il miglior servizio possibile al minor costo. In entrambi i contesti, il raggiungimento di questi obiettivi dipende sempre più dai dati e dalla loro analisi. Le decisioni aziendali possono essere migliorate notevolmente attraverso un uso consapevole e strategico delle informazioni, basandosi su dati quantitativi anziché su semplici intuizioni qualitative.

Con l'aumento esponenziale dei dati generati quotidianamente, l'informatica ha assunto nel tempo un ruolo sempre più significativo all'interno delle aziende. All'inizio la funzione principale dell'informatica era quella di memorizzare dati operazionali; veniva utilizzata quindi solo come supporto per rendere più rapide ed economiche le operazioni, ma non aveva un ruolo decisivo nel business e non creava di per sé ricchezza. Con il tempo si è capito che l'informatica ha il potenziale per rivoluzionare il business e ha di conseguenza assunto un duplice ruolo: sia essere di supporto nella gestione e nella registrazione delle operazioni, sia essere un fattore decisivo per le scelte, l'organizzazione e i processi aziendali.

Tuttavia la grande mole di dati generati quotidianamente è spesso troppo dettagliata e complessa affinché la parte alta dell'organigramma aziendale, come i direttori e i manager, possa utilizzarli in ottica decisionale e di business. Per tradurre questa mole di dati grezzi in informazioni utili e supporta-

re efficacemente le analisi manageriali, è necessario predisporre un'apposita infrastruttura hardware e software, nota come piattaforma di *Business Intelligence* (BI). Questa infrastruttura comprende hardware dedicato, infrastrutture di rete, sistemi di gestione dei database (DBMS), software di back-end per l'elaborazione dei dati e software di front-end per la visualizzazione e l'analisi.

1.1 Business Intelligence

Con Business Intelligence si fa riferimento a un insieme di strumenti e procedure che consentono di trasformare i dati in informazioni e quindi in conoscenza. Dati, informazioni e conoscenza non sono la stessa cosa:

- **Dati:** Si tratta di dati operazionali o transazionali, generati quotidianamente da transazioni aziendali. Sono caratterizzati da un'elevata quantità ma da una qualità spesso scarsa, con presenza di errori.
- **Informazioni:** Rappresentano il distillato dei dati, ottenuto tramite processi di pulizia, aggregazione e sintesi. Rispetto ai dati, le informazioni sono ridotte in volume ma di valore più elevato.
- **Conoscenza:** Costituisce il livello più alto a cui si può aspirare e rappresenta il risultato dell'interpretazione delle informazioni. La conoscenza è essenziale per prendere decisioni strategiche.

1.1.1 La piramide della Business Intelligence

La piramide della Business Intelligence (fig. 1.1) rappresenta i livelli di trasformazione e utilizzo dei dati all'interno di un'organizzazione, a partire dalle fonti dati fino alle decisioni.

Ogni livello della piramide riflette un passo nel processo di elaborazione e valorizzazione dei dati. È infatti possibile notare come, partendo dal basso e procedendo verso l'alto, la quantità di dati si riduca a favore di un maggiore valore.

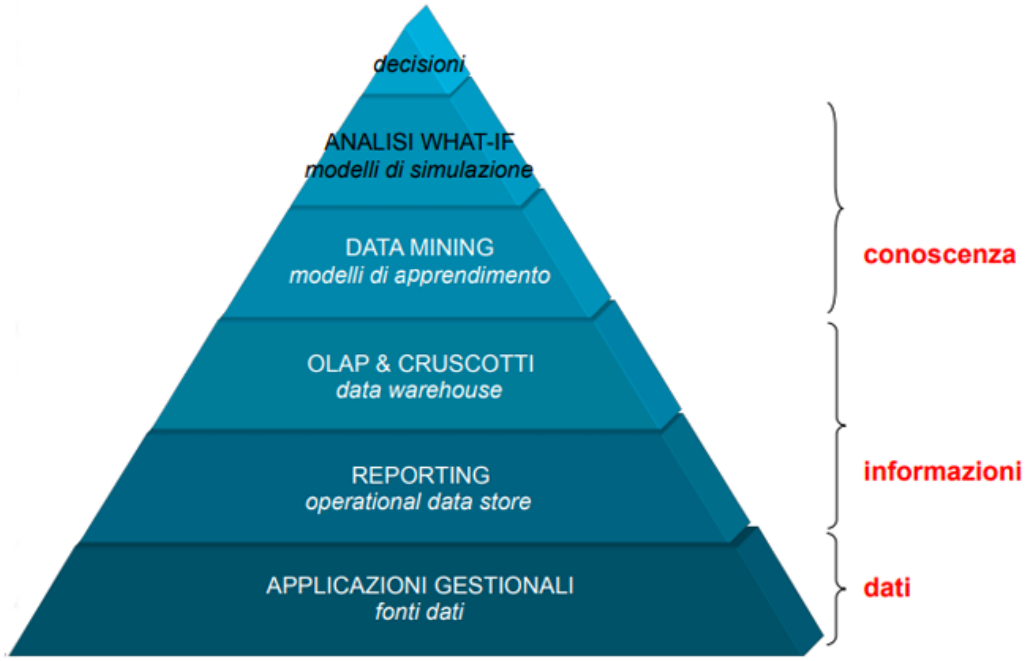

Figura 1.1: Piramide della BI

- **Applicazioni gestionali (fonti dati):** Questo è il livello più basso della piramide, dove i dati grezzi vengono raccolti dalle diverse applicazioni aziendali, come ERP (*Enterprise Resource Planning*), CRM (*Customer Relationship Management*) e altre fonti operative. Questi dati sono non strutturati o parzialmente strutturati e necessitano di trasformazione per essere utilizzati.

- **Reporting (Operational Data Store):** In questo livello, i dati raccolti vengono consolidati e strutturati in un *Operational Data Store* (ODS), consentendo la generazione di report. Questo livello è essenziale per fornire una visione corretta e integrata dei dati aziendali.

- **OLAP e cruscotti (Data Warehouse):** I dati vengono caricati in un Data Warehouse, dove possono essere analizzati tramite strumenti OLAP (*Online Analytical Processing*) e visualizzati su dashboard interattive. Questo livello consente analisi multidimensionali e offre supporto per comprendere trend e performance.

- **Data Mining (modelli di apprendimento):** A un livello più avanzato, le tecniche di *Data Mining* permettono di scoprire pattern nascosti nei dati. Attraverso algoritmi di apprendimento automatico, è possibile prevedere tendenze future o identificare comportamenti anomali.

- **Analisi What-If (modelli di simulazione):** Questo livello consente la simulazione di scenari ipotetici per valutare l'impatto di decisioni strategiche. I modelli di simulazione supportano analisi predittive e scenari complessi.

- **Decisioni:** Al vertice della piramide si trova il livello decisionale, dove i dati elaborati e le conoscenze estratte vengono utilizzati per guidare le scelte strategiche aziendali. Questo livello rappresenta l'obiettivo finale della BI: trasformare i dati in valore per il business.

La piramide evidenzia chiaramente il flusso dei dati, ad ogni livello si aggiunge valore e complessità. Mentre i livelli inferiori si concentrano sull'e-

laborazione e la gestione dei dati, quelli superiori si focalizzano sull'analisi e sul supporto decisionale [1].

1.2 Data Warehouse

Un Data Warehouse può essere visto come un contenitore (*repository*) di dati. Esso contiene dati provenienti da fonti di varia natura e li mette a disposizione per l'analisi. Il concetto di Data Warehouse (DW) è stato introdotto e formalizzato da Inmon [2], che lo definisce come:

> "Una collezione di dati di supporto per il processo decisionale che presenta le seguenti caratteristiche: è orientato ai soggetti di interesse, è integrato e consistente, è rappresentativo dell'evoluzione temporale e non volatile."

Questa definizione stabilisce i quattro principi fondamentali di un DW. Orientato ai soggetti vuol dire che il DW è strutturato intorno ai temi principali dell'azienda, come vendite, clienti e prodotti, piuttosto che su specifiche applicazioni. Integrato e consistente significa che i dati vengono trattati per garantire la restituzione di una visione unificata degli stessi, in quanto il DW si basa su dati provenienti da fonti eterogenee. Mentre i dati operazionali tipicamente coprono un arco temporale limitato, il DW deve registrare le informazioni in modo da permettere l'analisi nel tempo, includendo la storicità dei dati. Una volta caricati nel DW, in linea di principio i dati non vengono più cancellati, assicurando stabilità e affidabilità per l'analisi.

1.2.1 Architetture del Data Warehouse

Le architetture definiscono l'infrastruttura tecnologica necessaria per raccogliere, elaborare e presentare le informazioni in modo efficace. Di seguito viene fatta una classificazione delle architetture basandosi sul numero di livelli:

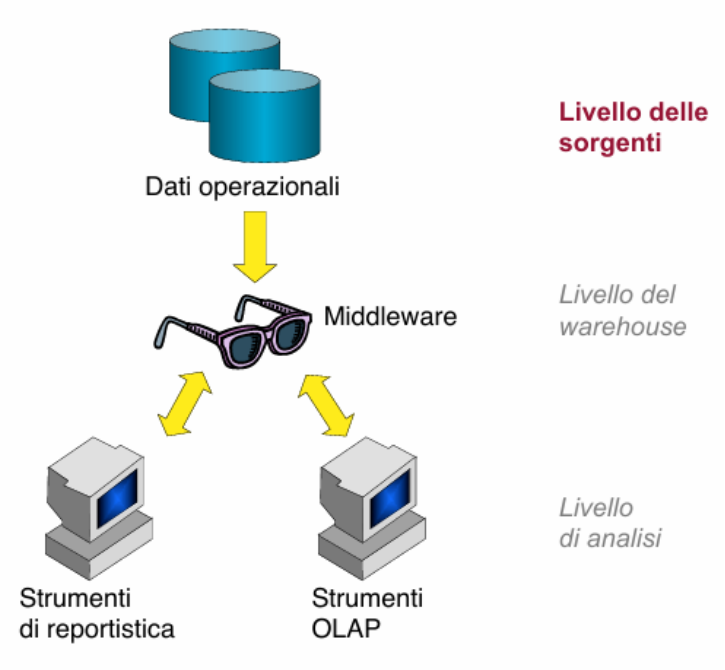

Figura 1.2: Architettura a un livello

Architettura a un livello Qui l'unico livello fisico è il livello delle sorgenti (fig. 1.2). Il livello del DW viene ricreato in maniera virtuale con delle viste sui dati operazionali. Questa architettura è semplice da implementare e non prevede duplicazione di dati, ma non soddisfa il requisito di mantenere il più possibile separate l'elaborazione analitica e quella transazionale. Le interrogazioni analitiche (OLAP) vengono eseguite direttamente sui database transazionali (OLTP), creando carichi di lavoro aggiuntivi.

Architettura a due livelli In questa architettura, oltre al livello delle sorgenti, viene reso fisico anche il livello del DW (fig. 1.3). Le sorgenti possono essere di varia natura e i dati vengono estratti, trasformati e caricati sul DW mediante un processo di *Extract, Transform, Load* (ETL). Il DW è basato sul modello multidimensionale ed è quindi ottimizzato per interrogazioni analitiche.

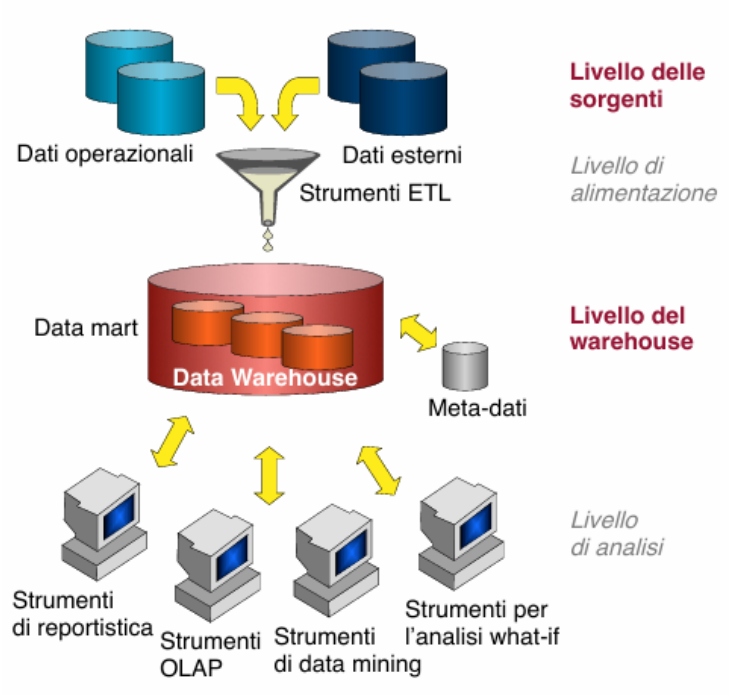

Figura 1.3: Architettura a due livelli

Architettura a tre livelli Qui si rende fisico anche il livello di alimentazione, il processo di ETL non è più una procedura ma si appoggia a un ODS (fig. 1.4). L'ODS consiste in un database con dati operazionali integrati, consistenti, corretti, volatili, correnti e dettagliati. Il suo utilizzo consente di semplificare il processo ETL, suddividendolo in due fasi: una prima fase di pulizia e integrazione dei dati e una seconda fase di aggregazione per il caricamento nel DW. Inoltre, l'ODS rappresenta una soluzione ottimale per la reportistica operativa.

1.2.2 Modello multidimensionale

Il modello multidimensionale rappresenta uno degli approcci più consolidati e intuitivi per la rappresentazione e l'analisi dei dati all'interno di un DW. I concetti principali di questo modello sono *fatti*, *misure*, *dimensioni* e *gerarchie*. I fatti rappresentano gli eventi aziendali di interesse, fenomeni

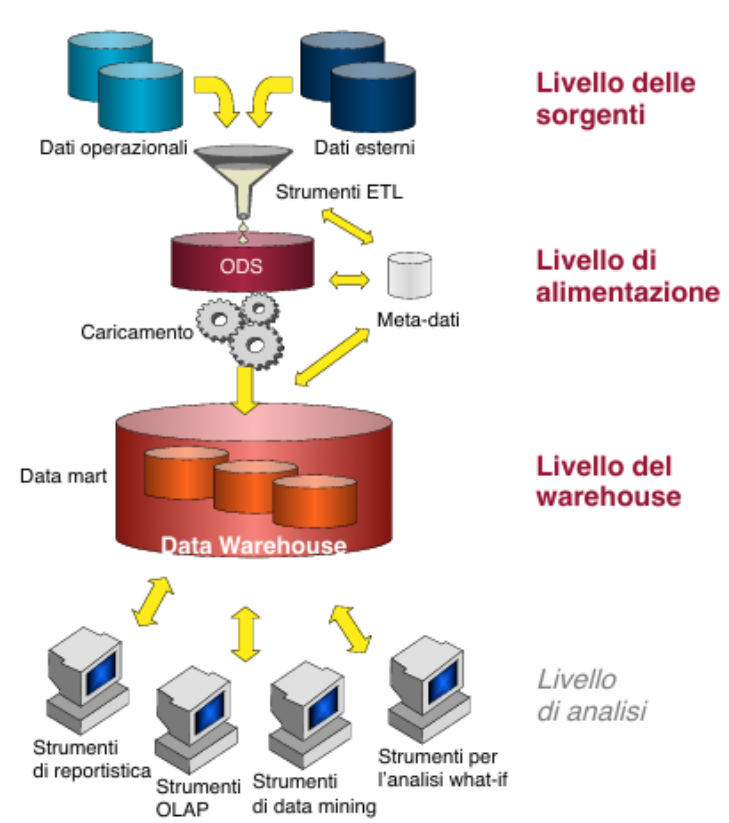

Figura 1.4: Architettura a tre livelli

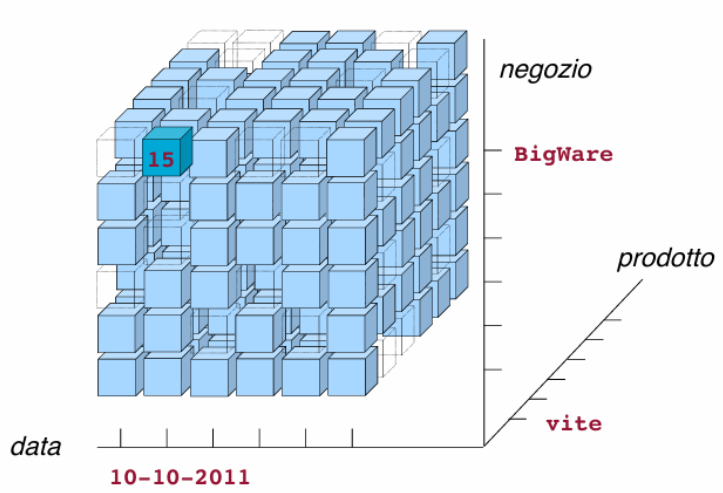

Figura 1.5: Esempio di un cubo per le vendite

di business, come vendite o ordini, che si vogliono analizzare. Per dare una rappresentazione intuitiva dei fatti si rappresentano gli stessi sotto forma di *cubi* (fig. 1.5). Ogni cella del cubo contiene misure che quantificano l'evento; gli assi del cubo rappresentano le dimensioni che forniscono i diversi punti di vista da cui analizzare i dati; da ogni dimensione può nascere una gerarchia di attributi usati per aggregare i dati memorizzati nel cubo.

Questa organizzazione dei dati consente di effettuare operazioni di analisi avanzate, come il *drill-down*, per esplorare i dati con un livello di dettaglio maggiore, e il *roll-up*, per sintetizzarli a un livello più aggregato.

Modellazione Concettuale Per rappresentare graficamente il modello multidimensionale si utilizza il *Dimensional Fact Model* (DFM). Essendo un modello concettuale offre diversi vantaggi. In primo luogo, consente di verificare già in fase di analisi se le interrogazioni espresse dagli utenti saranno effettivamente realizzabili all'interno del DW, fornendo un metodo per validare le query richieste. Inoltre, è progettato per essere comprensibile anche a utenti non tecnici, facilitando la comunicazione tra progettisti e stakeholder aziendali. Oltre a questo, il DFM costituisce un punto di riferimento per la successiva progettazione logica del DW e assicura una documentazione chia-

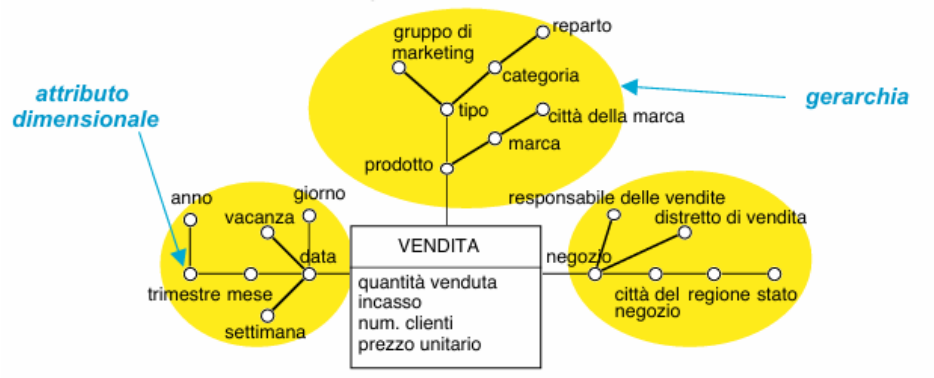

Figura 1.6: Esempio di uno schema di fatto per le vendite

ra e strutturata del sistema. Il risultato della modellazione con il DFM è lo schema di fatto, come mostrato in fig. 1.6.

Modellazione Logica Il modello multidimensionale è una rappresentazione astratta dei dati all'interno del DW. La modellazione logica può essere realizzata con una delle seguenti modalità:

- **ROLAP (Relational OLAP)** – Questo approccio si basa sull'uso di database relazionali per memorizzare e gestire i dati multidimensionali. I dati vengono archiviati all'interno di tabelle relazionali secondo schemi specifici come lo schema a stella (*star schema*) o lo schema a fiocco di neve (*snowflake schema*). In questa configurazione, le informazioni vengono denormalizzate per ridurre il numero di join e migliorare le prestazioni delle interrogazioni.

  Lo schema a stella è il modello più semplice ed efficace per rappresentare il modello multidimensionale in un database relazionale. Si compone di due elementi fondamentali, la tabella dei fatti e le tabelle delle dimensioni. Tabella dei Fatti (*Fact Table*): contiene i dati misurabili del sistema, come quantità vendute, ricavi, costi o performance. La chiave primaria della tabella dei fatti è composta dalle chiavi esterne che fanno riferimento alle tabelle delle dimensioni. Tabelle delle Dimensioni (*Dimension Table*): descrivono il contesto in cui i fatti si verificano,

includendo attributi dettagliati che permettono di analizzare i dati a diversi livelli di aggregazione. Ogni tabella delle dimensioni è collegata alla tabella dei fatti attraverso una chiave primaria surrogata.

Lo schema a fiocco di neve è una variante dello schema a stella, in cui le tabelle delle dimensioni sono in parte normalizzate per ridurre la ridondanza dei dati. In questo schema, le dimensioni vengono suddivise in più tabelle collegate tra loro, creando una struttura gerarchica che somiglia a un fiocco di neve. Questo schema è più efficiente in termini di spazio di archiviazione, poiché riduce la ridondanza normalizzando le dimensioni. L'aumento del numero di tabelle e delle relazioni tra di esse comporta un incremento del numero di join necessari per eseguire le query, rallentando le prestazioni delle analisi OLAP.

- **MOLAP (Multidimensional OLAP)** – Questo approccio utilizza strutture dati ottimizzate per l'analisi multidimensionale. È presente il problema della sparsità, in quanto in media solo una piccola percentuale delle celle del cubo contiene informazioni. L'adozione di questo modello logico è frenata dal fatto che non esiste ancora uno standard per le strutture dati: ogni produttore utilizza strutture proprietarie, limitando l'interoperabilità e la migrazione tra sistemi diversi.

- **HOLAP (Hybrid OLAP)** – Approccio che combina i vantaggi di ROLAP e MOLAP, memorizzando i sottocubi più densi in forma multidimensionale per evitare il problema della sparsità e il resto dei dati in forma relazionale.

1.3 DBMS per il Data Warehouse

Un *Database Management System* (DBMS) è il componente centrale di un DW, poiché ne gestisce i dati, la loro archiviazione e la loro interrogazione. Storicamente, i DW sono stati costruiti su database relazionali (RDBMS), che hanno dominato il settore per anni grazie alla loro affidabilità e alla diffusione

dello standard SQL per le interrogazioni. Tuttavia, la crescente complessità e varietà dei dati e delle relative esigenze di analisi ha portato alla creazione di diversi modelli di DBMS, come i DBMS NoSQL (*Not only Structured Query Language*). Essi sono stati progettati per superare i limiti dei sistemi relazionali in termini di scalabilità e gestione di dati semi-strutturati o non strutturati.

1.3.1 Principi ACID e BASE

I principi ACID (*Atomicità, Coerenza, Isolamento, Durabilità*) garantiscono integrità e affidabilità nelle operazioni transazionali. L'ACID si compone di quattro proprietà fondamentali:

- **Atomicità**: un'operazione viene eseguita completamente o non viene eseguita affatto. L'intera operazione viene annullata e viene ripristinato lo stato precedente se si verifica un errore.

- **Coerenza**: i dati nel database devono sempre rispettare le regole di integrità, evitando la scrittura di informazioni errate.

- **Isolamento**: le transazioni concorrenti non devono interferire tra di loro, garantendo l'integrità dei dati durante l'esecuzione parallela.

- **Durabilità**: una volta confermata una transazione, i dati devono essere permanentemente salvati nel database, senza il rischio di perdita in caso di guasti.

Tuttavia, in ambienti distribuiti, il modello ACID può risultare poco applicabile. In questi contesti, i DBMS adottano il paradigma BASE e si basano sul teorema CAP. Il paradigma BASE (*Basically Available, Soft state, Eventually consistent*) privilegia la disponibilità e le prestazioni rispetto alla coerenza immediata, accettando una consistenza eventuale per migliorare la scalabilità e la tolleranza ai guasti [3].

1.3.2 DBMS Relazionali

I DBMS relazionali rispettano i principi ACID e rappresentano la soluzione tradizionale per l'implementazione di DW. Utilizzano SQL per le operazioni di interrogazione e supportano la modellazione ROLAP, con schema a stella o a fiocco di neve.

1.3.3 DBMS NoSQL

Negli ultimi anni, l'espansione esponenziale dei dati, spesso non strutturati o semi-strutturati, e la necessità di gestire carichi di lavoro distribuiti con esigenze di scalabilità, hanno favorito la diffusione dei database NoSQL. I classici RDBMS basati su schema a stella e a fiocco di neve non sono adeguati alla mole di dati generata, ad esempio, dai social media [4].

Il termine NoSQL indica una famiglia eterogenea di DBMS caratterizzati da modelli dati non relazionali, alta scalabilità orizzontale e flessibilità dello schema. A differenza dei DBMS relazionali, che si basano sui principi ACID, i DBMS NoSQL rispettano i principi BASE e si suddividono, in linea di massima, in quattro categorie principali: *Key-Value Store*, *Column store*, *Document store* e *Graph store* [5] [3].

Chiave-Valore Memorizzano dati in coppie chiave-valore. Sono estremamente semplici e garantiscono alta velocità di accesso.

Colonnari I database colonnari si differenziano dai modelli relazionali tradizionali per la modalità di archiviazione dei dati. Invece di organizzare i dati per righe, essi li memorizzano per colonne, rendendo molto più efficienti le operazioni analitiche, in cui tipicamente si accede solo a un sottoinsieme di colonne.

Documentali Memorizzano dati in documenti, solitamente in formato JSON o BSON. Offrono grande flessibilità nel gestire dati semi-strutturati.

Grafo Specializzati per la rappresentazione di relazioni complesse tra entità. Ideali per analisi, ad esempio, di dati relativi ai social network.

1.3.4 DBMS Multi-Modello

Nessun modello preso singolarmente è ottimale per tutte le esigenze OLAP. Con l'aumento esponenziale della quantità e della diversità dei dati (strutturati, semi-strutturati e non strutturati), i database tradizionali (relazionali, documentali, a grafo, ecc.) mostrano limitazioni nel trattare contemporaneamente più modelli di dati. I DBMS multi-modello (*MMDBMS*) nascono dall'esigenza di gestire in modo efficiente i dati in formati differenti.

In [6] è stato analizzato l'utilizzo di un MMDBMS con tre modelli principali: relazionale, documentale e a grafo, con l'obiettivo di individuare le migliori strategie per rappresentare e interrogare dati multidimensionali, valutando un compromesso tra prestazioni, flessibilità e complessità di modellazione.

- Il **modello relazionale** è stato impiegato per strutturare i dati multidimensionali in uno schema a stella (*star schema*). Questo approccio offre vantaggi in termini di ottimizzazione delle query OLAP, grazie alla possibilità di sfruttare indici, join efficienti e vincoli di integrità referenziale. Tuttavia, può risultare meno flessibile nel trattare dati semi-strutturati o con gerarchie non rigidamente definite.

- Il **modello documentale** è stato utilizzato per gestire dati caratterizzati da una struttura più flessibile e da una maggiore variabilità. Due tipi di modellazione sono stati analizzati: uno *denormalizzato*, in cui i dati sono memorizzati in un singolo documento per ridurre il numero di join, e uno *shattered*, in cui i dati sono distribuiti su più documenti per evitare ridondanze. Il primo approccio semplifica l'accesso ai dati, ma porta a un consumo maggiore di spazio, mentre il secondo riduce la ridondanza ma aumenta la complessità delle interrogazioni.

- Il **modello a grafo** è stato adottato per rappresentare relazioni complesse e gerarchie flessibili. Questo modello ha la capacità di modellare facilmente relazioni molti-a-molti. Le prestazioni possono però essere peggiori rispetto agli altri modelli in presenza di grandi volumi di dati organizzati in tabelle.

1.4 Cloud Computing

Con il termine *cloud computing* si intende l'accesso e l'utilizzo di risorse e servizi informatici (come server, database, applicazioni e storage) tramite internet, eliminando la necessità di un'infrastruttura fisica locale. Questo approccio offre maggiore flessibilità e scalabilità rispetto alle tradizionali infrastrutture on-premise ed è basato su un sistema di pagamento pay-per-use, in cui i costi dipendono dall'effettivo utilizzo delle risorse [7].

1.4.1 Principali vantaggi del Cloud Computing

- **Riduzione dei costi iniziali**: il cloud computing elimina la necessità di investimenti iniziali significativi in hardware e infrastrutture, consentendo alle aziende di pagare solo per le risorse effettivamente utilizzate.

- **Disponibilità elevata**: in base al contratto che si ha con il cloud provider, le applicazioni basate sul cloud possono godere di un'elevata continuità operativa.

- **Scalabilità**: il cloud consente di espandere o ridurre le risorse in maniera semplice. È disponibile sia la scalabilità verticale (*scale UP*), ovvero l'aumento della capacità di calcolo di una singola macchina, aggiungendo ad esempio RAM o CPU, sia la scalabilità orizzontale (*scale OUT*), ovvero l'aumento del numero di istanze.

- **Elasticità**: è possibile configurare le applicazioni nel cloud in modo che scalino in automatico in base alle necessita, aumentando le risorse quando ci sono più richieste e diminuendo le risorse quando non sono più necessarie, per evitare spese inutili.

- **Distribuzione geografica**: i maggiori cloud provider mettono a disposizione datacenter collocati in tutto il mondo, consentendo di scegliere in quale area geografica distribuire l'applicazione per garantire prestazioni migliori ai clienti finali.

1.4.2 Principali modelli di Cloud Computing

Il cloud computing offre diversi modelli di servizio per soddisfare varie esigenze aziendali:

- **Iaas (Infrastructure as a Service)**: è il modello più simile alla gestione dei server fisici on-premise. Vengono messe a disposizione delle risorse hardware virtualizzate, affinchè l'utilizzatore possa creare e gestire la propria infrastruttura sul cloud senza preoccuparsi dell'allocazione delle risorse.

- **PaaS (Platform as a Service)**: in questo modello viene fornito un ambiente di sviluppo e gestione delle applicazioni che si intendono fornire. Il cloud provider garantisce il corretto funzionamento dell'infrastruttura sottostante.

- **SaaS (Software as a Service)**: questo modello racchiude applicativi e sistemi software, accessibili da qualsiasi tipi di dispositivo (computer, smartphone, tablet, ecc.), attraverso il semplice utilizzo di un'interfaccia web. L'utilizzatore ha come unica preoccupazione quella di dover utilizzare l'applicativo, senza dover pensare alla manutenzione del servizio o alla gestione dell'infrastruttura.

1.4.3 Il Cloud per Data Warehouse

Il cloud computing ha rivoluzionato anche il settore dei DW, offrendo soluzioni più scalabili, flessibili ed efficienti rispetto ai tradizionali sistemi on-premise. I DW on-premise continuano a offrire diversi vantaggi ancora oggi, come maggior controllo, sicurezza, sovranità dei dati e bassa latenza. Tuttavia, presentano delle limitazioni, come la poca elasticità in caso di esigenze di ridimensionamento e lo sforzo richiesto per la gestione degli stessi.

Per superare queste limitazioni molte aziende stanno adottando soluzioni basate sul cloud. Un DW cloud è un database basato su un DBMS situato nel cloud, che permette di archiviare, elaborare e analizzare grandi volumi di informazioni in maniera efficiente e ottimizzata per la BI. Questi sistemi sfruttano caratteristiche chiave come [8]:

- **Architettura MPP (Massively Parallel Processing)**: consente di distribuire il carico di lavoro su più nodi. A differenza dei sistemi SMP (Symmetric Multiprocessing), in cui un unico nodo gestisce l'intero processo di calcolo, l'MPP consente di suddividere le query tra più nodi che elaborano i dati contemporaneamente, riducendo i tempi di risposta.

- **Storage colonnare**: tipicamente i DBMS basati su architettura MPP utilizzano una modalità di archiviazione dei dati colonnare. Questo approccio migliora drasticamente le prestazioni delle query analitiche, perché consente di leggere solo le colonne necessarie.

1.5 Multi-Tenancy

Il termine *multi-tenancy* indica un'architettura software in cui un'unica istanza di un sistema o applicazione viene condivisa tra più utenti, detti *tenant*. Ogni tenant utilizza il sistema come se fosse dedicato esclusivamente a lui, ma in realtà le risorse sottostanti sono condivise tra più utenti. Questa soluzione è ampiamente utilizzata nei sistemi cloud, dove più organizzazioni o

utenti possono accedere alla stessa piattaforma, pur mantenendo un adeguato isolamento dei dati.

La **condivisione delle risorse** in un sistema multi-tenant può essere gestita in diversi modi. Una soluzione consiste nell'assegnare a ciascun tenant un database separato, garantendo così un elevato livello di isolamento. Un'altra opzione è l'uso di schemi distinti all'interno dello stesso database, mantenendo una separazione logica senza la necessità di gestire più istanze. Un altro approccio può prevedere invece la memorizzazione dei dati di tutti i tenant in un'unica tabella, con opportuni identificatori per distinguere l'appartenenza del dato al relativo tenant.

Tra i **vantaggi** della multi-tenancy troviamo sicuramente la riduzione dei costi operativi. Avendo un unico sistema condiviso, si evita la gestione di tante istanze separate, permettendo anche di effettuare manutenzione e aggiornamenti in maniera centralizzata. In questo modo è garantito che tutti i tenant utilizzino sempre la versione più recente del software.

La **scalabilità** è un altro aspetto chiave della multi-tenancy. Un sistema ben progettato deve essere in grado di gestire sia un aumento di richieste da parte degli utenti, sia un numero crescente di utenti, senza compromettere le prestazioni.

Vanno sicuramente tenuti in considerazione, e hanno più importanza rispetto ai sistemi pensati per il singolo tenant, gli aspetti di **sicurezza**. Deve essere garantito il fatto che i dati di ogni tenant siano accessibili solo da chi è autorizzato a farlo. È anche opportuno assegnare dei limiti sulle risorse utilizzabili da ciascun tenant; se un tenant utilizza un numero eccessivo di risorse, potrebbe degradare le prestazioni per gli altri utenti.

Capitolo 2

Benchmarking

La soluzione software attuale di BI, fornita dall'azienda ai propri clienti, consiste in un'istanza di queste tre componenti per ogni singolo tenant:

- **Backend**: sviluppato in .NET, responsabile della logica di business e della gestione dei dati.
- **Database**: basato su Microsoft SQL Server, contenente dati memorizzati in schemi a stella.
- **Frontend**: realizzato in Angular, integrato con i componenti DevExpress[1] per la visualizzazione e l'analisi dei dati in ottica BI.

Con questo approccio sono emersi diversi problemi legati ai tempi di esecuzione delle analisi, principalmente riconducibili alle prestazioni del DBMS. Questo avviene nonostante lo schema a stella sia stato progettato correttamente e gli indici siano stati definiti in modo adeguato. In particolare, i dati sono memorizzati in Microsoft SQL Server e vengono utilizzati indici B+Tree. Nella fact table sono stati creati indici sulle chiavi esterne relative alle dimension tables. Nelle dimension tables con alta cardinalità, sono stati creati indici su alcuni attributi chiave, come, ad esempio, la data nella dimensione temporale, la residenza dei clienti e la categoria degli articoli.

[1] `https://www.devexpress.com/`

Inoltre, il mantenimento di istanze separate per ogni tenant comporta un aumento dei costi, un'elevata complessità operativa e una scalabilità limitata. La scelta di una soluzione multi-tenant consente di ottenere diversi vantaggi, come già descritto nella sezione 1.5.

I principali requisiti da soddisfare sono i seguenti:

- **Prestazioni**: il tempo di risposta alle query deve essere inferiore a 20 secondi, considerato un limite massimo accettabile per un'interrogazione analitica.
- **Costi**: le spese operative e infrastrutturali del sistema devono essere basse e contenute, per garantire un'offerta competitiva sul mercato.
- **Scalabilità**: il sistema deve avere una scalabilità sia orizzontale sia verticale, per gestire l'aumento costante del numero di tenant.
- **Continuità operativa**: gli strumenti di BI (DevExpress) attualmente in uso devono essere riutilizzati, evitando la necessità di sviluppare un nuovo sistema da zero e garantendo un'esperienza coerente per gli utenti esistenti.
- **Flessibilità tecnologica**: si vuole evitare il rischio di provider lock-in, privilegiando soluzioni open-source o facilmente migrabili rispetto a tecnologie proprietarie.

In questo capitolo vengono prima valutati alcuni aspetti trattati in letteratura, e poi fatta una comparativa sulle prestazioni di alcuni DBMS ritenuti idonei, utilizzando un insieme di query OLAP rappresentative di scenari reali.

2.1 Letteratura

2.1.1 Sistemi di benchmarking

Nell'ambito dei sistemi di data warehousing la misurazione delle prestazioni e la comparabilità tra soluzioni diverse sono aspetti fondamentali. I

principali benchmark utilizzati per valutare l'efficacia di questi sistemi sono TCP-H (*Transaction Processing Council – H*), TCP-DS (*Transaction Processing Council – Decision Support*) [9] e SSB (*Star Schema Benchmark*) [10]. Essi si focalizzano su contesti di tipo analitico e utilizzano strutture dati costruite per riflettere pattern e complessità reali.

- **TPC-DS** rappresenta il benchmark più completo e realistico per i sistemi di BI. È costituito da un elevato numero di tabelle e il set di 99 query include operazioni complesse, rendendolo un valido strumento per la valutazione delle prestazioni in contesti avanzati di analisi dei dati.

- **TPC-H** è un benchmark consolidato, supportato da un'ampia community e da una documentazione dettagliata. Sono disponibili numerose implementazioni ottimizzate per DBMS sia di tipo relazionale che non relazionale. Il set di interrogazioni è composto da 22 query che coprono un certo range di operazioni analitiche, come join complessi, aggregazioni e filtraggi.

- **SSB** si basa su TPC-H ed è progettato per valutare le prestazioni dei sistemi di data warehousing utilizzando uno schema a stella. La struttura è relativamente semplice, più immediata da comprendere e implementare rispetto a TPC-DS. Si focalizza su join e aggregazioni frequenti in query analitiche. Permette di individuare ottimizzazioni specifiche per la struttura a stella, valutando tecniche di indicizzamento, partizionamento e compressione dei dati.

2.1.2 Modellazione dei dati in DBMS NoSQL

Nei DBMS relazionali, la modellazione logica dei dati segue un approccio ROLAP con schema a stella, o con schema a fiocco di neve, come descritto nella sezione 1.2.2. Questa modellazione non è ottimale in contesti NoSQL, ci sono per cui diversi studi concentrati sia sul passaggio diretto dal modello

concettuale al modello logico NoSQL, sia sul passaggio dal modello logico ROLAP a un modello logico adatto a sistemi NoSQL.

In [3] viene esaminato il ruolo dei DBMS NoSQL, in particolare MongoDB, nell'aggregazione di grandi volumi di dati per applicazioni di BI. Questo perchè in molti contesti applicativi c'è una produzione di dati molto elevata, rendendo necessaria l'aggregazione degli stessi. In questo articolo gli autori si concentrano sul comparare l'aggregazione dei dati tra SQL e NoSQL, utilizzando l'algoritmo MapReduce e dimostrando come esso possa essere utile in contesti con volume di dati elevato.

MapReduce è un modello di programmazione, nativamente implementato in MongoDB, per gestire grandi quantità di dati. A differenza dei database SQL, in cui l'aggregazione avviene con query che necessitano di molta RAM e CPU, MapReduce suddivide il calcolo tra più nodi, consentendo un'elaborazione scalabile. Il modello MapReduce si articola in due fasi principali: nella fase di *Map* i dati vengono processati e trasformati in una lista di coppie chiave-valore, nella fase di *Reduce* ogni coppia della lista viene processata per creare nuove coppie di chiave-valore in cui i valori sono aggregati.

Per quanto riguarda la modellazione logica, il punto di partenza è lo schema relazionale esistente. Viene prima considerata una mappatura diretta che emula la struttura relazionale, che viene scartata perchè richiederebbe di effettuare dei join tra le tabelle, riducendo così l'efficienza dell'architettura NoSQL. La scelta ottimale è risultata essere l'adozione di una struttura documentale in cui tutte le informazioni rilevanti sono contenute in un unico documento, evitando la necessità di operazioni di join. I risultati dello studio evidenziano il vantaggio prestazionale di NoSQL rispetto ai sistemi SQL tradizionali. In termini quantitativi, l'aggregazione dei dati senza l'uso di MapReduce ha impiegato tre ore in un ambiente SQL, a fronte di pochi minuti in MongoDB con l'uso di MapReduce.

In [4] vengono proposte delle regole per trasformare il modello multidimensionale concettuale in un modello per NoSQL documentali e colonnari. Gli autori definiscono due strategie di trasformazione: la trasformazione

semplice e la trasformazione *gerarchica*. La prima mira a rappresentare il modello multidimensionale preservando la distinzione tra fatti e dimensioni. Nel caso dei database colonnari, fatti e dimensioni sono memorizzati in famiglie di colonne differenti, collegando il fatto alle dimensioni attraverso la duplicazione degli identificatori delle dimensioni nella famiglia di colonne del fatto. Nel modello documentale, invece, ogni fatto viene rappresentato come un documento con attributi corrispondenti alle misure e agli identificatori delle dimensioni, mentre ogni dimensione viene salvata separatamente come un documento a sé stante. La trasformazione gerarchica organizza i dati in modo più efficiente secondo il modello NoSQL di destinazione. Nel modello colonnare, invece di salvare le dimensioni in tabelle separate, le gerarchie vengono rappresentate con super-colonne, che permettono di organizzare meglio le informazioni correlate. Nei database documentali, invece, le gerarchie vengono rappresentate tramite documenti annidati, permettendo di eliminare la necessità di join e migliorando le prestazioni delle interrogazioni che coinvolgono strutture gerarchiche. L'articolo conclude con un'analisi delle performance, in cui i due modelli NoSQL vengono implementati e confrontati in base ai tempi di scrittura e lettura, utilizzando il benchmark TPC-DS. Dall'analisi emerge che Cassandra, utilizzato come modello colonnare, offre performance migliori nelle operazioni di scrittura, mentre MongoDB, utilizzato come modello documentale, è più performante nelle query di lettura complesse. Dal confronto tra le trasformazioni, emerge che la trasformazione gerarchica migliora le prestazioni delle query rispetto alla trasformazione semplice.

In [11] viene proposto un approccio per la modellazione multidimensionale dei dati in sistemi NoSQL, confrontando modelli documentali e colonnari. Nel modello documentale con MongoDB, un fatto è rappresentato come un documento principale, con misure e dimensioni memorizzate come documenti innestati. Questa soluzione elimina la necessità di join, migliorando le prestazioni delle query di lettura. L'aspetto negativo è la ridondanza dei dati, mitigabile con un'architettura distribuita. Nel modello colonnare con

HBase, i dati vengono rappresentati in tabelle con colonne raggruppate in famiglie. Questo approccio ottimizza le operazioni di scrittura e riduce lo spazio occupato grazie alla compressione dei dati. Tuttavia, le interrogazioni che coinvolgono più attributi possono risultare meno efficienti rispetto al modello documentale, in quanto i dati possono essere distribuiti tra più colonne.

In [12] viene studiata l'implementazione di DW multidimensionali utilizzando sistemi NoSQL colonnari. L'obiettivo è fare un mapping diretto dal modello multidimensionale concettuale al modello logico su NoSQL, evitando il passaggio intermedio attraverso un DBMS relazionale. Vengono presentati tre approcci per organizzare i dati all'interno di database colonnari, ognuno con i suoi compromessi in termini di ridondanza, prestazioni delle query ed efficienza di archiviazione:

- MLC0: per ogni fatto, tutti i relativi attributi e dimensioni sono memorizzati in una singola tabella con una singola famiglia di colonne.
- MLC1: per ogni fatto, vengono separate le diverse dimensioni in famiglie di colonne distinte.
- MLC2: i fatti e le dimensioni sono memorizzati in tabelle separate, dove ogni tabella ha una sola famiglia di colonne.

L'analisi sperimentale, condotta su HBase utilizzando il benchmark SSB, evidenzia che MLC0 e MLC1 richiedono più tempo per il caricamento dei dati e occupano più spazio in memoria, ma offrono prestazioni simili nelle query OLAP che risultano più veloci rispetto a MLC2.

In [13] vengono presi in considerazione HBase e Cassandra, utilizzando MySQL come DBMS relazionale di riferimento. I risultati suggeriscono che HBase e Cassandra sono più adatti per applicazioni che richiedono elevata scrittura e distribuzione dei dati, mentre MySQL rimane una scelta valida per scenari con esigenze transazionali e dataset di dimensioni moderate.

In [5] gli autori evidenziano che, nei sistemi NoSQL colonnari, le prestazioni delle query possono essere ottimizzate attraverso un'adeguata or-

ganizzazione delle famiglie di colonne. Viene quindi proposto un metodo che utilizza tecniche di clustering, in particolare l'algoritmo k-means, per raggruppare gli attributi più frequentemente utilizzati insieme nelle query. L'efficacia del metodo è stata valutata utilizzando il benchmark TPC-DS, i risultati sperimentali mostrano che una corretta aggregazione degli attributi nelle famiglie di colonne può ridurre significativamente i tempi di esecuzione delle query OLAP.

In [14] vengono confrontati tre approcci di progettazione logica per DMBS NoSQL colonnari:

- NLA (*Normalized Logical Approach*): il modello mantiene una struttura normalizzata, separando fatti e dimensioni in tabelle distinte e utilizzando attributi semplici. Questo approccio riduce la ridondanza, ma comporta costi elevati nelle operazioni di join.

- DLA (*Denormalized Logical Approach*): i fatti e le dimensioni vengono memorizzati in un'unica tabella, con attributi semplici. Questo elimina la necessità di join, migliorando le prestazioni nelle query OLAP.

- DLA-CF (*Denormalized Logical Approach with Column Families*): come il DLA, ma con l'uso di famiglie di colonne per raggruppare gli attributi di una stessa dimensione.

L'analisi sperimentale, condotta su HBase con il benchmark SSB, mostra che i modelli denormalizzati DLA e DLA-CF riducono significativamente i tempi di esecuzione delle query rispetto al modello normalizzato NLA.

2.2 Tecnologie testate

Visti i requisiti richiesti, e alla luce dei lavori presenti in letteratura, sono stati presi in considerazione i seguenti DBMS: Microsoft SQL Server, Amazon Redshift, Citus PostgreSQL e Apache Doris.

2.2.1 Microsoft SQL Server

Microsoft SQL Server[2] è un DBMS relazionale sviluppato da Microsoft. Si tratta di una delle piattaforme per basi di dati più diffuse al mondo, utilizzata per gestire database delle dimensioni e strutture più disparate. Supporta il modello ACID e utilizza T-SQL, un linguaggio che estende le funzionalità di SQL.

Dal punto di vista della scalabilità, SQL Server non offre un'architettura nativa di tipo MPP. Per questo motivo, mentre la scalabilità verticale è supportata, la scalabilità orizzontale è più complessa da ottenere rispetto ad altre soluzioni distribuite. Un modo per ottenere la scalabilità orizzontale è quello di creare più istanze indipendenti, ognuna con il proprio set di dati.

Gli indici sono fondamentali per migliorare le prestazioni delle query e ridurre i tempi di accesso ai dati. In particolare i *Column Store Index* rappresentano una soluzione ottimizzata per i carichi di lavoro analitici. A differenza degli indici tradizionali basati su B+Tree, i Column Store Index organizzano i dati in colonne anziché in righe, permettendo un'elevata compressione dei dati e un accesso più efficiente nelle query OLAP [15].

2.2.2 Amazon Redshift

Amazon Redshift[3] è un DW cloud basato su PostgreSQL e completamente gestito da AWS. È stato progettato per gestire query OLAP su grandi volumi di dati in modo efficiente e scalabile, integrandosi con vari strumenti di BI.

Dal punto di vista architetturale, Amazon Redshift utilizza un'architettura a cluster composta da un *Leader Node* e diversi *Compute Nodes*. Il Leader Node ha il compito di gestire la distribuzione delle query ai nodi di calcolo e aggregare i risultati, mentre i Compute Nodes eseguono le query e memorizzano i dati. Ogni Compute Node è ulteriormente suddiviso in *Node Slices*, che permettono di parallelizzare l'elaborazione e ottimizzare le prestazioni complessive del sistema. L'approccio MPP consente di distribuire i carichi

[2] `https://www.microsoft.com/it-it/sql-server`

[3] `https://aws.amazon.com/it/redshift/`

di lavoro su più nodi, migliorando significativamente le prestazioni rispetto ai database tradizionali.

Una delle caratteristiche principali di Amazon Redshift è lo storage colonnare. Grazie a questo si ha una maggiore efficienza nelle query analitiche e un minor utilizzo dello spazio di archiviazione dovuto alla compressione dei dati.

L'integrazione con altri servizi AWS è un punto di forza di Amazon Redshift, consentendo di costruire architetture di analisi dati scalabili e flessibili. Alcuni servizi particolarmente rilevanti sono:

- **Redshift Spectrum**: permette di eseguire query direttamente su dati archiviati in Amazon S3 senza necessità di importarli nel cluster.
- **AWS Glue**: facilita l'automazione dei processi ETL.
- **Amazon QuickSight**: consente di analizzare e visualizzare i dati attraverso dashboard interattive e report avanzati.
- **AWS Lambda**: permette di automatizzare operazioni e rispondere a eventi in modo flessibile.

2.2.3 Citus PostgreSQL

Citus[4] è un'estensione open-source per PostgreSQL[5] che lo trasforma in un sistema distribuito scalabile orizzontalmente. Adotta un'architettura distribuita in cui i dati vengono suddivisi tra più nodi. Il *Coordinator Node* rappresenta il punto di ingresso per le connessioni al database e ha il compito di analizzare e distribuire le query tra i nodi di lavoro. I *Worker Nodes* eseguono effettivamente le operazioni richieste e memorizzano i dati, ciascuno gestendo un sottoinsieme del database complessivo. Per garantire una distribuzione efficiente, le tabelle vengono suddivise in *shard*, ossia partizioni più piccole, distribuite tra i worker in base a una chiave di distribuzione.

[4] `https://www.citusdata.com/`

[5] `https://www.postgresql.org/`

Tra gli aspetti rilevanti di Citus ci sono: il supporto allo storage colonnare, per ottimizzare l'esecuzione di query analitiche e ridurre il consumo di spazio; la replica dei dati su più nodi, per garantire tolleranza ai guasti e resilienza del sistema; l'esecuzione di query in maniera distribuita e parallelizzata, per ridurre i tempi di risposta e migliorare le prestazioni.

Citus può essere utilizzato sia in ambienti on-premise sia in soluzioni cloud, con implementazioni gestite che ne semplificano la configurazione e la manutenzione.

2.2.4 Apache Doris

Apache Doris[6] è un sistema open-source progettato per fornire elevate prestazioni e scalabilità in ambienti OLAP e di data warehousing. L'architettura di Doris si basa sul modello MPP, con un'alta efficienza nella gestione parallela delle query. Il sistema è composto principalmente da due tipologie di nodi, Frontend (FE) e Backend (BE). I nodi FE gestiscono le query degli utenti, eseguono operazioni di parsing e pianificazione, e coordinano la distribuzione delle richieste verso i nodi BE. I nodi BE sono responsabili dello storage dei dati e dell'effettiva esecuzione delle query. I dati sono organizzati in modo distribuito su più nodi BE per garantire scalabilità orizzontale, tolleranza ai guasti e affidabilità dei dati. Utilizza uno storage di tipo colonnare e ha dimostrato ottimi risultati nei principali benchmark analitici SSB, TPC-H e TPC-DS[7]. Uno degli aspetti che lo ha reso popolare e utilizzato è il fatto di essere compatibile con SQL, in particolare con la sintassi MySQL. È possibile utilizzarlo con driver JDBC e ODBC e con strumenti di BI come Tableau e Power BI.

L'integrazione con altri sistemi è un altro suo grande punto di forza. Può elaborare flussi di dati in tempo reale grazie alla compatibilità con Apache Kafka e Apache Flink, permettendo di analizzare eventi in modo immedia-

[6] `https://doris.apache.org/`

[7] `https://doris.apache.org/docs/benchmark/tpcds`

to. Inoltre, supporta l'integrazione nativa con Amazon S3, permettendo di leggere dati senza doverli importare.

2.3 Configurazione dei test

2.3.1 Schema dei dati

I test sono stati eseguiti su un DW organizzato secondo uno schema a stella, la sua modellazione logica è mostrata in fig. 2.1. Per motivi di riservatezza aziendale, lo schema presentato include solo alcuni elementi significativi. La fact table rappresenta le movimentazioni di magazzino e le relative dimension tables contengono informazioni aggiuntive per l'analisi. Le dimensioni principali includono: data, cliente, prodotto, magazzino, azienda, fornitore e staff. L'adozione dello schema a stella è motivata dalla necessità di ottimizzare le query analitiche, riducendo il numero di join richiesti da schemi più normalizzati, come lo schema a fiocco di neve, a vantaggio di prestazioni migliori nelle operazioni OLAP.

I dati nella fact table sono memorizzati con la massima granularità possibile, questo significa che ogni riga rappresenta il livello massimo di dettaglio disponibile per le movimentazioni di magazzino. Ad esempio, se un'operazione di carico o scarico di magazzino coinvolge più articoli, questi vengono registrati come righe separate nella fact table. Questo approccio consente massima flessibilità nelle analisi, evitando di pre-aggregare i dati, così da poter supportare diversi tipi di interrogazioni personalizzate per ogni tenant.

2.3.2 Caratteristiche delle macchine

Citus PostgreSQL

- **Tipo di istanza AWS:** EC2 r5.xlarge[8]
- **CPU:** Intel(R) Xeon(R) Platinum 8175M @ 2.50GHz (4 core)

[8] `https://instances.vantage.sh/aws/ec2/r5.xlarge`

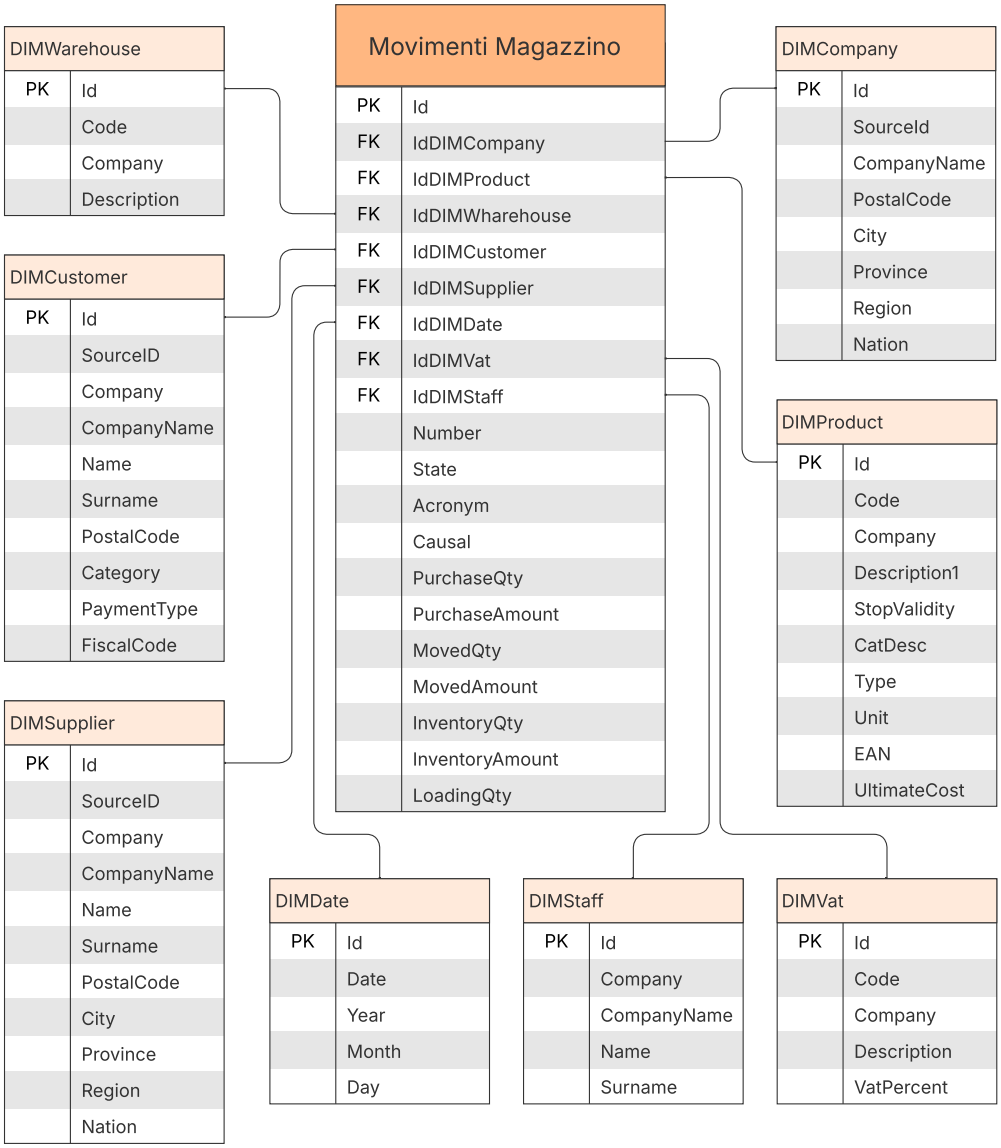

Figura 2.1: Schema a stella relativo ai movimenti di magazzino

- **RAM:** 32 GB
- **Disco:** SSD
- **Sistema operativo:** Red Hat Enterprise 8.9
- **DBMS:** postgresql16 con citus121_16.x86_64

Microsoft SQL Server

- **Tipo di istanza AWS:** EC2 r5.xlarge[9]
- **CPU:** Intel(R) Xeon(R) Platinum 8175M @ 2.50GHz (4 core)
- **RAM:** 32 GB
- **Disco:** SSD
- **Sistema operativo:** Windows Server 2022 Datacenter
- **DBMS:** SQL Server 2022 Column Store

Apache Doris

- **Tipo di istanza AWS:** EC2 r6i.xlarge[10]
- **CPU:** Intel(R) Xeon(R) 8375C (Ice Lake) @ 3.5 GHz (4 core)
- **RAM:** 32 GB
- **Disco:** SSD
- **Sistema operativo:** Ubuntu 24.04 LTS
- **DBMS:** Apache Doris 2.1.1

[9] `https://instances.vantage.sh/aws/ec2/r5.xlarge`
[10] `https://instances.vantage.sh/aws/ec2/r6i.xlarge`

Amazon Redshift

- **Tipo di istanza AWS:** Redshift ra3.xlplus[11]
- **CPU:** 4 core
- **RAM:** 32 GB
- **Nodi:** 2

2.3.3 Dati

Il dataset utilizzato per il benchmarking proviene da un cliente e rappresenta le movimentazioni di magazzino registrate nel sistema attualmente in uso. Il fatto analizzato contiene circa 26 milioni di record, rendendolo il più grande presente in produzione.

Per poter eseguire i test di benchmarking sui diversi sistemi, è stato necessario un processo di esportazione e importazione dei dati. L'esportazione della fact table e delle dimension tables è stata effettuata in formato CSV.

Il caricamento nei diversi DBMS testati è stato eseguito utilizzando le metodologie più efficienti per ciascun sistema:

- SQL Server: il caricamento è stato effettuato tramite SQL BULK INSERT[12], un metodo nativo che permette di inserire grandi volumi di dati.
- Amazon Redshift: il caricamento è stato effettuato tramite il comando SQL COPY, che permette di importare file direttamente da Amazon S3[13].
- Citus PostgreSQL: il caricamento è stato realizzato con il tool PSQL, per importare i dati dal file CSV locale.

[11] `https://instances.vantage.sh/aws/redshift/ra3.xlplus`

[12] `https://learn.microsoft.com/it-it/sql/t-sql/statements/bulk-insert-transact-sql?view=sql-server-ver16`

[13] `https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html`

Tabella	Numero di record
FTHWarehouseProg	26 461 686
DIMDate	4383
DIMCompany	1
DIMCustomer	204 890
DIMList	38
DIMProduct	11 129
DIMStaff	5
DIMSupplier	5
DIMVat	57
DIMWarehouse	358

Tabella 2.1: Numero di record nelle tabelle del fatto e delle dimensioni

- Apache Doris: è stato utilizzato Stream Load[14], un metodo che consente di caricare i dati tramite API HTTP.

La tabella 2.1 riporta il numero di record per ogni tabella, mentre la tabella 2.2 mostra i tempi di caricamento della fact table nei diversi DBMS. Apache Doris è risultato il più veloce, con un tempo di caricamento di 80 secondi, grazie all'utilizzo del metodo *Stream Load*, che permette un'elevata parallelizzazione nell'inserimento dei dati. Amazon Redshift ha completato il caricamento in 249 secondi, sfruttando il comando *COPY*, ottimizzato per l'importazione di grandi volumi di dati direttamente da file CSV archiviati in S3. Citus ha richiesto 311 secondi e il tempo più lungo è stato registrato da SQL Server con indici Column Store, che ha impiegato 549 secondi.

Oltre ai tempi di importazione, è stato analizzato anche lo spazio occupato dai dati nei vari sistemi, riportato in tabella 2.3. Quest'ultima evidenzia differenze significative nella gestione dello storage tra i diversi DBMS. SQL Server con indici B+Tree occupa 23,99 GB, mentre la versione Column Store

[14] `https://doris.apache.org/docs/data-operate/import/import-way/stream-load-manual`

DBMS	Tempo di caricamento	Metodo
SQL Server Column Store	549s	SQL BULK INSERT
Amazon Redshift	249s	SQL COPY
Citus PostgreSQL	311s	Tool PSQL
Apache Doris	80s	Stream Load (HTTP)

Tabella 2.2: Tempi di caricamento della fact table sui DBMS

DBMS	Spazio occupato
SQL Server b+tree	23,99 GB (Dati 9,17 GB, Indici 14,82 GB)
SQL Server Column Store	1,40 GB
Amazon Redshift	1,27 GB
Citus PostgreSQL	2,31 GB
Apache Doris	1,05 GB

Tabella 2.3: Spazio occupato dalla fact table nei diversi DBMS

dello stesso DBMS riduce lo spazio richiesto a 1,40 GB. Amazon Redshift e Apache Doris risultano i più efficienti in termini di compressione, con un utilizzo rispettivamente di 1,27 GB e 1,05 GB. Citus è il peggiore tra i DBMS colonnari, con 2,31 GB di spazio utilizzato.

2.3.4 Query OLAP

Per valutare le prestazioni dei diversi DBMS, è stato utilizzato un set di 17 query OLAP. Alcune query sono state estratte direttamente dal sistema di reportistica attuale, mentre altre sono state progettate per simulare scenari reali, variando non solo il group-by set, ma anche la selettività del predicato di selezione.

Le query prese in esame sono descritte nella tabella 2.4, che riporta le dimensioni coinvolte, gli attributi utilizzati nel group-by, i filtri applicati e la cardinalità del risultato. Le query contrassegnate con un asterisco (*) sono state estratte dal sistema attuale e vengono regolarmente eseguite dagli utenti finali.

2.4 Prestazioni singolo tenant

Dopo aver definito il set di query OLAP da utilizzare, è stata eseguita una prima fase di test per valutare le prestazioni in un ambiente single-tenant, ovvero con l'esecuzione delle interrogazioni in assenza di concorrenza tra utenti.

Per garantire una valutazione oggettiva delle prestazioni, ogni query è stata eseguita più volte, misurando sia il tempo di risposta alla prima esecuzione, quando i dati non sono ancora in cache, sia il tempo di esecuzione delle query successive, dove i DBMS possono sfruttare meccanismi di caching e ottimizzazione interna. I risultati di queste misurazioni sono riportati nelle tabelle 2.5 e 2.6.

SQL Server con indici **B+Tree** è il DBMS che ha mostrato i tempi di risposta più elevati, con alcune query che impiegano oltre 100 secondi, sia alla prima esecuzione che alle successive. È stato subito escluso per le scarse prestazioni.

SQL Server con indici **Column Store** riduce drasticamente i tempi, dimostrando come questi indici siano particolarmente efficaci in contesti OLAP. Sia nella prima esecuzione che nelle successive, i tempi di risposta sono molto buoni, tutti al di sotto dei 20 secondi richiesti. È già integrato con il sistema esistente, rendendolo una scelta familiare per l'azienda. Tuttavia, presenta alcune limitazioni: il costo della licenza, il vendor lock-in con Microsoft e la mancanza di supporto nativo alla distribuzione, che richiederebbe l'aggiunta di istanze separate per scalare.

Query	Dimensioni	Group-by	Selezione	Cardinalità
1	Warehouse, Product, Date	W.code, P.catdesc	D.date, W.code	48
2	Product, Company, Date	P.description1, C.CompanyName	D.date	5524
3	Date	D.Year, D.Quarter	-	45
4	Product, Customer	P.catdesc, C.Category	-	15 440
5	Product, Customer	P.Type, C.PaymentType	-	20
6	Customer	C.PostalCode	-	4196
7	Supplier, Product	S.CompanyName, P.Description1	-	13 155
8	Date, Customer	D.Year, C.CompanyName	-	327 078
9*	Warehouse, Product, Date	W.code, P.catdesc	P.Type, D.Year	338
10*	Warehouse, Product, Date	W.description, P.code, P.description1	D.Year	125 884
11*	Warehouse, Product, Date	W.description, P.code, P.description1	D.Year	1
12*	Company, Date	D.Year	F.acronym, F.state, D.Year	108
13*	Company, Date	-	F.acronym, F.state, D.Year	1
14*	Warehouse, Product, Date	W.description, P.code, P.description1	D.Year	1
15	Product, Vat	P.description1, V.VatPercent	P.AlwayAvailable	11 622
16	Product, Date, Staff, Customer	P.description1, D.Month, S.Id	F.Currency, D.Year	12
17	Product, Customer	C.CompanyName	F.Currency, P.BasePrice, C.Active	712

Tabella 2.4: Descrizione delle query OLAP prese in esame

Amazon Redshift si distingue per i tempi di esecuzione particolarmente bassi dopo la prima esecuzione. Questo perché, durante la prima esecuzione, genera e compila il codice per il piano di esecuzione della query, processo che introduce un overhead iniziale. Tuttavia, il codice compilato viene memorizzato in cache, permettendo alle esecuzioni successive della stessa query di saltare la fase di compilazione e di essere eseguite più rapidamente [16]. I tempi risultano accettabili per 16 query su 17 alla prima esecuzione, anche se più elevati rispetto a SQL Server Column Store e Apache Doris. Nelle esecuzioni successive si è rivelato il più performante. L'adozione di Redshift implicherebbe un vendor lock-in, un aspetto che l'azienda vorrebbe evitare.

Citus PostgreSQL, pur essendo una soluzione open-source con supporto alla distribuzione e scalabilità, ha evidenziato prestazioni deludenti, sia per la prima esecuzione che per le esecuzioni successive. Una percentuale importante di query è stata eseguita in un tempo superiore ai 20 secondi richiesti, suggerendo che questa soluzione non sia ottimale per carichi di lavoro OLAP di grandi dimensioni.

Apache Doris si è rivelato il DBMS con le prestazioni complessive migliori: è il più performante alla prima esecuzione delle query ed è molto performante anche nelle esecuzioni successive, quasi al pari di Amazon Redshift. Visto il caso d'uso dell'applicativo, dove ogni tenant può richiedere delle interrogazioni personalizzate, il tempo richiesto per la prima esecuzione è un fattore molto importante da tenere in considerazione. Inoltre non ha nessun costo per la licenza, è una soluzione open-source e ha un'architettura distribuita e scalabile. La tabella 2.7 riassume le principali caratteristiche emerse nei test.

La fase successiva del benchmark analizzerà le prestazioni in condizioni di concorrenza multi-tenant, per valutare come questi sistemi si comportano quando più utenti eseguono query contemporaneamente.

Query	SQL Server (B+Tree)	SQL Server (Column Store)	Redshift	Citus	Doris
1	1.704	0.087	3.830	4.693	0.168
2	82.854	7.000	7.100	24.133	0.939
3	11.156	1.114	7.290	9.300	0.396
4	34.964	1.722	9.630	15.686	1.198
5	27.208	1.382	13.170	14.630	0.894
6	37.675	17.161	17.700	120.667	4.465
7	26.212	2.129	7.020	15.689	1.264
8	24.952	5.037	11.832	16.952	1.129
9	126.428	2.649	5.230	41.261	2.804
10	33.032	7.061	15.570	47.419	1.576
11	62.611	2.525	8.840	22.321	0.341
12	4.628	0.186	9.810	4.092	0.123
13	3.699	0.199	6.542	4.130	0.046
14	37.469	11.396	8.039	24.326	2.187
15	22.007	1.625	11.560	16.159	1.097
16	28.110	7.375	27.197	37.573	1.196
17	3.076	0.195	13.110	5.590	0.183

Tabella 2.5: Tempi di esecuzione delle query (in secondi) per la prima esecuzione

Query	SQL Server (B+Tree)	SQL Server (Column Store)	Redshift	Citus	Doris
1	1.676	0.047	0.096	4.750	0.049
2	18.306	4.143	0.063	25.578	0.730
3	11.327	1.004	0.045	9.265	0.339
4	33.670	1.413	0.091	15.467	1.171
5	25.705	1.290	0.045	14.190	0.836
6	36.635	15.746	0.059	120.667	3.458
7	25.321	1.852	0.080	15.689	1.190
8	22.656	4.096	2.128	16.582	1.231
9	119.288	2.509	0.162	41.261	2.121
10	30.431	7.540	2.175	47.419	1.435
11	34.446	2.346	0.048	22.321	0.304
12	3.619	0.182	0.060	4.011	0.036
13	3.578	0.177	0.045	4.092	0.035
14	37.114	11.827	0.065	24.548	2.141
15	21.869	1.513	0.053	16.169	0.962
16	27.495	7.273	0.017	37.171	1.118
17	2.999	0.182	0.010	5.087	0.151

Tabella 2.6: Tempi di esecuzione delle query (in secondi) per le esecuzioni successive

Caratteristiche	SQL Server	Redshift	Citus	Doris
Familiarità				
Prestazioni				
Costo licenza				
Distribuzione				
Vendor Lock-in				
Scalabilità				

Tabella 2.7: Confronto caratteristiche SQL Server Column Store, Amazon Redshift, Citus PostgreSQL e Apache Doris

2.5 Test di carico multi-tenant

L'obiettivo principale di questa fase sperimentale è determinare il numero massimo di utenti che il sistema è in grado di supportare. Sulla base dei risultati ottenuti in assenza di concorrenza, i test di carico sono stati eseguiti esclusivamente su **Apache Doris** e **Microsoft SQL Server** in modalità Column Store, in quanto solo queste soluzioni hanno evidenziato prestazioni interessanti nelle interrogazioni OLAP.

2.5.1 Apache JMeter

Per testare le prestazioni dei vari DBMS in condizioni di utilizzo multi-utente, è stato utilizzato il software Apache JMeter[15]. Apache JMeter è uno strumento open-source che permette di effettuare test di carico e di stress su vari tipi di servizi e protocolli. In questo caso, grazie al supporto per il protocollo JDBC, è stato possibile simulare un carico di lavoro costituito da N utenti che eseguono in modo concorrente le query OLAP precedentemente definite.

L'obiettivo del test è quello di individuare quanti utenti possono essere supportati dal sistema mantenendo un tempo di risposta medio inferiore a 20 secondi. Per garantire la consistenza e la ripetibilità del test, è stato necessario controllare due aspetti critici della simulazione:

- Il tempo di attesa tra le richieste: per evitare che il comportamento del test sia influenzato da variazioni casuali nei tempi di attesa, il ritardo tra le richieste è stato generato utilizzando un generatore di numeri casuali con seed fisso.

- La selezione della query da eseguire: affinché ogni iterazione del test esegua le stesse query nelle stesse condizioni, anche la scelta della query da eseguire è stata determinata tramite un generatore casuale con seed fisso.

[15] `https://jmeter.apache.org/`

Queste operazioni sono state implementate in JSR223 Sampler all'interno di JMeter, utilizzando Groovy come linguaggio di scripting. Il codice utilizzato è riportato nei listati A.1 e A.2.

Grazie a questa impostazione, lanciando più volte il test su diversi DBMS, i risultati ottenuti saranno comparabili, poiché le variazioni nei tempi di risposta saranno dovute esclusivamente alle prestazioni del database e non a differenze accidentali nel carico generato.

La configurazione del test è la seguente:

- **Utenti**: da 5 a 100 utenti, a step di 5.
- **Durata del test**: 200min, 10min per ogni step.
- **Query**: esecuzione randomica delle 17 query OLAP predefinite.
- **Intervallo di tempo tra le richieste**: tempo random tra 60-120s.
- **Schemi presenti**: 100, per simulare la presenza dei dati di 100 tenant diversi.

2.5.2 Risultati per SQL Server

La tabella 2.8 riporta, per ogni livello di concorrenza testato, i seguenti dati:

- Query eseguite: il numero totale di query completate per quel livello di concorrenza.
- Query fallite: il numero di query che non sono state eseguite con successo a causa di timeout o errori del sistema.
- Tempo medio di esecuzione: il tempo medio impiegato per eseguire una query.
- Tempo massimo registrato: il tempo di esecuzione della query più lenta in ogni test.

- Tempo minimo registrato: il tempo di esecuzione della query più veloce in ogni test.
- Percentili 90°, 95° e 99°: i tempi di esecuzione della query che rientrano nei percentili indicati, ovvero il tempo entro il quale il 90%, 95% o 99% delle query è stato completato.
- Query eseguite in meno di 20 secondi: la percentuale di query che sono state completate entro 20 secondi, soglia considerata accettabile per l'applicativo.

L'analisi dei risultati mostra che SQL Server ha mantenuto buone prestazioni fino a circa 10 utenti concorrenti, con un tempo di esecuzione sotto la soglia dei 20 secondi per tutte le query. All'aumentare del numero di utenti, si osserva un degrado progressivo delle prestazioni. Il tempo medio di esecuzione è rimasto sotto i 20 secondi fino a 35 utenti. A questa soglia, solo il 79% delle query ha rispettato il vincolo dei 20 secondi, evidenziando un impatto crescente della concorrenza sul sistema. Per carichi superiori ai 50 utenti, la percentuale di query eseguite entro il tempo limite è scesa sotto il 50%, indicando una saturazione del sistema.

La fig. 2.2 evidenzia l'andamento del tempo medio e massimo di esecuzione rispetto al numero di utenti.

2.5.3 Risultati per Apache Doris

I risultati per Apache Doris sono riportati nella tabella 2.9, che presenta gli stessi indicatori già descritti per SQL Server nella sezione precedente. L'analisi mostra che Apache Doris ha mantenuto prestazioni nettamente superiori rispetto a SQL Server, dimostrando una migliore gestione della concorrenza.

Fino a 65 utenti concorrenti, il 100% delle query è stato eseguito in meno di 20 secondi. Con 70 utenti, Doris completa ancora il 96.72% delle query in meno di 20 secondi, mentre SQL Server con lo stesso carico era già sceso al 18.56%. Fino a 75 utenti il tempo medio rimane inferiore ai 20 secondi.

Utenti	Query eseguite	Query fallite	Tempo medio (ms)	Tempo max (ms)	Tempo min (ms)	90° perc. (ms)	95° perc. (ms)	99° perc. (ms)	Query < 20s (%)
5	29	0	2524	13436	54	6315	6848	11645	100.00
10	58	0	4251	18935	136	10360	14429	18761	100.00
15	95	0	5207	30875	90	13414	15686	27524	97.89
20	117	0	7897	36790	145	24189	29857	32067	86.32
25	151	0	10070	80278	84	25346	42419	70002	85.43
30	186	0	6143	49560	62	16407	25237	41382	93.01
35	198	0	13417	96863	60	38684	58034	83892	79.29
40	166	0	25727	110020	196	63920	91399	108423	56.02
45	240	0	22430	148710	119	53581	74468	108251	60.83
50	233	0	33576	158619	762	72420	102998	149233	42.91
55	241	0	44426	177908	1363	93380	125075	160018	31.95
60	228	0	65152	232490	2737	138225	163673	224369	21.49
65	217	0	82226	241696	4990	143092	172675	222252	22.58
70	237	0	81313	219328	2467	152680	172342	216204	18.56
75	265	0	86079	258483	3189	153149	186739	231900	20.37
80	268	0	84526	319523	6340	159779	193112	241040	21.64
85	219	0	124173	291121	8360	213653	229384	268878	14.61
90	219	0	157455	349201	13762	245890	260742	283205	10.04
95	193	0	192952	410337	13940	297741	315959	349493	3.10
100	201	0	159708	382288	14540	251624	283920	338697	8.45

Tabella 2.8: Performance di Microsoft SQL Server Column Store per test di carico multi-tenant

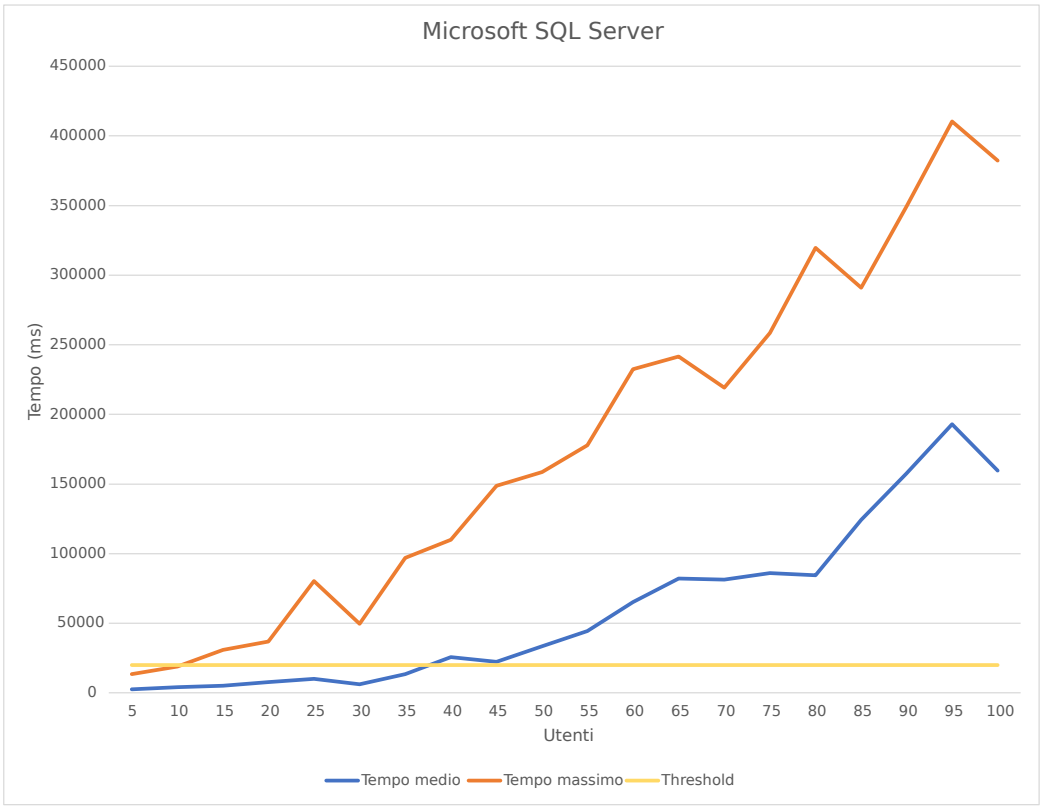

Figura 2.2: Tempo di esecuzione medio e massimo per Microsoft SQL Server Column Store

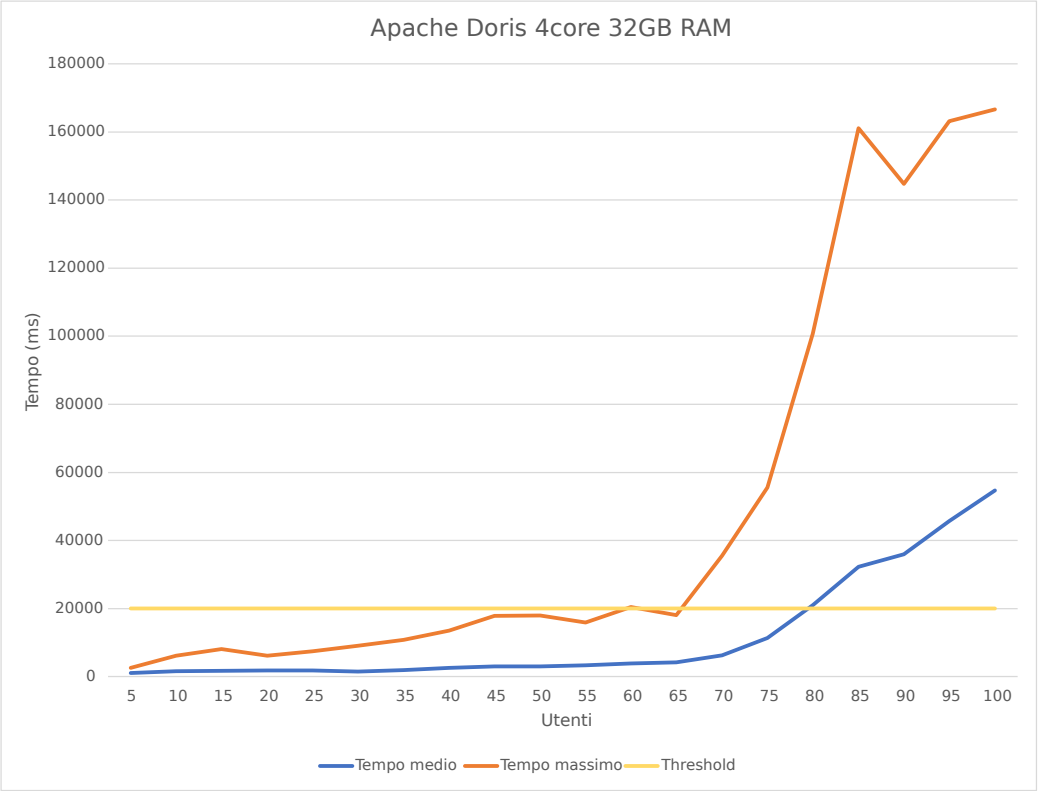

Figura 2.3: Tempo di esecuzione medio e massimo per Apache Doris con 100 schemi, 32GB di RAM e CPU 4 core

La fig. 2.3 evidenzia l'andamento del tempo medio e massimo di esecuzione rispetto al numero di utenti. Il tempo massimo di esecuzione è rimasto contenuto fino a circa 65 utenti, per poi aumentare rapidamente.

2.5.4 Scalabilità

Visti i risultati e le considerazioni precedenti, è stato scelto e tenuto in considerazione da ora in poi solo Apache Doris. È stata per prima cosa valutata la scalabilità verticale, passando a un'istanza EC2 r6a.2xlarge[16] in

[16] `https://instances.vantage.sh/aws/ec2/r6a.2xlarge`

Utenti	Query eseguite	Query fallite	Tempo medio (ms)	Tempo max (ms)	Tempo min (ms)	90° perc. (ms)	95° perc. (ms)	99° perc. (ms)	Query < 20s (%)
5	29	0	1025	2583	56	2143	2404	2555	100.00
10	60	0	1616	6112	73	3055	3995	5986	100.00
15	100	0	1734	8117	93	3078	4966	6012	100.00
20	126	0	1808	6196	91	3732	4494	5972	100.00
25	163	0	1794	7438	93	4199	4887	6562	100.00
30	195	0	1552	9035	91	3565	5009	8199	100.00
35	220	0	1912	10842	93	4429	5536	7325	100.00
40	263	0	2586	13567	94	6207	8150	12576	100.00
45	281	0	2990	17836	101	6882	9293	16222	100.00
50	316	0	3058	17958	110	6588	9095	15562	100.00
55	352	0	3397	15868	108	7510	9363	12618	100.00
60	385	0	3902	20419	101	9736	11997	16041	99.74
65	410	0	4260	18049	93	10566	13283	17214	100.00
70	427	0	6270	35561	108	13839	17285	26048	96.72
75	428	0	11378	55535	121	27006	32954	46474	81.54
80	430	3	21029	100742	190	44228	58564	76900	61.86
85	416	0	32254	161101	133	71367	98499	130791	41.10
90	435	10	35906	144697	135	80410	99542	119060	38.62
95	412	8	45717	163126	4410	101708	117401	143097	26.69
100	326	14	54737	166586	5897	108867	130564	164253	16.56

Tabella 2.9: Performance di Apache Doris per test di carico multi-tenant con 100 schemi, 32GB di RAM e CPU 4 core

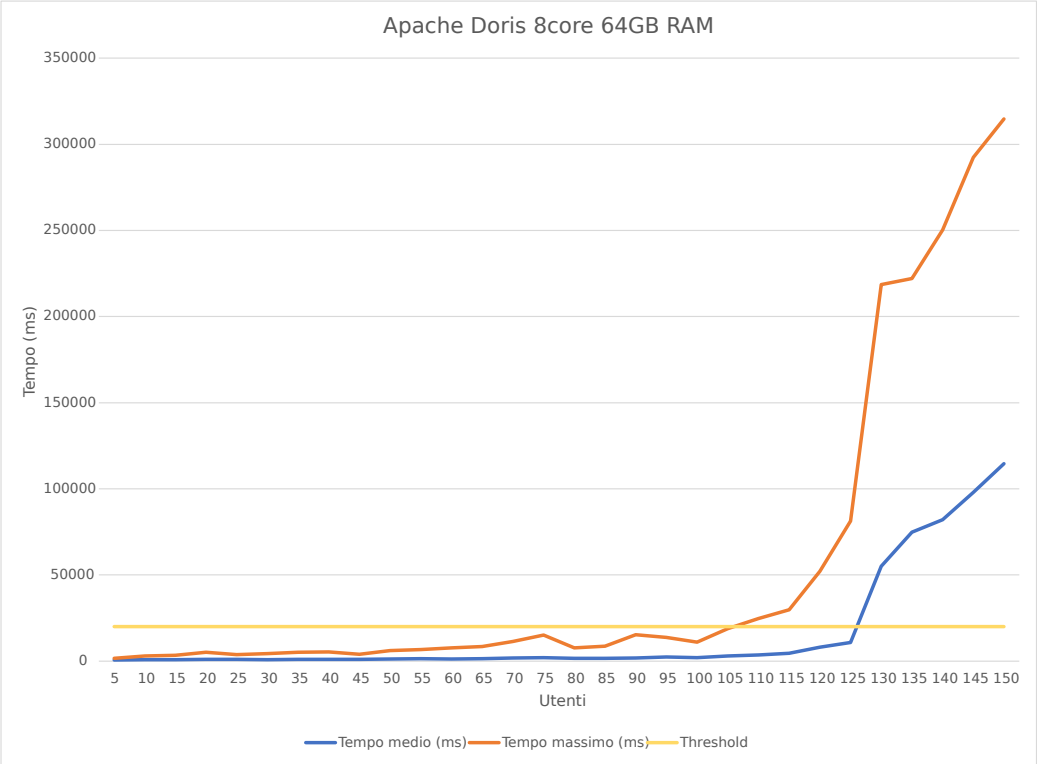

Figura 2.4: Tempo di esecuzione medio e massimo per Apache Doris con 100 schemi, 64GB di RAM e CPU 8 core

modo da raddoppiare la RAM e il numero di core della CPU, passando da 4 core e 32 GB di RAM a 8 core e 64 GB di RAM.

La tabella 2.10 e la fig. 2.4 mostrano i risultati ottenuti con questa configurazione. Il 100% delle query è stato eseguito in meno di 20 secondi fino a 105 utenti, mentre nella configurazione precedente il limite era di 65 utenti. Anche il tempo medio di esecuzione ha mostrato un incremento positivo, rimanendo inferiore ai 20 secondi fino a 125 utenti, rispetto ai 75 utenti della configurazione originale.

Successivamente, è stato eseguito un ulteriore test per analizzarne il comportamento in scenari con un numero significativamente maggiore di schemi e tabelle.

Utenti	Query eseguite	Query fallite	Tempo medio (ms)	Tempo max (ms)	Tempo min (ms)	90° perc. (ms)	95° perc. (ms)	99° perc. (ms)	Query < 20s (%)
5	29	0	626	1569	36	1296	1493	1565	100.00
10	61	0	867	2860	39	1449	2708	2856	100.00
15	99	0	882	3351	35	1681	2498	3048	100.00
20	126	0	1030	5060	58	2449	3027	3682	100.00
25	170	0	938	3806	61	2604	2827	3596	100.00
30	194	0	859	4236	56	2118	2707	3268	100.00
35	223	0	1049	5109	61	2615	3314	4210	100.00
40	260	0	1031	5213	58	2427	3009	4313	100.00
45	296	0	1053	3977	61	2324	2806	3732	100.00
50	326	0	1124	6112	60	2633	3328	4500	100.00
55	348	0	1339	6738	59	3235	4105	5275	100.00
60	397	0	1159	7577	60	2507	3331	4973	100.00
65	429	0	1458	8445	59	3503	4200	6594	100.00
70	452	0	1769	11407	62	3826	4921	9195	100.00
75	498	0	2035	15027	64	4877	6885	12744	100.00
80	518	0	1580	7559	60	3684	4585	6647	100.00
85	540	0	1547	8698	55	3414	4584	6886	100.00
90	589	0	1843	15232	59	4121	5539	11531	100.00
95	616	0	2461	13707	62	6117	7885	11235	100.00
100	663	0	1998	11017	67	4547	5351	7559	100.00
105	671	0	3009	18837	65	7558	11329	17242	100.00
110	701	0	3633	24727	65	8430	11778	18658	99.14
115	724	0	4425	29842	66	10698	13502	18799	99.44
120	720	1	7999	52068	73	17959	30985	43671	91.80
125	705	0	10715	81297	72	26213	37681	58633	84.25
130	515	0	54997	218538	178	142285	149930	197342	35.33
135	495	0	74871	222175	8999	147395	177013	210847	16.56
140	460	0	82146	250340	8238	175361	208713	243481	15.21
145	467	0	97913	292462	11917	192829	236513	276012	11.34
150	339	0	114500	314638	15645	205752	272369	307665	3.53

Tabella 2.10: Performance di Apache Doris per test di carico multi-tenant con 100 schemi, 64GB di RAM e CPU 8 core

Il team di sviluppo di Apache Doris ha evidenziato che il principale fattore critico per il sistema non è la quantità di dati all'interno delle tabelle, ma piuttosto la gestione di un elevato numero di tabelle e metadati. Per verificare questa affermazione, sono stati eseguiti test incrementando gradualmente il numero di schemi e tabelle nel sistema.

In un primo scenario, il numero di schemi è stato portato a 1000, simulando un aumento degli utenti di dieci volte rispetto alla configurazione precedente. I risultati di questa configurazione sono riportati nella tabella 2.11. In questa configurazione, Apache Doris ha mantenuto prestazioni eccellenti, con il 100% delle query eseguite in meno di 20 secondi fino a 105 utenti concorrenti, dimostrando una gestione efficiente del carico anche con un numero elevato di schemi.

Successivamente, il test è stato esteso aumentando ulteriormente il numero di schemi a 3000 e aggiungendo 19 nuove tabelle per schema, per un totale di 29 tabelle per schema, al fine di simulare l'aggiunta di nuovi cubi al sistema oltre a quello dei movimenti di magazzino. I risultati di questa configurazione sono riportati nella tabella 2.12. L'analisi mostra che, fino a 80 utenti concorrenti, Apache Doris è riuscito a mantenere prestazioni molto buone, con il 99% delle query eseguite in meno di 20 secondi.

Questi risultati confermano che Apache Doris è altamente scalabile, ma la gestione di un numero molto elevato di schemi e metadati rappresenta un fattore critico da tenere in considerazione in ambienti multi-tenant.

Utenti	Query eseguite	Query fallite	Tempo medio (ms)	Tempo max (ms)	Tempo min (ms)	90° perc. (ms)	95° perc. (ms)	99° perc. (ms)	Query < 20s (%)
5	29	0	641	2244	38	1423	1479	2033	100.00
10	61	0	861	2990	38	1603	2743	2919	100.00
15	99	0	893	3958	38	1689	2247	3632	100.00
20	126	0	1113	4989	64	2733	2966	4276	100.00
25	171	0	920	3584	58	2568	2720	3205	100.00
30	193	0	748	2939	59	1738	2454	2758	100.00
35	223	0	958	4296	61	2060	2669	3800	100.00
40	260	0	1008	4783	64	2124	2870	3822	100.00
45	296	0	1268	5091	61	2907	3440	4682	100.00
50	327	0	1112	5369	61	2605	3199	4339	100.00
55	347	0	1337	5692	61	3264	3843	4988	100.00
60	398	0	1159	6023	60	2692	3129	4233	100.00
65	426	0	1478	6967	62	3468	4708	6516	100.00
70	455	0	1531	7475	62	3460	4116	6078	100.00
75	500	0	1839	9115	61	4115	5099	6896	100.00
80	513	0	1543	10517	63	3473	4381	7664	100.00
85	545	0	1496	8417	61	3347	4075	5655	100.00
90	587	0	2088	11973	64	4661	5542	8798	100.00
95	619	0	2073	15594	74	4537	6165	12129	100.00
100	655	0	2965	27508	66	5870	10297	24933	98.01
105	672	0	2690	14685	66	6523	8276	10748	100.00
110	710	0	4049	20374	69	8867	11330	16258	99.85
115	637	0	15249	121029	82	44185	70070	93255	77.23
120	539	0	19062	123282	76	46626	67512	105467	69.38

Tabella 2.11: Performance di Apache Doris per test di carico multi-tenant con 1000 schemi, 64GB di RAM e CPU 8 core

Utenti	Query eseguite	Query fallite	Tempo medio (ms)	Tempo max (ms)	Tempo min (ms)	90° perc. (ms)	95° perc. (ms)	99° perc. (ms)	Query < 20s (%)
5	29	0	847	1965	46	1690	1800	1923	100.00
10	60	0	1221	4767	54	2469	3254	4753	100.00
15	99	0	1222	5359	40	2694	3241	5149	100.00
20	127	0	1291	4719	66	2689	3377	4380	100.00
25	165	0	1149	5725	62	2847	3228	4037	100.00
30	197	0	1030	4430	62	2237	3038	4253	100.00
35	221	0	1272	6713	65	3067	3701	5237	100.00
40	264	0	1225	5896	67	2696	3456	4746	100.00
45	289	0	1299	4892	69	2955	3733	4616	100.00
50	328	0	1258	5288	68	2910	3348	4543	100.00
55	351	0	1664	9353	63	3835	5204	6109	100.00
60	391	0	1915	11709	66	4337	5862	9030	100.00
65	421	0	2495	13779	66	5942	8698	12280	100.00
70	448	0	3383	27105	78	7633	11418	23067	98.21
75	484	0	3325	18320	67	7238	9670	14126	100.00
80	502	0	4176	28915	70	9012	11702	19792	99.00
85	491	0	11406	64115	83	26679	36688	46340	83.29
90	471	0	21345	113460	86	47049	61371	86900	60.93
95	469	0	32613	127665	17	79176	93688	122012	43.49
100	411	0	48712	193487	3749	100830	130907	162442	26.76
105	460	0	48579	169216	17	110500	123474	153625	28.91
110	403	0	74597	222393	8644	151941	170137	203073	14.39
115	415	0	74482	222696	3755	159172	178772	219181	18.79
120	303	3	91741	282768	11536	185327	241824	279539	10.56

Tabella 2.12: Performance di Apache Doris per test di carico multi-tenant con 3000 schemi, 64GB di RAM e CPU 8 core

Capitolo 3

Proof of Concept

Questo capitolo descrive l'implementazione pratica della migrazione e della messa in opera di un DW multi-tenant basato su Apache Doris, per verificarne la fattibilità e garantire l'integrazione con il sistema esistente.

3.1 Setup del sistema

L'obiettivo di questa PoC è sostituire il DBMS esistente con Apache Doris, verificando le prestazioni, la fattibilità e l'integrazione con il backend e il frontend già utilizzati.

Secondo la documentazione ufficiale di Apache Doris, l'architettura in un ambiente di produzione dovrebbe includere almeno un nodo frontend e tre nodi backend (fig. 3.1), con i seguenti requisiti minimi:

- CPU: 16 core
- RAM: 64 GB
- Storage: SSD da 100 GB
- Rete: 10 Gigabit

In un ambiente di produzione, questi nodi dovrebbero essere distribuiti su macchine diverse per garantire alta disponibilità e scalabilità orizzontale.

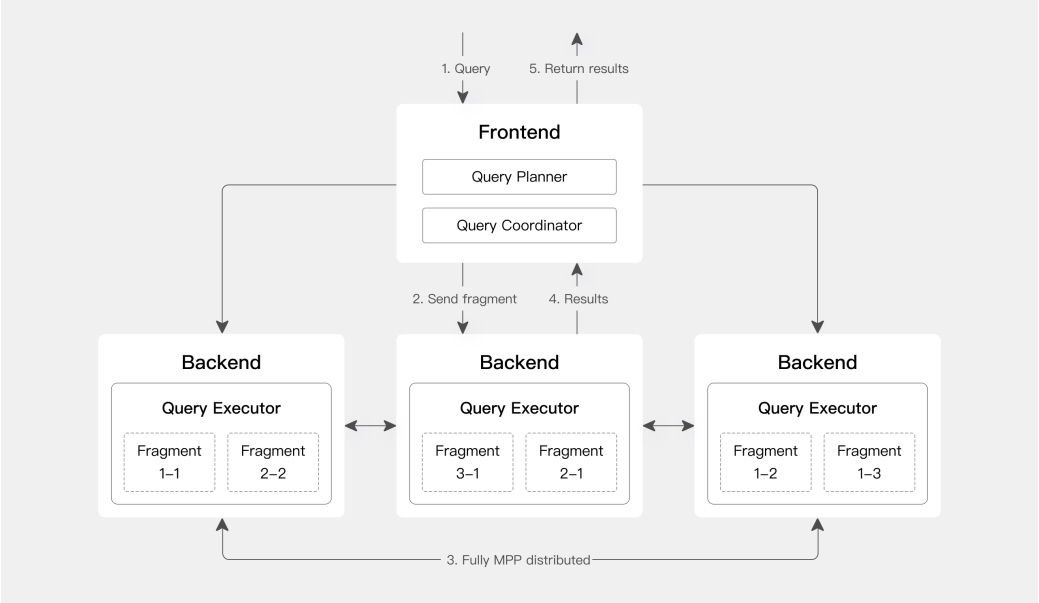

Figura 3.1: Architettura Apache Doris

Tuttavia, per questa fase iniziale di test e sviluppo, si è optato per un'installazione semplificata basata su Docker[1], al fine di facilitare il deploy e la gestione dell'infrastruttura.

Docker è una piattaforma open-source che consente di eseguire applicazioni all'interno di container, ambienti isolati e portatili che includono tutto il necessario per l'esecuzione dell'applicazione. I container vengono creati a partire da immagini, ovvero template preconfigurati che contengono l'applicazione e le sue dipendenze. Quando viene eseguito un container, Docker utilizza il kernel del sistema operativo host, garantendo così un overhead minimo rispetto alle tradizionali macchine virtuali.

Per orchestrare l'ambiente multi-container è stato utilizzato Docker Compose[2]. Questo strumento permette di definire l'intera infrastruttura tramite un file YAML, semplificando la configurazione dei container. Nel setup attuale sono stati avviati due container, uno per il nodo frontend e uno per il

[1] `https://www.docker.com/`

[2] `https://docs.docker.com/compose/`

nodo backend.

Dopo l'avvio dei nodi, è necessario eseguire una query per registrare il nodo backend all'interno del nodo frontend. Una volta completata questa operazione, Apache Doris espone interfacce compatibili con il protocollo MySQL, consentendo l'integrazione con strumenti e applicazioni che operano su database MySQL.

3.2 Data Warehouse

3.2.1 Caratteristiche di Apache Doris

Modello di tabella

A differenza di un RDBMS tradizionale, Apache Doris non supporta concetti come Primary Key e Foreign Key, poiché il suo obiettivo non è garantire l'integrità referenziale tra tabelle, ma fornire risposte rapide a query analitiche su grandi volumi di dati.

Le colonne in Apache Doris si dividono in due categorie, colonne *chiave* e colonne *valore*. Le colonne chiave sono utilizzate per identificare e organizzare i dati, mentre le colonne valore contengono i dati effettivi da analizzare.

Il modello di tabella scelto durante la creazione della stessa determina come Doris gestisce e organizza i dati, influenzando direttamente le prestazioni delle query. Esistono tre modelli:

- **Duplicate**: Questo modello consente di memorizzare righe duplicate in base alle colonne chiave specificate. È adatto per scenari in cui è essenziale preservare tutti i record di dati originali.

- **Unique**: In questo modello, ogni riga è unicamente identificata dalla combinazione di valori nelle colonne chiave. Questo garantisce che non esistano righe duplicate per un determinato insieme di valori chiave. Se viene aggiunta una nuova riga con la stessa chiave, la riga esistente verrà sovrascritta.

- **Aggregate**: Questo modello consente l'aggregazione dei dati basata sulle colonne chiave. È comunemente usato per scenari in cui sono richieste informazioni di riepilogo o aggregate, come totali o medie. Non è ottimale per query di COUNT, poiché richiede operazioni aggiuntive che possono impattare le prestazioni.

Se durante la creazione della tabella non viene specificato nessun modello di dati, Apache Doris imposta automaticamente il modello Duplicate come valore predefinito e seleziona automaticamente le colonne chiave e valore [17].

Viste materializzate

Caratteristiche Le viste materializzate sono uno strumento progettato per migliorare l'efficienza delle query attraverso la pre-aggregazione dei dati. A differenza delle viste standard, una vista materializzata salva i dati aggregati in memoria.

Doris gestisce automaticamente aggiornamenti e cancellazioni, sincronizzando le modifiche dalla tabella base alla vista materializzata. L'aggiornamento è incrementale, senza la necessità di dover ricalcolare l'intera vista. Ad esempio, se viene fatto un INSERT nella tabella base, la vista materializzata viene aggiornata immediatamente in automatico.

Quando si esegue una query che può sfruttare una vista materializzata, Apache Doris decide automaticamente se utilizzare la vista materializzata o accedere ai dati originali, a seconda di quale operazione è più efficiente. Non è necessario modificare le query per sfruttare le viste materializzate.

La creazione di una vista materializzata in Apache Doris è un'operazione asincrona. Questo significa che, una volta inoltrata la richiesta di creazione, il sistema elabora i dati esistenti in background.

Limitazioni Nonostante i vantaggi, l'uso delle viste materializzate presenta alcune limitazioni.

In caso di DELETE sulla tabella di base, per riflettersi nella vista è necessario che la condizione di cancellazione includa le colonne chiave utilizzate

nella vista stessa. In caso contrario, è necessario ricostruire completamente la vista.

Le viste materializzate basate sul modello Unique non possono eseguire operazioni di aggregazione. Ciò significa che non è possibile utilizzare queste viste per pre-calcolare somme, medie, conteggi o altre metriche aggregate. La vista può riorganizzare i dati, ma non può combinarli o sintetizzarli.

Un numero elevato di viste può rallentare il processo di importazione dei dati, poiché ogni aggiornamento della tabella base comporta un aggiornamento sincrono delle viste associate. Ad esempio, se ci sono 10 viste materializzate, eseguire un import sulla tabella base equivale a fare 10 import.

Partizionamento

Apache Doris utilizza un partizionamento a due livelli per organizzare i dati:

- **Bucketing** (obbligatorio): suddivisione delle partizioni in tablet (bucket), che rappresentano le unità di archiviazione fisica dei dati.
- **Partitioning** (opzionale): suddivisione dei dati in partizioni logiche.

Bucketing I dati sono sempre suddivisi in tablet, che rappresentano le unità di memorizzazione dei dati a livello fisico. Ogni tablet contiene un insieme di righe di dati, non vi è sovrapposizione tra i diversi tablet e ciascuno viene memorizzato indipendentemente. Le colonne scelte per il bucketing devono essere colonne chiave nel caso di tabelle basate sui modelli Aggregate o Unique. È possibile assegnare più colonne come chiavi di distribuzione. La scelta delle colonne di bucketing è un fattore importante da tenere in considerazione durante la creazione della tabella: l'uso di una sola colonna di bucketing migliora le prestazioni per query ad alta concorrenza, mentre un bucketing su più colonne aiuta a distribuire i dati in modo più uniforme, ottimizzando il throughput nelle query di aggregazione su grandi dataset. Doris permette anche di impostare un bucketing casuale, utile per l'analisi aggregata dell'intero dataset.

Partitioning Le tabelle possono essere partizionate o non partizionate. Questa decisione viene presa al momento della creazione della tabella e non può essere modificata successivamente. Se una tabella è partizionata, è possibile aggiungere o eliminare partizioni nel tempo, mentre per una tabella non partizionata non sarà possibile farlo.

Best practices L'efficacia del partizionamento dipende dalle dimensioni dei dati e dal carico di lavoro previsto. Per dataset di piccole e medie dimensioni, il bucketing può essere sufficiente senza la necessità di ulteriori partizionamenti. Per dataset di grandi dimensioni, Apache Doris raccomanda il partizionamento quando il volume dei dati per tabella supera i 500 GB, con una dimensione ottimale della singola partizione intorno ai 50 GB e tra 16 e 32 tablet per partizione.

Le partizioni possono essere gestite in modalità manuale, dinamica o automatica. In un contesto di DW multi-tenant, una strategia efficace potrebbe essere il partizionamento per data, in modo da ottimizzare le analisi su intervalli temporali ed eliminare facilmente i dati obsoleti operando su singole partizioni invece che sull'intera tabella.

3.2.2 Schema a stella su Apache Doris

In fig. 3.2 viene mostrato lo schema a stella adottato su Apache Doris. A livello di progettazione fisica, sono state apportate alcune modifiche per adattarsi al modello di tabelle e alle funzionalità di Doris.

In ciascuna dimension table è stato introdotto un campo ID surrogato, generato in modo autoincrementale. Tale campo non fa parte delle colonne chiave, ma è utilizzato come riferimento nella fact table per collegare le dimensioni. Nelle tabelle di dimensione, è presente un attributo *SourceId* (o *Code*) che identifica l'entità nelle sorgenti originali ed è utile in fase di ETL per l'allineamento dei dati.

Il **modello di tabella** adottato è il modello Unique. In questo modo si sfrutta la funzionalità di UPSERT (ovvero *Insert or Update*) di Apache Doris

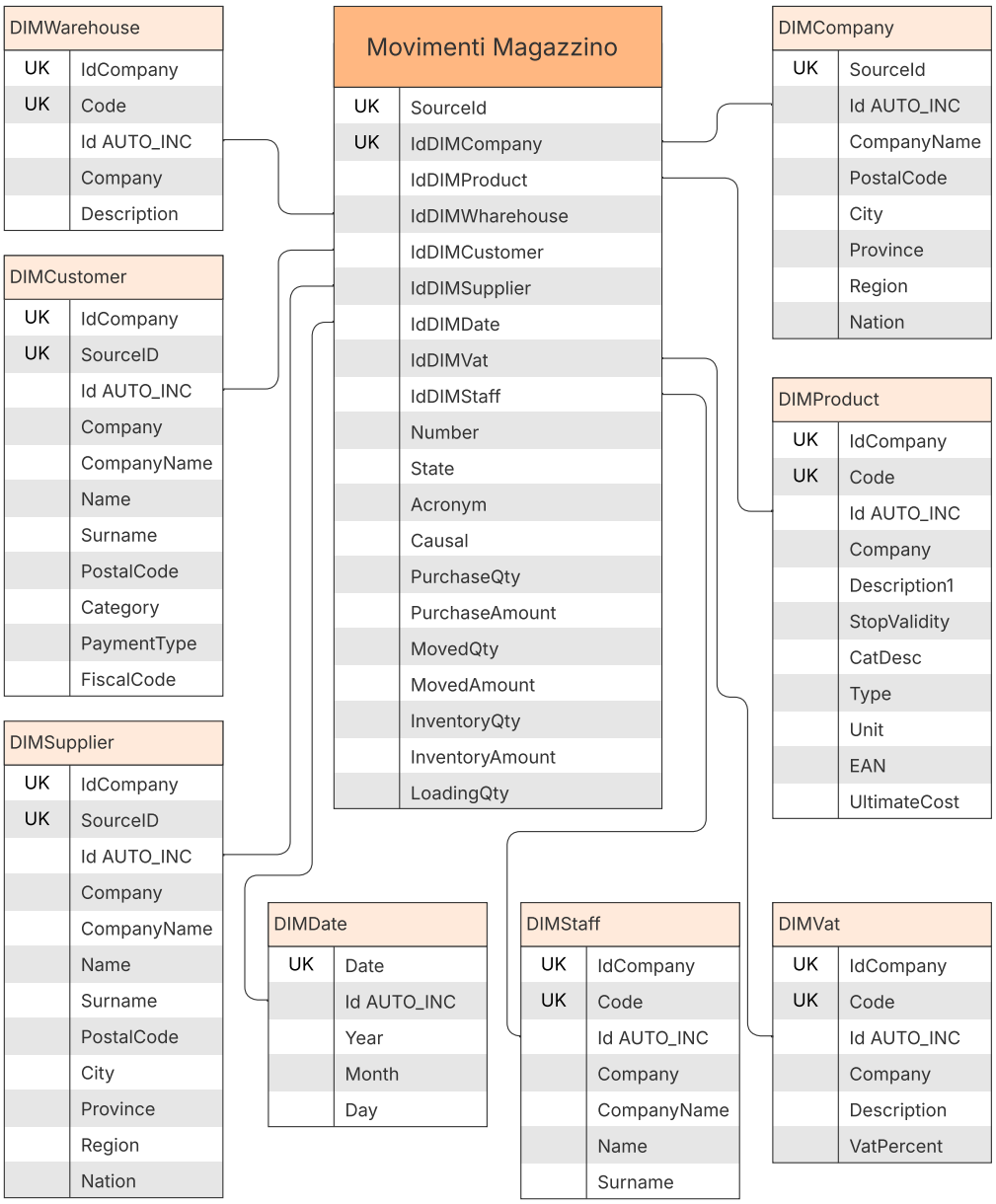

Figura 3.2: Schema a stella relativo ai movimenti di magazzino adattato per Apache Doris

per supportare lo scenario temporale di *attualizzazione*, in cui si vuole che i dati aggiornati sovrascrivano i precedenti senza tenere traccia delle modifiche.

Pur essendo uno strumento potente per velocizzare le interrogazioni, le **viste materializzate** non sono state utilizzate. Questo per via delle limitazioni che ci sono con il modello Unique, il quale non permette di eseguire operazioni di aggregazione. Le viste materializzate potrebbero tuttavia diventare utili in scenari futuri, ad esempio per limitare l'accesso a certi dati a determinati utenti, oppure per implementare il *drill-across*, ovvero la possibilità di collegare due cubi dimensionali distinti.

In base alle raccomandazioni ufficiali di Apache Doris, non è stato ritenuto necessario applicare un **partizionamento**. Attualmente, la tabella più grande occupa circa 1 GB di dati, mentre il partizionamento viene consigliato solo per tabelle superiori ai 500 GB. In questa configurazione, è stato quindi adottato un unico tablet. Le colonne selezionate per il bucketing corrispondono alle colonne chiave della tabella.

Non è stato definito alcun **indice** aggiuntivo, in quanto le prestazioni elevate sono già garantite dall'architettura colonnare, che ottimizza l'accesso ai dati per le operazioni analitiche.

3.2.3 Approccio multi-tenant

Per supportare la multi-tenancy, sono state prese in considerazione due possibili strategie:

1. **Duplicare lo schema per ogni tenant**: con questa soluzione la struttura a stella viene replicata per ogni singolo tenant. Ciascun tenant dispone quindi delle proprie tabelle isolate. Questo approccio semplifica la gestione dei dati, ma incrementa il numero totale di tabelle e, come osservato nella sezione 2.5.4, un elevato numero di tabelle peggiora le prestazioni di Apache Doris.

2. **Utilizzare un'unica tabella con un attributo tenant_id**: con questa soluzione ogni tabella contiene una colonna che identifica il te-

nant. In fase di query, i dati devono essere filtrati in base a questo attributo per garantire l'accesso isolato. Questo approccio riduce la proliferazione di schemi e tabelle, ma può complicare la gestione di partizionamento e sicurezza.

Si è optato per la prima soluzione, ovvero duplicare lo schema per ogni tenant. In questo modo i dati sono separati by design e diventa molto meno probabile che, a causa di un errore di programmazione o di configurazione, ci sia una condivisione non autorizzata di informazioni tra tenant diversi.

3.2.4 ETL

Architettura di riferimento

Per il processo di ETL, è stato preso come riferimento un modello presentato da AWS in un articolo relativo a Amazon Redshift [18].

La fig. 3.3 mostra l'architettura proposta. In particolare, questa soluzione prevede l'adozione di un ELT. Un ELT si differenzia da un ETL tradizionale per il fatto che i dati grezzi vengono caricati direttamente nel DW e trasformati internamente, invece di eseguire le trasformazioni prima del caricamento sul DW.

Le fasi previste sono descritte di seguito:

1. **Estrazione dei dati**: i dati vengono estratti dalle sorgenti operative e caricati su S3, che funge da Data Lake.

2. **Caricamento dei dati sul DW**: a questo punto sul DW vengono create delle tabelle di *Staging*, popolate con i dati grezzi provenienti da S3.

3. **Trasformazione dei dati**: in questa fase i dati vengono letti dalle tabelle di Staging, trasformati e caricati sulle tabelle dello schema a stella. Vengono prima popolate le tabelle delle dimensioni e poi quelle dei fatti, effettuando i JOIN con le dimensioni.

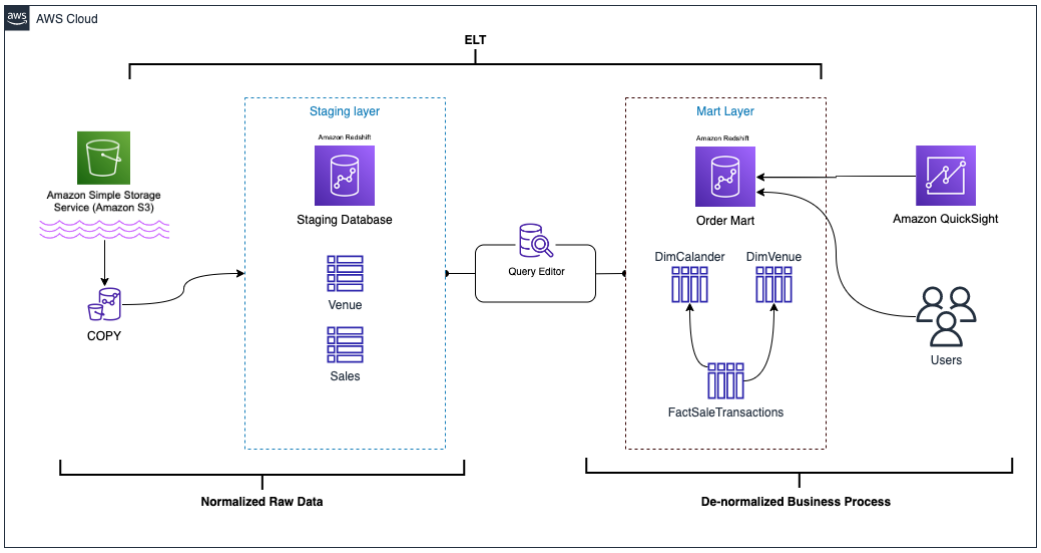

Figura 3.3: Architettura per il processo di ELT proposta da AWS

Architettura adottata

Considerando le elevate prestazioni di Apache Doris nella lettura e nella trasformazione dei dati, è stata adottata un'architettura ELT simile a quella appena descritta. L'approccio segue queste fasi:

1. **Estrazione e caricamento su S3**: i dati vengono estratti dalle sorgenti operazionali e caricati su S3, utilizzato come Data Lake.

2. **Lettura diretta da S3**: per questo punto sono state sfruttate le funzionalità di Lakehouse di Apache Doris [19]. In particolare, viene utilizzata una funzionalità avanzata chiamata Table Value Function (TVF) [20], che consente di leggere direttamente i file presenti su S3, trattandoli come tabelle virtuali senza doverli prima importare fisicamente nel database. Questo elimina la necessità di creare tabelle di staging, riducendo i passaggi intermedi. Grazie a questa funzionalità, il processo può essere visto come un ELT ibrido, dove i dati grezzi non vengono caricati su Doris ma rimangono in S3 fino al momento in cui vengono elaborati e trasformati nelle tabelle dello schema a stella. L'uso

di S3 come Data Lake consente di gestire grandi volumi di dati senza appesantire Doris con tabelle temporanee.

3. **Popolamento dello schema a stella**: La combinazione della TVF con il comando *INSERT INTO SELECT* [21] permette di leggere i file CSV da S3 e inserire i dati direttamente nelle tabelle del DW, eseguendo in questo punto eventuali JOIN, pulizie e trasformazioni.

Processo di popolamento Il caricamento dei dati nel DW segue un ordine preciso, in modo da garantire i corretti riferimenti tra le tabelle:

1. **Caricamento delle dimensioni principali**: le dimensioni della data (DIMDate) e del punto vendita (DIMCompany) vengono popolate per prime, in quanto contengono solo un campo chiave e non dipendono da altre entità.

2. **Caricamento delle altre dimensioni**: ogni altra dimension table ha come chiave due campi, l'id surrogato del punto vendita e l'identificatore della riga nelle sorgenti originali (SourceId o Code). Questo permette di allineare i dati provenienti dalle sorgenti mantenendo un riferimento univoco per ogni riga.

3. **Caricamento dei fatti**: per la fact table è necessario associare a ciascun record i relativi id surrogati delle dimensioni.

UDF

In Apache Doris, le *User-Defined Functions* (UDF) consentono di estendere il linguaggio SQL con funzioni personalizzate, scritte in Java, per gestire trasformazioni e manipolazioni dei dati. Nel contesto del processo di ETL, questo meccanismo risulta particolarmente utile per la pulizia e il casting di dati provenienti da file CSV, in quanto i valori letti in questo modo vengono inizialmente interpretati come semplici stringhe.

Per creare una UDF in Java, è sufficiente implementare una classe che estenda le interfacce fornite da Doris e compilare il codice per produrre un

jar. Il jar deve essere caricato su una destinazione accessibile da Doris e successivamente è possibile registrare la funzione su Doris tramite un comando SQL [22].

Sono state implementate delle UDF per i seguenti casi d'uso:

- **Parsing delle date**: conversione da stringhe di testo a tipi riconosciuti da Doris come data. In questo modo si possono gestire più formati di input e intercettare errori, dovuti a valori mancanti o date in formati non validi.
- **Validazione e sanitizzazione di stringhe**: verifica della lunghezza dei campi di testo, in modo da rispettare i vincoli della tabella di destinazione.
- **Gestione di enumerazioni**: conversione di codici numerici in valori significativi in base alle regole di business.
- **Conversione e controlli numerici**: interpretazione delle stringhe di testo come dati numerici, con gestione di eventuali errori e arrotondamenti.
- **Gestione campi vuoti**: nei casi in cui la tabella di destinazione richieda un valore NOT NULL, le UDF permettono di definire un valore di default o di generare un'eccezione in caso di dati mancanti.

3.3 Analisi OLAP

Gli strumenti di BI sono essenziali per estrarre informazioni dai dati e creare conoscenza a supporto dei processi decisionali dell'azienda.

Come già citato in precedeza, lo strumento utilizzato in questo progetto è DevExpress, strumento di BI che permette di fare reportistica, costruire cruscotti con grafici avanzati ed eseguire sessioni OLAP. Tra le funzionalità troviamo anche la possibilità di generare report dettagliati ed esportare i risultati in diversi formati, tra cui JSON, CSV e PDF.

Le principali operazioni che possono essere fatte in una sessione OLAP includono:

- **Drill-down**: permette di esplorare i dati con un livello di dettaglio maggiore, passando da una visione aggregata a dati più granulari. Ad esempio, da una visualizzazione delle vendite per anno si può scendere fino al dettaglio mensile o giornaliero.
- **Roll-up**: esegue l'operazione inversa del drill-down, aggregando i dati per ottenere una visione più generale. Ad esempio, si può passare da un dettaglio giornaliero a un riepilogo mensile o annuale.
- **Slicing**: consente di selezionare un sottoinsieme specifico dei dati filtrando una determinata dimensione. Ad esempio, si può visualizzare solo l'andamento delle vendite per uno specifico punto vendita.
- **Dicing**: estende il concetto di slicing permettendo di selezionare più dimensioni contemporaneamente. Ad esempio, si può analizzare l'andamento delle vendite per punto vendita e categoria di prodotto simultaneamente.

3.3.1 Esempi

DevExpress permette di creare dashboard personalizzate, con la possibilità di eseguire analisi multidimensionali in tempo reale. Per dimostrare il corretto funzionamento del sistema, sono state consultate alcune dashboard e sono stati navigati i dati simulando una sessione OLAP.

La fig. 3.4 mostra una dashboard dedicata al monitoraggio delle vendite aziendali. Questa interfaccia include diversi elementi interattivi:

- Una barra di selezione dell'intervallo temporale.
- Un riepilogo dei ricavi, che mostra il totale delle vendite.
- Una tabella con indicato il ricavo di ogni punto vendita.

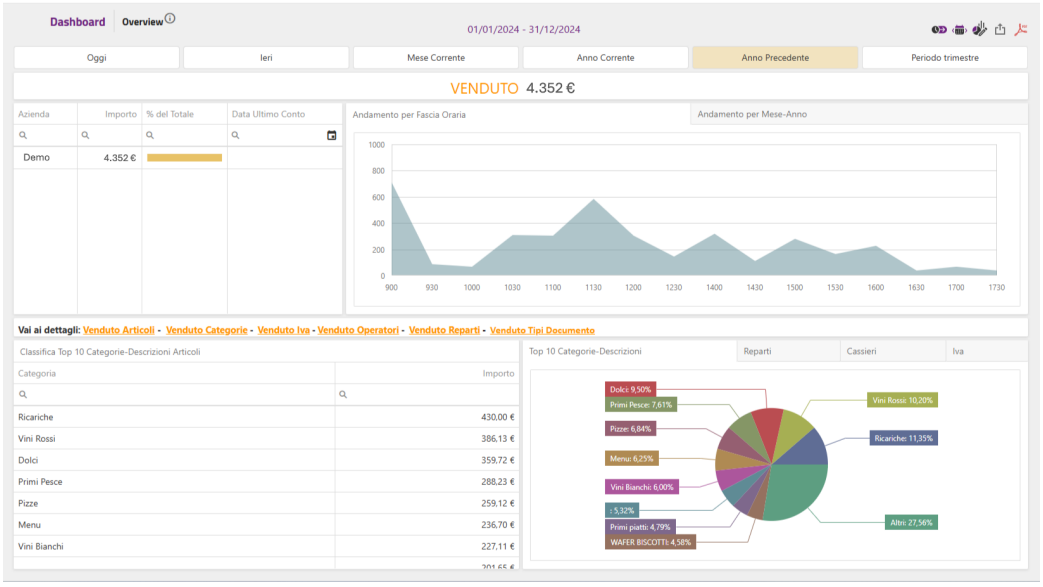

Figura 3.4: Dashboard per monitorare l'andamento delle vendite dell'azienda

- Un grafico a linee che mostra l'andamento delle vendite in base all'orario, utile per individuare i picchi di attività durante la giornata.
- Un grafico a torta per la distribuzione delle categorie di prodotto, che evidenzia le categorie più vendute.
- Una tabella dettagliata con la classifica delle categorie di prodotto più vendute.

L'utente può interagire con questi elementi con operazioni OLAP, come ad esempio fare drill-down sul grafico a torta o sulla tabella della classifica delle categorie, in modo da ottenere informazioni mirate.

La fig. 3.5 rappresenta una dashboard utilizzata per il monitoraggio dei movimenti di magazzino. Questa interfaccia è particolarmente utile per controllare lo stato delle scorte e tracciare le operazioni di carico e scarico. Gli elementi principali sono:

- La selezione del periodo temporale, con la possibilità di selezionare due periodi per fare un confronto tra di essi.

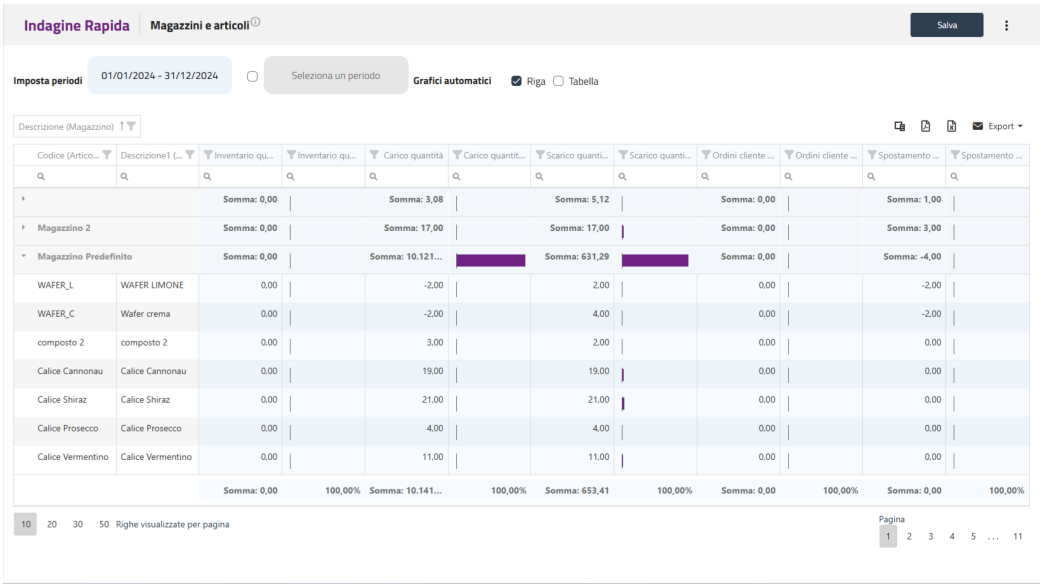

Figura 3.5: Dashboard per monitorare i movimenti di magazzino

- Una tabella con il dettaglio degli articoli in ogni magazzino che include informazioni sulla quantità disponibile, le operazioni di carico e scarico e l'inventario attuale.
- Indicatori grafici di riepilogo, che mettono in evidenza eventuali variazioni significative nelle scorte.

Anche in questo caso sono possibili alcune operazioni OLAP, tra cui il filtraggio su alcuni campi e il drill-down o roll-up sui magazzini.

Conclusioni

L'obiettivo di questa tesi è stato quello di individuare un DBMS adatto alla migrazione di un data warehouse in un ambiente cloud multi-tenant. Questo obiettivo è stato raggiunto attraverso un'analisi comparativa che ha portato all'identificazione di Apache Doris come la soluzione più adeguata.

La prima fase del lavoro ha riguardato l'identificazione delle problematiche del sistema esistente e la definizione dei requisiti da rispettare. Le principali criticità individuate riguardavano le scarse prestazioni, la gestione separata delle installazioni per ciascun cliente e l'impossibilità di effettuare analisi approfondite su un dataset aggregato con i dati di tutti i tenant. I requisiti fondamentali per la nuova soluzione erano quindi: scalabilità, supporto multi-tenant, costi contenuti e alte prestazioni.

Dopo una prima selezione di quattro possibili DBMS (Microsoft SQL Server, Amazon Redshift, Citus PostgreSQL e Apache Doris), è stato condotto un benchmark per valutare le prestazioni di ciascun sistema nell'esecuzione di interrogazioni analitiche in assenza di concorrenza. Per questo, è stato necessario definire un set di 17 query OLAP rappresentative di scenari reali, utilizzate per misurare i tempi di risposta.

I risultati hanno evidenziato come Citus PostgreSQL e Amazon Redshift non siano adatti per il sistema richiesto. Citus ha mostrato tempi di risposta troppo elevati, intorno ai due minuti per alcune query. Redshift non ha offerto prestazioni adeguate alla prima esecuzione delle query, aspetto da tenere in considerazione in quanto il sistema è pensato per permettere interrogazioni dinamiche e personalizzate per ogni tenant. SQL Server con indici colonnari

e Apache Doris hanno invece mostrato prestazioni molto promettenti sia per la prima esecuzione delle query che per le successive.

Visti i risultati del benchmark eseguito in assenza di concorrenza, è stato effettuato un test di carico solo sui due DBMS risultati adeguati dal punto di vista delle prestazioni, SQL Server con indici Column Store e Apache Doris. L'obiettivo a questo punto è stato quello di individuare quanti utenti il sistema fosse in grado di supportare mantenendo un tempo di risposta alle interrogazioni inferiore ai 20 secondi, tenendo conto dell'esecuzione di una query da parte di ogni utente ogni 60-120 secondi.

I risultati hanno mostrato come SQL Server sia in grado di mantenere tutte le query al di sotto dei 20 secondi fino a 10 utenti, mentre Apache Doris riesca a raggiungere i 65 utenti. Anche tenendo in considerazione la media dei tempi di esecuzione con la soglia dei 20 secondi, Apache Doris ha confermato la sua superiorità con 75 utenti supportati rispetto ai 35 di SQL Server.

A questo punto è stata valutata la capacità di scalare di Apache Doris, sia in termini di risorse hardware che di crescita del numero di tenant.

Per quanto riguarda le risorse hardware, l'istanza è stata scalata verticalmente raddoppiando la RAM e il numero di core della CPU. Siamo quindi passati da 4 core e 32 GB di RAM a 8 core e 64 GB di RAM. Rieseguendo il test di carico, si è osservato come il raddoppiare le risorse abbia portato a un aumento di prestazioni tangibile. Con questa configurazione il numero di utenti supportati è passato da 65 a 105 (vincolo dei 20 secondi su tutte le query), e da 75 a 125 (vincolo dei 20 secondi sul tempo medio delle query).

Per quanto riguarda la crescita del numero di tenant, sono stati inizialmente aggiunti 900 nuovi schemi, portando il totale a 1000 tenant. Questo incremento non ha portato a un deterioramento tangibile delle prestazioni, rimanendo con lo stesso numero di utenti supportati. Successivamente, sono stati aggiunti altri 2000 schemi e sono state triplicate le tabelle per ogni schema, in modo da simulare l'aggiunta di nuovi fatti e dimensioni al data warehouse. Questo ha portato il numero di utenti supportati da 105 a 75

(vincolo dei 20 secondi su tutte le query) e da 125 a 85 (vincolo dei 20 secondi sul tempo medio delle query). Con questo test si è dimostrato come la presenza di più schemi e tabelle, quindi di più metadati, influisca negativamente sulle prestazioni di Apache Doris.

Infine è stata implementata una proof of concept per dimostrare la fattibilità della migrazione. In questa fase vengono apportate delle modifiche allo schema a stella, in modo da sfruttare le caratteristiche e le funzionalità di Apache Doris. Doris non ha lo stesso concetto di primary key e foreign key che è presente nei classici RDBMS, per questo le tabelle sono state definite con il tipo di chiave Unique specifico di Doris. Viene anche descritto il processo di ETL, che si trasforma in un ELT sfruttando le capacità di lakehouse di Doris: l'object storage S3 viene utilizzato come data lake per salvare i dati provenienti dalle sorgenti operazionali; la trasformazione e il caricamento degli stessi avviene con query eseguite su Apache Doris, sfruttando funzionalità come TVF per l'accesso ai dati in S3 e UDF per ampliare il linguaggio SQL con funzioni personalizzate implementate in Java.

Il corretto funzionamento del sistema è stato validato mostrando alcune dashboard interattive per analisi OLAP, che permettono operazioni di drill-down e roll-up sui dati.

Sviluppi futuri Tra le attività da portare avanti in futuro, la migrazione completa di tutti i tenant verso il nuovo ambiente cloud multi-tenant rimane un punto essenziale.

Per un'analisi ancora più approfondita, si potrebbe valutare l'utilizzo di un cubo più complesso rispetto a quello utilizzato in questa tesi, includendo archi multipli, attributi cross-dimensionali e gerarchie ricorsive, elementi che potrebbero mettere alla prova le prestazioni e l'occupazione dello spazio.

Bibliografia

[1] Matteo Golfarelli and Stefano Rizzi. *Data warehouse: Teoria e pratica della progettazione*. McGraw-Hill, 2006.

[2] William H. Inmon. *Building the Data Warehouse*. QED Information Sciences, 1996.

[3] Laurent Bonnet, Anne Laurent, Michel Sala, Benedicte Laurent, and Nicolas Sicard. Reduce, you say: What nosql can do for data aggregation and bi in large repositories. In *2011 22nd International Workshop on Database and Expert Systems Applications*, page 483–488. IEEE, August 2011. doi: 10.1109/dexa.2011.71. URL `http://dx.doi.org/10.1109/DEXA.2011.71`.

[4] Rania Yangui, Ahlem Nabli, and Faiez Gargouri. Automatic transformation of data warehouse schema to nosql data base: Comparative study. *Procedia Computer Science*, 96:255–264, 2016. ISSN 1877-0509. doi: 10.1016/j.procs.2016.08.138. URL `http://dx.doi.org/10.1016/j.procs.2016.08.138`.

[5] Mohamed Boussahoua, Omar Boussaid, and Fadila Bentayeb. *Logical Schema for Data Warehouse on Column-Oriented NoSQL Databases*, page 247–256. Springer International Publishing, 2017. ISBN 9783319644714. doi: 10.1007/978-3-319-64471-4_20. URL `http://dx.doi.org/10.1007/978-3-319-64471-4_20`.

[6] Sandro Bimonte, Enrico Gallinucci, Patrick Marcel, and Stefano Rizzi. Logical design of multi-model data warehouses. *Knowledge and Information Systems*, 65(3):1067–1103, November 2022. ISSN 0219-3116. doi: 10.1007/s10115-022-01788-0. URL http://dx.doi.org/10.1007/s10115-022-01788-0.

[7] IBM. Cloud computing, 2024. URL https://www.ibm.com/it-it/topics/cloud-computing. Archived at https://web.archive.org/web/20250213084422/https://www.ibm.com/it-it/topics/cloud-computing.

[8] Qlik. Cloud data warehouse, 2025. URL https://www.qlik.com/us/cloud-data-migration/cloud-data-warehouse. Archived at https://web.archive.org/web/20250213084238/https://www.qlik.com/us/cloud-data-migration/cloud-data-warehouse.

[9] Transaction Processing Performance Council. TPC - Transaction Processing Performance Council, 2025. URL https://www.tpc.org/.

[10] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. *The Star Schema Benchmark and Augmented Fact Table Indexing*, page 237–252. Springer Berlin Heidelberg, 2009. ISBN 9783642104244. doi: 10.1007/978-3-642-10424-4_17. URL http://dx.doi.org/10.1007/978-3-642-10424-4_17.

[11] Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. Implementing multidimensional data warehouses into nosql. In *Proceedings of the 17th International Conference on Enterprise Information Systems*, page 172–183. SCITEPRESS - Science and and Technology Publications, 2015. doi: 10.5220/0005379801720183. URL http://dx.doi.org/10.5220/0005379801720183.

[12] Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. *Implementation of Multidimensional Da-*

tabases in Column-Oriented NoSQL Systems, page 79–91. Springer International Publishing, 2015. ISBN 9783319231358. doi: 10.1007/978-3-319-23135-8_6. URL http://dx.doi.org/10.1007/978-3-319-23135-8_6.

[13] Bogdan George Tudorica and Cristian Bucur. A comparison between several nosql databases with comments and notes. In *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*, page 1–5. IEEE, June 2011. doi: 10.1109/roedunet.2011.5993686. URL http://dx.doi.org/10.1109/RoEduNet.2011.5993686.

[14] Khaled Dehdouh, Fadila Bentayeb, Omar Boussaid, and Nadia Kabachi. Using the column oriented nosql model for implementing big data warehouses. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 469–474, 2015.

[15] Microsoft. Columnstore indexes overview, 2025. URL https://learn.microsoft.com/it-it/sql/relational-databases/indexes/columnstore-indexes-overview. Archived at https://web.archive.org/web/20250218125653/https://learn.microsoft.com/it-it/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver16.

[16] Amazon Web Services. Query Performance - Amazon Redshift Database Developer Guide, 2025. URL https://docs.aws.amazon.com/redshift/latest/dg/c-query-performance.html. Archived at https://web.archive.org/web/20250302123636/https://docs.aws.amazon.com/redshift/latest/dg/c-query-performance.html.

[17] Apache Doris. Table model overview, 2025. URL https://doris.apache.org/docs/table-design/data-model/overview. Archived at https://web.archive.org/web/20250302123647/https://doris.apache.org/docs/table-design/data-model/overview/.

[18] AWS Big Data Blog. Dimensional modeling in amazon redshift, 2022. URL https://aws.amazon.com/it/blogs/big-data/dimensional-modeling-in-amazon-redshift/. Archived at https://web.archive.org/web/20250206144637/https://aws.amazon.com/it/blogs/big-data/dimensional-modeling-in-amazon-redshift/.

[19] Apache Doris. Lakehouse file formats in apache doris, 2025. URL https://doris.apache.org/docs/lakehouse/file. Archived at https://web.archive.org/web/20250302123840/https://doris.apache.org/docs/lakehouse/file/.

[20] Apache Doris. S3 table-valued functions in apache doris, 2025. URL https://doris.apache.org/docs/sql-manual/sql-functions/table-valued-functions/s3. Archived at https://web.archive.org/web/20250302123729/https://doris.apache.org/docs/sql-manual/sql-functions/table-valued-functions/s3/.

[21] Apache Doris. Insert into manual - apache doris, 2025. URL https://doris.apache.org/docs/data-operate/import/import-way/insert-into-manual. Archived at https://web.archive.org/web/20250302123936/https://doris.apache.org/docs/data-operate/import/import-way/insert-into-manual.

[22] Apache Doris. Java user-defined function (udf), 2025. URL https://doris.apache.org/docs/query-data/udf/java-user-defined-function. Archived at https://web.archive.org/web/20250302123824/https://doris.apache.org/docs/query-data/udf/java-user-defined-function/.

Appendice A

Listati di Codice

Listing A.1: Codice per il timer con seed fisso

```
// Get current thread number
def threadNum = ctx.getThreadNum()

// Check if Random object is already defined for this
   thread
def random = vars.getObject("random_wait_" + threadNum)
if (random == null) {
        // Initialize the Random with seed to ensure
           repeatability
        random = new Random(1234 + threadNum)
        // Save the Random object to reuse it
        vars.putObject("random_wait_" + threadNum,
           random)
}

// Set the delay time between minWait and maxWait
def minWait = 60 * 1000 // 60 seconds in milliseconds
def maxWait = 120 * 1000 // 120 seconds in milliseconds
def waitTime = minWait + random.nextInt(maxWait -
   minWait + 1)

vars.put("wait-time", Integer.toString(waitTime))

// To set the timer return the value
return waitTime
```

Listing A.2: Codice per la selezione della query con seed fisso

```
//...

def schemaNumber = (threadNum % 100) + 1
def schemaName = "cliente" + schemaNumber

// Directory containing the SQL files
def sqlDirectoryPath = "C:\\path\\to\\queries"

// Iterate to all files in the directory
directory.eachFile(FileType.FILES) { file ->
    if (file.name.endsWith('.sql')) {
        // Read file content
        def content = file.text
        // Replace {schemaName} in the .sql with
           schemaName value
        def updatedContent = content.replaceAll("\\{
           schemaName\\}", schemaName)
        // Get the query number from the file name
        def queryNumber = file.name.replaceAll(".sql", "
           ")
        // Add the query to the list
        queriesMap[queryNumber] = updatedContent
    }
}

// Select a random entry from the map
def randomEntry = queriesMap.entrySet().toArray()[
    random.nextInt(queriesMap.size())]
def randomQuery = randomEntry.value

//...
```