# CONFRONTO TRA DBMS RELAZIONALI, A COLONNE E NOSQL

**Università degli Studi di Modena e Reggio Emilia**

**Dipartimento di Ingegneria “Enzo Ferrari” di Modena**

*Corso di Laurea in Ingegneria Informatica (L.270/04)*

Tutor

Prof. Sonia Bergamaschi

Co-Tutor

Dott. Giovanni Simonini

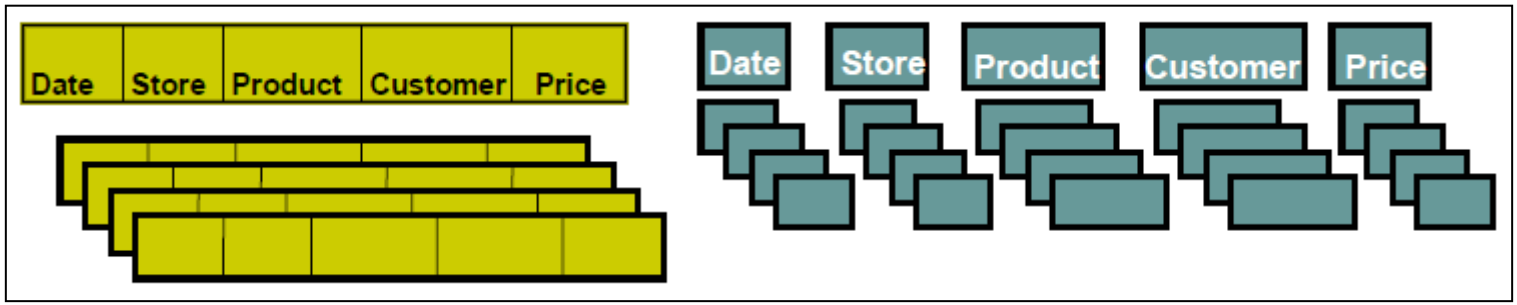
Candidato

Matteo Senardi

# Overview sui sistemi DBMS

L'elaborato valuta le possibilità offerte dai DBMS orientati a colonne.

- **Database system orizzontali**: tradizionali DBMS relazionali.
- **Database system orientati a colonne**: nuovi sistemi proposti per datawarehousing e read/only reporting, OLAP (*On-Line Analytical Processing*) → insieme di tecniche software per l'analisi interattiva e veloce di grandi quantità di dati.
- I DBMS a colonne memorizzano i dati a colonne invece che a righe.



# Caratteristiche dei DBMS a colonne

- In un DBMS relazionale una ipotetica tabella anagrafica avrebbe la seguente struttura:

| ID | COGNOME | NOME | DATA DI NASCITA |
|---|---|---|---|
| 1 | Rossi | Mario | 01/01/1960 |
| 2 | Verdi | Giuseppe | 01/01/1961 |
| 3 | Bianchi | Antonio | 01/01/1962 |

- Le informazioni sarebbero memorizzate nel seguente modo: 1, Rossi, Mario, 01/01/1960; 2, Verdi, Giuseppe, 01/01/1961; 3, Bianchi, Antonio, 01/01/1962;
- In un DBMS orientato a colonne le stesse informazioni vengono divise in colonne: 1, 2, 3; Rossi, Verdi, Bianchi; Mario, Giuseppe, Antonio; 01/01/1960, 01/01/1961, 01/01/1962;
- La tabella memorizza una colonna alla volta e alla fine procede con la memorizzazione della colonna successiva. Oltre alla colonna è memorizzata una chiave surrogata → necessaria per ricostruire il record.

# Benefici dei DBMS a colonne

- Query recupera solo valori da determinate colonne e **non da tutta la riga.**
- Colonne, composte da tipi di dati uniformi, più facili da **comprimere** → **aumento velocità di esecuzione e storicizzazione** arrivando a gestire svariati petabyte.

# Sap Hana

Sviluppato e commercializzato da Sap nel 2012 → punto di incontro tra tecnologia relazionale e colonnare traendo i migliori aspetti da entrambe e costituendo una sorta di passaggio graduale tra le due tecnologie → miglioramento prestazioni grazie al motore **in-memory**.

- Memorizza una tabella nella colonna di archiviazione come sequenza di colonne in locazioni di memoria contigue → **massimizza località spaziale delle colonne della tabella**.
- CPU accede ai dati in modo consecutivo senza tempi di attesa per operazioni di indirizzamento in memoria → **velocità di esecuzione elevate**.
- Storage delle tabelle in colonna → **compressione molto elevata dei dati**.

# Sistemi NoSQL

NOSQL acronimo di Not Only SQL → non contrario a utilizzo di database relazionali: diversi casi d'uso per cui modello relazionale rappresenta una forzatura, ma anche tanti altri per cui è ancora la soluzione migliore.

Caratteristiche:

- Notevole utilizzo nel **campo industriale con big data** e applicazioni web in tempo reale.
- Meno vincolati rispetto a DBMS tradizionali, no schema fisso (schemaless), evitano join e scalano orrizzontalmente → spesso estremamente **ottimizzati nelle memorizzazioni chiave-valore** destinate a semplici operazioni di recupero e aggiornamento.
- Obiettivo → significativi vantaggi prestazionali in termini di **latenza** e **throughput.**

Esempio Cassandra sviluppato da Facebook per ricerca all'interno dei messaggi inbox. Vi sono chiavi corrisposte a valori raggruppati in famiglie di colonne.

# Vertica

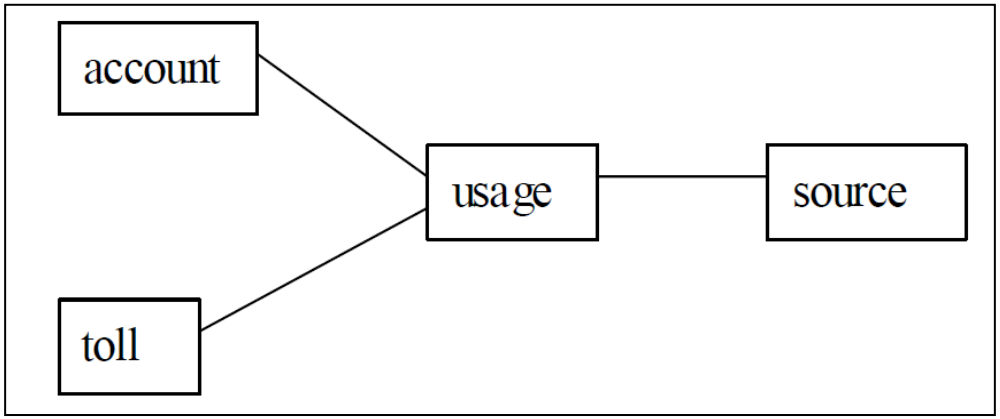
Fondata nel 2005 da Michael Stonebraker e acquistato da Hewlett Packard nel 2011.

Caratteristiche di modello:

- Storage orientato a colonna → **aumento prestazioni di accesso a record sequenziali** a **scapito di operazioni transazionali comuni come singolo recupero di record, aggiornamenti ed eliminazioni**. Query di data warehouse recuperano dati solo in determinate colonne → **riduzione I/O**.
- Piccolo storage (circa 1 gigabyte) a riga tradizionale dedicato a ricevere aggiornamenti, i cui contenuti sono periodicamente spostati in massa nello store di colonna → **in memoria principale, quindi super-veloce**.
- Interfaccia standard **SQL** e supporto a altre interfacce di programmazione.
- Alta compressione → **riduzione costi di storage e larghezza di banda di I/O**, poiché le colonne di dati di tipo omogeneo sono archiviate insieme.

# Telco example

Caso d'uso dettagliato (ricerca del MIT, "*One Size Fits All? – Part 2: Benchmarking Results*") → schema di struttura a stella prodotto da una ditta specializzata in analisi di business di compagnie telefoniche → situazione in cui il cliente necessita di mantenere grossi volumi di dati di tipo storico e di accedervi secondo criteri di volta in volta differenti.




- Tabella centrale **usage** (fact table) → record per ogni telefonata con una varietà di dati dettagliati sulle chiamate.
- Tabella **account** → numeri di telefono che rappresentano i soggetti di fatturazione.
- Tabella **source** → rete specifica sulla chiamata con relativa provenienza.
- Tabella **toll** → informazioni di fatturazione.

# Telco example

```
SELECT account.account_number,
       sum (usage.toll_airtime),
       sum (usage.toll_price)
FROM   usage, toll, source, account
WHERE  usage.toll_id = toll.toll_id
  AND  usage.source_id = source.source_id
  AND  usage.account_id = account.account_id
  AND  toll.type_ind in ('AE'. 'AA')
  AND  usage.toll_price > 0
  AND  source.type != 'CIBER'
  AND  toll.rating_method = 'IS'
  AND  usage.invoice_date = 20051013
GROUP BY  account.account_number
```

Scrittura tipica per query di data ware-house: inizialmente filtra utilizzando i predicati sulle colonne della tabella usage (fact table) o di una tabella dimensionale; in seguito raggruppa la tabella usage, precedentemente ristretta, secondo qualche attributo e aggregato → Memorizzazione di colonna (Vertica su CPU Opteron 2500$) superiore di quella riga (non-nominato DBMS orizzontale su macchina blade-28 300.000$ circa) di un fattore 47, con utilizzo di CPU pari a 1/7 e costo hardware conseguentemente inferiore di ben due ordini di grandezza.

# Telco example

Diverse ragioni a giustificare la differenza di prestazioni:

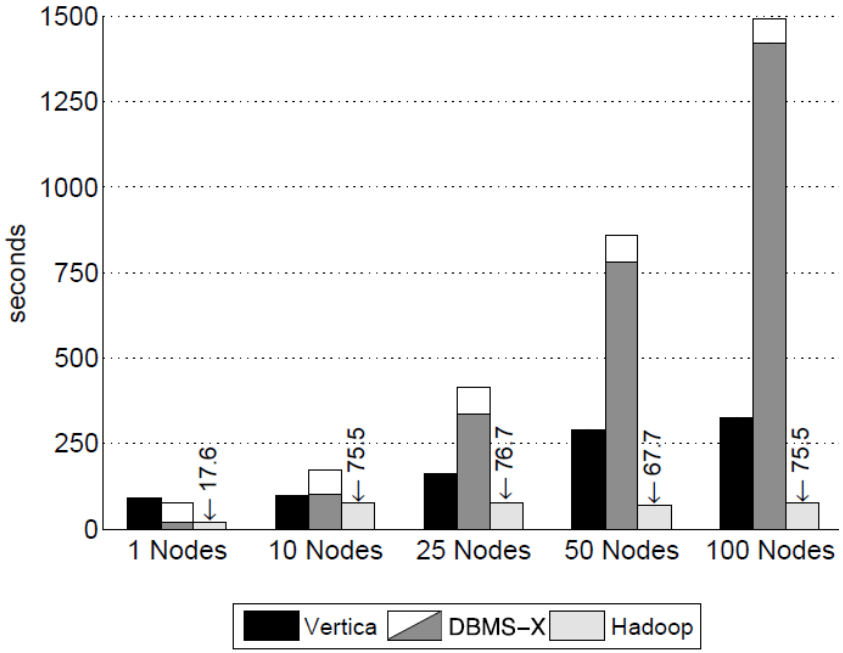
- La tabella usage contiene una miriade di attributi sulle telefonate, più di 200 colonne → complessità nell'andare a operare su una tabella tanto "piena". La **query legge esattamente 7 su 212 colonne** per operazione di store in colonna , mentre corrispondente operazione in riga leggerà tutte le 212 colonne → differenza notevole di quasi due ordini di grandezza in termini di flusso di byte dal disco.
- Compressione, solitamente più efficace per store di colonna → **oggetti posizionati in un blocco del disco con dati dello stesso tipo** e opzioni di compressione aggiutive. Vertica rapporto di compressione con fattore 10 → nettamente migliore rispetto a fattore di comprensione 3 possibile tramite le operazioni in riga concorrenti.
- **Ordinamento e indicizzazione a carico di Vertica** e non del calcolatore → per la macchina tempo di query costante, Vertica limita i tempi di utilizzo di alcune query.

# Vertica vs MapReduce and Horizontal DBMS

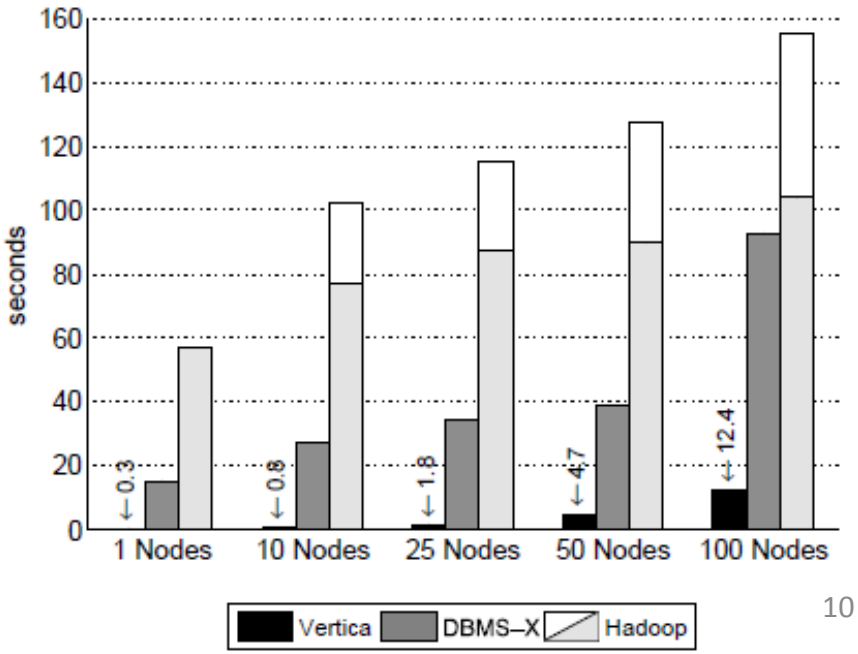
Benchmark del SIGMOD luglio 2009. Ad oggi le fonti bibliografiche in merito non consentono di verificarne gli esiti ovvero di confermarne i risultati in relazione a evoluzione e ottimizzazione dei sistemi.

**Tempi di caricamento per**

**Grep Task Data Set 535MB/nodo**




**Risultati Selection Task**



# Analisi dei risultati e conclusioni

A conclusione, l'articolo *"A Comparison of Approaches to Large-Scale Data Analysis"* giunge alle conclusioni che espongo:

**PRO**

- In contrasto con tempo di avviamento dell'approccio MapReduce, entrambi DBMS sono **avviati al boot del sistema operativo** → sempre "**caldi**", in attesa di query da eseguire, utilizzando più thread e processi.
- Vertica struttura interna altamente ottimizzata per compressione dati e motore di esecuzione che **opera direttamente sui dati compressi** → evita decompressione dati durante la lavorazione, ove possibile.
- Possibilità di **riorganizzare dati in ingresso** al momento del caricamento → ottimizzazioni, tipo memorizzazione di ogni attributo in tabella a parte, come per sistemi column-store (Vertica), vd. Figg. Selection Task.
- SQL rappresenta standard reale e di fatto → abbastanza **portabile**, essendo possibile **condividere i comandi SQL** tra DBMS relazionali e Vertica con piccole modifiche. Per MapReduce programmi principalmente scritti in Java, nonostante diversi binding compatibili con SQL.

# Analisi dei risultati e conclusioni

**CONTRO**

- Da risultati dei tempi di caricamento del Grep Task Data Set (vd. Figg), Hadoop ha raggiunto un throughput di carico fino a tre volte più veloce di Vertica e quasi 20 volte più veloce di DBMS-X → per **dati** che necessitano di essere **caricati una sola volta** per alcuni tipi di attività di analisi, potrebbe **non valere la pena pagarne costo di indicizzazione e riorganizzazione**, tipici dei DBMS.
- Operazioni di inserimento, aggiornamento e cancellazione dei record hanno bisogno di più tempo → dati column-oriented sono memorizzati in colonne, **più posti necessitano di essere aggiornati**.

DBMS a colonne orientati a gestione di svariati petabytes, e in un certo modo relegati ad usi statistici e scientifici all'interno di grandi data warehouse. Tuttavia con il passare degli anni la mole di dati da gestire e analizzare aumenterà considerevolmente → essenziale affiancare i DBMS a colonne ai classici DBMS relazionali, oggi ancora i più efficienti per gestione dei dati in **applicazioni transazionali**, nonché come riferimento per **impostazione delle strutture**.