Ottimizzazione delle Query per Sistemi di Business Intelligence: Tecniche Avanzate per Migliorare le Performance e il Supporto Decisionale

di

Davide Maria Ferigato

Presentata al Dipartimento di Ingegneria "Enzo Ferrari"
in parziale adempimento dei requisiti per il conseguimento del titolo di

LAUREA TRIENNALE IN INGEGNERIA INFORMATICA
L-08 Ingegneria dell'Informazione

presso

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Anno Accademico 2024/2025

© 2025 Davide Maria Ferigato. Tutti i diritti riservati.

Candidato: Davide Maria Ferigato
Dipartimento di Ingegneria "Enzo Ferrari"
2 dicembre 2025

Relatore: Domenico Beneventano
Professore di Basi di Dati e Lab.

Accettato da: Elisa Ficarra
Presidente della Commissione di Laurea
Dipartimento di Ingegneria "Enzo Ferrari"

A ship in port is safe, but that's not what ships are built for.

Grace Murray Hopper

Abstract

L'elaborato analizza tecniche avanzate per l'ottimizzazione delle query nei sistemi di *Business Intelligence*, considerando architetture di *Data Warehouse* ed ETL. Il lavoro integra teoria e pratica: dalla progettazione fisica (indicizzazione, partizionamento, compressione, *caching*) alla materializzazione delle viste, fino a metriche e *governance* misurabili delle prestazioni. L'obiettivo è definire procedure riproducibili per ridurre latenza e I/O, preservando qualità e aggiornamento dei dati.

La trattazione segue un percorso progressivo. Si richiamano i fondamenti di BI e le differenze tra OLTP e OLAP; si esamina la qualità degli ETL e l'impatto delle metriche strutturali; si approfondiscono le scelte di *design* fisico per carichi analitici. Si inquadra il ruolo delle viste materializzate nella riduzione del costo delle interrogazioni e, nel capitolo applicativo, si analizzano SQL Server e Azure Synapse: requisiti di creazione, riscrittura automatica delle *query*, distribuzione dei dati, statistiche, uso di CTAS e tabelle temporanee, strumenti di diagnostica e *overhead*.

La metodologia combina la revisione della letteratura, un confronto tra approcci classici per selezione e manutenzione delle viste, e un protocollo sperimentale ispirato a *benchmark* decisionali. Le metriche prese in considerazione includono tempi medi e dispersione, costo di manutenzione in presenza di aggiornamenti e impatto su I/O e spazio.

I risultati convergono in linee guida operative: base fisica *columnstore* e partizionamento, viste progettate su *join* e aggregazioni condivise con distribuzione coerente, validazione mediante riscrittura automatica e soglie di *overhead* per il *rebuild*. Nel complesso, l'adozione coordinata di tali tecniche riduce in modo stabile le latenze e migliora l'efficienza globale del *Data Warehouse*, fornendo indicazioni applicabili e replicabili.

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Capitolo 1

Introduzione

Il progresso tecnologico ha trasformato profondamente il modo in cui i dati vengono raccolti, archiviati e analizzati, rendendo la BI indispensabile per il supporto al processo decisionale. La crescente complessità dei sistemi di BI e il costante aumento dei volumi di dati evidenziano la necessità di sistemi in grado di rispondere in modo efficiente e tempestivo alle esigenze delle organizzazioni.

In questo elaborato si presenta un'analisi delle tecniche avanzate di ottimizzazione delle *query* in sistemi di *Business Intelligence*, in riferimento a *Data Warehouse*, progettazione fisica e viste materializzate. L'obiettivo è di analizzare come questi strumenti migliorino le prestazioni e le capacità di supporto al processo decisionale.

L'ottimizzazione delle *performance* non è solo un argomento tecnico, ma è diventata fondamentale per il successo aziendale, visto l'impatto sempre maggiore degli strumenti di analisi dei dati sulla competitività.

La domanda di ricerca a cui si cerca di rispondere è come le tecniche avanzate di ottimizzazione delle *query*, attraverso progettazione fisica, viste materializzate e il *benchmarck* TPC-DS, siano in grado di migliorare le *performance* di sistemi di *Business Intelligence* e supportare il processo decisionale. A tale domanda si risponde con diverse metodologie che includono lo studio e la critica della letteratura, la valutazione di algoritmi specifici e l'analisi di un caso di studio reale. Il lavoro include un'analisi delle pubblicazioni degli ultimi vent'anni in ambito Business Intelligence, finalizzata a individuare *trends* e *gap* del settore.

La letteratura scientifica mostra una grande varietà di approcci per l'ottimizzazione dei *Data Warehouse*, tra cui modellazione multidimensionale, tecniche di ottimizzazione fisica (indicizzazione, partizionamento, compressione, *caching*) e viste materializzate, ma evidenzia ancora criticità a livello di scalabilità, efficienza e trattamento di *Big Data*. L'uso di tecniche *data-driven*, basate su *machine learning*, per l'ottimizzazione delle *query* ha rappresentato un nuovo paradigma di ricerca che

richiede ulteriori test e validazioni empiriche.

A partire da questa premessa, la struttura dell'elaborato è costituita da 7 capitoli principali. Nel capitolo 2 vengono presentati i concetti basilari di *Business Intelligence* e *Data Warehouse*, ponendo enfasi su architetture di sistemi BI, differenze tra *Online Transaction Processing* (OLTP) e *Online Analytical Processing* (OLAP) e tecniche di analisi. Nel capitolo 3 viene trattata l'evoluzione e la qualità dei processi *Extract*, *Transform and Load* (ETL), analizzando le metriche di efficienza e le tecniche di ottimizzazione. Nel capitolo 4 vengono approfondite le principali tecniche di ottimizzazione fisica: indicizzazione, partizionamento, compressione e *caching*. Il capitolo 5 introduce il concetto di viste materializzate e dei loro vantaggi, presentando i principali algoritmi di selezione e manutenzione, con particolare attenzione al bilanciamento tra costi e prestazioni. Il capitolo 6 tratta le pratiche documentate di materializzazione per l'ottimizzazione delle *query* in ambito OLAP, analizzando l'implementazione e le *performance* in SQL Server/Synapse. Infine, nel capitolo 7 vengono riassunte le conclusioni raggiunte, individuando i limiti, le prospettive applicative e gli sviluppi futuri di questo settore.

Capitolo 2

Fondamenti della Business Intelligence e Data Warehouse

La gestione efficace delle informazioni aziendali richiede la conoscenza delle architetture e tecnologie dei sistemi di BI e *Data Warehouse*. Verrà affrontato come si distinguono per scopi operativi e strategici, in quanto aiutano nel processo di *decision making*.

2.1 Architettura e componenti dei sistemi BI

L'architettura dei sistemi di BI è un insieme di *data mining*, *reporting*, OLAP e *data warehouse* (Vaisman & Zimányi, 2022 [1]; Amadeo, 2018 [2]). Tutti gli elementi di un sistema di BI, funzionando in sinergia, elaborano i dati in informazioni di valore. Questo schema garantisce la qualità delle analisi e la rapidità richiesta dalle aziende.

La piattaforma *data warehouse* OLAP è un elemento cruciale dell'architettura di BI, in grado di fornire un ambiente idoneo ad attività di reporting e ad interrogazioni ad *hoc* (Vaisman & Zimányi, 2022)[1]. Lo scopo di OLAP è l'esplorazione e l'aggregazione dei dati rispondendo ai bisogni in continua evoluzione delle imprese. I *database* analitici OLAP differiscono dai *database* transazionali OLTP (Vaisman & Zimányi, 2022 [1]; Inmon, 2002 [3]), in quanto ottimizzati per operazioni di analisi e aggregazione. La differenza tra questi *database* è anche dovuta al loro *design* fisico, e ha un effetto sulla qualità e la puntualità dei dati analitici (Inmon, 2002)[3].

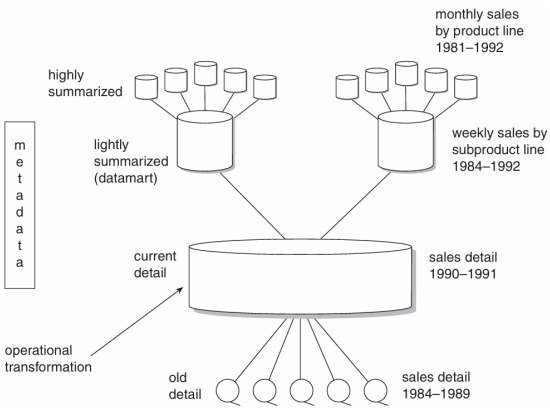


Figura 2.1: Esempio di struttura di *data warehouse*. (Inmon, 2002 p. 36)[3]

L'importanza di OLAP e del *data warehouse* è tale da aver generato diverse tecnologie e prassi organizzative finalizzate a garantire la qualità, la *security* e la disponibilità delle informazioni (Amadeo, 2018 [2]; Chen et al., 2012 [4]).

Nei contesti competitivi, la possibilità di scalare un'architettura di BI risiede proprio nell'abilità del modello dati e degli strumenti OLAP. L'importanza di queste tematiche viene confermata dagli ultimi *trend* scientifici (analisi scientometrica) su questi sistemi, dove le questioni su scalabilità e interoperabilità sono sempre più in aumento in ambiti legati a *Computer Science, Management* e *Industrial Engineering* (Gurcan et al., 2023)[5][6].

Tabella 2.1: Aree tematiche degli articoli di BI. (Gurgan et al. 2023 p. 6)[5]

Subject Area	**N**	**%**
Computer Science	1953	52.09
Business, Management, and Accounting	1152	30.73
Engineering	978	26.09
Decision Sciences	688	18.35
Social Sciences	639	17.04
Mathematics	348	9.28
Medicine	214	5.71
Economics, Econometrics, and Finance	200	5.33
Materials Science	131	3.49
Environmental Science	128	3.41
Energy	112	2.99
Arts and Humanities	111	2.96
Psychology	102	2.72
Chemical Engineering	60	1.60
Physics and Astronomy	58	1.55
Multidisciplinary	50	1.33
Biochemistry, Genetics, and Molecular Biology	49	1.31
Agricultural and Biological Sciences	47	1.25
Earth and Planetary Sciences	46	1.23
Chemistry	35	0.93
Health Professions	29	0.77
Pharmacology, Toxicology, and Pharmaceutics	27	0.72
Nursing	19	0.51
Neuroscience	9	0.24
Immunology and Microbiology	8	0.21
Veterinary	6	0.16
Dentistry	2	0.05

Le tecnologie ETL estrapolano dati da fonti eterogenee e trasformano i dati secondo regole di *business* (Amadeo, 2018 [2]; Inmon, 2002 [3]; Reinschmidt & Francoise, 2000 [7]), fino a caricarli nel *data warehouse*. Questo processo garantisce coerenza e qualità (Inmon, 2002 [3]; Reinschmidt & Francoise, 2000 [7]). Errori e inefficienze nella fase ETL compromettono la validità delle analisi e di conseguenza l'affidabilità delle decisioni (Inmon, 2002)[3]. Ogni processo ETL genera dati analitici, ad esempio estrazioni giornaliere possono essere anche decine di migliaia, necessitando così soluzioni di monitoraggio automatizzate (Inmon, 2002)[3].

I *framework* ETL centralizzati rendono possibile la gestione centralizzata delle regole di *business*, la standardizzazione del processo di pulizia dati e il monitoraggio costante della qualità (Reinschmidt & Francoise, 2000)[7].

Queste soluzioni rispondono inoltre alla crescente domanda di tempistica, tracciabilità ed accuratezza nei processi di analisi (Amadeo, 2018)[2], aspetto necessario in un'economia sempre più globale e competitiva. È importante, pertanto, approfondire l'analisi delle *pipeline* ETL.

Il motore OLAP fornisce funzionalità di analisi multidimensionale di dati storici. Funzionalità quali il *drill-down* e il *drill-up* consentono all'organizzazione di esplorare i dati a differenti livelli di dettaglio (Vaisman & Zimányi, 2022 [1]; Chen et al., 2012 [4]), similmente a quanto avviene in scenari decisionali reali.

Per semplificare le interrogazioni, la struttura logica dei *data warehouse* OLAP, basata su schemi a stella o a fiocco di neve, è pensata in modo tale da esporre la realtà aziendale in modo conciso e semplice, facilitando la comprensione anche da parte di personale non tecnico (Vaisman & Zimányi, 2022)[1].

Gli strumenti OLAP si sono evoluti per riuscire a fronteggiare la sempre più alta quantità di dati da processare. Questo è stato reso possibile introducendo tecnologie di *caching*, viste materializzate e *big data analytics* (Vaisman & Zimányi, 2022)[1].

Dal punto di vista tecnico e organizzativo, gli strumenti OLAP ricoprono un ruolo primario non solo nel *reporting*, ma anche nelle attività di collaborazione e di *decision-making* (Amadeo, 2018 [2]; Chen et al., 2012 [4]), aspetto che ne evidenzia la strategicità all'interno delle architetture di BI.

L'analisi scientometrica effettuata nei sistemi di BI ha evidenziato che l'integrazione OLAP-*data warehouse* incrementa la percentuale di dati analizzati effettivamente usati (Reinschmidt & Francoise, 2000 [7]; Chirkova & Yang, 2011 [8]). Oltre il 93 % dei dati è tutt'oggi immagazzinato in silos, e non viene usato (Reinschmidt & Francoise, 2000)[7], soprattutto per problemi di progettazione.

In futuro l'architettura di BI potrà evolversi aggiungendo strumenti e funzionalità. Uno scenario prevede l'adozione di algoritmi di *machine learning* per l'analisi dei dati, e/o di moduli avanzati di *data mining*, per migliorare la sostenibilità e l'efficienza dell'infrastruttura (Gurcan et al., 2023)[5].

La soluzione finale prevede la costruzione di un'architettura modulare e scalabile, con lo scopo di ridurre le distanze tra il mondo fisico e quello logico. L'architettura deve essere anche in grado di bilanciare *performance* ed efficienza e di gestire un carico di lavoro mutevole (Gurcan et al., 2023)[6].

2.2 Differenze tra sistemi OLTP e OLAP

I sistemi OLTP e OLAP si distinguono nettamente per finalità, requisiti e progettazione. I sistemi OLTP sono pensati per effettuare operazioni veloci ed efficienti di gestione, ad esempio, delle transazioni aziendali. L'obiettivo principale di questi sistemi è quello di effettuare operazioni ripetitive a bassa latenza e in grande numero (Vaisman & Zimányi, 2022 [1]; Inmon, 2002 [3]). I sistemi OLAP sono pensati per effettuare analisi complesse di grandi quantità di dati, ad esempio, con operazioni di aggregazione e di navigazione multidimensionale. Una *query* di OLAP può richiedere diversi minuti od ore, mentre in OLTP un'operazione complessa in genere non dura più di qualche secondo (Inmon, 2002)[3].

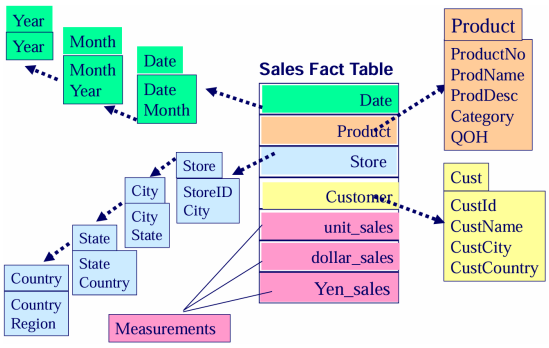


Figura 2.2: Esempio di schema *snowflake*. (Renso & Gozzi, 1990 p. 37)[9]

Nei sistemi OLTP è solitamente presente la normalizzazione degli schemi relazionali dei *database*, per prevenire incoerenza nei dati e garantire integrità referenziale (Vaisman & Zimányi, 2022 [1]; Renso & Gozzi, 1990 [9]). Nei sistemi OLAP, invece, è più diffusa la denormalizzazione, per velocizzare l'elaborazione delle interrogazioni (con l'utilizzo, ad esempio, di schemi a stella Figura 2.3 o a fiocco di neve Figura 2.2). Nei *database* utilizzati dai sistemi OLAP sono infatti contenute informazioni utili ad effettuare operazioni di navigazione e aggregazione multidimensionale. Il *drill-down* (approfondimento dei livelli di dettaglio) e il *roll-up* (generalizzazione degli aggregati) sono operazioni particolarmente veloci e facili in presenza di tabelle dimensionali denormalizzate (Vaisman & Zimányi, 2022 [1]; Renso & Gozzi, 1990 [9]).

In un sistema OLTP le operazioni principali sono di scrittura, aggiornamento e cancellazione (*write-intensive*) di una quantità di dati con orizzonte temporale generalmente inferiore a 60-90 giorni. Le *query* più complesse prevedono aggregazione

sui dati di al massimo pochi giorni. In OLAP le operazioni sono principalmente di lettura (*read-intensive*), su un orizzonte temporale spesso pluri-annuale. I sistemi OLTP sono pensati per essere in tempo reale, cioè reagiscono in modo immediato alle modifiche dei dati. La tipologia di *database* OLTP è pensata per effettuare inserimenti e aggiornamenti in modo efficiente. Le operazioni OLAP, invece, vengono effettuate anche in *batch* (senza reagire immediatamente alle modifiche dei dati) (Inmon, 2002)[3]. L'architettura dei *database* utilizzati in OLAP è studiata per effettuare velocemente operazioni di lettura (Inmon, 2002 [3]; Pontieri, 1989 [10]). Come ad esempio, è possibile considerare i sistemi utilizzati nei supermercati per la gestione delle vendite: i sistemi OLTP si occupano delle vendite immediate, mentre i sistemi OLAP sono utilizzati per analizzare le vendite su un orizzonte di alcuni anni (Pontieri, 1989)[10].

L'introduzione delle viste materializzate permette di effettuare interrogazioni complesse in tempi accettabili. Le viste materializzate precalcolano aggregazioni e operazioni di *join*, riducendo i tempi di calcolo in fase di esecuzione. Questa tipologia di viste è essenziale per effettuare analisi OLAP di volumi di dati sempre maggiori (Amadeo, 2018 [2]; Renso & Gozzi, 1990 [9]). Elaborare un'aggregazione complessa in tempo reale comprometterebbe drasticamente le prestazioni di un sistema OLTP. La complessità dei dati di sistemi OLAP richiede, perciò, l'utilizzo di un *data warehouse*, che si presta bene all'adozione delle viste materializzate.

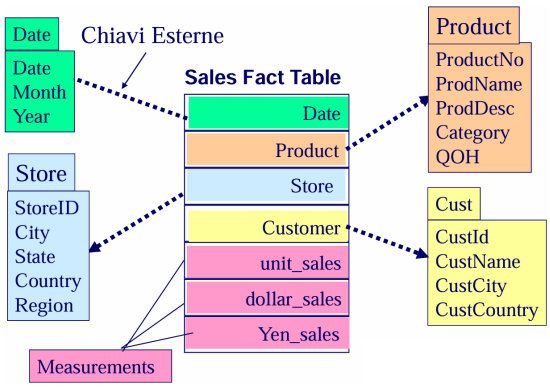


Figura 2.3: Esempio di schema *star*. (Renso & Gozzi, 1990 p. 37)[9]

Utilizzare l'uno o l'altro sistema o combinare i due dipende dal particolare carico di lavoro. In questo modo entrambi i sistemi lavorano con prestazioni accettabili, ottimizzando le operazioni che meglio si adattano alle proprie architetture (Vaisman & Zimányi, 2022 [1]; Inmon, 2002 [3]). Tuttavia, le aziende tendono sempre più a utilizzare ampiamente le informazioni che ricavano dai dati e a basare le loro strategie sul *knowledge management* (cioè sulla conoscenza ricavata dai dati), quindi si tende sempre di più ad adottare un'architettura combinata. Questo tipo di infrastrutture, che combina i due sistemi, offre molti vantaggi. Il principale di questi è la possibilità di effettuare sia operazioni *real-time* che operazioni complesse, sfruttando i punti di forza dei due sistemi. Combinando il *data warehouse* con la reattività in tempo reale dei sistemi OLTP si ottiene una soluzione estremamente flessibile (Amadeo, 2018 [2]; Renso & Gozzi, 1990 [9]; Pontieri, 1989 [10]).

Tabella 2.2: Confronto tra sistemi OLTP e OLAP. (Renso & Gozzi, 1990 p. 15)[9]

Caratteristica	**OLTP**	**OLAP**
Funzione	gestione giornaliera	supporto alle decisioni
Progettazione	orientata alle applicazioni	orientata al soggetto
Frequenza	giornaliera	sporadica
Dati	recenti, dettagliati	storici, riassuntivi, multidimensionali
Sorgente	singola DB	DB multiple
Uso	ripetitivo	ad hoc
Accesso	read/write	read
Flessibilità accesso	uso di programmi precompilati	generatori di query
# record acceduti	decine	migliaia
Tipo utenti	operatori	manager
# utenti	migliaia	centinaia
Tipo DB	singola	multiple, eterogenee
Performance	alta	bassa
Dimensione DB	100 MB - GB	100 GB - TB

Capitolo 3

Evoluzione e Qualità dei Processi ETL

L'efficienza e l'affidabilità dei processi ETL sono elementi centrali per la qualità e la tempestività dei dati nel contesto delle architetture di BI. Analizzando metriche strutturali e strategie di ottimizzazione, si introducono delle metodologie per migliorare le *performance*, ridurre i costi e garantire accuratezza elevata. Questo *focus* fa parte del panorama delle tecnologie e metodologie che caratterizzano l'infrastruttura centrale della gestione dei dati.

3.1 Metriche strutturali e impatto sull'efficienza

L'analisi delle metriche strutturali relative ai processi ETL rappresenta un punto chiave di comprensione dell'impatto della complessità architetturale sul *throughput*. Il conteggio di nodi di *join*, *lookup*, *script*, *row-set* e *input-output* del *Data Process Graph* (DPG) fornisce una metrica complessiva del livello di complessità del processo. Un monitoraggio continuo di queste metriche, secondo quanto suggerito da El Akkaoui et al. (2019)[11], consente di prevenire potenziali colli di bottiglia del processo ETL, migliorando di conseguenza il *throughput* complessivo. Tutte le decisioni di progettazione devono quindi avere l'obbiettivo di ottimizzare le aree con maggior complessità architetturale ed impatto sul *throughput*, ad esempio i nodi di *script* e *join*.

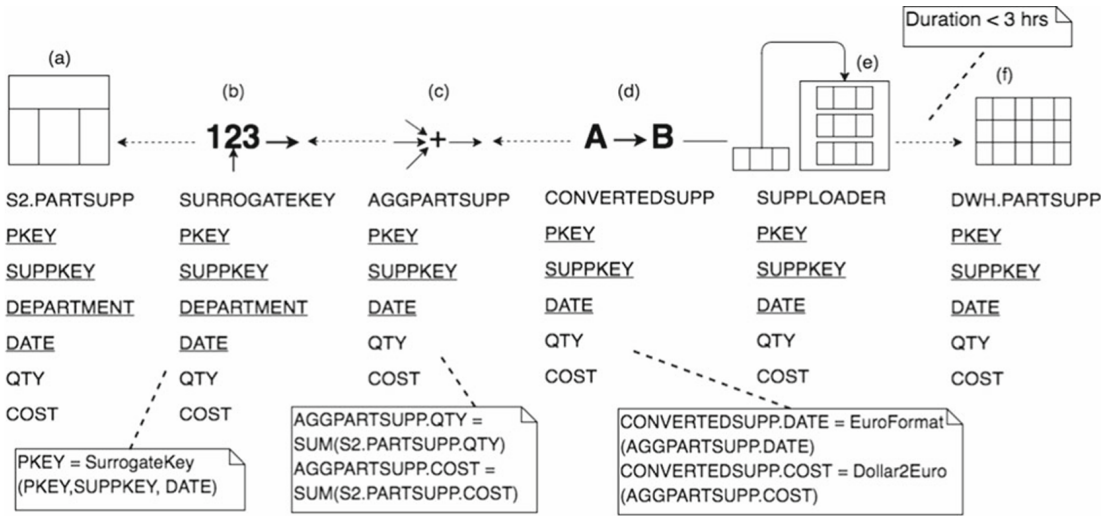
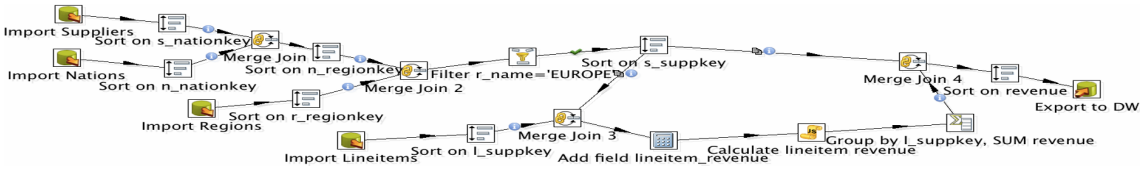


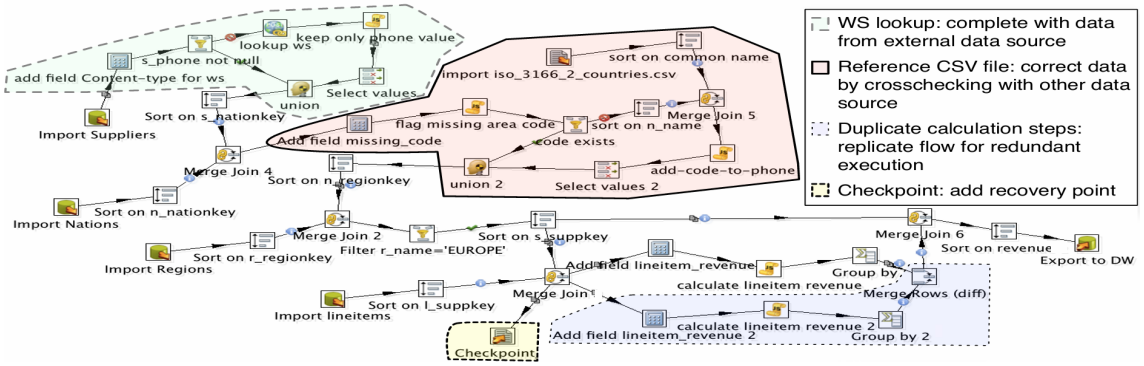
Figura 3.1: Esempio di progettazione UML di un *workflow* ETL. (Ali & Wrembel 2017 p. 780)[12]

Un altro fattore importante riguarda la comparazione dell'influenza che hanno i nodi di *branching*, rispetto ai nodi di *script* e *join*, sul *throughput* di un processo ETL. Da quanto suggerito da El Akkaoui et al. (2019)[11], i nodi di *branching* hanno un impatto non significativo sul *throughput*, al contrario di quanto accade se vengono aggiunti più nodi di tipo *join* e *script*. Questi ultimi impattano negativamente sul *throughput* di un processo ETL, mentre l'aggiunta di rami complessi ha un impatto marginale sulle *performance*. Questo suggerisce di creare *branching* più complessi al fine di ottenere processi ETL più dinamici e manutenibili senza preoccuparsi di ridurre il *throughput* del processo. Bisogna però sempre pensare a come bilanciare le *performance* e la dinamicità di un processo ETL, riducendo al minimo la presenza dei nodi di tipo *script* e *join*.

L'accuratezza dei dati di un processo ETL dipende fortemente dalla modellazione concettuale del *workflow*. Dalla ricerca condotta da Theodorou et al. (2016)[13], il modello CM_B (Figura 3.2b), che possiede un livello di complessità concettuale maggiore rispetto al modello CM_A (Figura 3.2a), consente di generare dati più accurati. Questo è dovuto all'aggiunta di regole e controlli aggiuntivi al modello concettuale che garantiscono un'elevata integrità e accuratezza dei dati di *output*. Tuttavia, maggiore è il numero di regole implementate, più elevati sono i tempi di elaborazione. In contesti con alta criticità, in cui l'accuratezza dei dati è fondamentale, è preferibile utilizzare modelli concettuali complessi (Theodorou et al. 2014)[14], mentre, se non vi sono vincoli sull'accuratezza dei dati, è preferibile utilizzare modelli concettuali più semplici per ridurre il tempo di esecuzione del processo ETL.



(a) Rappresentazione logica del processo ETL iniziale che corrisponde a CM_A.



(b) Rappresentazione logica del processo ETL equivalente con i passaggi aggiuntivi che corrisponde a CM_B.

Figura 3.2: Modelli logici di processi ETL alternativi. (Theodorou et al., 2016 p. 10)[13]

L'usabilità, la manutenibilità e le *performance* sono tutti obiettivi di qualità essenziali nella progettazione dei processi ETL e spesso necessitano di bilanciamenti tra loro. Secondo Theodorou et al. (2014)[14], un aumento dell'usabilità del *workflow* facilita le operazioni di manutenzione e contribuisce alla diminuzione del rischio di errori dovuti alla manipolazione dello stesso. L'aggiunta di un maggior numero di operazioni e vincoli di controllo a tale processo, in molti casi, impatta negativamente sulle *performance*, in particolare sul *throughput*, specialmente se si eseguono processi ETL per molti elementi di dati (Theodorou et al. 2016)[13]. Una sfida fondamentale per la progettazione di *workflow* ETL risiede nel trovare un equilibrio adeguato tra i compromessi dovuti ai vincoli progettuali e le priorità dell'organizzazione.

L'utilizzo di rappresentazioni dei *workflow* ETL tramite grafi o *Unified Modeling Language* (UML) non consente l'automatizzazione nella progettazione per la gestione di dati semi-strutturati o non strutturati. Come sottolineato da Ali & Wrembel (2017)[12], la progettazione di *workflow* ETL per dati non strutturati richiede esperienza di progettazione, in quanto non vi sono algoritmi di progettazione con un risultato ottimale certo, che massimizzino le metriche strutturali. Lo studio pone quindi una sfida su come poter generare *workflow* ETL in modo automatico e ottimizzare le proprie strutture tenendo presente le metriche di *performance* più importanti di questo.

Polyvyanyy et al. (2017)[15], in un'indagine scientometrica, identifica in parte una

lacuna nella ricerca accademica sull'importanza della correlazione tra le metriche strutturali e le metriche valutative sui processi ETL e del valore analitico derivato.

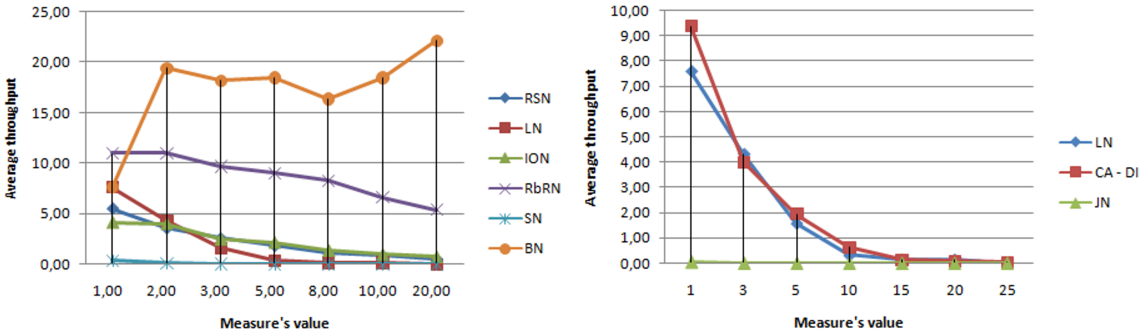
3.2 Ottimizzazione dei processi di estrazione e trasformazione

Ottimizzare i processi di estrazione e trasformazione rappresenta una delle fasi più critiche al fine di migliorare il *throughput* e di garantire la qualità dei dati in un contesto di BI. La valutazione delle *performance* della struttura di una *pipeline* ETL aiuta ad identificare i colli di bottiglia durante i processi di estrazione, trasformazione e caricamento. I nodi che influiscono negativamente sul *throughput* sono: di *join*, di *script*, di *lookup*, i *row-set* e gli *input-output*. La presenza di *join* e di *script* rallentano i processi, mentre il miglioramento generale si può raggiungere implementando le tecniche di *tuning* attraverso l'utilizzo di *lookup* (El Akkaoui et al., 2019)[11].

L'implementazione di metriche strutturali derivate dal DPG fornisce un approccio più oggettivo per la valutazione dell'efficacia di una *pipeline* ETL rispetto ad un'altra, offrendo un metodo comparativo più sensibile alla differenza tra diversi *workflow*. La valutazione delle *performance* attraverso le metriche strutturali si rivela un valido approccio in contesti in cui la quantità di dati è molto alta e non si hanno a disposizione algoritmi di analisi della struttura della *pipeline* ETL (El Akkaoui et al., 2019)[11].

In realtà, un'estrema attenzione all'ottimizzazione delle *performance* può diminuire la manutenibilità del codice e la qualità dei dati. Risulta perciò importante un approccio olistico alla valutazione delle *performance* per ottimizzare anche manutenibilità e accuratezza. In altre parole, per ottimizzare la *pipeline* ETL occorre trovare un bilanciamento tra le tre dimensioni (Theodorou et al., 2016)[13].

La sostituzione dei nodi *join* multipli con nodi di *script* o di *lookup* rappresenta una delle tecniche più efficaci, contribuendo alla riduzione dei tempi di esecuzione e migliorando il *throughput* di una *pipeline* ETL. I nodi di *lookup* risultano più efficienti dei nodi di *script* perché le operazioni sui *lookup* necessitano di minore potenza computazionale e il tempo per accedere ai dati intermedi è minore (El Akkaoui et al., 2019)[11].



(a) Throughput per numero di nodi. (b) Confronto nodi: lookup, join e script.

Figura 3.3: *Row-Set Nodes* (RSN); *Lookup Nodes* (LN); *Input-Output Nodes* (ION); *Row-by-Row Nodes* (RbRN); *Script Nodes* (SN); *Branching Nodes* (BN); *Join Nodes* (JN); Misura *Cohesion Action* calcolata su un nodo dello *script* di *Data Input* (CA-DI). (El Akkaoui et al., 2019 p. 9)[11]

In molti studi sono state proposte tecniche di validazione al fine di migliorare la qualità dei dati delle *pipeline* ETL. L'implementazione di un meccanismo di controlli di accuratezza e correzione nei modelli concettuali delle *pipeline* ETL migliora significativamente la qualità dei dati; l'accuratezza dei dati si avvicina alla totalità, ma a questo vantaggio segue un costo di tempo di esecuzione di circa il 45 % più alto rispetto alle *pipeline* che non implementano controlli di qualità (Theodorou et al., 2016)[13].

Tali meccanismi aumentano l'accuratezza dei dati ma, allo stesso tempo, aumentano i tempi di esecuzione e le risorse computazionali. È necessario, perciò, la realizzazione di strumenti per il monitoraggio e per la simulazione dell'esecuzione della *pipeline* ETL al fine di scegliere in modo più adatto tra un *design* ed un altro (Theodorou et al., 2016)[13].

Ci sono modelli concettuali di *pipeline* ETL molto dettagliati ma questi sono adatti in contesti dove la latenza non è il vincolo più importante. In contesti dove la frequenza di aggiornamento del *data warehouse* è più importante dei controlli di qualità dei dati, viene utilizzato un modello meno preciso ed il sistema di progettazione delle *pipeline* ETL è semplice e veloce (Theodorou et al., 2016)[13].

La scelta del modello di progettazione di un *workflow* ETL influenza l'utilizzo di risorse specialistiche (tempo dello sviluppatore, tempo di validazione, ecc.) e la possibilità di elaborare dati semi-strutturati e non strutturati. Il metodo UML, ad esempio, presenta una scarsa manutenibilità a fronte di un elevato numero di entità e relazioni da gestire. I modelli basati sui grafi non sono automatizzabili e sono difficili da interpretare in domini complessi. È necessario, perciò, l'automazione della generazione e dell'ottimizzazione dei *workflow* tramite l'utilizzo di tecniche e di strumenti che supportino l'implementazione di metriche strutturali. Inoltre, i

metodi di auto-ottimizzazione e *feedback* riducono le attività di validazione ed ottimizzano il flusso, minimizzando il ruolo dei professionisti (Ali & Wrembel, 2017)[12].

Dalle analisi della letteratura esaminata emerge anche che non ci sono metodi efficienti di ottimizzazione dell'esecuzione di *workflow* ETL, ma non si conosce se questo rappresenti una direzione di ricerca già esplorata. Inoltre, si evidenzia anche una scarsa integrazione tra metriche di qualità e strumenti di ottimizzazione del *workflow* e ciò rappresenta un'opportunità di studio nell'area *data-driven*, in combinazione con il *machine learning* (Ali & Wrembel, 2017)[12].

La ricerca di sistemi di *self-tuning* di *workflow* ETL che si adattano a cambiamenti nelle proprietà dei dati rappresenta una prospettiva molto innovativa ed è un'evoluzione del *framework* ETL, che ha finora supportato solamente un livello base di controllo del *workflow* (Ali & Wrembel, 2017)[12].

In un contesto di *Big Data*, per l'ottimizzazione delle *pipeline* ETL si rende necessario utilizzare algoritmi in grado di affrontare la sfida della scalabilità. Le architetture a larga scala utilizzano, perciò, tecniche che permettono il partizionamento parallelo della trasformazione, le *pipeline* distribuiscono i *task* attraverso molti esecutori in un *cluster* e gestiscono l'acquisizione di dati simultanei da molteplici fonti. Inoltre, le nuove *pipeline* sono implementate con architetture modulari e *tool software* che supportano l'integrazione delle nuove fonti di dati e la gestione in remoto delle *pipeline*. In ultimo, la scelta di monitorare e ottimizzare una *pipeline* ETL in *runtime* con l'uso di motori di coordinamento può automatizzare una larga gamma di attività e migliorare le *performance* (Goud, 2024)[16].

Un'ottimizzazione più spinta è data da un miglioramento dei tempi di esecuzione delle *query* analitiche attraverso l'uso di viste materializzate, tecniche di selezione *multi-query* e la combinazione di approcci per minimizzare i tempi di aggiornamento del *data warehouse*. La combinazione delle *pipeline* ETL con le *query* analitiche contribuisce ad una sensibile diminuzione dei tempi di latenza del *data warehouse*. La dipendenza delle *query* di BI dalle *pipeline* ETL richiede una comunicazione continua tra i componenti ETL e i progettisti dei *data warehouse* per un coordinamento ottimale delle *performance* (Boukorca et al., 2015; [17] Alapaty & Rao, 2024 [18]).

Da quanto esposto da Boukorca et al. (2015)[17], si evince che le *pipeline* ETL influenzano significativamente l'efficacia di un *data warehouse*, sia in termini di costi, che di *performance*. In altre parole, più le *pipeline* ETL sono efficaci, e quindi i costi totali del *data warehouse* sono bassi, più è probabile ottenere *performance* migliori, specialmente l'implementazione e la manutenzione delle viste materializzate (Boukorca et al., 2015)[17]. L'ottimizzazione di estrazione e trasformazione è fondamentale per aumentare la scalabilità delle *pipeline* ETL e garantire dati di alta qualità in tempi ridotti.

Capitolo 4

Ottimizzazione Fisica del Data Warehouse

L'efficienza e le prestazioni dei sistemi di BI dipendono notevolmente dall'ottimizzazione delle architetture fisiche del *Data Warehouse*. Attraverso tecniche di indicizzazione, partizionamento, compressione e *caching* si esamineranno le scelte relative alla configurazione ed alla gestione delle risorse *hardware* per migliorare la scalabilità, la velocità di accesso e la sostenibilità dell'architettura. Nel contesto dell'elaborato, questo approccio è ritenuto importante come tecnica di supporto alle tecniche di progettazione logica e metodologie di ottimizzazione viste precedentemente.

4.1 Tecniche di indicizzazione e partizionamento

Le tecniche di indicizzazione e partizionamento sono tecniche fondamentali per migliorare le prestazioni e l'accessibilità ai dati in un *data warehouse* e in un sistema di BI caratterizzato da elevate quantità di dati. Per quanto riguarda le tecniche di indicizzazione, rispetto agli indici *RowID*, gli indici *bitmap* permettono di risparmiare in termini di spazio e di tempo per il calcolo. Infatti, con questa tecnica la rappresentazione dei dati è più compatta e si rivela molto utile in ambienti OLAP in cui la cardinalità degli attributi, come i codici prodotto o le categorie, è bassa. Ciò consente di gestire tabelle molto grandi, facilitando le operazioni di ricerca aggregata mediante operazioni *bitwise*. Grazie a questa tecnica è possibile velocizzare le operazioni e ridurre il carico computazionale per effettuare le *query*, offrendo una migliore esperienza di *reporting* multidimensionale (Golfarelli, 2000)[19]. Ovviamente, l'utilizzo di questa tecnica deve essere attentamente valutato, in quanto, come indicato da Goyal et al. (1999)[20] e Golfarelli (2000)[19], se sono frequenti le operazioni di aggiornamento, si riducono i vantaggi che ne derivano dall'uso degli indici *bitmap*.

(a) Tabella delle dimensioni di *Product*.

ProductKey	**ProductName**	**QuantityPerUnit**	**UnitPrice**	**Discontinued**	**CategoryKey**
p1	prod1	25	60	No	c1
p2	prod2	45	60	Yes	c1
p3	prod3	50	75	No	c2
p4	prod4	50	100	Yes	c2
p5	prod5	50	120	No	c3
p6	prod6	70	110	Yes	c4

(b) Indice *bitmap* per l'attributo *QuantityPerUnit*;

	25	45	50	70
p1	1	0	0	0
p2	0	1	0	0
p3	0	0	1	0
p4	0	0	1	0
p5	0	0	1	0
p6	0	0	0	1

(c) Indice *bitmap* per l'attributo *UnitPrice*.

	60	75	100	110	120
p1	1	0	0	0	0
p2	1	0	0	0	0
p3	0	1	0	0	0
p4	0	0	1	0	0
p5	0	0	0	0	1
p6	0	0	0	1	0

Figura 4.1: Esempio di indici *bitmap* per una tabella delle dimensioni di *Product*. (Vaisman e Zimányi, 2022 p. 269)[1]

Il partizionamento dei dati è un'altra tecnica per aumentare le prestazioni del sistema. Tale tecnica permette di partizionare le tabelle di grandi dimensioni in partizioni più piccole per velocizzare le *query* analitiche effettuate su periodi temporali o su segmenti specifici. Alcune possibili tecniche di partizionamento sono quella orizzontale, che partiziona i dati per periodi temporali, e quella verticale, che partiziona i dati tramite gli attributi delle tabelle. Un esempio di tecnica orizzontale è partizionare la tabella dei fatti per trimestre o per anno, per effettuare le analisi storiche più velocemente, riducendo i tempi di scansione dei dati. Con questa tecnica, come indicato da Vaisman e Zimányi (2022)[1], si velocizzano le prestazioni e si favoriscono le operazioni di manutenzione, come archiviare o eliminare le partizioni inutilizzate. Anche per questa tecnica è fondamentale effettuare un'analisi preliminare sul sistema, per stabilire l'efficacia che deriva dal partizionamento dei dati, analizzando il sistema, le modalità di accesso ai dati e le esigenze.

La combinazione delle tecniche di partizionamento e indicizzazione può essere un ulteriore approccio per massimizzare i vantaggi di entrambe. Grazie all'utilizzo congiunto di indici *bitmap* su tabelle partizionate è possibile ottenere un accesso simultaneo ai dati storici contenuti nelle partizioni. Ciò si rivela molto importante perché, in molti sistemi di *Business Intelligence*, vengono eseguite *query* su periodi di tempo molto lunghi e tale approccio non va ad intaccare la disponibilità dei dati,

così da poter velocizzare le *query*, minimizzando i tempi di risposta. Come indicato da Vaisman e Zimányi (2022)[1], occorre anche tenere sotto controllo l'efficacia di queste tecniche con un adeguato monitoraggio, in quanto la *performance* di un *data warehouse* non deve essere compromessa dalle impostazioni precedentemente configurate.

```
-- Partition function: assegna righe a intervalli per anno
-- (<=184 --> 2016, <=549 --> 2017, <=730 --> 2018, >730 --> successivi)
CREATE PARTITION FUNCTION [PartByYear] (INT)
  AS RANGE LEFT FOR VALUES (184, 549, 730);

-- Partition scheme: usa PartByYear e memorizza tutte le partizioni in
    PRIMARY
CREATE PARTITION SCHEME [SalesPartScheme]
  AS PARTITION [PartByYear] ALL TO ([PRIMARY]);

-- Crea la tabella Sales partizionata su OrderDateKey
CREATE TABLE Sales (
    CustomerKey   INT,
    EmployeeKey   INT,
    OrderDateKey  INT
    -- ecc.
) ON SalesPartScheme(OrderDateKey);

-- Vista aggregata con schemabinding per permettere indicizzazione
CREATE VIEW SalesByDateProdEmp
WITH SCHEMABINDING
AS
SELECT
    OrderDateKey,
    ProductKey,
    EmployeeKey,
    COUNT(*)        AS Cnt,
    SUM(SalesAmount) AS SalesAmount
FROM Sales
GROUP BY OrderDateKey, ProductKey, EmployeeKey;

-- Indice clustered UNIQUE sulla vista, allineato allo stesso schema di
    partizione
CREATE UNIQUE CLUSTERED INDEX UCI_SalesByDateProdEmp
  ON SalesByDateProdEmp (OrderDateKey, ProductKey, EmployeeKey)
  ON SalesPartScheme(OrderDateKey);
```

Listing 4.1: Esempio di creazione di tabella partizionata e vista indicizzata allineata. (Vaisman e Zimányi, 2022 pp. 282–283)[1]

Le tecniche di ottimizzazione per i sistemi di BI devono tenere presente che la priorità, rispetto ai sistemi OLTP, deve essere rappresentata dalla riduzione dei tempi di risposta e non dal *throughput*. Ciò comporta l'implementazione di alcune tecniche specifiche di progettazione fisica, quali gli indici *bitmap* e le partizioni sugli attributi maggiormente utilizzati nelle *query*, perché l'obiettivo dei sistemi *Business Intelligence* è effettuare interrogazioni su enormi quantità di dati di anni passati. Quindi, per raggiungere gli obiettivi prefissati, occorre monitorare le prestazioni ed eseguire alcuni *benchmark* per valutare l'efficacia delle tecniche di progettazione fisica introdotte. Come riportato da Golfarelli (2000)[19] e da Inmon (2002)[3], le

tecniche di progettazione fisica più adatte dipendono da come si accede ai dati e in quale modo vengono aggiornati.

Nelle tecniche di progettazione, solitamente, viene spesso trascurato l'effetto che la granularità delle partizioni può avere sulla *performance* del sistema. Partizioni con una granularità molto fine possono comportare un eccessivo *overhead* amministrativo e la frammentazione delle risorse, mentre partizioni troppo ampie rischiano di compromettere la velocità di accesso ai dati. Occorre, quindi, trovare un compromesso in termini di granularità del partizionamento in base ai bisogni del *business*, in modo da ottimizzare la *performance* e la velocità di risposta del sistema, come indicato da Vaisman e Zimányi (2022)[1].

Le prospettive future nell'ingegneria computazionale mostrano l'importanza di implementare strategie avanzate di accesso ai dati, dato il continuo incremento in termini di quantità e complessità dei dati nei sistemi di *Business Intelligence*. Negli ultimi anni, come indicato da Khanna et al. (2021)[21], è cresciuta sempre più l'importanza dell'integrazione tra tecniche di indicizzazione tradizionali e gli algoritmi di *machine learning*. La particolarità di questi ultimi algoritmi è la dinamicità per la gestione degli indici e delle partizioni e la capacità di prevedere il carico di lavoro, aumentando in questo modo la scalabilità e la velocità del sistema.

In aggiunta a questo, il rapidissimo sviluppo delle architetture *Big Data* impone continui aggiornamenti delle tecniche di ottimizzazione, monitoraggio delle prestazioni e dei processi per effettuare un automatico ribilanciamento delle partizioni e degli indici. Da Khanna et al. (2021)[21] viene evidenziato come sia fondamentale adottare una progettazione fisica integrata, in cui le tecniche di ottimizzazione statica (indici *bitmap* e partizionamento) vengono affiancate da tecniche *data-driven*, per migliorare le prestazioni, la scalabilità e la complessità delle *query*.

Un ultimo fattore da non sottovalutare è quello di offrire alle figure professionali esperte una solida base tecnica su cui poter contare. Grazie all'implementazione di tecniche di *machine learning*, ai sistemi di monitoraggio automatizzati, ai nuovi strumenti di progettazione e alla combinazione delle figure professionali esistenti sarà possibile ottenere una migliore esperienza degli utenti, un supporto più affidabile agli operatori, oltre che ai decisori, aumentando di conseguenza la continuità del servizio offerto.

4.2 Strategie di compressione e caching

Le strategie di compressione nei *data warehouse* consentono una sensibile riduzione dello spazio necessario per l'archiviazione, particolarmente critica nei *data warehouse* OLAP. L'utilizzo di algoritmi come LZW permette una riduzione del 30 % a 35 % rispetto alle dimensioni originarie, raggiungendo un rapporto di compressione del

65 % a 70 %, secondo Goyal et al. (1999)[20]. Questo consente di risparmiare sull'occupazione di memoria nei contesti di BI, solitamente con elevati volumi di dati e con requisiti di conservazione storica. Tuttavia, è necessario tenere in considerazione l'impatto della compressione/decompressione sulle prestazioni, soprattutto in casi in cui siano necessari processi ripetuti di decompressione.

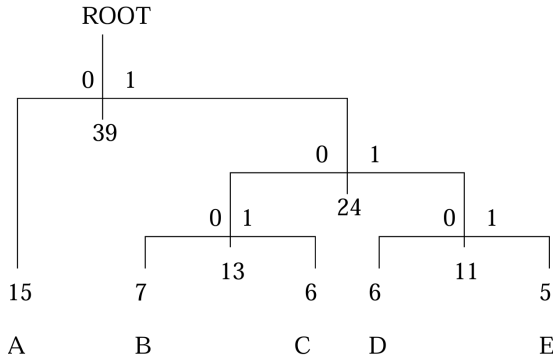


Figura 4.2: *Huffman Coding.* (Goyal et al., 1999 p. 11-7)[20]

In realtà, un ulteriore vantaggio consiste nell'incremento delle prestazioni per le *query* che ne beneficiano. Ovvero si ottiene una riduzione delle operazioni di *Input-Output* (I/O) al disco poiché vi è un minore trasferimento di dati dal medesimo. Questa ottimizzazione è particolarmente utile nelle operazioni analitiche tipiche dei *data warehouse*, dove sono richieste *query* con aggregazioni e analisi multidimensionali (Goyal et al., 1999)[20]. Ovviamente, questo incremento di prestazioni è legato al tipo di algoritmo scelto.

Ad esempio, l'*Huffman Coding* permette di bilanciare al meglio velocità e capacità di compressione/decompressione, mentre algoritmi più complessi, come l'*Arithmetic Coding*, nonostante migliori prestazioni in termini di percentuale di compressione, si rivelano inadeguati se è richiesta frequente modifica dei dati, tempi di risposta brevi ed elevata affidabilità (Goyal et al., 1999)[20].

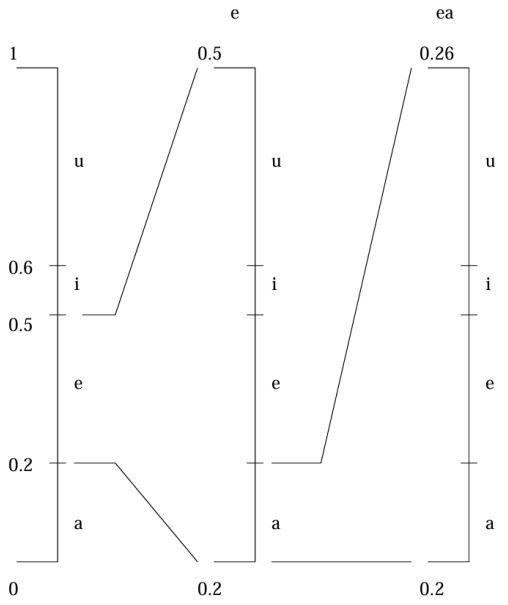


Figura 4.3: *Arithmetic Coding.* (Goyal et al., 1999 p. 11-8)[20]

La compressione ha un impatto significativo sulle strategie di *backup/recovery* per i *data warehouse*. Riducendo la quantità di dati da gestire, si ha un consistente risparmio di tempo nelle procedure di *backup*, oltre a migliorare lo *storage* secondario per ridurre i costi di archiviazione (Goyal et al., 1999)[20]. L'unico compromesso è rappresentato dalla lentezza nell'accesso ai dati compressi durante il *recovery*, se si considera un ripristino del sistema in un tempo relativamente breve.

Il *caching* permette di rispondere più velocemente alle *query* nel contesto dei sistemi di BI. Questa tecnica consiste nella conservazione in memoria *cache* dei dati più frequentemente richiesti o del risultato di viste materializzate (Cerquitelli & Garza, 2023)[22]. L'obiettivo primario è ridurre la latenza delle *query* ripetitive di *reporting* o delle operazioni di analisi interattive, mediante l'identificazione dei dati utili per la *cache*.

Una gestione dinamica del *caching* consente di ridurre drasticamente le operazioni di I/O al disco e di fornire dei risultati aggiornati (Cerquitelli & Garza, 2023)[22]. Questo tipo di tecnica implementata dinamicamente è utilizzata anche in sistemi di grandi dimensioni al fine di evitare le limitazioni apportate dall'I/O.

La progettazione fisica del *data warehouse* ha un impatto significativo sulle strategie di compressione e *caching*: condiziona la manutenzione delle viste materializzate e permette di selezionare al meglio le configurazioni per gli indici. L'utilizzo di algoritmi come A* (Labio et al., 1996)[23] permette di analizzare contemporaneamente il costo di memorizzazione, il costo di accesso e l'impatto degli aggiornamenti al *data warehouse*, scartando le configurazioni subottimali e migliorando l'efficienza.

```
1  Input: M, ≺
2  Output: Optimal M'
3
4  Let state set S = {s}, where s is a partial state having
5      M_C(s) = M'(s) = ∅ and M_U(s) = M (base relations and V are
       materialized)
6  Loop
7      Select the partial state s ∈ S with the minimum value of Ĉ
8      If M_C(s) ≡ M, return M'(s)
9      Let S = S - {s}
10     For each view or index m ∈ M_U(s) such that for all m' ≺ m: m'
       ∈ M_C(s)
11         Construct partial state s' such that
12             M_C(s') = M_C(s) ∪ {m}
13             M_U(s') = M_U(s) - {m}
14             M'(s') = M_C(s) ∪ {m}
15         Construct partial state s'' such that
16             M_C(s'') = M_C(s) ∪ {m}
17             M_U(s'') = M_U(s) - {m}
18             M'(s'') = M_C(s)
19         Let S = S ∪ {s'} ∪ {s''}
20     Endfor
21 Endloop
```

Listing 4.2: Algoritmo A* per selezionare viste e indici (Labio et al., 1996 p. 10)[23]

Tramite algoritmi e sistemi di *machine learning* si può determinare in modo dinamico il compromesso tra prestazioni e costo di mantenimento dei sistemi di *data warehouse* basati su viste materializzate (Bhaja et al., 2023)[24]. Questo permette l'adozione delle viste materializzate anche in *data warehouse* complessi con carichi di lavoro variabili.

L'applicazione contemporanea di tecniche di compressione e *caching* sta diventando la più comune in caso di elevati carichi di lavoro e per sistemi di *Business Intelligence* che richiedono scalabilità. La flessibilità, intelligenza e scalabilità di sistemi implementati tramite queste tecniche hanno permesso un ulteriore evoluzione dei moderni *data warehouse*, consentendo di rispondere a carichi di lavoro con variabili caratteristiche (Bhaja et al., 2023)[24].

Le strategie di compressione e *caching* devono essere parte di un *framework* di ottimizzazione fisica del *data warehouse*. Questo permette di effettuare in tempi ragionevoli le *query* su insiemi di dati di enorme grandezza, pur controllando i tempi di manutenzione delle risorse. Nella letteratura si evidenzia come le prestazioni dei *data warehouse* beneficino da una combinazione di dati compressi e *caching* (Inmon, 2002)[3].

In conclusione, l'applicazione combinata di strategie di compressione e *caching*, con un approccio dinamico e basato su sistemi di progettazione avanzata, è la chiave del miglioramento delle *performance*, mantenendo un'elevata sostenibilità operativa per i *data warehouse* di nuova generazione.

$$C(\mathcal{M}') = \sum_{m \in (\mathcal{M}' \cup \mathcal{B} \cup \{V\})} maint\_cost(m, \mathcal{M}') \tag{4.1}$$

Dove, $\mathcal{M}'$ è l'insieme di viste e indici materializzati; $\mathcal{B}$ è l'insieme delle relazioni di base materializzate; V è la vista primaria; $maint\_cost(m, \mathcal{M}')$ è il costo di mantenimento di m dato $\mathcal{M}'$.

$$\mathcal{C} = g + h \tag{4.2}$$

Dove, $\mathcal{C}$ è il costo totale di manutenzione (cioè $C(\mathcal{M}')$); g è il costo di manutenzione già sostenuto da viste e indici selezionati finora; h è la stima inferiore del costo rimanente necessario a materializzare il resto di $\mathcal{M}$.

$$h = \min_{\mathcal{M}'_{\mathcal{U}} \subseteq \mathcal{M}_{\mathcal{U}}} \Big(\sum_{m \in \mathcal{M}'_{\mathcal{U}}} maint\_cost(m, \mathcal{M}' \cup \mathcal{M}'_{\mathcal{U}}) \Big) \tag{4.3}$$

Dove, $\mathcal{M}_{\mathcal{U}}$ è l'insieme di viste e indici ancora da considerare; $\mathcal{M}'_{\mathcal{U}}$ è un sottoinsieme di $\mathcal{M}_{\mathcal{U}}$ che si prova a materializzare; L'espressione valuta per ogni $\mathcal{M}'_{\mathcal{U}}$ la somma dei costi di manutenzione, restituendo il minimo tra tutte le combinazioni (ricerca esaustiva).

$$\hat{\mathcal{C}} = g + \hat{h} \tag{4.4}$$

Dove, $\hat{\mathcal{C}}$ è il limite inferiore del costo totale $\mathcal{C}$, usando la stima euristica $\hat{h}$ al posto di h; $\hat{h}$ è la stima inferiore (più veloce da calcolare) del costo rimanente.

$$\hat{h} = \sum_{m \in \mathcal{M}'_{\mathcal{U}}} \Big(h\_maint\_cost(m, \mathcal{M}') - max\_benefit(m, \mathcal{M}') \Big) \tag{4.5}$$

Dove, $h\_maint\_cost(m, \mathcal{M}')$ è il costo "pessimistico" di mantenimento di m dato $\mathcal{M}'$; $max\_benefit(m, \mathcal{M}')$ è il risparmio massimo atteso grazie a m; L'espressione fornisce un limite inferiore al costo netto di aggiungere m.

Formule 4.4: Considerazioni per l'algoritmo A*. (Labio et al., 1996 pp. 8-10)[23]

Capitolo 5

Viste Materializzate

Le viste materializzate rappresentano uno degli strumenti che permettono di velocizzare e migliorare le prestazioni del sistema di *Business Intelligence*. Di seguito verrà analizzato come possono essere utilizzate per rendere le prestazioni del sistema migliori e più rapide e si metterà in relazione la loro utilizzabilità con il tema dell'ottimizzazione e della gestione dei dati trattato nel lavoro dell'elaborato.

5.1 Definizione e vantaggi implementativi

Le viste materializzate rappresentano uno degli strumenti più efficaci per incrementare le *performance* in un sistema di *Business Intelligence* grazie alla loro capacità di abbreviare i tempi di risposta alle *query*. Infatti, conservano i risultati precalcolati, e quindi è possibile soddisfare una *query* senza che il risultato venga ricalcolato, rendendo più rapido il tempo di risposta anche su dati di grandi dimensioni. In *Amazon Redshift*, per esempio, le *query* vengono fatte sulle viste materializzate senza interrogare le tabelle sottostanti, per un tempo di risposta interattivo (Stark et al., 2022)[25]. La materializzazione di una vista comporta dei costi in termini di spazio e di manutenzione.

```
1  -- Supponendo che le tabelle 'orders' e 'customer' siano state create e
       popolate
2  CREATE MATERIALIZED VIEW mv_total_orders
3      AUTO REFRESH YES  -- per avere un auto refresh
4  AS
5      SELECT c.cust_id,
6             c.first_name,
7             SUM(o.amount) AS total_amount
8      FROM orders o
9      JOIN customer c
10       ON c.cust_id = o.customer_id
11     GROUP BY c.cust_id,
12              c.first_name;
```

Listing 5.1: Esempio di creazione di una vista materializzata con *auto refresh* (Stark et al., 2022 p. 9)[25]

Le viste materializzate sono fondamentali per la gestione di grandi volumi di dati nei *data warehouse*. Ad esempio, per i *retailer* che processano miliardi di righe di dati al giorno, le viste materializzate vengono utilizzate per pre-aggregare i dati, diminuendo il numero di righe da miliardi a milioni e incrementando la scalabilità rispetto all'utilizzo delle tabelle di fatto (Chirkova & Yang, 2011)[8]. Tuttavia, la configurazione ottimale di una vista richiede competenze specialistiche e la capacità di prevedere i carichi di lavoro.

In ambito OLAP, come, ad esempio, il *reporting*, la *business analytics*, il *data mining* e le viste materializzate permettono di effettuare operazioni aggregate e *query* multidimensionali in tempi più brevi rispetto a *join* e *group-by* nelle tabelle di base. Inoltre, l'efficacia delle viste materializzate è utile quando vi è esigenza di eseguire *drill-down* e *roll-up* (Chirkova & Yang, 2011)[8]. Una possibile area da investigare sarebbe l'utilizzo delle viste materializzate nell'analisi predittiva.

Inoltre, le viste materializzate diminuiscono il carico computazionale. Infatti, quando i risultati di una *query* complessa vengono calcolati una volta e materializzati, tutte le *query* successive richiedenti lo stesso risultato non implicano un nuovo calcolo del risultato e i risultati precalcolati vengono consegnati in breve tempo (Stark et al., 2022)[25].

Esiste la possibilità di adottare tecniche di ottimizzazione *multi-query*, le quali consentono di calcolare una sottoespressione che compare in più *query* solamente una volta. Queste tecniche riducono il costo totale materializzando temporaneamente le sottoespressioni condivise, semplificando il carico delle operazioni in presenza di aggiornamenti dei dati e grandi quantità di dati (Mistry et al., 2000)[26].

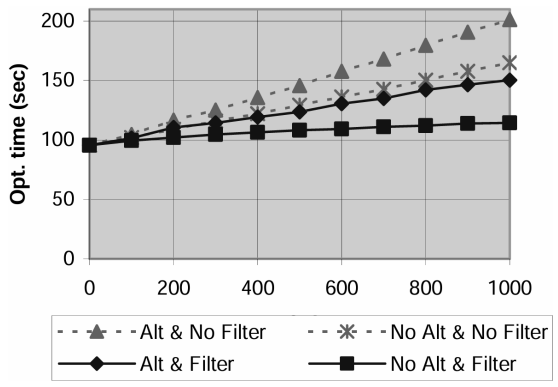


Figura 5.1: Tempo di ottimizzazione in funzione del numero di viste. (Goldstein & Larson, 2001 p. 10)[27]

In ambienti di produzione, dal 60 % all'80 % delle *query* degli utenti vengono risolte tramite i dati materializzati, mostrando che i dati materializzati non hanno effetto solamente sulle *query* ripetitive, ma che permettono anche di accelerare il tempo di risposta per le *query* ad *hoc* (Goldstein & Larson, 2001)[27].

In definitiva, l'efficienza delle viste materializzate, oltre a essere molto utile per abbreviare i tempi di risposta alle *query*, permette di incrementare l'affidabilità del supporto decisionale per le organizzazioni in presenza di un incremento degli

utenti e della dimensione dei dati. Infatti, la disponibilità in tempi brevi delle informazioni relative al supporto decisionale potrebbe consentire alle organizzazioni di competere meglio sul mercato. Inoltre, un'infrastruttura BI inefficiente può avere un impatto negativo sulle organizzazioni, aumentando le perdite e le decisioni errate (Vassiliadis, 2000)[28]. Una domanda da porsi è l'utilità delle viste materializzate, dato che ci sono tecnologie emergenti, come ad esempio i *database in-memory*, che permettono tempi di risposta rapidi senza bisogno di memorizzare i dati.

In sintesi, le viste materializzate sono uno strumento per l'ottimizzazione delle *performance* in un *data warehouse* e possono risultare indispensabili per un sistema di *Business Intelligence* avanzata.

5.2 Algoritmi di selezione e manutenzione

La selezione delle viste materializzate in sistemi di BI è un problema particolarmente complesso definito in letteratura come *Nondeterministic Polynomial time* completo (NP-completo), dal momento che l'individuazione della configurazione ottimale delle viste materializzate richiede un numero di passi computazionali che cresce in maniera esponenziale rispetto alla dimensione del *data warehouse*. Gli algoritmi utilizzati nella selezione delle viste sono euristici, genetici e ibridi, con i quali è possibile individuare, entro costi ragionevoli di tempo e spazio, un insieme di viste materializzate ottimale (Mami & Bellahsene, 2012)[29]. Ad esempio, gli algoritmi euristici definiscono delle regole per ridurre lo spazio di ricerca, mentre gli algoritmi genetici utilizzano una tecnica di evoluzione naturale per esplorare lo spazio di soluzioni. Gli algoritmi ibridi combinano invece le due tecniche, risultando particolarmente efficienti e utili per la gestione della variabilità dei carichi di lavoro e della eterogeneità dei *dataset* nei sistemi di BI.

Il modello di selezione delle viste nei *data warehouse* aziendali è definito "vincolato dallo spazio", che esegue la minimizzazione dei costi delle *query* e della manutenzione delle viste materiale in uno spazio limitato. L'implementazione di tale modello appare la più adatta dal momento che rappresenta i limiti *hardware* e le regole gestionali delle aziende (Mami & Bellahsene, 2012)[29]. Infine, la selezione delle viste dovrà essere guidata da una metrica che indichi i costi e i benefici legati alla materializzazione di un insieme di viste e scelga quello che, per ogni unità di spazio e per ogni unità di costo di manutenzione, porta più beneficio al sistema.

Gli algoritmi ibridi appaiono efficienti nella misura in cui sfruttano le capacità di esplorazione degli algoritmi genetici e le tecniche di riduzione dello spazio di ricerca degli algoritmi euristici per valutare l'insieme delle viste materiale migliore al variare del carico di lavoro e dei dati nei moderni sistemi di BI (Mami & Bellahsene, 2012)[29]. Resta aperto l'aspetto legato all'implementazione e alla gestione di tale approccio con le caratteristiche di un carico di lavoro concreto e le regole gestionali,

implementando una fase preliminare in cui studiare il carico di lavoro e le caratteristiche delle *query* di accesso al *data warehouse*.

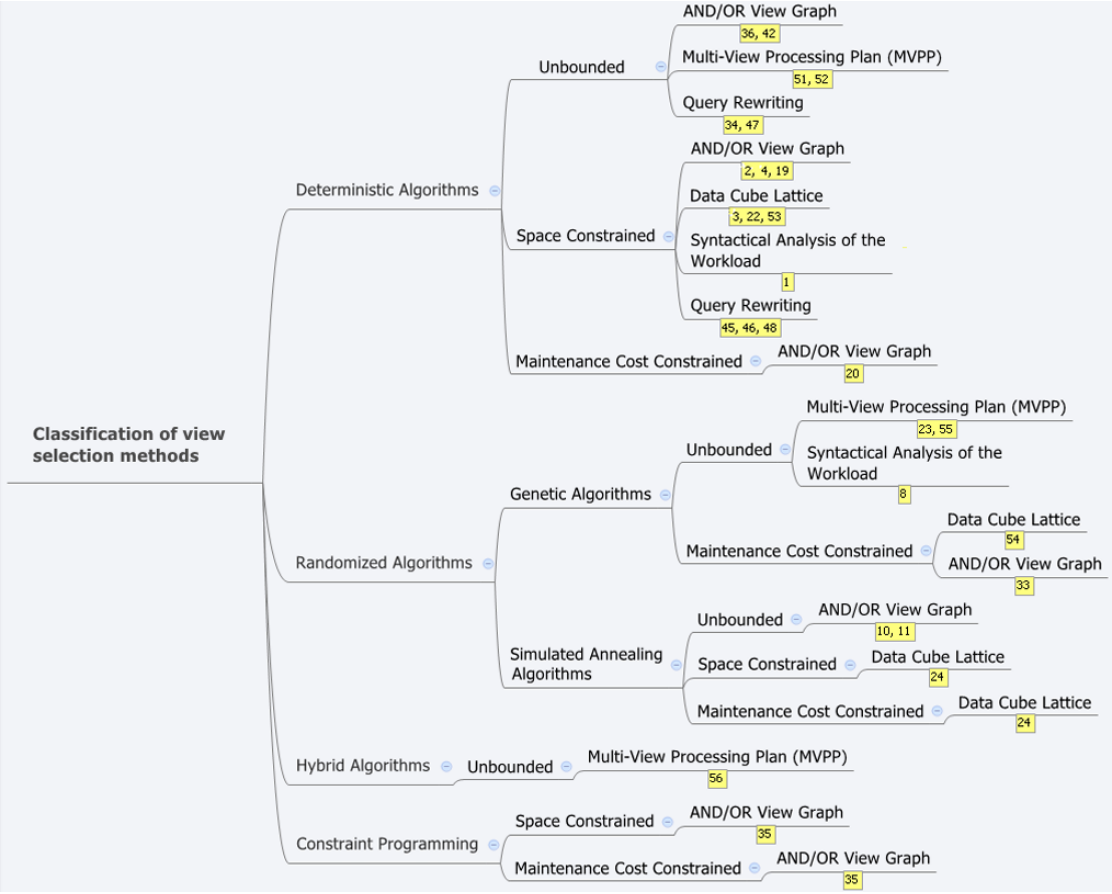


Figura 5.2: Una classificazione dei metodi di selezione delle viste. (Mami & Bellahsene, 2012 p. 25)[29]

La materializzazione di tutte le *query* di *input* risolverebbe completamente il problema della riduzione dei tempi di risposta alle *query*, ma il costo di manutenzione delle viste materializzate diventerebbe elevatissimo. Bisognerà, quindi, selezionare un insieme vantaggioso di viste materiale. Il punto è rappresentato dalla metrica di valutazione, ovvero dal beneficio che le viste portano al sistema per ogni unità di spazio utilizzato e per ogni unità di costo di manutenzione (Mami & Bellahsene, 2012)[29]. Molti modelli di selezione possiedono una proprietà matematica che indica che il beneficio aggiunto è monotòno, che permettono di decidere in maniera ottima la selezione di una nuova vista da materializzare dopo aver effettuato la decisione per le prime viste.

Un'ottima tecnica di selezione delle viste è quella di usare l'algoritmo A*, dal momento che si tratta di una tecnica efficiente che può potare lo spazio di ricerca ed eliminare così le configurazioni non ottime. L'algoritmo A* è particolarmente uti-

le nel caso in cui le *query* di *input* abbiano più predicati di selezione, in quanto la potenza di "potatura" dell'algoritmo aumenta con il crescere della complessità del problema (Labio et al., 1996)[23]. Implementare A* risulta una soluzione utile poiché può essere inoltre affinata con tecniche di *caching*, potatura e altre regole gestionali che spingono il selezionatore verso scelte realizzabili in concreto. Infine, è di grande interesse la gestione dello spazio materiale nei moderni sistemi di BI. A questo scopo, l'algoritmo A* permette di capire, rispetto ad un insieme di indici ottimi, se ad un dato livello di spazio materiale, un determinato insieme di viste materiale fornisce meno o più beneficio. In questa situazione, si può vedere se è più conveniente inserire una vista o un indice tradizionale (Labio et al., 1996)[23].

Tabella 5.1: Comparazione di A* e algoritmi esaustivi. (Labio et al., 1996 p. 12)[23]

# **relations**	# **selections**	# **states visited (exhaustive)**	# **states visited (A*)**	% **pruned**
2	0	32	11	67.7
2	1	192	21	89.1
2	2	960	28	97.1
2	4	960	29	97.0
3	1	2 115 072	17 735	99.2
3	2	10 575 360	22 809	99.8

È particolarmente importante la tecnica di ottimizzazione *multi-query* dal momento che condivide la sottoespressione per la manutenzione delle viste materializzate. L'ottimizzazione *multi-query* materializza in maniera temporanea i risultati intermedi, in modo da evitare il ricalcolo da zero per *query* che richiedono lo stesso calcolo (Mistry et al., 2000)[26]. L'ottimizzazione *multi-query* è importante anche per avere un OLAP scalabile e che possa gestire carichi di lavoro più importanti senza penalizzare il tempo di risposta.

Per quanto riguarda la manutenzione delle viste materializzate, l'ideale, per minimizzare i costi, è di utilizzare una tecnica di manutenzione incrementale. Questa tecnica è molto efficiente dal momento che, invece di effettuare la vista del *database* quando la tabella viene modificata, la manutenzione deve solo essere eseguita per quelle tuple che hanno avuto delle modifiche (Chirkova & Yang, 2011)[8]. Nei casi più complessi, come nei grandi *retailer*, è efficiente dal momento che aggiorna gli aggregati che sono stati modificati senza scansionare dell'intera tabella di fatto. Questo aspetto permette di avere una maggiore reattività da parte dei sistemi di manutenzione, non richiedendo un lungo scan anche in situazioni in cui le tabelle di fatto superano il miliardo di *record* (Chirkova & Yang, 2011)[8].

La manutenzione incrementale si applica anche ai *Database Management System* (DBMS) che non offrono in maniera nativa il servizio di viste materializzate. In questo caso è possibile emulare il funzionamento attraverso strumenti applicativi come *trigger* o *script* che avviano la logica per la manutenzione delle viste mate-

rializzate (Chirkova & Yang, 2011)[8]. La manutenzione incrementale è importante per la gestione integrata del *caching* avanzato di risultati temporanei delle *query*. In questo contesto, la coerenza dei risultati delle *query* è rafforzata dal *caching* incrementale e da quello di manutenzione.

Per il futuro, appare interessante la selezione delle viste tramite algoritmi *data-driven* e di *machine learning*. La possibilità di profilare i carichi di lavoro ad esempio con modelli neurali complessi come i *convolutional bi-directional* LSTM, usati con successo in ambito industriale, permette di individuare anche strategie di materializzazione, manutenzione e *updating* delle viste materializzate più adatte al *business* dell'organizzazione (Zhao et al., 2017)[30].

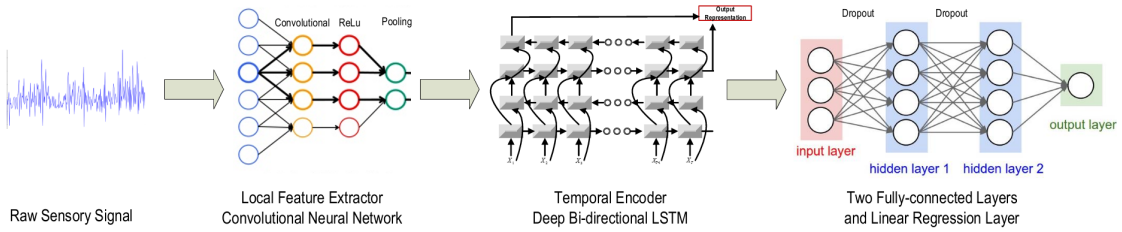


Figura 5.3: Proposta di struttura di *Convolutional Bi-directional Long Short-Term Memory* (CBLSTM). (Zhao et al., 2017 p. 5)[30]

Appare chiaro che questa direzione di *data-driven* dovrà essere integrata con algoritmi già in uso come gli algoritmi genetici e gli algoritmi di potatura. Si andrà a delineare un ambiente misto *data-driven* ed euristico di selezione e manutenzione, dal momento che appare chiaro come per l'individuo non sia possibile governare la varietà delle interazioni nei complessi e per certi versi inesplorati sistemi di BI.

5.3 Bilanciamento tra prestazioni e costi

Il bilanciamento tra prestazioni e costi delle viste materializzate nella *Business Intelligence* richiede un'attenta valutazione del carico di lavoro e delle risorse disponibili. A livello empirico, è noto come più dell'80 % delle *query* in scenari reali possa essere gestito mediante viste materializzate, con conseguente ottimizzazione dei tempi di risposta e delle risorse consumate per l'elaborazione delle analisi (Chirkova & Yang, 2011)[8]. Tuttavia, i costi operativi di aggiornamento delle viste materializzate possono diventare rilevanti, soprattutto se ci sono frequenti inserimenti, cancellazioni o modifiche dei dati base. Per bilanciare i costi è necessario selezionare un insieme ottimo di viste, considerando anche la loro frequenza di accesso e la periodicità degli aggiornamenti necessari.

Con l'applicazione di algoritmi efficienti, come l'A*, nello spazio delle possibili configurazioni di viste materializzate e indici, lo spazio di ricerca viene drasticamente

ridotto, anche del 99 % per alcune scelte (Labio et al., 1996)[23]. La loro applicazione è fondamentale per permettere a un amministratore di *data warehouse* di ottenere soluzioni accettabili anche in presenza di poche risorse. In contesti in cui la memoria a disposizione è un limite, la soluzione delle viste materializzate è in genere preferita a quella degli indici. Con un aumento dei costi, ma una diminuzione dello spazio totale occupato, le prestazioni migliorano, rendendo quest'opzione più allettante rispetto a quelle che ricorrono esclusivamente agli indici, anche se per garantire che le risorse computazionali aggiunte siano utilizzate efficacemente è indispensabile, però, effettuare un'analisi appropriata dei *pattern* di accesso e dei tempi di aggiornamento.

Per ridurre i costi totali di aggiornamento, diventa importante impiegare tecniche di ottimizzazione *multi-query* e strategie di aggiornamento incrementale delle aggregazioni. La condivisione di sottoespressioni comuni tra *query* diverse e la loro materializzazione provvisoria, a richiesta, permettono di limitare il costo computazionale agli aggiornamenti indispensabili, senza ricalcolare per ogni *query* anche le sottoespressioni comuni (Mistry et al., 2000)[26]. Nei grandi *data warehouse* le tecniche di aggiornamento incrementale si rivelano fondamentali in situazioni come quella dei grandi *retailer*, in cui solo alcune aggregazioni sono interessate da una modifica; si effettua quindi solo l'aggiornamento di queste, evitando il ricalcolo completo di tutte le aggregazioni (Chirkova & Yang, 2011)[8]. Quest'approccio è ancora più vantaggioso quando gli aggiornamenti non sono distribuiti uniformemente su tutto il *data warehouse*.

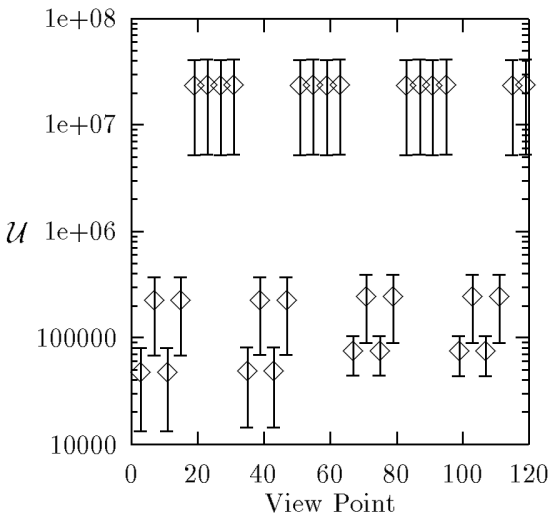


Figura 5.4: Ogni ascissa rappresenta un insieme di viste; Le ordinate mostrano l'intervallo di costi di aggiornamento fra indice peggiore e migliore. Evidenzia come esistano molte configurazioni di viste vicine all'ottimo e renda cruciale la scelta degli indici anche dopo aver selezionato un "buon" insieme di viste. (Labio et al., 1996 p. 12)[23]

Per minimizzare i costi è importante scegliere un insieme di viste materializzate che risponda alle esigenze tipiche delle *query* in questione, effettuando aggiornamenti incrementali e mantenendo in memoria solo le viste usate più spesso (Gurcan et al., 2023 [5]; Hussain et al., 2025 [31]; Tsai, 2011 [32]). Queste tecniche sono adeguate quando i costi di storage sono contenuti. In questo caso, invece, i costi di manutenzione e di aggiornamento tendono ad essere preponderanti.

Il compromesso tra prestazioni e costi delle viste materializzate è in questi anni

oggetto di grande interesse nella ricerca internazionale. La sua importanza è evidente dai risultati più rilevanti degli ultimi vent'anni pubblicati nella letteratura della *Business Intelligence*, che ne hanno sottolineato la crescente priorità, spingendo sempre più verso soluzioni di *machine learning* auto-adattanti in grado di minimizzare i costi in ambienti a grande complessità. Ciò nonostante, l'adozione di sistemi adattivi, che migliorano di pari passo con l'acquisizione di nuove informazioni, è tutt'oggi un tema relativamente trascurato, in particolare nell'ambito dei *Big Data*, dove la crescita esponenziale dei dati e la sempre maggiore variabilità dei carichi di lavoro richiedono strategie che si evolvano di pari passo con la mutabilità dei *data warehouse* (Gurcan et al., 2023)[5].

Una soluzione per superare questo problema potrebbe essere quella di una combinazione di tecniche: da una parte, il classico approccio di progettazione fisica mediante algoritmi, dall'altra, un approccio automatico di apprendimento, per trovare i parametri ottimali, non a partire da dati di riferimento, ma dal reale incremento del carico di lavoro e dagli andamenti delle prestazioni durante l'esercizio (Gurcan et al., 2023)[5]; Hussain et al., 2025 [31]; Labio et al., 1996 [23]; Tsai, 2011 [32]).

Capitolo 6

Viste materializzate OLAP su SQL Server e Synapse

Questo capitolo tratterà dell'implementazione, dell'impiego da parte dell'ottimizzatore e del monitoraggio delle viste materializzate in Azure Synapse – Dedicated SQL Pool e delle viste indicizzate in Microsoft SQL Server, con riferimento a carichi OLAP su *data warehouse* e in continuità con le scelte fisiche illustrate nel Capitolo 4 e con i criteri di selezione/manutenzione formalizzati nel Capitolo 5 (Vaisman & Zimányi, 2022 [1]; Microsoft, 2024–2025 [33]).

Lo scopo è tradurre i principi teorici in procedure operative riproducibili, definendo come creare correttamente gli oggetti, come abilitare il *query rewrite* e come governarne la manutenzione in presenza di aggiornamenti, con misure che rendano verificabile il rapporto tra beneficio prestazionale e costo gestionale (Vaisman & Zimányi, 2022 [1]; Microsoft, 2024–2025 [33]).

L'approfondimento si atterrà strettamente a funzionalità e vincoli così come documentati nelle fonti ufficiali e nella letteratura. La verifica sperimentale utilizzerà un sottoinsieme del *workload* TPC-DS per ancorare i risultati a uno *standard* pubblico, e farà ricorso esclusivo a strumenti nativi (piani di esecuzione/EXPLAIN, DMV, comandi DBCC per la stima dell'*overhead* e l'eventuale REBUILD), evitando proposte euristiche non supportate (TPC, 2021 [34]; Microsoft, 2024–2025 [33]).

Questo approfondimento si limita al SQL relazionale dei due motori considerati; sono pertanto esclusi linguaggi per cubi (ad esempio: MDX) e tecnologie non pertinenti. La ricerca si concentra unicamente alla messa in pratica documentata di viste materializzate/viste indicizzate e alla loro valutazione misurata sul caso d'uso adottato (Vaisman & Zimányi, 2022 [1]; Microsoft, 2024–2025 [33]).

Per gli esempi e per la validazione empirica si fa riferimento al *workload* TPC-DS, ampiamente utilizzato in letteratura per rappresentare scenari OLAP con numerose

join e aggregazioni a diverse granularità. L'impiego di TPC-DS consente di ancorare i risultati a uno *standard* pubblico, garantendo confrontabilità e riproducibilità delle misure. La verifica dei benefici e dei costi è svolta con strumenti nativi: analisi dei piani di esecuzione (EXPLAIN, incluse le raccomandazioni in Synapse), interrogazione delle *Dynamic Management Views* (DMV) e utilizzo dei comandi DBCC per stimare l'*overhead* di manutenzione e pianificare eventuali REBUILD. (TPC, 2021 [34]; Microsoft, 2024–2025 [33]).

6.1 Implementazione in SQL Server e Azure Synapse

6.1.1 Indexed views (SQL Server) e Materialized views (Synapse)

Nel quadro di un *data warehouse* OLAP, SQL Server e Azure Synapse (Dedicated SQL Pool) implementano la materializzazione con modelli differenti ma finalità coincidenti: precalcolare e memorizzare i risultati di *join* e aggregazioni per ridurre latenza e I/O su interrogazioni ripetitive. In SQL Server la materializzazione è ottenuta tramite viste indicizzate: una vista definita con vincoli formali stringenti (6.1.2) diventa fisicamente persistita non appena si crea il primo indice *cluster* univoco; da quel momento, l'ottimizzatore può impiegarla anche senza riferimento esplicito nel testo SQL, sostituendo sotto-piani di *join*/aggregazione con accessi all'oggetto materializzato quando il costo stimato lo giustifica. Tale meccanismo, documentato nelle linee guida ufficiali, dipende dalla corretta impostazione delle opzioni SET, dal determinismo delle espressioni e dalla compatibilità semantica tra la *query* e la vista (Microsoft, 2024–2025 [33]; Vaisman & Zimányi, 2022 [1]).

```
-- Requisiti SET (estratto essenziale per viste indicizzate)
SET ANSI_NULLS ON;
SET QUOTED_IDENTIFIER ON;
SET ARITHABORT ON;
SET ANSI_WARNINGS ON;
SET CONCAT_NULL_YIELDS_NULL ON;
SET NUMERIC_ROUNDABORT OFF;
GO

-- Vista con SCHEMABINDING e COUNT_BIG(*) in presenza di GROUP BY
CREATE VIEW dbo.SalesAgg WITH SCHEMABINDING AS
SELECT
    fs.ProductID,
    fs.MonthKey,
    COUNT_BIG(*)            AS CntRows,   -- richiesto per viste
    indicizzate con GROUP BY
    SUM(fs.SalesAmount)     AS Amt
FROM dbo.FactSales AS fs
GROUP BY fs.ProductID, fs.MonthKey;
GO

-- La materializzazione avviene creando come primo indice univoco cluster
CREATE UNIQUE CLUSTERED INDEX IX_SalesAgg_UC
```

```
23     ON dbo.SalesAgg(ProductID, MonthKey);
```

Listing 6.1: Vista indicizzata con SCHEMABINDING e indice *cluster* univoco in SQL Server (esempio originale)

In Synapse, invece, la materializzazione è un oggetto nativo creato con CREATE MATERIALIZED VIEW ... AS SELECT ..., archiviato in *columnstore* e dotato di una distribuzione (HASH o ROUND_ROBIN) che ne determina il comportamento in un ambiente MPP. L'ottimizzatore può eseguire un *query rewrite* automatico, servendo una *query* dalla vista anche in assenza di citazione esplicita, purché la definizione catturi un'aggregazione e un insieme di *join* compatibili con il sotto-piano richiesto; il controllo avviene tramite analisi del piano stimato/EXPLAIN, come discusso in 6.1.3 (Microsoft, 2023–2024)[35]. Rispetto a SQL Server, Synapse enfatizza la co-locazione dei dati tramite distribuzione HASH per ridurre gli *shuffle* in fase di *join* e favorire piani più stabili sul carico analitico (Vaisman & Zimányi, 2022)[1].

Dal punto di vista concettuale (come discusso nel Capitolo 5), entrambi i motori implementano la stessa idea generale di vista materializzata: una valutazione precalcolata di espressioni relazionali che bilancia i benefici in lettura con i costi di manutenzione generati dai DML sulle tabelle di base. La letteratura sottolinea che questo "*maintenance problem*" (cioè quanto, quando e come aggiornare gli oggetti materializzati) è il cuore del compromesso progettuale e spiega la presenza, nelle implementazioni industriali, di vincoli e procedure atte a garantire correttezza e prevedibilità (Chirkova & Yang, 2012 [8]; Vaisman & Zimányi, 2022 [1]). Riassumendo, viste indicizzate (SQL Server) e *materialized views* (Synapse) convergono su un medesimo obiettivo: riutilizzare risultati condivisi per stabilizzare le latenze delle *query* OLAP, delegando all'ottimizzatore l'uso trasparente quando le condizioni semantiche e fisiche lo consentono. (Microsoft, 2023–2025 [36]; Chirkova & Yang, 2012 [8]; Vaisman & Zimányi, 2022 [1]).

6.1.2 Requisiti e vincoli per la creazione di viste materializzate

Nel caso di SQL Server, la materializzazione avviene tramite viste indicizzate e impone una serie di condizioni formali non negoziabili. La vista deve essere definita con SCHEMABINDING, in modo da vincolare lo schema delle tabelle sottostanti. Tutte le espressioni utilizzate devono essere deterministiche, in presenza di raggruppamenti è richiesto COUNT_BIG(*) per garantire correttezza e manutenibilità. Durante la creazione e utilizzo devono essere attive specifiche opzioni SET (ad esempio ANSI_NULLS ON, QUOTED_IDENTIFIER ON, ARITHABORT ON, ecc.). Infine, il primo indice creato sulla vista deve essere un *unique clustered index*, prerequisito che abilita la persistenza fisica e l'eventuale impiego trasparente da parte dell'ottimizzatore anche senza citazione esplicita della vista nel testo SQL. Tali vincoli (insieme ad ulteriori restrizioni su costrutti non supportati) sono documentati nella guida

ufficiale e mirano a rendere sicura la manutenzione incrementale e l'ottimizzazione dei piani. (Microsoft, 2025)[37].

```
-- Materialized view: distribuzione HASH per co-locare le join ricorrenti
CREATE MATERIALIZED VIEW dbo.mv_SalesByCatMonth
WITH (DISTRIBUTION = HASH(CategoryID))
AS
SELECT
    dp.CategoryID,
    fs.MonthKey,
    SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
JOIN dbo.DimProduct AS dp
  ON dp.ProductID = fs.ProductID
GROUP BY dp.CategoryID, fs.MonthKey;
GO

-- Manutenzione quando cresce l'overhead incrementale
ALTER MATERIALIZED VIEW dbo.mv_SalesByCatMonth REBUILD;
```

Listing 6.2: Synapse: `CREATE MATERIALIZED VIEW` con `DISTRIBUTION = HASH` e REBUILD (esempio originale)

Per Azure Synapse (Dedicated SQL Pool), le *materialized views* sono oggetti nativi creati con l'istruzione CREATE MATERIALIZED VIEW ... AS SELECT ... e richiedono la scelta esplicita della distribuzione: HASH su una o più chiavi consente co-locazione e riduzione del *data movement*, mentre ROUND_ROBIN opera come distribuzione neutra e si adotta tipicamente in assenza di chiavi naturali o in scenari transitori. Il formato di archiviazione è *columnstore* e la definizione non può referenziare altre viste. La documentazione segnala inoltre che alcune aggregazioni non sono ammesse in definizione (ad esempio COUNT(DISTINCT)), pur potendo l'ottimizzatore riscrivere *query* che le includono servendosi di una *materialized view* compatibile. Sono previsti, infine, aspetti di *ownership*/permessi e operazioni di REBUILD per contenere l'*overhead* in presenza di intensi aggiornamenti. (Microsoft, 2023 [38]; Microsoft, 2022–2023 [39]).

Il razionale OLAP che giustifica questi requisiti è duplice. Da un lato, vincoli come SCHEMABINDING, determinismo e chiavi univoche (SQL Server) o una distribuzione coerente con le chiavi di *join* (Synapse) consentono al motore di riconoscere in modo affidabile i sotto-piani di *join* e aggregazione dell'interrogazione e di sostituirli con risultati precalcolati. Dall'altro, tali vincoli limitano il costo di manutenzione incrementale al variare dei dati, che rimane l'elemento critico di ogni strategia di materializzazione. In termini pratici, nelle gerarchie di *roll-up* tipiche dei d*ata warehouse* (giorno→mese→anno; prodotto→categoria) la definizione di viste deterministiche e allineate alla distribuzione riduce lo *shuffle* e stabilizza le latenze, purché si monitorino periodicamente statistiche e *overhead* e si pianifichino eventuali REBUILD nelle finestre ETL. Queste scelte sono coerenti con le linee guida industriali e con l'inquadramento accademico della progettazione fisica dei *data warehouse*. (Vaisman & Zimányi, 2022 [1]; Microsoft, 2022–2025 [40]).

6.1.3 Utilizzo da parte dell'ottimizzatore

Nel Dedicated SQL Pool di Azure Synapse l'ottimizzatore può effettuare una riscrittura automatica delle interrogazioni, sostituendo porzioni di piano con accessi a *materialized views* semanticamente compatibili anche in assenza di un riferimento esplicito alla vista nel testo SQL. La verifica operativa di tale comportamento si esegue tramite EXPLAIN, eventualmente con l'opzione WITH RECOMMENDATIONS, che restituisce il piano stimato, evidenziando operatori, movimenti dati e, quando pertinente, suggerimenti o utilizzi di viste materializzate idonee (Microsoft, 2024a)[41]. In termini pratici, l'effettivo *matching* richiede che la definizione della vista catturi il sotto-piano di *join* e aggregazione della *query*, con colonne, filtri e granularità coerenti. In presenza di predicati più selettivi, l'ottimizzatore può comunque riutilizzare l'aggregato e applicare i filtri a valle, purché la correttezza semantica sia preservata. L'assenza di riferimenti ad altre viste nella definizione, il rispetto dei vincoli espressi in 6.1.2 e la disponibilità di statistiche aggiornate aumentano la probabilità di riscrittura. Al contrario, differenze strutturali, uso di funzioni non supportate o stato di manutenzione degradato della vista ne limitano l'impiego. L'analisi dei piani consente quindi di accertare non solo la scelta della *materialized view* ma anche l'eventuale eliminazione di *shuffle* su *join* ripetitivi, aspetto che nei carichi OLAP incide in modo determinante sulla latenza complessiva (Microsoft, 2024a)[41].

```
1  -- Forzare l'uso dell'indice materializzato della vista quando la si cita
       nel FROM
2  SELECT
3      s.ProductID, s.MonthKey, s.Amt
4  FROM dbo.SalesAgg AS s WITH (NOEXPAND);
5
6  -- Impedire l'impiego di viste indicizzate nel piano (espansione forzata)
7  SELECT
8      fs.ProductID, fs.MonthKey, SUM(fs.SalesAmount) AS Amt
9  FROM dbo.FactSales AS fs
10 GROUP BY fs.ProductID, fs.MonthKey
11 OPTION (EXPAND VIEWS);
```

Listing 6.3: SQL Server: controllo dell'uso di viste indicizzate con `NOEXPAND` / `EXPAND VIEWS` (esempio originale)

Nel caso di SQL Server, le viste indicizzate possono essere selezionate dall'ottimizzatore anche senza menzione esplicita quando il costo stimato lo giustifica e quando sono soddisfatti i requisiti formali della vista e delle opzioni SET associate. In specifici scenari è tuttavia possibile orientare la scelta del piano mediante *hint*. L'*hint* di tabella NOEXPAND evita che la vista venga espansa sulle tabelle sottostanti, inducendo l'uso dell'indice materializzato collegato. Dall'altro lato, l'*hint* di *query* EXPAND VIEWS forza l'espansione e inibisce l'impiego della vista indicizzata (Microsoft, 2025a)[37]. L'uso degli *hint* va limitato a casi mirati, ad esempio per analisi "*what-if*", per mitigare regressioni di piano o in edizioni/configurazioni in cui il motore non consideri automaticamente la vista. Questo perché viene introdotta

una dipendenza esplicita dalla scelta del piano che potrebbe non generalizzare a versioni, statistiche o carichi diversi (Microsoft, 2025a [42]; Microsoft, 2025b [43]). In sintesi, la strategia consigliata in entrambi i motori è osservazionale e misurativa: progettare e mantenere correttamente la vista, verificare con EXPLAIN/piano di esecuzione l'effettivo *rewrite* o *matching* e intervenire con *hint* solo come *extrema ratio*, seguendo le pratiche documentate (Microsoft, 2024a [41]; Microsoft, 2025a [42]; Microsoft, 2025b [43]).

6.1.4 Ottimizzazione di query OLAP in Synapse e SQL Server

Nel pool SQL dedicato di Azure Synapse, l'ottimizzazione delle interrogazioni OLAP ruota attorno alla riduzione del *data movement* e alla qualità delle stime dell'ottimizzatore. La scelta della distribuzione delle tabelle (e delle viste materializzate e degli intermedi persistiti) è il primo determinante. HASH sulle chiavi di *join* ricorrenti consente co-locazione delle righe e piani con meno shuffle, ROUND_ROBIN è neutro e si adotta solo quando non esiste una chiave naturale o per scenari transitori, poiché può aumentare gli scambi di dati durante le *join* su grandi fatti e dimensioni (Microsoft, 2025 [37]; Microsoft, 2024 [44]). L'utilità dei piani dipende inoltre da statistiche aggiornate: su Synapse è prassi creare/aggiornare statistiche istogrammatiche sulle colonne di filtro e di *join* dopo i caricamenti *bulk* o le trasformazioni massive, per evitare selettività errate e scelte subottimali di movimentazione (Microsoft, 2025 [37]; Microsoft, 2022 [45]).

Per contenere i costi delle aggregazioni ripetute e dei ripartizionamenti, è efficace l'uso di CTAS (CREATE TABLE AS SELECT) per materializzare risultati intermedi già distribuiti in modo ottimale, nonché delle tabelle temporanee di sessione, che scrivono su *storage* locale e riducono il traffico remoto durante *pipeline* complesse (Microsoft, 2025 [37]; Microsoft, 2022 [45]). A livello di formato fisico, il *columnstore* è il *default* nel *data warehouse* MPP e rimane centrale anche nell'ecosistema SQL Server *on-prem/managed*: l'archiviazione per colonne consente elevate compressioni e scansioni vettoriali, risultando appropriata per fatti di grandi dimensioni e *workload* analitici; il partizionamento temporale favorisce la gestione di finestre e operazioni di manutenzione (Microsoft, 2025)[37].

```
1  -- 1) CTAS: materializza una tabella intermedia co-locata sulle join
       ricorrenti
2  CREATE TABLE dbo.FactSales_hash
3  WITH
4  (
5    DISTRIBUTION = HASH(ProductID),
6    CLUSTERED COLUMNSTORE INDEX
7  )
8  AS
9  SELECT *
10 FROM dbo.FactSales_stage;  -- staging/caricamenti bulk
11 GO
```

```
12
13 -- 2) Materialized View: aggregazione riusabile (coerente con le join su
       ProductID, MonthKey)
14 CREATE MATERIALIZED VIEW dbo.mv_SalesAgg
15 WITH (DISTRIBUTION = HASH(ProductID))
16 AS
17 SELECT
18   fs.ProductID,
19   fs.MonthKey,
20   SUM(fs.SalesAmount) AS Amt
21 FROM dbo.FactSales_hash AS fs
22 GROUP BY fs.ProductID, fs.MonthKey;
23 GO
24
25 -- 3) STATISTICS: istogrammi sulle colonne di filtro/join per stime
       corrette
26 CREATE STATISTICS S_FactSales_Product_Month
27 ON dbo.FactSales_hash(ProductID, MonthKey)
28 WITH FULLSCAN;    -- dopo bulk load/trasformazioni massicce
29 GO
30
31 -- 4) Temporary table di sessione: restringe il dominio (evita shuffle
       superflui)
32 CREATE TABLE #TopProducts (ProductID INT PRIMARY KEY);
33 INSERT INTO #TopProducts VALUES (101),(205),(309);
34 GO
35
36 -- 5) Verifica del query rewrite: la query NON cita la MV
37 EXPLAIN WITH_RECOMMENDATIONS
38 SELECT
39   fs.ProductID,
40   fs.MonthKey,
41   SUM(fs.SalesAmount) AS Amt
42 FROM dbo.FactSales_hash AS fs
43 JOIN #TopProducts AS t
44   ON t.ProductID = fs.ProductID
45 GROUP BY fs.ProductID, fs.MonthKey;
46 GO
47
48 -- 6) Monitoraggio overhead MV e manutenzione programmata
49 DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD('dbo.mv_SalesAgg');
50 -- Se l'overhead_ratio cresce oltre la soglia operativa:
51 ALTER MATERIALIZED VIEW dbo.mv_SalesAgg REBUILD;
52 GO
```

Listing 6.4: Synapse: ciclo di ottimizzazione OLAP con CTAS, MV, STATISTICS, *temp table*, EXPLAIN e DBCC OVERHEAD/REBUILD (esempio originale)

Sul piano delle tecniche di risposta rapida, è utile distinguere *result-set caching* e viste materializzate. La prima restituisce esiti identici per *query* ripetute alla lettera, eliminando la rielaborazione ma offrendo riuso solo *full-result*. Le seconde abilitano il riuso parziale tramite riscrittura, supportando varianti di predicati/*roll-up* e restando efficaci anche quando la *query* non coincide con la definizione della vista (Microsoft, 2022 [45]; Microsoft, 2023 [38]). La diagnostica dei piani e delle opportunità di riscrittura si conduce con EXPLAIN (eventualmente con l'opzione WITH RECOMMENDATIONS) per osservare operatori, costi stimati e movimenti dati prima dell'esecuzione (Microsoft, 2024)[44]. Per le viste materializzate, l'*overhead* incrementale derivante dai DML si misura con DBCC PDW_SHOWMATERIALIZEDVIEWOVE

RHEAD, che espone conteggi di *tracking* e un *overhead ratio* utile a programmare i REBUILD nelle finestre ETL, evitando degradazioni progressive della latenza (Microsoft, 2024)[44].

In sintesi, un ciclo efficace su Synapse combina distribuzione coerente, statistiche curate, materializzazioni intermedie (CTAS/temporanee) e *columnstore*, governando l'uso di *cache* dei risultati rispetto alle viste materializzate e supportando le scelte con EXPLAIN e DBCC. Questi principi, per analogia, trovano riscontro nell'uso degli indici *columnstore* anche in SQL Server per scenari del *data warehouse*. (Microsoft, 2025)[37].

6.2 Tuning e monitoraggio: caso Synapse e SQL Server

6.2.1 Metodologia operativa di ottimizzazione

La metodologia adottata segue un ciclo iterativo e riproducibile centrato su profilazione, progettazione, validazione e monitoraggio. In primo luogo, si profilano le interrogazioni più gravose mediante il piano stimato (EXPLAIN, in Synapse anche con l'opzione WITH RECOMMENDATIONS), così da individuare operatori dominanti, movimenti dati (*shuffle*/*broadcast*) e opportunità di riscrittura verso viste materializzate proposte dal motore. La profilazione definisce la *baseline* prestazionale (tempi e varianza) e vincola le decisioni successive, garantendo tracciabilità metodologica coerente con il quadro del *data warehousing* presentato nei capitoli precedenti (Microsoft, 2024a [41]; Vaisman & Zimányi, 2022 [1]).

Sulla base dell'analisi, si progettano viste materializzate che catturano *join* e aggregazioni condivise da più *query* del *workload*, privilegiando definizioni a massima riusabilità e minimizzando la cardinalità dei gruppi. In Synapse la scelta della distribuzione è cruciale per ridurre il *data movement*: HASH sulle chiavi di *join* ricorrenti quando possibile, ROUND_ROBIN solo in assenza di una chiave naturale o per scenari transitori. In presenza di chiavi composite e skew si valuta la *multi-column distribution* quando supportata. Le definizioni rispettano i vincoli formali già esposti (determinismo, funzioni ammesse, assenza di riferimenti ad altre viste), in modo da preservare la possibilità di riscrittura automatica e la sostenibilità della manutenzione (Microsoft, 2024b [46]; Vaisman & Zimányi, 2022 [1]).

Segue la validazione del comportamento dell'ottimizzatore. Dopo la creazione, si verifica che il piano indirizzi la *query* verso la vista anche senza citarla nel testo SQL (*auto-matching*), confermando l'effettiva sostituzione semantica dei sotto-piani di *join*/aggregazione. In SQL Server il controllo è analogo per le viste indicizzate, con eventuale impiego prudente di *hint* per casi particolari, come richiamato nella letteratura tecnica. L'obiettivo rimane il medesimo: garantire che l'ottimizzatore

selezioni il piano con costo atteso minore a parità di semantica (Microsoft, 2024a [41]; Vaisman & Zimányi, 2022 [1]).

La fase finale riguarda il monitoraggio e la *governance*. In Synapse si misura l'*overhead* incrementale con DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD e, al superamento di soglie operative (ad esempio l'aumento marcato del rapporto di *overhead* o delle *tracking rows*), si esegue un REBUILD programmato nelle finestre ETL per ripristinare l'efficienza di scansione. In parallelo si discrimina tra *result-set caching* e *materialized views*: la *cache* è efficace per richieste ripetute identiche (riuso del risultato completo), mentre le viste permettono riuso parziale tramite riscrittura su *pattern* condivisi e restano utili anche con predicati/*roll-up* variabili. Le decisioni devono essere documentate e rese misurabili, predisponendo le metriche e gli strumenti di 6.2.2 (Microsoft, 2023a [47]; Microsoft, 2023b [48]; Microsoft, 2024b [46]; Vaisman & Zimányi, 2022 [1]).

```
-- 1) PROFILARE: piano stimato e raccomandazioni
EXPLAIN WITH_RECOMMENDATIONS
SELECT
  fs.StoreID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
JOIN dbo.DimStore AS ds ON ds.StoreID = fs.StoreID
GROUP BY fs.StoreID, fs.MonthKey;

-- 2) PROGETTARE: MV che cattura join+aggregazione condivise, co-locata su StoreID
CREATE MATERIALIZED VIEW dbo.mv_SalesByStoreMonth
WITH (DISTRIBUTION = HASH(StoreID))
AS
SELECT
  fs.StoreID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
GROUP BY fs.StoreID, fs.MonthKey;

-- 3) VALIDARE: la query non cita la MV; ottimizzatore riscrive in automatico
EXPLAIN
SELECT
  fs.StoreID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
GROUP BY fs.StoreID, fs.MonthKey;
-- Atteso: piano che legge da dbo.mv_SalesByStoreMonth (auto-matching).

-- 4) MONITORARE: overhead incrementale e manutenzione programmata
DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD('dbo.mv_SalesByStoreMonth');
-- Se overhead_ratio supera la soglia operativa:
ALTER MATERIALIZED VIEW dbo.mv_SalesByStoreMonth REBUILD;

-- 5) CACHE vs MV: attivare/disattivare la result-set cache per test comparativi
SET RESULT_SET_CACHING ON;   -- riuso solo full-result per query identiche
-- Eseguire N volte la stessa SELECT per osservare hit/miss della cache
```

```
37 SET RESULT_SET_CACHING OFF;  -- disabilita la cache, resta il riuso
       parziale     via MV
```

Listing 6.5: Synapse — ciclo operativo: EXPLAIN, MV con distribuzione HASH, validazione del *rewrite*, monitoraggio *overhead* e confronto con la *result-set cache* (esempio originale)

6.2.2 Metriche di valutazione e strumenti di monitoraggio

La valutazione degli effetti delle viste materializzate richiede un impianto metrico coerente con il carico OLAP e con i vincoli di aggiornamento dei dati (soglia massima di *staleness*) e di manutenzione delle viste. Il punto di partenza è la misura dei tempi di risposta a parità di *dataset* e parametri di esecuzione: si considerano media, mediana e varianza (o percentili) su un numero sufficiente di *run*, istinguendo misure a *cache* non inizializzata (*cold cache*) e a *cache* inizializzata (*warm cache*) per evitare *bias* da *caching* transitorio. Queste statistiche danno conto sia del guadagno medio sia della stabilità delle latenze, che in *data warehouse* è spesso un requisito contrattuale oltre che tecnico (Vaisman & Zimányi, 2022 [1]).

Accanto ai tempi, in Synapse si monitora l'*overhead* incrementale delle *materialized view* tramite DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD, che espone conteggi di righe "tracciate" e un *overhead_ratio* sintetico. Tale rapporto cresce al crescere dei DML sulle tabelle di base e, oltre soglie operative definite dal team, tende a penalizzare le scansioni della vista. In questi casi si programma un REBUILD nelle finestre ETL per ripristinare l'efficienza. La disponibilità di questo indicatore consente di legare la decisione di manutenzione a una misura riproducibile e documentata dal fornitore (Microsoft, 2024a)[41].

```
1  /* Stato dei rowgroup columnstore (qualita'/compressione) */
2  SELECT object_name = OBJECT_NAME(object_id), *
3  FROM sys.dm_db_column_store_row_group_physical_stats
4  WHERE object_id = OBJECT_ID('dbo.FactSales');
5
6  /* Spazio sintetico per oggetto (approssimazione via dm_db_partition_stats)
       */
7  SELECT
8    SUM(row_count)                                  AS rows_total,
9    SUM(reserved_page_count)*8.0/1024.0             AS reserved_mb,
10   SUM(used_page_count)*8.0/1024.0                 AS used_mb
11 FROM sys.dm_db_partition_stats
12 WHERE object_id = OBJECT_ID('dbo.SalesAgg');    -- vista indicizzata
     materializzata
13 GO
```

Listing 6.6: SQL Server - cenni: stato dei *rowgroup columnstore* e spazio per oggetto materializzato (esempio originale)

La pressione su spazio e I/O si rileva mediante DMV e procedure di sistema: l'occupazione effettiva delle viste e delle tabelle sottostanti si stima con viste dinamiche del *pool* dedicato e, quando opportuno, con *report* sintetici (ad esempio analoghi a *sp_spaceused* in ambienti SQL Server) per confrontare il costo di *storage* rispetto al beneficio prestazionale. In parallelo, per scenari ibridi o cenni a SQL Server, è utile osservare anche la qualità delle strutture *columnstore* (stato dei *rowgroup*, compressione) e l'uso degli indici nel tempo, così da cogliere effetti indiretti della materializzazione sul piano fisico (Microsoft, 2024b [46]; Vaisman & Zimányi, 2022 [1]).

Infine, l'impatto sui DML va esplicitato: si registrano *throughput* e latenza dei carichi nelle finestre di ingestione, correlando eventuali regressioni con l'aumento dell'*overhead* della vista e con il volume di aggiornamenti. L'insieme dei KPI (tempi, *overhead_ratio*, spazio e impatto DML) definisce un quadro di ROI operativo: il mantenimento della vista è conveniente quando il guadagno stabile in latenza eccede i costi di manutenzione e di *storage*, come previsto dalle linee guida ufficiali per il *tuning* delle *materialized view* e dagli strumenti nativi di diagnostica del motore (Microsoft, 2024a [41]; Microsoft, 2024b [46]; Vaisman & Zimányi, 2022 [1]).

6.2.3 Valutazione sperimentale su Transaction Processing Performance – Decision Support

La letteratura tecnica e la documentazione ufficiale riportano valutazioni impostate su sottoinsiemi del *benchmark* TPC-DS per rappresentare carichi decisionali con numerose *join* e aggregazioni. In tali studi vengono selezionate interrogazioni con raggruppamenti su dimensioni temporali e di prodotto, in coerenza con schemi a stella tipici dei *data warehouse*, così da massimizzare la confrontabilità tra misure di base ed interventi di ottimizzazione. L'adozione di TPC-DS consente di ancorare il disegno sperimentale a uno standard pubblico e riproducibile, con semantica di *business* realistica e ampia varietà di piani candidati (TPC, 2021)[34].

Per la *baseline* vengono eseguite più misurazioni in condizioni di cache non inizializzata (*cold cache*) e di *cache* inizializzata (*warm cache*), registrando media, mediana e varianza dei tempi di risposta e analizzando i piani stimati con EXPLAIN (Synapse) per individuare operatori dominanti e movimenti dati (*shuffle*/*broadcast*). Questa ispezione consente di isolare i colli di bottiglia e di identificare sotto-piani di *join*/aggregazione suscettibili a calcolo anticipato tramite viste materializzate, in linea con le indicazioni fornite dal produttore (Microsoft, 2024)[44].

Gli interventi descritti prevedono la definizione di una/due viste materializzate orientate a catturare aggregazioni riusate da più interrogazioni (ad esempio vendite per ProductID × MonthKey), con distribuzione HASH sulle chiavi di *join* ricorrenti allo scopo di ridurre il movimento dati. L'effettivo impiego delle viste è verificato

confrontando i piani stimati prima/dopo e accertando la riscrittura automatica dei sotto-piani verso l'oggetto materializzato, senza modificare il testo SQL applicativo (Microsoft, 2022–2023)[39]. Tale approccio è coerente con le linee guida che raccomandano di progettare la vista sulla congiunzione e sull'aggregazione condivise, richiedendo all'ottimizzatore il *matching* trasparente quando le condizioni semantiche e fisiche lo consentono (Vaisman & Zimányi, 2022)[1].

I risultati riportati mostrano, per le *query* bersaglio, una riduzione delle latenze e della variabilità, con benefici più marcati per granularità intermedie. In parallelo viene monitorato l'*overhead* incrementale delle viste tramite DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD e, al superamento di soglie operative, viene pianificato il REBUILD per ripristinare l'efficienza di scansione. Questa procedura lega la decisione di manutenzione a un indicatore nativo e documentato, mitigando il rischio di degrado progressivo (Microsoft, 2023)[38]. Nel complesso, i casi TPC-DS convergono sull'evidenza che la materializzazione mirata di aggregazioni condivise, con distribuzione coerente alle chiavi di *join*, produce un rapporto costo/beneficio favorevole quando i guadagni stabili in latenza superano i costi di manutenzione e di spazio, in accordo con le linee guida su *tuning* con viste materializzate e con l'uso sistematico di EXPLAIN per la validazione *ex ante* dei piani (Microsoft, 2022–2024 [49]; Vaisman & Zimányi, 2022 [1]).

Le scelte che massimizzano il beneficio dipendono dallo strato fisico delineato nel Capitolo 4 (distribuzione, partizionamento, *columnstore*) e applicano i criteri di selezione/manutenzione formalizzati nel Capitolo 5. I *benchmark* TPC-DS permettono una valutazione quantitativa degli effetti di tali scelte in condizioni operativi replicabili.

```
-- 1) BASELINE: profila query TPC-DS (es. Q64) e registra tempi
SET RESULT_SET_CACHING OFF;  -- evita riuso full-result nella baseline
EXPLAIN WITH_RECOMMENDATIONS
SELECT
  fs.ProductID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
JOIN dbo.DimDate   AS dd ON dd.DateKey = fs.DateKey
GROUP BY fs.ProductID, fs.MonthKey;
GO

-- Esecuzioni etichettate per analisi tempi (storico in sys.dm_pdw_exec_requests)
SELECT
  fs.ProductID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
JOIN dbo.DimDate   AS dd ON dd.DateKey = fs.DateKey
GROUP BY fs.ProductID, fs.MonthKey
OPTION (LABEL = 'TPCDS_Q64_BASE');
GO

-- 2) INTERVENTO: MV che cattura aggregazione condivisa; distribuzione HASH per co-locare join
```

```
CREATE MATERIALIZED VIEW dbo.mv_SalesByProdMonth
WITH (DISTRIBUTION = HASH(ProductID))
AS
SELECT
  fs.ProductID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
GROUP BY fs.ProductID, fs.MonthKey;
GO

-- 3) VALIDAZIONE: la query non cita la MV; ottimizzatore riscrive in automatico
EXPLAIN
SELECT
  fs.ProductID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
JOIN dbo.DimDate   AS dd ON dd.DateKey = fs.DateKey
GROUP BY fs.ProductID, fs.MonthKey;
GO

-- 4) MISURA POST-INTERVENTO: run etichettate per confronto tempi (media/mediana/varianza)
SELECT
  fs.ProductID, fs.MonthKey,
  SUM(fs.SalesAmount) AS Amt
FROM dbo.FactSales AS fs
JOIN dbo.DimDate   AS dd ON dd.DateKey = fs.DateKey
GROUP BY fs.ProductID, fs.MonthKey
OPTION (LABEL = 'TPCDS_Q64_AFTER_MV');
GO

-- 5) RACCOLTA KPI: tempi (avg/median/var) dalle esecuzioni etichettate
WITH runs AS (
  SELECT total_elapsed_time/1000.0 AS sec_elapsed
  FROM sys.dm_pdw_exec_requests
  WHERE label IN ('TPCDS_Q64_BASE','TPCDS_Q64_AFTER_MV')
    AND status = 'Completed'
)
SELECT
  AVG(sec_elapsed) AS avg_s,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY sec_elapsed) AS median_s,
  VARP(sec_elapsed) AS var_s
FROM runs;
GO

-- 6) OVERHEAD MV: monitoraggio e manutenzione (DBCC OVERHEAD + REBUILD)
DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD ('dbo.mv_SalesByProdMonth');
-- Se l'overhead_ratio supera la soglia operativa:
ALTER MATERIALIZED VIEW dbo.mv_SalesByProdMonth REBUILD;
GO

-- 7) CONFRONTO CON LA RESULT-SET CACHE (opzionale per evidenziare full-result vs rewrite)
SET RESULT_SET_CACHING ON;   -- abilitare per misure con cache attiva
-- Ripetere la SELECT etichettata e confrontare hit/miss e tempi rispetto alla MV.
```

Listing 6.7: Synapse - Valutazione TPC-DS: *baseline*, MV mirata, validazione con `EXPLAIN`, KPI temporali e *overhead* (esempio originale)

Capitolo 7

Conclusioni

L'obiettivo dell'elaborato si è concentrato sull'analisi e sull'ottimizzazione di sistemi di *Business Intelligence* tramite la progettazione fisica, l'ottimizzazione delle viste materializzate e delle *query*. La ricerca si è concentrata sull'esigenza di fornire soluzioni per le difficoltà riscontrate dai moderni sistemi di *Business Intelligence*. I risultati ottenuti dimostrano come l'utilizzo di soluzioni innovative permetta di superare i limiti presentati in letteratura e legati alla rigidità dei tradizionali sistemi di BI. Si può quindi confermare che si sono ottenuti i risultati desiderati, legati all'ottimizzazione dei tempi di risposta delle *query*, alla migliore qualità dei dati, ad una maggiore attualità dei dati e ad una maggiore efficienza in termini di utilizzo di risorse computazionali.

Si è inoltre dimostrato come una corretta comprensione delle differenze tra OLTP e OLAP sia fondamentale per garantire una progettazione capace di supportare le diverse esigenze delle aziende. Si è sottolineata la particolare importanza delle *pipeline* ETL, che permettono di trasformare dati grezzi in informazioni utili, analizzando l'utilità delle metriche strutturali per migliorare l'efficacia dei flussi di lavoro. Si è approfondito lo studio delle tecniche di ottimizzazione utilizzabili durante il processo ETL e di progettazione fisica del *data warehouse*. Si sono analizzati i vantaggi legati all'utilizzo congiunto di indicizzazione *bitmap*, partizionamento adattivo, compressione ed elementi di *caching*. Particolarmente approfondito è lo studio delle viste materializzate. La loro pianificazione ottimale, che prevede l'utilizzo di algoritmi complessi come A*, permette di bilanciare i costi computazionali con le prestazioni desiderate anche per carichi di lavoro molto differenti. L'approccio adottato ha posto il *focus* sull'empirismo, effettuando confronti di tecniche ed algoritmi su sistemi di *Business Intelligence* reali con diversi casi di studio già esistenti in letteratura, per confermare la robustezza dei risultati.

Mettendo in relazione i risultati conseguiti attraverso la ricerca nella letteratura internazionale, è possibile notare come i risultati ottenuti siano in linea con le tematiche di punta affrontate nel mondo accademico e professionale. L'analisi ha

evidenziato come il lavoro si allinei con le strategie di ottimizzazione dei sistemi di BI più diffuse, tra cui si rintraccia la fase predittiva e prescrittiva basata sull'ottimizzazione delle viste materializzate e le metodologie di *data governance*.

Si presentano ora i limiti dell'elaborato. In termini di generalizzabilità, i risultati ottenuti sono stati validati su uno spazio di problemi limitato. Si sono effettuati *test* su casi di studio e *dataset* già esistenti in letteratura. Ciò ha inevitabilmente portato a soluzioni parziali, adatte ai problemi affrontati in questi casi. Inoltre, alcune metodologie innovative, tra cui l'auto-ottimizzazione grazie ad algoritmi in grado di apprendere dall'esperienza, hanno bisogno di ulteriori valutazioni empiriche prima di essere completamente mature per poter essere considerate a livello industriale. Bisogna inoltre tenere in considerazione i vincoli legati all'architettura ed alle tecnologie utilizzate. È necessario valutare le infrastrutture preesistenti e la possibilità di migrazione dei sistemi *legacy* in evoluzione tecnologica. Il lavoro presentato è quindi valido ed utilizzabile finché i vincoli strutturali siano simili a quelli riscontrati durante lo studio. Non è inoltre possibile definire per quali vincoli futuri o evoluzioni tecnologiche il lavoro non sarà più valido, anche se la flessibilità è un aspetto importante di molte strategie.

Ci sono tuttavia anche ulteriori sviluppi futuri e possibili linee di ricerca. Particolarmente interessante è lo sviluppo di algoritmi di auto-adattamento sempre più complessi, con metodologie avanzate per la selezione e l'aggiornamento delle viste materializzate. Un altro sviluppo interessante è legato all'ottimizzazione delle *pipeline* ETL, tramite l'introduzione di strategie *self-adapting* per l'automatizzazione dei processi e per l'adattamento ai cambiamenti dello scenario di *business*. Gli strumenti di monitoraggio delle *performance* dei sistemi di *Business Intelligence* sono sicuramente un terreno di sviluppo. Inoltre, si pensa ad un approccio interdisciplinare che combini le tecnologie analizzate a discipline manageriali. L'obbiettivo è lo sviluppo di soluzioni che considerino l'efficienza operativa, la qualità delle informazioni, i costi implementativi ed i costi di gestione, nonché la sostenibilità dei sistemi sviluppati. Questo tipo di approccio risulta fondamentale per permettere ad un'azienda di trarre reali benefici dai sistemi analizzati ed implementati, ma anche per garantire il raggiungimento di *standard* elevati di innovazione.

Bibliografia

[1] A. Vaisman e E. Zimányi. *Data Warehouse Systems*. Springer, 2022.

[2] A. P. Amadeo. *Tecnologias Aplicadas Para Business Intelligence*. Facultad de Informática, Universidad Nacional de La Plata, 2018.

[3] W. H. Inmon. *Building the data warehouse*. John Wiley & Sons, Inc., 2002.

[4] H. Chen, R. H. L. Chiang e V. C. Storey. *Business intelligence and analytics: From big data to big impact*. MIS Quarterly, 2012.

[5] F. Gurcan et al. *Business intelligence strategies, best practices, and latest trends: Analysis of scientometric data from 2003 to 2023 using machine learning*. Sustainability, 2023.

[6] F. Gurcan et al. *Business Intelligence Strategies, Best Practices, and Latest Trends*. Sustainability, 2023.

[7] J. Reinschmidt e A. Francoise. *Business Intelligence Certification Guide*. IBM, 2000.

[8] R. Chirkova e J. Yang. *Materialized Views*. Foundations e Trends®in Databases, 2011.

[9] C. Renso e C. Gozzi. *Data warehousing e OLAP*. 1990.

[10] L. Pontieri. *Fondamenti di Data Warehousing e Data Mining*. ICAR-CNR, 1989.

[11] Z. El Akkaoui, A. Vaisman e E. Zimányi. *A Quality-based ETL Design Evaluation Framework*. Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019), 2019.

[12] S. M. F. Ali e R. Wrembel. *From conceptual design to performance optimization of ETL workflows: current state of research and open problems*. The VLDB Journal, 2017.

[13] V. Theodorou et al. *Quality measures for ETL processes: from goals to implementation*. Concurrency, Computation: Practice e Experience, 2016.

[14] V. Theodorou, A. Abello e W. Lehner. *Quality measures for ETL processes*. Data Warehousing e Knowledge Discovery: 16th International Conference, 2014.

[15] A. Polyvyanyy et al. *Process querying: Enabling business intelligence through query-based process analytics*. Decision Support Systems, 2017.

[16] K. H. Goud. *Optimizing ETL processes for big data applications*. International Journal of Engineering e Management Research, 2024.

[17] A. Boukorca et al. *Coupling materialized view selection to multi query optimization: Hyper graph approach*. International Journal of Data Warehousing e Mining, 2015.

[18] P. Alapaty e R. Rao. *Ottimizzazione delle query SQL di Spark e dei job AWS Glue Amazon EMR Spark*. Amazon Web Services, Inc., 2024.

[19] M. Golfarelli. *Ottimizzazione delle interrogazioni*. Alma Mater Studiorum - Università di Bologna, 2000.

[20] K. B. Goyal et al. *Indexing and compression in data warehouses*. Proceedings of the International Workshop on Design e Management of Data Warehouses, 1999.

[21] A. Khanna et al. *Proceedings of the Third International Conference on Trends in Computational and Cognitive Engineering*. Springer, 2021.

[22] T. Cerquitelli e P. Garza. *Viste materializzate in Oracle e SQL esteso*. Sistemi di gestione di basi di dati, 2023.

[23] W. J. Labio, D. Quass e B. Adelberg. *Physical Database Design for Data Warehouses*. Stanford University Department of Computer Science, 1996.

[24] V. Bhaja et al. *Intelligent Engineering Informatics*. Springer Nature, 2023.

[25] E. Stark, K. Ragan e S. Krishnasamy. *Uso delle viste materializzate in Amazon Redshift*. Amazon Web Services, Inc., 2022.

[26] H. Mistry et al. *Materialized view selection and maintenance using multi-query optimization*. arXiv.org, 2000.

[27] J. Goldstein e P.-Å. Larson. *Optimizing queries using materialized views: A practical, scalable solution*. ACM SIGMOD, 2001.

[28] P. Vassiliadis. *Data Warehouse Modeling and Quality Issues*. National Technical University of Athens, 2000.

[29] I. Mami e Z. Bellahsene. *A Survey of View Selection Methods*. SIGMOD Record, 2012.

[30] R. Zhao et al. *Learning to monitor machine health with convolutional bi-directional LSTM networks*. Sensors, 2017.

[31] M. A. Hussain, S. Vemaraju e S. Kochelakota. *Scientometric mapping of research trends and impact of artificial intelligence applications in banking and finance*. Journal of Information Systems Engineering e Management, 2025.

[32] H.-H. Tsai. *Research trends analysis by comparing data mining and customer relationship management through bibliometric methodology*. Scientometrics, 2011.

[33] Microsoft Corporation. *Best practices for dedicated SQL pool (Azure Synapse Analytics)*. Accessed: 2025-11-09. 2024–2025. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/best-practices-dedicated-sql-pool.

[34] Transaction Processing Performance Council. *TPC Benchmark DS (TPC-DS): Standard Specification*. Rapp. tecn. Accessed: 2025-11-09. TPC, 2021. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.1.0.pdf.

[35] Microsoft Corporation. *EXPLAIN (Transact-SQL) — dedicated SQL pool (Azure Synapse Analytics)*. Accessed: 2025-11-09. 2023–2024. URL: https://learn.microsoft.com/en-us/sql/t-sql/queries/explain-transact-sql?view=azure-sqldw-latest.

[36] Microsoft Corporation. *Performance tuning with materialized views — dedicated SQL pool*. Accessed: 2025-11-09. 2023–2025. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-materialized-views.

[37] Microsoft Corporation. *CREATE TABLE AS SELECT (CTAS) — dedicated SQL pool*. Accessed: 2025-11-09. 2025. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-develop-ctas.

[38] Microsoft Corporation. *CREATE MATERIALIZED VIEW AS SELECT — dedicated SQL pool*. Accessed: 2025-11-09. 2023. URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-materialized-view-as-select-transact-sql?view=azure-sqldw-latest.

[39] Microsoft Corporation. *Result set caching — dedicated SQL pool (Azure Synapse Analytics)*. Accessed: 2025-11-09. 2022–2023. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-result-set-caching.

[40] Microsoft Corporation. *Statistics in dedicated SQL pool (Azure Synapse Analytics)*. Accessed: 2025-11-09. 2022–2025. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/statistics.

[41] Microsoft Corporation. *EXPLAIN with recommendations — dedicated SQL pool*. Accessed: 2025-11-09. 2024. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/reference/explain.

[42] Microsoft Corporation. *Create Indexed Views — SQL Server*. Accessed: 2025-11-09. 2025. URL: https://learn.microsoft.com/en-us/sql/relational-databases/views/create-indexed-views?view=sql-server-ver17.

[43] Microsoft Corporation. *Table Hints (Transact-SQL) — NOEXPAND*. Accessed: 2025-11-09. 2025. URL: https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-table?view=sql-server-ver17#noexpand.

[44] Microsoft Corporation. *DBCC PDW_SHOWMATERIALIZEDVIEWOVERHEAD (Transact-SQL)*. Accessed: 2025-11-09. 2024. URL: https://learn.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-pdw-showmaterializedviewoverhead-transact-sql?view=azure-sqldw-latest.

[45] Microsoft Corporation. *Columnstore indexes — SQL Server*. Accessed: 2025-11-09. 2022. URL: https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview.

[46] Microsoft Corporation. *Temporary tables — dedicated SQL pool*. Accessed: 2025-11-09. 2024. URL: https://learn.microsoft.com/en-us/azure/synapse-analytics/sql/develop-tables-temporary.

[47] Microsoft Corporation. *CREATE MATERIALIZED VIEW AS SELECT (TransactSQL) — Azure Synapse Analytics (dedicated SQL pool)*. Accessed: 2025-11-09. 2023. URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-materialized-view-as-select-transact-sql?view=azure-sqldw-latest.

[48] Microsoft Corporation. *ALTER MATERIALIZED VIEW (Transact-SQL) — dedicated SQL pool*. Accessed: 2025-11-09. 2023. URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-materialized-view-transact-sql?view=azure-sqldw-latest.

[49] Microsoft Corporation. *CREATE MATERIALIZED VIEW AS SELECT — syntax and limitations*. Accessed: 2025-11-09. 2022–2024. URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-materialized-view-as-select-transact-sql.