Politecnico di Torino

Corso di Laurea Magistrale in

Ingegneria Gestionale

Tesi di Laurea Magistrale

Importanza degli indici nell'ottimizzazione delle prestazioni del Database Oracle

Relatrice:
Prof.ssa Tania Cerquitelli

Supervisori:
Dott. Davide Burdese
Dott.ssa Sonia Regis

Candidato:
Alessandro Viviano

Anno accademico 2023/2024

Sommario

Questa trattazione si propone di studiare l'importanza degli indici nel panorama dell'ottimizzazione delle prestazioni del database Oracle. Si offre una panoramica sui database relazionali (RDBMS), con un focus sull'architettura del **Database Oracle**, strumento centrale nello sviluppo di questo studio.

Vengono esplorati in dettaglio vari aspetti del Database Oracle, con un'attenzione particolare al concetto di piano di esecuzione, fondamentale per il processamento efficace delle query, e al Query Optimizer, un componente cruciale che determina il miglior piano di esecuzione per lo svolgimento delle query. Un'analisi approfondita dell'ottimizzatore include un'indagine su come venga stimato il suo costo e come questo valore si inserisca nel contesto della teoria delle scale di misurazione.

Successivamente vengono introdotti gli indici come strumenti per ottimizzare le prestazioni del database, con approfondimenti su frammentazione e rebuild. Il nucleo dell'elaborato approfondisce l'ottimizzazione delle prestazioni del Database Oracle attraverso l'utilizzo di indici B-Tree e indici clustered, supportato da una dimostrazione a livello di piano di esecuzione. Per raggiungere questo obiettivo, sono state interrogate alcune viste dinamiche fornite da Oracle all'interno dei suoi database.

Infine, si evincerà come dall'analisi di alcune metriche scelte a priori il miglioramento delle prestazioni sarà confermato e tirando le somme ci sarà spazio per eventuali spunti da poter approfondire a livello di contesto aziendale.

Indice

Capitolo 1

Introduzione

Nel contesto attuale, la rilevanza imprescindibile della raccolta e della gestione efficace dei dati, in particolare dei Big Data, emerge in modo inequivocabile. Questo concetto trova chiara conferma nelle intense battaglie che le aziende, operanti in una vasta gamma di settori, affrontano al fine di acquisire il maggior numero possibile di informazioni. Gli obiettivi di questa corsa ai dati variano dalla profilazione accurata dei clienti, alle decisioni strategiche di mercato, all'implementazione di campagne pubblicitarie, fino persino all'orientamento delle scelte di direzione aziendale.
La crescente importanza attribuita al tema dei dati si riflette nella dinamica incessante e nella robusta espansione del settore aziendale dedicato a identificare le modalità più efficaci per la loro gestione. Questo settore si trova così ad affrontare una continua evoluzione, alimentata dalla necessità di affrontare sfide sempre più complesse legate all'acquisizione, alla conservazione e all'analisi di masse sempre crescenti di informazioni. La presente tesi si propone di esplorare in dettaglio le dinamiche e le sfide di questo contesto in continua evoluzione, contribuendo a delineare le strategie ottimali per una gestione avanzata e proficua dei Big Data nell'ambito aziendale.

1.1 Contesto aziendale

L'azienda che ha fornito i mezzi per lo sviluppo e ospitato questa trattazione è Mediamente Consulting S.R.L., che si occupa nello specifico di gestire a tutti gli effetti ogni stato del dato attraverso l'utilizzo di strumenti che mette a disposizione Oracle, gigante della data analysis. Il fine di tale gestione è la realizzazione di infrastrutture tecnologiche che aiutano le aziende clienti a produrre decisioni strategiche di business. La sottoscritta trattazione è stata sviluppata nella business unit di infrastruttura informatica, che regge le architetture su cui lavorano tutte le altre unit dell'azienda. All'interno dell'azienda infrastrutture si occupa nello specifico di attività di gestione e manutenzione sui DBA e sugli strumenti promossi gravitanti attorno ad essi, come Oracle Cloud Infrastructure o WebLogic. Quindi, la divisione si occupa di installare i prodotti Oracle da remoto presso i clienti, ma soprattutto di gestire le problematiche in cui questi possono incorrere, nonché le attività ordinarie che secondo i vari contratti Mediamente si impegna a svolgere, come l'estensione di un tablespace di un database. Inoltre, gestendo tutte le problematiche relative all'infrastruttura informatica ci sono numerose situazioni in cui invece il lavoro si sposta più su attività di networking e perfino di manutenzione a livello di sistema operativo. Entrambi elementi chiave quando si ha a che fare con il funzionamento di un database.

1.2 Campi di studio

Nella suddetta trattazione l'obiettivo prefissato è quello di analizzare e studiare da quanti più punti di vista possibili il comportamento del database Oracle di fronte all'utilizzo e alla gestione degli indici sulle tabelle. Questi oggetti risultano essere estremamente importanti nel contesto del dimensionamento dell'impianto strutturale tabellare di qualsiasi base dati, ma non è semplice capire come creare le condizioni ottimali perché possano fornire il loro contributo nel modo più efficiente

possibile. Nel corso dell'elaborato si analizzeranno diversi aspetti caratterizzanti del topic mediante l'uso delle viste di sistema, la cui visualizzazione è fornita direttamente da Oracle nei suoi database. L'aspetto principale a cui sarà lasciato più spazio, sarà lo studio delle variazioni di alcuni parametri temporali rispetto ai cambiamenti di query e di numerosità, nonché un'eventuale digressione per comprendere a fondo i legami che intercorrono tra le variabili in gioco. Non solo, l'analisi sarà poi portata ad un livello ulteriormente dettagliato con lo studio di performance a partire direttamente dai piani di esecuzione, metodo con cui il database processa le query che chiedono di essere eseguite da parte del sistema stesso o degli utenti che si interfacciano al database.In collegamento alle prestazioni delle diverse condizioni di indicizzazione, non si potrà prescindere dal trattare anche le questioni di gestione e manutenzione degli indici, i quali sobbarcano il rischio concreto della frammentazione. Questo problema verrà trattato in modo approfondito per essere compreso al meglio e, in funzione dell'importanza che riveste nel quotidiano dell'azienda che ha ospitato la stesura, verrà ricercata una soluzione che possa essere utile ad una prima fase dell'approccio alla sua risoluzione.

1.3 Stato dell'arte

Dall'esperienza nel contesto della business unit di infrastruttura di Mediamente, si evince come, nonostante l'importanza che gli indici rivestono nella fase architetturale del database e nell'analisi di monitoraggio e prestazioni, non esista e non ci siano strumenti utili all'analisi specifica di questi oggetti, di come essi si comportano e di quali siano le migliori mosse da mettere in campo per aiutare l'ottimizzatore Oracle a farne l'uso migliore. Lo strumento che viene fornito direttamente da Oracle, utile al monitoraggio delle prestazioni e delle anomalie di funzionamento dei database, è l'Enterprise Manager. In sintesi, si tratta di un mezzo conferente un'interfaccia che può, attraverso gestione delle viste e comandi intuitivi, essere estremamente d'aiuto

ai consulenti che si occupano di gestire le problematiche dei database produttivi delle varie aziende. Nonostante sia un mezzo strutturato da molti punti vista, è pensato per conferire una visione generale e non permette di avere un focus sugli indici o sullo studio specifico dei piani di esecuzione che l'ottimizzatore prepara in vista della risoluzione di una query. Inoltre, non permette di avere una visualizzazione grafica chiara e intuitiva del problema della frammentazione della memoria logica del database.

1.4 Struttura dello studio

Questo studio si occuperà principalmente di svolgere un'analisi sui benefici dell'introduzione di un elemento fondamentale della corretta costruzione e implementazione delle tabelle, oggetto fondante dei database relazionali, come quello con cui si avrà a che fare. Al fine di portare a termine l'obiettivo bisognerà gradualmente introdurre e spiegare in modo approfondito gli strumenti utilizzati a partire proprio dal database generico per poi scendere nello specifico di come è costituito e come lavora un database Oracle, fino ad arrivare alla spiegazione di concetti come piano di esecuzione o ottimizzatore Oracle. Per svolgere questo compito sarà fondamentale un intelligente utilizzo delle viste di sistema Oracle, predefinite nei database. Vi sarà un'importante digressione indispensabile sull'argomento della frammentazione e l'implementazione di una sua possibile soluzione che possa risultare automatizzata ed utilizzabile per facilitare il lavoro a chi deve risolvere problemi simili.

Capitolo 2

Database Oracle

Una corretta introduzione al problema da trattare prevede obbligatoriamente un focus su alcuni elementi importanti che hanno permesso questo studio. Il primo tra questi è il Database Oracle.
Un database all'interno del contesto informatico consiste in una raccolta di informazioni archiviate come dati all'interno di un sistema ed il suo utilizzo è vitale per qualsiasi tipo di ambiente, in particolar modo quello aziendale. Nel contesto di azienda, infatti, si può trovare il database come struttura di memorizzazione dati per ogni tipo di incombenza: dalla profilazione dei clienti alla registrazione dei materiali all'interno di un magazzino.
Più in generale i database si dividono in due grandi gruppi:

- **Database relazionali**: altamente strutturati e supportanti il linguaggio SQL (Structured Query Language).
- **Database non relazionali**: che in molti casi non supportano il linguaggio sopra citato, ma quello denominato NoSQL che ne indica l'evidente diversità.

Per gli studi relativi a questa tesi è stato utilizzato un database relazionale [1] quindi qui di seguito ne verranno presentate le principali caratteristiche concentrandosi in particolar modo sulla versione 19.0.0 sviluppata da Oracle.

2.1 Panoramica sull'istanza del Database Oracle

Server Oracle è la denominazione corretta che viene utilizzata per indicare il sistema associato alla parola Database, utilizzata colloquialmente al posto della prima. Il Server Oracle, infatti, è composto da due strutture principali: Il database e l'istanza [2].

Il **database** comprende tutti i file fisici in cui vengono effettivamente memorizzati i dati: di conseguenza diversi tipi di file sono incaricati di memorizzare diversi tipi di dati. È composto da tre categorie di file diverse: file di controllo, file di ripristino e file di dati. In aggiunta per alcuni casi possono essere facoltativamente presenti anche i file di password. I file di controllo hanno il compito di tenere traccia di tutti i file di dati e il ripristino dei file stessi, inoltre sono utili per tenere traccia di alcune informazioni importanti per il corretto utilizzo del sistema, ad esempio i timestamps, le informazioni critiche o le informazioni di backup. Hanno altrettanta importanza i file di ripristino che conservano le informazioni delle modifiche e le relative informazioni cronologiche, necessarie nei casi in cui un utente debba rifare tutti o solo alcuni cambiamenti al database. Importante infine ricordare che un'istanza può aprire solo un database, mentre un database può essere aperto da più istanze (nel caso di RAC, database a più istanze).

L'**istanza** comprende una serie di strutture di memoria e processi in background che permettono di accedere ai file presenti nel database. L'istanza rappresenta l'interfaccia tra l'utente e i dati fisici raccolti nel database. Essa è composta da tre parti distinte che lavorano insieme per permettere l'archiviazione, la cancellazione e l'interrogazione dei dati. Si tratta di SGA (System Global Area), PGA (Private Global Area) e processi in background. In particolare, la SGA è una struttura di memoria condivisa temporanea che ha vita e durata dall'avvio dell'istanza al suo arresto.

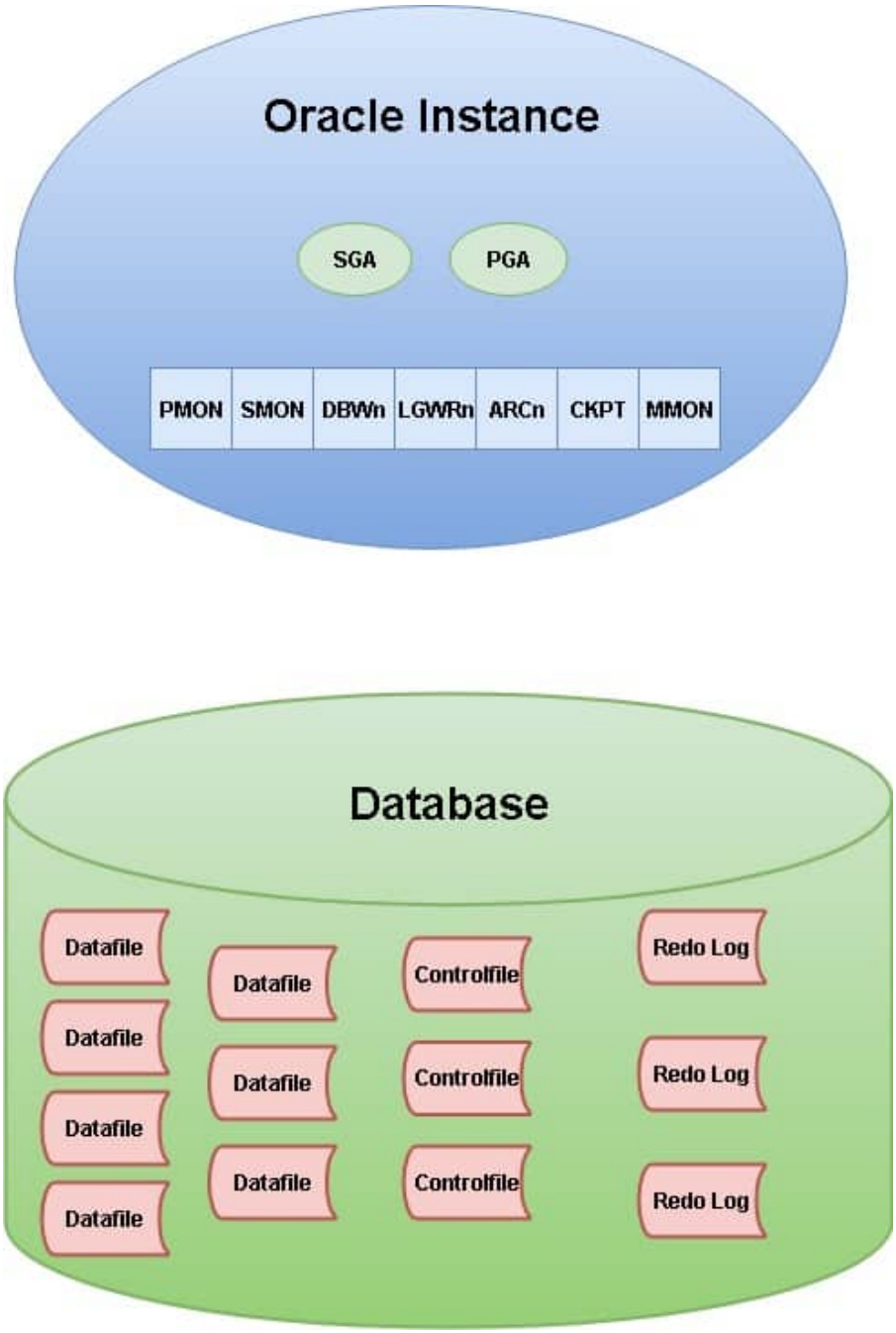

Figura 2.1. Separazione tra database e istanza.

2.2 Memorizzazione logica e fisica

Come in ogni sistema informatico complesso, anche nel database esistono due tipi distinti di memorizzazione: logica e fisica.
La memorizzazione fisica si riferisce all'effettiva locazione tecnica di immagazzinamento del dato effettivo, mentre quella logica serve essenzialmente a rendere più ottimizzato ed efficiente il modo migliore per raggiungere gli stessi nel momento in cui serve eliminarli, modificarli, aggiungerli oppure visualizzarli. Viene da sé come sia più dispendioso andare a reperire i dati dai file su disco tutte le volte che servono, piuttosto che limitarne l'effettivo accesso, e utilizzarne una rappresentazione logica per risparmiare risorse. Ai fini di questa trattazione è molto importante specificarne le differenze proprio per il suo orientamento all'ottimizzazione di processi, è per questo che l'organizzazione della memoria verrà approfondita.
L'**architettura fisica** si riferisce all'organizzazione e alla gestione dei dati scritti direttamente sul disco e si divide in tre tipi di file diversi: Data Files, Control files e Redo Log Files [3].

- **Datafiles**: costituiscono lo spazio di archiviazione principale per tabelle, indici e altri oggetti del database ed è importante ricordare che questi dati possono essere organizzati a loro volta in tabelle. Oracle utilizza una struttura a blocco per memorizzare i dati all'interno dei loro file.

- **Control Files**: includono informazioni cruciali sullo stato del database stesso, compresi di nomi e percorsi dei file di dati e dei file di redo log. Vengono utilizzati per garantire l'integrità del database e per ripristinarlo in caso di guasto.

- **Redo Log Files**: registrano le modifiche che vengono effettuate sui dati nel database e sono utilizzati per il recupero e la ripetizione delle operazioni di transazione in caso di guasto. In questo modo Oracle riesce a garantire la

durabilità delle transazioni prima che queste siano effettivamente applicate sui dati a disposizione nei Data Files.

L'**architettura logica** del database segue una linea gerarchica ed è organizzata secondo un impianto top down, in quanto i contenitori più capienti contengono contenitori sempre più piccoli fino ad arrivare al dato effettivo. Questo tipo di memorizzazione rende molto più semplice sia agli utenti che al sistema la ricerca e il conseguente accesso ai dati [4]. Partendo dall'alto si trovano i Tablespace, che contengono Segments, composti da Extents. Questi ultimi contengono i Blocks, dove risiedono effettivamente i dati, la loro dimensione può essere scelta dal DBA, DataBase Administrator, in fase di creazione dell'ambiente. Importante ricordare che al momento della creazione del database viene creato di default il Tablespace System, proprio dell'utente System, cioè del sistema. Si tratta di una sorta di utente con privilegi da superadmin.

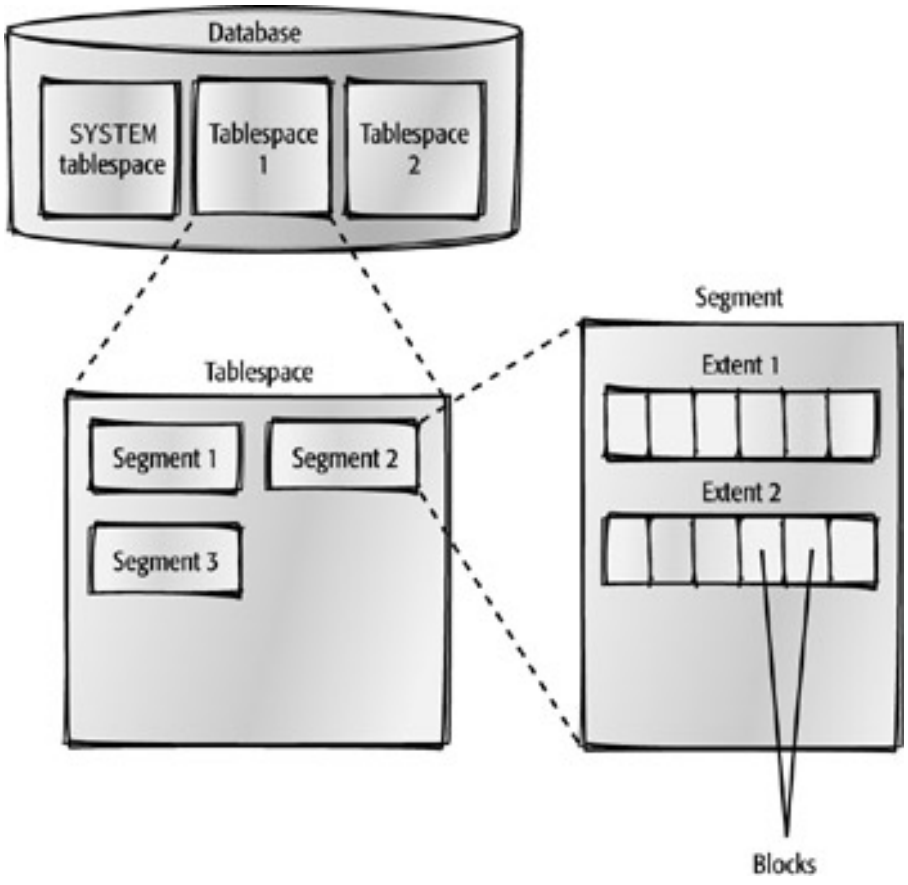

Figura 2.2. Struttura dell'architettura logica e collegamenti tra le parti.

2.3 Aree di memoria

All'interno dell'istanza, la memoria è divisa in due parti con compiti e metodi di funzionamento differenti: si tratta della System Global Area (SGA) e della Private Global Area (PGA).

La **System Global Area** rappresenta una regione di memoria condivisa da tutti i processi di una singola istanza del database, ciò significa che i dati che contiene sono accessibili a tutti gli utenti e gli oggetti attivi durante una sessione [4]. Infatti, come detto precedentemente, la SGA inizia la sua attività quando l'istanza viene attivata e la termina quando quest'ultima viene arrestata. Oltre a rendere il suo contenuto visibile a tutti, è altrettanto modificabile da tutti e ogni modifica è anch'essa visibile ad ogni processo. Al suo interno vi è una divisione che ne rende più semplice e strutturato il funzionamento. Vi si trovano infatti, il Redo Log Buffer, la Shared Pool e la cache, che saranno approfonditi in una sezione a parte.

La **Private Global Area** è una porzione di memoria privata assegnata ad ogni processo di server dedicato o processo in background. Dunque, a differenza della SGA che è condivisa, ogni processo singolo ha diritto alla sua area indipendente e protetta dagli altri processi [5]. Al suo interno stanno: la Sort Area, spazio dedicato alle operazioni di ordinamento, la Session Memory, memoria dedicata ad una sessione utente per vari scopi (dalla conservazione dei cursori a quella delle variabili di sessione), e le SQL Work Areas, spazi per eseguire le operazioni di ordinamento. Solamente il processo a cui è assegnata la PGA può accedere, modificare e visualizzare le modifiche alla sua area privata. Le differenze tra le due quindi, risultano molto chiare: mentre SGA è condivisa, PGA è privata: questa differenza è applicata al contempo ad accesso, modifica e visualizzazione di cambiamenti. Si può affermare che sono entrambe vitali in quanto, SGA permette il corretto funzionamento del sistema grazie alla condivisione dei dati cruciali, PGA permette l'isolamento e la gestione dei dati specifici per un singolo processo.

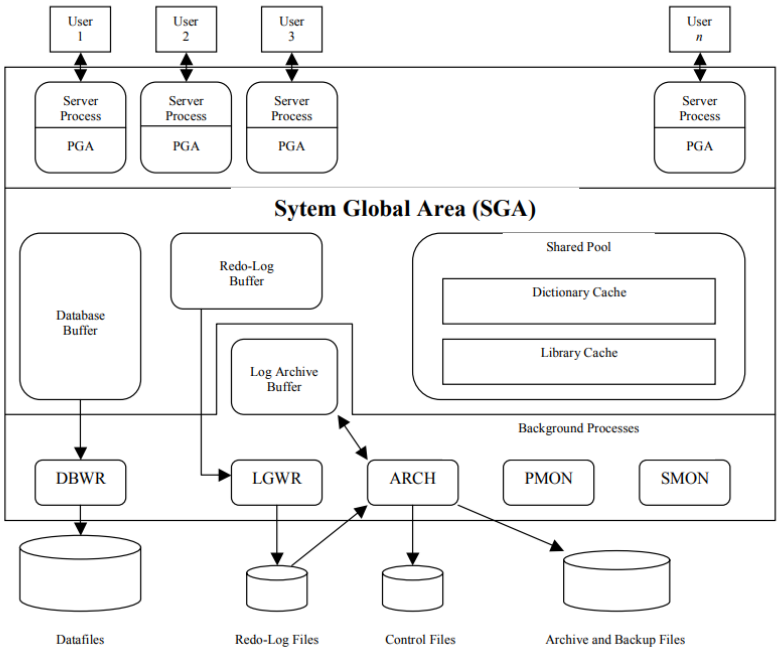

Figura 2.3. Struttura del server Oracle con focus sulla divisione di memoria tra SGA e PGA.

2.4 Processi in background

I processi che fanno parte di questa categoria sono essenziali per il corretto funzionamento del database [5]. Il sistema inizializza tali processi ad ogni avvio di un'istanza. Ognuno di essi ha uno scopo specifico e svolge diversi compiti al fine di garantire l'integrità, la disponibilità e le prestazioni del database. Le categorie che raccolgono i più comuni sono:

- **System Monitor (SMON)**: gestiscono l'integrità del database effettuando la recovery delle transazioni non completate in caso di crash del sistema ed eseguendo la pulizia degli oggetti temporanei e degli spazi non utilizzati.

- **Process Monitor (PMON)**: gestiscono l'integrità e la coerenza dei processi

dell'istanza rilevando e gestendo le connessioni dei client che si interrompono in modo anomalo e rilasciando risorse allocate da un processo client che termina in modo anomalo.

- **Archiver (ARCH)**: copiano su file esterni al fine di archiviare i redo log in modo che possano essere utilizzati in caso di guasto.
- **Checkpoint (CKPT)**: riducono il tempo di recupero in caso di crash del sistema scrivendo i dati modificati su disco e riducendone la quantità da recuperare in situazione emergenziale.
- **Log Writer (LGWR)**: scrivono i redo log sul disco per garantirne la persistenza utilizzando la SGA.
- **Database Writer (DBWn)**: scrivono i blocchi di dati modificati dalla SGA su disco nei datafile del database garantendo la persistenza e l'aggiornamento di essi.

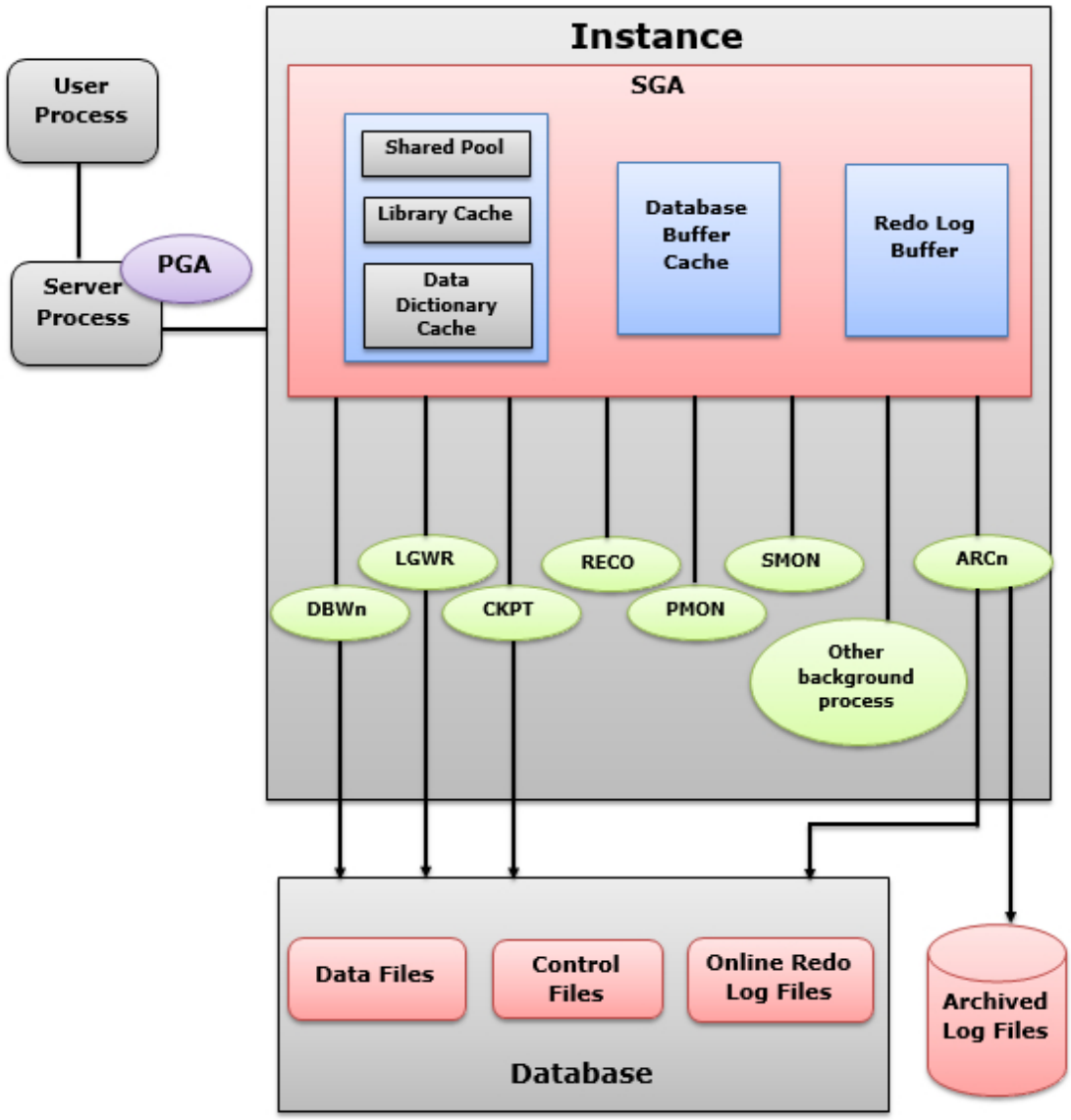

Figura 2.4. Processi in background appena citati con collegamento alla relativa area di memoria con la quale vi è comunicazione.

2.5 Cache

La cache del database Oracle merita uno spazio a sé per via della sua estrema importanza, sia per il normale funzionamento che per l'ottimizzazione delle prestazioni del database relazionale [6]. Esattamente come altre parti costituenti del database, anche la cache è divisa in aree, che non comunicano tra loro e che hanno compiti e ruoli diversi. Le aree principali sono cinque:

- **Buffer Cache**: certamente una delle più importanti, conserva al suo interno blocchi di dati che sono stati precedentemente letti da disco. Il suo funzionamento è semplice, quando un'istruzione SQL richiede un blocco di dati, il sistema controlla nella buffer cache se questi sono già presenti per restituirli immediatamente, se non fosse così, verrebbero letti da disco e registrati nella buffer cache per possibili utilizzi futuri.

- **Shared Pool**: è responsabile della memorizzazione di informazioni condivise, come definizione di query, definizione di piani di esecuzione, di pacchetti PL/SQL e altri metadati. In generale, il suo ruolo è importante perché alleggerisce il parser e l'optimizer del database, fornendo uno strumento di immagazzinamento di istruzioni SQL già elaborate ed utilizzate.

- **Redo Log Buffer**: viene utilizzato per la memorizzazione temporanea delle informazioni di redo log prima che vengano effettivamente scritte sui redo log files. Come tutto ciò che riguarda la questione redo log, il buffer è importante al fine della garanzia della durabilità del database. La sua presenza garantisce che le modifiche da apportare al database siano salvate anche in caso di guasto anteriore all'effettiva scrittura sui file corrispondenti.

- **Large Pool**: è usata per l'allocazione della memoria a favore di operazioni di grandi dimensioni come backup, ripristino, replica e altre attività di notevoli

dimensioni del sistema. Quest'area può essere configurata a parte rispetto alla shared pool.

- **Java Pool**: specificamente introdotta per la memorizzazione degli oggetti Java, come piani di esecuzione, classi o metadati relativi a Java.

La gestione del database Oracle è dinamica e molti dei parametri di costituzione delle aree sopra descritte, come le dimensioni, possono essere configurate proprio dagli utenti. Il corretto funzionamento della cache può aumentare di molto le prestazioni che restituisce il database.

2.6 Viste statiche e dinamiche

Le viste di sistema in Oracle sono oggetti speciali e predefiniti in ogni database che forniscono accesso a informazioni dettagliate sullo stato e sulla struttura del database stesso. Esse consentono agli utenti l'accesso a metadati, statistiche e altre informazioni sul software senza essere costretti ad accedere direttamente alle tabelle di sistema o ai cataloghi del database. Sono estremamente utili nel facilitare le operazioni di monitoraggio e ottimizzazione delle prestazioni. Le viste sono disponibili all'interno dello schema SYS, insieme di dati e caratteristiche speciali usato per mantenere oggetti di sistema, il cui accesso è possibile dagli utenti in possesso delle grants (permessi) necessarie concesse dall'utente amministratore.
Importante ricordare che, se il piano di acquisto del database Oracle lo prevede, il database viene fornito di uno strumento aggiuntivo denominato Enterprise Manager, che si occupa di fornire una comoda interfaccia per facilitare l'accesso alle informazioni contenute nelle viste in questione, rendendo molto più facile l'attività di monitoraggio e miglioramento delle prestazioni [7].
Le viste di sistema si dividono in due categorie, le viste statiche o DBA, e le viste dinamiche o V$:

- **Viste statiche**: il loro risultato viene determinato al tempo della loro definizione e rimane costante finché non vengono ricreate, ciò deriva dal fatto che la query alla base viene eseguita e il risultato viene memorizzato nella definizione della vista. Data la staticità dei dati quindi, le tabelle richiedono un aggiornamento manuale, se vi sono dei cambiamenti risulta necessario ricreare la vista per sincronizzare i dati. A livello di prestazioni possono offrire più possibilità in determinati casi, in quanto i risultati sono precalcolati e memorizzati.

- **Viste dinamiche**: il loro risultato viene determinato al momento dell'esecuzione, quindi, la query che sta alla base viene eseguita ogni volta che la vista viene visualizzata. La loro principale caratteristica è la dinamicità dei dati contenuti perché i suoi risultati rispecchiano lo stato corrente dei dati in questione. Ovviamente esse non richiedono un aggiornamento manuale. A tutto ciò consegue un'estrema flessibilità e capacità di adattamento alle esigenze momentanee.

Le prime vengono utilizzate principalmente quando di desidera preservare un insieme specifico di dati in uno specifico momento e non si immagina di doverli modificare frequentemente. Le seconde sono scelte quando si desidera un accesso sempre aggiornato ai dati di base senza doversi preoccupare di eventuali aggiornamenti manuali. Chiaramente la principale differenza sta nella persistenza dei dati, nel caso delle statiche vi è un mantenimento di una copia dei dati durante la creazione, nell'altro invece l'aggiornamento è pressoché continuo. Da questo consegue l'altra grande differenza che comporta il bisogno di un aggiornamento manuale alle viste statiche e la presenza di un aggiornamento automatico alle viste dinamiche.

Capitolo 3

Ottimizzatore

L'ottimizzatore è un componente molto importante dell'architettura del database Oracle. Il suo compito è quello di trovare la strada più efficace, secondo le sue regole di ottimizzazione, per giungere alla soluzione di una istruzione SQL che viene eseguita. Ogni query eseguita dall'utente o dal sistema scatena l'implementazione di un processo che porta l'ottimizzatore a decretare il migliore tra i piani di esecuzione prodotti, permettendo di restituire il risultato richiesto [8].
Il piano di esecuzione è una divisione in task della query da eseguire. Tale divisione, attraverso un'attenta analisi del testo della query, permette di dividere la grande operazione in operazioni minori in modo che il compito risulti diviso, organizzato e strutturato. Seguirà una spiegazione specifica di cosa sia e dei motivi per cui risulta importante ai fini di questa trattazione.
Il lavoro dell'ottimizzatore ha un costo per il sistema, poiché rappresenta inevitabilmente un dispendio di risorse. Quindi, dopo una spiegazione dei suoi componenti e del suo funzionamento, verrà trattato il tema del costo del piano di esecuzione per l'ottimizzatore e di come il database riesca a stimarlo.

3.1 Piano di esecuzione

In generale, quando il sistema recepisce una query da eseguire, esso deve muoversi al fine di determinare un modo efficiente per accedere ed elaborare i dati [9]. L'analisi di questa situazione è un compito assegnato all'ottimizzatore in tutti i principali sistemi di gestione dei dati, compreso Oracle. Concretamente, un piano di esecuzione consiste nella definizione della sequenza di accesso alle tabelle d'origine con metodi utilizzati per l'estrazione e dall'esecuzione di calcoli e metodi usati per filtrare, aggregare e ordinare. La sequenza di accesso alle tabelle può cambiare con il numero di tabelle interessate, ad esempio, se l'istruzione SELECT riguarda tre tabelle, l'accesso può essere effettuato secondo tutte le sequenze possibili a cui possono partecipare le tabelle. A proposito dell'estrazione dei dati dalle tabelle, argomento necessario per questa trattazione, l'ottimizzatore potrebbe utilizzare o meno gli indici, se presenti, che permettono di non scorrere le intere tabelle. La decisione su come svolgere in modo più efficiente calcoli, aggregazioni o filtraggi viene stimolata dalla presenza nella query di partenza di clausole come GROUP BY, ORDER BY, WHERE, HAVING o funzioni di aggregazione.

Nell'immagine è mostrato un esempio di piano di esecuzione scelto dall'ottimizzatore per la risoluzione di una query composta da una selezione di una JOIN di una tabella con sé stessa sotto una condizione di WHERE con un raggruppamento attraverso una GROUP BY. Questo metodo di visualizzazione è proprio dell'applicazione Oracle SQLDeveloper, ma è corretto ricordare che su ogni client con connessione a un database è possibile, in modi diversi, reperire il piano di esecuzione della query corrente.
Lo schema segue una struttura gerarchica e la posizione ricoperta dall'operazione rispecchia la sua stessa posizione nella sequenza di quelle da svolgere, in particolare, si nota come l'accesso ai predicati tramite il filtraggio e la risoluzione delle due condizioni di JOIN vengano fatte insieme, essendo allo stesso livello. I passi seguiti

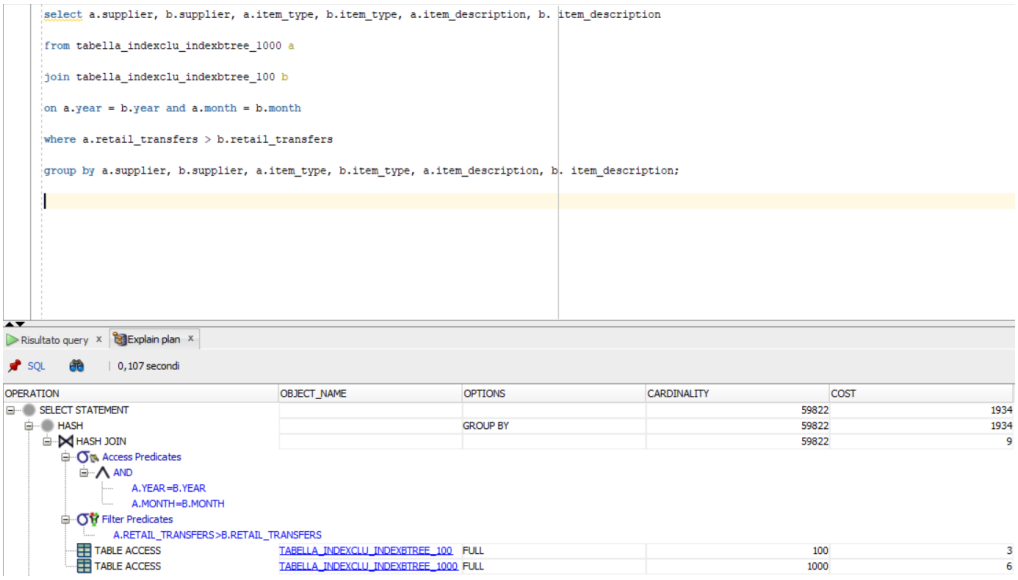

Figura 3.1. Esempio di piano di esecuzione in SQLDeveloper.

dall'esempio sono i seguenti:

- **TABLE ACCESS**: l'operazione riguarda l'accesso alle due tabelle e viene fatta in Full Scan, cioè, vengono scandite le intere tabelle.
- **ACCESS PREDICATES**: la dicitura segnala quali sono i predicati con cui vengono effettuate le operazioni di filtraggio delle righe e quella di join, in particolare le righe vengono filtrate con la disuguaglianza tra il parametro RETAIL_TRANSFERS delle due tabelle, mentre il join è realizzato su YEAR e MONTH.
- **HASH JOIN**: il task indica l'effettiva realizzazione del join eseguita con il metodo di hash.
- **HASH**: l'ultimo passo prima della restituzione del risultato è il raggruppamento della GROUP BY effettuato anch'esso con il metodo di HASH.
- **SELECT STATEMENT**: quest'operazione di selezione corrisponde all'effettiva stampa dei risultati in modo che siano fruibili all'utente.

3.2 Struttura dell'ottimizzatore

Dal momento in cui viene eseguito un insieme di istruzioni SQL, l'ottimizzatore si muove effettuando azioni che si possono raccogliere in alcune categorie: genera dei piani di esecuzione potenziali in base alle informazioni in suo possesso, stima il costo di ciascun piano in base allo sfruttamento di risorse necessario con lo scopo di minimizzarlo, confronta i piani ed infine sceglie il più conveniente [10].
Per svolgere questo tipo di attività ha bisogno di avere un'architettura strutturata che possa permettere la divisione dei compiti nelle sue componenti, da ciò viene la divisione dell'ottimizzatore in tre componenti principali. Il primo è il **Query Transformer**, un componente che si occupa di ricevere il testo della query così come è stata eseguita e valutare se riscrivendola in modi equivalenti viene reso più semplice il lavoro ai componenti successivi o meno. Riscrivere la query in modo equivalente significa solamente cambiare le parole chiave in istruzioni che possano ottimizzare i processi in questione, ad esempio il cambio di un OR con un UNION ALL. Le principali tecniche di trasformazione utilizzate sono:

- **View Merging**: l'ottimizzatore espande la definizione di una vista e la incorpora direttamente nella query principale, al posto di eseguire la query sulla vista separatamente, questo può far diminuire il numero di join necessari.
- **Predicate Pushing**: consiste nello spingere i predicati di filtraggio ad essere analizzati per primi, in modo da diminuire maggiormente i dati da scandire da un'operazione alla successiva.
- **Subquery Unnesting**: implica il trasformare le subquery in join o forme equivalenti più semplici da analizzare.
- **Query Rewrite with Materialized Views**: viene sfruttata la presenza di viste materializzate che possono soddisfare parte della query per riscrivere la query stessa di modo che le utilizzi.

Il secondo è l'**Estimator**, che si occupa di stimare il costo delle possibili strade da intraprendere per dare la possibilità di scelta al componente successivo. I parametri di misura che vengono sfruttati dallo stimatore sono:

- **Selectivity**: rappresenta la capacità di un predicato di selezionare quantità di dati, corrisponde quindi a una frazione di righe sulla totalità di righe presenti in tabella. Può variare da 0 a 1, in relazione al fatto che 0 indica la selezione di nessuna riga e 1 la selezione dell'intera tabella.

- **Cardinality**: rappresenta il volume di dati effettivamente estratto.

- **Cost**: rappresenta la quantità di unità di lavoro o di risorse utilizzate per l'operazione. Le euristiche che lo stimatore prende in considerazione sono l'I/O del disco, l'utilizzo di CPU, l'utilizzo di memoria o il tempo impiegato.

Infine, vi è il **Plan Generator** che si occupa di generare i possibili piani di esecuzione per raggiungere il risultato richiesto dalla query, confrontarne i costi e scegliere quello a costo minore. Importante ricordare in questo senso che qualsiasi vista o subquery presente nell'istruzione da eseguire viene trattata da questo componente con un sottopiano ad hoc.
La spiegazione specifica dei componenti potrebbe indurre a pensare che il lavoro sia approcciato a stadi successivi, in realtà non è così, infatti, le tre parti lavorano insieme e sono in interazione di continuo.
Per il suo corretto funzionamento, è indispensabile che le informazioni di cui necessità siano aggiornate di continuo. Tra le informazioni che sfrutta si possono ricordare le statistiche relative alle tabelle da scandire, le statistiche relative agli indici apposti su queste e gli eventuali hints (suggerimenti) forniti dagli utenti nelle query con la relativa sintassi.

L'ottimizzatore utilizza di default la Cost-Based Optimization, ma è possibile costringere l'ottimizzatore a utilizzare un approccio Rule-Based [11]. Il primo

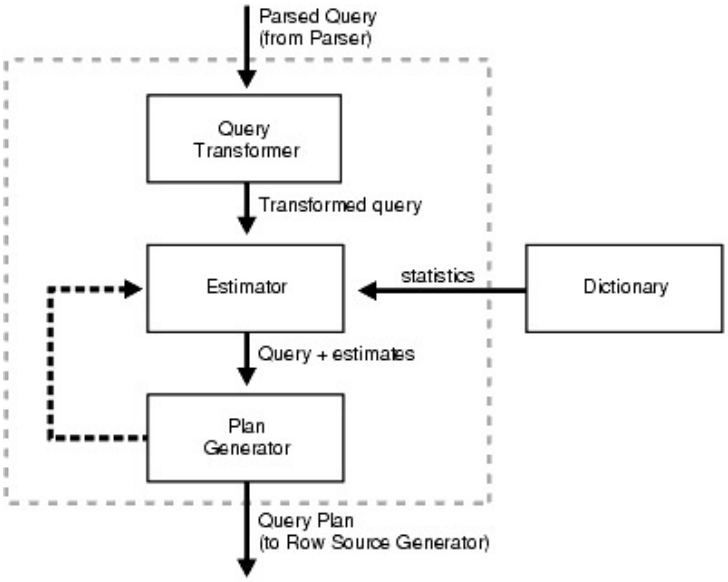

Figura 3.2. Componenti dell'ottimizzatore.

è consigliato in quanto permette all'ottimizzatore di basarsi su tutte le informazioni già descritte, aumentando le prestazioni e diminuendo gli sprechi di risorse. Il secondo si basa su regole fisse, intima l'ottimizzatore a basarsi principalmente sull'analisi della sintassi della query da eseguire lasciando quasi del tutto da parte la considerazione dei costi delle operazioni da eseguire. Questa seconda modalità è considerata obsoleta dalla versione 10g, ma rimane tutt'ora possibile cercare di influenzare il comportamento dell'ottimizzatore in vari modi, tra cui la seguente query: "ALTER SESSION SET OPTIMIZER_MODE = RULE/ALL_ROWS" . ALL_ROWS indica la modalità basata sui costi ed è in ogni caso sconsigliato procedere a un cambio per non incorrere in un deterioramento delle prestazioni non controllato. In ultima analisi, si può dire che la Rule-Based Optimization è preferibile in casi in cui mantenere le statistiche risulta troppo oneroso e i piani di esecuzione sono strettamente molto semplici. Introdotto il discorso dei costi, nel seguente paragrafo si tratterà riguardo al costo che il sistema stima relativamente alle operazioni svolte dall'ottimizzatore.

3.3 Optimizer cost

È interessante soffermarsi sul concetto di costo dell'ottimizzatore. Nell'ambito delle metriche che è possibile valutare tra le varie statistiche che mettono a disposizione le viste di sistema, è possibile valutare con delle misure quasi tutti i parametri principali che sono interessati durante il lavoro del database. La maggior parte di questi parametri può sfruttare l'esistenza di una unità di misura, ad esempio i tempi vengono stimati in microsecondi o la memoria viene stimata in bytes. Nel caso delle risorse che vengono spese sull'attività dell'ottimizzatore esiste solamente un numero puro, sprovvisto di unità di misura. Il valore associato al dispendio di risorse legate al lavoro dell'ottimizzatore su una determinata query è visualizzabile estrapolando le statistiche relative alla query stessa dalla vista dinamica V$SQL. Al suo interno si può notare la presenza di un campo denominato OPTIMIZER_COST. La scala utilizzata dal sistema relativamente a questo caso è una scala di tipo ordinale [12]. All'interno del panorama della teoria delle scale di misurazione, la scala ordinale permette di categorizzare i dati a disposizione, tuttavia le proprietà delle categorie che ne derivano non sono note. Dunque, quello che permette questo tipo di scala è di capire sicuramente se una query comporta un costo dell'ottimizzatore maggiore di un'altra, ma rimane sconosciuta l'entità della differenza tra i due costi. Ciò deriva dal fatto che non si sa che cosa comporti la differenza di un'unità di costo dell'ottimizzatore tra una misura e l'altra. In generale, infatti, non è possibile all'utente conoscere nello specifico le logiche con cui l'ottimizzatore prende le sue scelte. Questo parametro è stimato ed è il risultato dell'applicazione di un modello di costo che l'ottimizzatore utilizza e per cui è stato programmato, ma ogni ottimizzatore viene programmato dall'azienda che lo produce e il vantaggio che può derivare da prestazioni migliori dell'ottimizzatore, può risultare un importante vantaggio sui competitors.

Il modello di costo implementato da Oracle segue le regole basilari di qualsiasi altro

calcolo di effort all'interno di un ambiente tecnologico. Vengono presi in considerazione dei parametri che possono costare lavoro e risorse al sistema e ad ognuno viene associato un coefficiente in base al quale verrà stimato il suo costo in relazione alla totalità dell'operazione [10]. Dunque, l'ottimizzatore, dopo avere raccolte tutte le informazioni e statistiche possibili sugli oggetti con cui dovrà lavorare al fine di restituire il risultato, analizza alcuni centri di costo:

- **Costo degli operatori di accesso.**
- **Costo dei metodi di join.**
- **Costo delle operazioni di ordinamento e raggruppamento.**
- **Costo delle operazioni di I/O.**
- **Costo delle operazioni di trasmissione dei dati.**

Questi costi vengono assegnati direttamente dall'ottimizzatore dopo aver valutato tutte le statistiche in suo possesso e, dopo aver creato un prospetto di costo per ogni alternativa che lui stesso ha ipotizzato, riassume tutto il modello nel numero che Oracle definisce OPTIMIZER COST. Il minore tra quelli presenti indica il piano di esecuzione scelto. Le stime di costo di base che prescindono dall'effettiva query da eseguire possono essere note a priori, sono raccolte in alcune tabelle di sistema e sono scritte in fase di creazione dagli sviluppatori. Importante ricordare che ogni utente del database, possedente le grants necessarie, può riconfigurare le metriche in base alle quale vengono effettuate le stime.

Capitolo 4

Indici

Gli indici sono oggetti estremamente importanti nel contesto del database Oracle. Vengono utilizzati da tutti i principali gestori dei dati, infatti, la loro presenza e la corretta manutenzione è un evidente sintomo del benessere di un DBMS [13].
Concretamente un indice rappresenta una struttura applicata a una o più tabelle con lo scopo di permettere la veloce individuazione del dato che si cerca, evitando di dover scorrere interamente la tabella ogni volta che ci si approccia ad una ricerca. Un indice ben bilanciato e dimensionato intima l'ottimizzatore a utilizzarlo per le operazioni di ricerca all'interno del piano di esecuzione. Dunque, in questa trattazione si discorrerà a proposito della differenza esistente tra una Table Full Scan ,operazione che tipicamente scorre la totalità della tabella per raggiungere il dato o i dati desiderati, e una Table Index Scan, operazione che invece permette di non scorrere tutta la tabella, ma basandosi sull'indice, di arrivare solamente al punto che serve. Quasi superfluo citare il risparmio di risorse principalmente temporali e di memoria che questa differenza comporta. Un ulteriore aspetto di non poco conto, è che ciò che si guadagna in termini di risorse in ambito di ricerca, si perde in parte rispetto a operazioni di aggiornamento. Una buona struttura di indici permette di guadagnare sicuramente in fatto di ricerca e di minimizzare le perdite sugli altri tipi di operazioni.

4.1 Classificazioni

Gli indici possono essere classificati per categorie e in questa trattazione verranno usati due metodi principali, uno più generico per dare una panoramica e uno più specifico per entrare nel merito della spiegazione di quelli usati per la sperimentazione.
In generale gli indici di ogni tipo di database si dividono in due macro categorie [13]:

- **Indice Clustered:** determina l'ordine fisico dei dati all'interno di una tabella, ciò vuol dire che, quando un dato nuovo viene inserito, assume la posizione fisica ed effettiva regolata da questo indice. La relazione che esiste tra indice clustered e tabella è di 1:1, in quanto una tabella può avere un solo indice clustered e un indice clustered si può riferire ad una e una sola tabella. Viene da sé, che tutte le query aventi all'interno il campo su cui domina la chiave dell'indice clustered godono di ottime prestazioni, grazie al fatto che beneficiano dell'ordinamento fisico. Per quasi tutti i tipi di database, compreso quello Oracle, l'indice clustered è rappresentato dalla Primary Key, che solitamente è già individuata in fase di creazione di tabella e questo fa sì, che venga sempre creato già di default un indice del tipo appena descritto.

- **Indice Non-Clustered:** non altera l'ordinamento fisico dei dati nella tabella, ma crea una struttura dati separata, per mantenere i riferimenti alle righe della tabella in base all'indice. Grazie alla natura logica di quest'indice è possibile la coesistenza di più indici non-clustered in una stessa tabella a differenza dei precedenti. Si può affermare che le prestazioni delle query che sfruttano gli indici non-clustered tendono a migliorare in modo più contenuto di quelle che riguardano i clustered, in via definitiva si può ripetere che una buona struttura combinante i due tipi di indici possa essere la via migliore in fatto di prestazioni.

Parlando nello specifico della situazione del database Oracle è opportuno citare un nuovo tipo di classificazione [14]:

- **Indice B-Tree (Balanced Tree):** è il tipo di indice che più comunemente viene utilizzato, soprattutto perché corrisponde all'indice non-clustered, che Oracle crea di default quando gli viene richiesto. Si tratta di una struttura ad albero bilanciato dove ogni nodo può avere un numero variabile di chiavi (il bilanciamento permette che la profondità dell'albero rimanga ragionevole promuovendo l'ottimizzazione delle prestazioni). Lo sfruttamento del B-Tree è ottimale quando sono interessate query di intervallo ("maggiore di", "minore di", "tra"), laddove sono importanti le clausole di WHERE. Infine, è da ricordare che l'uso di questo tipo di indice risulta non ottimale nei casi in cui la tabella di riferimento è soggetta a frequenti INSERT, UPDATE e DELETE perché ciò costringe il sistema a un ribilanciamento dell'indice.

- **Indice Bitmap:** questo tipo di indice è indicato per migliorare le prestazioni delle query che operano su colonne con pochi valori distinti e soprattutto univoci, per esempio colonne booleane o rappresentanti categorie. Ogni valore distinto presente nella colonna è associato a un bitmap di bit che rappresentano la presenza o l'assenza del valore corrispondente per ogni riga della tabella (nello specifico, 1 se è presente e 0 se è assente). La sua ottimizzazione avviene quando viene usato in query che contengono operatori come AND, OR o NOT e quando contengono «nome-categoria» = "...". Come nel caso dei B-Tree, anche per questi indici non è consigliabile l'uso su tabelle soggette in modo frequente ad aggiornamenti. Inoltre, per causa del fatto che ogni bit nel bitmap rappresenta un valore distinto contenuto all'interno della colonna, essa non può contenere troppi valori univoci, altrimenti si rischia il deterioramento delle prestazioni.

- **Indice Funzionale:** è basato su una funzione o su un'espressione, e risulta

utile quando si desidera indicizzare i risultati di una funzione anziché i valori effettivi delle colonne. Un suo particolare vantaggio è il risparmio di CPU dato dal fatto che l'elaborazione della funzione avviene una sola volta alla creazione dell'indice, invece che tutte le volte in cui viene lanciata la query. È molto importante che l'espressione che produce la funzione sia deterministica, cioè, produca sempre lo stesso risultato. Anche per questo tipo di indice vi è la limitazione di utilizzo nel caso di tabelle soggette a numerosi aggiornamenti.

- **Indice Full-Text:** è pensato per la ricerca avanzata di stringhe o sotto-stringhe all'interno di grandi quantità di testo in specifici campi testuali di tabelle. Il loro funzionamento prevede prima di tutto una tokenizzazione, un accorpamento del testo con eliminazione dei caratteri speciali compresa la punteggiatura, poi vi è l'indicizzazione, che associa ad ogni termine la sua posizione all'interno del contesto di partenza, infine è possibile procedere all'effettiva ricerca.

4.2 Fragmentation

La frammentazione è un concetto molto comune in campo informatico e serve ad indicare il fenomeno secondo il quale avviene la suddivisione o dispersione fisica e logica di dati, risorse o informazioni composti da unità separate [14]. Questo causa in quasi tutti in casi una mancanza di coesione e continuità tra le parti. Gli ambiti dell'informatica in cui può verificarsi una situazione simile sono vari, qui di seguito ne verranno presentati alcuni:

- **Frammentazione del disco rigido**: si verifica quando i dati sono distribuiti sul disco in modo discontinuo, questo può essere causato dalle continue eliminazioni ed aggiunte di file nel tempo. Rappresenta un problema in quanto il disco deve effettuare più letture a causa delle posizioni fisiche diverse dei dati

con conseguente verificarsi di un significativo deterioramento delle prestazioni nel disco stesso.

- **Frammentazione della memoria**: si verifica quando la memoria è organizzata da un computer in strutture a blocchi contigui e risulta non semplice al sistema riuscire a registrare un dato o un processo in blocchi vicini tra loro. Questo tipo di contesto può risultare problematico per il deterioramento delle prestazioni e per l'occupazione di spazio inutile in memoria.

- **Frammentazione di rete**: si verifica quando, durante una comunicazione di rete, i pacchetti di dati vengono divisi in frammenti per essere inviati. Questo accade per varie motivazioni, ad esempio per limiti di dimensioni del file. Se tutti i frammenti raggiungono la destinazione non si può parlare di frammentazione, al contrario il problema diventa reale nel momento in cui un frammento viene perso in seguito ad un invio non ottimale. Tutto ciò può influire sulla qualità di una determinata comunicazione di rete.

Una digressione più approfondita va fatta per ciò che riguarda la frammentazione all'interno del database. Ogni oggetto presente nel database viene memorizzato nelle caselle di memoria definite, come spiegato nel paragrafo relativo alla memorizzazione logica del database Oracle. Questo dà la possibilità di immaginare la struttura di un tablespace come una griglia di blocchi, ognuno dei quali può essere utilizzato dal database per la registrazione di parte di un oggetto. Quando un utente crea, ad esempio, un nuovo indice il sistema tende a memorizzarlo nel numero di blocchi continuo che ne richiede la sua estensione. È corretto affermare che una volta inseriti i dati per la prima volta, questi non venissero più modificati in nessun modo, non si potrebbe incorrere in questa problematica. Infatti, la frammentazione si verifica quando vengono inseriti, eliminati o aggiornati i dati nel database. Sono queste le principali cause che inducono il sistema a dover rivedere l'organizzazione dei blocchi, creando situazioni in cui i blocchi facenti parte dello

stesso oggetto non risultino contigui all'interno del tablespace. La conseguenza più importante di tutto questo è il fatto che, quando lo spazio libero del tablespace si avvia alla saturazione, all'interno della memoria occupata siano presenti dei blocchi vuoti, non utilizzati, i quali tuttavia non possono essere utilizzati. Ci sono alcune possibili soluzioni che possono risolvere o quantomeno ridurre gli effetti di questa problematica.
Se la frammentazione è relativa agli indici, una possibile soluzione è la pratica di rebuild dell'indice stesso. Consiste nell'eliminazione dell'indice esistente e la successiva ricreazione da zero. Tuttavia, è un'operazione che urge di essere pianificata nel dettaglio e, se possibile, di evitare di essere implementata per il suo carattere invasivo:

- Su indici di grandi dimensioni, praticare una rebuild necessita di importanti allocazioni di risorse, in particolare spazio su disco e capacità I/O.

- Durante il processo, l'indice può risultare temporaneamente disabilitato, rischiando di influenzare le operazioni di scrittura e lettura delle tabelle coinvolte.

- Oltre all'eliminazione dell'indice stesso, vi è la possibilità di perdita delle statistiche relative, che dovranno successivamente essere nuovamente ricalcolate.

Più in generale, il problema della frammentazione può essere combattuto con le operazioni di MOVE degli oggetti, se possibile, ancor più delicate delle rebuild per il rischio concreto di perdita dei dati durante implementazioni non perfettamente pianificate. Per questo motivo, prima di procedere ad effettuare delle MOVE importanti è sempre consigliabile l'esecuzione di un backup che permetta nel peggiore di casi il ripristino del database o del tablespace precedenti ai cambi di posizione.

4.2.1 Fragmentation index

Nell'ambito del contesto della realtà aziendale che ha ospitato questo studio, più nello specifico all'interno della BU di infrastruttura, può capitare di imbattersi in problemi di frammentazione. In particolare, le aziende clienti richiedono spesso l'estensione dei tablespace presenti nei propri database, in crescita continua, grazie alle attività che giornalmente vive qualsiasi tipo di sede aziendale. Per capire meglio il concetto di estensione di un tablespace è utile soffermarsi su come il tablespace viene creato.

```
01 | CREATE TABLESPACE tablespace_name
02 | DATAFILE 'file_path' SIZE size_specification
03 | [ AUTOEXTEND { ON | OFF } ]
04 | [ MAXSIZE max_size_specification ]
05 | [ LOGGING | NOLOGGING ]
06 | [ ONLINE | OFFLINE ]
07 | [ EXTENT MANAGEMENT { LOCAL | DICTIONARY } ]
08 | [ SEGMENT SPACE MANAGEMENT { AUTO | MANUAL } ];
```

Questa è la sintassi base di un'istruzione di creazione di un tablespace. Le parti importanti ai fini della trattazione sono tre:

- **DATAFILE ... SIZE ...**: in questo tratto viene indicato il percorso di accesso al datafile che conterrà tutti i dati memorizzati nel tablespace. Questo è importante in funzione della comprensione del concetto che il livello a cui ci si muove sul tablespace è logico, qualsiasi sia il modo in cui il sistema sceglie di gestire la memorizzazione, tutti i dati beneficiano sicuramente di una loro scrittura sul disco all'interno del datafile dichiarato in questa fase.
- **AUTOEXTEND**: tramite questa voce di configurazione il tablespace può essere reso o meno in grado di auto estendersi, cioè di non richiedere ogni volta che raggiunge la saturazione un intervento manuale per la concessione di ulteriore memoria.

- **MAXSIZE**: questo parametro risulta estremamente importante in funzione dell'abilitazione dell'auto estensione. Il tablespace, anche in auto estensione, non si trova su una macchina con memoria infinita, dunque necessita di avere un limite oltre il quale non possa più estendersi se non con un intervento manuale del DBA.

Quindi, il problema si manifesta quando il tablespace in AUTOEXTEND è completamente saturo e ha raggiunto perfino la MAXSIZE impostata. In questi casi è possibile che, in seguito a un problema di frammentazione, all'interno della memoria occupata siano disponibili degli extents non ancora riempiti, che potrebbero essere occupati con operazioni come quelle di cui sopra. In questo modo si potrebbe evitare di aumentare la MAXSIZE in modo da non sprecare ulteriori risorse e migliorare le prestazioni generali.
È proprio in questo contesto che nasce la necessità di fornire, a chi si pone l'obiettivo di risolvere il problema della frammentazione, uno strumento che permetta di avere una visualizzazione grafica del tablespace e dell'entità del problema con cui deve interfacciarsi. Per questo scopo, partendo da una base che nel contesto aziendale era già in uso, si è implementato uno script applicativo in grado di fornire una possibile soluzione a questa esigenza e, sicuramente, di facilitare gli interventi di questo tipo.
Lo script in questione verrà presentato, spiegato e ne verrà chiarito il funzionamento. In particolare, è opportuno dire che lo script è scritto in PL/SQL, un linguaggio sviluppato per l'utilizzo nel contesto dei database, che è in grado di corredare le istruzioni SQL per la manipolazione dei dati con le procedure proprie dei classici linguaggi di programmazione come C. Verrà presentato per parti, così renderne più chiara e agevole la spiegazione e di conseguenza anche la comprensione.

```
01 | CREATE OR REPLACE type output_extmap
02 | IS
03 |   object
```

```
04 |    (  html_row VARCHAR2(4000));
05 |    /
06 |  CREATE OR REPLACE type output_extmap_table
07 |  IS
08 |    TABLE OF output_extmap;
09 |    /
10 |  CREATE OR REPLACE FUNCTION extmap_html(
11 |    tbsp_name IN VARCHAR2)
12 |  RETURN output_extmap_table pipelined
13 |  IS
```

In questa prima fase, vengono create le strutture dati che serviranno successivamente a richiamare la procedura e a contenere i risultati. Nello specifico si possono notare tre diverse CREATE:

- Creazione di un OBJECT TYPE denominato output_extmap: l'oggetto tipo in Oracle corrisponde ad una struttura dati generica e personalizzabile ad opera dell'utente stesso. È fornito al fine di poter organizzare e manipolare dati in maniera più complessa rispetto alle tipologie predefinite messe a disposizione da Oracle. In questo caso, al suo interno vi è un solo attributo, denominato html_row di tipo VARCHAR2, in grado di ospitare un massimo di 4.000 caratteri a riga.

- Creazione di un tipo di tabella denominata output_extmap_table: questa funzione permette la creazione di una struttura dati che possa contenere delle istanze dell'oggetto output_extmap. Queste due fasi insieme creano la struttura che ospiterà il risultato della funzione al momento in cui viene lanciata.

- Creazione della FUNCTION extmap_html(tbsp_name IN VARCHAR2): viene creata l'effettiva funzione da lanciare per ottenere il risultato obiettivo, notabile che il parametro che riceve tra le parentesi è il nome del tablespace

che si vuole sia analizzato. Il risultato è la tabella creata al passaggio precedente compilata in modo PIPELINED che indica il modo di restituire un risultato secondo cui, viene restituita una riga per volta al posto che memorizzare l'intero risultato per poi restituirlo, questo può aiutare a migliorare le prestazioni del database stesso.

```
01 |    CURSOR extent_map
02 |    IS
03 |      SELECT file_id,
04 |        block_id,
05 |        block_id + blocks - 1 end_block,
06 |        blocks,
07 |        owner,
08 |        segment_name,
09 |        partition_name,
10 |        segment_type
11 |      FROM dba_extents
12 |      WHERE tablespace_name = tbsp_name
13 |    UNION ALL
14 |    SELECT file_id,
15 |      block_id,
16 |      block_id + blocks - 1 end_block,
17 |      blocks,
18 |      'free' owner,
19 |      'free' segment_name,
20 |      NULL partition_name,
21 |      NULL segment_type
22 |    FROM dba_free_space
23 |    WHERE tablespace_name = tbsp_name
24 |    ORDER BY 1,2;
25 |
26 |    CURSOR fragmentation_data
27 |    IS
```

```
28 |          SELECT
29 |                  segment_name,
30 |                  MIN(block_id) AS min_block_per_segment,
31 |                  MAX(block_id + blocks) AS
       max_block_per_segment,
32 |                  MAX(block_id + blocks) - MIN(block_id) AS
       block_distance,
33 |                  COUNT(*) AS occurance_per_segment,
34 |                  COUNT(*) * MIN(blocks) AS block_dimension,
35 |              (COUNT(*) * MIN(blocks))/(MAX(block_id + blocks)
        - MIN(block_id)) AS frag_index,
36 |                  SUM(bytes) AS bytes_dimension
37 |          FROM
38 |                  dba_extents
39 |          WHERE
40 |                  tablespace_name = tbsp_name
41 |          GROUP BY segment_name
42 |          ORDER BY frag_index;
```

A questo punto vengono definiti i cursori che serviranno ad eseguire le due query di cui vi è bisogno per costruire la tabella risultante.
Il primo cursorse serve all'esecuzione di una UNION ALL tra una selezione sulla vista statica dba_extents e una sulla vista statica dba_free_space. Attraverso la prima, vengono recuperate le informazioni su tutti i segmenti presenti nel tablespace da analizzare. In particolare, vengono reperiti:

- L'id dei file.
- Il primo blocco utile alla memorizzazione del segmento.
- Il calcolo dell'ultimo blocco.
- Il numero di blocchi totali allocati alla memorizzazione del segmento.
- Il proprietario.

- Il nome.
- Il tipo.
- La partizione di cui fa parte.

Attraverso la seconda, vengono selezionate le stesse informazioni in merito ai blocchi liberi e sfruttabili per la memorizzazione. In quest'ultimo caso, tuttavia, trattandosi di spazio libero, non vi sono proprietario, nome, tipo e partizione del segmento. Per questo motivo proprietario e nome del segmento vengono inizializzati a "free", mentre nome della partizione e tipo del segmento a NULL. Con la clausola di ORDER BY i risultati vengono ordinati secondo i primi di campi della SELECT, cioè FILE_ID e BLOCK_ID. Il secondo cursore è stato utilizzato per una seconda interrogazione alla dba_extents al fine di ottenere dati più strutturati, utili a dare informazioni al contorno che possano chiarire la visualizzazione grafica proposta. Precisamente, sono stati selezionati il nome del segmento, la posizione del primo blocco di memorizzazione del segmento, la posizione dell'ultimo, la distanza in blocchi tra il primo e l'ultimo e l'effettivo numero di blocchi che serve a memorizzare l'intero segmento. L'ultimo parametro selezionato è l'indice di frammentazione, che corrisponde al rapporto tra il numero di blocchi da cui è composto il segmento e l'effettiva distanza che il segmento occupa all'interno del tablespace.

$$\text{Indice di Frammentazione} = \frac{\text{Numero di blocchi del segmento}}{\text{Distanza effettiva occupata in blocchi}}$$

I blocks corrispondono all'unità di misura che il database Oracle usa per gestire lo storage al suo interno. Inoltre, vengono selezionati anche i bytes da cui è costituito ogni segmento per dare un'idea all'utente dell'effettivo spazio occupato in un'unità di misura più congeniale. Infine, i dati vengono raggruppati per nome del segmento e ordinati in base all'indice di frammentazione calcolato.

È utile chiarire meglio il concetto di indice di frammentazione. L'indice, come detto, corrisponde al rapporto tra effettiva estensione in blocchi Oracle del segmento e distanza tra primo ed ultimo blocco occupato dal segmento nel tablespace. Il valore è adimensionale, in quanto entrambe le grandezze sono misurate in blocchi Oracle. Il range di variabilità del rapporto va da 0 a 1. Il valore zero non può essere effettivamente raggiunto poiché comporterebbe che la distanza tra il primo e l'ultimo blocco sia pressoché infinita, ma è in grado di dare una giusta panoramica sul significato dell'indice; infatti, più aumenta la distanza tra il primo e l'ultimo blocco rispetto all'effettiva estensione, più aumenta l'indicatore di frammentazione del segmento. Il valore 1, al contrario, corrisponde a una situazione perfettamente deframmentate, in quanto i due valori sono identici. Questo dimostra che per il dato segmento non si pongono problemi, i blocchi che lo costituiscono sono uno vicino all'altro nel tablespace. In ultima analisi, andando verso lo 0 aumenta la frammentazione e andando verso l'1 sparisce.

```
01 |  l_row extent_map%ROWTYPE;
02 |    l_cellcolor VARCHAR2(10);
03 |    l_cellwidth NUMBER(3);
04 |    l_datafile  VARCHAR2(100);
05 |    l_output output_extmap := output_extmap(NULL);
06 |
07 |    l_fragmentation_row fragmentation_data%ROWTYPE;
```

In questa fase vengono semplicemente create le variabili che serviranno alla produzione dell'output html.

```
01 |  BEGIN
02 |    l_output.html_row := '<HTML>';
03 |    pipe row(l_output);
04 |    l_output.html_row := '<H2 style="clear:both; font-family:
        monospace; font-weight: bold;">Tablespace '|| tbsp_name
        ||' Block Map </H2>';
```

```
05 |    pipe row(l_output);
06 |    l_output.html_row := '<style>.datafile { margin-bottom: 15
        px; padding: 10px; color: #0A416B; clear:both; } .blocks{
         float:left; width:5px; height:5px; border:1px solid #
        CEDCEA;  padding:5px; } </style>';
07 |    pipe row(l_output);
08 |    l_output.html_row := '<div class="datafile">';
09 |    pipe row(l_output);
10 |    OPEN extent_map;
11 |    LOOP
12 |      EXIT
13 |    WHEN extent_map%NOTFOUND;
14 |      FETCH extent_map INTO l_row;
15 |      IF mod(extent_map%ROWCOUNT,50) = 0 THEN
16 |        l_output.html_row := '<div style="clear:both;"></div>'
        ;
17 |        pipe row(l_output);
18 |      ELSE
19 |        IF l_row.segment_name = 'free' THEN
20 |          l_cellcolor        := '#00FF00';
21 |        ELSE
22 |          l_cellcolor := '#FF0000';
23 |        END IF;
24 |        IF (l_row.block_id = 128) THEN
25 |          SELECT name INTO l_datafile FROM v$datafile WHERE
        file#=l_row.file_id;
26 |          l_output.html_row := '<div style="clear:both; font-
        family: monospace; font-weight: bold;">'||'File ' ||
        l_row.file_id || ' : ' || l_datafile||'</div>';
27 |          pipe row(l_output);
28 |        END IF;
```

```
29 |        l_output.html_row := '<div ' || 'name="' || l_row.
       segment_name || '" title=' || '"' || l_row.segment_name
       || ',' || l_row.blocks || '"' ||'class="blocks" style="
       background-color:'|| l_cellcolor || ';" onClick="
       SetSelectionColor('''||l_row.segment_name ||''')";></div>
       ';
30 |        pipe row(l_output);
31 |      END IF;
32 |    END LOOP;
33 |
34 |    l_output.html_row := '<table border="1" style="margin-top:
         20px;">';
35 |    pipe row(l_output);
36 |    l_output.html_row := '<tr>';
37 |    pipe row(l_output);
38 |    l_output.html_row := '<th>Segment Name</th><th>Block
       Dimension</th><th>Bytes Dimension</th><th>Frag Index</th>
       ';
39 |    pipe row(l_output);
40 |    l_output.html_row := '</tr>';
41 |    pipe row(l_output);
42 |
43 |    FOR l_fragmentation_row IN fragmentation_data LOOP
44 |      l_output.html_row := '<tr>';
45 |      pipe row(l_output);
46 |      l_output.html_row := '<td onclick="SetSelectionColor('''
       || l_fragmentation_row.segment_name || ''')";>' ||
       l_fragmentation_row.segment_name || '</td>';
47 |      pipe row(l_output);
48 |      l_output.html_row := '<td>' || l_fragmentation_row.
       block_dimension || '</td>';
49 |      pipe row(l_output);
50 |          l_output.html_row := '<td>' || l_fragmentation_row.
       bytes_dimension || '</td>';
```

```
51 |      pipe row(l_output);
52 |  l_output.html_row := '<td>' || CASE WHEN MOD(
       l_fragmentation_row.frag_index, 1) = 0 THEN TO_CHAR(
       l_fragmentation_row.frag_index, 'FM99990') ELSE TO_CHAR(
       l_fragmentation_row.frag_index, 'FM99990.99999') END || '
       </td>';
53 |      pipe row(l_output);
54 |      l_output.html_row := '</tr>';
55 |      pipe row(l_output);
56 |    END LOOP;
57 |
58 |    l_output.html_row := '</table>';
59 |    pipe row(l_output);
60 |
61 |    l_output.html_row := '<script>';
62 |    pipe row(l_output);
63 |    l_output.html_row := 'function SetSelectionColor(prm){';
64 |    pipe row(l_output);
65 |    l_output.html_row := 'var elements = document.
       getElementsByName(prm);';
66 |    pipe row(l_output);
67 |    l_output.html_row := 'for(var i=0; i<elements.length; i++)
       {';
68 |    pipe row(l_output);
69 |    l_output.html_row := 'if (elements[i].title.search(/free/i
       ) < 0) {';
70 |    pipe row(l_output);
71 |    l_output.html_row := 'if (elements[i].style.
       backgroundColor == ''rgb(0, 0, 255)'') {';
72 |    pipe row(l_output);
73 |    l_output.html_row := 'elements[i].style.background=''#
       FF0000'';}';
74 |    pipe row(l_output);
75 |      l_output.html_row := 'else { ';
```

```
76 |    pipe row(l_output);
77 |    l_output.html_row := 'elements[i].style.background=''#0000
       FF''; }}}}';
78 |    pipe row(l_output);
79 |    l_output.html_row := '</script>';
80 |    pipe row(l_output);
81 |    l_output.html_row := '</div>';
82 |    pipe row(l_output);
83 |    l_output.html_row := '</HTML>';
84 |    pipe row(l_output);
85 |    RETURN;
86 |  END;
87 |  /
```

Mediante questa parte di script, vengono generate le righe del codice html che letto dal giusto applicativo fornirà il risultato cercato. Ogni riga html compone una riga della tabella generata all'inizio della procedura. Il risultato è che l'intera pagina html sarà contenuta all'interno di una tabella.
Dal tag script in poi vi è un tratto in javascript è stata generata la funzione che si occupa di colorare i blocchi occupati in rosso, i blocchi già occupati e in verde quelli ancora liberi, inoltre si occupa di colorare in blu alla selezione di un blocco occupato dal puntatore, tutti i blocchi facenti parte dello stesso segmento di cui fa parte il segmento in questione.
Il risultato che ne consegue è una pagina html che presenta:

- Il nome del tablespace analizzato.
- Il percorso del datafile in cui avviene lo storage fisico dei dati analizzati.
- Una griglia in cui ogni cella rappresenta un extent del tablespace in questione, rossa se occupata e verde se libera.

- Una tabella in cui vengono riportati il nome del segmento, la sua estensione in blocchi Oracle, la sua estensione in bytes e il rispettivo indice di frammentazione. Inoltre, risultano ordinati rispetto all'indice di frammentazione in ordine crescente.

La pagina è interattiva più precisamente:

- Passando il cursore su una casella della griglia colorata si genera una label che riporta il nome del segmento di cui l'extent fa parte e la sua estensione in blocchi Oracle.
- Selezionando una casella diventa blu insieme a tutte quelle che rappresentano un extent facente parte dello stesso segmento.
- Selezionando dalla tabella il nome di un segmento, automaticamente diventeranno blu tutte le caselle rappresentanti un extent di quel particolare segmento.

L'intero processo è stato automatizzato ulteriormente attraverso un algoritmo Python riportato di seguito.

```
01 | import cx_Oracle
02 |
03 | db_username = 'SYS'
04 | db_password = 'welcome1!'
05 | db_host = '192.168.56.101'
06 | db_service = 'odb'
07 |
08 | cx_Oracle.init_oracle_client(lib_dir=r"C:\oracle\
       instantclient_19_21")
09 |
10 | connection_str = f"{db_username}/{db_password}@{db_host}/{
       db_service}"
```

```
11 |
12 | connection = cx_Oracle.connect(connection_str, mode=
       cx_Oracle.SYSDBA)
13 | cursor = connection.cursor()
14 |
15 | try:
16 |     execute_function_query = "SELECT * FROM TABLE(
       extmap_html('USERS'))"
17 |     cursor.execute(execute_function_query)
18 |
19 |
20 |
21 |     with open("output.html", "w") as file:
22 |         for result_row in cursor.fetchall():
23 |             file.write(str(result_row[0]) + "\n")
24 |
25 |     print("Risultato della funzione scritto nel file 'output
       .html'.")
26 |
27 | finally:
28 |     cursor.close()
29 |     connection.close()
```

L'unica libreria Python utilizzata è stata cx_Oracle, che serve principalmente a poter eseguire istruzioni SQL da quest'ambiente mediante una connessione al database. I passaggi effettuati sono i seguenti:

- Vengono registrati in alcune variabili i parametri utili alla realizzazione della connessione, quali username, password, indirizzo della macchina virtuale ospitante il database e nome del servizio.
- Attraverso la funzione INIT viene passato il percorso dell'Oracle Instant Client allo script, che serve a poter effettuare connessioni a un database che non è fisicamente installato sulla stessa macchina da cui si esegue lo script.

- Viene creata la stringa di connessione poi passata alla funzione CONNECT.
- Viene inizializzato un cursore che sarà utile ad eseguire una query direttamente dallo script.
- A questo punto, mediante l'uso di un blocco try, viene lanciata una query che seleziona tutte le righe della tabella generata attraverso la funzione extmap_html, a cui viene passato come parametro il nome di un tablespace (in questo caso a scopo di esempio viene passato "USERS").
- Viene aperto un documento di testo vuoto creato in precedenza in modalità scrittura (in questo caso denominato "output.html").
- Attraverso un ciclo for vengono lette tutte le righe della tabella risultante ottenuta dalla query appena eseguita e una per una vengono stampate sul documento html.
- Infine, dopo aver scritto un messaggio di buona riuscita delle operazioni sul prompt, vengono chiusi cursore e connessione.

Si fa menzione del fatto che nell'esempio l'utente SYS ha i privilegi da SYSDBA, tutto ciò è obbligatorio nel senso che non è possibile eseguire tutte le procedure appena citate senza i privilegi da amministratore del database. In particolare, senza i privilegi da amministratore, non è possibile interrogare le viste statiche a cui si fa riferimento nello script di partenza. Questo perché le informazioni che si possono scoprire risultano sensibili per il sistema e inoltre, si presuppone la creazione di dipendenze che un utente senza tutti i privilegi potrebbe far fatica a gestire. È importante sottolineare che tutto il procedimento può essere definito come non invasivo se non per la creazione del tipo output_extmap, quindi rimane possibile utilizzare il database mentre è in corso e non rischia di modificarne in alcun modo il contenuto.

Di seguito, attraverso alcuni screenshots, verrà chiarito meglio tutto ciò che è stato spiegato relativamente all'output html.
La label è generata passando il cursore del mouse su una casella. Si può poi notare la funzione secondo la quale, selezionando un'extent, viene colorato in blu insieme a tutti quelli che fanno parte dello stesso segmento.
Il tablespace di esempio non mostra una particolare frammentazione dello spazio libero ma più una frammentazione dei segmenti che ne fanno parte. Ad ogni modo, i tre blocchi rappresentano gli extents ancora non utilizzati all'interno del tablespace. Anche in questa situazione, passando il cursore del mouse su un blocco verde, viene rivelato che l'extent corrispondente è libero e ne viene chiarita l'estensione in blocchi Oracle.
Infine, è raffigurata la tabella, strutturata come spiegato in precedenza, in quest'esempio si possono notare una maggior parte di oggetti che sono serviti allo scopo dell'analisi di performance del capitolo successivo. Importante ricordare la funzione secondo cui, cliccando sul nome del segmento dalla tabella, vengono evidenziate in blu le caselle corrispondenti sulla griglia.

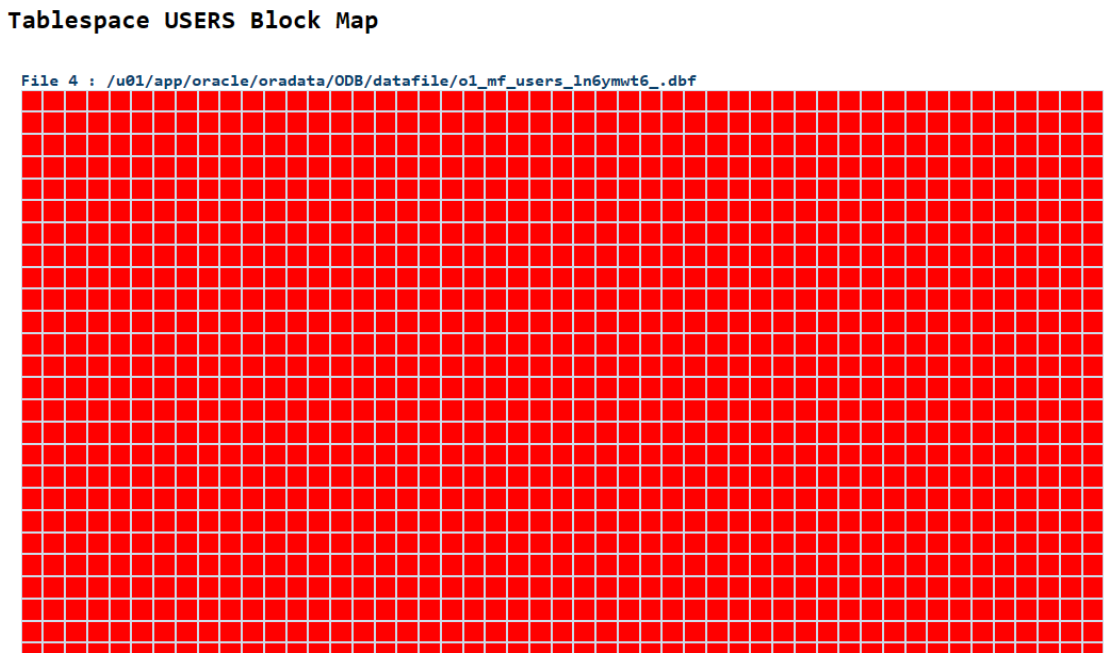

Figura 4.1. Nome del tablespace, percorso del datafile utilizzato e alcuni extents occupati.

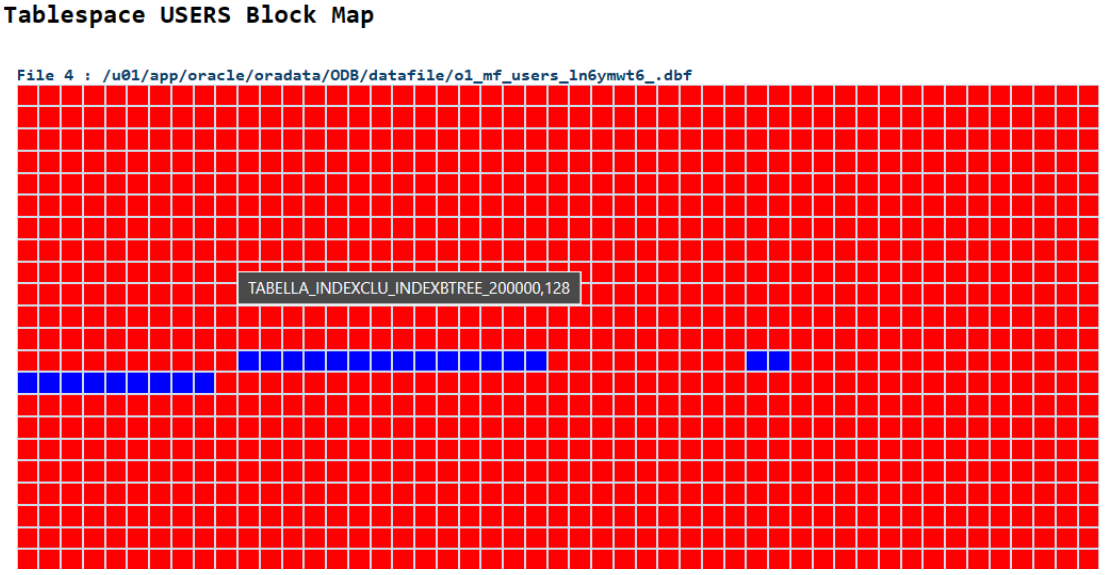

Figura 4.2. Label generata con cursore, raffigurante informazioni sull'extent

Figura 4.3. Blocchi liberi colorati in verde.

Segment Name	Block Dimension	Bytes Dimension	Frag Index
INDICEBTREE_10000_MONTH	32	262144	0.00328
INDICEBTREE_100000_WAREHOUSE_SALES	136	2097152	0.00358
INDICEBTREE_150000_YEAR	144	3145728	0.00364
INDICEBTREE_200000_SUPPLIER	184	8388608	0.0044
INDICEBTREE_100000_RETAIL_TRANSFERS	136	2097152	0.005
INDICEBTREE_100000_ITEM_TYPE	136	2097152	0.00511
INDICEBTREE_100000_MONTH	136	2097152	0.00597
INDICEBTREE_100000_YEAR	136	2097152	0.00776
INDEXBTREE_60000_YEAR	136	2097152	0.00814
INDICEBTREE_150000_WAREHOUSE_SALES	144	3145728	0.01164
INDEXBTREE_60000_MONTH	136	2097152	0.01406
SYS_C006045	184	8388608	0.01407
INDEXBTREE_75000_WAREHOUSE_SALES	136	2097152	0.01411
INDICEBTREE_150000_MONTH	144	3145728	0.01468
INDICEBTREE_150000_ITEM_TYPE	144	3145728	0.01468
INDICEBTREE_200000_YEAR	152	4194304	0.01591
SYS_C006044	160	5242880	0.01869
INDICEBTREE_150000_RETAIL_TRANSFERS	144	3145728	0.01887
INDEXBTREE_55000_YEAR	128	1048576	0.02
TABELLA_INDEXCLU_INDEXBTREE_55000	168	6291456	0.02386
TABELLA_INDEXCLU_75000	184	8388608	0.02684
TABELLA_INDEXCLU_INDEXBTREE_291474	360	31457280	0.02778
INDEXBTREE_75000_ITEM_TYPE	136	2097152	0.02872
INDICEBTREE_200000_MONTH	152	4194304	0.03074

Figura 4.4. Tabella riepilogativa.

Capitolo 5

Analisi di performance

Il cuore di quest'elaborato è concentrato sull'analisi di performance del Database Oracle, effettuata dal punto di vista degli indici sopra trattati. Di seguito verranno presentate, in un primo momento, le assunzioni e tutte le condizioni al contorno utili a creare l'ambiente in cui sono stati effettuati test per raggiungere il target della trattazione, successivamente, verrà presentata l'analisi effettiva. Questa sarà divisa in due parti che prenderanno rispettivamente il nome delle viste dinamiche Oracle da cui sono stati estrapolati i dati, poi studiati e manipolati con l'uso di codice Python e librerie grafiche. In particolare, i dati raccolti sono stati plottati con l'utilizzo della libreria Python Matplotlib, che permette di realizzare attraverso i valori comunicati grafici di vario tipo, alcuni anche molto complessi. Il tipo di visualizzazione che è servita maggiormente e si è rivelata più adatta ai dati raccolti di quest'analisi è l'istogramma. Quindi, l'analisi verrà resa più chiara dalla presenza di alcuni di questi e di tabelle. La scelta dell'istogramma è stata presa in relazione al fatto che le quantità di misurazioni da graficare non erano sufficienti a poter godere delle proprietà fondamentali attraverso un line chart, dunque l'istogramma ha comunicato meglio gli andamenti e i trends di interesse.
Nello specifico la prima delle due parti si riferisce alla vista V$SQLSTATS, in cui si trovano metriche che misurano le prestazioni del database divise per sql_id. Il

sql_id rappresenta un codice univoco che il sistema assegna al testo di ogni quer eseguita, è importante chiarire che può cambiare solamente se viene alterato il testo dello statement, non se lo stesso viene eseguito più di una volta. Nella seconda invece, i dati vengono presi dalla V$SQL_PLAN, una vista in cui, interrogando con un filtro sul SQL_ID, si ottengono tante righe quante sono le operazioni che compongono il piano di esecuzione utilizzato per lo svolgimento della query in questione. In ognuno dei due sottoparagrafi vi sarà una spiegazione delle caratteristiche delle due viste e del mondo in cui sono state utili a raccogliere i dati per questo studio.

5.1 Assunzioni

In primo luogo, è importante specificare gli strumenti utilizzati. La base dati in questione è un Database Oracle versione 19.0.0, installato su una virtual machine Linux montata grazie a VirtualBox.
L'installazione degli strumenti necessari è stata interamente intrapresa. Quindi, dopo avere installato e configurato la macchina virtuale Linux su un sistema Windows, è stato installato il database su di essa. Essenziale precisare che, come prima opera di normalizzazione dell'ambiente, il database in questione era completamente vuoto, così da poter simulare a tutti gli effetti un ambiente di test. Questa scelta è stata compiuta allo scopo di ridurre al minimo il rumore di fondo attorno alle prove effettuate, perché i dati raccolti potessero essere maggiormente puliti e consistenti. In generale è corretto dire che, il rumore è ridotto al minimo, non del tutto assente, in funzione del fatto che, come si è trattato ampiamente nel primo capitolo, esistono processi in background che agiscono in modo silente continuamente permettendo al database di funzionare correttamente in ogni momento.
Per comunicare con il Database si potevano utilizzare due strade differenti, entrambe promosse da Oracle: SqlPlus, uno strumento utilizzabile principalmente dalla

Shell della Virtual Machine, e SqlDeveloper, un'applicazione scaricabile direttamente dal sito di Oracle e tranquillamente installabile su sistema operativo Windows. Per lo svolgimento di quest'analisi è stato scelto di usare il secondo strumento per motivazioni grafiche e visuali, date dal fatto che l'output delle query in SqlPlus è più complesso e necessita di aggiustamenti per poter essere ottimizzato. Inoltre, SqlDevoloper fornisce una comoda interfaccia che permette di lanciare alcuni comandi senza dover per forza soffermarsi a scrivere codice SQL.
È stato indispensabile durante la fase di analisi l'utilizzo i strumenti grafici che aiutassero a poter visualizzare nel miglior modo le tendenza di interesse dei dati, a tal fine è stato utilizzato Python. Nella sezione relativa a V$SQLSTATS l'utilizzo è stato indirizzato soprattutto alla creazione di grafici con l'ausilio della libreria Matplotlib [15]. Invece, nella sezione relativa all'analisi di correlazione sono state utilizzate Matplotlib, Seaborn [16] e Pandas [17] per creare la heatmap.

- Pandas è servito a creare il dataframe che potesse essere trattato dalle altre librerie.

- Seaborn è servita alla manipolazione dei dati a fine statistico, in particolare al calcolo del coefficiente di Spearman.

- Matplotlib ha permesso effettivamente di produrre la visualizzazione grafica migliore.

Il codice è riportato in appendice.
La tabella utilizzata è di notevoli dimensioni, composta da 291.474 records, tagliata a livello di diverse numerosità per avere così più tabelle degli stessi dati su cui fosse possibile capire l'entità del ruolo degli indici nell'esecuzione dei diversi statements pensati. Si tratta di 10 campi totali, di cui quattro VARCHAR di diverse lunghezze e sei NUMBER decimali e interi. L'impianto di creazione delle tabelle è stato il seguente:

- Importazione del dataset come file csv e creazione della corrispondente tabella.
- Replica per tre volte della stessa tabella per avere i tre casi di indicizzazione diversa.
- Prima tabella rimasta senza indici; aggiunta indice clustered alla seconda tabella; aggiunta indice clustered e più indici non clustered alla terza tabella.
- Tagli gradualmente maggiori alle tre tabelle di partenza per avere la variazione del parametro numerosità.

L'impianto appena descritto è chiarito meglio dallo schema seguente.

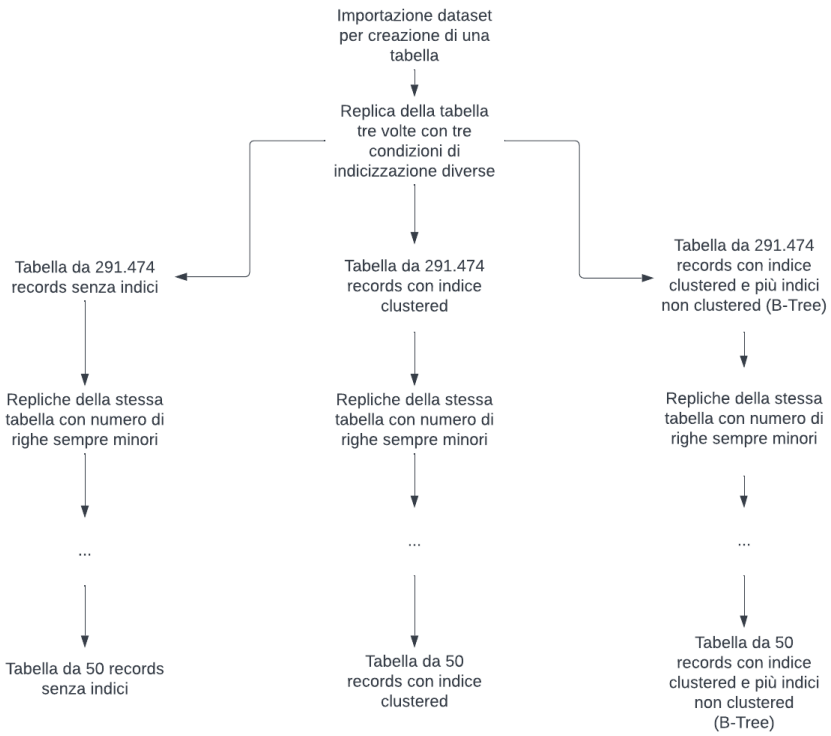

Figura 5.1. Impianto di creazione delle tabelle di test.

Inoltre, si è cercato di normalizzare il più possibile le condizioni del database tra l'esecuzione di uno statement e la successiva. In questo senso, ricopre un ruolo fondamentale la figura della cache. Questa componente potrebbe falsare alcuni risultati ottenuti come le letture da disco o il tempo di svolgimento della query, in quanto, il reperimento da cache dei risultati, se presenti, ne abbassa notevolmente i valori. Avendo appreso la difficoltà nel trovare un'istruzione che potesse obbligare il sistema a non utilizzare la cache in modo permanente, si è deciso di utilizzare uno script di restore dell'ambiente dopo il lancio di ogni statement composto da "ALTER SYSTEM FLUSH SHARED_POOL" e "ALTER SYSTEM FLUSH BUFFER_CACHE". Con il lancio di queste query vengono completamente svuotate shared_pool e buffer_cache, in questo modo il sistema, al lancio dello statement successivo, sarà costretto a reperire obbligatoriamente i dati direttamente da disco,

conferendo consistenza ai risultati poi ottenuti.
Merita menzione il fatto che gli indici utilizzati in quest'analisi non sono affetti dal problema della frammentazione trattato nel capitolo precedente. Esso si verifica nella maggior parte dei casi quando le tabelle in questione sono soggette a frequenti modifiche (inserimenti, aggiornamenti ed eliminazioni di righe). In merito alla tabella in questione si può parlare solamente di eliminazioni di righe nella pratica dei tagli utili alla costituzione delle numerosità differenti. Per supplire alla frammentazione che avrebbe potuto verificarsi, gli indici sono stati creati e applicati alle colonne solamente dopo aver effettuato le rimozioni. Di conseguenza, non si è dovuto procedere ad eventuali ricostruzioni di questi ultimi.
Infine, sono state scelte delle query che si concentrassero sulla selezione, quindi principalmente di SELECT. Questa scelta è stata conseguenza di alcune riflessioni: l'analisi in questione è focalizzata all'ottimizzazione di prestazioni portata dagli indici, i quali per loro stessa natura sono estremamente utili a velocizzare le ricerche più che gli aggiornamenti delle tabelle, inoltre, nel contesto della business unit di infrastruttura in Mediamente assume particolare importanza l'analisi di query di ricerca lente che rischiano di rallentare i database dei clienti fino a portare ad effettivi blocchi. Proprio nel contesto di quest'ultima necessità nasce l'idea per l'approccio alla trattazione del suddetto problema. Le query scelte cercano si spaziare tra tutti gli operatori possibili utilizzabili in combo con una SELECT, quindi sono state usate alcune condizioni di WHERE, di GROUP BY, di aggregazione (SUM, COUNT, AVG, MAX, MIN) e di RANK. La scelta si è soffermata principalmente su codici che mirassero ad applicare filtri e metodi di ordinamento della tabella per poi, in alcuni casi, sommare e manipolare i dati in modo da valutare le prestazioni del database di fronte a questo tipo di lavoro, con e senza l'utilizzo degli indici. I campi selezionati, laddove possibile, sono stati sempre gli stessi (tutti i disponibili) ancora una volta in nome della normalizzazione delle condizioni per ogni test.

5.2 V$SQLSTATS

La vista dinamica V$SQLSTATS [18] si occupa di reperire un enorme mole di statistiche sulle prestazioni di cursori SQL e contiene una riga per ogni SQL_ID diverso (quindi per ogni istruzione SQL diversa). La vista organizza i dati che raccoglie in colonne, quindi, oltre a quelle utili ad identificare la specifica istruzione SQL, si trovano una moltitudine di metriche misurate su ognuno di essi. Oltretutto, se la stessa query viene lanciata più di una volta, i risultati registrati nella vista sono cumulativi, si sommano di volta in volta. Alcuni esempi sono: le esecuzioni di uno statement, il numero di ordinamenti fatti per restituire un risultato, le scritture fisiche sul disco e molte altre. Importante ricordare che le metriche in questione sono pressoché le stesse raccolte da V$SQL e V$SQLAREA, ma la vista in differisce da queste per il suo aggiornamento veloce, per la sua estrema scalabilità e per la sua maggiore capacità di conservare i dati, infatti, i dati relativi ai cursori rimangono al suo interno anche dopo essere usciti dalla shared pool. Soprattutto per l'ultima motivazione appena citata, si è deciso di utilizzare questa vista per reperire i risultati sulle prestazioni delle singole query [19]. Come già spiegato, la vista contiene moltissime metriche distribuite in altrettante colonne e per questa analisi ne sono state selezionate alcune. La scelta delle metriche è stata compiuta pensando soprattutto a quali potessero essere quelle più esplicative e restituissero risultati soddisfacenti senza bisogno di ulteriori analisi al contorno.

5.2.1 Metriche

- **Elapsed time**: si riferisce al tempo che impiega una query ad essere processata, dal lancio della stessa al termine della restituzione del risultato [20]. In particolare, rappresenta il risultato di una somma di tempi: CPU time, user I/O wait time, application wait time, concurrency wait time, cluster wait time, PL/SQL exec time e Java exec time. Si ricorda che Oracle non può

raccogliere statistiche sui tempi di network al di fuori dell'istanza, fattore che verrà considerato come variabilità naturale. Il primo di questi tempi, il CPU time è stato registrato su ogni query e aiuterà al chiarimento di alcuni punti dell'analisi, l'user I/O wait time, essendo un tempo di attesa dipendente dalla sessione dell'utente corrente, verrà assunto simile per ogni statement. Le metriche rimanenti verranno considerate ininfluenti in quanto non vi sono tabelle clusterizzate, si utilizza una sessione per volta che fa capo ad un solo utente (sempre uguale) e non vengono usate istruzioni PL/SQL o Java. Essendo misurato in modo cumulativo, come molte altre grandezze della vista, assume molta importanza la cancellazione della cache ad ogni nuova esecuzione, poiché se la stessa query viene lanciata due volte di fila l'elapsed time ritornato rappresenta la somma di quello relativo alla prima più quello relativo alla seconda. L'unità di misura utilizzata per la sua espressione è il microsecondo, che nell'analisi, talvolta, verrà trasformato in millisecondo per una maggior chiarezza.

- **CPU time**: si riferisce al tempo di CPU impiegato da una query specifica. Come già anticipato è parte dell'elapsed time, ma se ne distacca rappresentando il tempo effettivo di lavoro svolto dal sistema pulito dalle operazioni di I/O [21]. Anche per questa metrica l'unità di misura è espressa dai microsecondi e per una più chiara visualizzazione verrà trasformata in millisecondi. Questa grandezza, come la seguente, verrà utilizzata per chiarire alcuni aspetti dell'analisi effettuata sull'elapsed time, in particolare, chiarirà meglio il tempo utilizzato sulla CPU legato all'utilizzo degli indici. Rispetto a questa grandezza assume particolare importanza l'assunzione di avere un database statico che non avesse processi concorrenti, oltre a quelli di quest'analisi al suo interno, infatti, la loro eventuale presenza, avrebbe potuto portare al rallentamento del lavoro della CPU sovraccarica.

- **Disk reads**: le letture da disco esprimono quante volte vi è stato bisogno di leggere direttamente da disco i dati per restituire un risultato di query [18]. La grandezza è stata presa in considerazione per tutto lo studio, ma verrà utilizzata meno nell'analisi per via delle sue caratteristiche esplicative meno spiccate rispetto alle grandezze temporali. Inoltre, assume particolare importanza dal momento in cui viene assunto come nullo il contributo della cache nel processa mento degli statement. Infatti, la cache è quello strumento che il database utilizza per reperire i dati che servono evitando, dove possibile, la più laboriosa lettura da disco.

- **Buffer gets**: rappresenta il numero di volte che sono state reperite informazioni dalla buffer cache per giungere al risultato [18]. Questa grandezza era stata inizialmente presa in considerazione per poi essere lasciata da parte dopo la decisione di escludere il ruolo della cache allo scopo di normalizzare maggiormente i risultati ottenuti. In generale, avrebbe potuto dare un contributo in caso contrario alla spiegazione di come gli indici potessero modificarla in relazione alle diverse numerosità.

5.2.2 Analisi

In una prima fase, sono state selezionate alcune tabelle caratterizzate da alcune numerosità salienti comprese tra 0 e 291.474, nello specifico: 200.000, 100.000, 10.000, 1.000 e 500. In particolare, sono state create tre copie da 291.474 records della stessa tabella, ognuna rappresentante una situazione di indicizzazione diversa. I tre diversi casi corrispondono alle seguenti tre situazioni: nessun indice, indice clustered (primary key), indice clustered insieme ad alcuni indici b-tree. Per questioni di comodità da ora in avanti verranno denominati caso NOINDEX, caso INDEXCLU e caso INDEXBTREE. A proposito della terza casistica, è importante precisare che è stato creato un indice b-tree per ogni colonna che veniva effettivamente utilizzata

nelle operazioni di filtraggio ad opera delle query. In particolare, le primary key sono state apposte sul campo ID e gli indici b-tree su 5 campi, sia letterali che numerici. Sono state scritte 10 query diverse che riuscissero a comprendere clausole di WHERE, GROUP BY, ORDER BY, RANK, JOIN e funzioni di aggregazione. Al fine di effettuare un focus solamente su query che potessero dare maggiori risultati in termini di prestazioni, è stata praticata una prima scrematura di cui verranno riportati i risultati divisi per statement eseguito e organizzati in tabelle, una per ogni caso di indicizzazione:

Tabella 5.1. Esecuzioni delle query nel caso NOINDEX.

Tipo query/ Numerosità	**200.000**	**100.000**	**10.000**	**1.000**	**500**
WHERE	59,14	36,85	46,48	16,79	13,89
WHERE + GROUP BY	280,71	136,82	20,79	10,05	9,68
GROUP BY + SUM	309,24	96,53	21,74	11,41	5,84
WHERE + GROUP BY + AVG	173,74	134,22	10,32	4,01	12,18
WHERE + RANK	255,58	163,19	22,89	8,63	12,98
WHERE + GROUP BY + COUNT	995,02	671,26	53,31	4,83	3,25
JOIN + WHERE	369,62	213,38	5,53	11,55	4,23
WHERE + GROUP BY + MIN	457,75	206,44	22,68	33,57	4,80
WHERE + GROUP BY + MAX	117,50	668,99	23,72	21,24	8,45
WHERE + ORDER BY	478,08	63,83	13,10	6,34	12,64

Tabella 5.2. Esecuzioni delle query nel caso INDEXCLU.

Tipo query/ Numerosità	**200.000**	**100.000**	**10.000**	**1.000**	**500**
WHERE	49,61	32,65	21,14	13,77	10,97
WHERE + GROUP BY	219,53	126,19	11,54	8,72	5,45
GROUP BY + SUM	267,00	87,23	11,19	7,66	10,49
WHERE + GROUP BY + AVG	238,54	97,80	10,65	10,07	3,5
WHERE + RANK	253,33	165,97	42,38	6,16	5,12
WHERE + GROUP BY + COUNT	1.003,36	876,65	42,38	6,15	6,18
JOIN + WHERE	326,52	178,04	8,64	4,42	3,33
WHERE + GROUP BY + MIN	478,98	150,85	11,02	22,87	14,20
WHERE + GROUP BY + MAX	127,43	635,05	27,33	15,98	16,15
WHERE + ORDER BY	164,01	120,51	30,45	18,16	7,88

Tabella 5.3. Esecuzioni delle query nel caso INDEXBTREE.

Tipo query/ Numerosità	**200.000**	**100.000**	**10.000**	**1.000**	**500**
WHERE	49,61	32,65	21,14	13,77	10,97
WHERE + GROUP BY	219,53	126,19	11,54	8,72	5,45
GROUP BY + SUM	267,00	87,23	11,19	7,66	10,49
WHERE + GROUP BY + AVG	238,54	97,80	10,65	10,07	3,5
WHERE + RANK	253,33	165,97	42,38	6,16	5,12
WHERE + GROUP BY + COUNT	1.003,36	876,65	42,38	6,15	6,18
JOIN + WHERE	326,52	178,04	8,64	4,42	3,33
WHERE + GROUP BY + MIN	478,98	150,85	11,02	22,87	14,20
WHERE + GROUP BY + MAX	127,43	635,05	27,33	15,98	16,15
WHERE + ORDER BY	164,01	120,51	30,45	18,16	7,88

I dati rappresentano l'elapsed time in millisecondi della query (identificata dalle clausole utilizzate al suo interno) in relazione alla tabella, la cui numerosità è identificata dalla colonna, su cui viene lanciata. Si è scelto di effettuare questa prima selezione utilizzando l'elapsed time perché è la metrica principale su cui verranno analizzate le prestazioni.
Qui di seguito un elenco di quelle scelte:

- **WHERE**
- **WHERE, GROUP BY, SUM**
- **WHERE, ORDER BY**
- **WHERE, JOIN**

La scelta è ricaduta su query che hanno dato dei risultati di miglioramento di oltre 50% dalla situazione di NOINDEX a quella di INDEXBTREE e su quelle in generale più utilizzate nei sistemi di accesso, reperimento e visualizzazione dei dati livello aziendale. In questo senso sono state selezionate le query di JOIN e di WHERE insieme alle altre. La scelta della SUM è stata presa in relazione alla volontà di un'analisi su un una funzione di aggregazione che fosse una via di mezzo in termini di costo computazionale sulla CPU tra MIN, MAX, COUNT e AVG, e che utilizzasse obbligatoriamente una clausola di GROUP BY. Ciò che ne scaturisce è la grande capacità della presenza di indicizzazione di migliorare di molto prestazioni di query in presenza di filtraggi, ordinamento e funzioni di aggregazione soprattutto.

5.2.3 Query di filtro

La query pensata è la seguente:

```
01 | SELECT *
02 | FROM <<TABELLA NOINDEX/INDEXCLU/INDEXBTREE>>
```

```
03 | WHERE item_code > 10 AND item_code < 126000 AND
04 |       item_type = 'wine' AND retail_transfers > 5 AND
05 |       retail_transfers < 2500 AND warehouse_sales > 50 AND
06 |       warehouse_sales < 2500;
```

Rispetto a quest'ultima il lavoro fatto è stato il seguente: si è proceduto a riscrivere la query con particolare attenzione all'aumento di condizioni di WHERE, cercando di porre almeno una di esse su ogni colonna indicizzata. Si è notato un trend secondo il quale, ad alti valori di N, l'elapsed time risultava diminuito nel contesto degli indici, tuttavia, ad N sempre più bassi, l'indicizzazione diventava sempre meno conveniente, fino a diventare perfino svantaggiosa. Quindi, si è applicata una strategia di tagli successivi dagli N più bassi e dagli N più alti con lo scopo di trovare il range meno largo possibile in cui era evidente il cambio di tendenza. I risultati sono riassunti nel plot seguente.

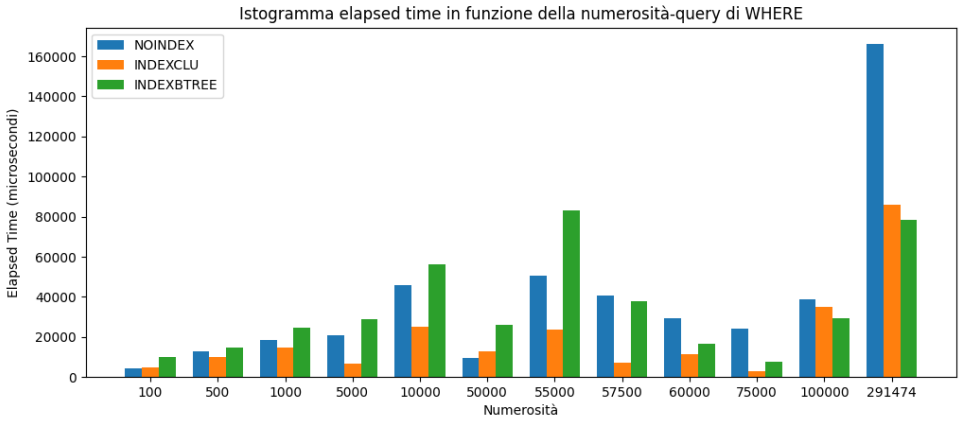

Figura 5.2. Isotgramma elapsed time-numerosità nei tre casi di indicizzazione.

Dal grafico si possono evincere alcune considerazioni. Innanzitutto, ci si potrebbe aspettare a priori che all'aumentare della numerosità, l'elapsed time aumenti e quest'ipotesi in generale è confermata, ma in alcuni punti non è così. Ci si riferisce in particolare al range compreso tra 50.000 e 291.474, laddove i tempi sembrano

diminuire per poi aumentare nella totalità della tabella di partenza. Per spiegare completamente questa tendenza bisognerebbe riproporre la stessa query su tabelle di tutte le numerosità comprese tra 50.000 e 291.474, ma questo sarebbe estremamente laborioso e non oggetto di questo elaborato. Tuttavia, nonostante non si possa affermare che la relazione tra elapsed time numerosità sia direttamente proporzionale, si può affermare che la tendenza espressa in precedenza sia confermata. Si è proceduto con un metodo "a forbice" secondo il quale, la query è stata eseguita sequenzialmente sulle tabelle con N sempre più discendente partendo dal massimo e sempre più crescente partendo dal minimo, in modo da poter isolare un range di valori in cui presumibilmente si poteva trovare la numerosità in cui il trend di convenienza degli indici si trasforma in un trend di sconvenienza. In questo modo si è stati in grado di poter isolare un range di circa 2.500 valori tra 57.500 e 60.000 in cui probabilmente risiede il punto ricercato. Con altrettanta sicurezza si può affermare che il punto esatto non risulterebbe semplice da trovare a causa della variabilità a cui è soggetta la misura dell'elapsed time per i motivi legati alla connessione di rete, di cui si è trattato in precedenza. In funzione del risultato appena ottenuto, sarebbe interessante capire, con l'aiuto di altri dataset, se la numerosità trovata rispetto a questo tipo di query potrebbe cambiare oppure rimanere in un range di valori accettabile la medesima, tuttavia, risulterebbe estremamente complicato creare un ambiente di test all'interno di un database che risulti completamente privo di rumore, molto probabilmente bisognerebbe ricercare metodi alternativi per la misura dell'elapsed time. Molto importante ricordare che la buona regola nella costruzione di una tabella è la scelta di una chiave primaria adatta, questo rende in generale più comodo ogni tipo di operazione e, come si evince da questo plot, migliora anche le prestazioni, infatti, le barre di INDEXCLU, anche se raramente di un gap eccessivo, risultano più basse di quelle di NOINDEX. Infine, si può evincere chiaramente come su grandi numerosità l'ottimizzazione delle prestazioni conferite dagli indici, corrispondente in questo a caso a circa il 53% (calcolato come scostamento tra i

casi NOINDEX e INDEBTREE a N = 291.474), sia ben evidenziata.

5.2.4 Query di aggregazione

La query pensata è la seguente:

```
01 | SELECT year, month, supplier, item_code, item_description,
       item_type,
02 |         retail_sales, warehouse_sales, id_generico, SUM(
       retail_transfers) AS total_retail_transfers
03 | FROM <<TABELLA NOINDEX/INDEXCLU/INDEXBTREE>>
04 | WHERE item_code > 10 AND item_code < 126000 AND
05 |       item_type = 'wine' AND retail_transfers > 5 AND
06 |       retail_transfers < 2500 AND warehouse_sales > 50 AND
07 |       warehouse_sales < 2500
08 | GROUP BY id_generico, year, month, supplier, item_code,
       item_description,
09 |          item_type, retail_sales, warehouse_sales;
```

Come già accennato, la funzione SUM ha un costo computazionale più elevato delle COUNT e MAX/MIN, ma pur sempre meno di AVG. È importante ricordare che il costo computazionale di tutte quante cresce linearmente con il numero di righe presenti nella tabella da scandire. Lo studio effettuato sulle prestazioni di questa query si è soffermato sui trend già esplorati riguardo alla precedente e sulla frazione di elapsed time ricoperta dal tempo speso sulla CPU. Per quest'ultima motivazione risulta importante la scelta di una funzione di aggregazione, essendo proprio la CPU quella incaricata a dover sostenere il costo computazionale dell'elaborazione. Di seguito viene riportato il grafico realizzato per capire la variazione di elapsed time rispetto alla numerosità per le tre casistiche in esame.

La caratteristica più evidente del grafico è l'andamento quasi costante del valore

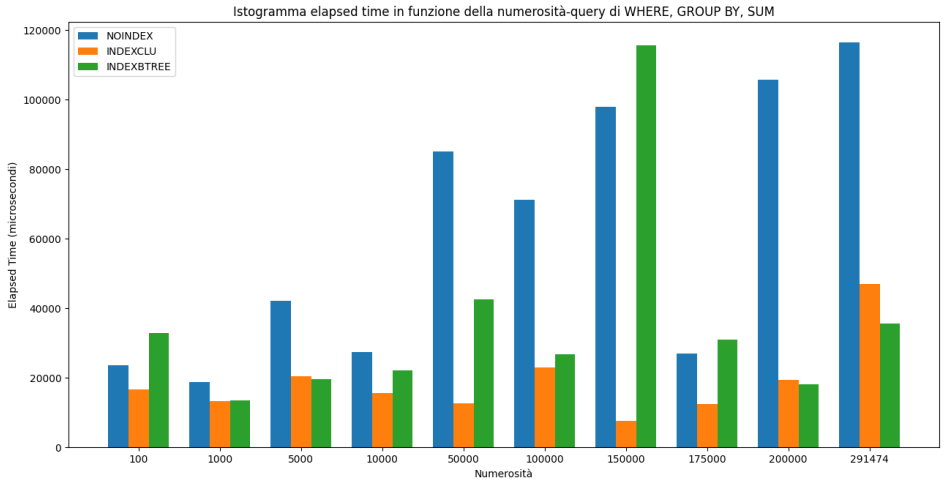

Figura 5.3. Isotgramma elapsed time-numerosità nei tre casi di indicizzazione.

di elapsed per il caso INDEXCLU, infatti, si nota chiaramente che le sue prestazioni sono pressoché sempre le migliori, ad esclusione di rari casi, nei quali comunque rasenta quelle del caso INDEXBTREE (si ricorda che anche in questo caso è presente l'indice clustered). Questa tendenza è sicuramente causata dal fatto che in questa query il campo ID_GENERICO, che è chiave primaria, è stato inserito come primo nella clausola GROUP BY. È utile ricordare appunto, che la GROUP BY intima il raggruppamento dei risultati secondo l'ordine in cui vengono inseriti i campi. Dunque, se viene inserito ID_GENERICO prima di YEAR, il risultato verrà raggruppato secondo il primo campo, e quel che viene restituito ordinato secondo YEAR. In generale, si può scorgere come la tendenza secondo cui gli indici avvantaggiano ad alti N e svantaggiano nel contrario è confermata anche per questa query, seppur in modo meno accentuato. In N = 291.474 la prestazione di elapsed time migliora del 70% a seguito dell'introduzione di indici clustered e non-clustered, ma ad N = 100 la situazione è inversa, nonostante in proporzione lo svantaggio sia minimo, circa il 30%. Per questo statement è risultato più complesso applicare la

strategia della “forbice”, in quanto è difficile scorgere la tendenza del vantaggio o svantaggio nell’uso degli indici a rimanere costante. Infatti, subito dopo la numerosità 100 si ha la situazione inversa alla numerosità 5.000, un intervallo considerato per i parametri di quest’analisi non molto esteso. Non avendo ritrovato un trend particolare, come nel caso precedente, si è scelto di analizzare il rapporto tra elapsed e CPU time. A questo proposito sono stati realizzati tre grafici, uno per ogni situazione di indicizzazione, nei quali le due barre dell’istogramma rappresentano l’elapsed time e il CPU time in funzione della numerosità.

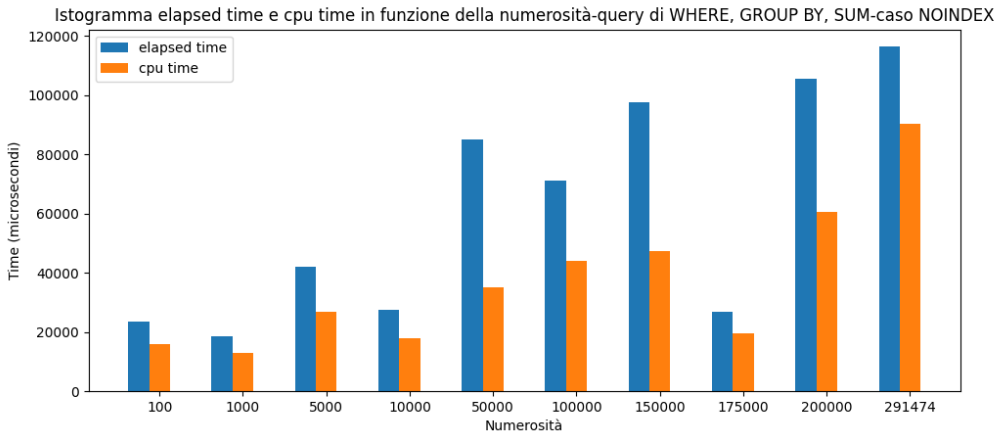

Figura 5.4. Isotgramma CPU time-numerosità nel caso NOINDEX.

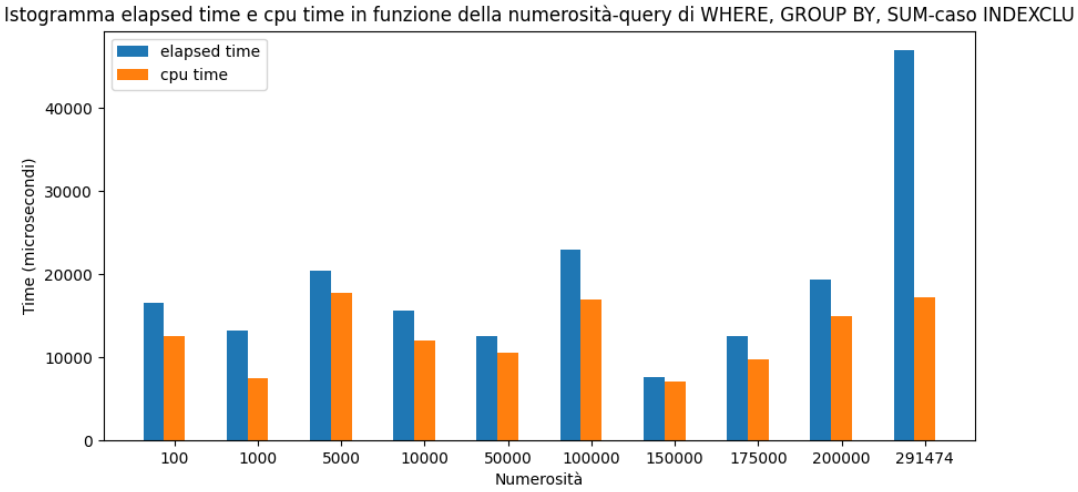

Figura 5.5. Isotgramma CPU time-numerosità nel caso INDEXCLU.

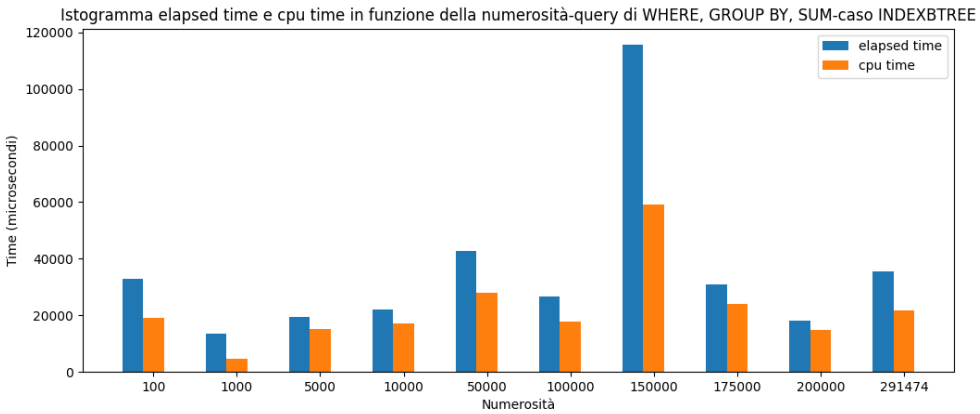

Figura 5.6. Isotgramma CPU time-numerosità nel caso INDEXBTREE.

Dal primo grafico si può facilmente notare come nella maggior parte dei record sia evidente un particolare scostamento tra CPU ed elapsed e dall'andamento dell'elapsed si sa che per alte numerosità esso aumenta molto più che negli altri due casi. Questo può testimoniare come l'aumento dell'elapsed non sia per forza da attribuire a un maggior carico sulla CPU, ma probabilmente ad altri fattori. Tra quelli

citati in fase di definizione dell'elapsed time, il più probabile sembrerebbe essere il tempo da dedicare alle operazioni di input/output, quindi, molto semplicemente, avendo da scandire l'intera tabella rispetto agli altri due casi forniti di indice, lo scostamento dipende dalla maggior quantità di dati che all'esecuzione della query il sistema riceve in input.
Inoltre, è facile notare come nell'ambito del caso INDEXCLU la quasi totalità dell'elpsed time è da imputare a tempo speso sulla CPU. In questo caso i tempi di quest'impianto di indicizzazione sono i migliori sull'insieme dei dati a disposizione, ma se così non fosse, il primo parametro da indagare per un miglioramento di prestazione sarebbe il rilascio di parte delle risorse allocate sulla CPU e meno, rispetto agli altri due casi, al reperimento di informazioni in input.
Oltretutto si può affermare che, se il primo campo utilizzato nell'ordinamento coincide con il campo scelto come chiave primaria della tabella, le prestazioni di elapsed time risultano perfino migliori di quelle che sfruttano gli indici B-Tree in generale. Quest'ultima alternativa paga sicuramente il tempo e le risorse utilizzati gestire gli indici nella ricerca. Va fatta menzione del fatto che sulla grande numerosità continua a convenire, in termini di tempo, la soluzione di un buon impianto di indicizzazione equilibrata tra indice clustered e indici non-clustered. Ciò è testimoniato da una diminuzione dell'elapsed time rispetto al caso NOINDEX del 69% e, seppure molto minore, del 24% rispetto al caso INDEXCLU. Si paga in termini di CPU rispetto a quest'ultimo caso uno scarto di circa 4 millisecondi e 9 letture da disco in più, ma sicuramente a livello delle numerosità delle tabelle utilizzate a livello aziendale non ci sarebbe alcun dubbio su quale tipo di scelta intraprendere.
Infine, va evidenziata la particolare tendenza che si verifica a basse numerosità, infatti, a N = 100 elapsed time e CPU time sono minori nel caso di INDEXCLU. Si era già sottolineato in merito alla query precedente come sia difficile trovare prestazioni eccelse di più indici a basse numerosità. Tuttavia, questo dato aiuta a capire l'importanza della chiave primaria nel contesto di creazione di una tabella,

infatti, è sufficiente un ordinamento secondo la chiave primaria ad ottenere una notevole diminuzione del tempo di svolgimento della query. Particolarmente minori, oltre ai due tempi, anche le letture da disco, 54 in questo caso contro le 71 di INDEXBTREE e le 83 di NOINDEX.

5.2.5 Query di ordinamento

La query pensata è la seguente:

```
01 | SELECT *
02 | FROM <<TABELLA NOINDEX/INDEXCLU/INDEXBTREE>>
03 | WHERE item_code > 10 AND item_code < 126000 AND
04 |       item_type = 'wine' AND retail_transfers > 5 AND
05 |       retail_transfers < 2500 AND warehouse_sales > 50 AND
06 |       warehouse_sales < 2500
07 | ORDER BY retail_transfers ASC, warehouse_sales DESC;
```

La ORDER BY è la clausola più semplice da utilizzare in ambito SQL che permetta di ordinare i risultati di uno statement in modo ascendente o discendente, rispetto ad una o a più colonne e perfino rispetto a combinazioni di più colonne, quali somme o sottrazioni. La presenza di WHERE per filtrare i dati si è resa necessaria per capire, in relazione al cambiamento della numerosità, se e come cambiasse il metodo di accesso alla tabella. L'operazione di ORDER BY può risultare molto dispendiosa su grandi numerosità in termini di risorse allocate per la soluzione ed in termini di tempo. Per questa query si è plottato il comportamento dell'elapsed time in funzione della numerosità considerando il caso ulteriore di obbligo attraverso un hint all'utilizzo dell'indice. In Oracle, gli hint sono dei suggerimenti aggiunti all'interno del testo della query, utili all'ottimizzatore per avere degli spunti di movimento quando approccia alla produzione del piano di esecuzione.

In questo caso viene utilizzato il metodo per intimare all'ottimizzatore l'uso degli indici nell'approccio alla tabella.

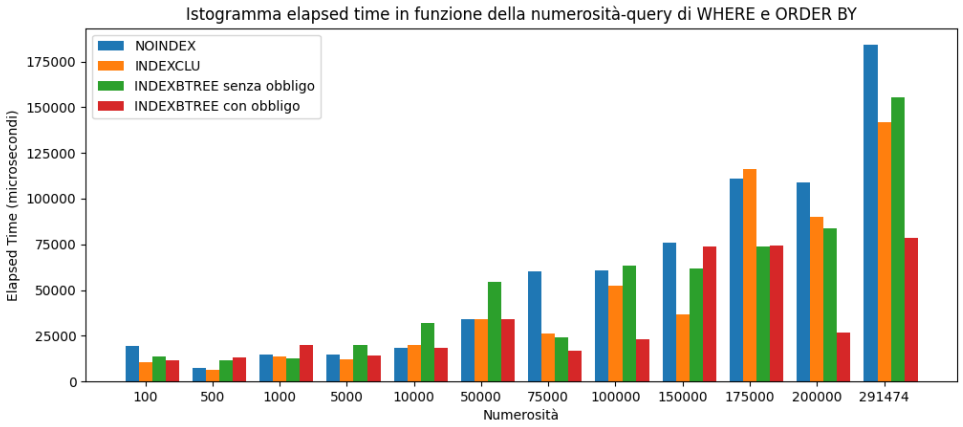

Figura 5.7. Isotgramma elapsed time-numerosità nei tre casi di indicizzazione e in quello con hint di utilizzo degli indici.

Appare subito chiara la motivazione dietro alla considerazione degli hint, infatti, alla numerosità più alta l'utilizzo degli hint sopracitati porta a una diminuzione dell'elapsed time circa del 50% rispetto al caso senza obbligo. Ciò suggerisce che, molto probabilmente, non tutte le volte che l'ottimizzatore si trova davanti ad un indice sceglie di utilizzarlo, nello specifico evidentemente a primo impatto non lo farebbe, ma a posteriori non ve ne sono ragioni. Si nota come fino a numerosità nell'intorno di N = 175.000 continua a convenire l'utilizzo degli indici. Proprio a 175.000 si nota come per la prima volta, scendendo con la numerosità, l'ottimizzatore scelga da solo di utilizzare gli indici. Da questo punto fino a zero, non considerando un solo outlier, le prestazioni con e senza obbligo sono simili, ciò significa che diminuendo la numerosità l'ottimizzatore risulta più propenso all'utilizzo degli indici rispetto alle alte numerosità. Questo comportamento risulta in controtendenza con quello che assumeva rispetto alle query precedenti e, molto probabilmente, risulta strettamente connesso alla presenza della clausola di ordinamento ORDER BY.

Nell'analisi di questa query si è scelto di valutare anche la variazione delle disk reads in funzione della numerosità, per capire se, rispetto al parametro di elapsed time, fossero visibili dei trend più definiti. Il seguente è il grafico prodotto dai dati:

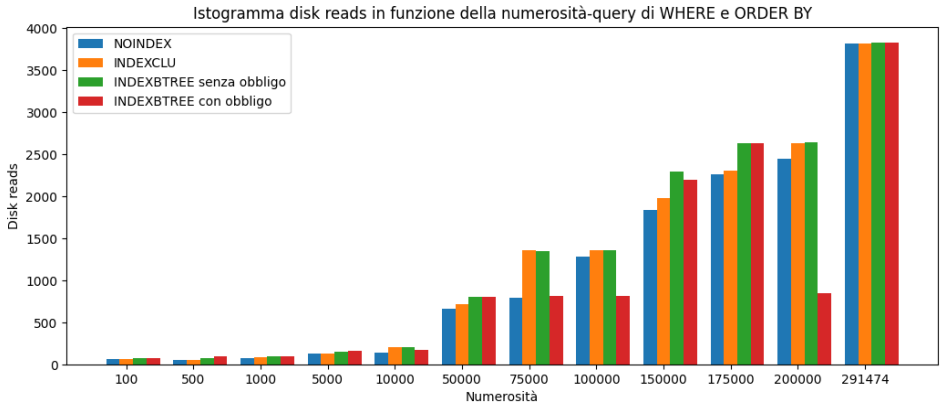

Figura 5.8. Isotgramma disk reads-numerosità nei tre casi di indicizzazione e in quello con hint di utilizzo degli indici.

Si può notare come in questo caso esiste un range di numerosità chiaro all'interno del quale le operazioni all'interno del piano di esecuzioni richiedono un utilizzo decisamente maggiore di risorse. Infatti, tra 10.000 e 50.000 righe di tabella le letture da disco passano da circa 200 a più di 650. Il cambio di tendenza non riguarda soltanto il numero di letture di tutti e quattro i casi, prima di questo range i valori rimangono quasi invariati tra loro, dopo N = 50.000 iniziano a distaccarsi, fino a tornare quasi allineati a 291.474. Dal grafico si può evincere come il parametro studiato peggiori con l'utilizzo degli indici. Questo è certamente conseguenza del fatto che l'accesso alla tabella in Full Scan implica che i dati siano letti una volta tutti insieme, mentre l'Index Scan ha bisogno di più letture, conseguenza della presenza degli indici. In generale, si può affermare che la presenza degli indici su numerosità abbastanza alte può portare ad un notevole risparmio di tempo, ma può essere pagata con il bisogno di un eccesso di letture da disco. In ultima analisi, è molto importante conoscere a priori i parametri che si vogliono migliorare perché

quasi tutte le implementazioni possono portare a miglioramenti di alcuni fattori a discapito di altri.

5.2.6 Query di join

La query pensata è la seguente:

```
01 | SELECT a.supplier AS a_supplier, a.item_description AS
       a_item_description,
02 |        b.supplier AS b_supplier, b.item_description AS
       b_item_description
03 | FROM <<TABELLA NOINDEX/INDEXCLU/INDEXBTREE>> a
04 | JOIN <<TABELLA NOINDEX/INDEXCLU/INDEXBTREE>> b
05 | ON a.id_generico = b.id_generico
06 | WHERE a.retail_transfers > b.retail_transfers AND
07 |       a.warehouse_sales > b.warehouse_sales;
```

La scelta di analizzare un join è stata presa in relazione all'importanza che questa tecnica riveste nell'ambito dei database aziendali. Il join è alla base delle architetture di quasi tutti i sistemi basati su RDBMS per il suo ruolo fondamentale di creazione di collegamenti tra tabelle. In particolare, esistono diversi tipi di join supportati da Oracle: Inner Join, Left Join, Right Join, Full Join, Cross Join e Self Join. In questo caso si è dovuti ricorrere all'ultimo tipo per via del fatto che la tabella a disposizione è solamente una. Nonostante ciò, si è comunque riusciti a pervenire a risultati interessanti in analisi dei tre parametri considerati.

In questa fase si è scelto di mettere un tetto ai valori di elapsed time raggiunti nel caso NOINDEX per il loro valore fuori range rispetto agli altri, in particolare, per le misurazioni a N superiore a 50.000, si è sempre superato i 100.000 microsecondi. Per via dei valori nettamente minori degli altri due casi, si sarebbe rischiato di perdere l'apprezzabilità delle differenze tra 0 e 100.000 microsecondi riportando

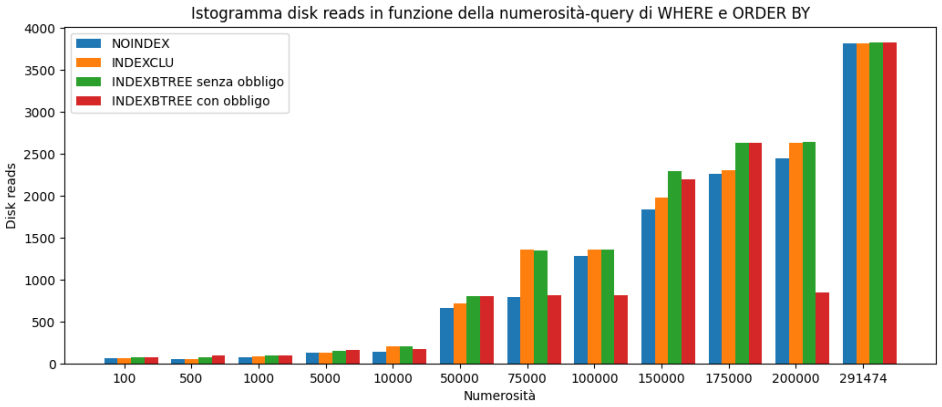

Figura 5.9. Istogramma elapsed time-numerosità nei tre casi di indicizzazione.

le superiori sul grafico.

Nel raccogliere i risultati sui tempi, sono state chiare da subito alcuni andamenti. A prescindere dalla numerosità, sul self join le soluzioni che prevedono la presenza di indici raramente superano i 400.000 microsecondi sotto queste condizioni di filtro. Il dato è molto importante perché sottolinea l'estremo bisogno della presenza di una chiave primaria per sopperire all'intasamento temporale che si creerebbe in sua assenza. L'ultimo valore limitato nel grafico è 1.219.197 microsecondi, rappresentante l'elapsed time nel caso NOINDEX quando la query è lanciata sulla tabella da 291.474 righe. Nel contesto di business come quello dove è stata realizzata questa tesi, si ha a che fare giornalmente con tabelle e dataset di dimensioni anche decisamente più grandi di quelle di questa tabella, e soprattutto, le query che esegue il database sono molte e nello stesso istante, tempi di svolgimento del genere risulterebbero insostenibili per qualsiasi tipo di sistema. Invece, soffermandosi sulla relazione tra caso INDEXCLU e INDEXBTREE, si può notare la strana tendenza secondo la quale i tempi realizzati nel caso INDEXCLU siano pressoché sempre migliori degli altri. Il trend risulta anomalo perché, a priori, sembra normale aspettarsi che la presenza di indici non-clustered sulle principali colonne di filtraggio

possa solamente migliorare le prestazioni temporali, al contrario il grafico dimostra che non è sempre così. Risulta comunque importante ricordare che in questo caso il join è realizzato sulla colonna ID_GENERICO, che è anche chiave primaria per entrambi i tipi di tabella. Dunque, essendo il join una operazione costosa dal punto di vista computazionale e di risorsa, è molto probabile che tra i presenti l'indice che riesce in misura maggiore a cambiare i dati raccolti sia la chiave primaria.
L'altra grande caratteristica che si evince facilmente dal grafico è la tendenza dei benefici portati dagli indici a scomparire gradualmente con la diminuzione della numerosità. Nello specifico lo studio si è soffermato sul ricercare un range non troppo largo di valori all'interno del quale si può individuare il salto netto che rende il gap, tra l'elapsed time del caso N0INDEX e quello relativo al più alto tra i due restanti, significativamente minore. Quel che risulta è l'intervallo di valori tra N = 20.000 e N = 25.000 è quello che ospita probabilmente la numerosità di inversione di tendenza. Rispetto ad altre query già trattate, è da sottolineare che l'elapsed del NOINDEX non risulta essere il migliore sotto al valore di N appena trovato, ma le prestazioni dei tre casi tendono ad allinearsi e appianare i gap di differenza.

5.3 Analisi di correlazione

Prima di passare alla trattazione specifica sui piani di esecuzione, si è scelto di soffermarsi in breve sull'esistenza o meno di correlazione tra alcune delle variabili considerate durante l'analisi appena conclusa. In particolare, si è scelto di studiare la correlazione tra le varie combinazioni di elapsed time, numerosità della tabella e condizione dii indicizzazione. La selezione è stata condotta escludendo altre variabili tra le quali: CPU time, disk reads e la diversità tra le query. Il CPU time non è stato preso in considerazione in virtù del fatto che è parte dell'elapsed time, in quanto quest'ultimo corrisponde alla somma di questo ed altri tempi. Questo vuol dire che nel contesto dell'analisi di correlazione i due andamenti possono essere

assunti come simili e che i risultati ottenuti sull'elapsed time possono rispecchiare con approssimazione a quelli che si sarebbero ottenuti considerando anche il CPU time. Le disk reads non sono state considerate per il loro uso estremamente ridotto rispetto alle altre variabili nell'analisi, sono state utili a tratti come appoggio alla spiegazione di altri aspetti ,dunque, si è pensato di escluderle da un eventuale analisi di correlazione per cui serve avere a disposizione più dati possibile.
Infine, un discorso distaccato va affrontato per la diversità delle query: sarebbe stato possibile raccogliere dati sull'elapsed time al variare della numerosità delle righe della tabella, della condizione di indicizzazione e delle query che venivano eseguite. Purtroppo, questa variabile avrebbe dovuto essere trattata come non quantitativa e poi essere normalizzata in categorie, dove un numero potesse rappresentare ogni categoria di query, tuttavia, si è riscontrata difficoltà a trovare una scala coerente, consistente e robusta al punto da essere presa in considerazione costringendo allo scarto della variabile ai fini del risultato.
Prima di tutto bisogna introdurre brevemente il concetto di correlazione. Rappresenta un concetto statistico che permette di esplorare o quantificare le relazioni esistenti tra due o più variabili. Dunque, grazie a questo strumento, è possibile determinare l'esistenza e l'entità di eventuali connessioni o collegamenti tra le variabili in gioco. In generale, nel contesto dell'Anomaly Detection e, più in generale nel contesto aziendale, può essere estremamente utile conoscere questo tipo di informazione perché potrebbe facilitare di molto il lavoro di ricerca al fine della scoperta di eventuali anomalie e collegamenti tra esse. Invece, nel contesto di questo particolare studio, può essere utile per dare un'ulteriore sfumatura alla trattazione e a comprendere con maggior forza i dati che compongono le reportistiche.
Il metodo più efficacie per misurare la correlazione è quello di utilizzare un indice che produca un numero compreso tra 1 e -1, più precisamente:

- **1**: indica una correlazione postiva perfetta.

- **0**: indica correlazione assente, le due variabili non sono in alcun modo collegate.

- **-1**: indica una correlazione negativa perfetta.

Ovviamente i casi reali capitano solitamente all'interno degli intervalli citati, ciò significa che ogni risultato necessita di essere interpretato successivamente.
È importante ricordare che la presenza di correlazione non implica per forza causalità tra le variabili, infatti, dopo avere appurato l'esistenza di una forte correlazione si rende opportuna un'ulteriore ispezione volta a studiare la vera natura dietro il risultato riscontrato.
Il primo passo seguito, al fine dello svolgimento di una corretta analisi, è stato scegliere il coefficiente che potesse esprimere al meglio la correlazione tra le variabili in gioco tra coefficiente di Pearson e coefficiente di Spearman. Il più largamente utilizzato risulta essere quello di Pearson [22] ma, per una sua corretta implementazione necessita di alcune assunzioni fondamentali sui dati e sull'ambiente dello studio, più precisamente:

- Le variabili da confrontare devono essere tutte quantitative.

- Le variabili devono essere tutte appaiate sugli stessi casi.

- Il grafico di correlazione di ogni coppia di variabili deve mostrare in ogni caso una relazione di tipo lineare.

- Non devono essere presenti nel set di dati degli outliers influenti.

- La distribuzione di tutte le variabili deve essere normale, o almeno deve essere possibile assumerla come tale.

Nel caso di questa trattazione non vengono soddisfatte tutte le assunzioni. Due delle tre variabili sono quantitative, ma la condizione di indicizzazione rispecchia

più una divisione in categorie che una variabile quantitativa. Non è possibile dimostrare l'esistenza di un legame lineare tra tutte le coppie di variabili in gioco, soprattutto tra la condizione di indicizzazione e le altre due. Sono presenti valori anomali all'interno dei dati raccolti, non si è proceduto a una pulizia di questi per l'importanza che hanno ricoperto nella trattazione e in tutti i casi non si potrebbe assumere come ininfluente il loro apporto. Infine, la condizione di normalità non può essere assunta per via della ridotta dimensione del set di dati a disposizione.
Per tutti questi motivi si è scelto l'utilizzo del coefficiente di Spearman [23], che al contrario del primo, prevedeva la possibilità dii essere utilizzato in situazioni più generali. Dunque si è selezionata una query di selezione con parametri di filtraggio e raggruppamento, eseguita sulla maggior parte delle tabelle create per l'analisi precedente, nelle tre condizioni di indicizzazione diversa. Il risultato è stata la produzione di un file csv che riportasse l'elapsed time misurato, la numerosità delle righe della tabella corrispondente e una stringa tra "NOINDEX", "INDEXCLU", "INDEXBTREE". I dati sono stati elaborati attraverso l'uso del codice Python citato nelle assunzioni, che permettesse, ricevendo in input il file in formato csv, di produrre una heatmap con coefficienti calcolati con il metodo di Spearman appunto. La heatmap [24] è una visualizzazione grafica che studia la direzione e la forza della correlazione tra le variabili in gioco, producendo delle zone colorate al suo interno. La tonalità del colore indica il livello di correlazione esistente tra le due variabili considerate, secondo una scala spiegata a lato della mappa. Bisogna ricordare che i dati in input comunicati con il file csv dovevano essere obbligatoriamente numerici, a questo fine si è scelto di assegnare un numero alle tre condizioni di indicizzazione, in particolare 1, 2 e 3, che indicasse l'appartenenza alla categoria indicata dalla stringa di partenza. Il risultato è visibile nell'immagine seguente:

Si può notare come all'interno di ogni riquadro è presente proprio il coefficiente di correlazione di Spearman calcolato sulle due variabili che si incrociano. I risultati ottenuti sono molto chiari, infatti, si può affermare con certezza che non esiste in

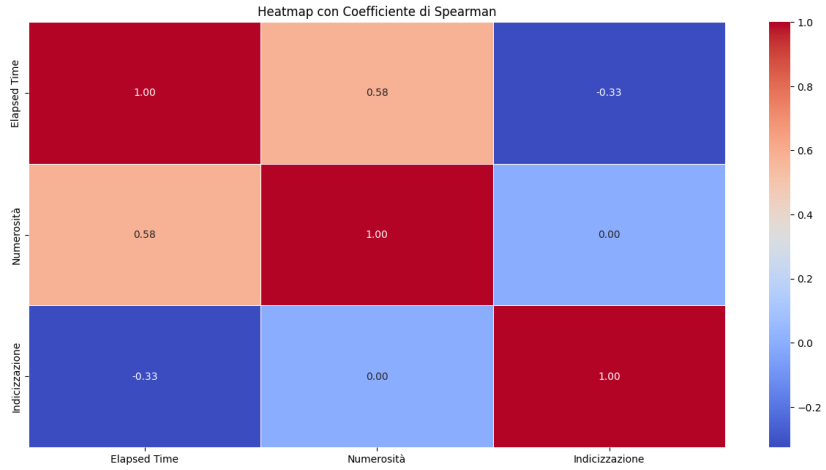

Figura 5.10. Heatmap di elapsed time, numerosità e condizione di indicizzazione.

alcun modo correlazione tra la presenza di indici nella tabella e la numerosità. Sono parametri decisi dall'utente a priori, è normale che non sia mostrata in alcun modo correlazione. Sulla diagonale i valori sono pari a 1 in quanto vengono confrontate le variabili con loro stesse. Tuttavia, sono sottolineati due valori che possono risultare interessanti: il primo è lo 0,58 di correlazione esistente tra la dimensione della tabella e l'elapsed time e il secondo è il -0,33 di correlazione tra la condizione di indicizzazione e l'elapsed time. Prima di approfondire il loro significato bisogna soffermarsi sul concetto di soglia. La soglia rappresenta un valore limite oltre il quale si può affermare con certezza la presenza di un fenomeno di correlazione forte. Il valore in questione, in seguito al contesto e alla letteratura riguardante, è stato posto a 0,7, quindi, con certezza si può affermare che in questo contesto non esistono episodi rispecchianti questa condizione. Tuttavia, se si considera 0,5 come valore soglia per una correlazione moderata e 0,3 per una correlazione debole, ci si può certamente avvalere della facoltà di non escludere che possa esistere una correlazione di tipo negativo tra elapsed time e condizione di indicizzazione

e una correlazione positiva tra elapsed time e numerosità della tabella. La prima dovrebbe indicare concretamente che la relazione tra tempo e condizione di indicizzazione sia direttamente proporzionale in senso negativo, quindi che l'elapsed time tende debolmente a diminuire assieme alla diminuzione di categoria di indicizzazione (muovendo quindi da INDEBTREE a NOINDEX). Riveste invece più risonanza e importanza il dato di 0.58 tra elapsed time e numero di righe della tabella. la correlazione, in base alla scala scelta a priori, risulta moderata, quindi, di gran lunga più apprezzabile rispetto alla prima. La relazione mostra che è presumibilmente presente un legame positivo tra le due variabili, di conseguenza tendono ad aumentare insieme. Il dato trova ampiamente riscontro nell'analisi precedente. Potrebbe essere utile approfondire la debole correlazione negativa presente tra elapsed time e condizione di indicizzazione, anche se in parte potrebbe venire chiarita nel paragrafo successivo riguardante i piani di esecuzione, nel quale verrà mostrato come, anche su grandi numerosità, potrebbe rimanere più performante la Full Scan rispetto all'uso degli indici.

5.4 V$SQL_PLAN

La vista dinamica V$SQL_PLAN contiene informazioni su tutti i piani di esecuzione che al momento corrente sono memorizzati all'interno della cache [25]. Ogni piano di esecuzione è costituito da singole operazioni che servono ad eseguire le varie istruzioni che compongono la query eseguita dall'utente o dal sistema. Ogni riga della vista in questione rappresenta un'operazione all'interno di un piano di esecuzione. Se l'esigenza è quella di visualizzare solamente le informazioni riguardanti l'esecuzione del piano di uno statement particolare, è sufficiente filtrare la ricerca all'interno della vista per SQL_ID. La vista contiene un numero considerevole di informazioni. Innanzitutto, ogni operazione è identificata da un ID e ha una riga interamente dedicata ad essa. Dunque le metriche riferiscono, per ogni singolo

task, il timestamp di quando è stato creato il piano, eventuali opzioni aggiuntive all'operazione, informazioni sulla posizione nell'albero del piano occupata dall'operazione, costo dell'operazione stimato dall'ottimizzatore, cardinalità e bytes delle righe in output all'operazione, costo della CPU, costo di I/O, elapsed time, ID della connessione corrente ed altri parametri identificativi dell'operazione, della query o del piano di esecuzione. Risulta opportuno porre l'attenzione sul fatto che alcune di queste informazioni sarebbero visualizzabili anche in seguito all'esecuzione del comando EXPLAIN PLAN, tuttavia, le due strade differiscono perché con quest'ultima soluzione si visualizza il piano di esecuzione di un'istruzione SQL senza che questa venga obbligatoriamente eseguita, con l 'uso della vista invece viene mostrato il piano per un'istruzione SQL che è già stata compilata in un cursore. Non solo, ci sono rari casi in cui i piani di esecuzione mostrati possono differire, ad esempio in presenza di variabili di associazione, esse vengono prese in considerazione solo nei piani presenti in V$SQL_PLAN [26]. Inoltre, dalla versione Oracle Database 10g, è stata introdotta la V$SQL_PLAN_STATISTICS, riportante la stessa organizzazione di righe della V$SQL_PLAN e contenente statistiche aggiuntive utili principalmente all'ottimizzazione delle prestazioni e al monitoraggio [27]. La sua particolarità è che di default non viene popolata e non è conveniente attivarne la popolazione a livello di sistema (per ogni sessione aperta) per via dell'enorme mole di dati che verrebbero memorizzati e il conseguente rischio di deterioramento delle prestazioni del database. L'unico modo che esiste affinché possa essere utilizzata è l'esecuzione del comando "ALTER SESSION SET SQL_TRACE = TRUE" all'inizio della sessione corrente, che attiva il raccoglimento di tutte le statistiche relative a quest'ambito, da qui fino a fine sessione la vista verrà popolata con le statistiche relative ad ogni operazione di ogni piano di esecuzione relativo a query lanciate dall'utente nella session corrente. Questa vista verrà utilizzata insieme alla V$SQL_PLAN per ottenere tutte le statistiche possibili al fine di comprendere con più efficacia le operazioni che compongono i piani di esecuzione e la loro connessione

con gli indici e con le altre tematiche trattate nell'elaborato.

5.4.1 Analisi sui piani di esecuzione

L'ambito dell'ottimizzazione delle query a partire dai piani di esecuzioni è enorme, in particolare, in questa fase dello studio, è stato utile alla trattazione soffermarsi sul mostrare il processo di miglioramento dei parametri citati in precedenza e di altri in aggiunta mostrando chiaramente il vantaggio che può apportare un corretto utilizzo degli indici.
Prima di tutto si è pensato ad una query che potesse raccogliere più condizioni possibili tra le più impattanti sulle prestazioni del database, prendendo spunto dai dati raccolti nella sezione di V$SQLSTATS.
Il testo della query in questione viene riportato e spiegato di seguito:

```
01 | SELECT a.year, a.month, a.supplier, a.item_code, a.
       item_description, a.item_type,
02 |         a.retail_transfers, a.warehouse_sales, a.id_generico,
        b.retail_transfers
03 | FROM TABELLA_INDEXCLU_INDEXBTREE_100000 a
04 | JOIN TABELLA_INDEXCLU_INDEXBTREE_100000 b
05 | ON a.year = b.year AND a.month = b.month
06 | WHERE a.item_code < 100000 AND a.retail_transfers BETWEEN 10
        AND 2500 AND
07 |       a.warehouse_sales BETWEEN 15 AND 2600
08 | GROUP BY a.retail_transfers, a.warehouse_sales, a.year, a.
       month, a.supplier,
09 |           a.item_code, a.item_description, a.item_type, a.
       id_generico, b.retail_transfers;
```

Come si evince dal testo, la query implementa le seguenti istruzioni:

- Un join tra la tabella in questione e se stessa sulle condizioni di uguaglianza di YEAR e MONTH.

- Attraverso le condizioni di WHERE, utilizza per il join solamente le righe che soddisfano le condizioni di filtraggio elencate al suo interno.
- Il set di righe risultante viene raggruppato secondo i campi indicati nella GROUP BY prima di essere restituito.

Oltre ciò, è stata scritta una query che permettesse di estrapolare solamente i dati utili all'analisi da V$SQL_PLAN e V$SQL_STATISTICS. Prima di mostrarla è importante notare che è stata utilizzata la query citata nell'introduzione di questa sessione affinché fosse attivo il raccoglimento di statistiche a beneficio della V$SQL_PLAN_STATISTICS. La query in questione implementa un join tra le due viste, seleziona i campi di interesse e viene riportata di seguito:

```
01 | SELECT p.operation, p.options, p.id, p.cost, p.cardinality,
       p.bytes, p.cpu_cost,
02 |         s.output_rows, s.elapsed_time
03 | FROM V$SQL_PLAN p
04 | JOIN  V$SQL_PLAN_STATISTICS s
05 | ON p.sql_id = s.sql_id AND p.plan_hash_value = s.
       plan_hash_value AND p.id =
06 |    s.operation_id
07 | WHERE p.sql_id = 'sql_id_di_interesse';
```

Attraverso il join presentato si assicura che le operazioni che compongono il piano di esecuzione del SQL_ID inserito nella WHERE compaiano una sola volta all'interno della tabella restituita come risultato.
In alcuni passaggi verrà mostrato come è possibile in semplici accorgimenti poter migliorare di gran lunga le prestazioni della query sopracitata.
Prima di tutto vengono presentati i dati relativi alla prima esecuzione della query scritta come è stata presentata:

La prima immagine mostra lo schema ad albero del piano di esecuzione, la seconda invece mostra l'output della query di analisi. In aggiunta di seguito vengono

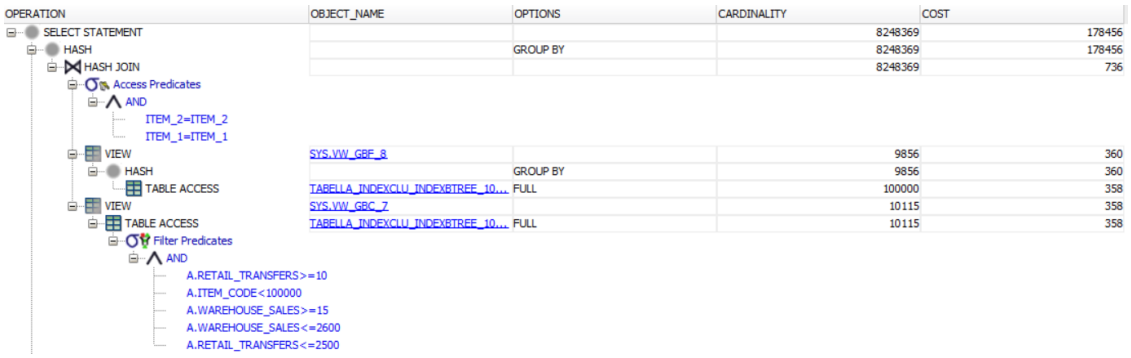

Figura 5.11. Schema ad albero dell'execution plan relativo alla prima esecuzione.

OPERATION	OPTIONS	ID	COST	CARDINALITY	BYTES	CPU COST	OUTPUT ROWS	ELAPSED TIME
HASH	GROUP BY	1	178456	8248369	775346686	12458985408	50	10918150
HASH JOIN		2	736	8248369	775346686	1037780706	4264814	351597
VIEW		3	360	9856	98560	161012679	3308	120816
HASH	GROUP BY	4	360	9856	98560	161012679	3308	120815
TABLE ACCESS	FULL	5	358	100000	1000000	38343329	100000	5960
VIEW		6	358	10115	849660	49852727	10843	8855
TABLE ACCESS	FULL	7	358	10115	849660	49852727	10843	8359

Tabella 5.4. Risultato dell'interrogazione delle viste relativo alla prima esecuzione.

riportate le metriche generali raccolte sulla query per mostrarne in seguito l'effettivo miglioramento generale:

- **Elapsed time totale**: 14809,585 millisecondi.
- **CPU time totale**: 5620,126 millisecondi.

In primo luogo, è utile spiegare che cosa implicano le operazioni nelle immagini. L'ordine con cui vengono svolte le operazioni è chiarito dall'albero:

- Si accede alla prima delle due copie della tabella tramite una TABLE FULL SCAN, quindi viene scansionata l'intera tabella utilizzando i criteri di filtraggio espressi dalla WHERE. La strada alternativa sarebbe stata l'accesso tramite una TABLE INDEX SCAN, che consiste nell'accesso alla tabella tramite l'indice. In questo caso l'ottimizzatore, nella stima dei costi, ha reputato

più utile accedere su tutti i dati perché evidentemente le condizioni di filtraggio avrebbero fatto sì che la ricerca per indice non fosse la più conveniente. Una delle principali metriche su cui presumibilmente si è basato è il peso che l'operazione avrebbe avuto sulla CPU, componente che si occupa di effettuare la scansione della tabella.

- Con i dati reperiti al punto precedente, attraverso l'operazione di VIEW, viene costruita una vista, una tabella temporanea che sarà utile logicamente alla realizzazione della JOIN. Importante notare come i suoi costi sono pressoché molto simili all'operazione di TABLE SCAN.

- A questo punto viene creata la seconda vista sulla stessa tabella di partenza accedendo tramite un ulteriore TABLE FULL SCAN, che in questo caso è sicuramente il metodo migliore, in quanto, in assenza di condizioni di filtraggio, viene richiesto di selezionare l'intera tabella. importante notare che tutto ciò è rispecchiato dall'utilizzo di risorse minori della sequenza precedente, in particolare in termini di costo del lavoro della CPU e di elapsed time. Oltretutto, prima della creazione effettiva della vista, viene implementata una operazione di HASH, cioè di raggruppamento, obbligata dalla presenza all'interno della GROUP BY di un campo da ordinare relativo alla seconda tabella. Quindi l'ottimizzatore reputa più conveniente ordinate subito le righe selezionate dalla seconda tabella secondo il campo richiesto, prima di realizzare la JOIN.

- Successivamente è il momento di implementare la logica della JOIN, riconoscibile facilmente dall'operazione HASH JOIN. Oracle, se non vengono specificate ulteriori informazioni, utilizza di default un INNER JOIN, che restituisce tutte le righe che hanno corrispondenza con le condizioni specificate nella query eseguita, in questo caso le condizioni sono di uguaglianza tra i campi YEAR e MONTH delle due tabelle. L'operazione è estremamente costosa,

è quella che restituisce di gran lunga più righe in output e, anche se l'ultima HASH pesa maggiormente sulla CPU, si può tranquillamente considerare come parte del collo di bottiglia di questa procedura.

- L'ultima operazione prima della restituzione del risultato è la HASH connessa alla maggior parte della GROUP BY della query. Quest'ultima operazione di raggruppamento si basa su tutti i campi della clausola escluso quello sul quale l'operazione è già stata effettuata. Il costo per l'ottimizzatore è espresso dal campo COST del report ed è esorbitante rispetto alle operazioni precedenti. L'elapsed time e la CPU cost sono estremamente alti e ciò significa che la maggior parte del collo di bottiglia risiede proprio in quest'operazione.

- L'ultimo passaggio è il SELECT STATEMENT, rappresentante l'effettiva restituzione del risultato a chi ha lanciato la query. Solitamente questa operazione non costa quasi niente al sistema, l'allocazione delle risorse può essere considerata trascurabile in questo caso.

La prima manipolazione alla query di partenza è stata soffermarsi su come sono state espresse le clausole di WHERE e, successivamente, riconsiderare il loro ordine all'interno del testo. Per esprimere gli intervalli di filtraggio dei campi RETAIL_TRANSFERS e WAREHOUSE_SALES era stata usato inizialmente il BETWEEN, che è stato poi sostituito dall'espressione degli intervalli con i simboli di maggiore e minore. La letteratura spiega che, in generale, l'ottimizzatore è meno invogliato a utilizzare gli indici quando si trova di fronte espressioni come BETWEEN o INTO, quindi tende a tradurre i testi delle query da eseguire in un linguaggio più congeniale al suo. Effettuando a priori questo cambio di sintassi, si è pensato che in generale i parametri temporali potessero migliorare, generalmente ed in modo specifico sulle operazioni interessate nel filtraggio. Inoltre, in funzione dell'ottimizzazione delle prestazioni di esecuzione, risulta estremamente importante l'ordine delle condizioni di filtraggio. Nella scelta dell'ordine delle condizioni di

WHERE da applicare alle tabelle conta molto sia la decisione dell'ottimizzatore, che l'ordine in cui vengono scritte, di conseguenza risulterà ottimizzato un ordine in cui la prima condizione risulta più filtrante delle altre. Per una maggior comprensione verrà analizzato il seguente esempio: se ci sono due condizioni di filtraggio, di cui la prima manda in output 10000 righe e la seconda 50, in funzione del fatto che successivamente i dati verranno ancora manipolati da altre operazioni, conviene che la condizione che restituisce 50 righe sia analizzata per prima, così che poi la condizione che ne avrebbe restituite 10000 può essere applicata solamente alle 50 righe, viene da sé che il contrario comporterebbe che dalla prima condizione escono 10000 righe e poi la condizione più stringente deve analizzare ancora 10000 record per restituire un risultato all'operazione successiva. In funzione di questo concetto, si è proceduto a scrivere tre query che selezionassero sulla tabella in questione il numero di righe in output e le condizioni sono state poi ordinate a partire da quella che ha restituito il valore minore a quella che ha restituito il maggiore. Qui di seguito viene riportato il testo della query modificata:

```
01 | SELECT a.year, a.month, a.supplier, a.item_code, a.
       item_description, a.item_type,
02 |         a.retail_transfers, a.warehouse_sales, a.id_generico,
        b.retail_transfers
03 | FROM TABELLA_INDEXCLU_INDEXBTREE_100000 a
04 | JOIN TABELLA_INDEXCLU_INDEXBTREE_100000 b
05 | ON a.year = b.year AND a.month = b.month
06 | WHERE a.retail_transfers > 10 AND a.retail_transfers < 2500
       AND a.item_code < 100000
07 |        AND a.warehouse_sales > 15 AND a.warehouse_sales <
       2600
08 | GROUP BY a.retail_transfers, a.warehouse_sales, a.year, a.
       month, a.supplier,
09 |           a.item_code, a.item_description, a.item_type, a.
       id_generico, b.retail_transfers;
```

I dati generali sulla query sono notevolmente migliorati e ora corrispondono a:

- **Elapsed time totale**: 8307,854 millisecondi.
- **CPU time totale**: 1602,793 millisecondi.

Si parla di un miglioramento di quasi il 44% rispetto alla prima esecuzione, di conseguenza, la CPU risulta effettivamente scaricata e abbattuto il tempo speso su di essa, in quanto gli accorgimenti miravano proprio a facilitarne il lavoro. Di seguito viene riportato il report delle operazioni del piano di esecuzione, ma non la struttura ad albero, in quanto le operazioni sono rimaste le medesime dell'esecuzione precedente, è possibile solamente apprezzarne un miglioramento dei parametri.

OPERATION	OPTIONS	ID	COST	CARDINALITY	BYTES	CPU COST	OUTPUT ROWS	ELAPSED TIME
HASH	GROUP BY	1	177015	8181501	769061094	12355772365	50	8766976
HASH JOIN		2	736	8181501	769061094	1031091501	4194283	270791
VIEW		3	360	9856	98560	161012679	3308	40559
HASH	GROUP BY	4	360	9856	98560	161012679	3308	40558
TABLE ACCESS	FULL	5	358	100000	1000000	38343329	100000	3218
VIEW		6	358	10033	842772	49850321	10660	7131
TABLE ACCESS	FULL	7	358	10033	842772	49850321	10660	6797

Tabella 5.5. Risultato dell'interrogazione delle viste relativo alla seconda esecuzione.

I cambiamenti salienti si possono facilmente notare nel miglioramento di tutti gli elapsed time a cascata, cambiano le righe in output alla prima TABLE ACCESS e sono, anche se in minima parte, minori. Il costo della CPU così come quello stimato dall'ottimizzatore per ogni operazione non varia molto, ma in generale, considerati soprattutto i dati generali sulla query il miglioramento di prestazione è innegabile. In ultima analisi, si è scelto di considerare il tema degli indici nell'ottimizzazione. Dalla prima esecuzione della query è risultato da un veloce sguardo ai dati come l'operazione di join e il successivo raggruppamento componessero un effettivo collo di bottiglia se si considera il piano di esecuzione come un procedimento seriale. Queste erano le due operazioni che aumentavano esponenzialmente l'elapsed time totale della query. L'accorgimento studiato è stato quello di introdurre una nuova

condizione di uguaglianza nel JOIN così che l'operazione risultasse ulteriormente stringente e, di conseguenza, più veloce. Il termine di uguaglianza è stato posto sul campo SUPPLIER e, ancor prima di poter soffermarsi sulle prestazioni della query scritta in questo modo, si è deciso di porre un ulteriore indice su questo campo, in relazione al concetto di selettività.

È utile in questa fase una piccola digressione esplicativa sul tema della selettività. Il termine rappresenta l'abilità dell'indice di selezionare più valori possibili, di essere utile. La selettività è una frazione tra le righe distinte della colonna che ha implementato l'indice e le righe totali selezionate. Il suo valore varia da query a query poiché viene misurato sul risultato e può variare da 0 a 1. A uno corrisponde che ogni valore è diverso dall'altro e quindi la colonna è perfettamente indicizzata, a 0 ovviamente il contrario [28]. In questo caso, la selettività di un possibile indice sul campo SUPPLIER è stata verificata come in linea con quelle degli altri indici già implementati in precedenza. Dunque, di seguito si riporta la query con l'inserimento delle correzioni appena citate:

```
01 | SELECT a.year, a.month, a.supplier, a.item_code, a.
       item_description, a.item_type,
02 |         a.retail_transfers, a.warehouse_sales, a.id_generico,
        b.retail_transfers
03 | FROM TABELLA_INDEXCLU_INDEXBTREE_100000 a
04 | JOIN TABELLA_INDEXCLU_INDEXBTREE_100000 b
05 | ON a.year = b.year AND a.month = b.month AND a.supplier = b.
       supplier
06 | WHERE a.retail_transfers > 10 AND a.retail_transfers < 2500
       AND a.item_code < 100000
07 |       AND a.warehouse_sales > 15 AND a.warehouse_sales <
       2600
08 | GROUP BY a.retail_transfers, a.warehouse_sales, a.year, a.
       month, a.supplier,
```

```
09 |          a.item_code, a.item_description, a.item_type, a.
       id_generico, b.retail_transfers;
```

Inoltre, vengono riportati, prima le statistiche generali sulla query, e poi il report del piano di esecuzione. Questo perché in questo caso risulta un cambio di operazioni eseguite dall'ottimizzatore.

- **Elapsed time totale**: 1253,611 millisecondi.
- **CPU time totale**: 992,573 millisecondi.

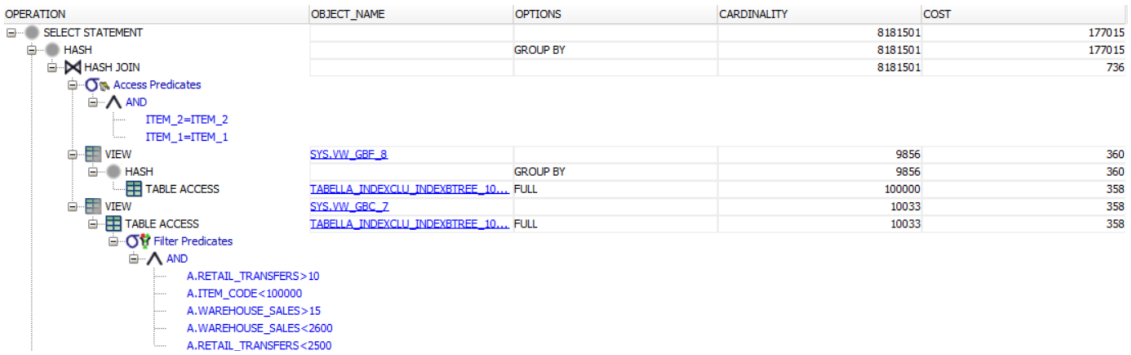

Figura 5.12. Schema ad albero dell'execution plan relativo alla terza esecuzione.

OPERATION	OPTIONS	ID	COST	CARDINALITY	BYTES	CPU COST	OUTPUT ROWS	ELAPSED TIME
HASH	GROUP BY	1	6555	220644	26035992	429800498	50	1231824
HASH JOIN		2	716	220644	26035992	112363001	2583499	360213
NESTED LOOPS		3	716	220644	26035992	112363001	10660	7281
NESTED LOOPS		4					10660	6647
STATISTICS COLLECTOR		5					10660	6010
VIEW		6	358	10033	842772	49850321	10660	5361
TABLE ACCESS	FULL	7	358	10033	842772	49850321	10660	4343
INDEX	RANGE SCAN	8						
TABLE ACCESS	BY INDEX ROWID	9	358	22	748	38343329		
TABLE ACCESS	FULL	10	358	100000	3400000	38343329	100000	13335

Tabella 5.6. Risultato dell'interrogazione delle viste relativo alla terza esecuzione.

Dall'analisi di questo piano di esecuzione, si può notare la presenza dell'operazione STATISTICS COLLECTOR. Consiste nella continua raccolta di informazioni

ad opera dell'ottimizzatore, pratica che non viene eseguita solo a livello statico, ma anche a livello dinamico durante l'esecuzione della query stessa. In questo caso l'ottimizzatore valuta attraverso i NESTED LOOPS se è possibile accedere ad almeno una delle due tabelle tramite un indice. Effettua due tentativi diversi:

- Attraverso lo sfruttamento degli indici sulle colonne YEAR e MONTH, prova ad accedere tramite TABLE ACCESS BY INDEX ROWID, una pratica secondo la quale viene scandito l'indice interamente per mantenere memoria dei valori chiave di riga corrispondenti ai criteri di ricerca, successivamente il database recupera i rowid associati a quei valori. Infine, può accedere alla tabella utilizzando le informazioni appena raccolte.

- Attraverso lo sfruttamento dell'indice sulla colonna SUPPLIER, prova ad accedere attraverso una INDEX RANGE SCAN. L'operazione inizia con la definizione di un intervallo di valori chiave da utilizzare per la ricerca all'interno dell'indice, poi vi è l'individuazione delle voci dell'indice che corrispondono ai criteri di ricerca e infine attraverso questi valori si scandisce la tabella.

Come si può notare dal report, l'ottimizzatore sceglie di non percorrere queste ultime che, infatti, riportano elapsed time uguale a zero. Bisogna ricordare che un elapsed time nullo potrebbe anche indicare un'operazione talmente veloce da riportare un tempo trascurabile ma, in questo caso, non è possibile, in quanto è presente subito dopo la TABLE FULL SCAN scelta. Tuttavia, si può affermare che le prestazioni hanno beneficiato assolutamente della presenza di una nuova condizione di JOIN su colonna indicizzata, in quanto la logica della procedura viene modificata. Rispetto alle esecuzioni precedenti, non vengono più create due viste differenti per poi essere unite dal JOIN, ne viene creata una sola dopo aver raccolto i dati che servivano attraverso le TABLE SCAN. Questo permette anche di non dover più effettuare la prima delle due operazioni di raggruppamento che, seppure meno impattante della seconda, implicava un inevitabile dispendio di risorse.

In generale si può dunque affermare con certezza che attraverso questi passaggi si è migliorato l'elapsed time della query del 91,5 per cento, un'ottimizzazione decisamente considerevole considerando che il tempo speso sulla CPU è circa un quinto di quello di partenza. Inoltre, è molto importante ricordare che con le modifiche apportate i dati che vengono restituiti sono esattamente gli stessi che alla prima esecuzione. Infine, bisogna sempre considerare quali sono le metriche da ottimizzare, infatti, a seguito di questo processo sono presenti sicuramente parametri che al contrario sono aumentati, quindi, si può concludere sul concetto che non esiste un metodo che permetta di migliorare tutti i parametri in gioco allo stesso modo, ma esistono molti metodi che permettono di migliorare le prestazioni a partire dalle esigenze definite in fase di impostazione dell'analisi.
A scopo illustrativo verrà cambiato il testo della query per mostrare come può essere efficiente l'utilizzo degli indici quando il contesto lo permette.
È bastato restringere i range di valori filtrati dalle clausole di WHERE. Il testo della query è riportato di seguito:

```
01 | SELECT a.year, a.month, a.supplier, a.item_code, a.
       item_description, a.item_type,
02 |        a.retail_transfers, a.warehouse_sales, a.id_generico,
        b.retail_transfers
03 | FROM TABELLA_INDEXCLU_INDEXBTREE_100000 a
04 | JOIN TABELLA_INDEXCLU_INDEXBTREE_100000 b
05 | ON a.year = b.year AND a.month = b.month AND a.supplier = b.
       supplier
06 | WHERE a.retail_transfers > 2100 AND a.retail_transfers <
       2500 AND
07 |       a.item_code < 100000 AND a.warehouse_sales > 2000 AND
       a.warehouse_sales < 2600
08 | GROUP BY a.retail_transfers, a.warehouse_sales, a.year, a.
       month, a.supplier,
```

```
09 |          a.item_code, a.item_description, a.item_type, a.
       id_generico, b.retail_transfers;
```

L'albero del piano di esecuzione mostra chiaramente lo sfruttamento degli indici.

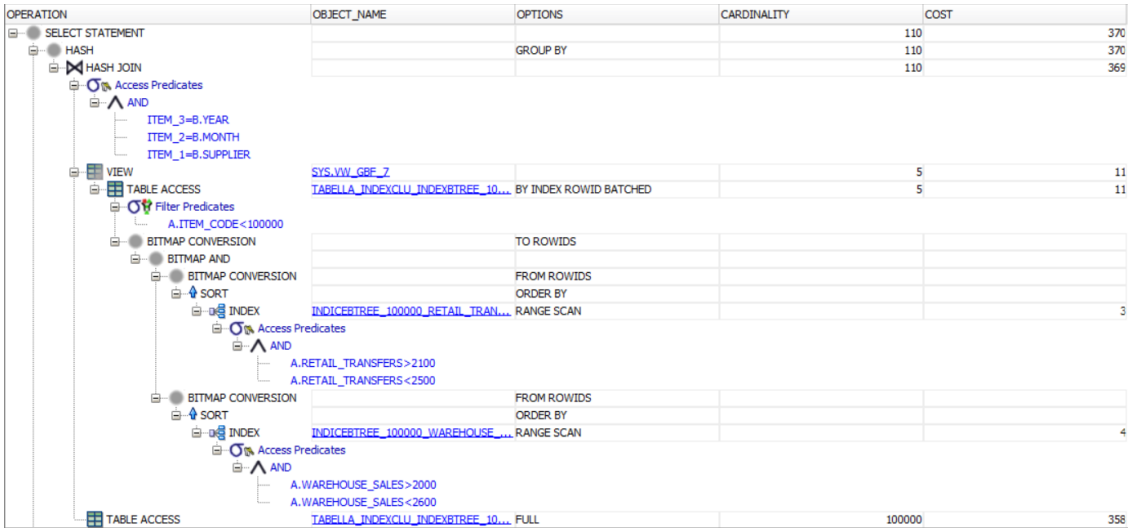

Figura 5.13. Schema ad albero dell'execution plan relativo all'ultima esecuzione.

La logica con cui viene svolta la query è la seguente:

- Per l'accesso alla prima tabella viene usata una normale TABLE FULL SCAN per via del fatto che non ci sono condizioni di filtraggio da rispettare, quindi, risulta più veloce e conveniente recuperare tutte le righe della tabella insieme.

- Per quanto riguarda la seconda tabella, vengono effettuate due INDEX RANGE SCAN, sull'indice relativo a WAREHOUSE_SALES è su quello relativo a RETAIL_TRANSFERS.

- I rowid ottenuti al passo precedente vengono ordinati attraverso le due operazioni di SORT.

- Dopo l'ordinamento, attraverso la BITMAP CONVERSION FROM ROWIDS, ogni rowid viene mappato su un insieme di bit, ognuno rappresentante una riga della tabella. ogni insieme indica se la riga soddisfa o meno la condizione desiderata.

- Con la BITMAP AND vengono scartate tutte le righe che non soddisfano entrambe le condizioni insieme. A livello booleano corrisponde quindi all'intersezione data dall'AND, la riga si mantiene solamente se sono vere allo stesso istante le due condizioni da seguire.

- A seguito di tale operazione, i bitmap vengono riconvertiti a rowid attraverso la BITMAP CONVERSION TO ROWIDS.

- L'accesso ai dati prodotti viene effettuato tramite la TABLE ACCESS BY INDEX ROWID BATCHED, un tipo di accesso che non considera i rowid singolarmente ma a gruppi, diminuendo notevolmente il tempo di risposta.

Il resto del piano rimane invariato, poiché evidentemente continua a rappresentare la via più conveniente da seguire per l'ottimizzatore.
È innegabile l'ottimizzazione che apporta l'utilizzo degli indici nel giusto contesto, testimoniata soprattutto dai dati generali sulla query appena trattata:

- **Elapsed time totale**: 136,418 millisecondi.

- **CPU time totale**: 101,539 millisecondi.

Non è possibile confrontare questi risultati con quelli della prima esecuzione perché chiaramente il numero di righe selezionate è notevolmente differente, ma bisogna soffermarsi sul fatto che a priori, si poteva stimare come più dispendiosa una selezione di dati con condizioni di dati più stringenti rispetto al contrario per via del lavoro che deve incessantemente praticare la CPU. Quest'ultimo esempio testimonia il contrario e ciò è, appunto, reso possibile dalla presenza degli indici.

Tuttavia, si può ancora soffermare l'attenzione su un particolare: è innegabile che l'elapsed time rispetto alle precedenti esecuzioni è diminuito vertiginosamente, ma è altrettanto vero che lo scarto tra questo e il CPU time ha raggiunto livelli quasi trascurabili. Quest'ultimo concetto dà la misura di quanto l'utilizzo degli indici possa aiutare a rendere totalmente meno dispendiose le operazioni di reperimento di dati, pagando d'altra parte il lavoro della CPU nei processi di sfruttamento degli indici.

Capitolo 6

Conclusione

Nell'elaborato è possibile scorgere due filoni principali, che sono stati sviluppati separatamente e permettono di fornire punti di vista differenti, dando un contributo alla tematica principale dello studio che è calato nell'abito della realtà aziendale di Mediamente. La prima area fa capo al problema della frammentazione, al suo significato e alla possibile soluzione pensata per il suo monitoraggio. La seconda corrisponde all'analisi di performance condotta mantenendo colonna portante il tema degli indici e della loro importanza nel contesto di un buon dimensionamento del database e dell'organizzazione dei dati al suo interno. Per questo motivo le conclusioni seguiranno la stessa linea e verranno trattate separatamente per non confondere le argomentazioni trattate.

La **frammentazione**, qualora sia implementata e controllata, non rappresenta un problema, al contrario si pone come un mezzo che può migliorare le prestazioni di un database. La presenza di questa situazione si trasforma in un problema quando non è prevista e tantomeno controllata. In particolare, essendo quest'elaborato incentrato sul tema degli indici, si può affermare che la frammentazione incontrollata di un indice possa portare, anche da sola, ad un rallentamento tale del database da poter essere assimilato ad un down, con conseguente caduta dei servizi interconnessi. Più in generale, come anticipato nel paragrafo riguardante, la gestione

della richiesta di estensione della memoria allocata ad un tablespace rappresenta una richiesta abbastanza comune tra le aziende che affidano la gestione del proprio database Oracle ad aziende di consulenza. Lo script realizzato è un metodo chiaro e consistente che permette al consulente, che deve interfacciarsi con il problema, di avere una visualizzazione chiara delle discontinuità di storage, utilizzabili per ricompattare la memoria occupata creando un risparmio di tempo e risorse non indifferente. È molto semplice immaginare che un database di produzione di cliente potrebbe beneficiare estremamente da questa soluzione, si tratta di una mole innumerevole di dati che quasi certamente vanno incontro a problemi simili. Non è l'unico vantaggio, la tabella e la funzione di visualizzazione di tutti i blocchi che fanno parte dello stesso segmento mediante selezione di uno di questi sono strumenti che facilitano l'analisi di performance del tablespace. Queste features conferiscono una chiara e strutturata visione della situazione di storage all'interno di un tablespace, permettendo di comprendere le quantità di memoria occupate dai vari oggetti presenti in unità di misura Oracle e in classici bytes. Inoltre, assume importanza l'interconnessione che esiste tra la tabella, più analitica, e la griglia, più grafica, che riesce a creare una sinergia utile alla comprensione dell'interezza del tablespace. L'indice di frammentazione è una grandezza pensata ad hoc per il problema presentato e risulta una buona soluzione per comprendere una prima classificazione di quali siano gli oggetti più frammentati che contribuiscono ad accentuare il problema nella generalità del tablespace. A posteriori si nota che l'unica situazione limite della formulazione dell'indice è la seguente: un segmento da otto blocchi Oracle può essere memorizzato con sette blocchi vicini ed uno distante, oppure con tutti i blocchi distanti tra loro, se l'ultimo blocco fosse nella stessa posizione nei due casi, l'indice di frammentazione risulterebbe uguale per come è stato calcolato. È chiaro che la situazione rappresenti un limite perché, nonostante le due situazioni presentino lo stesso indice di frammentazione, la loro effettiva situazione di frammentazione è diversa, sicuramente nel primo caso minore che nel secondo. A

difesa del modello è corretto dire che sia estremamente difficile, se non impossibile, che i segmenti vengano memorizzati tutti in questo modo.

Next steps. In relazione ai limiti palesati sulla formulazione dell'indice di frammentazione, risulta un buon obiettivo da porre quello di studiare un metodo ancor più strutturato utile alla misurazione della frammentazione degli oggetti del database e che possa essere universalmente utilizzata. Riguardo allo script, potrebbe essere utile estendere il calcolo dell'indice e la reportistica dello storage anche rispetto alla totalità dei tablespace presenti in un sistema. L'utilità di una situazione analitica che abbia i tablespace come elementi studiati al posto dei segments è conferita dal fatto che negli ambienti di produzione dei clienti il numero di tablespace presenti può essere numeroso e nel contesto di Anomaly Detection e monitoraggio operati dai consulenti è fondamentale che gli strumenti per visualizzare le situazioni in cui versano i database siano sviluppate sia dal punto di vista quantitativo e sia da quello qualitativo. Inoltre, il processo risulta già in parte automatizzato dall'uso di Python, ma è indubbio che altre soluzioni architettoniche potrebbero migliorare quest'aspetto, ad esempio una web app che, collegandosi in remoto alla macchina di installazione del database, possa gestire tutto il processo, permettendo di reperire tutte le informazioni utili da una console. Quest'ultimo possibile sviluppo prende spunto dall'Enterprise Manager, uno strumento estremamente utile per l'interfaccia che conferisce alle viste Oracle utilizzate nel capitolo di analisi di performance. Infatti, è importante citare che l'Enterprise Manager prevede una vista in cui viene dato un report grafico simile, tuttavia risulta estremamente più difficile da capire ed utilizzare, nonché estremamente più povero di statistiche rispetto a quello realizzato.

L'**analisi di performance** condotta, premesse le assunzioni relative, ha fornito determinati risultati che vanno raccolti e interpretati al fine di poter trovare nuovi modi di indagare le stesse situazioni o perfino di trovare situazioni nuove da poter analizzare.

L'elapsed time totale delle query si è dimostrato essere estremamente variabile. A priori il risultato atteso da questo punto di vista era che l'elapsed time si sarebbe rivelato crescente proporzionalmente alla crescita del numero di righe delle tabelle interrogate, tanto è vero che non era questo l'aspetto scelto da studiare. Tuttavia, sembra giusto annoverarlo tra i risultati reperiti e provare a ricercarne una causa. Presumibilmente la variabilità in questione dipende dai diversi componenti che partecipano insieme alla costituzione della metrica, a partire dalla variabilità legata alla comunicazione di rete sfruttata dal database durante il suo funzionamento, le reti aziendali sono spesso affette da problemi di concorrenza per via dei diversi processi che gli utenti eseguono durante una normale giornata lavorativa. Non vi è solo questo, il CPU time è parte dell'elapsed, come si è notato dalla definizione, nonostante ciò, non sembra soggetto al particolare fenomeno sopra citato. Premesso che nell'analisi appena conclusa quasi tutti i componenti temporali dell'elapsed time sono stati normalizzati a zero, presumibilmente sono le operazioni di I/O ad essere insieme alla rete, causa del fenomeno. Inoltre, risulta possibile che l'assunzione di annullare l'effetto della cache, attuata per dare consistenza ai risultati, da un altro punto di vista possa aver creato variabilità.

L'indice clustered risulta essere estremamente importante nel contesto di realizzazione di una tabella ben dimensionata. Quando il campo con chiave primaria è interessato dalle clausole della query le prestazioni temporali migliorano notevolmente, a prescindere da quali clausole si utilizzano e soprattutto da quale sia la dimensione della tabella. In generale si può affermare che il contributo ottimizzante dell'indice clustered sia maggiore rispetto a quello dell'indice non-clustered, quando trattati singolarmente. È pur vero che non è possibile assegnare più chiavi primarie ad una tabella quindi, in funzione dei miglioramenti di prestazioni apportati dagli indici non-clustered, è comunque ampiamente consigliabile una struttura di indicizzazione ibrida. Quest'ultima è la soluzione migliore in funzione di una visione ampia sulla totalità dei risultati ottenuti, tenendo conto del fatto che un database

di produzione a pieno regime si trova a dover eseguire svariati tipo di query di continuo senza poter beneficiare di momenti in cui vengono riprodotte situazioni silenti come quella del database preso ad esempio. L'indice non-clustered invece mostra una tendenza importante a convenire soprattutto ad alte numerosità, ciò significa che ne vengono sfruttati i benefici dall'ottimizzatore soprattutto quando i valori da leggere per costruire il risultato sono molti. Quello che si faceva più fatica ad aspettarsi, è stata la sfumatura di non convenienza a basse numerosità. A monte dell'analisi si poteva immaginare che al più i tempi con e senza indici si potessero equiparare, invece, non è così, i tempi degli indici risultano in generale più alti. Questo scarto risulta variabile con il tipo di query ma costante nella sua presenza e la causa è certamente da imputare al fatto che il lavoro di lettura dell'indice viene svolto dalla CPU, quindi, quando il tempo che serve al sistema per leggere i dati senza indici è minore di quello che servirebbe a leggere gli indici cambia il trend di convenienza dell'uso di questi tipi di oggetti. È proprio in quest'ottica che si è cercato di isolare il range di valori nel quale la tendenza si inverte e si può affermare con certezza che il risultato trovato potrebbe essere soggetto a cambiamenti se cambiassero le condizioni al contorno, soprattutto il tipo di query.

L'hint suggerito all'ottimizzatore è un metodo da non sottovalutare in ambito di monitoraggio del lavoro del database e anche nel contesto dell'ispezione di errori o anomalie. È stato dimostrato che in alcuni casi potesse essere più utile l'utilizzo degli indici nella soluzione ad una query che il contrario senza il suggerimento di utilizzo. Questo trova spiegazione nel modello di costi che l'ottimizzatore applica per decidere quale sia il piano di esecuzione migliore al raggiungimento dell'obiettivo. Tuttavia, quest'analisi si concentrava su alcune metriche e non su tutte, questo mostra quanto sia importante capire e fissare i target reali, dunque quali siano le metriche più importanti al momento dell'approccio all'analisi, se questa non le comprende tutte. La funzione di costo dell'ottimizzatore si basa invece sulla totalità delle risorse, cercando in ogni modo di garantire le migliori possibili nel pieno di

risparmio di ognuna di esse. È chiaro come nella normale attività di un database in funzione tutto ciò rimanga comunque fondamentale.

Il join risulta un'operazione estremamente pesante dal punto di vista computazionale, è per questo motivo che si è scelto di analizzarlo in maniera molto dettagliata considerandone uno studio ulteriore sui piani di esecuzione. In particolare, dalla prima fase dell'analisi si nota come quest'operazione risulti insostenibile a grandi numerosità senza una buona struttura di indicizzazione. Questo mostra il beneficio che gli indici portano generalmente alle esecuzioni delle query, infatti, anche laddove non vi sono espressamente operazioni che li riguardano, molte altre operazioni, quali ordinamenti, raggruppamenti e operazioni di join o di aggregazione, ne beneficiano largamente.

Dallo studio di ottimizzazione a partire dai piani di esecuzione, risulta estremamente chiara la condizione nella quale gli indici possano essere sfruttati a meglio. Si è dimostrato che gli indici risultano fortemente ottimizzanti quanto più le condizioni di filtro sono stringenti. In generale, più il numero di righe da reperire si avvicina alla totalità della tabella interrogata e più l'ottimizzatore è invogliato ad agire in FULL SCAN, probabilmente perché, quando lo scarto tra i due valori rimane in certi limiti, risulta più conveniente leggere l'intera tabella e applicare le condizioni di filtro al posto che leggerne solo una parte per indici. Evidentemente il costo computazionale non varrà il vantaggio di lettura.

Le modalità con cui viene scritta la query sono fondamentali: devono essere scritte tenendo in considerazione di diminuire nel maggior modo possibile le viste di appoggio che vengono generate durante lo svolgimento dal sistema e oltre ciò di diminuire il più possibile la mole di righe che queste si portano dietro. Assume particolare importanza l'ordine delle condizioni di WHERE, un aspetto che, come spiegato nel paragrafo interessato, può migliorare notevolmente i parametri prestazionali di un sistema nell'eseguire uno statement.

Next steps. Prima di affrontare qualsiasi tipo di riflessione relativa ad approfondimenti futuri è importante soffermarsi su quanto possano essere importanti i risultati ottenuti in entrambe le fasi di analisi in relazione all'ambiente creato mediante le assunzioni di partenza. Si può affermare con certezza di aver realizzato un ambiente totalmente utopico rispetto a quello reale proprio di qualsiasi database utilizzato in contesto aziendale. Soprattutto i risultati ottenuti nell'analisi dei piani di esecuzione risultano fondamentali quando vengono rilassate le due assunzioni più importanti: l'utilizzo della cache e l'utilizzo di un database quasi completamente silente. Generalmente la cache è uno strumento pensato proprio in ottica di ottimizzazione di prestazioni, basti pensare che, quando una query viene eseguita una seconda volta su un database, il suo elapsed time può scendere vertiginosamente solamente grazie al lavoro silente della cache. Non solo, risulta fondamentale tenere a mente i concetti appresi per poter evitare il sovraccarico del database in un suo quotidiano contesto produttivo, quindi, con numerosissimi processi concorrenti che rischiano in ogni momento di bloccare il normale funzionamento. In questo contesto, la capacità di diminuire l'elapsed time di una query del 100 % circa, significa poter scongiurare qualsiasi situazione emergenziale dal punto di vista della disponibilità di risorse e dal punto di vista temporale.
La prima parte di analisi, relativa a V$SQLSTATS potrebbe essere ripetuta prendendo in considerazione altre metriche, come la memoria utilizzata, oppure potrebbe essere ripetuta in merito a più esecuzioni delle stesse query, studiando l'eventuale nuova entità del rapporto tra metriche e cambio di dimensione della tabella sotto la condizione di effettuare numerose esecuzioni delle stesse query. Oltre ciò, potrebbe risultare estremamente interessante, grazie ai risultati ottenuti con V$SQL_PLAN, capire se e come si modificherebbero i risultati concentrandosi sulle variazioni dei parametri e delle condizioni di filtro all'interno delle clausole WHERE. In relazione alla questione dei piani di esecuzione, scaturisce l'enorme importanza dello strumento utile all'ottimizzazione delle prestazioni. La divisione delle esecuzioni della

query in task permette l'analisi delle prestazioni al livello più dettagliato possibile, ottimizzando direttamente le strutture atomiche che compongono il processo per arrivare al risultato. Data la mole di informazioni che il sistema deve memorizzare, per poter effettuare un'analisi a livello di piano di esecuzione potrebbe risultare molto utile, in contesti aziendali simili a Mediamente, la realizzazione di una struttura dotata di interfaccia interamente dedicata all'analisi dei piani di esecuzione, in cui sia prevista un'interfaccia ben strutturata che permetta in modo semplice di seguire puntualmente tutte le evoluzioni che segue un SQL_ID. In parte tutto ciò viene realizzato dall'Enterprise Manager, ma questo strumento non è ottimizzato mettendo al primo posto il piano di esecuzione ed è comprensibile, non essendo il suo obiettivo. In generale, rispetto a quanto studiato, un nuovo spunto di analisi viene da qualsiasi tipo di query lanciato in qualsiasi tipo di contesto diverso, ogni situazione può manifestare una possibile situazione da studiare a livello di execution plan, alcune per effettive anomalie, altre per migliorare prestazioni già ottime in partenza. Il tutto sempre mantenendo in estrema considerazione il ruolo che gli indici possono ricoprire in questo panorama. Lo studio ha testimoniato come un semplice indice in un campo di una tabella può influenzare significativamente il comportamento dell'ottimizzatore, il che obbliga, non solo a prevederne sempre l'utilizzo in fase di dimensionamento, ma soprattutto a farne un uso sempre oculato e ottimizzato, senza dimenticare di effettuarne una continua e corretta manutenzione, affinché sia puntualmente nelle condizioni di miglior efficienza.

Bibliografia

[1] Oracle US, "Che cos'è un database relazionale (rdbms)?," https://www.oracle.com/it/database/what-is-a-relational-database/.

[2] V. Lavecchia, "Differenza tra database e istanza in informatica," https://vitolavecchia.altervista.org/differenza-tra-database-e-istanza-ininformatica/#:~:text=In%20informatica%2C%20l'istanza%20%C3%A8,contiene%20la%20raccolta%20di%20file.

[3] M. Mangione, "Oracle - un sistema operativo," 2011. http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/ingegneriabiomedica/informaticamedica/inf.med.oracle.pdf.

[4] M. Gertz, "Oracle/sql tutorial," 2000. https://www.brescianet.com/appunti/db/oraclesqltutorial.pdf.

[5] S. Mola, "L'architettura di oracle," 2006. https://www.html.it/pag/16910/l-architettura-di-oracle/.

[6] Oracle US, "Introducing oracle database cache," 2001. https://docs.oracle.com/cd/A97335_02/caching.102/a88706/ic_intro.htm#:~:text=Oracle%20Database%20Cache%20maintains%20a%20history%20of%20queries%20and%2C%20based,a%20particular%20middle%2Dtier%20cache.

[7] Sanjaybali, "Oracle database provides two type of metadata views," 2019. https://sanjaybali.wordpress.com/2019/08/17/data-dictionary-and-dynamic-performance-views/.

[8] C. Cudizio, "Ottimizzatore delle query in oracle 12c," 2018. https://cristiancudizio.wordpress.com/2018/04/27/ottimizzatore-delle-query-in-oracle-12c/#:~:text=Il%20%E2%80%9Cquery%20optimizer%E2%80%9D%2C%20chiamato,il%20%E2%80%9Cpiano%20di%20esecuzione%E2%80%9D.

[9] R. West, "Panoramica del piano di esecuzione," 2023. https://learn.microsoft.com/it-it/sql/relational-databases/performance/execution-plans?view=sql-server-ver16.

[10] Oracle US, "The query optimizer," 2011. https://docs.oracle.com/cd/E15586_01/server.1111/e16638/optimops.htm.

[11] "Oracle–ottimizzazione delle query–explain plain," 2013. https://matadeveloper.wordpress.com/2013/07/24/oracle-ottimizzazione-delle-query-explain-plain/.

[12] A. Bhat, "Scala ordinale: Definizione, livello di misura ed esempi," https://www.questionpro.com/blog/it/scala-ordinale/.

[13] K. Nath, "Uno sguardo approfondito all'indicizzazione del database," 2023. https://www.freecodecamp.org/italian/news/indicizzazione-database/.

[14] "Frammentazione," https://it.frwiki.wiki/wiki/Fragmentation_(informatique).

[15] The Matplotlib development team, "Matplotlib 3.8.3 documentation," 2012. https://matplotlib.org/stable/index.html#learn.

[16] M. Waskom, "Seaborn," 2012. https://seaborn.pydata.org/.

[17] "Pandas," 2012. https://pandas.pydata.org/.

[18] O. US, "V$sqlstats," https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SQLSTATS.html#GUID-495DD17D-6741-433F-871D-C965EB221DA9.

[19] C. McDonald, "Use v$sqlstats not v$sql to find expensive sql," 2019. https://connor-mcdonald.com/2019/03/04/less-slamming-vsql/.

[20] Burleson Consulting, "Oracle elapsed time tips," 2020. https://www.dba-oracle.com/m_sql_execute_elapsed_time.htm.

[21] DevX, "Definition of cpu time," 2023. https://www.devx.com/terms/cpu-time/.

[22] P. Pozzolo, "La correlazione lineare r di pearson," 2020. https://paolapozzolo.it/coefficiente-correlazione-statistica-pearson/.

[23] P. Pozzolo, "Coefficiente di correlazione di spearman: quando si usa?," 2020. https://paolapozzolo.it/coefficiente-correlazione-statistica-spearman/.

[24] D. Soriano, "Che cosa è la matrice di correlazione? come si costruisce in python?," 2021. https://www.domsoria.com/2021/09/che-cosa-e-la-matrice-di-correlazione-come-si-costruisce-in-python/.

[25] Oracle US, "V$sql_plan," https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SQL_PLAN.html#GUID-87561B21-721C-42EB-8E3D-28251C9BC50C.

[26] Shalabh, "Difference in explain plan – explain plan for vs v$sql_plan," 2020. https://dbatracker.com/2020/01/27/difference-in-explain-plan-explain-plan-for-vs-vsql_plan/.

[27] Oracle US, "V$sql_plan_statistics," https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SQL_PLAN_STATISTICS.html#GUID-983DE0B1-1824-4A03-9C96-DCAFF5662B1A.

[28] M. N. Huda, "Index vs full table scan," 2022. https://www.nazmulhuda.info/index-vs-full-table-scan.