



UNIVERSITÀ DEGLI STUDI DI VERONA

Facoltà di Scienze e Ingegneria

Corso di Laurea in Informatica

OTTIMIZZAZIONE FUNZIONI SAP-ABAP IN AMBITO INDUSTRIALE

Relatore:

Prof. Carlo Combi

Laureando:

Thomas Remelli

ANNO ACCADEMICO 2017 - 2018

RINGRAZIAMENTI...

Desidero ringraziare il prof. Carlo Combi, relatore di questa tesi, per la grande disponibilità e cortesia dimostratemi, e per tutto l'aiuto fornito durante la stesura.

Un sentito ringraziamento ai miei genitori, che, con il loro incrollabile sostegno morale ed economico, mi hanno permesso di raggiungere questo importantissimo traguardo.

Desidero inoltre ringraziare la ditta Marfin srl, tutti i ragazzi dello staff di ricerca e sviluppo, e in particolare Bruno e Andrea, per tutto quanto hanno fatto per me durante il periodo di stage.

Un altro ringraziamento ai compagni di studi, per essermi stati vicini sia nei momenti difficili, sia nei momenti felici: sono stati per me più veri amici che semplici compagni.

Un ultimo ringraziamento a Valentina e tutti miei amici che mi hanno sempre sostenuto ed incoraggiato in questo bellissimo percorso della mia vita.

INDICE

1. INTRODUZIONE	3
1.1 SAP HANA	3
1.2 ABAP	5
1.3 SQL	6
2. OBIETTIVO	7
2.1 OBIETTIVO PRINCIPALE	7
2.2 SPECIFICHE FUNZIONE	8
3. DEFINIZIONI	11
3.1 AMDP	11
3.2 CALCULATION VIEW	12
4. OTTIMIZZAZIONI PRINCIPALI	13
4.1 TIPS & TRICKS	13
4.2 CONSIDERAZIONI	15
5. ANALISI FUNZIONE	16
5.1 ANALISI GENERALE FUNZIONE	16
5.2 ANALISI DETTAGLIATA TEMPI	18
5.2.1 ANALISI TEMPI PROVA 1	19
5.2.1 ANALISI TEMPI PROVA 2	21
5.2.1 ANALISI TEMPI PROVA 3	22
6. ANALISI SOLUZIONI FUNZIONE	23
6.1 ANALISI POSSIBILI SOLUZIONI	23
6.2 DETTAGLI SOLUZIONI	25

7. FASE DI OTTIMIZZAZIONE	28
7.1 PIANIFICAZIONE	28
7.2 INDIVIDUAZIONE DEI PUNTI DA	30
7.3 OTTIMIZZAZIONI PRINCIPALI	31
7.4 ESITI OTTIMIZZAZIONE	42
7.4.1 ESITI PROVA 1	42
7.4.1 ESITI PROVA 2	43
7.4.1 ESITI PROVA 3	44
8. FASE DI TEST	45
8.1 INTRODUZIONE	45
8.2 STRUTTURA PROGRAMMA	46
8.3 DEBUGGING	49
8.4 RISULTATI TEST	50

1. INTRODUZIONE

1.1 SAP HANA

SAP HANA è una piattaforma di gestione di basi di dati colonna e in-memory, sviluppato e commercializzato dalla società tedesca SAP (Systeme Anwendungen Produkte) che permette di processare grandi volumi di dati in tempo reale.

Molte aziende hanno a che fare con enormi volumi di dati provenienti da diverse fonti che hanno la necessità di essere processati in real-time, a cui si aggiunge anche la necessità di dover ottenere tempi di risposta molto veloci, aumentando quindi la complessità di analisi.

Per soddisfare i requisiti appena citati, SAP HANA risulta un software adatto.

SAP HANA nasce dall'acquisizione di alcune tecnologie, tra le quali il motore di ricerca column oriented TREX, la piattaforma OLTP in-memory P*TIME e Max DB con il suo motore in-memory live Cache.

I dati da elaborare ora risiedono in memoria ram e non più nell'hard disk : questo porta ad una velocità notevolmente maggiore.

Il cuore di SAP HANA è rappresentato dal database HANA che risulta essere diverso dagli altri database.

Il database è sviluppato in C++ ed è in esecuzione su Suse Linux Enterprise Server.

Esso è costituito da elementi hardware e software che sfruttano il calcolo in-memory: unisce la tecnologia di database row-based e column-based.

I principali fattori che caratterizzano SAP sono l'elevato livello di sicurezza dell'ambiente di produzione e la portabilità, infatti esso funziona su qualunque sistema operativo e su qualunque database di mercato.

1.2 ABAP

Una parte integrante di SAP è il linguaggio di programmazione ABAP/4 (Advanced Business Application Programming).

Questo semplice linguaggio permette agli sviluppatori di creare grandi e piccole applicazioni destinate agli utenti finali: fornisce la possibilità di creare completamente una nuova applicazione Client/Server o di implementare nuovi moduli a quelli preesistenti.

I vantaggi di un linguaggio di programmazione non sono evidenti solo per la funzionalità del linguaggio stesso, ma anche per l'ambiente di sviluppo che lo supporta.

ABAP/4 è lo strumento centrale di un eccellente ambiente di sviluppo chiamato ABAP/4 DEVELOPMENT WORKBENCH, il quale garantisce un efficiente processo di sviluppo per ogni tipo di applicazione distribuita su un ambiente Client/Server.

ABAP/4 Development Workbench è stato progettato per poter utilizzare i seguenti strumenti:

- L'Object Browser
- L'editor del linguaggio ABAP/4
- Il dizionario dati
- Il Repository (Screens e struttura delle tabelle)
- Il Data modeler
- Ambiente di test dei programmi

1.3 SQL

Il linguaggio SQL (Structured Query Language) è stato definito negli anni '70 ed è stato standardizzato negli anni '80 e '90 per diventare successivamente il linguaggio più diffuso per i DBMS (Database Management System) relazionali dei giorni nostri.

SQL è un linguaggio di interrogazione dichiarativo e la sua semantica fa riferimento al CALCOLO RELAZIONALE.

Il linguaggio SQL si può suddividere in quattro sezioni:

- Linguaggio per la definizione delle strutture dati e dei vincoli di integrità: Data Definition Language (DDL).
- Linguaggio per manipolare i dati: inserimento, aggiornamento e cancellazione: Data Manipulation Language (DML).
- Linguaggio per interrogare: Data Query Language (DQL o QL).
- Linguaggio per controllare la base di dati: Data Control Language (DCL).

2. OBIETTIVO

2.1 OBIETTIVO PRINCIPALE

Lo scopo di questo progetto è quello di ottimizzare i tempi di esecuzione di una funzione che viene chiamata dell'utente per ottenere dei dati dal database.

Il miglioramento di questa funzione può considerarsi effettuato quando i tempi di esecuzione vengono ridotti in modo significativo per l'utente finale, ma bisogna anche verificare attraverso alcuni test che i dati estratti siano effettivamente gli stessi che vengono ricavati dalla funzione originale.

In questo caso la funzione è scritta in linguaggio ABAP/4 e deve ricavare delle informazioni da SAP mediante l'utilizzo di Query scritte in SQL e in ABAP/4.

2.2 SPECIFICHE FUNZIONE

La funzione che verrà analizzata nel progetto si chiama Z_EXTRACT_VIAGGI_01C.

La sua applicazione è nell'ambito della comunicazione con sistemi esterni (comunemente detti sistemi di fabbrica) a SAP che richiedono informazioni riguardanti il viaggio stesso. Per farlo fanno uso di RFC (Remote Function Call) per richiamare dall'esterno la funzione.

L'utente prima dell'esecuzione può specificare alcuni parametri per limitare la ricerca solo ad alcuni viaggi con determinate caratteristiche.

Questi parametri sono:

- Divisione
- Luogo di spedizione
- Numero di viaggio
- Tipo esecuzione
- Stato
- Distinta

Una volta inserite le specifiche desiderate, l'utente può eseguire la funzione che, una volta terminata, potrà visualizzare le tabelle contenenti i dati inerenti ai viaggi relativi ai parametri inseriti come input.

Nel caso non venga specificato nessun valore, la funzione verrà eseguita per tutti i viaggi presenti nel database di SAP e un lancio di questo tipo bloccherebbe il sistema.

Nell'immagine seguente viene raffigurata la schermata di specifica dei parametri da parte dell'utente.

Parametro import.	Valore
P_WERKS	
P_VSTEL	
STATO	
VIAGGIO	6901453
DISTINTA	
TRAID	
ZPRVI	0000
LFSNR	
TIPO_ESEC	VS
P_TIPO_ESTRAZIONE	
P_WORKSTATION	
P_ESTRAI_CODA	N
P_FILE_CREATE	N
P_FILE_TESTATA	ZTESTATAVIAGGIO
P_FILE_POS	ZPOSVIAGGIO
P_FILE_NOTE	ZNOTETPOSVIAGGIO
P_FILE_CHARG	ZCHARGVIAGGIO
P_FILE_CARA	ZCARAVIAGGIO
P_PULIZIA	N
P_FILLHISTORYERROR	S
P_FILTRA_LOG	I

Alcuni di questi parametri vengono utilizzati per l'estrazione di dati supplementari mentre altri sono parametri di default che non sono modificabili dall'utente. I campi P_WERKS e P_VSTEL indicano rispettivamente la divisione e il luogo di spedizione delle merci.

Tabelle	Valore
T_TESTATAVIAGGIO	0 Ins.
	Risultato:
T_POSVIAGGIO	1 Ins.
	Risultato:
T_POSVIAGGIO1	0 Ins.
	Risultato:
T_CARAVIAGGIO	0 Ins.
	Risultato:
T_NOTEPOSVIAGGIO	0 Ins.
	Risultato:
T_NOTESESVIAGGIO	875 Ins.
	Risultato:
T_CHARGVIAGGIO	0 Ins.
	Risultato:
T_PACCOVIAGGIO	33 Ins.
	Risultato:
T_ZSGTP_CHARG	0 Ins.
	Risultato:
T_ZSGTP_PAL	522 Ins.
	Risultato:
T_POSVIAGGIO_ALTRE	0 Ins.
	Risultato:
T_TESTATAVIAGGIO_ALTRE	0 Ins.
	Risultato:
T_ZSGTP_CHARG_ALTRE	1 Ins.
	Risultato:
	519 Ins.

Una volta terminata l'esecuzione verranno restituite una serie di tabelle ciascuna contenente determinate informazioni sul viaggio. In questa videata utilizzata per i test è possibile ottenere e analizzare ciò che l'utente finale vede. Cliccando successivamente su una tabella si può visualizzare il suo contenuto come raffigurato nella figura seguente.

ZZXABLN	ZZPPOSN	NFALL	CHARG	WERK	LGOR	UBICA	PIRGMG	GIACE	MATNR
0000000006897485	000010	17C7020424	17C7020424	0103	CAMP4B		294.00000	1155.00000	58002196
0000000006897485	000010	17C7020425	17C7020425	0103	CAMP4B		294.00000	1160.00000	58002196
0000000006897485	000010	17C7020426	17C7020426	0103	CAMP4B		294.00000	1161.00000	58002196
0000000006897485	000010	17C7020427	17C7020427	0103	CAMP4B		294.00000	1154.00000	58002196
0000000006897485	000010	17C7020428	17C7020428	0103	CAMP4B		294.00000	1155.00000	58002196
0000000006897485	000010	17C7020429	17C7020429	0103	CAMP4B		294.00000	1156.00000	58002196
0000000006897485	000010	17C7020450	17C7020450	0103	CAMP4B		294.00000	1156.00000	58002196
0000000006897485	000010	17C7020451	17C7020451	0103	CAMP4B		294.00000	1157.00000	58002196
0000000006897485	000010	17C7020452	17C7020452	0103	CAMP4B		294.00000	1156.00000	58002196
0000000006897485	000010	17C7020453	17C7020453	0103	CAMP4B		294.00000	1156.00000	58002196
0000000006897485	000010	17C7020454	17C7020454	0103	CAMP4B		294.00000	1155.00000	58002196
0000000006897485	000010	17C7020455	17C7020455	0103	CAMP4B		294.00000	1156.00000	58002196
0000000006897485	000010	17C7020465	17C7020465	0103	CAMP4B		294.00000	1146.00000	58002196
0000000006897485	000010	17GC013127	17GC013127	0103	CAMP11A		294.00000	1172.00000	58002196
0000000006897485	000010	17GC013128	17GC013128	0103	CAMP11A		294.00000	1172.00000	58002196
0000000006897485	000010	18P5000769	18P5000769	0103	CAMP11A		294.00000	1163.00000	58002196
0000000006897485	000020	17CT000807	17CT000807	0103	CAMP29		1254.00000	1850.00000	58001672
0000000006897485	000020	17CT000883	17CT000883	0103	CAMP29A		900.00000	1300.00000	58001672
0000000006897485	000020	17CT000884	17CT000884	0103	CAMP29A		900.00000	1300.00000	58001672
0000000006897485	000030	17C8015278	17C8015278	0103	CAMP2B		300.00000	563.00000	58001676
0000000006897485	000030	17C9013027	17C9013027	0103	CAMP3A		762.00000	1377.00000	58001676
0000000006897485	000030	17C9013029	17C9013029	0103	CAMP3A		762.00000	1388.00000	58001676
0000000006897485	000030	17C9013030	17C9013030	0103	CAMP3A		762.00000	1379.00000	58001676
0000000006897485	000030	17C9013031	17C9013031	0103	CAMP3A		762.00000	1383.00000	58001676
0000000006897485	000030	17C9013032	17C9013032	0103	CAMP3A		762.00000	1389.00000	58001676
0000000006897485	000030	17C9013063	17C9013063	0103	CAMP3A		762.00000	1380.00000	58001676
0000000006897485	000030	17C9013071	17C9013071	0103	CAMP3B		762.00000	1376.00000	58001676
0000000006897485	000030	17C9013072	17C9013072	0103	CAMP3B		762.00000	1379.00000	58001676
0000000006897485	000030	17C9013073	17C9013073	0103	CAMP3B		762.00000	1375.00000	58001676
0000000006897485	000030	17C9013074	17C9013074	0103	CAMP3B		762.00000	1365.00000	58001676
0000000006897485	000030	17C9013075	17C9013075	0103	CAMP3B		762.00000	1374.00000	58001676

3. DEFINIZIONI

3.1 AMDP

Le AMDP (ABAP Managed Database Procedure) sono delle nuove funzionalità di ABAP che permettono agli sviluppatori di scrivere delle procedure più efficienti.

Questa procedura può essere vista come una funzione memorizzata ed eseguita direttamente nel database.

Il linguaggio di implementazione varia da un database ad un altro: nel caso di questo progetto con SAP HANA il linguaggio utilizzato è SQLScript.

I vantaggi principali riguardano le prestazioni: un database ha sicuramente una potenza di calcolo superiore rispetto ad un normale PC, quindi i tempi di esecuzione saranno di conseguenza inferiori. A migliorare ulteriormente le prestazioni vi è il minor numero di chiamate al database.

Le AMDP sono un ibrido di due linguaggi: si tratta infatti di un oggetto nel quale le dichiarazioni di variabili e di metodi sono scritte in ABAP mentre le loro implementazioni sono in SQLScript.

3.2 CALCULATION VIEW

Le Calculation View sono un tool grafico di Eclipse che permette di creare un codice in SQL nativo.

Esistono cinque tipi di operazioni possibili:

- Proiezione, per selezionare solamente alcune colonne di una tabella
- Join, per unire due o più tabelle sotto determinate condizioni di uguaglianza
- Aggregazione, per calcolare somme, massimi, minimi o medie relativi ad alcuni valori in una tabella
- Ordinamento relativo alle colonne specificate
- Unione di due o più tabelle

Il vantaggio delle Calculation View sta sia nella semplicità di creazione di codice per via grafica, che nell'elevata velocità di esecuzione di un programma implementato in SQL nativo.

Naturalmente a livello di tempo è più dispendioso creare un codice per via grafica rispetto ad una query SQL direttamente nell'ambiente ABAP, ma quando la complessità risulta molto elevata le Calculation View risultano essere la soluzione ideale.

4. OTTIMIZZAZIONI PRINCIPALI

4.1 TIPS & TRICKS

SAP HANA possiede una sezione denominata “Tips & Tricks” contenente parecchi consigli su come ottimizzare dei programmi attraverso numerosi esempi.

The screenshot shows two code snippets in SAP HANA Studio. Both snippets are titled "Select ... Where + Check" and "Select using an aggregate function". The first snippet has a runtime of 3.360 Microsecondi and the second has a runtime of 932 Microsecondi. Both snippets are identical, showing a SELECT statement that retrieves the maximum value from a table T100 into a variable MAX_MSGNR, filtered by SPRSL = 'D' and ARGBB = '00'. A CHECK clause ensures that the value in T100_WA-MSGNR is greater than MAX_MSGNR, and then updates MAX_MSGNR to T100_WA-MSGNR. The second snippet uses an aggregate function MAX(MSGNR) instead of a WHERE clause. A note at the bottom explains that using aggregate functions instead of computing aggregates yourself can reduce network load.

Select ... Where + Check	Select using an aggregate function
Runtime: 3.360 Microsecondi	Runtime: 932 Microsecondi
<pre>DATA: MAX_MSGNR type t100-msgnr. MAX_MSGNR = '000'. SELECT * FROM T100 INTO T100_WA WHERE SPRSL = 'D' AND ARGBB = '00'. CHECK: T100_WA-MSGNR > MAX_MSGNR. MAX_MSGNR = T100_WA-MSGNR. ENDSELECT.</pre>	<pre>DATA: MAX_MSGNR type t100-msgnr. SELECT MAX(MSGNR) FROM T100 INTO max_msgnr WHERE SPRSL = 'D' AND ARGBB = '00'. Riga 1, colonna 1 Riga 1 - Riga 8 di 8 righe Riga 1, colonna 1 Riga 1 - Riga 4 di 4 righe</pre>
<p>Document. If you want to find the maximum, minimum, sum and average value or the count of a database column, use a select list with aggregate functions instead of computing the aggregates yourself. Network load is considerably less.</p>	

In questo esempio viene fatto un confronto fra due metodi per trovare il massimo: da una parte viene eseguito un

controllo per ogni riga trovata dalla SELECT, dall'altra viene utilizzata funzione di aggregazione MAX.

Analizzando tutti i casi presenti in questa sezione si può facilmente comprendere quali sono i principali punti in cui bisogna ottimizzare un programma nei limiti del possibile.

METODO 1	TEMPO	METODO 2	TEMPO	MIGLIORIA
SELECT senza buffer (BYPASSING BUFFER)	405	SELECT con supporto del buffer	8	98,02%
LOOP con APPEND (copiare tabella)	49	Tabella1 [] = Tabella2 []	1	97,96%
Ricerca lineare (READ)	37	BINARY SEARCH	1	97,30%
DELETE n volte (eliminare righe)	35	DELETE ... FROM ... TO ...	1	97,14%
SELECT annidate	156830	SELECT con VIEW	5184	96,69%
LOOP con INSERT	437	INSERT LINES OF ... TO ... INDEX ...	21	95,19%
LOOP annidati	11053	Cursori paralleli	604	94,54%
LOOP con APPEND	253	APPEND LINES OF ... TO ...	17	93,28%
LOOP con DELETE	359	DELETE ADJACENT DUPLICATES FROM ... COMPARING ...	25	93,04%
SELECT annidate	11946	SELECT con JOIN	1001	91,62%
SELECT	8127	CONTEXT	748	90,80%
READ senza indice secondario	21	BINARY SEARCH con indice secondario	2	90,48%
Controlli vari sulle tabelle	175	IF Tab1 [] = Tab2 []	17	90,29%
Operazioni su tabelle tramite HEADER LINE	1	Operazioni su tabelle tramite un' area esplicita	0,1	90,00%
Assegnamento numerico tramite tipo C	1	Assegnamento numerico tramite tipo F	0,1	90,00%
Operazioni aritmetiche con tipo N	1	Operazioni aritmetiche con tipo N	0,1	90,00%
Operazioni aritmetiche con tipi misti	1	Operazioni aritmetiche con lo stesso tipo	0,1	90,00%
SORT	121	SORT ... BY ...	23	80,99%
SELECT + APPEND	176	SELECT INTO TABLE	35	80,11%
SINGLE-LINE UPDATE	4235	COLUMN UPDATE	878	79,27%
LOOP con CHECK interno	47	LOOP con WHERE	11	76,60%
SELECT ... WHERE + CHECK	3523	SELECT con funzioni di aggregazione	899	74,48%
SELECT *	4053	SELECT lista attributi	1329	67,21%
SINGLE-LINE INSERT	5.888.149	ARRAY INSERT	2.164.498	63,24%
Approccio One-Step	1460	Three-Step: MOVE, SORT, DELETE DUPL.	635	56,51%

Nella figura precedente sono raffigurate le ottimizzazioni più incisive in un programma, “Metodo 1” indica la descrizione di una normale procedura e “Metodo 2” indica la versione migliorata della stessa.

Alcuni casi potrebbero non essere qui presenti poiché gli esempi vengono fatti su alcune tabelle che potrebbero anche essere vuote, quindi i tempi di una procedura lenta potrebbero risultare gli stessi di una procedura ottimizzata.

4.2 CONSIDERAZIONI

Nella funzione da ottimizzare Z_Extract_Viaggi_01C ci sono punti in cui intere porzioni di codice sono da ottimizzare attraverso combinazioni delle migliorie viste in precedenza: di conseguenza è necessario di volta in volta valutare come applicare i Tips & Tricks sopracitati adattandoli alla situazione.

Questa funzione non è stata implementata già in modo ottimizzato perché essa è stata scritta molti anni fa, quindi sono stati introdotte nuove funzionalità dall'epoca.

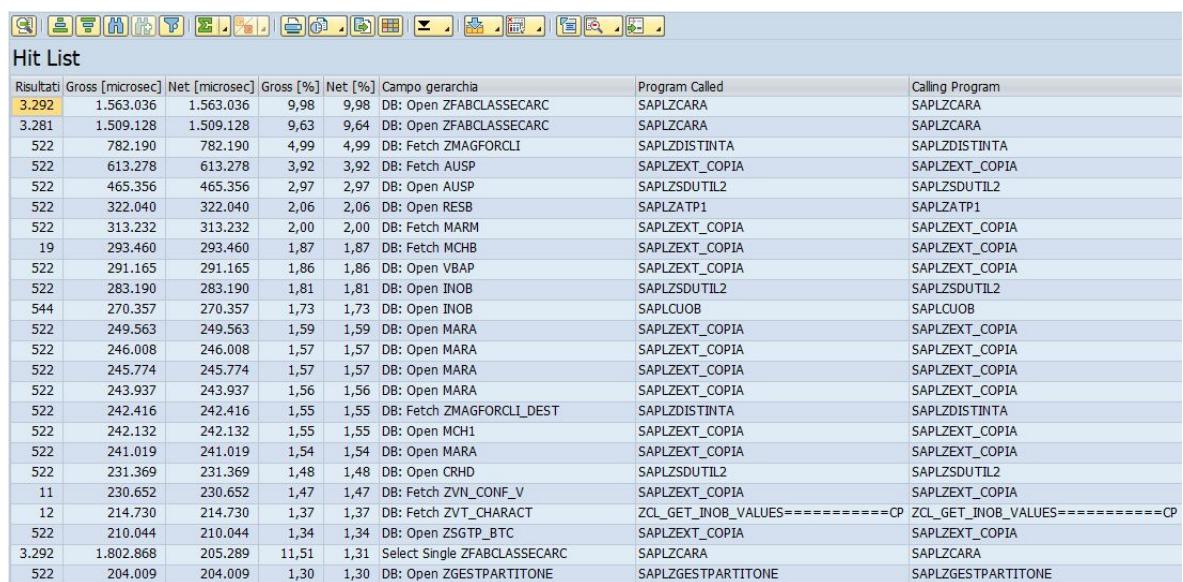
Questo originariamente era nato come un programma molto semplice, quindi non necessitava di prestazioni eccellenti: sono state introdotte negli anni delle nuove funzionalità da programmatore differenti, il che rende ancora più complicata la scrittura di un codice già ottimizzato.

5. ANALISI FUNZIONE

5.1 ANALISI GENERALE FUNZIONE

SAP HANA possiede un tool che permette di visualizzare i tempi di esecuzione e altri dati relativi ad ogni singola procedura che viene eseguita in un programma.

Di seguito ne viene mostrato un esempio.



The screenshot shows a SAP HANA studio interface with a toolbar at the top and a 'Hit List' table below. The table has columns for 'Risultati', 'Gross [microsec]', 'Net [microsec]', 'Gross [%]', 'Net [%]', 'Campo gerarchia', 'Program Called', and 'Calling Program'. The data in the table represents various database operations and their execution times.

Risultati	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia	Program Called	Calling Program
3.292	1.563.036	1.563.036	9,98	9,98	DB: Open ZFABCLASSECARC	SAPLZCARA	SAPLZCARA
3.281	1.509.128	1.509.128	9,63	9,64	DB: Open ZFABCLASSECARC	SAPLZCARA	SAPLZCARA
522	782.190	782.190	4,99	4,99	DB: Fetch ZMAGFORCLI	SAPLZDISTINTA	SAPLZDISTINTA
522	613.278	613.278	3,92	3,92	DB: Fetch AUSP	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	465.356	465.356	2,97	2,97	DB: Open AUSP	SAPLZSDUTIL2	SAPLZSDUTIL2
522	322.040	322.040	2,06	2,06	DB: Open RESB	SAPLZATP1	SAPLZATP1
522	313.232	313.232	2,00	2,00	DB: Fetch MARM	SAPLZEXT_COPIA	SAPLZEXT_COPIA
19	293.460	293.460	1,87	1,87	DB: Fetch MCHB	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	291.165	291.165	1,86	1,86	DB: Open VBAP	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	283.190	283.190	1,81	1,81	DB: Open INOB	SAPLZSDUTIL2	SAPLZSDUTIL2
544	270.357	270.357	1,73	1,73	DB: Open INOB	SAPLCUOB	SAPLCUOB
522	249.563	249.563	1,59	1,59	DB: Open MARA	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	246.008	246.008	1,57	1,57	DB: Open MARA	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	245.774	245.774	1,57	1,57	DB: Open MARA	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	243.937	243.937	1,56	1,56	DB: Open MARA	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	242.416	242.416	1,55	1,55	DB: Fetch ZMAGFORCLI_DEST	SAPLZDISTINTA	SAPLZDISTINTA
522	242.132	242.132	1,55	1,55	DB: Open MCH1	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	241.019	241.019	1,54	1,54	DB: Open MARA	SAPLZEXT_COPIA	SAPLZEXT_COPIA
522	231.369	231.369	1,48	1,48	DB: Open CRHD	SAPLZSDUTIL2	SAPLZSDUTIL2
11	230.652	230.652	1,47	1,47	DB: Fetch ZVN_CONF_V	SAPLZEXT_COPIA	SAPLZEXT_COPIA
12	214.730	214.730	1,37	1,37	DB: Fetch ZVT_CHARACT	ZCL_GET_INOB_VALUES=====CP	ZCL_GET_INOB_VALUES=====CP
522	210.044	210.044	1,34	1,34	DB: Open ZSGTP_BTC	SAPLZEXT_COPIA	SAPLZEXT_COPIA
3.292	1.802.868	205.289	11,51	1,31	Select Single ZFABCLASSECARC	SAPLZCARA	SAPLZCARA
522	204.009	204.009	1,30	1,30	DB: Open ZGESTPARTITONE	SAPLZGESTPARTITONE	SAPLZGESTPARTITONE

- “Risultati” indica il numero di volte che una funzione viene eseguita.
- “Gross[microsec]” indica il tempo in microsecondi totale di tutte le esecuzioni della stessa funzione.

- “Net[microsec]” indica il tempo in microsecondi di una singola esecuzione di una funzione.
- “Gross[%]” indica la percentuale relativa all’esecuzione di tutte le stesse funzioni rispetto al tempo totale.
- “Net[%]” indica la percentuale relativa all’esecuzione di una singola funzione rispetto al tempo totale.
- “Campo gerarchia” indica il nome della funzione chiamata ed eventualmente anche se è una funzione o una chiamata al database.
- “Program Called” è il nome del programma contenente la funzione chiamata.
- “Calling Program” è il nome del programma che chiama la funzione.

Grazie a questo tool si possono facilmente individuare i punti critici nell’esecuzione di un programma ordinando la colonna Gross[microsec] o Gross[%] in modo decrescente, identificando in questo modo le procedure che impiegano più tempo.

I tempi di esecuzione per semplicità saranno scaricati su Excel come si vedrà nel paragrafo successivo.

5.2 ANALISI DETTAGLIATA TEMPI

Per fare un'analisi dettagliata dei tempi di lancio si esegue la funzione Z_Extract_Viaggi_01C su tre casistiche differenti su uno stabilimento di Marcegaglia a Casalmaggiore (Divisione “0103”).

Sono state scelte tre situazioni completamente differenti, sia per tempi di esecuzione che per funzioni chiamate: questo per poter verificare se le ottimizzazioni riguardano solo alcune casistiche o se sono state applicate a tutte quelle presenti.

Questi tre lanci sono identificati come:

- Prova 1
 - Divisione 0103
 - Viaggio 6897485
- Prova 2
 - Divisione 0103
 - Viaggio 6901235
- Prova 3
 - Divisione 0103
 - Viaggio 6901497

Inserendo successivamente i rispettivi input di queste tre casistiche si può iniziare a rilevare i tempi di esecuzione della funzione in questione.

5.2.1 ANALISI TEMPI PROVA 1

Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
1	15.146.503	0	100	0	Runtime analysis
1	15.146.034	1.175	100	0,01	Call Function Z_EXTRACT_VIAGGI_01C
1	11.802.612	57	77,92	<0,01	Perform CALL_EXTR_VIAGGIO_DIST
1	11.802.541	47	77,92	<0,01	Call Function Z_EXTRACT_LISTA_PRELIEVO_01C
1	11.800.662	4.833	77,91	0,03	Perform START_MAIN_EXTR_VIAGGIO
11	8.882.342	43.222	58,64	0,29	Perform FILL_PARTITE_POSSIBILI
3.292	3.628.289	32.299	23,95	0,21	Call Function Z_FIND_NAME_CARATT
3.292	3.595.990	37.446	23,74	0,25	Perform SELECT_TAB
1	3.317.255	137.047	21,9	0,9	Perform EXTR_PARTITE_PACCO_VIAGGIO
11	2.575.106	256	17	<0,01	Perform FILL_CHARG_STOCK_COMM
11	2.574.850	66.379	17	0,44	Open SQL MCHB
3.281	1.878.077	56.023	12,4	0,37	Open SQL ZFABCLASSECARC
11	1.792.065	10.253	11,83	0,07	Perform FILL_CARATT_VIAGGIO
3.292	1.680.210	204.592	11,09	1,35	Select Single ZFABCLASSECARC
3.281	1.578.485	168.152	10,42	1,11	Open Cursor ZFABCLASSECARC
522	1.474.550	5.310	9,74	0,04	Perform CONTROLL_MAG_FORN
522	1.469.158	27.386	9,7	0,18	Call Function Z_GET_ZMAGFORCLI
3.292	1.440.787	1.440.787	9,51	9,51	DB: Open ZFABCLASSECARC
3.281	1.409.742	1.409.742	9,31	9,31	DB: Open ZFABCLASSECARC
1.044	1.137.718	6.971	7,51	0,05	Perform FIND_NAME_CARATT
522	1.078.323	18.039	7,12	0,12	Call Function ZSD_UBICAZIONE_FROM_BATCH
522	1.059.683	15.586	7	0,1	Call Function Z_GEST_PARTITONE
693	940.143	9.279	6,21	0,06	Perform FILL_CARA
522	890.491	4.533	5,88	0,03	Open SQL ZMAGFORCLI
522	865.922	19.588	5,72	0,13	Perform CONTROLLA_UM_ODV
522	849.587	55.972	5,61	0,37	Fetch ZMAGFORCLI
522	793.615	793.615	5,24	5,24	DB: Fetch ZMAGFORCLI

Nella figura precedente si può constatare che il tempo di esecuzione della funzione (evidenziato in verde) è di 15.146.503 microsecondi, che equivalgono a 15 secondi circa.

Le prime cinque righe di questa tabella non danno molte informazioni in quanto costituiscono il MAIN del programma in questione e sarebbe come dire che la parte da ottimizzare è il programma stesso (il che non ha alcun senso).

Già dalla sesta riga si può individuare una situazione abbastanza critica: la funzione FILL PARTITE POSSIBILI.

Questa funzione ha una durata di circa 9 secondi, che rappresentano quasi il 60% del tempo di esecuzione del programma intero.

Successivamente si possono vedere altre procedure dal tempo di esecuzione abbastanza elevato come Z FIND NAME CARATT, SELECT TAB, EXTR PARTITE PACCO VIAGGIO, FILL CHARG STOCK COMM, OPEN SQL MCHB e così via.

Queste sono solo alcune delle sezioni che si andranno ad ottimizzare, poiché si miglioreranno tutte le procedure che presentano delle imperfezioni nei tempi di esecuzione e non solo quelle con i tempi più elevati.

Come si può notare la somma delle percentuali non è il 100%, questo perché in questo programma capita spesso che una funzione ne contenga altre annidate: per esempio se una funzione impiega il 50% e contiene tre funzioni, la somma delle tre funzioni sarà il 50% ma questi due valori naturalmente non sono da sommare tra di loro.

5.2.1 ANALISI TEMPI PROVA 2

Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
1	8.580.669	0	100	0	Runtime analysis
1	8.580.194	674	99,99	0,01	Call Function Z_EXTRACT_VIAGGI_01C
1	6.493.322	51	75,67	<0,01	Perform CALL_EXTR_VIAGGIO_DIST
1	6.493.253	36	75,67	<0,01	Call Function Z_EXTRACT_LISTA_PRELIEVO_01C
1	6.490.163	2.607	75,64	0,03	Perform START_MAIN_EXTR_VIAGGIO
6	4.896.139	22.438	57,06	0,26	Perform FILL_PARTITE_POSSIBILI
1	2.068.238	49.548	24,1	0,58	Perform EXTR_PARTITE_PACCO_VIAGGIO
1,747	2.027.643	18.197	23,63	0,21	Call Function Z_FIND_NAME_CARATT
1,747	2.009.446	20.386	23,42	0,24	Perform SELECT_TAB
6	1.403.762	157	16,36	<0,01	Perform FILL_CHARG_STOCK_COMM
6	1.403.605	42.035	16,36	0,49	Open SQL MCHB
1,741	1.041.824	30.982	12,14	0,36	Open SQL ZFABCLASSECARC
6	947.306	5.436	11,04	0,06	Perform FILE_CARATT_VIAGGIO
1,747	947.078	114.133	11,04	1,33	Select Singl ZFABCLASSECARC
1,741	876.764	95.985	10,22	1,12	Open Cursor ZFABCLASSECARC
1,747	813.374	813.374	9,48	9,48	DB: Open ZFABCLASSECARC
1,741	780.284	780.284	9,09	9,09	DB: Open ZFABCLASSECARC
275	763.089	2.756	8,89	0,03	Perform CONTROLL_MAG_FORN
275	759.516	14.321	8,85	0,17	Call Function Z_GET_ZMAGFORCLI
550	642.817	3.771	7,49	0,04	Perform FIND_NAME_CARATT
275	585.258	9.896	6,82	0,12	Call Function ZSD_UBICAZIONE_FROM_BATCH
275	573.161	8.149	6,68	0,09	Call Function Z_GEST_PARTITIONE
275	533.583	10.959	6,22	0,13	Perform CONTROLLA_UM_ODV
378	503.814	4.750	5,87	0,06	Perform FILE_CARA
677	482.733	168.469	5,63	1,96	Select Singl MARA
275	465.401	2.385	5,42	0,03	Open SQL ZMAGFORCLI
275	443.321	28.932	5,17	0,34	Fetch ZMAGFORCLI

In questo secondo caso il tempo di esecuzione totale è di 8.580.669 microsecondi, quindi circa 8 secondi e mezzo.

Si può notare che le funzioni interessate maggiormente, anche se con ordine differente, sono sempre le stesse del caso precedente.

5.2.1 ANALISI TEMPI PROVA 3

Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
1	5.705.956	0	100	0	Runtime analysis
1	5.705.582	544	99,99	0,01	Call Function Z_EXTRACT_VIAGGI_01C
1	4.544.547	42	79,65	<0,01	Perform CALL_EXTR_VIAGGIO_DIST
1	4.544.492	35	79,64	<0,01	Call Function Z_EXTRACT_LISTA_PRELIEVO_01C
1	4.543.025	1.933	79,62	0,03	Perform START_MAIN_EXTR_VIAGGIO
7	2.906.733	12.342	50,94	0,22	Perform FILL_PARTITE_Possibili
1.284	1.439.988	12.939	25,24	0,23	Call Function Z_FIND_NAME_CARATT
1.284	1.427.049	15.015	25,01	0,26	Perform SELECT_TAB
1	1.147.115	25.838	20,1	0,45	Perform EXTR_PARTITE_PACCO_VIAGGIO
7	1.129.698	6.167	19,8	0,11	Perform FILL_CARATT_VIAGGIO
7	799.672	169	14,01	<0,01	Perform FILL_CHARG_STOCK_COMM
7	799.503	17.350	14,01	0,3	Open SQL_MCHB
1.277	744.490	22.269	13,05	0,39	Open SQL_ZFABCLASSECARC
1.284	667.373	82.104	11,7	1,44	Select Single ZFABCLASSECARC
1.277	625.180	68.411	10,96	1,2	Open Cursor ZFABCLASSECARC
441	599.683	5.648	10,51	0,1	Perform FILL_CARA
1.284	572.059	572.059	10,03	10,03	DB: Open ZFABCLASSECARC
1.277	556.769	556.769	9,76	9,76	DB: Open ZFABCLASSECARC
434	493.083	2.891	8,64	0,05	Perform FIND_NAME_CARATT
170	467.572	1.691	8,19	0,03	Perform CONTROLL_MAG_FORN
170	465.811	8.678	8,16	0,15	Call Function Z_GET_ZMAGFORCL
340	381.674	2.336	6,69	0,04	Perform FIND_NAME_CARATT
170	359.691	5.106	6,3	0,09	Call Function Z_GEST_PARTITONE
170	356.267	6.109	6,24	0,11	Call Function ZSA_UBICAZIONE_FROM_BATCH
170	292.732	6.666	5,13	0,12	Perform CONTROLLA_UM_ODV
170	288.092	1.484	5,05	0,03	Open SQL_ZMAGFORCL
170	275.508	18.073	4,83	0,32	Fetch ZMAGFORCL

In questo terzo caso il tempo di esecuzione totale è di 5.705.956 microsecondi, quindi poco meno di 6 secondi.

Come nel caso precedente le funzioni chiamate in causa che allungano il tempo di esecuzione totale del programma sono sempre le stesse del primo caso ma con ordine differente.

6. ANALISI SOLUZIONI FUNZIONE

6.1 ANALISI POSSIBILI SOLUZIONI

La prima cosa da fare è studiare il programma in sé per cercare di capire le funzioni che svolge e attraverso quali passaggi. Questo perché se si dovessero guardare solamente poche righe di codice alla volta non si comprenderebbe il suo compito, e a volte può anche succedere che la stessa funzionalità possa essere svolta in un modo totalmente diverso attraverso altri metodi.

Un'altra possibile soluzione può essere quella di crearsi delle strutture permanenti in un'area globale. Questo metodo risolve i casi in cui una funzione viene chiamata molte volte e ad ogni chiamata si deve estrarre sempre gli stessi identici dati: in questo caso se la funzione viene chiamata un certo numero di volte, essa non si deve estrarre i valori ogni volta ma solamente alla prima occorrenza, dopodiché recupererà i valori dai dati globali.

Le altre possibili soluzioni sono quelle di agire direttamente sulle istruzioni del codice, ovvero cambiarne la struttura per ridurne il tempo di esecuzione.

Per esempio quando si incontra una serie di LOOP annidati è opportuno cercare di eliminarli o ridurne al minimo la complessità. Un altro esempio può essere quello di estrarre i dati per poi leggerli da una tabella interna piuttosto che fare una chiamata al database per ogni lettura del dato.

Infine si può semplicemente velocizzare una parte di codice attraverso Calculation View o AMDP.

6.2 DETTAGLI SOLUZIONI

Qui di seguito sono elencati i principali punti in cui è necessaria l'ottimizzazione del codice:

- **SELECT SINGLE**

Questo costrutto seleziona una riga da una database e può essere utilizzato o per leggere un elemento o per verificarne la presenza.

Se questa istruzione viene chiamata più volte (per esempio in un LOOP) conviene creare una tabella interna e utilizzare il comando READ.

- **READ WITH KEY**

Quando si utilizza questo comando è sempre conveniente ordinare la tabella in cui si sta leggendo in base alla chiave di ricerca, in questo modo è possibile utilizzare il comando READ BINARY SEARCH, che utilizza la ricerca binaria.

Un metodo ancora più efficiente sono le Hash Table.

- **SELECT/ENDSELECT**

Questo costrutto viene utilizzato per eseguire delle operazioni su ogni riga trovata dalla SELECT in modo sequenziale, quindi ogni volta che viene trovato un record viene effettuata una chiamata al database.

La principale soluzione è quella di creare una tabella interna per poi eseguire le operazioni su ogni sua riga attraverso un LOOP.

Un'alternativa ancora più efficiente, se possibile, è quella di utilizzare Calculation View o AMDP.

- **SORT**

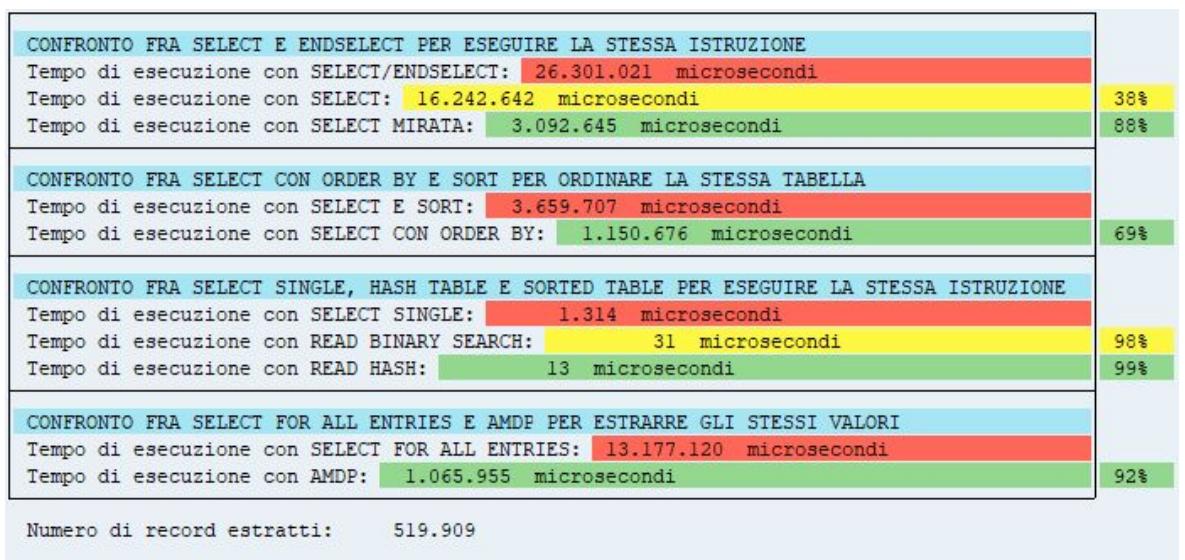
Quando si utilizza il comando SORT per ordinare una tabella interna conviene controllare se era possibile ordinarla direttamente nella SELECT attraverso il comando ORDER BY.

Il database ha una potenza di calcolo sicuramente maggiore di un normale computer, quindi il tempo di esecuzione sarà inferiore.

- **SELECT**

Per migliorare l'efficienza di una SELECT conviene:

- ★ Ridurre al minimo il numero di righe estratte specificando al meglio la clausola WHERE.
- ★ Ridurre al minimo il numero delle colonne estratte.
- ★ Ridurre al minimo il numero delle SELECT, per esempio se sono presenti più SELECT è opportuno cercare di unirle in una sola.
- ★ In alternativa, come citato in precedenza si possono usare Calculation View o AMDP.



Nell'immagine precedente viene mostrato un esempio dell'esecuzione di tutti i costrutti citati in precedenza con i relativi miglioramenti.

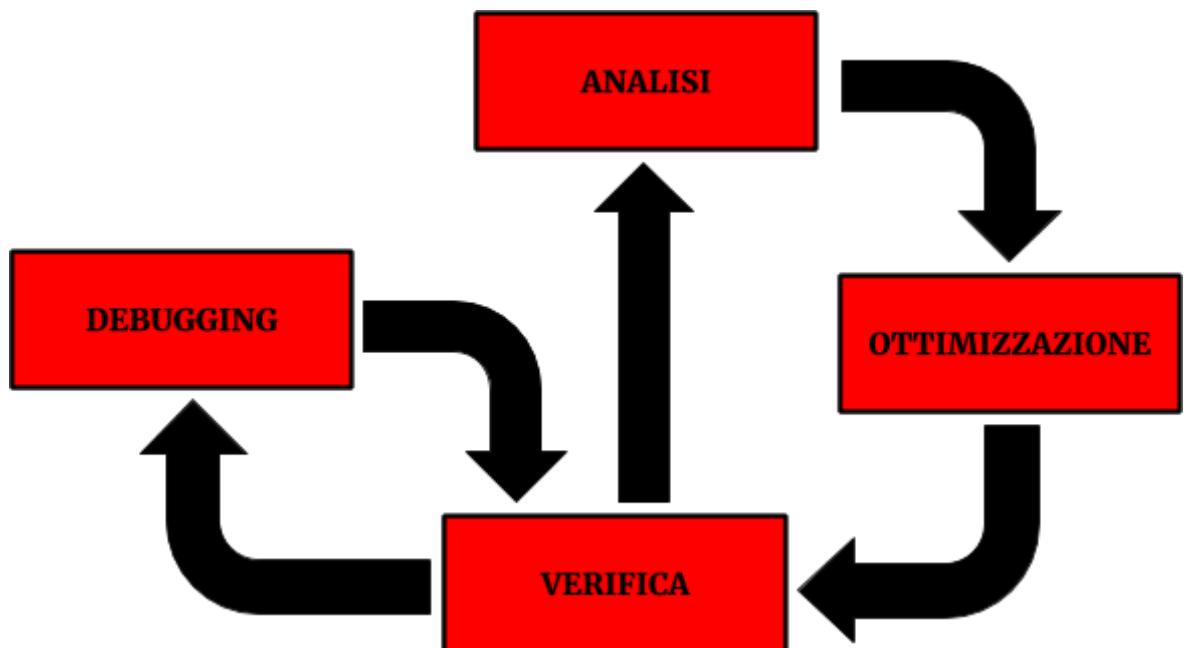
Come si può notare, con le ottimizzazioni applicate, il tempo di esecuzione viene notevolmente ridotto.

7. FASE DI OTTIMIZZAZIONE

7.1 PIANIFICAZIONE

Prima di iniziare ad ottimizzare una funzione molto complessa, in questo caso oltre 100.000 righe di codice, occorre pianificare il tutto al meglio per non perdere mai il punto della situazione.

La pianificazione si può riassumere con il seguente schema:



- **ANALISI**

Vengono individuati alcuni punti da ottimizzare, generalmente pochi, per facilitare l'individuazione di eventuali errori.

- **OTTIMIZZAZIONE**

Questa è la fase principale, ovvero la fase in cui si ottimizza la funzione Z_EXTRACT_VIAGGI_01C.

- **VERIFICA**

Attraverso alcuni test, che verranno specificati dettagliatamente nel capitolo successivo, occorre verificare la bontà dei dati estratti dalla funzione.

Il numero dei record e i valori estratti devono essere gli stessi ma non necessariamente nello stesso ordine (se non esplicitamente richiesto).

- **DEBUGGING**

Se al punto precedente vengono rilevate delle incongruenze fra i valori originali e i valori estratti dalla funzione da ottimizzare occorre correggere gli errori con l'utilizzo del debugger.

La fase di debug/Test non si conclude finchè non vengono risolte tutte le anomalie.

Quando gli output sono tornati uguali a quelli della funzione di origine si torna alla prima fase di analisi.

7.2 INDIVIDUAZIONE DEI PUNTI DA OTTIMIZZARE

La scelta della funzione da ottimizzare è abbastanza semplice, in quanto è sufficiente osservare l'analisi dei tempi di esecuzione e andare per ordine partendo dal tempo più elevato (capitolo 5.2).

La parte un po' più complessa sta nell'individuare i punti critici nel codice e trovare una soluzione come specificato nel capitolo 6.2.

Siccome i tre casi in questione, anche se con tempi diversi, utilizzano le stesse funzioni, possiamo individuare i punti critici osservando solamente la “Prova 1”.

Una volta terminata l'ottimizzazione relativa al primo caso è sufficiente verificare se ci sono altri punti non ottimizzati utilizzati da “Prova 2” e “Prova 3”.

7.3 OTTIMIZZAZIONI PRINCIPALI

In questo paragrafo si andranno ad analizzare in modo dettagliato le principali ottimizzazioni effettuate alle varie funzioni.

- **FILL PARTITE POSSIBILI**

```
IF l_pacco(1) = 'D'.
  READ TABLE p_paccoviaggio WITH KEY
    zzxabln = p_chargviaggio-zzxabln
    zzposnr = p_chargviaggio-zzposnr
    npack   = l_pacco.
  MOVE sy-subrc TO l_subrc_pacco.
ELSE.
  READ TABLE p_paccoviaggio WITH KEY
    zzxabln = p_chargviaggio-zzxabln
    zzposnr = p_chargviaggio-zzposnr
    npack   = l_pacco.
  MOVE sy-subrc TO l_subrc_pacco.
ENDIF.
```

Si può notare che le READ non sono effettuate con la BINARY SEARCH quindi sono sufficienti due modifiche: SORT e BINARY SEARCH.

```
SORT p_paccoviaggio BY zzxabln npack zzposnr.
IF l_pacco(1) = 'D'.
  READ TABLE p_paccoviaggio WITH KEY
    zzxabln = p_chargviaggio-zzxabln
    zzposnr = p_chargviaggio-zzposnr
    npack   = l_pacco
    BINARY SEARCH.
  MOVE sy-subrc TO l_subrc_pacco.
ELSE.
  READ TABLE p_paccoviaggio WITH KEY
    zzxabln = p_chargviaggio-zzxabln
    npack   = l_pacco
    zzposnr = p_chargviaggio-zzposnr
    BINARY SEARCH.

  MOVE sy-subrc TO l_subrc_pacco.
ENDIF.
```

- **SELECT TAB**

```
SELECT SINGLE * FROM ZFABCLASSECARC WHERE KLART = P_KLART
    AND CLASS = P_CLASS
    AND ID_CARAT = P_ID_CARAT.

IF SY-SUBRC = 0.
    MOVE ZFABCLASSECARC-ATNAM TO P_ATNAM.
    SELECT SINGLE * FROM CABN WHERE ATNAM = ZFABCLASSECARC-ATNAM.
    IF SY-SUBRC = 0.
        MOVE CABN-ATINN TO P_ATINN.
    ENDIF.
ELSE.
    SELECT * FROM ZFABCLASSECARC WHERE ID_CARAT = P_ID_CARAT.
    CHECK ZFABCLASSECARC-KLART IS INITIAL.
    CHECK ZFABCLASSECARC-CLASS IS INITIAL.
    IF SY-SUBRC = 0.
        MOVE ZFABCLASSECARC-ATNAM TO P_ATNAM.
        SELECT SINGLE * FROM CABN WHERE ATNAM = ZFABCLASSECARC-ATNAM.
        IF SY-SUBRC = 0.
            MOVE CABN-ATINN TO P_ATINN.
        ENDIF.
    ELSE.
        RAISE NOT_FOUND.
    ENDIF.
ENDSELECT.
ENDIF.
```

Si nota che questa sezione contiene tre SELECT SINGLE, di cui una è a sua volta contenuta in una SELECT/ENDSELECT.

La prima SELECT SINGLE viene sempre effettuata e serve per leggere un valore in una tabella del database: se questo valore viene trovato viene effettuata un'altra SELECT SINGLE, altrimenti viene eseguita una SELECT SINGLE annidata in una SELECT/ENDSELECT.

I tempi di esecuzione in questa parte di codice sono molto elevati, quindi occorre ottimizzarla al meglio.

```

IF ht_zfabclassecarc IS INITIAL.
  SELECT DISTINCT z~atnam,z~klart,z~class,z~id_carat,c~atinn
  FROM zfabclassecarc AS z INNER JOIN cabn AS c
    ON z~atnam = c~atnam
  INTO CORRESPONDING FIELDS OF TABLE @ht_zfabclassecarc.
  INTO TABLE @DATA(TEMP).
  HT_ZFABCLASSECARC[] = TEMP.
ENDIF.

READ TABLE ht_zfabclassecarc
  WITH KEY klart = p_klart
    class = p_class
      id_carat = p_id_carat
  ASSIGNING FIELD-SYMBOL(<wa>).

IF sy-subrc = 0.
  MOVE <wa>-atnam TO p_atnam.
  READ TABLE ht_zfabclassecarc
  WITH KEY atnam = <wa>-atnam
  ASSIGNING <wa>.
  SELECT SINGLE * FROM cabn WHERE atnam = zfabclassecarc-atnam.
  IF sy-subrc = 0.
    MOVE <wa>-atinn TO p_atinn.
  ENDIF.
ELSE.
  READ TABLE ht_zfabclassecarc
  WITH KEY id_carat = P_id_carat
    klart    = ''
    class    = ''
  ASSIGNING <wa>.
  SELECT * FROM zfabclassecarc WHERE id_carat = p_id_carat.
  CHECK <wa>-klart IS INITIAL.
  CHECK <wa>-class IS INITIAL.
  IF sy-subrc = 0.
    MOVE <wa>-atnam TO p_atnam.
    READ TABLE ht_zfabclassecarc
    WITH KEY atnam = <wa>-atnam
    ASSIGNING <wa>.
    SELECT SINGLE * FROM cabn WHERE atnam = zfabclassecarc-atnam.
    IF sy-subrc = 0.
      MOVE <wa>-atinn TO p_atinn.
    ENDIF.
  ELSE.
    RAISE not_found.
  ENDIF.
ENDSELECT.
ENDIF.

```

Per velocizzare questa funzione è stata creata una tabella Hash globale poiché questa funzione veniva chiamata circa 500 volte, quindi fare tutte queste letture su una tabella Hash (che verrà creata solo alla prima occorrenza) piuttosto che fare una chiamata al database ogni volta incrementa notevolmente le prestazioni.

Inizialmente viene effettuata una READ sulla tabella Hash, quindi molto veloce, e successivamente, sia che si trovi l'elemento cercato o che non si trovi, vengono effettuate una o due READ sulla tabella Hash. Tutto questo sostituisce le SELECT SINGLE e la SELECT/ENDSELECT che rallentavano notevolmente la funzione.

Il simbolo “<wa>” rappresenta un Field-Symbol, cioè un puntatore, il cui scopo è quello di velocizzare l’assegnamento dei valori andando a modificare direttamente la cella di memoria a cui esso è collegato.

• Z GEST PARTITONE

```
CLEAR E_ZGESTPARTITONE.

SELECT SINGLE * FROM ZGESTPARTITONE WHERE WERKS = I_WERKS
    AND SPART = I_SPART
    AND MATKL = I_MATKL
    AND AUART = I_AUART
    AND KUNNR = I_KUNNR.

IF SY-SUBRC NE 0.
    SELECT SINGLE * FROM ZGESTPARTITONE WHERE WERKS = I_WERKS
        AND SPART = I_SPART
        AND MATKL = I_MATKL
        AND AUART = I_AUART
        AND KUNNR = ''.

ENDIF.

IF SY-SUBRC NE 0.
    SELECT SINGLE * FROM ZGESTPARTITONE WHERE WERKS = I_WERKS
        AND SPART = I_SPART
        AND MATKL = I_MATKL
        AND AUART = ''
        AND KUNNR = ''.

ENDIF.

IF SY-SUBRC NE 0.
    SELECT SINGLE * FROM ZGESTPARTITONE WHERE WERKS = I_WERKS
        AND SPART = I_SPART
        AND MATKL = ''
        AND AUART = ''
        AND KUNNR = ''.

ENDIF.

IF SY-SUBRC NE 0.
    SELECT SINGLE * FROM ZGESTPARTITONE WHERE WERKS = I_WERKS
        AND SPART = ''
        AND MATKL = ''
        AND AUART = ''
        AND KUNNR = ''.

ENDIF.

IF SY-SUBRC NE 0.
    RAISE NOT_FOUND.
ELSE.
    MOVE ZGESTPARTITONE TO E_ZGESTPARTITONE.
ENDIF.
```

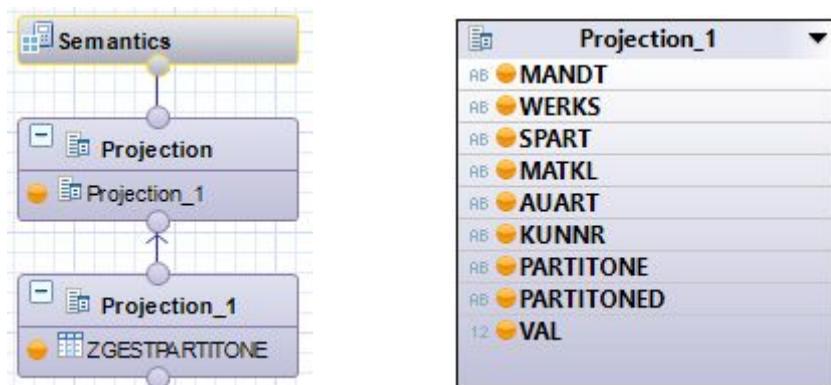
Come accennato in precedenza le SELECT SINGLE sono un problema in termini di prestazioni, quindi è essenziale eliminarle o perlomeno cercare di limitarne l'uso.

Questa parte di codici è decisamente lenta poiché nel peggiore dei casi viene eseguito per cinque volte il costrutto SELECT SINGLE, e il tutto viene eseguito 522 volte.

Lo scopo di questa funzione è quello di estrarre il record con il maggior numero di campi corrispondenti ai parametri di ricerca (I_WERKS, I_SPART, I_MATKL, I_AUART, I_KUNNR) e che abbia almeno il campo della divisione corretto (WERKS).

Per estrarre questo record, questa funzione fa una serie di SELECT SINGLE una dopo l'altra ed eliminando ogni volta un parametro di ricerca finché non viene trovato un valore corretto.

Per ottimizzare questa sezione verranno utilizzate Calculation View e AMDP.



Nelle figure precedenti è raffigurata la struttura della Calculation View utilizzata per l'ottimizzazione.

Rispetto alla tabella originale è stata aggiunta la colonna VAL che contiene il numero di campi non nulli (i cinque parametri di ricerca) per ogni record.

VAL è calcolata nel modo seguente:

```
if(( "WERKS" != '' ) and ( "SPART" != '' ) and ( "MATKL" != '' ) and ( "AUART" != '' ) and ( "KUNNR" != '' ), 5,
if(( "WERKS" != '' ) and ( "SPART" != '' ) and ( "MATKL" != '' ) and ( "AUART" != '' ), 4,
if(( "WERKS" != '' ) and ( "SPART" != '' ) and ( "MATKL" != '' ), 3,
if(( "WERKS" != '' ) and ( "SPART" != '' ), 2,
if(( "WERKS" != '' ), 1, 0))))
```

Una volta creata questa tabella si può procedere con l'estrazione del valore vera e propria attraverso AMDP.

```
TYPES : itchar TYPE STANDARD TABLE OF zchargviaggioc,
itpos TYPE STANDARD TABLE OF zposviaggioc,
BEGIN OF ty_zgpar,
vbeln      TYPE vbap-vbeln,
posnr      TYPE vbap-posnr.
INCLUDE TYPE z_gest_partitone.

TYPES : END OF ty_zgpar,
zgpar TYPE STANDARD TABLE OF ty_zgpar,
BEGIN OF ty_odv,
vbeln      TYPE vbap-vbeln,
posnr      TYPE vbap-posnr,
matnr      TYPE mara-matnr,
matkl      TYPE mara-matkl,
spart      TYPE vbap-spart,
werks      TYPE vbap-werks,
auart      TYPE vbak-auart,
kunnr      TYPE vbak-kunnr,
arktx      TYPE vbap-arktx,
gestpar(1),
lgort      TYPE vbap-lgort,
END OF ty_odv,
tty_odv TYPE STANDARD TABLE OF ty_odv,

CLASS-METHODS gestione_partitone
IMPORTING
VALUE(itchar) TYPE itchar
VALUE(itpos)  TYPE itpos
VALUE(w_odv)   TYPE tty_odv
VALUE(vuoto)   TYPE char01
EXPORTING
VALUE(zgpar)  TYPE zgpar.
```

Nell'immagine precedente vengono dichiarate inizialmente le strutture e le tabelle utilizzate (itchar, itpos, zgpar) ed infine viene dichiarato il metodo principale GESTIONE_PARTITONE, che verrà poi implementato nella figura successiva:

```

METHOD gestione_partitone BY DATABASE PROCEDURE FOR HDB LANGUAGE
  SQLSCRIPT OPTIONS READ-ONLY USING z_gest_partitone.
zgpar = SELECT DISTINCT od.vbeln,od.posnr,zg.*
  FROM :itchar AS ch INNER JOIN :itpos AS po
  ON ch.zzxabln = po.zzxabln AND
  ch.zzposnr = po.zzposnr
  INNER JOIN :w_odv AS od
  ON po.vbelv = od.vbeln AND
  po.posnv = od.posnr
  INNER JOIN z_gest_partitone as zg
  ON od.werks = zg.werks
  AND ( od.spart = zg.spart OR zg.spart = :vuoto ) AND
  ( od.matkl = zg.matkl OR zg.matkl = :vuoto ) AND
  ( od.auart = zg.auart OR zg.auart = :vuoto ) AND
  ( od.kunnr = zg.kunnr OR zg.kunnr = :vuoto )
ORDER BY od.vbeln,od.posnr,zg.val DESC;
ENDMETHOD.
```

Nelle prime due righe vengono specificati il metodo che si sta implementando e la Calculation View da cui prendere i valori (Z_GEST_PARTITONE).

Il resto del codice esegue una sola SELECT (invece che cinque come nel codice originale) per estrarre i valori ordinati in modo decrescente per VBELN, POSNR e VAL.

A questo punto è sufficiente richiamare il programma appena creato nella funzione ottimizzata.

```
z_amdp_gest_partitone=>gestione_partitone(
  EXPORTING
    itchar = it_chargviaggio[]
    itpos  = it_zposviaggioc[]
    w_odv  = w_odv[]
    vuoto  =
  IMPORTING
    zgpar  = tzgpar[]
).

```

- **Z EXTR PARTITE PACCO VIAGGIO**

In questa funzione verranno analizzate solo le due ottimizzazioni principali poiché le altre risulterebbero ripetitive.

```
LOOP AT w_vbuk.
  LOOP AT w_lips WHERE vbeln = w_vbuk-vbeln.
    CHECK w_lips-charg(1) CO '0123456789'.
    LOOP AT it_chargviaggio WHERE charg = w_lips-charg.
      DELETE it_chargviaggio.
    ENDLOOP.
  ENDLOOP.
ENDLOOP.
```

In questo caso vengono eseguiti tre LOOP annidati per eliminare alcune righe di una tabella, il che richiede parecchio tempo.

In questo caso l'ottimizzazione è molto semplice in quanto è sufficiente aggiungere la clausola WHERE al comando DELETE per eliminare un LOOP.

```
LOOP AT w_vbuk.
  LOOP AT w_lips WHERE vbeln = w_vbuk-vbeln AND
    charg(1) CO '0123456789'.
    DELETE it_chargviaggio WHERE charg = w_lips-charg.
  ENDLOOP.
ENDLOOP.
```

Viene eliminata anche la clausola CHECK poiché è possibile aggiungere un controllo alla clausola WHERE del secondo LOOP.

Ad aumentare i tempi di questa funzione non erano solamente queste righe ma vi era anche un costrutto SELECT/ENDSELECT come mostrato nella seguente immagine:

```
SELECT vbap~vbeln vbap~posnr vbap~werks vbak~auart
      vbap~spart vbak~kunnr mara~matkl vbap~arktx
      vbap~lgort
  INTO CORRESPONDING FIELDS OF w_odv
  FROM ( ( vbap INNER JOIN vbak
            ON vbap~vbeln = vbak~vbeln )
         INNER JOIN mara
            ON vbap~matnr = mara~matnr )
 FOR ALL ENTRIES IN w_selodv
 WHERE vbap~vbeln = w_selodv~vbeln
   AND vbap~posnr = w_selodv~posnr.
 APPEND w_odv.
ENDSELECT.
```

Come citato in precedenza questo costrutto è sempre necessario evitarlo siccome esegue molte chiamate al database, ed in questo caso sono superflue.

```
SELECT vbap~vbeln,vbap~posnr,vbap~werks,vbak~auart,
      vbap~spart,vbak~kunnr,mara~matkl,vbap~arktx,
      vbap~lgort
  INTO CORRESPONDING FIELDS OF TABLE @w_odv
  FROM ( ( vbap INNER JOIN vbak
            ON vbap~vbeln = vbak~vbeln )
         INNER JOIN mara
            ON vbap~matnr = mara~matnr )
 FOR ALL ENTRIES IN @w_selodv
 WHERE vbap~vbeln = @w_selodv~vbeln
   AND vbap~posnr = @w_selodv~posnr.
 SORT w_odv BY vbeln posnr.
```

Per ottimizzare questa parte è stato sufficiente effettuare una SELECT e salvare i valori in una tabella

tutti in una volta: la funzione originale invece estraeva un record alla volta per poi salvarlo nella tabella attraverso il comando APPEND.

Il comando SORT finale non è per ottimizzare questa parte di codice ma servirà successivamente per eventuali READ in ricerca binaria.

7.4 ESITI OTTIMIZZAZIONE

7.4.1 ESITI PROVA 1

Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
1	4.474.845	0	100	0	Runtime analysis
1	4.474.337	1.502	99,99	0,03	Call Function ZT_EXTRACT_VIAGGI_01C
1	3.930.091	60	87,83	<0,01	Perform CALL_EXTR_VIAGGIO_DIST
1	3.930.012	48	87,82	<0,01	Call Function ZT_EXTRACT_LISTA_PRELIEVO_01C
1	3.927.325	52.379	87,76	1,17	Perform START_MAIN_EXTR_VIAGGIO
11	2.296.614	81.680	51,32	1,83	Perform FILL_PARTITE_POSSIBILI
11	1.056.714	8.797	23,61	0,2	Perform FILL_CARATT_VIAGGIO
11	768.952	1.126	17,18	0,03	Call M_Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP
11	603.843	12.981	13,49	0,29	Perform FILL_CHARG_STOCK_COMM
11	545.909	392	12,2	0,01	Native SQL: Execute Procedure "Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP#stb2
11	545.517	545.517	12,19	12,19	DB: Exec "Z_AMDP_GEST_PARTITONE=>ESTRAZ
1	523.666	67.324	11,7	1,5	Perform EXTR_PARTITE_PACCO_VIAGGIO
11	338.280	101	7,56	<0,01	Open SQL MCHB
11	335.800	8.319	7,5	0,19	Fetch MCHB
19	327.481	327.481	7,32	7,32	DB: Fetch MCHB
522	316.809	11.734	7,08	0,26	Call Function ZTSD_UBICAZIONE_FROM_BATCH
522	305.075	82.501	6,82	1,84	Select Single CRHD
11	294.197	657	6,57	0,01	Perform CACL_CLASS_CHARG2
522	271.539	223.445	6,07	4,99	Perform CONTROLLA_NEXT_FASE_ALT
522	252.582	4.825	5,64	0,11	Perform CONTROLL_MAG_FORN
522	247.723	15.221	5,54	0,34	Call Function ZT_GET_ZMAGFORCLI
3.292	244.459	17.180	5,46	0,38	Call Function ZT_FIND_NAME_CARATT
522	242.597	211.924	5,42	4,74	Perform F_ESTRAI_DATI_DA_CLASS
11	241.271	288	5,39	0,01	Call Function CUCB_GET_OBJECT
11	240.052	135	5,36	<0,01	Call M. {O:54*CL_CUCB}->GET_OBJECT
22	231.734	589	5,18	0,01	Call M. {O:54*CL_CUCB}->GET_SINGLE_INSTANCE
12	229.777	1.878	5,13	0,04	Call M. {O:95*ZCL_GET_INOB_VALUES}->GET_OBJECT USING_ZVT_CHARACT

FUNZIONE INIZIALE						FUNZIONE OTTIMIZZATA					
Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia	Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
11	8.382.342	43.222	58,64	0,29	Perform FILL_PARTITE_POSSIBILI	11	2.296.614	81.680	51,32	1,83	Perform FILL_PARTITE_POSSIBILI
3.292	3.628.289	32.299	23,95	0,21	Call Function Z_FIND_NAME_CARATT	3.292	244.459	17.180	5,46	0,38	Call Function ZT_FIND_NAME_CARATT
3.292	3.595.990	37.446	23,74	0,25	Perform SELECT_TAB	3.292	227.279	223.263	5,08	4,99	Perform SELECT_TAB
1	3.317.255	137.047	21,9	0,9	Perform EXTR_PARTITE_PACCO_VIAGGIO	1	523.666	67.324	11,7	1,5	Perform EXTR_PARTITE_PACCO_VIAGGIO
11	2.575.106	256	17	<0,01	Perform FILL_CHARG_STOCK_COMM	11	603.843	12.981	13,49	0,29	Perform FILL_CHARG_STOCK_COMM
11	2.574.850	66.379	17	0,44	Open SQL MCHB	11	338.280	101	7,56	<0,01	Open SQL MCHB
3.281	1.878.077	56.023	12,4	0,37	Open SQL ZFABCLASSECARC	11	11.389	94	0,25	<0,01	Open SQL ZFABCLASSECARC
11	1.792.065	10.253	11,83	0,07	Perform FILL_CARATT_VIAGGIO	11	1.056.714	8.797	23,61	0,2	Perform FIL_CARATT_VIAGGIO
3.292	1.680.210	204.592	11,09	1,35	Select Single ZFABCLASSECARC	—	—	—	—	—	ISTRUZIONE NON NECESSARIA
3.281	1.578.485	168.152	10,42	1,11	Open Cursor ZFABCLASSECARC	11	766	531	0,02	0,01	Open Cursor ZFABCLASSECARC
522	1.474.550	5.310	9,74	0,04	Perform CONTROLL_MAG_FORN	522	252.582	4.825	5,64	0,11	Perform CONTROLL_MAG_FORN
522	1.469.158	27.386	9,7	0,18	Call Function Z_GET_ZMAGFORCLI	522	247.723	15.221	5,54	0,34	Call Function ZT_GET_ZMAGFORCLI
3.292	1.440.787	1.440.787	9,51	9,51	DB: Open ZFABCLASSECARC	11	237	237	0,01	0,01	DB: Open ZFABCLASSECARC
3.281	1.409.742	1.409.742	9,31	9,31	DB: Open ZFABCLASSECARC	1	17	17	<0,01	<0,01	DB: Open ZFABCLASSECARC
1.044	1.137.718	6.971	7,51	0,05	Perform FIND_NAME_CARATT	1.044	72.497	4.179	1,62	0,09	Perform FIND_NAME_CARATT
522	1.078.323	18.039	7,12	0,12	Call Function ZTSD_UBICAZIONE_FROM_BATCH	522	316.809	11.734	7,08	0,26	Call Function ZTSD_UBICAZIONE_FROM_BATCH
522	1.059.683	15.586	7	0,1	Call Function Z_GEST_PARTITONE	11	545.909	392	12,2	0,01	Native SQL: Execute Procedure Z_AMDP_GEST_PARTITONE
693	940.143	9.279	6,21	0,06	Perform FILL_CARA	693	696	216.117	6.712	4,83	0,15
522	890.491	4.533	5,88	0,03	Open SQL ZMAGFORCLI	1	2.128	9	0,05	<0,01	Open SQL ZMAGFORCLI
522	865.922	19.588	5,72	0,13	Perform CONTROLLA_UM_ODV	522	5.936	0,13	0,13	0,13	Perform CONTROLLA_UM_ODV
522	849.587	55.972	5,61	0,37	Fetch ZMAGFORCLI	1	2.012	160	0,04	<0,01	Fetch ZMAGFORCLI
522	793.615	793.615	5,24	5,24	DB: Fetch ZMAGFORCLI	1	1.852	1.852	0,04	0,04	DB: Fetch ZMAGFORCLI
682	776.590	4.595	5,13	0,03	Perform FIND_NAME_CARATT	682	57.593	3.141	1,29	0,07	Perform FIND_NAME_CARATT
522	680.003	11.111	4,49	0,07	Perform CONVERTI_QTA_UMB_PESO	522	7.750	3.281	0,17	0,07	Perform CONVERTI_QTA_UMB_PESO
522	662.196	9.209	4,37	0,06	Perform CLFM_SELECT_AUSP2	—	—	—	—	—	ESEGUITA TRAMITE ALTRE ISTRUZIONI
											Perform CLFM_SELECT_AUSP2

Il caso “Prova 1” risulta essere molto ottimizzato: si è passati da 15 secondi a 4,5 secondi.

7.4.1 ESITI PROVA 2

Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
1	2.263.711	0	100	0	Runtime analysis
1	2.263.407	727	99,99	0,03	Call Function ZT_EXTRACT_VIAGGI_01C
1	1.896.031	30	83,76	<0,01	Perform CALL_EXTR_VIAGGIO_DIST
1	1.896.001	29	83,76	<0,01	Call Function ZT_EXTRACT_LISTA_PRELIEVO_01C
1	1.893.962	14.923	83,67	0,66	Perform START_MAIN_EXTR_VIAGGIO
6	1.053.297	25.992	46,53	1,15	Perform FILL_PARTITE_Possibili
6	551.151	4.400	24,35	0,19	Perform FILL_CARATT_VIAGGIO
6	373.819	8.457	16,51	0,37	Perform FILL_CHARG_STOCK_COMM
1	353.664	29.216	15,62	1,29	Perform EXTR_PARTITE_PACCO_VIAGGIO
5	332.916	482	14,71	0,02	Call M_Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP
5	219.782	171	9,71	0,01	Native SQL: Execute Procedure "Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP#stb2
5	219.611	219.611	9,7	9,7	DB: Exec "Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP#stb2
6	216.581	53	9,57	<0,01	Open SQL MCHB
6	215.320	5.577	9,51	0,25	Fetch MCHB
11	209.743	209.743	9,27	9,27	DB: Fetch MCHB
275	195.902	6.542	8,65	0,29	Call Function ZTSD_UBICAZIONE_FROM_BATCH
275	189.360	42.802	8,37	1,89	Select Single CRHD
275	148.781	2.510	6,57	0,11	Perform CONTROLL_MAG_FORN
275	146.271	8.014	6,46	0,35	Call Function ZT_GET_ZMAGFORCLI
275	143.944	143.944	6,36	6,36	DB: Open CRHD
1.747	137.167	9.587	6,06	0,42	Call Function ZT_FIND_NAME_CARATT
7	135.417	953	5,98	0,04	Call M. {O:95*ZCL_GET_INOB_VALUES}->GET_OBJECT_USING_ZVT_CHARACT
275	135.307	2.349	5,98	0,1	Open SQL KNA1
6	134.530	181	5,94	0,01	Call M. {O:95*ZCL_GET_INOB_VALUES}->GET_INOB_VALUES
7	127.675	59	5,64	<0,01	Open SQL ZVT_CHARACT
1.747	127.580	123.362	5,64	5,45	Perform SELECT_TAB
7	126.877	514	5,6	0,02	Fetch ZVT_CHARACT

FUNZIONE INIZIALE						FUNZIONE OTTIMIZZATA					
Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia	Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
6	4.996.139	22.438	57,06	0,26	Perform FILL_PARTITE_Possibili	6	1.053.297	25.992	46,53	1,15	Perform FILL_PARTITE_Possibili
1	2.068.238	49.548	24,1	0,58	Perform EXTR_PARTITE_PACCO_VIAGGIO	1	353.664	29.216	15,62	1,29	Perform EXTR_PARTITE_PACCO_VIAGGIO
1.747	2.027.643	18.197	23,63	0,21	Call Function Z_FIND_NAME_CARATT	1.747	137.167	9.587	6,06	0,42	Call Function ZT_FIND_NAME_CARATT
1.747	2.009.446	20.386	23,42	0,24	Perform SELECT_TAB	1.747	127.590	123.362	5,64	5,45	Perform SELECT_TAB
6	1.403.762	157	16,36	<0,01	Perform FILL_CHARG_STOCK_COMM	6	373.819	8.457	16,51	0,37	Perform FILL_CHARG_STOCK_COMM
6	1.403.605	42.035	16,36	0,49	Open SQL MCHB	6	216.581	33	9,57	<0,01	Open SQL MCHB
1.741	1.041.824	30.982	12,14	0,36	Open SQL ZFABCLASSECARC	6	6.393	51	0,28	<0,01	Open SQL ZFABCLASSECARC
6	947.306	5.436	11,04	0,06	Perform FILL_CARATT_VIAGGIO	6	551.151	4.400	24,35	0,19	Perform FILL_CARATT_VIAGGIO
1.747	947.078	114.133	11,04	1,33	Select Single ZFABCLASSECARC	0	0	0	0	0,01	ISTRUZIONE NON NECESSARIA
1.741	876.764	95.985	10,22	1,12	Open Cursor ZFABCLASSECARC	6	442	309	0,02	0,01	Open Cursor ZFABCLASSECARC
1.747	813.374	813.374	9,48	9,48	DB: Open ZFABCLASSECARC	6	133	133	0,01	0,01	DB: Open ZFABCLASSECARC
1.741	780.284	780.284	9,09	9,09	DB: Open ZFABCLASSECARC	1	19	19	<0,01	<0,01	DB: Open ZFABCLASSECARC
275	763.089	2.756	8,89	0,03	Perform CONTROLL_MAG_FORN	275	148.781	2.510	6,57	0,11	Perform CONTROLL_MAG_FORN
275	759.516	14.321	8,85	0,17	Call Function Z_GET_ZMAGFORCLI	275	146.271	8.014	6,46	0,35	Call Function ZT_GET_ZMAGFORCLI
550	642.817	3.771	7,49	0,04	Perform FIND_NAME_CARATT	550	40.625	2.359	1,79	0,1	Perform FIND_NAME_CARATT
275	585.258	9.896	6,82	0,12	Call Function ZTSD_UBICAZIONE_FROM_BATCH	275	195.902	6.542	8,65	0,29	Call Function ZTSD_UBICAZIONE_FROM_BATCH
275	573.161	8.149	6,68	0,09	Call Function Z_GET_PARTITONE	62%	5	219.782	9.587	0,01	Native SQL: Execute Procedure "Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP#stb2"
275	533.583	10.959	6,22	0,13	Perform CONTROLLA_UM_ODV	275	2.660	2.660	0,12	0,12	Perform CONTROLLA_UM_ODV
378	503.814	4.750	5,87	0,06	Perform FILL_CARA	378	121.411	3.722	5,36	0,16	Perform FILL_CARA
677	462.733	168.469	5,63	1,96	Select Single MARA	11	11.396	3.232	0,5	0,14	Select Single MARA
275	465.401	2.385	5,42	0,03	Open SQL ZMAGFORCLI	1	2.225	9	0,1	<0,01	Open SQL ZMAGFORCLI
275	443.321	28.932	5,17	0,34	Fetch ZMAGFORCLI	1	2.114	1.56	0,09	0,01	Fetch ZMAGFORCLI
275	414.389	414.389	4,83	4,83	DB: Fetch ZMAGFORCLI	1	1.958	1.958	0,09	0,09	DB: Fetch ZMAGFORCLI
372	411.537	2.455	4,8	0,03	Perform FIND_NAME_CARATT	90%	550	40.625	2.359	1,79	0,1 Perform FIND_NAME_CARATT
275	373.339	5.932	4,35	0,07	Perform CONVERTI_QTA_UMB_PESO	275	4.656	1.944	0,21	0,09	Perform CONVERTI_QTA_UMB_PESO

Il caso “Prova 2” risulta essere molto ottimizzato: si è passati da 8,6 secondi a 2,3 secondi.

In proporzione, questa risulta essere il caso con il miglior guadagno in termini di prestazioni.

7.4.1 ESITI PROVA 3

Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
1	2.275.544	0	100	0	Runtime analysis
1	2.275.159	634	99,98	0,03	Call Function ZT_EXTRACT_VIAGGI_01C
1	1.969.493	49	86,55	<0,01	Perform CALL_EXTR_VIAGGIO_DIST
1	1.969.431	41	86,55	<0,01	Call Function ZT_EXTRACT_LISTA_PRELIEVO_01C
1	1.968.149	10.546	86,49	0,46	Perform START_MAIN_EXTR_VIAGGIO
7	933.335	16.228	41,02	0,71	Perform FILL_PARTITE_POSSIBILI
7	688.205	5.065	30,24	0,22	Perform FILL_CARATT_VIAGGIO
7	523.584	653	23,01	0,03	Call M. Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP
7	432.426	316	19	0,01	Native SQL: Execute Procedure "Z_AMDP_GEST_PARTITONE=>ESTRAZIONE_AUSP#stb2
7	431.468	431.468	18,96	18,96	DB: Exec "Z_AMDP_GEST_PARTITONE=>ESTRAZ"
1	289.363	15.808	12,72	0,69	Perform EXTR_PARTITE_PACCO_VIAGGIO
7	202.473	3.749	8,9	0,16	Perform FILL_CHARG_STOCK_COMM
7	165.481	403	7,27	0,02	Perform CACL_CLASS_CHARG2
8	164.390	1.257	7,22	0,06	Call M. {O:95*ZCL_GET_INOB_VALUES}->GET_OBJECT_USING_ZVT_CHARACT
7	163.739	208	7,2	0,01	Call M. {O:95*ZCL_GET_INOB_VALUES}->GET_INOB_VALUES
7	159.321	57	7	<0,01	Open SQL ZVN_CONF_V
7	158.097	374	6,95	0,02	Fetch ZVN_CONF_V
7	157.723	157.723	6,93	6,93	DB: Fetch ZVN_CONF_V
8	155.136	67	6,82	<0,01	Open SQL ZVT_CHARACT
8	154.196	721	6,78	0,03	Fetch ZVT_CHARACT
8	153.475	153.475	6,74	6,74	DB: Fetch ZVT_CHARACT
7	152.021	1.587	6,68	0,07	Perform FILL_POS_VIAGGIO
441	142.169	4.161	6,25	0,18	Perform FILL_CARA
7	131.029	176	5,76	0,01	Call Function CUCB_GET_OBJECT
7	130.243	75	5,72	<0,01	Call M. {O:54*CL_CUCB}->GET_OBJECT
14	124.451	333	5,47	0,01	Call M. {O:54*CL_CUCB}->GET_SINGLE_INSTANCE
7	121.285	292	5,33	0,01	Perform FILL_NOTE_POSIZIONE

FUNZIONE INIZIALE						FUNZIONE OTTIMIZZATA					
Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia	Ris.	Gross [microsec]	Net [microsec]	Gross [%]	Net [%]	Campo gerarchia
7	2.306.713	12.342	50,94	0,22	Perform FILL_PARTITE_POSSIBILI	7	933.335	16.228	41,02	0,71	Perform FILL_PARTITE_POSSIBILI
1.284	1.439.988	12.939	25,24	0,23	Call Function Z_FIND_NAME_CARATT	1.284	101.541	7.394	4,46	0,32	Call Function ZT_FIND_NAME_CARATT
1.284	1.427.049	15.015	25,01	0,26	Perform SELECT_TAB	1.284	94.147	89.384	4,14	3,93	Perform SELECT_TAB
1	1.147.115	25.838	20,1	0,45	Perform EXTR_PARTITE_PACCO_VIAGGIO	1	289.363	15.808	12,72	0,69	Perform EXTR_PARTITE_PACCO_VIAGGIO
7	1.129.698	6.167	19,8	0,11	Perform FILL_CHARG_STOCK_COMM	7	688.205	5.065	30,24	0,22	Perform FILL_CARATT_VIAGGIO
7	799.672	169	14,01	<0,01	Open SQL MCHB	7	202.473	3.749	8,9	0,16	Perform FILL_CHARG_STOCK_COMM
7	799.503	17.350	14,01	0,3	Open SQL MCHB	7	115.700	58	5,08	<0,01	Open SQL MCHB
1.277	744.490	22.269	13,05	0,39	Open SQL ZFABCLASSECARC	7	7.762	59	0,35	<0,01	Open SQL ZFABCLASSECARC
1.284	667.373	82.104	11,7	1,44	Select Single ZFABCLASSECARC	0					ISTRUZIONE NON NECESSARIA
1.277	625.180	68.411	10,96	1,2	Open Cursor ZFABCLASSECARC	7	1.377	550	0,06	0,02	Open Cursor HT_ZFABCLASSECARC
441	599.683	5.648	10,51	0,1	Perform FILL_CARA	7	142.169	4.161	6,25	0,18	Perform FILL_CARA
1.284	572.059	572.059	10,03	10,03	DB: Open ZFABCLASSECARC	7	139	139	0,01	0,01	DB: Open ZFABCLASSECARC
1.277	556.769	556.769	9,76	9,76	DB: Open ZFABCLASSECARC	1	12	12	<0,01	<0,01	DB: Open ZFABCLASSECARC
434	493.083	2.891	8,64	0,05	Perform FIND_NAME_CARATT	344	37.435	1.950	1,65	0,04	Perform FIND_NAME_CARATT
170	467.572	1.691	8,19	0,03	Perform CONTROLL_MAG_FORN	170	83.024	1.509	3,65	0,07	Perform CONTROLL_MAG_FORN
170	465.811	8.678	8,16	0,15	Call Function Z_GET_ZMAGFORCLU	170	81.493	4.645	3,58	0,2	Call Function ZT_GET_ZMAGFORCLU
340	381.674	2.336	6,69	0,04	Perform FIND_NAME_CARATT	434	37.435	1.950	1,65	0,09	Perform FIND_NAME_CARATT
170	359.691	5.106	6,3	0,09	Call Function Z_GET_PARTITONE	7	432.426	316	39	0,01	Native SQL: Execute Procedure Z_AMDP_GEST_PARTITONE
170	356.267	6.109	6,24	0,11	Call Function ZSD_UBICAZIONE_FROM_BATCH	101.337	3.597	4.45	0,1	Call Function ZTSD_UBICAZIONE_FROM_BATCH	
170	292.732	6.666	5,13	0,12	Perform CONTROLLA_UN_ODV	170	1.740	1.740	0,08	0,06	Perform CONTROLLA_UN_ODV
170	288.092	1.484	5,05	0,03	Open SQL ZMAGFORCLU	1	2.947	9	0,12	<0,01	Open SQL ZMAGFORCLU
170	275.508	18.073	4,83	0,32	Fetch ZMAGFORCLU	1	1.752	159	0,08	0,01	Fetch ZMAGFORCLU
170	257.435	257.435	4,51	4,51	DB: Fetch ZMAGFORCLU	1	1.593	1.593	0,07	0,07	DB: Fetch ZMAGFORCLU
170	231.675	3.045	4,06	0,05	Perform CLFM_SELECT_AUSP2	0					ESEGUITA TRAMITE ALTRE ISTRUZIONI
170	229.564	3.646	4,02	0,06	Perform CONVERTI_QTAUMB_PESO	170	2.900	1.230	0,13	0,05	Perform CONVERTI_QTAUMB_PESO

Questa funzione, in proporzione, risulta essere quella meno ottimizzata ma si è comunque passati dai 5,7 secondi iniziali ai 2,3 secondi finali, quindi il tempo è più che dimezzato.

8. FASE DI TEST

8.1 INTRODUZIONE

La fase di test è una delle più importanti poiché essa permette di verificare che tutte le operazioni di ottimizzazione svolte producano effettivamente gli stessi risultati per l'utente.

In questo caso per testare la funzione viene creato un programma chiamato “Z_PROVA_COMP_EXTR”.

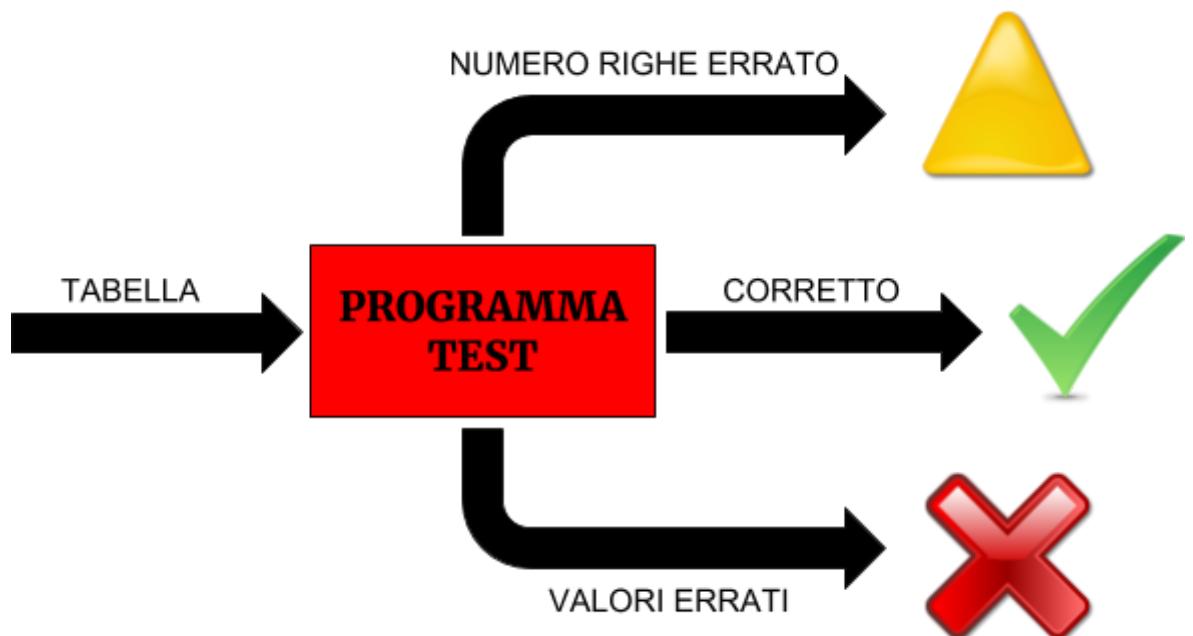
Il suo compito è quello di eseguire sia la funzione originale che quella ottimizzata per poi confrontare i tempi di esecuzione e i valori estratti dal database.

Il report deve inoltre indicare su quali tabelle sono stati riscontrati degli errori, indicando anche la loro tipologia (valori o numero di righe differenti).

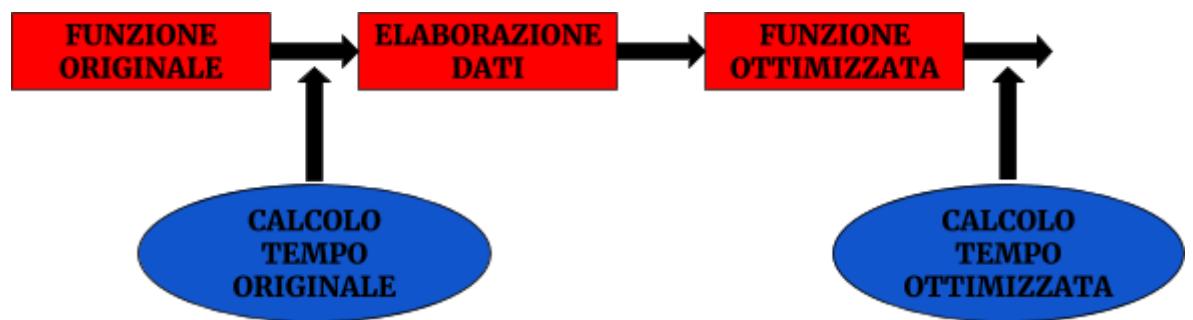
8.2 STRUTTURA PROGRAMMA

La struttura del programma si può suddividere in due parti, da una parte il controllo dei valori delle tabelle, e dall'altra il confronto fra i tempi di esecuzione.

- La correttezza di ogni tabella viene verificata nel seguente modo:



- Le tempistiche vengono calcolate nel seguente modo:



Il programma di test richiede all'utente alcuni parametri prima di iniziare il controllo:

- Divisione
- Luogo spedizione
- Viaggio
- Tipo estrazione
- Tipo esecuzione
- Ordine lancio funzioni
- Tempo minimo

Il tempo minimo indica il tempo (in millisecondi) sopra il quale si vogliono considerare i viaggi passati come input, ovvero se la funzione originale termina in un tempo inferiore a quello indicato, essa non viene nemmeno confrontata e si passa direttamente al viaggio successivo.

“Ordine lancio funzioni” indica se si vuole eseguire per prima la funzione originale o quella ottimizzata.

La schermata di input del test è la seguente:

Prova comparazione funzioni estr.viaggi



DIVISIONE	<input type="text" value="A"/>
LUOGO SPEDIZIONE	<input type="text"/> A <input type="button" value="..."/>
VIAGGIO	<input type="text"/> A <input type="button" value="..."/>
TIPO ESECUZIONE	<input type="text" value="VS"/>
TIPO ESTRAZIONE	<input type="text"/>
TEMPO MINIMO (millisecondi)	<input type="text" value="1.000"/>

Ordine lancio funzioni

FUNZIONE INIZIALE
 FUNZIONE OTTIMIZZATA

Come specificato in precedenza, questo test è utile per capire dove si sono commessi degli errori di programmazione. Di seguito viene mostrato come:

Divisione	Luogo	Sped.	Viaggio	T INIZ	T OTT	PERC	TESV	POS	POS1	CARA	NPOSV	NVIAG	CHARG	PACCO	CARAC	TARG	ERR	ZCHARG	ZPAL	POS A	TES A	ZCHARG A
0103			0000000000300000	22.185	62.348	-181%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓	
0103			0000000000300001	52.213	70.760	-36%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000000000100000	47.958	63.733	-33%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000110	7.323	7.038	4%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
0103			0000000000300002	41.622	59.750	-44%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000106	7.103	7.458	-5%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000107	6.994	7.174	-3%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000102	7.024	7.105	-1%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000103	7.026	7.616	-8%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000104	7.137	7.034	1%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000105	7.333	6.940	5%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000109	7.233	6.970	4%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000108	7.318	6.996	4%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000101	9.352	9.209	2%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			00000070000002	7.328	6.996	5%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000111	51.600	61.909	-20%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000201	34.737	60.541	-74%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000203	36.742	58.855	-60%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000204	7.277	7.299	0%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000205	7.251	7.587	-5%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000206	7.029	7.483	-6%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000207	7.447	7.264	2%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000208	52.151	60.553	-16%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000209	51.956	60.416	-16%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			000000001000001	38.415	67.638	-76%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000112	54.814	64.110	-17%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000113	54.526	62.523	-15%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000114	53.197	65.840	-24%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000115	55.938	64.061	-15%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000101	8.497	7.340	14%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103			0000007000000202	35.723	58.397	-63%	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Da questa immagine si può dedurre che la tabella POSVIAGGIO1 (POS1) ha un numero di righe differente da quella originale, mentre le tabelle NOTETESVIAGGIO (NVIAG) e T_ZSGTP_PAL (ZPAL) presentano errori sul contenuto.

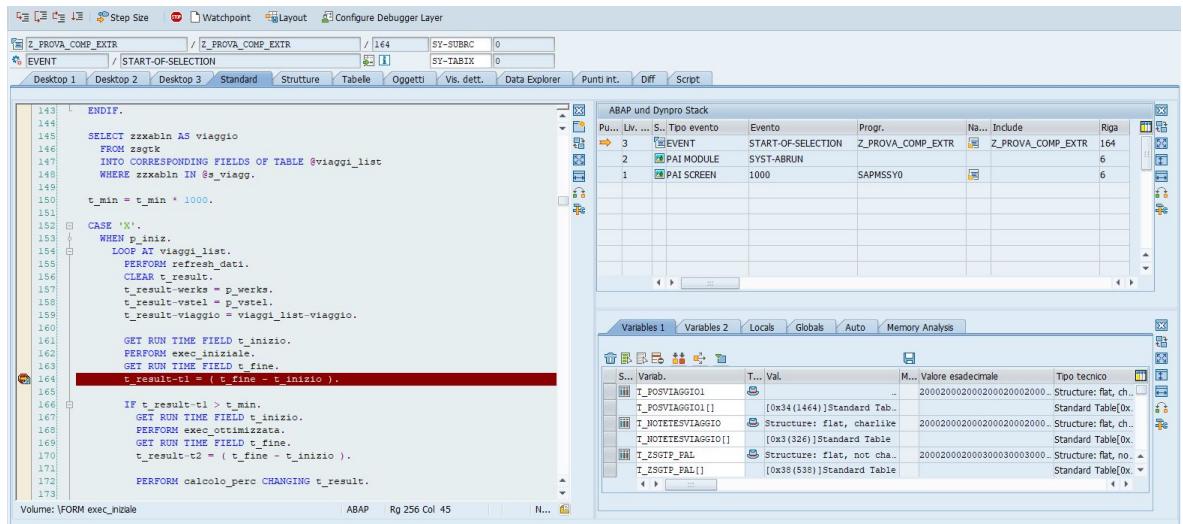
Come si nota da questo esempio di test le funzioni risultano quasi tutte peggiorate in termini di prestazioni, questo perché sono stati inseriti dei viaggi senza contenuti significativi (che quindi l'utente non cercherà mai), e per questo è stato inserito il tempo minimo di esecuzione.

8.3 DEBUGGING

Qualora si siano riscontrati degli errori su qualche tabella, il metodo più semplice è quello di monitorare la modifica delle tabelle attraverso il Debugger ABAP.

Questo Debugger offre numerose possibilità ma in questo caso bastano solamente due metodi:

- Quando si conosce la riga di codice che si vuole analizzare si utilizza un BREAKPOINT.
- Quando si conosce il nome della variabile (in questo caso una tabella) che si vuole analizzare si utilizza un WATCHPOINT.



In questo caso si stanno monitorando i valori contenuti nelle tabelle prese in esame dal test nel capitolo 8.3.

Osservando come vengono aggiornati i valori delle tabelle si possono individuare gli errori commessi.

8.4 RISULTATI TEST

In questo paragrafo verranno effettuati i test sui tre casi presi in considerazione in questo progetto, ovvero “Prova 1”, “Prova 2” e “Prova 3”.

Inserendo come input i viaggi dei rispettivi tre casi si ottiene il seguente output:

Divisione	LuogoSped.	Viaggio	T INIZ	T OTT	DELTA	PERC	TESV	POS	POS1	CARA	NPOSV	NVIAG	CHARG	PACCO	CARAC	TARG	ERR	ZCHARG	ZPAL	POS A	TES A	ZCHARG A
0103		0000000006897485	16,88	5,05	11,83	70	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901235	9,34	2,31	7,03	75	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901497	6,25	2,13	4,12	66	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Questi sono i risultati del test eseguendo per prima la funzione originale.

Se invece si esegue per prima la funzione ottimizzata si ottengono i seguenti risultati:

Divisione	LuogoSped.	Viaggio	T INIZ	T OTT	DELTA	PERC	TESV	POS	POS1	CARA	NPOSV	NVIAG	CHARG	PACCO	CARAC	TARG	ERR	ZCHARG	ZPAL	POS A	TES A	ZCHARG A
0103		0000000006897485	16,12	5,17	10,95	68	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901235	9,13	2,37	6,76	74	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901497	6,17	2,27	3,90	63	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Si può notare che se viene eseguita per prima la funzione ottimizzata la percentuale di miglioramento diminuisce: ciò accade perché il database memorizza alcuni dati in memoria, quindi la funzione che viene lanciata per seconda ne trarrà dei vantaggi.

I tempi di esecuzione non sono sempre gli stessi poiché la velocità di elaborazione dei dati dipende anche dal carico attuale del database.

In conclusione si può dedurre che il test per questi tre casi è stato superato dato che non si sono riscontrati errori.

Per controllare in modo migliore se effettivamente non vi siano errori di programmazione si può effettuare un lancio massivo sul test contenente tutti i viaggi esistenti.

Una volta eseguita la funzione per tutti i viaggi esistenti tramite il test si ottiene il seguente output:

Divisio...	LuogoSped.	Viaggio	T INIZ	T OTT	DELTA	PE...	TESV	POS	POS1	CARA	NPOSV	NVIAG	CHARG	PACCO	CARAC	TARG	ERR	ZCHARG	ZPAL	POS A	TES...	ZCHARG A
0103		0000000006900062	6,95	1,25	5,70	82	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898990	5,43	1,00	4,43	82	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898986	5,32	0,98	4,34	82	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898980	10,19	1,98	8,21	81	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898747	7,83	1,50	6,35	81	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898985	5,50	1,06	4,44	81	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898982	5,39	1,02	4,37	81	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898988	11,31	2,21	9,10	80	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898978	10,60	2,11	8,49	80	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898999	5,42	1,07	4,35	80	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898984	5,38	1,06	4,30	80	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898987	5,30	1,05	4,25	80	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898477	9,08	1,90	7,18	79	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006899165	2,83	0,59	2,24	79	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006899891	13,87	3,08	10,79	78	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898991	5,93	1,33	4,60	78	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898538	4,93	1,08	3,85	78	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006901425	4,44	0,99	3,45	78	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898979	3,54	0,77	2,77	78	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006899164	2,74	0,61	2,13	78	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006897981	18,49	4,26	14,23	77	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006901179	2,99	0,69	2,30	77	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006901423	14,80	3,50	11,30	76	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006899811	10,72	2,59	8,13	76	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006897897	5,52	1,33	4,19	76	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006898992	2,91	0,71	2,20	76	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006899166	2,73	0,65	2,08	76	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006901220	2,11	0,50	1,61	76	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006896509	17,90	4,53	13,37	75	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006901519	11,88	3,01	8,87	75	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006900402	7,51	1,91	5,60	75	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0103		0000000006896300	3,96	1,01	2,95	75	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

I viaggi presenti in questa immagine sono solo alcuni dei viaggi esistenti, ma la funzione risulta corretta per ogni viaggio, quindi il test finale sulla correttezza può considerarsi superato.

Una volta che è stato superato il test di correttezza delle tabelle bisogna verificare che la funzione risulti ottimizzata per ognuno dei viaggi presenti.

Per fare questo è sufficiente controllare la percentuale di miglioramento dei casi peggiori del lancio massivo effettuato in precedenza.

I risultati ottenuti sono i seguenti:

Divisio...	LuogoSped.	Viaggio	T INIZ	T OTT	DELTA	PE..	TESV	POS	POS1	CARA	NPOSV	NVIAG	CHARG	PACCO	CARAC	TARG	ERR	ZCHARG	ZPAL	POS A	TES...	ZCHARG A
0103		0000000006896298	1,96	0,85	1,11	57	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006898357	3,31	1,45	1,86	56	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900200	1,56	0,69	0,87	56	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901630	4,79	2,15	2,64	55	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006899614	2,61	1,17	1,44	55	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006899898	1,33	0,61	0,72	55	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006898363	5,59	2,58	3,01	54	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006899522	1,56	0,71	0,85	54	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006894422	1,29	0,59	0,70	54	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900860	7,52	3,56	3,96	53	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006899592	3,01	1,41	1,60	53	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900311	2,36	1,11	1,25	53	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900415	1,63	0,76	0,87	53	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006891287	2,26	1,09	1,17	52	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900647	1,65	0,79	0,86	52	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006896415	1,29	0,63	0,66	51	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006896535	2,44	1,22	1,22	50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900265	2,17	1,08	1,09	50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900610	1,09	0,54	0,55	50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900705	2,75	1,39	1,36	49	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900029	1,07	0,55	0,52	48	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006893738	4,16	2,20	1,96	47	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006892364	2,27	1,22	1,05	46	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006898396	1,72	0,94	0,78	46	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006899732	1,73	0,96	0,77	45	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901202	1,00	0,55	0,45	45	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006899564	2,84	1,65	1,19	42	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006901256	1,68	0,98	0,70	42	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006894413	1,92	1,14	0,78	41	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900523	1,97	1,20	0,77	39	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006898095	1,06	0,66	0,40	38	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006891539	1,02	0,65	0,37	36	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
0103		0000000006900309	2,11	1,38	0,73	34	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

In questa immagine sono visualizzati i viaggi con la percentuale di miglioramento più bassa eseguendo per prima la funzione ottimizzata, quindi il caso peggiore in cui ci si possa trovare.

Analizzando questo output si può notare che la percentuale di miglioramento non scende al di sotto del 34% quindi anche il test relativo ai tempi può considerarsi superato per ognuno dei viaggi esistenti.

Avendo superato completamente la fase di test il progetto può considerarsi concluso con successo.