POLITECNICO DI TORINO

Corso di Laurea Magistrale
In Ingegneria Informatica

Tesi di Laurea Magistrale

Progetto e implementazione di un sistema di monitoraggio

near real-time di applicativi in ambito bancario

Relatore
prof. Giovanni Malnati

Candidato
Claudio Branciforte

Anno Accademico 2019/2020

Sommario

1. L'attività dell'Ufficio Incident Managment e Monitoraggi

L'attività svolta nell'Ufficio Incident Managment e Monitoraggi, si occupa di monitorare i sistemi core banking di intesa Sanpaolo e di gestire il coordinamento degli Incident della direzione sistemi informativi, l'ufficio è suddiviso in varie attività che coprono diverse aree, dal monitoraggio degli ATM, il monitoraggio dei principali processi di erogazione dei servizi (i mutui ad esempio) al mondo dei pagamenti.

Nello specifico l'attività in cui sono allocato si chiama Strumenti di Monitoraggio applicativo e si occupa del monitoraggio di tutto l'ambito dei pagamenti da e verso Intesa Sanpaolo e più in generale di monitoraggio di Processi di Business.

1.2 MOBU: Monitor di Business

L'attività si occupa di fornire ai referenti applicativi dei processi monitorati, dei cruscotti da cui possono monitorare lo stato dei vari pagamenti o dei vari processi di pagamento in cui viene coinvolta la capogruppo, le Banche Tramitate (le banche che per alcuni determinati tipi di pagamenti si appoggiano a Banca Intesa Sanpaolo) e le banche del gruppo Intesa, a tal scopo, sono stati sviluppati degli applicativi web che offrissero una visione su questi tipi di processi.

Allo stato attuale, i cruscotti sono ventuno, ognuno di loro si prende carico di monitorare una determinata area di pagamento, bonifici, SEPA Direct DEBIT, Stipendi, Pagamenti tramite Internet, Pagamenti P2P, che com'è facile intuire, rappresentano una delle attività core della banca.

Un pagamento è un processo che, per essere portato a termine, vede il coinvolgimento di più applicativi bancari (che chiamerò *Attori*) che concorrono alle varie fasi di lavorazione di esso fino a portarlo ad uno stato finale. Ognuno di essi, si può fare afferire ad una determinata area in base al contesto. Ogni Attore può svolgere diverse lavorazioni in processi di pagamento diversi, data questa loro natura è stata definita una entità logica chiamata *Componente* che modella appunto una peculiarità interna dell'attore o semplicemente una diversa funzione svolta da questo all'interno del processo. Ed è quindi nelle componenti che si può definire il nodo del processo, che abbiamo chiamato *Misura*. Una misura quindi è una rappresentazione logica di un particolare *nodo di processo*. In questo modo abbiamo definito una struttura gerarchica che ci permette di modellare i processi di business che ci vengono richiesti di monitorare

In figura è mostrata l'anagrafica delle misure che compongono il processo chiamato "BONSCT", in questa tabella abbiamo anche l'ID_INPUT che corrisponde all'alimentazione che popola quel nodo, l'ID_PROC che indica l'evoluzione del processo e il campo LIV_PROC che indica se quel nodo è un nodo finale del processo.

La peculiarità che offre il sistema di monitoraggio è che su ogni misura, si possono definire uno o più controlli di vario tipo, che offrono un sistema di

Results | Messages

	ID_INPUT	ID_COMPONENTE	ID_MISURA	NOME_MISURA	DESC_MISURA	MISURA	ID_PROC	TIPO_PROC	LIV_PROC
1	BONPOLI_INFOG	CM005	MI044	PiazzNBE	PiazzNBE	NBE	015	BONSCT	NULL
2	BONPOLI_INFOG	CM005	MI031	Parch. GEFLU	Parcheggiato su GEFLU	NBE	020	BONSCT	NULL
3	BONPOLI_INFOG	CM005	MI032	InLav_NBE	InLav_NBE	NBE	030	BONSCT	NULL
4	BONPOLI_INFOG	CM005	MI034	SpedRBO_Tramit	SpedRBO - Tramitato	NBE	040	BONSCT	END
5	BONPOLI_INFOG	CM005	MI036	Invio_A_Dovetail	Invio a Dovetail	NBE	050	BONSCT	NULL
6	BONPOLI_INFOG	CM005	MI037	Revocati	Flussi revocati da Nuovo Back End	NBE	060	BONSCT	END
7	BONPOLI_INFOG	CM005	MI062	Sped_Estero	Sped_Estero	NBE	061	BONSCT	END
8	BONPOLI_BONIFICI	CM006	MI040	InLav_DT_Futura	InLav_DT_Futura	DoveT	065	BONSCT	NULL
9	BONPOLI_BONIFICI	CM006	MI038	InLav_DT_Automatica	InLav_DT_Automatica	DoveT	070	BONSCT	NULL
10	BONPOLI_BONIFICI	CM006	MI039	InLav_DT_Manuale	InLav_DT_Manuale	DoveT	070	BONSCT	NULL
11	BONPOLI_BONIFICI	CM006	MI061	Repair	Repair	DoveT	070	BONSCT	NULL
12	BONPOLI_BONIFICI	CM006	MI041	Lavorati DT	Lavorati DT	DoveT	080	BONSCT	END
13	BONPOLI_BONIFICI	CM006	MI042	Fatal	In Fatal su Dovetail	DoveT	090	BONSCT	NULL
14	BONPOLI_BONIFICI	CM006	MI043	Cancelled	Cancelled su Dovetail	DoveT	100	BONSCT	END

segnalazione atto a informare gli Utenti del cruscotto (applicativi o gruppi preposti al presidio dei cruscotti) che potrebbe esserci una anomalia in qualche oggetto del processo o in qualche attore del processo. Un tipico esempio di controllo effettuato sulle misure può essere dato dal tempo di permanenza sul nodo, quindi verrà definita una soglia temporale oltre la quale quella condizione potrebbe essere sintomo di un problema.

1.3 L'Attività di Funzionamento

Per attività di funzionamento, o funzionamento , in generale, si intende l'attività di presidio e supporto di primo e secondo livello svolti verso gli utenti che utilizzano un applicativo aziendale. Risoluzione di ticket, bug-fixing, micro-evolutive fanno parte di questo tipo di attività e sono svolte in Intesa Sanpaolo, cosi come in molte altre aziende, da personale interno e da personale esterno.

Questa attività rappresenta una voce di costo importante per ogni azienda, un costo che non è capitalizzabile , come può essere lo sviluppo di un nuovo applicativo, perché non è ammortizzabile nel tempo inoltre come ben si può immaginare, per ogni nuovo applicativo rilasciato si introduce sempre una quantità, seppur minima, di possibilità di errori. Per definire meglio le dimensioni dei volumi di investimenti fatti nel campo IT da Intesa Sanpaolo è significativo sapere che la Direzione Sistemi Informativi (DSI), da sola è al primo posto in Italia per spesa nel campo IT, il che la costituisce come la prima azienda in campo IT in Italia per capitali investiti.

Nel corso degli anni si sono sviluppati una moltitudine di applicativi, stratificati tra loro, che interagiscono tra loro e comunicano tra loro. Sono nate

nuove tecnologie e altre sono diventate obsolete, che vedevano la necessità di vari livelli di presidio per non degradare i livelli di servizio e la qualità di esso.

Chi si trova a dover fare attività di supporto spesso lo fa per applicativi che non conosce alla perfezione perché probabilmente non ha sviluppato direttamente ho dei vincoli operativi dovuti a policy aziendali o ad accordi contrattuali presi tra le aziende. Molto spesso il passaggio di consegne non è fatto nel modo corretto o la documentazione non è esaustiva, tutti questi fattori si riflettono direttamente sul costo da sostenere per questo tipo di attività ed hanno sottolineato la necessità di creare delle viste generalizzate per poter seguire questi applicativi, per poter categorizzare i determinati tipi di errori o problemi e rendere quindi più efficace l'attività di funzionamento. Anche nella realtà aziendale in cui mi trovo, ci sono esempi di questo genere, uno per tutti è un tool utilizzato dalla Gestione Operativa che presidia 24x7, sia a livello sistemistico che applicativo, alcuni ambiti core della banca. A questo tool ci si può integrare in modo automatico, tramite web services, per segnalargli allarmi, problemi, malfunzionamenti che vengono presi in carico da loro e gestiti secondo delle action List concordate in precedenza che dovrebbero portare alla risoluzione del problema. Questo descrive molto bene come, la conoscenza degli applicativi o dei sistemi sia a livello molto alto, di conseguenza, se il problema non viene risolto seguendo la action list, questi procedono a ingaggiare il reperibile di turno , referente per quel determinato ambito, che provvederà a sanare la situazione.

C'è da sottolineare che non tutti gli applicativi sviluppati necessitano una reperibilità costante o un presidio 24x7, ma soltanto quello svolto durante l'orario lavorativo settimanale, ed è questo il caso in cui mi sono trovato io

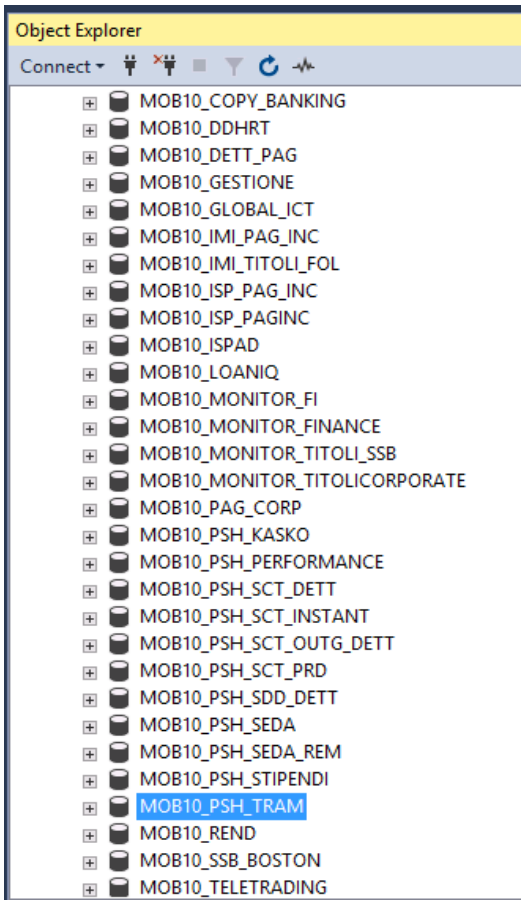
quando sono stato assunto in azienda. Dover svolgere il funzionamento per applicativi di cui non avevo scritto il codice, con caratteristiche simili, ma non uguali e logiche di funzionamento spesso non esaustivamente documentate. Era già presente quindi il bisogno di poter analizzare, ed accentrare tutta una serie di problematiche per riuscire ad avere uno sguardo più ampio per rendere questa attività più efficiente e proattiva.

2. L'architettura di MOBU

L'architettura di un cruscotto di monitoraggio si può suddividere in tre parti che andrò ad analizzare per poter contestualizzare l'ecosistema su cui ho effettuato il mio lavoro.

Le tre parti fondamentali sono la parte software costituita da eseguibili batch e servizi, il database ed il front end.

Per ogni cruscotto di monitoraggio, abbiamo un database dedicato con le proprie basi dati ma moltissime strutture identiche che modellano il flow di lavoro di ogni cruscotto.



2 Elenco di Alcuni Moduli

2.2 Il frontend

La parte front-end di un cruscotto è un'applicazione MVC .NET in cui la parte grafica è stata realizzata in React o AngularJS. All'interno di esso non vi è alcun tipo di logica di manipolazione del dato, infatti, questa parte del cruscotto è totalmente supportata dalle stored procedures sul database e da molte tabelle di anagrafica. Tale scelta è stata fatta per avere una maggiore configurabilità ed una totale riutilizzabilità del progetto web, che infatti può essere usato come "stampo" e poi personalizzato tramite le anagrafiche preposte e l'opportuno file di configurazione.

2.3 Il backend

Un cruscotto di monitoraggio è definito solitamente da un Database dedicato.
In questo database sono presenti tutti gli elementi necessari a costruire un modulo di monitoraggio. Vi sono degli oggetti che sono replicati per ogni modulo, in quanto facenti parte della struttura del modulo, del suo funzionamento e della sua configurazione. Questi oggetti si possono individuare in varie tabelle di anagrafica, le tabelle dedicate alla costruzione del front-end, le tabelle di log.

Peculiari per ogni modulo invece sono le tabelle che si riferiscono ai vari flussi alimentati, chiamate tutti con lo stesso tipo di nomenclatura, per ogni input troviamo rispettivamente

- T_BULK_{NOME_INPUT}
- T_INPUT_{NOME_INPUT} _APP
- T_INPUT_{NOME_INPUT}
- T_INPUT_{NOME_INPUT} _STO

Oltre alle tabelle ci sono anche dei set di stored procedures che sono replicate, altre invece, mantengono una nomenclatura simile ma poi sviluppano una logica che è peculiare per il modulo. Questo tipo di organizzazione è ottimale per effettuare la manutenzione su un gran numero di moduli che condividono una logica simile tra loro.
Suddividiamo le stored procedures in quattro categorie, ognuna delle quali ha una radice particolare nel nome o un particolare schema.

- Amm_SP_{nome_procedura}: sono stored per l'amministrazione delle anagrafiche o per effettuare qualche operazione nel database.
- Car_SP_{nome_procedura}: sono stored procedures utilizzate per il caricamento dei file e la prima elaborazione verso le tabelle di input
- [FE].mvc_SP{nome_procedura} : sono le stored procedures che servono al render del front-end, appartengono allo schema FE sono comuni in tutti i moduli.
- [FE_CUSTOM].{nome_procedura}: Sono stored procedures per il front-end, per recuperare dati, ma queste sono peculiari per ogni modulo.

2.4 Il processo di caricamento del dato

Dopo aver analizzato brevemente la struttura del database e la sua organizzazione, passo ad analizzare due elementi anch'essi fondamentali e il loro processo di lavoro.

Il primo elemento è un servizio windows installato in un server, chiamato MOB10_GESTORE_FILE. Questo svolge, principalmente due fondamentali compiti

- Smistamento dei file che alimentano i vari cruscotti, nelle opportune cartelle di input create per ogni modulo.
- Schedula gli eseguibili per il caricamento del dato, o per l'acquisizione dello stesso.

Questo servizio rimane in polling su una data cartella preposta per ricevere i file provenienti da un middleware FTP così che all'arrivo di un file, basandosi su una anagrafica, riesce e a smistare lo stesso nella corretta posizione ed invocare un eseguibile che è preposto al caricamento del dato, dal file alle tabelle di INPUT.

Il secondo elemento chiave per il caricamento dei dati e la loro elaborazione è un eseguibile windows preposto ad applicare la logica di caricamento del dato di quel determinato modulo. Tale logica è pilotata da una tabella di anagrafica salvata nel database di ogni modulo chiamata T_ANAGRAFICA_CMD_LOAD, dove sono contenuti tutti i comandi e i passi per caricare il dato sulla base dati, per popolare i nodi di processo ed infine per eseguire tutti i controlli attestati sui nodi. Il comportamento di que-

sto eseguibile è fondamentale per l'analisi del mio lavoro, quindi provvederò ad esporre maggiormente in dettaglio il funzionamento logico di questo elemento.

All'arrivo di uno o più file sulla cartella di input del modulo, viene invocato l'eseguibile di caricamento, a cui viene passato come parametro l'ID_INPUT che è stato appena spostato e la lista dei file che ha in elaborazione. Come prima operazione il caricatore sposta tutti i file per quell'input in una cartella nominata DA_ELAB che indica che in quel momento, si stanno caricando il o i file di quel determinato ID_INPUT. Tramite questo id, viene selezionata sulla tabella T_ANAGRAFICA_CMD_LOAD la lista dei comandi da eseguire per caricare i files presenti in quel momento, categorizzati in tre tipologie: CORPO_CARICAMENTO, POST_CARICAMENTO, ALL

- CORPO_CARICAMENTO: questi comandi, portano il dato nella base dati e popolano i nodi delle misure
- POST_CARICAMENTO: in questa fase, vengono eseguiti i controlli ed eventuali azioni di segnalazioni, come l'invio di e-mail di segnalazione

ALL: Questi comandi vengono eseguiti qualunque sia il file appena caricato
Il primo set di comandi eseguiti è finalizzato a portare il dato sulla prima tabella di appoggio chiamata T_BULK_{NOME_INPUT}, questa operazione viene effettuata mediante una stored procedures chiamata carsp_BulkOpenRowSet che utilizza la funzione BULK di SQL che permette di caricare dei dati in una tabella destinataria partendo da un provider di dati esterno o da un file al quale è possibile associare un file di formato

come parametro che descrive al comando BULK quali sono i campi che troverà nel file, quale sarà il separatore e le lunghezza di ogni colonna.

Dalla tabella di BULK, i dati vengono trasferiti in una seconda tabella di appoggio, chiamata T_INPUT_{NOME_INPUT}_APP, durante questo passaggio i dati vengono formattati, elaborati per estrarre informazioni di business o tradurre alcuni codici di stato in un formato più leggibile per l'utente, queste tabelle di appoggio, hanno già la definizione di un primary key composta di solito da un sub set di attributi e tale definizione del vincolo sarà mantenuto pure sulle tabelle di INPUT e sulle corrispondenti tabelle di storico.

L'ultimo passaggio dalla tabella di appoggio alla tabella di INPUT, vera e propria base dati per quel determinato attore, viene fatto tenendo conto dei dati già presenti in modo da non replicarli (operazione comunque impossibile data la definizione del vincolo di integrità) e tenendo conto di altre regole di business per l'elaborazione del dato.

Arrivati a questo punto del processo, i dati caricati, si possono considerare consolidati. Invece, nei punti precedenti (ovvero il trasferimento nella tabella di BULK e nella tabella di appoggio) questo assunto non è vero, infatti ad ogni nuovo ciclo di caricamento le tabelle di BULK e appoggio vengono svuotate del tutto dai dati presenti perché servono solamente per il caricamento del file.

L'eseguibile di caricamento, ha pure una logica di gestione degli eventi imprevisti: errori avvenuti durante la fase di caricamento, timeout dei comandi eseguiti, indisponibilità del database o errori generati da SQL Server (violazioni dei vincoli referenziali o errori di sintassi durante la costruzione

dinamica di una query); Quando uno di questi eventi accade, l'eseguibile sposta il file corrente e quelli in coda in una cartella preposta che è nominata FILE_ERRATI, logga l'azione sia sul database (in cui è presente un log dei comandi di caricamento) e termina la sua esecuzione attendendo di essere eseguito per un nuovo ciclo di caricamento.

3. L'analisi del problema: I file scartati e i falsi positivi.

Come spiegato nel paragrafo precedente, durante il caricamento possono accadere una serie di errori che causano lo scarto di uno o più file e per la precisione dei file che in quel momento si trovano nella cartella DA_ELAB, questo succede indipendentemente dal punto in cui l'errore è avvenuto.
Una parte delle attività di funzionamento, consisteva nel mantenere sotto controllo i log di caricamento affinché, nel caso di un file scartato, si potesse provvedere a mandarlo nuovamente in elaborazione. Tale processo che ad una prima analisi potrebbe sembrare un banale taglia, incolla, in realtà per vari vincoli di sicurezza era molto oneroso, il motivo è che con le utenze di produzione di un dipendente normale non era possibile copiare, tagliare, modificare un file dalle cartelle di produzione, e per questo motivo si usava un eseguibile in cui andavano censiti a mano i path e i nomi dei file che si dovevano spostare, questo processo, in giornate in cui per cause esterne le istanze dei database avevano dei problemi , diventava lungo e tedioso. In aggiunta, per alcuni moduli era fondamentale non mandare in esecuzione due volte lo stesso file in quanto mancavano dei vincoli di integrità referenziale dovuto alla particolarità dei processi e al modo di lavorazione di alcuni enti bancari, questa premessa è doverosa in quanto durante il lavoro di funzionamento era emerso che spesso, i file presenti nella cartella FILE_ERRATI, in realtà erano dei falsi positivi, ovvero il

dato era già presente in base dati e consolidato ma il file era stato scartato lo stesso, di conseguenza il solo monitorare il file di LOG non era condizione che garantisse una corretta attività. A quel punto si doveva procedere con una analisi più approfondita, andando ad analizzare le basi dati si aveva la certezza se un file fosse o meno stato già elaborato.

Per i motivi descritti sopra, spesso a causa di file scartati e non rielaborati in tempo, provocavano dei ritardi di monitoraggio che facevano scattare molti allarmi, si mantenevano le basi dati molto popolate perché gli oggetti ancora in fase di monitoraggio non vengono storicizzati e di conseguenza, quando si andava ad elaborare i nuovi dati i volumi che entravano in gioco erano davvero onerosi, tutto questo comportava un degrado della qualità del monitoraggio che , con oltre quindici cruscotti , diventava non accettabile.

3.2 La Richiesta di ISP

A fronte di queste problematiche e questa inefficienza nell'ambito del funzionamento, è arrivata da parte del mio responsabile, la richiesta di trovare una soluzione che riuscisse a individuare in modo automatico i file scartati, annullando tutti i casi di falsi positivi, dando anche evidenza ad alcune caratteristiche dei file. Dall'analisi dei fabbisogni sono emersi alcuni requisiti chiave che la soluzione dovesse avere, oltre a dei vincoli a cui dovevo attenermi nello sviluppo e nella architettura, andrò a riassumerli qui in seguito

Vincoli:

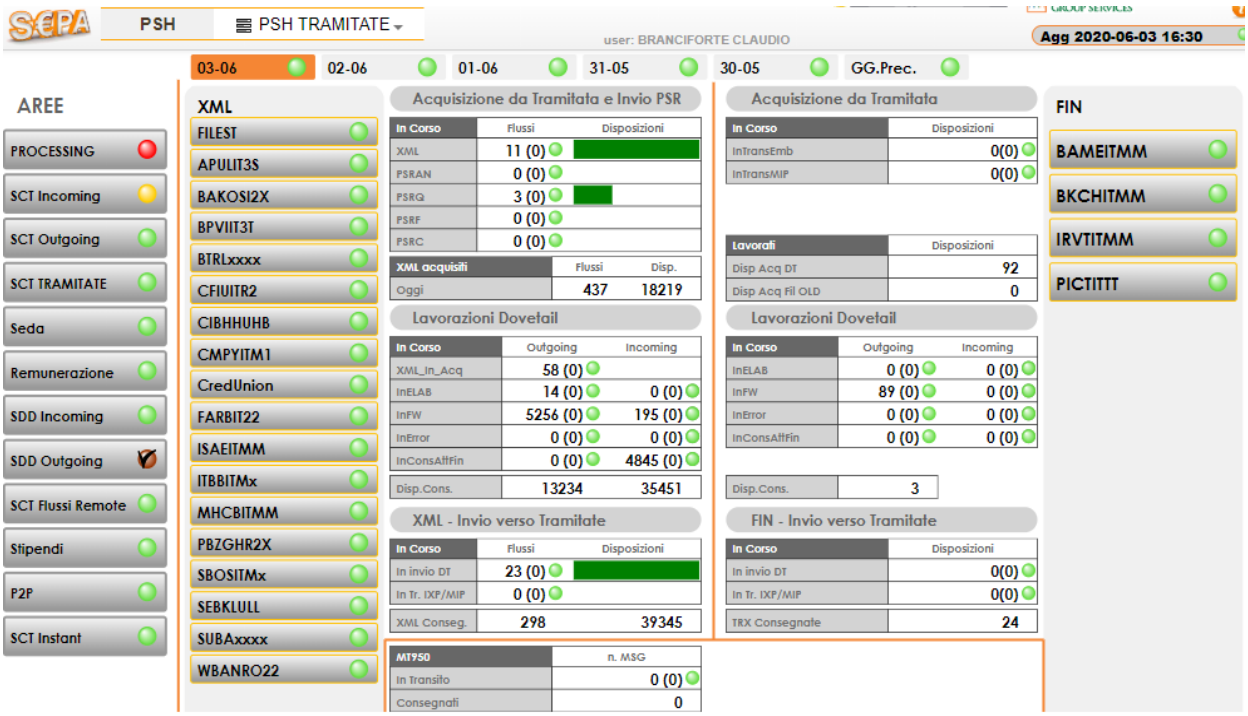
- Stack .NET (sia per il backend che per il frontend)

- Integrazione nell'attuale architettura MOBU
- Effort di sviluppo totalmente interno
- Nessuna modifica agli eseguibili di caricamento

Requisiti:

- Eliminare Segnalazioni di Falsi Positivi
- Tracciare un file scartato con informazioni a corredo
- Tracciare gli errori SQL del Modulo
- Funzione per rielaborare un file da una dashboard
- Ricerca sul log del modulo della istruzione causa dello scarto
- Chiusura automatica di una segnalazione quando in file viene ri-sottomesso e caricato correttamente
- Integrazione semplice in altri cruscotti

Al fine di soddisfare i vincoli di sviluppo imposti e i requisiti mi sono focalizzato su un solo cruscotto, sapendo che grazie alla architettura replicata con cui sono realizzati i vari moduli avrei ottenuto una facile integrazione



pure negli altri. Il modulo su cui mi sono voluto concentrare si chiama PSH Tramitate, che monitora i bonifici SCT delle Banche Tramitate di Intesa Sanpaolo, si tratta del modulo dove la numerosità di file ricevuti è la più alta e in cui ci sono volumi considerevoli di dati.

4. L'architettura Proposta

Per la realizzazione di questa funzione di monitoraggio ho deciso di mantenere l'architettura di MOBU e di sviluppare una applicazione web .NET MVC per la parte di visualizzazione degli allarmi, la consultazione, la ricerca e la gestione di un file o gruppo di file o di una segnalazione. Ho invece sviluppato una serie di stored procedures per la parte di elaborazione del dato, ho voluto sfruttare alcuni elementi già presenti su MOBU, come il gestore file, per schedulare le funzioni per trovare i file in errore o gli errori SQL presenti sul cruscotto.

4.2 L'applicazione WEB

Per quanto riguarda l'aspetto grafico, mi sono basato sulle linee guida visuali d'azienda per il menù, i font e le tabelle disegnate.

La web application è stata così pensata:

- Model: generato con il framework di Microsoft .NET Entity Framework di cui parlerò brevemente in seguito.
- View: È stato utilizzato ASP.NET Razor che [*è un linguaggio a Markup, ideato da Microsoft, per integrare codice lato server in una pagina web. La sintassi di razor consiste principalmente in markups, C# e HTML. I file che solitamente contengono razor hanno una estensione*

.cshtml]. (Anderson, Mullen, & Vicarel, s.d.)

- Controller: Sviluppati in C# come da vincolo tecnico fornito dalla azienda.

4.3 Entity Framework: Breve introduzione

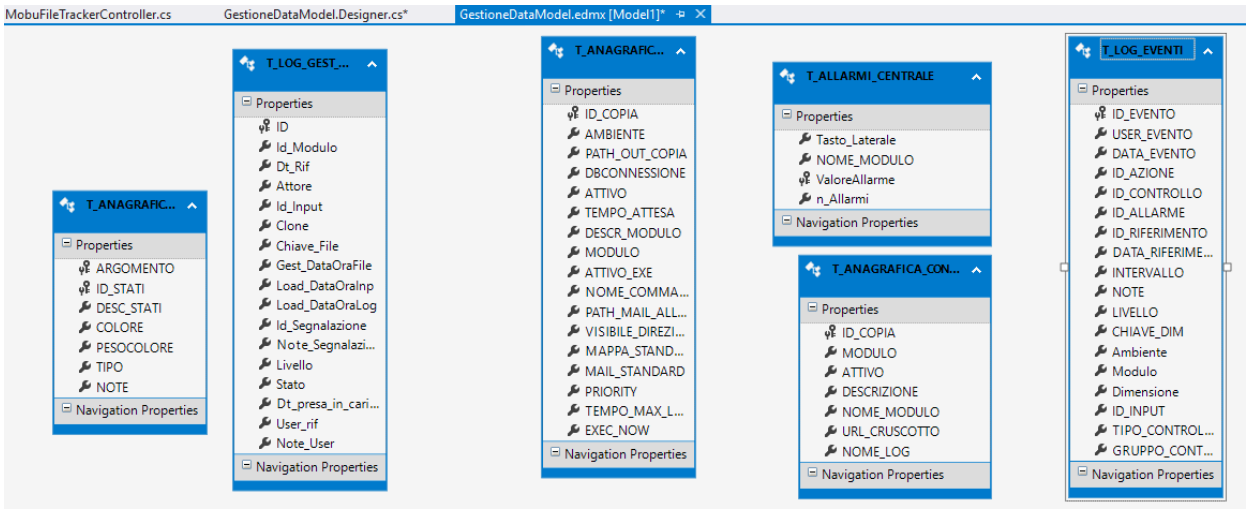
[Entity Framework (EF) è un mapper relazionale a oggetti che consente agli sviluppatori .NET di utilizzare i dati relazionali utilizzando oggetti specifici del dominio. In questo modo la maggior parte del codice di accesso ai dati che in genere gli sviluppatori devono scrivere non è più necessaria]. (Microsoft, 2010)

Questo framework è stato utilizzato per generare, a partire dagli oggetti del database, delle classi .NET che modellano questi oggetti e che posseggono una serie di metodi per poter effettuare le operazioni di Insert, Delete, Update, il che rende estremamente veloce lo sviluppo di una applicazione con un backend già esistente e consolidato. Il framework si integra in un progetto .NET MVC utilizzando lo strumento di gestione dei pacchetti di Microsoft, NUGET, poi si può utilizzare da riga di comando dedicata o sfruttando il wizard visuale che permette la sua configurazione. I parametri sono molto semplici, serve la stringa di connessione al database, per permettergli di accedere alle strutture dati, si possono selezionare le tabelle, le funzioni e le stored procedure che si intendono mappare e poi si dà il via allo strumento. Ciò che viene generato è un contesto DB che viene creato come una classe Singleton e utilizzato per le varie operazioni da effettuare con il database, una serie di classi C# che modellano le tabelle selezionate nel wizard.

```
        }

        dbRow.Dt_presa_in_carico = DateTime.Now;

        /*Inserisco nella T_LOG_EVENTI*/
        T_LOG_EVENTI evento = new T_LOG_EVENTI();
        evento.ID_INPUT = dbRow.Id_Input;
        evento.USER_EVENTO = User_rif;
        evento.Ambiente = "MOB10";
        evento.DATA_EVENTO = DateTime.Now;
        evento.NOTE = "Evento su MOBU GM: " + btn_gestione;
        evento.ID_RIFERIMENTO = ID;
        evento.Modulo = "MOB10_GM";

        gestione.T_LOG_EVENTI.AddObject(evento);
        gestione.SaveChanges();

    }
    return RedirectToAction("Index", new
    {
        page = _page,
        dateRif = _dateRif,
        filterString = _filterString,
```

2 Utilizzo Della Classe T_LOG_EVENTI

Lo strumento inoltre genera un modello grafico delle strutture scelte, con cui che si può facilmente interagire dalla finestra di presentazione, si può trascinare con un semplice drag/drop, si possono pure aggiungere campi e definizioni e permette, attraverso il meccanismo delle *Migrations* di portare le modifiche effettuate direttamente nel Database.



3 Modello ER Generato da Entity Framework

Questo framework, mi ha quindi permesso di abbattere i tempi di sviluppo del data layer e di tutta quella parte di strutture dati che servono a sostegno della applicazione web e che di solito prendono una parte non indifferente di tempo nello sviluppo.

4.4 Back End

Dovendo rendere questa soluzione facilmente integrabile sugli altri moduli e configurabile da una anagrafica, la scelta che ho intrapreso è stata quella di trattarla come una nuova funzione del cruscotto oggetto di analisi, che però fosse possibile abilitare o disabilitare andando a gestire una anagrafica dal database centrale di MOBU (dbo.[MOB10_GESTIONE]) T_ANAGRAFICA_CONSOLE_CENTRALE. Per ottenere questo risultato, ho aggiunto al database del modulo dbo.[MOB10_PSH_SCT_TRAM] una tabella denominata T_LOG_CARICAMENTO_FILE ed una stored procedures denominata dbo.carsp_Popola_Log_Caricamento_File ed ho modificato la stored procedure dbo.carSP_BulkOpenRowSet, questi tre elementi, per ogni cruscotto in cui si vuole integrare tale funzionalità devono essere presenti mantenendo questa nomenclatura, in conformità con la struttura e le caratteristiche di MOBU, inoltre non essendo integrati nel processo di monitoraggio di business, hanno un basso rischio di produrre una regressione delle funzionalità.
Per avere un punto centrale in cui raccogliere i dati sui file scartati, gli errori SQL ed altre eventuali segnalazioni ho implementato una tabella sul database centrale MOB10_GESTIONE di nome T_LOG_GEST_ALLARMI_LOAD, per popolarla correttamente mi sono servito di due stored procedures che ho chiamato dbo.carsp_TrovaFileInErrore e dbo.carSp_TrovaFileInRitardo.

5. La soluzione Sviluppata

Andrò adesso ad analizzare in dettaglio come ho sviluppato i vari elementi, quali scelte ho preso e come ho integrato tra loro le varie funzionalità.

5.2 Integrazione su MO10_PSH_TRAM

La prima domanda che mi sono posto è stata: come faccio a identificare un file in modo univoco partendo dal nome? I file di alimentazione di uno stesso input hanno la stessa radice e lo stesso nome ma contengono, chiaramente, dati diversi. Il middleware di File Transfer, in coda ad ogni file, pone un codice esadecimale univoco e di dimensione fissa e quindi può essere usato come campo per identificare univocamente il file. Eccone un esempio:

MOBU_PSH_CVF_IP_20200606_051320800377_TO-SIRE_SCTO_BCITGB2L_ISOBK_PSR_01_20200606051047_MIP-MOBCV.*0000000000A0E077*

Con la giusta funzione, posso estrapolare una chiave univoca, semplice, per poter identificare un file. Partendo da questo assunto, ho quindi creato la tabella T_LOG_CARICAMENTO_FILE con questa struttura:

```
CREATE TABLE [dbo].[T_LOG_CARICAMENTO_FILE](
    [Attore] [nchar](10) NULL,
    [DataRif] [varchar](50) NULL,
    [Id_input] [varchar](50) NULL,
    [Clone] [varchar](11) NULL,
    [Chiave_file] [varchar](50) NOT NULL,
    [QDH_stato] [varchar](10) NULL,
    [Nome_file] [varchar](200) NOT NULL,
    [Dimensione_file] [bigint] NULL,
    [Num_righe] [int] NULL,
```

```
    [StatoLogCar] [varchar](50) NULL,
    [Num_rec_load] [int] NOT NULL,
    [Kpi_CAR] [nchar](10) NULL,
    [DataOraFile] [datetime] NOT NULL,
    [DataOraLoadInp] [datetime] NULL,
    [DataOraLogCar] [datetime] NULL,
 CONSTRAINT [PK_T_LOG_CARICAMENTO_FILE] PRIMARY KEY CLUSTERED
(
    [Chiave_file] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, AL-
LOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[T_LOG_CARICAMENTO_FILE] ADD  CONSTRAINT [DF_T_LOG_CARICA-
MENTO_FILE_Num_rec_load]  DEFAULT ((0)) FOR [Num_rec_load]
GO
```

La tabella ha come primary key il campo Chiave_file che sarà quello dedicato al codice univoco di trasferimento, altre dimensioni di interesse sono *StatoLogCar* che descrive lo stato del caricamento, se fallito o andato a buon fine. In questa tabella, modello il caricamento di un file per il modulo di Tramitate, dando una visione dello stato di un file, di quale attore lo abbia inviato, quando lo ha inviato, senza ovviamente tralasciare informazioni quali il nome, le righe che conteneva il file e l'ID_INPUT a cui appartiene.

5.3 L'analisi dei file

Il tema principale su cui mi sono soffermato, è stato quello di definire delle regole che mi garantissero di capire se un file fosse stato o meno scartato per errore o meno. Per questa analisi, mi sono basato sulle azioni che venivano fatte dalle due risorse esterne prima di risottomettere un file ovvero, controllo sulla cartella FILE_ERRATI di un modulo della presenza di qualche file e verifica sulle relative tabelle di input della presenza o meno di questo. In quel caso, il file non veniva risottomesso, altrimenti si.

Per poter effettuare queste analisi in modo automatico avevo bisogno di alcuni elementi:

- La lista di tutti i file ricevuti, indipendentemente se fossero scartati o meno.
- Individuare quali tra questi file ricevuti, erano stati scartati

Incrociando queste informazioni con quelle relative alle tabelle di input, avrei potuto identificare in modo sicuro se un file fosse o meno stato scartato e inserirlo con questo stato nella tabella T_LOG_CARICAMENTO_FILE. Per il primo punto, potevo fare affidamento ad una tabella del database centrale di MOBU, che contiene tutti i file presenti nelle cartelle di storico, tali cartelle vengono riempite dal gestore file nel momento in cui un file sta per essere spostato nella cartella del modulo. Poi un file batch, una volta ogni ora cerca in queste cartelle e inserisce i nomi dei file nella tabella T_LOG_FILE_CARTELLE_STO.

Per il secondo punto, invece ho trovato le informazioni che mi servivano nella tabella T_LOG_CARICAMENTO , riempita dal caricatore, dove scrive

i passaggi di caricamento di un file ed elaborazione del dato. A questo punto, ho creato la stored procedures carSP_Popola_Log_Caricamento_file.

```
CREATE PROCEDURE [dbo].[carSp_Popola_Log_Caricamento_File]
AS
BEGIN
SET NOCOUNT ON;
/*Svecchiamo i record della tabella*/
delete from t_log_caricamento_file
where convert(varchar,DataOraFile,112) <= GETDATE()-5



create table #tmp_carteelle_file
(
	Nome_file_arrivato varchar(200),
	Chiave_file varchar(20) NOT NULL PRIMARY KEY,
	Attore varchar(20),
	Data_ora_inserimento datetime2,
	[filename] varchar(200),
	DataOraFile datetime2,
	Byte bigint,
	Id_input varchar(30)

);
/*
Prendiamo dalla tabella dei file nello storico (raggruppo per il campo chiave così
da avere un solo record per ognuna di esse),i dati per aggiornare la tabella
T_LOG_CARICAMENTO_FILE.
A seconda dell'input riusciamo a discriminare il campo attore
*/

insert into #tmp_cartelle_file
select
max(SUBSTRING(filename,CHARINDEX('MOBU_',filename),200)) as nome_file_arrivato,
case when charindex('fuoriorario',filename)>0
  then reverse(substring(reverse(fileName),charindex(REVERSE('_Fuoriorario'),
reverse(fileName))+12,16))
  else reverse(substring(reverse(fileName),
charindex('KO_',reverse(fileName))+3,16)) end as chiave_file,
max(case
  when id_input  in ('MOBU_PSH_CVF_IP','MOBU_PSH_ICF_IP') then  'MIP'
  else  'BROD0' end) as attore,
```

```
max(cast(Data_Ora_inserimento as datetime2)) as data_ora_inserimento,
max([FileName]),
max(cast(DataOraFile as datetime2)) as DataOraFile,
max(byte) as byte,
max(id_input) as id_input
from MOB10_GESTIONE..T_LOG_FILE_CARTELLE_STO
WHERE id_in-
put IN ('MOBU_PSH_CVF_IP','MOBU_PSH_ICF_IP','MOBU_PSH_IP_SCT_DETT') AND con-
vert(varchar,dataoraFile,112) >= GETDATE()-4
group by case when charindex('fuoriorario',filename)>0
  then reverse(substring(reverse(fileName),charindex(REVERSE('_Fuoriorario'),
reverse(fileName))+12,16))
  else reverse(substring(reverse(fileName),
charindex('KO_',reverse(fileName))+3,16)) end

MERGE t_log_caricamento_file AS T
USING (
select * from
(
    select *
    from #tmp_cartelle_file

) AS S
ON (T.chiave_file = S.chiave_file)
WHEN MATCHED THEN
    UPDATE SET
    t.attore = s.attore,
    t.dimensione_file = s.byte,
    t.clone='MULTI',
    t.datarif=convert(varchar,s.dataorafile,112),
    t.dataorafile= s.dataorafile,
    t.id_input = s.id_input
WHEN NOT MATCHED THEN
    INSERT (Chiave_file,Nome_file, DataOraFile, attore, dimensione_file, clone,
      datarif, id_input, StatoLogcar)
    VALUES(
    S.chiave_file, s.nome_file_arrivato,s.DataoraFile, s.attore, s.byte, 'MULTI',
      convert(varchar,s.dataorafile,112), s.id_input,
    CASE WHEN charindex('fuoriorario',filename)>0 THEN 'FuoriOrario' ELSE null END
    );

drop table #tmp_cartelle_file


--aggiornamento dei record degli ICF
valorizziamo il numero di record nella tabella di input per ogni file che ab-
biamo caricato
```

```
MERGE T_LOG_CARICAMENTO_FILE AS T
USING
(
SELECT NOME_FILE, COUNT(*) AS NUM_RECORD, MAX(DATA_FILE) DATA_FILE,
BIC_RECEIVER FROM
    (SELECT *
    FROM T_INPUT_ICF_IP
    UNION ALL
    SELECT * FROM T_INPUT_ICF_IP_STO
    WHERE convert(varchar,DATA_FILE,112) >= GETDATE()-4
    ) AS A
    GROUP BY NOME_FILE, BIC_RECEIVER
) AS SRC
ON (T.NOME_FILE = SRC.NOME_FILE)
WHEN MATCHED THEN
    UPDATE SET
    T.NUM_REC_LOAD = SRC.NUM_RECORD,
    T.DataOraLoadInp = SRC.DATA_FILE,
    T.Clone = CASE WHEN SRC.BIC_RECEIVER = '' THEN 'N/A' ELSE SRC.BIC_RECEIVER END;

--AGGIORNO I VALORI PER I CVF
--valorizziamo il numero di record nella tabella di input per ogni file che ab-
biamo caricato
MERGE T_LOG_CARICAMENTO_FILE AS T
USING
(
    SELECT A.NOME_FILE, COUNT(*) AS NUM_REC-
ORD, MAX(DT_LOAD_MOBU) DT_LOAD_MOBU, BIC_RECEIVER FROM
    (SELECT *
    FROM T_INPUT_CVF_IP
    UNION ALL
    SELECT * FROM T_INPUT_CVF_IP_STO
      WHERE convert(varchar,DATA_FILE,112) >= GETDATE()-4
    ) AS A
    GROUP BY A.NOME_FILE, BIC_RECEIVER
) AS SRC
ON ( T.NOME_FILE = SRC.NOME_FILE )
WHEN MATCHED THEN
    UPDATE SET
    T.NUM_REC_LOAD = SRC.NUM_RECORD,
    T.DataOraLoadInp = SRC.DT_LOAD_MOBU,
    T.Clone = CASE WHEN SRC.BIC_RECEIVER = '' THEN 'N/A' ELSE SRC.BIC_RECEIVER END;

--AGGIORNO I VALORI PER FILE DI SCT_DETT
--valorizziamo il numero di record nella tabella di input per ogni file che ab-
biamo caricato

MERGE T_LOG_CARICAMENTO_FILE AS T
```

```
USING
(
    SELECT A.NOME_FILE, COUNT(*) AS NUM_RECORD, MAX(DT_LOAD_MOBU) DT_LOAD_MOBU FROM
    (SELECT *
    FROM T_INPUT_IP_SCT_DETT
    UNION ALL
    SELECT * FROM T_INPUT_IP_SCT_DETT_STO
      WHERE convert(varchar,DATA_FILE,112) >= GETDATE()-4
    ) AS A
    GROUP BY A.NOME_FILE
) AS SRC
ON ( T.NOME_FILE = SRC.NOME_FILE )
WHEN MATCHED THEN
    UPDATE SET
    T.NUM_REC_LOAD = SRC.NUM_RECORD,
    T.DataOraLoadInp = SRC.DT_LOAD_MOBU;



/*
Se sulla tabella del QDH c'è un record in KO e sulla tabell T_LOG_CARICAMENTO_FILE
non abbiamo informazioni riguardante quel file, lo inseriamo in stato KO con una
chiave provvisoria
*/

MERGE T_LOG_CARICAMENTO_FILE AS T
USING
(
    SELECT stato, data_ora_elab, descr_breve_msg
    FROM [MOB10_PSH_SDD_DETT].[dbo].[T_INPUT_QDH_TRATTE_YQ_MSG]
    WHERE STATO = 'KO' AND (DESCR_BREVE_MSG LIKE '%ICF_IP%' OR DE-
SCR_BREVE_MSG LIKE '%CVF_IP%')
) AS SRC
ON (substring(T.NOME_FILE, 1, 50) = substring(SRC.descr_breve_msg, 1, 50))
WHEN NOT MATCHED THEN
INSERT (chiave_file, Nome_file, DataOraFile, qdh_stato)
VALUES (substring(SRC.descr_breve_msg, 1, 50), SRC.descr_breve_msg, data_ora_elab,
stato);
```

```
/*
Aggiorniamo i campi dal log del caricamento dei file. Passando per una tempora-
nea per poter ordinare per ID asc.
La join su sé stessa serve a selezionare solo l'ultima elabora-
zione di un file, nel caso sia stato
risottomesso o rielaborato,
in modo da non avere match multipli sulla merge.
*/
--
SELECT id, reverse(substring(reverse(messaggio),1,16)) as Chiave_file, step,
data_insert
into #log_tmp
FROM [MOB10_PSH_TRAM].[dbo].[T_LOG_CARICAMENTO] t1
JOIN (
      SELECT max(id) as idx
      FROM [MOB10_PSH_TRAM].[dbo].[T_LOG_CARICAMENTO]
      Where left(messaggio,27)
       in('Cancellato il File Caricato','Spostamento del File Errato')
      group by reverse(substring(reverse(messaggio),1,16))
      ) as t2
  ON (t1.id = t2.idx)
where left(messaggio,27)
in('Cancellato il File Caricato','Spostamento del File Errato')
order by t1.id asc


MERGE T_LOG_CARICAMENTO_FILE AS T
USING
(
 select * from #log_tmp
) AS SRC
ON (T.chiave_file = src.Chiave_file)
WHEN MATCHED THEN
    UPDATE SET
    t.statoLogCar = src.step,
    t.DataOraLogCar = src.data_insert,
    t.QDH_stato = CASE WHEN src.step = 'cancellaFile' THEN 'OK'
    END;

drop table #log_tmp
/*
Controlliamo se i record precedentemente in KO sono stati elabo-
rati dal qdh ed in caso positivo eliminiamo il record relativo al KO
mantenendo solo quello in OK
*/

SELECT
CASE WHEN substring(NOME_FILE, 1, 3) IN ('OK_','VA_')
```

```
THEN substring(NOME_FILE, 4, 50)
ELSE substring(NOME_FILE, 1, 50) end as nome_file_troncato
INTO #RECORD_TO_DELETE
FROM T_LOG_CARICAMENTO_FILE

DELETE FROM T_LOG_CARICAMENTO_FILE
WHERE NOME_FILE IN (
        SELECT nome_file_troncato
        FROM #RECORD_TO_DELETE
        GROUP BY nome_file_troncato
        HAVING COUNT(nome_file_troncato) >1
    )
AND QDH_STATO = 'KO'

DROP TABLE #RECORD_TO_DELETE

/* Calcolo del KPI escludendo i file vuoti con dimensione = 0 */
UPDATE T_LOG_CARICAMENTO_FILE SET Kpi_CAR = DIMEN-
SIONE_FILE/case when Num_rec_load = 0 then DIMENSIONE_FILE else Num_rec_load end
 WHERE DIMENSIONE_FILE > 0

--fine

END
```

In questa stored procedure ho inserito tutti i passaggi utili a tracciare lo stato di un file caricato del cruscotto, sia in positivo che in negativo.

La prima operazione che faccio è la insert in una tabella temporanea, in cui ho definito la primary key per far creare un indice in modo implicito, prendendo i dati dall'archivio di tutti i file caricati in giornata, ovviamente filtrando i dati per gli ID_INPUT appartenenti al modulo del cruscotto. In questa tabella temporanea, inserisco già il campo *Chiave_file* che mi sarà fondamentale per aumentare le prestazioni durante le operazioni di MERGE. La funzione per estrarre la chiave è la seguente:

```
reverse(substring(reverse(fileName),charindex('KO_',reverse(file-
Name))+3,16)) end as chiave_file
```

Si basa su alcuni assunti:

- Il gestore file, quando sposta un file nella relativa cartella di storico, inserisce sempre i caratteri "_OKXY" dove XY sono due numeri da 1 a 9.
- Il codice univoco inserito è composto sempre da 16 caratteri

Il nome del file quindi avrà questa forma:

MOBU_PSH_CVF_IP_20200606_051320800377_TO-SIRE_SCTO_BCITGB2L_ISOBK_PSR_01_20200606051047_MIP-MOBCV.*0000000000A0E077*_*OK39*

Ho usato la funzione SQL SUBSTRING così definita

SUBSTRING (expression, start, length)

Expression: È un'espressione di tipo **character**, **binary**, **text**, **ntext** o **image**.

Start: Valore intero o espressione **bigint** che specifica l'inizio dei caratteri restituiti. (La numerazione è in base 1, ovvero il primo carattere dell'espressione è 1). Se *start* è minore di 1, l'espressione restituita inizierà con il primo carattere specificato in *expression*. In questo caso, il numero di caratteri restituito è il valore maggiore tra la somma di *start* + *length*- 1 oppure 0. Se *start* è maggiore del numero di caratteri nell'espressione valore, viene restituita un'espressione di lunghezza zero.

Length: Valore integer positivo o espressione bigint che specifica quanti caratteri di *expression* verranno restituiti. Se *length* è negativo, viene generato un errore e l'istruzione viene terminata. Se la somma di *start* e *length* è

maggiore del numero di caratteri di *expression*, viene restituita l'intera espressione del valore che inizia con *start*.

- ***expression*** ho passato il nome file, ma al contrario, ottenuto con la funzione REVERSE(expression).
- ***start***, sono andato ad utilizzare la funzione CHARINDEX che esegue la ricerca dell'espressione di un carattere all'interno di una seconda espressione di caratteri, e restituisce la posizione iniziale della prima espressione, se trovata. Così definita:

  CHARINDEX (expressionToFind , expressionToSearch [, start_location])

  - ***expressionToFind***: 'KO_' che, come da assunto trovo sempre nei file di storico alla fine del nome file
  - ***expressionToSearch***: reverse di fileName
- ***Lenght:*** Essendo il codice univo lungo 16, ho passato tale valore.

In questo modo, il valore estrapolato dal nome file corrispondeva alla chiave che mi ero prefissato di cercare. La funzione REVERSE più esterna, serve a ottenere il valore nel verso corretto: 0000000000A0E077

Nella tabella temporanea ho anche portato altre dimensioni appartenenti alla tabella di storico, per popolare le informazioni utili della tabella T_LOG_CARICAMENTO_FILE.

La seconda operazione è il comando MERGE tra la tabella temporanea e la T_LOG_CARICAMENTO_FILE, effettuata con il campo Chiave_file come condizione di MERGE. L'istruzione MERGE Esegue operazioni di inserimento, aggiornamento o eliminazione in una tabella di destinazione dai risultati di una join con una tabella di origine. Sincronizzare, ad esempio, due tabelle

inserendo, aggiornando o eliminando righe in una tabella in base alle differenze trovate nell'altra tabella. La tabella di origine per me è la #tmp_cartelle_file e la tabella di destinazione è la t_log_caricameto_file. Ovviamente, al primo impianto della tabella, questa operazione effettuerà solo delle insert, in quanto la tabella destinazione è vuota, ma dalla seconda esecuzione in poi, inserirà solo il delta dei nuovi file caricati e aggiornerà le informazioni di quelli esistenti.

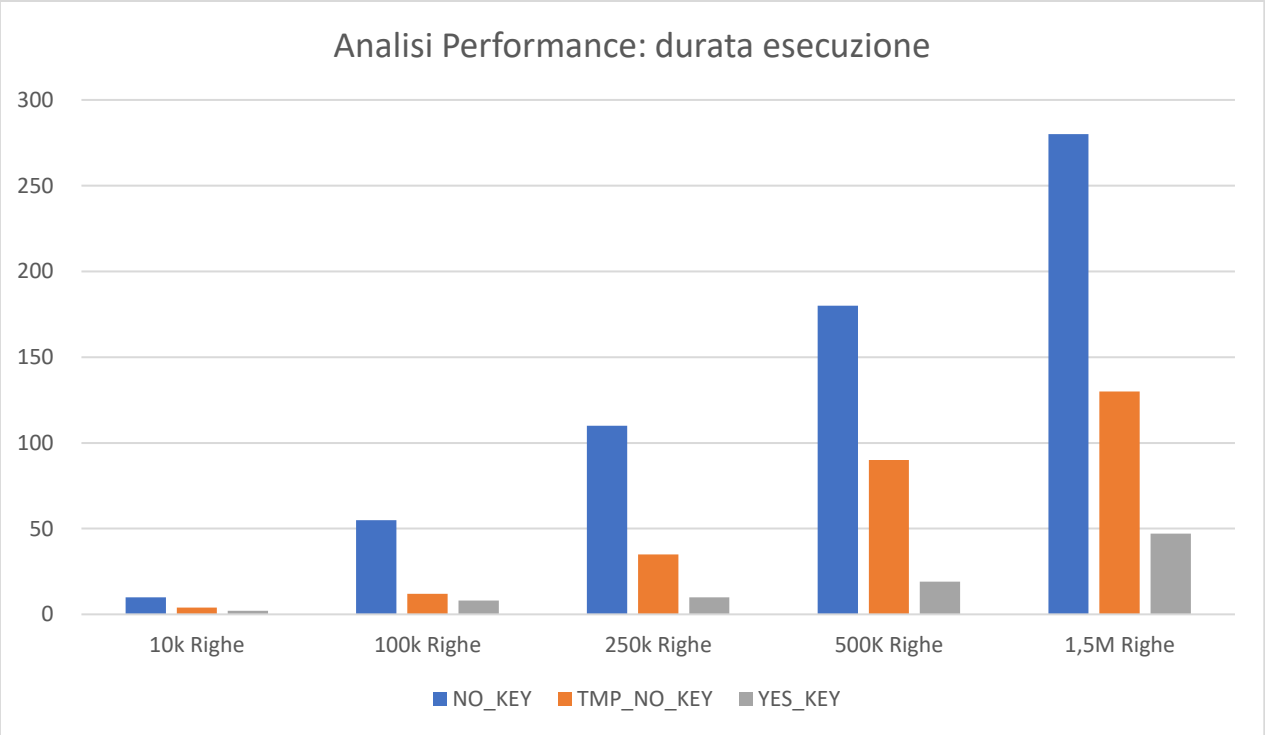
La scelta della definizione della chiave primaria sul campo Chiave_File, della T_LOG_CARICAMENTO_FILE, è stata anche avallata dalle prestazioni ottenute per queste operazioni, infatti, come suggerito dalla documentazione ufficiale Microsoft: [Per migliorare le prestazioni delle istruzioni MERGE, si consiglia di attenersi alle linee guida relative agli indici seguenti:

- Creare un indice univoco e di copertura sulle colonne di join della tabella di origine.
- Creare un indice cluster univoco nelle colonne di join della tabella di destinazione.

Tali indici garantiscono l'univocità delle chiavi di join e l'ordinamento dei dati contenuti nelle tabelle. Le prestazioni delle query risultano migliorate poiché Query Optimizer non deve eseguire alcuna ulteriore elaborazione della convalida per individuare e aggiornare righe duplicate né è necessario eseguire operazioni di ordinamento aggiuntive] (Microsoft, s.d.)

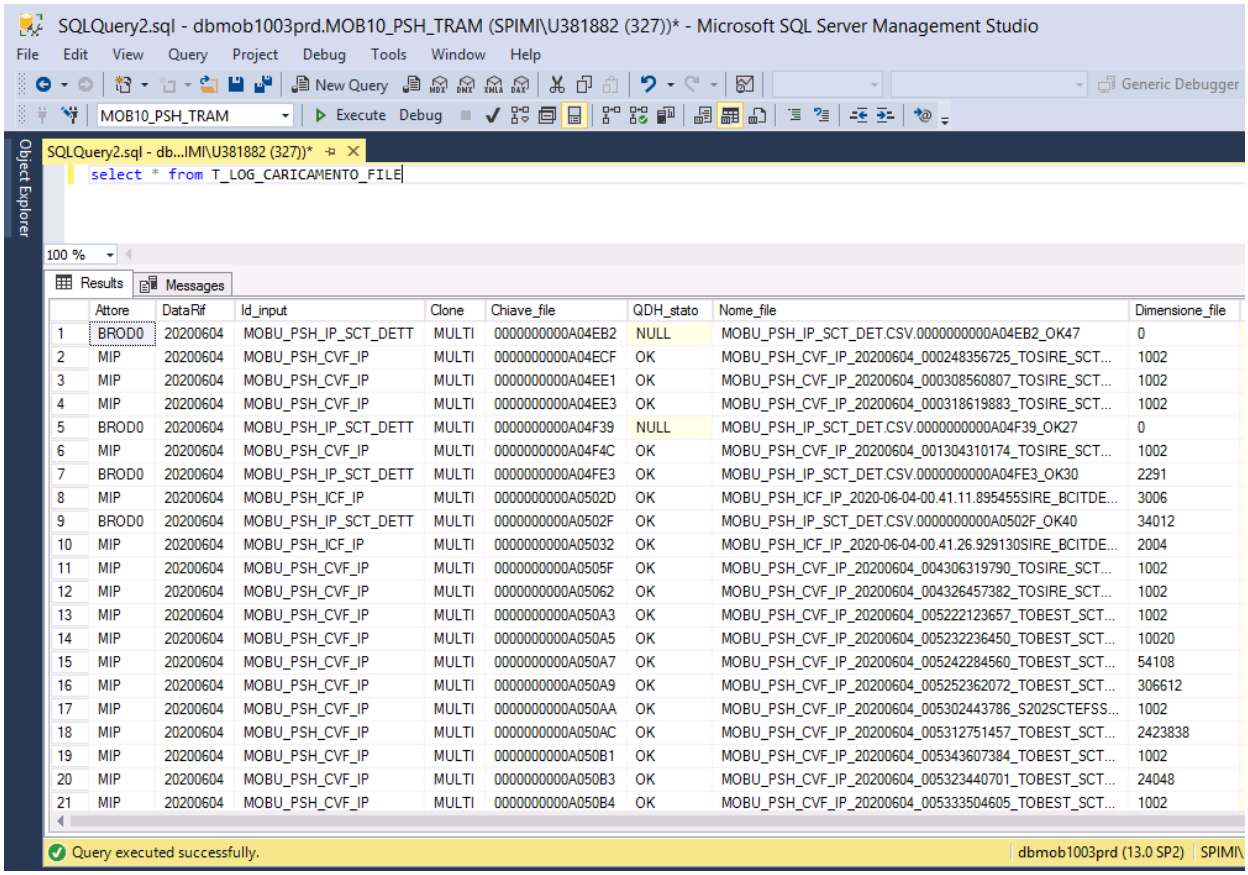
Che è esattamente quello che ho ottenuto con la creazione delle primary key sul campo chiave file. Ho inoltre analizzato le performance senza adottare queste scelte tecniche, tali test sono stati eseguiti con un subset dei dati di produzione per via della disponibilità dell'ambiente di system, ed

una volta rilasciata la stored in ambiente di produzione, ne ho verificato le performance in maniera definitiva.



La seconda parte dello script, va a verificare se un determinato nome_file, sia presente o meno nella tabella del relativo input, in caso positivo, raggruppando e contando le righe, ottengo dei dati relativi a quel file da poter inserire nella tabella T_LOG_CARICAMENTO_FILE, inoltre, la presenza sulla tabella di input, ci garantisce che il file è stato caricato.

La terza operazione importante è data invece dalla ricerca della chiave_file, sulla tabella di LOG del caricatore. Qui, possiamo vedere se quel file era stato scartato (stato = ScartatoFileErrato) o correttamente elaborato (stato = CancellatoFileCaricato) così, l'update della tabella T_LOG_CARICAMENTO_FILE è concluso e contiene la situazione complessiva dei caricamenti.



	Attore	DataRif	Id_input	Clone	Chiave_file	QDH_stato	Nome_file	Dimensione_file
1	BROD0	20200604	MOBU_PSH_IP_SCT_DETT	MULTI	0000000000A04EB2	NULL	MOBU_PSH_IP_SCT_DET.CSV.0000000000A04EB2_OK47	0
2	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A04ECF	OK	MOBU_PSH_CVF_IP_20200604_000248356725_TOSIRE_SCT...	1002
3	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A04EE1	OK	MOBU_PSH_CVF_IP_20200604_000308560807_TOSIRE_SCT...	1002
4	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A04EE3	OK	MOBU_PSH_CVF_IP_20200604_000318619883_TOSIRE_SCT...	1002
5	BROD0	20200604	MOBU_PSH_IP_SCT_DETT	MULTI	0000000000A04F39	NULL	MOBU_PSH_IP_SCT_DET.CSV.0000000000A04F39_OK27	0
6	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A04F4C	OK	MOBU_PSH_CVF_IP_20200604_001304310174_TOSIRE_SCT...	1002
7	BROD0	20200604	MOBU_PSH_IP_SCT_DETT	MULTI	0000000000A04FE3	OK	MOBU_PSH_IP_SCT_DET.CSV.0000000000A04FE3_OK30	2291
8	MIP	20200604	MOBU_PSH_ICF_IP	MULTI	0000000000A0502D	OK	MOBU_PSH_ICF_IP_2020-06-04-00.41.11.895455SIRE_BCITDE...	3006
9	BROD0	20200604	MOBU_PSH_IP_SCT_DETT	MULTI	0000000000A0502F	OK	MOBU_PSH_IP_SCT_DET.CSV.0000000000A0502F_OK40	34012
10	MIP	20200604	MOBU_PSH_ICF_IP	MULTI	0000000000A05032	OK	MOBU_PSH_ICF_IP_2020-06-04-00.41.26.929130SIRE_BCITDE...	2004
11	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A0505F	OK	MOBU_PSH_CVF_IP_20200604_004306319790_TOSIRE_SCT...	1002
12	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A05062	OK	MOBU_PSH_CVF_IP_20200604_004326457382_TOSIRE_SCT...	1002
13	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050A3	OK	MOBU_PSH_CVF_IP_20200604_005222123657_TOBEST_SCT...	1002
14	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050A5	OK	MOBU_PSH_CVF_IP_20200604_005232236450_TOBEST_SCT...	10020
15	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050A7	OK	MOBU_PSH_CVF_IP_20200604_005242284560_TOBEST_SCT...	54108
16	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050A9	OK	MOBU_PSH_CVF_IP_20200604_005252362072_TOBEST_SCT...	306612
17	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050AA	OK	MOBU_PSH_CVF_IP_20200604_005302443786_S202SCTEFSS...	1002
18	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050AC	OK	MOBU_PSH_CVF_IP_20200604_005312751457_TOBEST_SCT...	2423838
19	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050B1	OK	MOBU_PSH_CVF_IP_20200604_005343607384_TOBEST_SCT...	1002
20	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050B3	OK	MOBU_PSH_CVF_IP_20200604_005323440701_TOBEST_SCT...	24048
21	MIP	20200604	MOBU_PSH_CVF_IP	MULTI	0000000000A050B4	OK	MOBU_PSH_CVF_IP_20200604_005333504605_TOBEST_SCT...	1002

Figura 4 - Particolare T_LOG_CARICAMENTO_FILE

Per gestire il caso in cui viene risottomesso un file per l'elaborazione e non infrangere il vincolo di integrità referenziale formato dalla chiave_file, ho effettuato una modifica nella procedura che carica il dato dal file alla prima tabella di staging. Quando lavoro il file, calcolo dinamicamente la chiave, e vado a fare una DELETE nella tabella t_log_caricamento_file per eliminare l'occorrenza, se presente, della chiave. Seguito poi da una insert, per tracciare poi il caricamento del file risottomesso, quindi con tutti i passaggi necessari. Questo ha un impatto minimo sulle performance perché

la stored procedures elabora un file per volta e trattandosi di una delete/insert su un campo chiave non inficia sulle performance e mi mantiene una situazione allineata e pulita della tabella.

5.4 La ricerca delle segnalazioni

Con la tabella T_LOG_CARICAMENTO_FILE correttamente popolata, posso ricavare quei record che indicano un file scartato. Questi record, servono poi per popolare una tabella a cui farà riferimento la web application per poter visualizzare il dato a front-end, ed è in questa tabella che teniamo traccia di un file scartato e delle sue modifiche di stato. Questa tabella si chiama T_LOG_GEST_ALLARMI_LOAD ed ha la seguente struttura

```
CREATE TABLE [dbo].[T_LOG_GEST_ALLARMI_LOAD](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [Id_Modulo] [varchar](50) NULL,
    [Dt_Rif] [varchar](50) NULL,
    [Attore] [varchar](50) NULL,
    [Id_Input] [varchar](200) NULL,
    [Clone] [varchar](50) NULL,
    [Chiave_File] [varchar](50) NULL,
    [Gest_DataOraFile] [datetime] NULL,
    [Load_DataOraInp] [datetime] NULL,
    [Load_DataOraLog] [datetime] NULL,
    [Id_Segnalazione] [int] NOT NULL,
    [Note_Segnalazione] [text] NULL,
    [Livello] [varchar](50) NULL,
    [Stato] [varchar](50) NULL,
    [Dt_presa_in_carico] [datetime] NULL,
    [User_rif] [varchar](50) NULL,
    [Note_User] [text] NULL,
 CONSTRAINT [PK_T_LOG_GEST_ALLARMI_LOAD] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

La tabella presenta dei campi per la gestione della segnalazione da parte dell'utente, come , STATO,DT_PRESA_IN_CARICO, NOTE_USER, le altre invece servono per modellare l'occorrenza di un file in allarme, con data di riferimento, nome file, chiave, attore e le altre dimensioni utili allo scopo. Per Popolare ed aggiornare correttamente questa tabella, ho creato, nel database di gestione una stored procedures che si occupava di questo scopo che andrò qui a descrivere:

```
CREATE PROCEDURE [dbo].[spTrovaFileInErrore]
AS
BEGIN
SET NOCOUNT ON;
/*
Svecchiamo i record della tabella
*/
set language us_english

delete from [MOB10_GESTIONE].[dbo].[T_LOG_GEST_ALLARMI_LOAD]
where convert(varchar,Gest_DataOraFile,112) <= GETDATE()-20

declare @dbconnessione as varchar(50)
declare @descrizione as varchar(50)
declare @sql_query as varchar(2000)

DECLARE cursore_moduli CURSOR FOR
SELECT DBCONNESSIONE, A.MODULO
from [MOB10_GESTIONE].[dbo].[T_ANAGRAFICA_CONSOLE_CENTRALE] A
JOIN [MOB10_GESTIONE].[dbo].[T_ANAGRAFICA_MODULI] B
ON A.MODULO = B.MODULO AND A.ATTIVO = 1 AND B.ATTIVO = 1

OPEN cursore_moduli;
FETCH NEXT FROM cursore_moduli INTO @dbconnessione,@descrizione;
WHILE @@FETCH_STATUS = 0
   BEGIN

/*
Lancia le store che popolano la T_LOG_CARICAMENTO_FILE dei vari cruscotti attivi
*/

SET @sql_query =
'exec ['+@dbconnessione+'].[dbo].[carSp_Popola_Log_Caricamento_File]'
EXEC(@sql_query)
```

```
/*
Identifica i file il cui caricamento non è andato a buon fine e li inseri-
sce sulla tabella in Gestione
*/

SET @sql_query ='
MERGE [MOB10_GESTIONE].[dbo].[T_LOG_GEST_ALLARMI_LOAD] AS S
USING(
SELECT DataRif, Attore, Id_Input, Clone, Chiave_file, DataOraFile,
DataOraLoadInp, DataOraLogCar, StatoLogCar
FROM ['+@dbconnessione+'].[dbo].[T_LOG_CARICAMENTO_FILE]
WHERE (StatoLogCar = ''spostaFileErrati'' OR StatoLogCar = ''FuoriOrario''
OR (StatoLogCar is NULL AND datediff(hour,DataOraFile, getdate()) > 2))
AND Dimensione_file > 0
AND Num_Rec_Load = 0
) AS T
ON (S.Chiave_file = T.Chiave_file)
WHEN NOT MATCHED THEN
INSERT (Id_Modulo,Dt_Rif,Attore,Id_Input,Clone,Chiave_File,
Gest_DataOraFile,Load_DataOraInp,Load_DataOraLog,Id_Segnalazione,
Note_Segnalazione,Livello,Stato,Dt_presa_in_carico,User_rif,Note_User)
VALUES ('''+@descrizione+''',
DataRif,Attore, Id_Input, Clone, Chiave_file, DataOraFile, DataOraLoadInp,
DataOraLogCar,
case when StatoLogCar = ''FuoriOrario'' THEN 9900 ELSE 9999 END,
case when StatoLogCar = ''FuoriOrario'' THEN ''File ricevuto fuori ora-
rio e non elaborato'' ELSE ''File non elaborato'' END,
''ALERT'', ''INSERITO'', NULL, NULL, NULL);'
EXEC(@sql_query)
/*
Verifichiamo se alcune delle segnalazioni precedentemente inserite sono state
risolte da un caricamento successivo e, in caso affermativo, vengono chiuse au-
tomaticamente
*/
SET @sql_query ='MERGE [MOB10_GESTIONE].[dbo].[T_LOG_GEST_ALLARMI_LOAD] AS S
USING(
SELECT DataRif, Attore, Id_Input, Clone, Chiave_file, DataOra-
File, DataOraLoadInp, DataOraLogCar, StatoLogCar
FROM ['+@dbconnessione+'].[dbo].[T_LOG_CARICAMENTO_FILE]
WHERE (StatoLogCar = ''cancellaFile'' OR Num_Rec_Load > 0)
AND Dimensione_file > 0) AS T
```

```
ON (S.Chiave_file = T.Chiave_file AND Stato <> ''CHIUSO'')
WHEN MATCHED THEN
UPDATE SET
Note_Segnalazione = ''Segnalazione chiusa automaticamente. Stati di errore pre-
cedenti: Load_DataOraLog(''+isnull(convert(varchar,Load_DataOra-
Log,121),''N/A'')+''),
Load_DataOraInp(''+isnull(convert(varchar,Load_DataOraInp,121),''N/A'')+''),
Id_Segnalazione(''+convert(varchar,Id_Segnalazione)+'')'',
Livello = ''INFO'',
Stato = ''CHIUSO'',
Load_DataOraInp = T.DataOraLoadInp,
Load_DataOraLog = T.DataOraLogCar;'

EXEC(@sql_query)

/*
Inserisce tutti gli errori SQL dei monitor interessati su T_LOG_GEST_AL-
LARMI_LOAD
*/

SET @sql_query ='MERGE [MOB10_GESTIONE].[dbo].[T_LOG_GEST_ALLARMI_LOAD] AS S
USING(
SELECT ID, DATA_INSERT, UTENTE, STEP, MESSAGGIO
FROM ['+@dbconnessione+'].[dbo].[T_LOG_ERRORI_SQL]
WHERE datediff(day,DATA_INSERT,getDate()) < 4
and messaggio not like ''%deadlocked%''
)AS T
ON (S.Chiave_File = convert(varchar,T.ID) AND
S.Id_modulo = '''+@descrizione+''')
WHEN NOT MATCHED THEN
INSERT (Id_Modulo,Dt_Rif,Attore,Id_Input,Clone,Chiave_File,
Gest_DataOraFile,Load_DataOraInp,Load_DataOraLog,Id_Segnalazione,
Note_Segnalazione,Livello,Stato,Dt_presa_in_carico,User_rif,Note_User)
VALUES ('''+@descrizione+''',
convert(varchar,DATA_INSERT,112),
UTENTE, ''SQL_ERROR'', NULL, ID,
DATA_INSERT, NULL, NULL, 8888, ''Step: ''+STEP+'' Errore: ''+MESSAGGIO, ''ER-
ROR'', ''INSERITO'', NULL, NULL, NULL);'
EXEC(@sql_query)
FETCH NEXT FROM cursore_moduli INTO @dbconnessione,@descrizione;
END;
CLOSE cursore_moduli;
DEALLOCATE cursore_moduli;
END
```

In questa stored procedures posso anche spiegare la tecnica con cui ho risolto la richiesta di avere una funzionalità che possa essere attivata o disattivata modificando un campo di una tabella anagrafica. Per fare questo, ho utilizzato le proprietà delle stringhe per creare del SQL dinamico sfruttando anche un elemento che si chiama Cursore; Per introdurre questo elemento devo fare una premessa:

Nei database relazionali le operazioni vengono eseguite su set di righe completi. Il set di righe restituito da un'istruzione SELECT, ad esempio, è costituito da tutte le righe che soddisfano le condizioni specificate nella clausola WHERE dell'istruzione. Il set di righe completo restituito dall'istruzione è noto come set di risultati. Le applicazioni, soprattutto le applicazioni online interattive, non sono sempre in grado di gestire in modo efficiente un intero set di risultati come singola unità. In tali applicazioni deve essere pertanto disponibile un meccanismo per l'elaborazione di una riga singola o di un blocco di righe di dimensioni ridotte. I cursori sono un'estensione dei set di risultati che implementano appunto tale meccanismo. I cursori estendono l'elaborazione dei risultati nel modo seguente:

- Consentono il posizionamento su righe specifiche del set di risultati.
- Recuperano una riga o un blocco di righe dalla posizione corrente del set di risultati.
- Supportano la modifica dei dati delle righe in corrispondenza della posizione corrente del set di risultati.
- Supportano livelli diversi di visibilità per le modifiche apportate da altri utenti ai dati del database inclusi nel set di risultati.
- Consentono alle istruzioni Transact-SQL incluse in script, stored procedure e trigger di accedere ai dati di un set di risultati.

Quindi ho usato un cursore per eseguire le operazioni per ogni set di risultati ottenuti dalla seguente query:

```
SELECT DBCONNESSIONE, A.MODULO from [MOB10_GESTIONE].[dbo].[T_ANAGRAFICA_CON-
SOLE_CENTRALE] A
JOIN [MOB10_GESTIONE].[dbo].[T_ANAGRAFICA_MODULI] B
ON A.MODULO = B.MODULO AND A.ATTIVO = 1 AND B.ATTIVO = 1
```

Ritornando questo risultato:

DBCONESSIONE	**MODULO**
MOB10_PSH_SCT_DETT	MSCD
MOB10_PSH_SCT_OUTG_DETT	MPAD
MOB10_PSH_TRAM	MPSHTRAM

Che estrae il Database e il Codice del modulo, di quei moduli in cui voglio attivare la funzione per trovare i file scartati ed gli errori SQL. In questo caso specifico, quindi , il cursore effettuerà tre cicli e per ogni ciclo popolerà le variabili @dbconnessione e @descrizione con i relativi valori della SELECT.

A questo punto, l'istruzione

```
SET @sql_query =
'exec ['+@dbconnessione+'].[dbo].[carSp_Popola_Log_Caricamento_File]'
EXEC(@sql_query)
```

Costruisce la seguente stringa:

```
exec [MOB10_PSH_TRAM].[dbo].[carSP_Popola_Log_Caricamento_File]
```

che con il comando EXEC(@sql_Query) viene eseguito.

Con lo stesso metodo proposto costruisco poi l'istruzione MERGE che andrà a popolare la tabella T_LOG_GEST_ALLARMI_LOAD con i record dei file che sono stati scartati. Le condizioni where applicate applicano le seguenti regole:

1. Deve essere stato segnalato come "fileScartato" o come "fuoriOrario" oppure lo statoLogCar ha valore nullo da più di due ore. Questa condizione, verifica che una chiave, inserita dal comando di BULK non è stata poi trovato nel log del caricamento, e quindi è potenzialmente un caso di file scartato.
2. Deve avere una dimensione in byte maggiore di zero, segno questo che il file non era vuoto
3. Deve avere il valore di *Num_REC_Load* uguale a zero, questa condizione ci indica che nelle tabelle di input non vi sono record che appartengono a quel file, di conseguenza, non è presente in esse.

Quando queste condizioni sono verificate , inserisco nella tabella un record, andando a definire un ID della segnalazione , il livello della segnalazione impostato ad "ALERT" e lo stato "INSERITO" che indica una nuova segnalazione.

Costruisco un'altra query dinamica, effettuando un altro comando MERGE, questo serve per aggiornare eventuali segnalazioni che erano state inserite in precedenza. Le condizioni di MERGE vanno questa volta a selezione i record che descrivono dei file caricati correttamente quindi:

- *StatoLogCar* = "CancellaFile" , il caricatore ha cancellato il file caricato, oppure *num_rec_load* è maggiore di zero, segno questo

che il nome del file è stato trovato in alcune righe delle tabelle di input.
- La dimensione del file è maggiore di zero.

In questo caso, viene effettuato un update sulle chiavi che soddisfano il match e si vanno ad aggiornare i dati e a chiudere le segnalazioni portandole a livello "INFO", stato "CHIUSO", ed aggiornando i valori sulla data e ora in cui è stato caricato ed è stato inserito nella tabella di input.

Infine, l'ultima parte dello script mi crea una query di MERGE per inserire i record trovai nella tabella T_LOG_ERRORI_SQL, in questa tabella vengono loggati gli errori prodotti dalle stored procedures presenti sul modulo. Questa tabella ha una struttura identica per ogni modulo di MOBU, per questo è possibile usare una query dinamica , pilotata dalla stessa anagrafica, per poter raccogliere anche queste informazioni.

Ho dovuto però adattare il dato: Infatti nel campo chiave_file vado ad inserire l'ID della riga della tabella dei LOG. Questo non causa problemi perché si tratta di valori numerici incrementali, ed essendo il campo chiave_file un varchar ho la garanzia che non possa creare problemi con cast impliciti. Questo tipo di segnalazioni, non seguono la stessa logica delle altre, per ogni segnalazione viene inserito un record, in modo da sapere se una stored sta andando costantemente in errore o meno, inoltre, andando a loggare pure i parametri , ho la possibilità di poter subito analizzare il problema per cercare di porre rimedio.

ID	Id_Modulo	Attore	Id_Input	Chiave_File	Gest DataOraFile	Load DataOraInp	Load_DataOraLog	Id_Segnalazione	Note_Segnalazione	Livello	Stato
4573	MPSHTRAM	SPIMI\MOB10_USR	SQL_ERROR	269	2017-05-10 13:53:14.300	NULL	NULL	8888	Step: carSp_MisureWhere Errore: Cannot insert the value NULL into column 'CHIAVE_PROC', table 'MOB10_PSH_TRAM.dbo.T_DETT_STATI_TRX_APP'; column does not allow nulls. INSERT fails.	ERROR	INSERITO
7022	MPSHTRAM	MIP	MOBU_PSH_ICF_IP	0000000000C9637B	2017-05-23 23:58:22.000	2017-05-23 23:58:22.000	2017-05-24 04:04:14.837	9999	Segnalazione chiusa automaticamente. Stati di errore precedenti: Load_DataOraLog(N/A), Load_DataOraInp(N/A), Id_Segnalazione(9999)	INFO	CHIUSO

Figura 5 - Esempio di segnalazioni

5.5 File in ritardo

La struttura di questa tabella si presta bene quindi ad ospitare più tipi di segnalazione , anche diverse da quelle dei file scartati, questo mi ha permesso di integrare anche altre segnalazioni come possibili file in ritardo, conoscendo l'anagrafica degli input e il loro intervallo di alimentazione. Per integrarle, ho creato una stored procedures che andasse a cercare nella anagrafica degli input dove sono presenti gli intervalli con cui un attore alimenta il cruscotto. Anche questa ricerca è fatta utilizzando la tabella di anagrafica creata nel database di gestione quindi segue il modello di attivazione del resto delle funzionalità.

```
CREATE PROCEDURE [dbo].[spTrovaFileInRitardo]
AS
BEGIN
SET NOCOUNT ON;
/*Cerca tutti i moduli con file in ritardo e li inserisce in una tabella tempo-
ranea. */

set language us_english
```

```
SELECT
MAX(az.ORA_FILE) AS ULTIMA_ALIMENTAZIONE,
az.ID_INPUT, gm.MODULO, AZ.AMBIENTE,
an.ORA_DA, an.ORA_A, an.INTERVALLO,
DATEDIFF(MINUTE,MAX(az.ORA_FILE),GETDATE()) AS MIN_RITARDO,
LTRIM(DATEDIFF(MINUTE, 0, an.TEMPO_ALLARME)) AS TOLLERANZA_RITARDO
INTO #TMP_RITARDI
FROM [MOB10_GESTIONE].[dbo].[LOG_AZIONI_INPUT] as az
join [MOB10_GESTIONE].[dbo].[GESTORE_ANAGRAFICA_INPUT] as an
on az.ID_INPUT = an.id_input
join [MOB10_GESTIONE].[dbo].[T_ANAGRAFICA_CONSOLE_CENTRALE] as gm
on gm.ID_COPIA like an.ID_COPIA
WHERE
az.AZIONE = 2 and an.Attivo_GM = 1
GROUP BY az.ID_INPUT, gm.MODULO, an.ORA_DA,an.ORA_A,an.INTERVALLO, an.TEMPO_AL-
LARME, AZ.AMBIENTE
HAVING DATEDIFF(MINUTE,MAX(az.ORA_FILE),GETDATE()) > LTRIM(DATEDIFF(MI-
NUTE, 0, an.TEMPO_ALLARME))
/*
Inserisce l'alert di un file in ritardo se non ancora presente, altri-
menti ne aggiorna i datetime per coprire la casistica
in cui sia arrivato un file in ritardo e, prima che venisse riese-
guita la SP schedulata, sia stato rilevato un nuovo ritardo
sullo stesso modulo.
*/
MERGE [MOB10_GESTIONE].[dbo].[T_LOG_GEST_ALLARMI_LOAD]
AS S
USING(
    SELECT * FROM #TMP_RITARDI
) AS T
ON(S.Chiave_File = T.id_input AND S.Stato <> 'CHIUSO')
WHEN NOT MATCHED THEN
INSERT (
Id_Modulo,Dt_Rif,Attore,Id_Input,Clone,Chiave_File,
Gest_DataOraFile,Load_DataOraInp,Load_DataOraLog,Id_Segnalazione,
Note_Segnalazione,
Livello,Stato,Dt_presa_in_carico,User_rif,
Note_User
)
VALUES (modulo, convert(varchar,GETDATE(),112), ambiente,
'RITARDO_FILE', NULL, id_input,
T.ultima_alimentazione, NULL, getdate(), 7777,
'File atteso in ritardo di ' + CONVERT(VARCHAR,T.MIN_RITARDO)
+ ' minuti. Tollerenza (' + T.TOLLERANZA_RITARDO + ' minuti)',
'ALERT', 'INSERITO', NULL, NULL, NULL
)
WHEN MATCHED THEN
UPDATE
```

```
SET
Dt_Rif=convert(varchar,GETDATE(),112),
Gest_DataOraFile=T.ultima_alimentazione,
Load_DataOraLog=getdate(),
Note_Segnalazione = 'File atteso in ritardo di ' + CONVERT(VARCHAR,T.MIN_RI-
TARDO) + ' minuti. Tollerenza (' + T.TOLLERANZA_RITARDO + ' minuti)';
/*
Le segnalazioni che non sono più presenti sulla tabella temporanea possono es-
sere chiuse automaticamente, in quanto
deve essere arrivato il file atteso.
*/
UPDATE [MOB10_GESTIONE].[dbo].[T_LOG_GEST_ALLARMI_LOAD]
SET Stato = 'CHIUSO', Livello = 'INFO', Note_Segnalazione = 'Segnala-
zione chiusa automaticamente. Il file è stato ricevuto in ritardo.'
WHERE Stato <> 'CHIUSO'
AND Id_Input = 'RITARDO_FILE' AND Chiave_File
NOT IN (SELECT id_input FROM #TMP_RITARDI)
DROP TABLE #TMP_RITARDI
```

Una volta realizzate le stored procedures del database di Gestione, ovvero quelle che popolano il dato sia a livello di Modulo che a livello di console centrale, ho creato un eseguibile batch che prendesse come parametro di input la lista delle stored procedures da lanciare e l'ho schedulato utilizzando il servizio MOB10_GESTORE_FILE, in questo modo avevo la libertà di impostare sia l'orario in cui volevo venissero lanciate, nel mio caso, H24, sia l'intervallo di tempo, che ho tarato tenendo conto dei tempi di esecuzione e delle necessità operative. Per non sovraccaricare la macchina e i moduli, ho optato per un intervallo di 15 minuti che era un giusto compromesso tra performance e intervallo di monitoraggio.

5.6 L'applicazione WEB – MOB10_GM

Per la visualizzazione delle segnalazioni, e l'interazione con esse, come anticipato nei capitoli precedenti, è stata sviluppata una applicazione web .NET MVC chiamata MOBU GLOBAL MONITOR, con acronimo MOB10_GM.

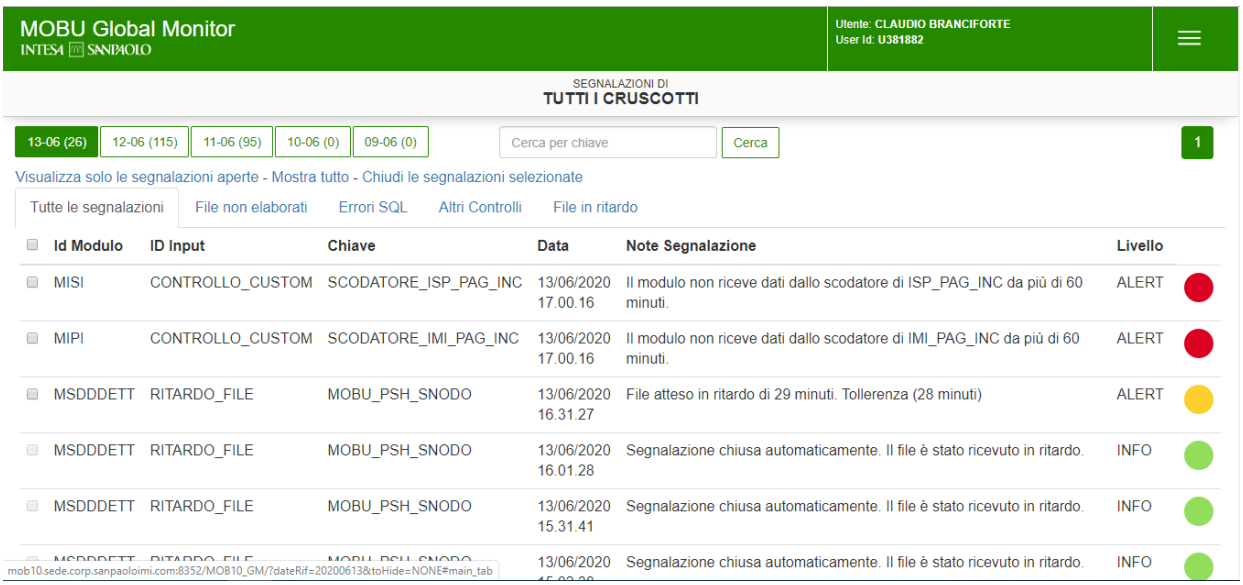


Figura 6 - Interfaccia MOBU GM

La dasboard principale si apre direttamente sulla situazione della giornata attuale. Le informazioni sono organizzate in una tabella con un semaforo che mi indica la gravità dell'alert mostrato. La pagina è organizzata in varie TAB che raggruppano le segnalazioni per tipologia, File non Elaborati, Errori SQL, Altri Controlli, File in Ritardo, che corrispondono alle segnalazioni che popolano il database, vi è comunque una TAB che raggruppa tutte le segnalazioni insieme, ordinate cronologicamente.

Sopra la tabella riepilogativa, troviamo alcune opzioni di visualizzazione, per nascondere i record che sono stati chiusi o per mostrarli tutti, ed una

funzione che , selezionati i record con la checkbox li chiude tutti massivamente. Si può ovviamente filtrare per data, utilizzando i tasti che riportano la data e tra parentesi il numero di segnalazioni presenti.

In alto, il titolo della dashboard, indica se sono stati effettuati dei filtri per modulo, infatti, aprendo il menù a scomparsa, appare la lista dei cruscotti per cui è attiva la console, che mostra in rosso il numero di segnalazioni totali per quel modulo, e quanti allarmi sono presenti attualmente nel cruscotto, con la possibilità di accesso diretto.

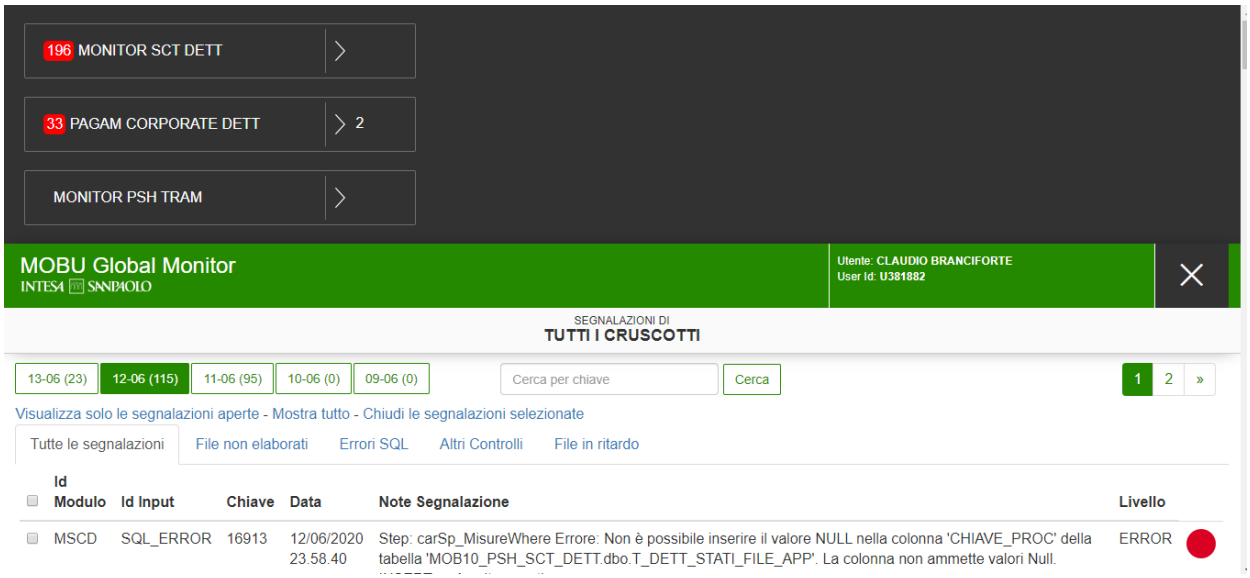
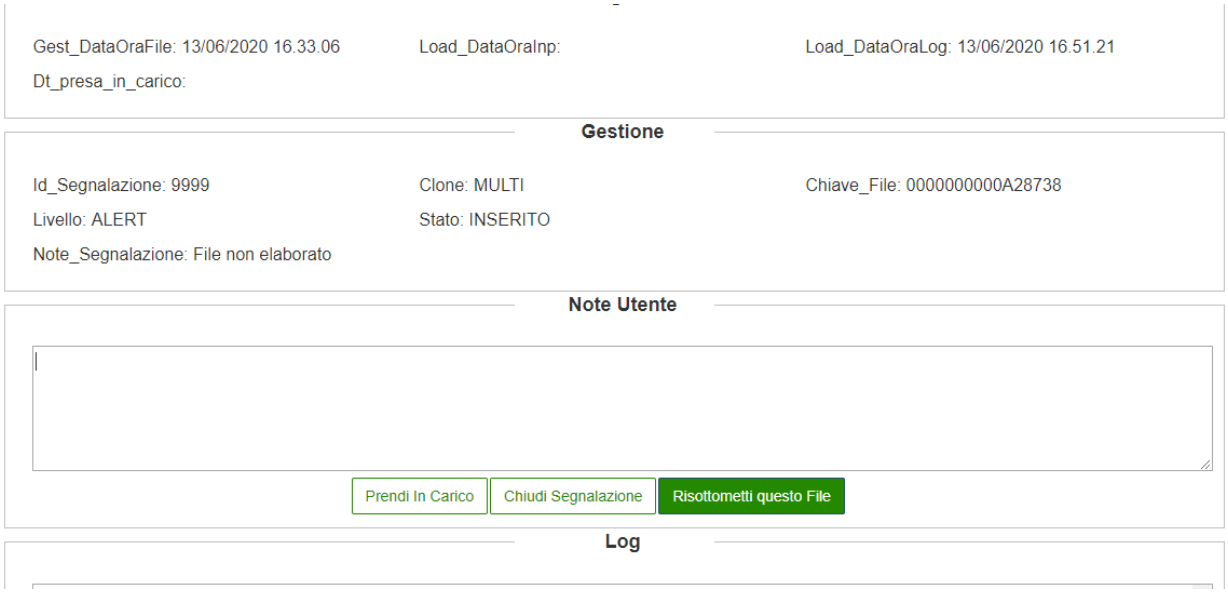


Figura 7 - 33 Segnalazioni su PAGAMENTI CORPORATE DET e 2 allarmi attivi nel monitoraggio

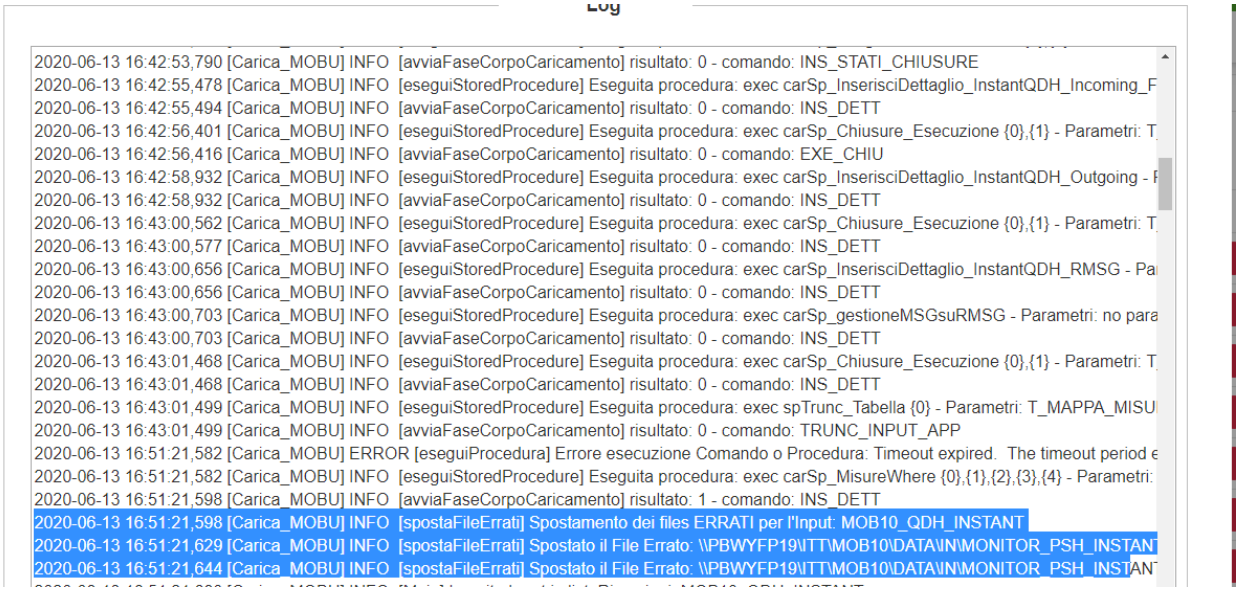
Cliccando su una delle righe della tabella, si apre un pop-up per la gestione della segnalazione. Nella parte superiore sono contenute delle informazioni più complete riguardo la segnalazione, come mostrato in figura

Nella parte inferiore invece si trovano i comandi per interagire con la segnalazione stessa. Infatti, essa può essere presa in carico da un utente, comodo per indicare a chi guarda la console che il problema è sotto analisi e inserire poi delle note, per specificare qualche commento che possa essere di aiuto.



I tasti di interazione sono semplici: Uno ti permette di prendere in carico la segnalazione, senza chiuderla, uno ti permette di chiuderla, ma solo dopo aver scritto una nota, ed il terzo, che appare solo nel caso in cui si stia gestendo la segnalazione di un file scartato, ti permette di mandarlo di nuovo in esecuzione. La parte grafica è stata curata con Bootstrap, un framework CSS che aiuta a realizzare delle pagine con aspetto dinamico, sfruttando il concetto della griglia a colonne. L'interazione con il back-end è stata fatta sfruttando chiamate in Ajax in modo da non dover ricaricare la pagina ad ogni interazione ed avere una esperienza utente più in linea con gli standard attuali. Razor invece è stato utilizzato per gestire e realizzare i model ricevuti dal back-end per poterli modellare, per nasconderli se non servono.

La funzione "Cerca Log" permette di andare a ricercare nel file di LOG del modulo, il punto in cui quel determinato errore si è verificato, essa risulta estremamente utile nel caso dei file scartati per avere una chiara idea di cosa sia accaduto. Per realizzarla è stato utilizzato un algoritmo di ricerca molto semplice, mi sono sempre basato sulla conoscenza della struttura del file. Il caricatore ogni qual volta carica un file o inizia un ciclo di caricamento scrive la parola START e quando conclude, utilizza la parola END,



andando a ricercare quindi all'interno della sezione START-END il nome del file scartato, si va a trovare la sezione di LOG in cui è stato elaborato e di conseguenza, si vede se durante quella elaborazione c'è stato un problema.

```
/* Action che viene richiamata via AJAX POST dalla pagina web e resitui-
sce la porzione di log da mostrare.
Prima individua il file corretto da aprire cercato tra tutti quelli del mo-
dulo in oggetto e utilizando la data della segnalazione e la data di ultima mo-
difica per individuare quello corretto. Poi cerca all'interno informa-
zioni sulla segnalazione.
*/
[HttpPost]
public JsonResult getLog(string filename, string idModulo,
DateTime dataOraFile, DateTime? dataOraLog){
try
{
log.Debug("Ricerca log per la segnalazione: " + filename + "(" + idMo-
dulo + ")");

var dbRow = gestione.T_ANAGRAFICA_CONSOLE_CENTRALE.First(x => x.MODULO == idMo-
dulo);

string[] fileList = System.IO.Directory.GetFiles(ConfigurationManager.AppSet-
tings["logFolderUrl"], dbRow.NOME_LOG + "*");
DateTime dateToCompare = dataOraLog != null ? (DateTime) dataOraLog : dataOra-
File;
string fileToRead = "";
double fileMinutesDiff = double.MaxValue;
/*Faccio una ricerca, andando a vedere quale file di log è quello più recente
rispetto a quanto segnalato nella tabella. */
foreach (string file in fileList) {
double minutesDiff = (System.IO.File.GetLastWriteTime(file) - dateToCompare).To-
talMinutes;
if (minutesDiff > 0 && minutesDiff < fileMinutesDiff){
      fileMinutesDiff = minutesDiff;
      fileToRead = file;
}
}
/*Apro il file in lettura, creo uno stream per leggerlo e farne il parsing uso
il flag trovato come condizione di uscita dal ciclo di ricerca*/
FileStream fStream = System.IO.File.Open(fileToRead, FileMode.Open, File-
Access.Read, FileShare.ReadWrite);
StreamReader reader = new StreamReader(fStream);
string line;
```

```
LinkedList<string> parsedLog = new LinkedList<string>();
bool trovato = false;
string timestampForLog = dataOraFile.ToString("yyyy-MM-dd HH:mm:ss");
    while ((line = reader.ReadLine()) != null) {
        if (idInput.Equals("SQL_ERROR")) { //se cerco errori SQL
             if (line.StartsWith(timestampForLog)) {trovato = true; }
        }else{ //se cerco il motivo di un file scartato
            if (line.Contains(filename)) {trovato = true; }
        }
/*Tra due sezioni che iniziano con start, sicuramente ci deve stare la segnala-
zione, se non è stata trovata, posso pulire la mia lista e proseguire a scorrere
il file fin quando non la trovo ed esco dal ciclo*/
        if (line.Contains("[Carica_MOBU] INFO  [Main] START")) {
            if (trovato) {break; }
            else{ parsedLog.Clear(); }
        }
        parsedLog.AddLast(line + "&#13;&#10;"); //caratteri di fine riga e a
capo
    }
//inserisco una riga di spiegazione e dei caratteri di fine riga e a capo per
dare leggibilità al file
    parsedLog.AddFirst("File: " + fileToRead +"&#13;&#10;&#13;&#10;");
    reader.Close();
    if (trovato) { return Json(parsedLog, JsonRequestBehavior.AllowGet); }
    else {
    return Json("Attenzione: non è stato possibile trovare la por-
zione di log interessata in nessun file", JsonRequestBehavior.AllowGet);
    }
}
catch (FileNotFoundException e)
{//Prima catturo le eccezioni con granularità minore
    log.Error("Si è tentato di aprire un file che non esiste");
    log.Error(e.Message);
    log.Error(e.StackTrace);
    return Json("Errore: Non è stato possibile recupe-
rare il file di log, il file richiesto non esiste.", JsonRequestBehavior.Allow-
Get);
}
catch (Exception e)
{
    log.Error("Errore durante la ricerca e l'apertura di un file di log");
    log.Error(e.Message);
    log.Error(e.StackTrace);
    return Json("Errore: Non è stato possibile recupe-
rare il file di log.", JsonRequestBehavior.AllowGet);
}
}
```

Questo controller viene richiamato quando si vuole cercare, all'interno del file corrispondente, la motivazione dello scarto del file. I parametri da passare al controller sono

- nome_file: il nome del file che è stato scartato
- idModulo: questo parametro mi serve per prendere le informazioni dalla anagrafica centrale, come ad esempio il nome del file di LOG da cercare.
- dataoraFile: serve per confrontare gli orari con quelli del file di log
- DataOraLog: parametro opzionale che in caso positivo, mi rende la ricerca più veloce.

Partendo dall'assunto che i file arrivati si trovano ad un path ben noto e che sono raggruppati in cartelle per ID_INPUT . Per trovare il file di mio interesse, prendo tutti i file con il nome del log trovato in anagrafica, essi differiscono perché in quelli dei giorni precedenti, vi è inserita la data di riferimento, con la funzione di sistema GetFiles prendo la lista di tutti quelli con la stessa radice, utilizzando il carattere speciale asterisco:

```
System.IO.Directory.GetFiles(ConfigurationManager.AppSettings["log-
FolderUrl"], dbRow.NOME_LOG + "*");
```

La funziona ritorna un elenco di nomi file in cui posso effettuare la mia ricerca. Per farlo, mi baso sul timbro orario passata come parametro al controller, poi calcolo la differenza con ogni file, andando a prendere il nome file in cui questa differenza è minima

```
string fileToRead = "";
double fileMinutesDiff = double.MaxValue;
/*Faccio una ricerca, andando a vedere quale file di log è quello più recente
rispetto a quanto segnalato nella tabella. */
foreach (string file in fileList) {
```

```
double minutesDiff = (System.IO.File.GetLastWriteTime(file) - dateToCompare).To-
talMinutes;
if (minutesDiff > 0 && minutesDiff < fileMinutesDiff){
      fileMinutesDiff = minutesDiff;
      fileToRead = file;
}
}
```

Una volta trovato il file in cui cercare, lo analizzo riga per riga andando a cercare il nome_file all'interno, ed in caso positivo, impostando un flag al valore true così da avere la condizione di terminazione del ciclo:

```
while ((line = reader.ReadLine()) != null) {
    if (idInput.Equals("SQL_ERROR")) { //se cerco errori SQL
      if (line.StartsWith(timestampForLog)) {trovato = true; }
    }else{ //se cerco il motivo di un file scartato
      if (line.Contains(filename)) {trovato = true; }
}
```

A questo punto, inserisco le righe trovate in una Lista di appoggio che mi servirà per restituire il risultato al chiamante, sfruttando la semplice serializzazione in JSON. In questo modo, sfruttando i metodi AddFirst() e AddLast() posso aggiungere una riga di intestazione e una riga finale per avere un risultato ottimale a video.

```
if (line.Contains("[Carica_MOBU] INFO  [Main] START")) {
    if (trovato) {break; }
    else{ parsedLog.Clear(); }
}
parsedLog.AddLast(line + "&#13;&#10;"); //caratteri di fine riga e a capo
}
//inserisco una riga di spiegazione e dei caratteri di fine riga e a capo per
dare leggibilità al file
parsedLog.AddFirst("File: " + fileToRead +"&#13;&#10;&#13;&#10;");
reader.Close();
if (trovato) { return Json(parsedLog, JsonRequestBehavior.AllowGet); }
else {
return Json("Attenzione: non è stato possibile trovare la porzione di log inte-
ressata in nessun file", JsonRequestBehavior.AllowGet);
}
```

Altro aspetto da considerare sulla soluzione realizzata è che la console risulta essere molto generalizzabile. Infatti, come ho adattato la struttura per poter segnalare gli errori SQL e i file in ritardo, allo stesso modo, si possono creare controlli generici che possano generare una segnalazione da visualizzare. Attualmente l'uso che viene fatto dello strumento è limitato alla mia attività, ma è facilmente estendibile pure alle altre attività dell'ufficio di cui faccio parte o ad altre attività di altri uffici. Questo potrebbe quindi fornire un ottimo strumento per aumentare l'efficienza nell'ambito di chi gestisce e segue il supporto AM di primo e secondo livello. Si potrebbe fornire uno strumento che possa essere adattato alle metriche e alle segnalazioni proprie per quel determinato ambito o ufficio fornendo anche a chi non ha una approfondita conoscenza dell'ambito, una segnalazione abbastanza chiara e precisa. Anche senza dover personalizzare le segnalazioni , quelle monitorate nella soluzione da me realizzata, costituiscono comunque un punto comune con molte altre applicazioni presenti nei nostri sistemi informativi. Il sistema di file transfer infatti è il principale middleware utilizzato per lo scambio dei file e , seppure con differenze, la logica di caricamento del dato può quindi essere estesa pure alle altre applicazioni, stesso discoro vale per le segnalazioni SQL, anche se lo stack su cui ho lavorato è .NET, i ragionamenti fatti possono essere estesi anche ad altri sistemi che , sicuramente, prevedono dei log tracciati sul database.

Nonostante la possibilità di generalizzazione, non ritengo però, che possa essere utilizzata una unica console in ambito di funzionamento, innanzitutto per via della organizzazione aziendale in strutture e uffici che gestiscono contratti e budget in modo autonomo, in secondo luogo, perché diventerebbe un single point of failure condiviso con altre strutture e altri uffici. Ritengo, che possa essere adottata nella realtà di un intero ufficio, suddividendo così le segnalazioni per attività e poi per ambito, in questo

modo anche l'implementazione può essere seguita internamente e utilizzata dai fornitori in funzionamento fornendo loro uno strumento unico che possa servire per rendere più efficiente le attività di supporto e funzionamento.

6. L'analisi dei risultati

Passo adesso ad analizzare gli impatti e i risultati ottenuti dalla applicazione della soluzione sviluppata sul modulo MOB10_PSH_TRAM, quali benefici sono stati ottenuti.

La funzionalità realizzata ha da subito eliminato del tutto le segnalazioni di falsi positivi, centrando così in pieno la richiesta pervenutami. Ha reso le attività di monitoraggio in questo specifico ambito più rapide ed efficienti. Con la funzione per rielaborare il file si riesce a mantenere una costante alimentazione dei cruscotti che permetteva un monitoraggio più uniforme ed una diminuzione degli allarmi presenti. Questo ha avuto un impatto positivo sui volumi trattati perché meno oggetti rimangono in fase di monitoraggio minore sono i volumi presenti sulle tabelle di dettaglio.

Ma il risultato migliore e più degno di nota , a mio avviso, non è quello di eliminare le false segnalazioni dei file scartati, ma è stato quello di monitorare sulla console, gli errori SQL prodotti dal cruscotto in fase di elaborazione del dato. È infatti stato uno strumento molto utile per poter analizzare tali situazioni in maniera precisa e puntuale. Spesso infatti non si riusciva a riprodurre il caso preciso in cui l'errore avveniva, perché il dato veniva storicizzato, perché avveniva su una tabella di staging che viene svuotata ad ogni caricamento, con queste segnalazioni, e con un presidio della dashboard realizzata, nei tre mesi successivi abbiamo praticamente eliminato quasi tutti i bachi software presenti nelle varie stored procedures o nelle condizioni where utilizzate per popolare i nodi del processo di monitoraggio. Il modulo di TRAMITATE ad oggi risulta quello con il tasso di errori

di elaborazioni minori , quelli che avvengono, salvo eccezioni, sono dovuti a file formattati male o che contengono dei dati con degli errori e che quindi non dipendo direttamente dalle elaborazioni da noi effettuate.

Considerando quindi gli ottimi risultati ottenuti sul singolo modulo, si è optato per estendere tale funzionalità ad altri due moduli con le stesse esigenze sperando di ottenere risultati simili. Gli effetti sono stati gli stessi, le segnalazioni errate di file scartati sono state eliminate e la possibilità di intervenire con analisi sugli errori di elaborazione in modo tempestivo ha portato ad un ottimo lavoro di bug-fixing che hanno aumentato la stabilità dei cruscotti.

Dopo un anno di utilizzo di tale soluzione, oramai integrata di default nella realizzazione di ogni nuovo cruscotto, si può effettuare una valutazione quantitativa del risparmio ottenuto, prendendo in esame le seguenti voci di costo:

- Risparmio sul costo di sviluppo software: Questa voce di costo rappresenta la spesa che avrebbe effettuato, una tantum, l'azienda per lo sviluppo di tale soluzione. Questa è stata stimata in un range che va dai 15k ai 18k.
- Riduzione degli FTE interni da 2 a 1 impiegati nel funzionamento: L' FTE (Full Time Equivalent) è l'unità di misura che rappresenta l'impiego di una risorsa, interna o esterna, full time. Il costo medio del fornitore esterno in ambito di funzionamento è circa 200 € più IVA, mentre si può approssimare a circa 205 € ivati per un collaboratore interno.

Quindi avendo ridotto il funzionamento da 2 FTE interni a 1 FTE possiamo estrarre i seguenti dati:

$$Sviluppo\ Software\ 15.000\ €\ una\ tantum$$
$$1\ FTE: 205 * 365 = 74.825\ €\ per\ anno$$

Si evince che il maggior beneficio è stato ottenuto nelle spese in conto esercizio, che sono quelle che non possono essere capitalizzate, come ad esempio il costo dello sviluppo software. A livello organizzativo, eravamo due risorse interne a svolgere il lavoro di funzionamento, la situazione non è cambiata ma adesso siamo impegnati in una percentuale minore del tempo, precisamente viene dedicato 0,5 FTE da ognuno di noi due per effettuare funzionamento e manutenzione, il restante tempo, può essere utilizzato in alla gestione progettuale in azienda.

Opere citate

ANDERSON, R., MULLEN, T., & VICAREL, D. (S.D.). SINTASSI RAZOR. TRATTO DA MICROSOFT.COM: HTTPS://DOCS.MICROSOFT.COM/IT-IT/ASPNET/CORE/MVC/VIEWS/RAZOR?VIEW=ASPNETCORE-3.1

MICROSOFT. (2010, 03 12). DOCUMENTAZIONE. TRATTO DA MICROSOFT.COM: HTTPS://DOCS.MICROSOFT.COM/IT-IT/ASPNET/ENTITY-FRAMEWORK

MICROSOFT. (S.D.). DOCUMENTAZIONE. TRATTO DA MICROSOFT.COM: HTTPS://DOCS.MICROSOFT.COM/IT-IT/SQL/T-SQL/STATEMENTS/MERGE-TRANSACT-SQL?VIEW=SQL-SERVER-VER15