ISTITUTO NAZIONALE DI FISICA NUCLEARE
Direzione Sistemi Informativi

INFN-24-10-DSI
24 Ottobre 2024

Datawarehouse della Direzione Sistemi Informativi INFN: Aggiornamenti alla Struttura dell'ETL

Thomas Angelini [1,2], Claudio Galli [1,2]

1) INFN, CNAF Centro Nazionale Analisi Fotogrammi, I-40121 Bologna Italy
2) INFN, Direzione Sistemi Informativi, I-00044 Frascati (Roma), Italy

Abstract

L'ETL (Extract, Transform, Load) è il processo che, in notturna, si occupa di aggiornare la banca dati del Data Warehouse INFN al fine di esporre agli utenti dati consistenti e certificati tramite strumenti di Business Intelligence (BI).

Essendo un processo complesso e articolato, la sua esecuzione tramite un tool dedicato richiede molto tempo, eppure è fondamentale per la certificazione dei dati da rendere fruibili tramite report e dashboard.

Uno degli obiettivi dell'Ufficio Data Warehouse e Business Intelligence della Direzione Sistemi Informativi (DSI) è quello mantenere ed evolvere questo processo, migliorarne le performance ed occuparsi della correttezza, completezza e consistenza del contenuto informativo della banca dati.

Questa nota tecnica si concentra sull'evoluzione dell'architettura di ETL

DOI n. 10.15161/oar.it/211860

Pubblicato da Laboratori Nazionali di Frascati

1 Introduzione

Con il termine ETL in genere si identifica una serie di processi che si occupano di importare sul data warehouse i dati grezzi dalle sorgenti, processarli per ripulirli e normalizzarli, ed infine caricarli sui sistemi di Analytics o Reportistica. Compito dell'ETL è anche la produzione di modelli dati multidimensionali particolarmente efficaci per una analisi qualitativa dei dati.

Il flusso di esecuzione dell'ETL viene progettato tramite il tool Tibco Jaspersoft JETL, il quale definisce la logica di estrazione, trasformazione e caricamento dei dati nel Data Warehouse. In JETL è possibile utilizzare diverse componenti per costruire un workflow operativo che implementi tutti i processi necessari.

I principali componenti consentono di:

- Aprire/Chiudere connessioni verso i principali database (nel nostro caso Oracle, PostgreSQL e MariaDB).
- Invocare una **procedura** o una **funzione** presente nel DB a cui si è collegati.
- Richiamare componenti che implementano costrutti del linguaggio *SQL* come Join, Filter, Union specifici per il database a cui si è collegati.
- Incapsulare un processo complesso nell'invocazione di un componente "*sub-job*" che si compone di due parti distinte:
  - *Selezione dati*: si occupa di caricare in memoria i dati a partire da una Select SQL che coinvolge una o più tabelle.
  - *Inserimento dati*: si occupa di riversare il risultato in una nuova tabella.

Il meccanismo descritto fin qui era di gran lunga il più utilizzato per la sua facilità d'uso. Implementa un flusso di esecuzione in diverse fasi che viene istanziato sulla macchina dedicata ai processi di ETL. Le fasi in ordine sono:

1. Viene aperta una connessione verso la sorgente dei dati.
2. Viene aperta una connessione verso il database di destinazione.
3. I dati vengono letti in funzione di una query SQL che ne manipola la rappresentazione e successivamente trasferiti in memoria sulla macchina che sta eseguendo i processi di ETL.
4. Il dataset risultante viene passato via rete al datawarehouse per la sua scrittura sulla nuova tabella.

In Figura 1. è riportato un esempio di sub-job.

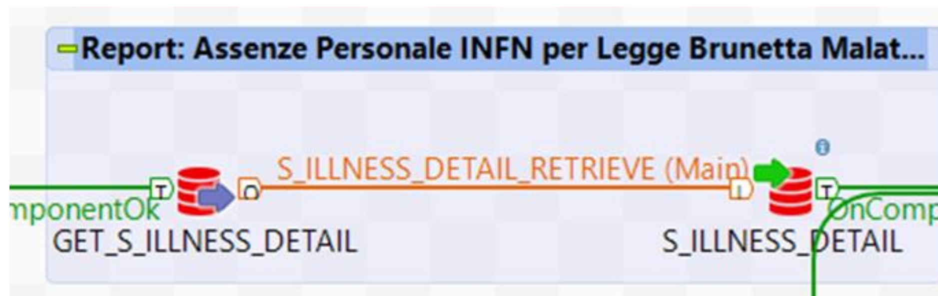

Figura 1: Nell’immagine il nodo GET_S_ILLNESS_DETAIL è adibito a leggere le righe da una tabella del database sorgente, le invia a S_ILLNESS_DETAIL il quale le memorizza in una nuova tabella nel db di destinazione. Il collegamento in rosso S_ILLNESS_DETAIL_RETRIEVE rappresenta il canale di traffico di rete tra la macchina sorgente, il nodo di elaborazione dell’ETL ed infine il canale di scrittura verso il data warehouse.

2 Ristrutturazione dell’ETL

L’ETL appena descritta è affetta da diverse limitazioni:

- Traffico in rete gravoso tra i nodi delle sorgenti e i nodi che eseguono i job dell’ETL e tra questi ed il data warehouse.
- Tempi di esecuzione complessivamente molto lunghi (circa 7 h 30 min per tutte le 4 fasi di elaborazione che si componevano di circa 30 job distinti).
- Processi di debugging e di messa in produzione estremamente onerosi e complicati.

Per ovviare a questi problemi sono state investigate diverse soluzioni, tra le quali si è scelto di armonizzarne alcune per ridurre il carico di lavoro, lo stress sull’infrastruttura ed alleggerire i processi di sviluppo, debug e messa in produzione.

In particolar modo ci si è concentrati su:

- Conversione in package e procedure delle business logic implementate nell’ambiente grafico JETL.
- Introduzione di Catene e Job a livello di database Oracle per l’esecuzione parallela e la gestione delle dipendenze tra i vari processi.

2.1 Conversione della Business logic grafica (JETL) in Procedure e Packages su Database

In questa fase di ottimizzazione ci siamo occupati di sostituire i sub-job descritti precedentemente con nodi di ETL che invocano l'esecuzione sul database di procedure PL-SQL. Queste sono state raccolte in Package secondo logiche di razionalizzazione ed aggregazione di funzionalità rispettando un naming che rispecchiasse le fasi logiche di elaborazione e i modelli dati in oggetto. All'interno delle singole procedure oltre alle fasi di creazione delle tabelle a partire da opportune query di S*elect* vengono gestite anche le fasi di logging su tabelle di registro dedicate.

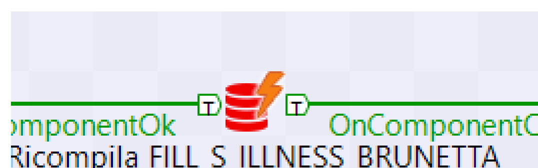

Figura 2: La figura mostra l'invocazione di una procedura PL/SQL dal tool JETL.

Questa modifica nel flusso di esecuzione dei processi di ETL abbatte la quantità di dati in transito sulla rete nelle fasi di lettura e scrittura delle risorse lavorando interamente sul nodo del data warehouse. Il primo effetto di questo cambio di prospettiva è una drastica riduzione dei tempi di esecuzione delle procedure in confronto alle equivalenti fasi dell'ETL grafica. I tempi di elaborazione complessiva dei job in questa fase sono passati da circa 7 ore e mezza a 4.

Un secondo effetto tangibile del cambio di prospettiva riguarda la resilienza del meccanismo di creazione e importazione delle risorse. Infatti, essendo utilizzata una formula "C*reate Table as SELECT …"* all'interno delle procedure, ne conseguono i seguenti benefici:

- **Errori di formato dei dati;** dato che non era più necessario specificare uno schema rigido per la formattazione, e dimensionamento, dei campi, non si verificano più errori a Runtime qualora sulle sorgenti compaiano dati di tipo diverso o lunghezza maggiore a quella prevista.
- **Ottimizzazione dimensione campi;** La query "C*reate Table as …*" gestisce a Runtime sia il formato sia la corretta dimensione dei dati ottenendo di fatto un risparmio in termini di spazio di memoria utilizzato per il caricamento dei dati.
- **Direct Path Oracle** (cit. [1])**;** l'utilizzo della sintassi "C*reate Table as SELECT…*" sfrutta delle ottimizzazioni specifiche di Oracle che rendono la query molto più rapida rispetto alla sequenza "Create Table" + "Insert Into"
- **Processo di sviluppo/bug-fix;** il fatto di aver portato le logiche di elaborazione sul database garantisce maggior flessibilità e possibilità di intervento, bug-fix o integrazione di nuove logiche, poiché sarà necessario solo intervenire sul sorgente della procedura.

Una volta sostituiti i vari sub-job con l'invocazione di procedure si è notato che risultava molto più efficiente e pulito raccogliere le varie procedure in un unico package "tematico"; ad esempio, tutte le procedure per la creazione del modello multidimensionale inerenti a Gare e Ordini sono presenti nello stesso package.

Un elemento importante del package, oltre alle varie procedure, è un particolare metodo (in genere chiamato CREATE_MDL_XXX) che si occupa di invocare nel corretto ordine tutte le procedure necessarie per la creazione delle risorse del modello. L'identificazione delle dipendenze è cruciale ai fini del corretto aggiornamento dei dati.

Questo nuovo approccio ha il grande vantaggio di garantire una maggiore flessibilità di intervento. Di fatto per **integrare nuove funzionalità in un modello è sufficiente aggiungere una nuova procedura e richiamarla all'interno del metodo principale del package**.

Figura 3: La funzione *CREATE_MDL_MIXED* genera tutte le tabelle che il package deve originare richiamando al suo interno tutte le altre procedure del pacchetto in ordine.

2.2 Esecuzione Parallela delle procedure

A questo punto della revisione dell'ETL siamo riusciti a ridurre di molto il carico di lavoro al costo di dover necessariamente serializzare le fasi di elaborazione e le chiamate a procedura. Chiaramente il tempo totale di esecuzione della nuova versione risulta comunque di molto inferiore a quella precedente, perché non affetta dalle latenze di rete.

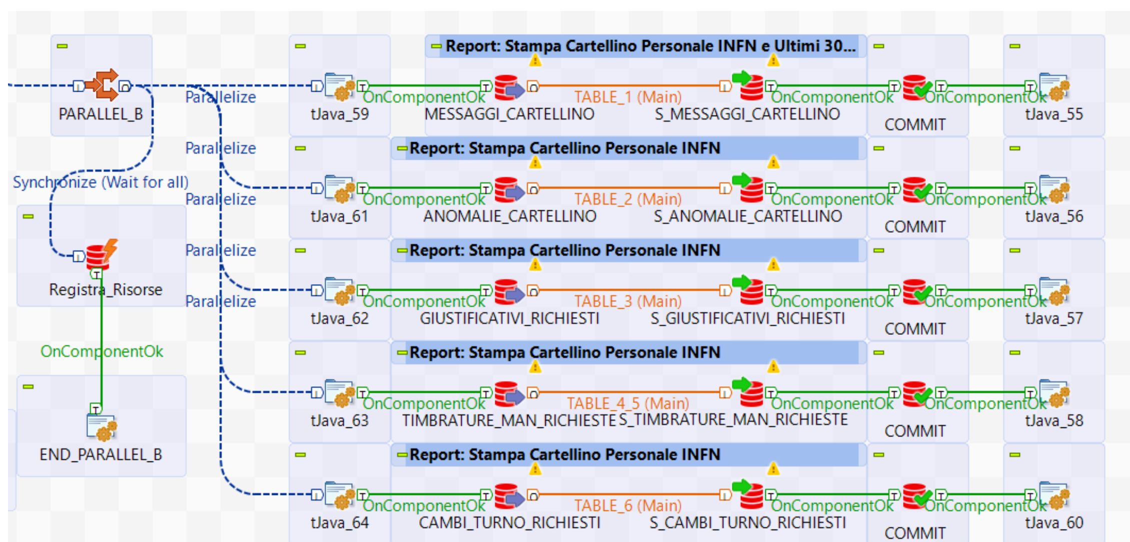

Figura 4: Esecuzione parallela in JETL di diverse tabelle e successiva sincronizzazione una volta terminati i flussi.

La limitazione dovuta alla serializzazione delle operazioni è stata superata introducendo l'uso delle **Oracle Job Chains** fornite dallo Scheduler Oracle che consente di avviare in parallelo tutte quelle procedure di uno stesso package che tra loro non hanno dipendenze.

Per poter impostare correttamente il lavoro è stato necessario profilare correttamente i permessi degli utenti di database in accordo con le policy di gestione introdotte dal reparto Infrastruttura che ci ha affiancato nel setup di base.

Una Catena di Job si compone di diversi nodi (job) per ognuno dei quali vengono definiti tre elementi base:

- **Program**: identifica la procedura che viene mandata in esecuzione.
- **Step**: associa il programma ad una Catena
- **Rule**: definisce le eventuali dipendenze del nodo da altre risorse (nodi della stessa catena).

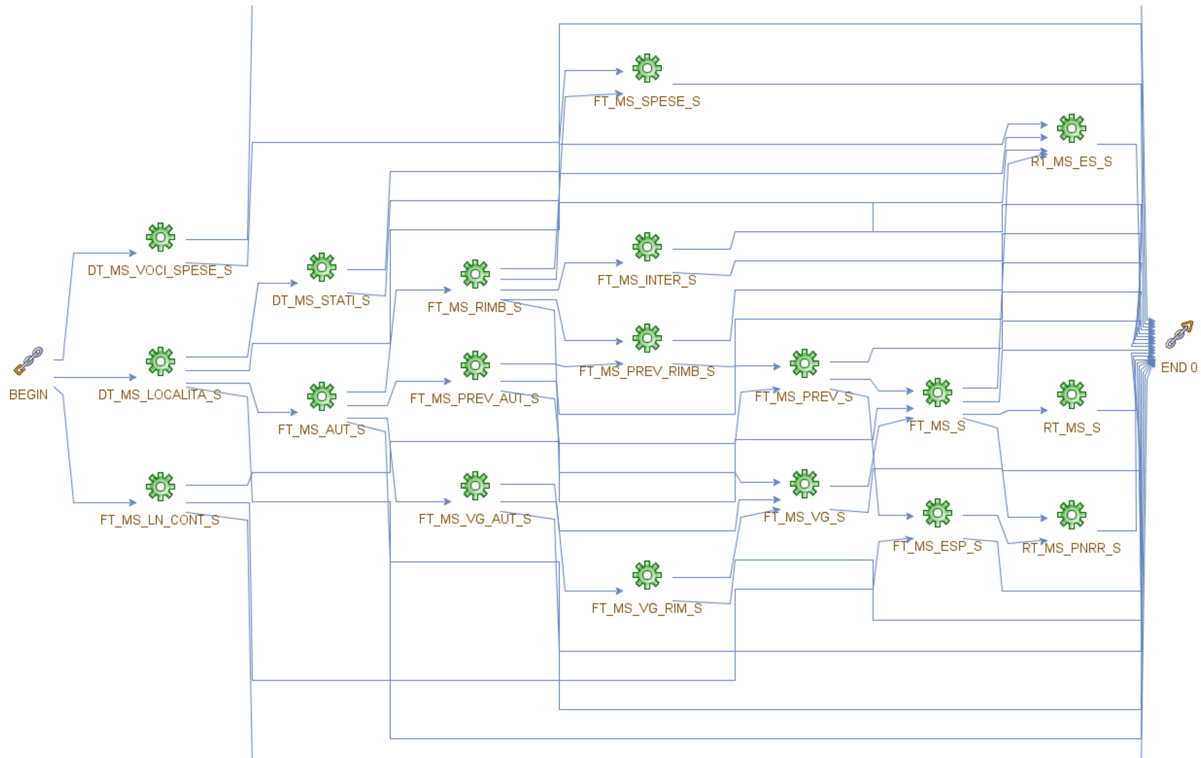

Figura 5: L'immagine mostra le dipendenze tra i nodi della catena di creazione delle tabelle inerenti al modello Missioni generate nella fase Preparatory ramo Contabilità. Gli ingranaggi verdi indicano i nodi, le frecce in blu sono le dipendenze. Ad esempio: una volta eseguito lo step *FT_MS_S* (sul lato destro della figura) si può procedere con il successivo *RT_MS_S*. A propria volta la generazione di *FT_MS_S* dipende da: *FT_MS_PREV_S*, *FT_MS_VG_S* e *FT_MS_AUT_S*. I nodi che non hanno interdipendenze possono essere eseguiti in parallelo tra loro, il compito è lasciato in gestione al DBMS_Scheduler di Oracle.

Fatte queste premesse è stato sviluppato un framework per rendere dichiarativa la fase di creazione e costruzione della Chain che viene quindi compilata sul database Oracle e configurata in automatico. L'effetto è che a partire da questa descrizione il compilatore Oracle riconosce le interdipendenze tra gli step e quali possono essere eseguiti in parallelo.

Per l'avvio della singola Chain è stata anche sviluppata una API specifica caratterizzata da due differenti modalità di esecuzione:

- **Chiamata non bloccante**: è possibile effettuare altre query mentre la catena sta eseguendo.
- **Chiamata bloccante**: il flusso restituisce il controllo all'utente solo al termine dell'esecuzione della catena.

È sempre possibile fermare l'esecuzione della Chain qualora venissero riscontrati problemi. Inoltre, si è scelto di tenere separati i package di libreria e quelli che definiscono le catene in modo da rendere più facile identificare il punto su cui intervenire nelle fasi di manutenzione evolutiva.

2.3 Esecuzione parallela di più Chain

Una volta ottenute le catene parallele per ogni modello, è stato possibile passare al livello successivo di ottimizzazione identificando quali **catene** non presentano interdipendenze e quindi possono a loro volta essere **eseguite simultaneamente**. In caso contrario è necessario identificare e risolvere le dipendenze per porle nella giusta sequenza.

Un miglioramento introdotto all'API sopra descritta ha consentito di gestire anche il parallelismo tra le singole Chain.

Un esempio viene riportato di seguito in relazione alla fase di Preparatory dei dati provenienti dal database di Godiva, in cui vengono gestiti in parallelo tre diversi modelli dati (Anagrafiche/Contratti, Timesheet e Ruoli/Profili/Gruppi).

```
utils_parallel.run_chains(
    UTILS_PARALLEL.PROCEDURES_NAMES(
                        CHAIN_MDL_GODIVA.nome, -- catena GODIVA
                        CHAIN_MDL_TIMESHEET.nome, -- catena TIMESHEET
                        --CHAIN_MDL_ASSEGNAZIONI.nome,
                        CHAIN_MDL_GODIVA_GRUPPI_BI.nome -- catena RUOLI - PROFILI
        ),
    WAIT_FOR_FINISH)
    ;
```

Figura 6: Nell'immagine viene mostrato il codice che implementa le operazioni della fase Preparatory per i dati provenienti dalla sorgente dati di Godiva. Si avviano le catene e il parametro *WAIT_FOR_FINISH* stabilisce se l'invocazione deve essere bloccante (come in questo caso), ovvero prima di procedere con altre operazioni si deve attendere la terminazione di tutti i Job delle catene coinvolte, oppure no.

In questo scenario il tempo di esecuzione complessivo dipende dalla durata dell'esecuzione della catena più lunga (temporalmente).

In altri casi, come nella fase di Preparatory dei dati di Bilancio, è possibile gestire in un unico step di elaborazione tutte le dipendenze comuni, per poi eseguire simultaneamente le Chain che si occupano dei diversi modelli dati.

```
CHAIN_MDL_COMMON_CONT.run_chain(1);

CHAIN_MDL_ESPERIMENTI.run_chain(1);

utils_parallel.run_chains(
    UTILS_PARALLEL.PROCEDURES_NAMES(
        CHAIN_MDL_BILANCIO.nome, -- catena BILANCIO
        CHAIN_MDL_GARE_ORDINI.nome, -- catena ORDINI
        CHAIN_MDL_MISSIONI.nome -- catena MISSIONI
        ),
    1);

chain_mdl_fatture.run_chain(WAIT_FOR_FINISH);
```

Figura 7: L'immagine mostra le catene avviate durante il Preparatory di Contabilità. Prima viene eseguita la catena *CHAIN_MDL_COMMON_CONT,* successivamente la catena per la generazione del modello Esperimenti. Entrambe sono utili per creare tutte le tabelle necessarie alle successive catene (il parametro *1* indica che la chiamate dev'essere bloccante, le altre catene partiranno una volta che questa sarà terminata).

In questa soluzione la durata complessiva è la somma del tempo di processamento della parte comune più la catena di maggior durata.

Il miglioramento riscontrato introducendo i vari livelli di parallelismo con le catene ha portato dalle circa quattro ore (ETL serializzata) a circa due ore e mezza (ETL con catene) migliorando ulteriormente le statistiche.

In definitiva si è passati dalle quasi otto ore iniziali a poco più di un quarto del tempo.

2.4 Architettura di base dell'ETL; Fasi e Rami

La primissima versione dell'ETL (v5.6) era caratterizzata da un unico processo java monolitico che avviato da uno script batch sulla macchina. Questo significava tempi estremamente lunghi, oltre le 9 ore, e una scarsa resilienza a fronte di errori minori nel flusso di lavoro.

Per ovviare a questo la prima revisione apportata nel 2021 ha introdotto una architettura modulare divisa in Fasi e Rami, dove:

- le Fasi mappano i concetti di importazione, pulitura e normalizzazione dei dati, correlazione e caricamento finale sul data warehouse.
- i rami rappresentano le sorgenti dati.

Di seguito si riporta una descrizione breve di entrambe le componenti architetturali dell'ETL che sono rimaste parte integrante della versione attualmente in produzione (v12.2).

Le Fasi vengono eseguite in serie e sono:

1. **Import**: come suggerisce il termine è la fase che importa i dati dal database sorgente sul Data Warehouse. Il nome delle tabelle importate ha il prefisso *T_*;
2. **Preparatory**: è la fase che prevede un'elaborazione dei dati importati per prepararli in seguito alla disponibilità dei report. Il nome delle tabelle importate ha il prefisso *P_*;
3. **Staging**: questa fase esegue un'elaborazione sulle tabelle realizzate dal Preparatory, molto spesso sono operazioni su tabelle di rami diversi. Il prefisso è *S_*;
4. **Datamart**: è la fase che rende visibili le tabelle agli strumenti di reportistica. Attualmente il gruppo usufruisce dei servizi PowerBi (prefisso delle tabelle *PB_*) e Jasperserver (prefisso *L_*).

I rami invece sono legati alle sorgenti dati che ne influenzano l'ordine di esecuzione:

- **Godiva**: sistema di autenticazione. Tipicamente viene svolto prima degli altri rami in quanto ha dati fondamentali per le elaborazioni successive;
- **Contabilita**: sono contenuti i dati in merito alle spese dell'ente;
- **Presenze**: database pesante riferito alle presenze ed ai giustificativi di tutto l'ente;
- **Mixed**: ramo eseguito a posteriori dei precedenti. Non è presente nella fase di import poiché non esiste una sorgente Mixed, è un ramo creato ad hoc per essere eseguito a posteriori degli altri
- **AS400**: ramo oramai non più utilizzato che si riferiva ad un import statico di dati dalla vecchia contabilità su tool AS400.
- **Fatture**: ramo in disuso che trattava dati di Contabilità specificamente sulle Fatture. È stato creato poiché si pensava che non fosse sufficiente un aggiornamento dati giornaliero per fatture bensì rifatto a frequenza maggiore;
- **Jasperserver**: Si occupa di reimportare nel data warehouse e quindi rielaborare i dati di setup e login del Jasper Report Server.

Ogni combinazione di fase e ramo è gestita da un flusso.

Nell'immagine a seguire viene riportato il flusso di esecuzione completo dell'ETL per ogni fase e ramo.

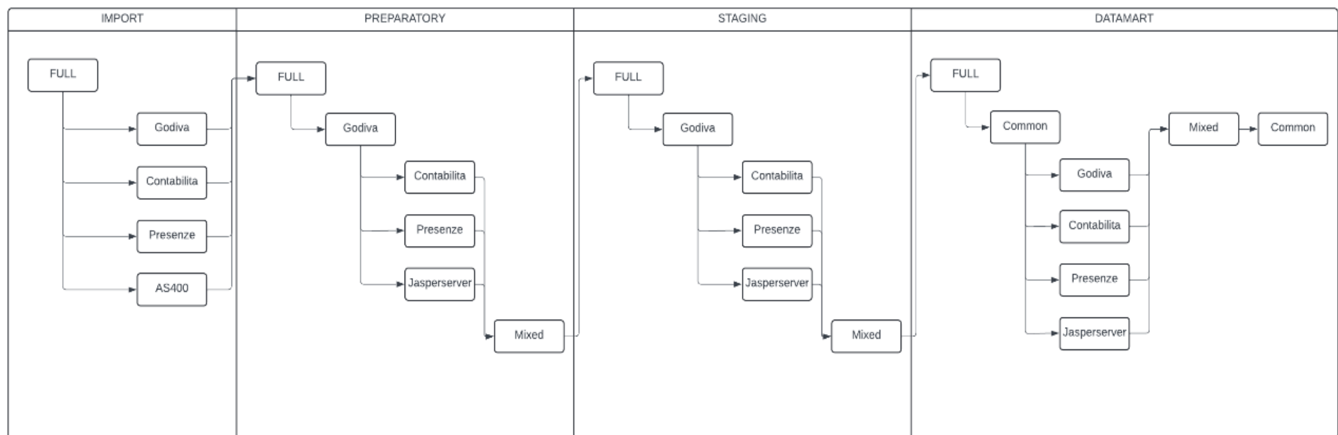

Figura 1: Flowchart generale dell'ETL che mostra le dipendenze tra Fasi e Rami. Da notare il fatto che Common viene richiamato due volte in Datamart, nel primo caso si utilizza la procedura Activate, a posteriori Deactivate.

2.5 Standardizzazione delle chiamate ETL

Al fine di standardizzare la nomenclatura dei componenti (packages) sviluppati per la business logic dell'ETL (cit. [2]), si è scelto il seguente formato:

<< Abbreviazione della FASE_RAMO >>

- Secondo lo schema:
- Import => IMP
- Preparatory => PREP
- Staging => STG
- Datamart => DATAM

(eg. "PREP_CONTABILITA", "DATAM_GODIVA", "STG_PRESENZE" ...).

Del package selezionato si richiamerà la funzione ***Activate()*** la quale contiene al suo interno tutte le istruzioni per la creazione di quelle tabelle che dovranno essere originate. Si agisce secondo il principio dell'incapsulamento, infatti gli oggetti possono essere creati da catene o in maniera seriale ma nella funzione non si desidera fornire il dettaglio su come vengano creati al fine di beneficiare la comprensione del codice. In questo modo sarà semplice lato JETL invocare la corretta chiamata.

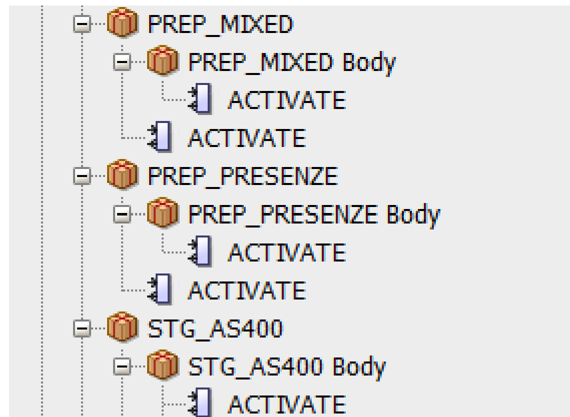

Figura 8: Esempio di Package descritti precedentemente.

2.6 Processo di Deployment dei Job di ETL

Ad ogni modifica sostanziale il processo di deployment della nuova versione di ETL contenente le nuove funzionalità deve seguire il percorso qui riportato:

1. A seconda della complessità della modifica può essere necessario intervenire solo sul codice SQL oppure ristrutturare completamente il sub-job cambiando anche la struttura della tabella finale.
2. Attuare tutti gli step del flusso di lavoro necessari a caricare i pacchetti software della nuova versione dell'ETL sui sistemi di sviluppo per poterli poi testare tramite Rundeck.
3. Attendere il riscontro dell'esecuzione della nuova versione del job (se coinvolti tutti i rami di una fase come Preparatory o Staging i tempi di attesa possono variare da 1h:40min a fino circa 4h)
4. Propagare il cambiamento all'ambiente di TEST per poi pianificare la messa in produzione.
5. Pubblicazione finale della soluzione su ambiente di Produzione.

Questo iter, prima delle modifiche discusse finora, era coinvolto in ogni singola fase di debug o ad ogni minima modifica all'ETL, e questo, insieme ad altre limitazioni tecniche, non consentiva la gestione di più branch. Quindi sia le attività di sviluppo che i bug-fix risultavano forzatamente serializzate.

Questo scenario risultava particolarmente refrattario alle innovazioni importanti come l'introduzione di modelli multidimensionali o lo sviluppo di nuovi moduli di ETL.

Le attività di ristrutturazione e ottimizzazione hanno quindi migliorato la capacità di risposta del gruppo a fronte di problemi sull'ETL, di richieste di integrazione di nuovi dati, e di sviluppo di nuove funzionalità.

2.7 Libreria per l'analisi degli errori delle catene

Una controindicazione dell’inserimento delle catene è la **difficoltà nell’effettuare l’azione di debug** in caso di errori. Di fatti, eseguendo la catena da strumenti IDE (Integrated Development Environment) come SQL-Developer, il comportamento di default è di portare a compimento l’esecuzione dei rami prima di ritornare il controllo all’IDE.

Qualora un nodo della catena fallisca, allora semplicemente non verranno eseguiti gli step che dipendono da quel nodo, viene generata una eccezione che non viene rilevata dall’IDE, a differenza di quanto accade durante l’esecuzione seriale di una procedura, quindi, non viene mostrato a video l’errore.

Per ovviare a questo problema abbiamo inserito nel package di utility che gestisce le chiamate alle catene un meccanismo di trace degli errori che li riporta in una tabella di log dedicata sul db in modo da poter verificare eventuali errori al termine delle operazioni di una o più catene.

	NOME	FASE	RAMO	CODICE	DESCRIZIONE	LINEA
1	VINCOLI MODELLO	DATAMART	GODIVA	-2260	ORA-02260: una tabella può avere solo una chiave primaria	ORA-06512:

LINEA	TIMESTAMP
ORA-06512: a "POWERBI.CONSTRAINTS", line 125ORA-06512: a "STAGEUSR.MDL_TIMESHEET_PBI", line 153	13-05-2024 14:49:19,843450000

Figure 9 e 10: Le due immagini mostrano il contenuto di una riga di tabella degli errori. Come si può evincere dalle figure sono descritti: nome (riassuntivo), fase, ramo, codice (di errore), descrizione, linee, timestamp.

Ci possono essere diverse cause di fallimento di una catena:

- Chiamata di **un metodo che contiene errori**
- **Non sono state definite correttamente le dipendenze tra gli step**; perciò, alla chiamata del metodo non è ancora presente una tabella necessaria
- **Più raramente condizioni di Deadlock** su una risorsa.

3 Conclusioni

La revisione dell'ETL ha migliorato il flusso di lavoro dei componenti dell'Ufficio Datawarehouse e Business Intelligence, oltre che migliorare drasticamente le performance dei processi coinvolti.

Di seguito, riportiamo i principali vantaggi ottenuti:

- Riduzione a circa ¼ dei tempi di elaborazione notturna (da circa 8h a 2,5h).
- Possibilità di intervenire integrando nuovi dati (aggiungere, modificare, cancellare tabelle) senza dover rilasciare una nuova versione dell'ETL grafica.
- Considerevole risparmio dell'uso della memoria sui nodi applicativi che eseguono i processi dell'ETL.
- Riduzione delle dipendenze tecnologiche dal tool JETL.
- Ridotto l'impatto sulla rete dell'infrastruttura di HA INFN, sia in termini di carico (MB nell'unità di tempo), sia in termini di tempo di utilizzo della banda.
- Possibilità di creare indici e chiavi referenziali sulle tabelle dei modelli dati multidimensionali che si riflettono in relazioni tra gli oggetti dei modelli dati di Power BI.
- Possibilità di tracciare sviluppi e bug-fix del sorgente in un unico repository Git su cui in futuro potremo abilitare anche tool di analisi del codice.

In conclusione, i vantaggi apportati da questo lavoro sono stati importanti e rilevanti ponendo solide basi per lo sviluppo e la crescita del backend del servizio. La riduzione dei tempi di elaborazione consentirà di allocare slot di elaborazione per i nuovi dati che dovranno essere integrati e correlati alle informazioni già gestite. La riduzione delle dipendenze tecnologiche ci consentirà di avere maggior margine di manovra nell'integrare nuove tecnologie ed intercettare dati provenienti da fonti eterogenee.

4 Riferimenti

[1] https://docs.oracle.com/cd/B10500_01/server.920/a96652/ch09.htm .

[2] http://www.lnf.infn.it/sis/preprint/getfilepdf.php?filename=INFN-23-08-SDI.pdf.

[3] https://docs.oracle.com/cd/B28359_01/server.111/b28310/scheduse009.htm

[4] https://docs.oracle.com/en/database/oracle/sql-developer-web/22.1/sdweb/chains.html