Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Gestionale

A.a. 2025/2026

Reingegnerizzazione dei flussi di integrazione dati in un data warehouse aziendale: architettura multilivello e implementazione di un framework ETL con Oracle Data Integrator

Relatore:
Dott. Simone Monaco

Candidato:
Leonardo Pompozzi

Tutor Aziendale:
Luca Bregata

Abstract

La trasformazione digitale e l'eterogeneità delle sorgenti informative impongono ai Data Warehouse aziendali elevati requisiti di scalabilità, governabilità e qualità del dato. Tali flussi di integrazione, spesso frutto di evoluzioni incrementali e soluzioni eterogenee, presentano criticità in termini di manutenibilità, tracciabilità e capacità di incorporare nuove fonti. La presente tesi descrive un progetto di reingegnerizzazione dei processi di integrazione dati realizzato presso l'azienda di consulenza informatica Var Group - Data Science Operations, volto a modernizzare l'architettura di alimentazione del Data Warehouse interno.

L'analisi dell'architettura attuale evidenzia pipeline non uniformi, tipizzazione debole delle sorgenti, carichi non incrementali e controlli qualitativi non strutturati; l'architettura obiettivo adotta un framework multilivello costruito da tre livelli (L0, L1 ed L2) allo scopo di separare preparazione, consolidamento e pubblicazione del dato.

L'implementazione dei livelli L0 ed L1 è stata realizzata attraverso l'utilizzo di Oracle Data Integrator (ODI), integrando una nuova sorgente, Webgate, alle due esistenti, SAM e OS Ticket, tramite API REST. Nel livello L0, dedicato alla preparazione del dato, è stata realizzata la replica controllata delle sorgenti con conservazione del dato grezzo e caricamenti incrementali basati sul calcolo del delta al fine di ridurre i volumi e garantire un allineamento temporale. Il livello L1 è invece orientato al consolidamento del dato, e introduce controlli strutturati di qualità del dato e integrità referenziale, con tracciamento degli scarti, storicizzazione del dato validato e consolidamento anagrafico secondo le logiche di Master Data Management. Questo livello produce infine strutture intermedie coerenti con il modello di pubblicazione, predisposte per l'alimentazione del livello L2 in cui il dato verrà utilizzato per la produzione di report utili all'analisi di business.

Il risultato del progetto è un flusso ETL standardizzato, scalabile e manutenibile, capace di supportare l'evoluzione del Data Warehouse e l'integrazione di nuove sorgenti informative, riutilizzabile e replicabile per l'implementazione dei livelli L0 ed L1 in diversi contesti progettuali, lasciando al cliente la definizione dei requisiti di reportistica, ambito principale del livello L2.

Indice

Elenco delle tabelle

Elenco delle figure

Elenco dei listing

Glossario

ICT Information and Communication Technology
DBA Database Administration
DWH Data Warehouse
BI Business Intelligence
CPM Corporate Performance Management
ETL Extract, Transform, Load
ELT Extract, Load, Transform
OLTP Online Transaction Processing
OLAP Online Analytical Processing
DSS Decision Support System
DFM Dimensional Fact Model
MPP Massively Parallel Processing
CDC Change Data Capture
API Application Programming System
REST Representational State Transfer
SSIS SQL Server Integration Services
iPaaS Integration Platform as a Service
SCD Slowly Changing Dimensions
ODS Operational Data Storage
MDM Master Data Management
ODI Oracle Data Integrator
SQL Structured Query Language
KM Knowledge Modules
ETLT Extract, Transform, Load Transform
ELTL Extract, Load, Transform, Load
AMS Application Management Services
PL/SQL Procedural Language / SQL
JSON JavaScript Object Notation
IKM Integration Knowledge Module
SK Surrogate Key

Capitolo 1

Introduzione

1.1 Contesto aziendale

Il mio tirocinio curricolare è stato avviato presso Mediamente Consulting S.r.l. [1], società inizialmente nata come startup sviluppata nel 2012 presso l'incubatore I3P del Politecnico di Torino. Nel corso dei sei mesi di tirocinio, precisamente nel febbraio 2026, si è concluso il processo di acquisizione di Mediamente Consulting S.r.l. da parte di Var Group, parte del gruppo Sesa S.p.A., cominciato nel novembre del 2022, con l'obiettivo di potenziare le competenze del gruppo nell'ambito della Data Science. A seguito di questa operazione, il mio percorso di tirocinio è stato formalmente riqualificato all'interno di Var Group - Data Science Operations, dove ho proseguito la mia esperienza formativa e professionale, in continuità con le attività e le modalità operative della precedente struttura aziendale.

Var Group è stata fondata all'interno del gruppo Sesa S.p.A. e ha sviluppato un modello di crescita basato su innovazione continua, integrazione di competenze tecnologiche avanzate e una forte capacità di accompagnare le imprese nei processi di evoluzione digitale, e oggi rappresenta una delle principali realtà completamente italiane nel settore dell'ICT e dei servizi per la trasformazione digitale. Var Group ha sede centrale ad Empoli, in provincia di Firenze, opera su tutto il territorio nazionale ed è presente anche a livello internazionale, dove si posiziona come partner strategico per le organizzazioni che intendono sfruttare la tecnologia per aumentare l'efficienza operativa e la competitività [2].

La sede presso cui ho svolto il tirocinio è quella di Torino, dove sono presenti le diverse Business Units tematiche in cui è suddivisa Var Group, specializzate nello sviluppo di soluzioni IT avanzate. Il mio percorso si è svolto all'interno della Business Unit di Data Science, articolata in più team funzionali:

- DBA: responsabile della gestione, sicurezza e disponibilità dei database aziendali; l'obiettivo di questo team è garantire che i dati siano organizzati correttamente, protetti e sempre accessibili, assicurando integrità, affidabilità e continuità operativa 24 ore su 24, 7 giorni su 7;
- Data Integration: si occupa dell'acquisizione, pulizia ed armonizzazione dei dati provenienti da sorgenti eterogenee. Il team estrae informazioni dai vari sistemi operativi, trasformandole secondo le regole di business e le esigenze del cliente e, infine, le carica all'interno di un DWH centralizzato. Grazie a questo processo, i dati risultano coerenti, standardizzati e pronti per l'analisi tramite strumenti di BI (principalmente Oracle Business Intelligence e Power BI), consentendo insights affidabili e significativi;
- Data Visualization: è il team responsabile della progettazione ed ottimizzazione di dashboard e report tramite l'utilizzo di piattaforme di BI, consentendo ai propri clienti di esplorare le informazioni che hanno a disposizione in modo intuitivo e prendere decisioni informate;
- CPM: si occupa di pianificazione, budgeting, forecasting e monitoraggio delle performance aziendali, supportando le organizzazioni nella gestione efficace delle risorse e nel raggiungimento degli obiettivi strategici.

Nel corso del tirocinio sono stato inserito all'interno del team di Data Integration, dove mi è stato affidato un progetto focalizzato sulla reingegnerizzazione dei flussi ETL/ELT del DWH interno e sull'integrazione delle nuove sorgenti applicative. Questo progetto costituisce l'oggetto principale della presente tesi e si inserisce all'interno di un'iniziativa aziendale più ampia, illustrata nei paragrafi successivi.

1.2 Flussi di integrazione dati attuali: criticità riscontrate

I flussi di integrazione dati attualmente utilizzati da Var Group - Data Science Operations sono il risultato di una crescita aziendale rapida e di sviluppi realizzati in momenti diversi, spesso con logiche differenti e non sempre coordinate tra loro. Nel tempo questo ha portato alla coesistenza di processi eterogenei, basati sia su strumenti di integrazione dati dedicati sia su procedure sviluppate direttamente a livello di database, senza un framework metodologico unitario.

Le principali sorgenti informative integrate nell'architettura attuale sono OS Ticket, utilizzato per la gestione interna dei ticket e delle attività operative, e SAM, ovvero il sistema gestionale che raccoglie informazioni relative a ordini, commesse, sotto-commesse, ore rendicontate e fatturazione. Le modalità con cui queste fonti sono state incorporate nel DWH sono state condizionate dalle esigenze contingenti

del momento, con sviluppi spesso mirati a risolvere necessità immediate più che a definire un impianto stabile e scalabile.

L'attuale modello dati, concepito originariamente per Oracle Business Intelligence e fortemente normalizzato, risulta oggi meno adatto rispetto agli strumenti di analisi più moderni verso cui l'azienda si sta orientando. A ciò si aggiungono ulteriori criticità operative, come la mancanza di standard uniformi nei processi di integrazione e l'utilizzo limitato di controlli strutturati sulla qualità dei dati. Una descrizione dettagliata dell'architettura esistente (AS-IS), delle sue criticità e dei limiti dei processi di integrazione è riportata nel Capitolo 3, che costituisce il riferimento tecnico per l'analisi dello stato attuale del sistema.

1.3 Obiettivi del progetto di reingegnerizzazione

L'obiettivo del progetto di reingegnerizzazione è riprogettare in modo organico il processo di integrazione dati su cui si basa il DWH aziendale, superando le principali limitazioni architetturali ed operative emerse nell'analisi dello stato attuale. L'intervento mira a definire un assetto più strutturato, scalabile ed allineato alle pratiche moderne di Data Integration, in grado di supportare sia l'evoluzione del modello informativo sia l'adozione di strumenti di analisi più recenti.

In questa prospettiva, il progetto prevede la razionalizzazione delle sorgenti esistenti, in particolare OS Ticket e SAM, e l'integrazione di nuove fonti informative, tra cui Webgate, all'interno di un framework di integrazione unitario. Un ruolo centrale è svolto dall'architettura a livelli basata sul framework L0 – L1 – L2, già utilizzato in azienda, che viene assunto come riferimento metodologico per organizzare in modo coerente le diverse fasi del ciclo di vita del dato. Nel Capitolo 3 tali obiettivi vengono approfonditi nel dettaglio, descrivendo il nuovo modello architetturale TO-BE e il contributo specifico dei livelli L0 e L1 all'intervento di reingegnerizzazione, mentre il livello L2 è inquadrato rispetto alla successiva migrazione verso strumenti di analisi come Power BI.

1.4 Struttura della tesi

La tesi è articolata in più capitoli, ciascuno dei quali affronta una fase specifica del progetto di reingegnerizzazione dei flussi di integrazione dati. Dopo il capitolo introduttivo, che presenta il contesto aziendale, le criticità riscontrate e gli obiettivi del lavoro, il secondo capitolo è dedicato alla rassegna dei riferimenti teorici e degli strumenti concettuali necessari per comprendere i principi alla base dell'integrazione e della modellazione dei dati.

Il terzo capitolo analizza in dettaglio l'architettura attuale dei flussi informativi, descrivendo le caratteristiche delle sorgenti OS Ticket e SAM, le modalità con cui

sono state integrate nel tempo e le principali problematiche tecniche ed operative che rendono necessaria una reingegnerizzazione complessiva del sistema; vengono inoltre delineati l'architettura target, organizzata secondo il framework aziendale L0 – L1 – L2, i principi progettuali alla base del nuovo modello, le scelte architetturali adottate ed il ruolo attribuito ai diversi livelli dell'infrastruttura, con particolare attenzione all'integrazione della nuova sorgente Webgate tramite REST API.

Il quarto capitolo è dedicato alla fase di implementazione, in cui vengono descritte le attività svolte per la costruzione dei livelli L0 e L1 utilizzando ODI; vengono presentate le logiche di acquisizione, normalizzazione e consolidamento dei dati, insieme alle soluzioni adottate per migliorare la qualità e la tracciabilità delle informazioni.

Nel quinto capitolo vengono infine raccolti i risultati ottenuti e viene proposta una valutazione del nuovo impianto rispetto alla situazione di partenza, evidenziando i miglioramenti ottenuti in termini di robustezza, coerenza e scalabilità; vengono inoltre esposte le possibili evoluzioni future.

Capitolo 2

Stato dell'arte e riferimenti teorici

In questo capitolo vengono esposti i fondamenti teorici e metodologici che fanno da cornice al lavoro presentato, con l'obiettivo di contestualizzare approcci, scelte progettuali e strumenti impiegati nel corso dell'attività di tirocinio. Attraverso lo studio delle principali teorie e dei contributi più rilevanti presenti in letteratura, spaziando dai modelli concettuali e logici del data warehousing, alle differenze tra implementazioni *on-premise* e *cloud-native*, fino alle metodologie di modellazione dimensionale e agli strumenti di integrazione, sarà possibile evidenziare come le soluzioni adottate nella pratica siano fondate su principi teorici consolidati. A partire da questo quadro teorico, i paragrafi successivi introducono i concetti necessari per comprendere in modo consapevole il flusso ETL che verrà analizzato nei capitoli successivi, creando un collegamento diretto tra i modelli descritti nella letteratura di riferimento e la loro applicazione nel contesto del tirocinio.

2.1 Architetture di DWH: modelli concettuali, logici e fisici

Il concetto di DWH nasce dall'esigenza di separare nettamente i sistemi che gestiscono le operazioni quotidiane da quelli che supportano le decisioni di medio e lungo periodo. In questa prospettiva, i sistemi operazionali (OLTP) sono dedicati alla registrazione efficiente delle transazioni, mentre i sistemi informazionali servono a osservare l'evoluzione dei fenomeni aziendali nel tempo e a guidare il management nelle scelte strategiche. William H. Inmon [3] formalizza questa distinzione definendo il DWH come una collezione di dati *"subject-oriented, integrated, time-variant e non-volatile"*, progettata per sostenere i processi decisionali e le analisi storiche.

Questa definizione implica che i dati siano organizzati per grandi aree tematiche aziendali (ad esempio cliente o prodotto), integrati da sorgenti eterogenee, dotati di una dimensione temporale esplicita e caricati in modalità prevalentemente append-only, evitando aggiornamenti continui voce per voce. Da ciò deriva la necessità di un'architettura separata rispetto ai sistemi OLTP, in cui i dati vengano ristrutturati e resi idonei a interrogazioni analitiche complesse tipiche dei DSS.

Parallelamente, Ralph Kimball [4] propone un approccio orientato ai data mart e alla modellazione dimensionale, secondo cui il DWH può essere visto come "la somma dei data mart dipartimentali integrati", collegati tra loro tramite dimensioni condivise. L'obiettivo è rendere i dati facilmente interrogabili dagli utenti di business, privilegiando schemi denormalizzati (come lo schema a stella o a fiocco di neve) che migliorano le prestazioni delle interrogazioni.

In questa prospettiva, la modellazione logica assume un ruolo centrale e viene strutturata tramite schemi denormalizzati (schemi a stella o a fiocco di neve) che ottimizzano le analisi multidimensionali; entrambi gli approcci convergono sull'idea che l'architettura del DWH debba essere multilivello. Matteo Golfarelli e Stefano Rizzi [5] formalizzano questa visione proponendo un insieme di livelli concettuali e logici: un livello di *staging*, in cui i dati vengono estratti e puliti, un livello centrale di DWH, spesso strutturato secondo uno schema multidimensionale, e un livello di presentazione orientato agli strumenti di analisi.

In questo modello, l'architettura logica viene progettata come una rappresentazione astratta della realtà decisionale, mentre il modello concettuale, spesso basato su un formalismo multidimensionale, descrive i fatti, le misure e le dimensioni che caratterizzano i fenomeni aziendali da analizzare.

Inmon [3] propone un'architettura corporativa centralizzata, in cui esiste un unico DWH integrato, dal quale derivano i data mart dipartimentali; Kimball, al contrario, promuove uno sviluppo bottom-up, in cui i data mart vengono progettati come componenti congiunte di un modello unificato dei processi aziendali e delle dimensioni, basato su dimensioni coerenti e condivise tra i vari domini. Entrambi gli approcci, tuttavia, riconoscono che l'architettura del DWH è principalmente un'architettura di dati, non un'implementazione tecnologica, e che il modello logico deve rappresentare un livello stabile tra i requisiti informativi e la realizzazione fisica.

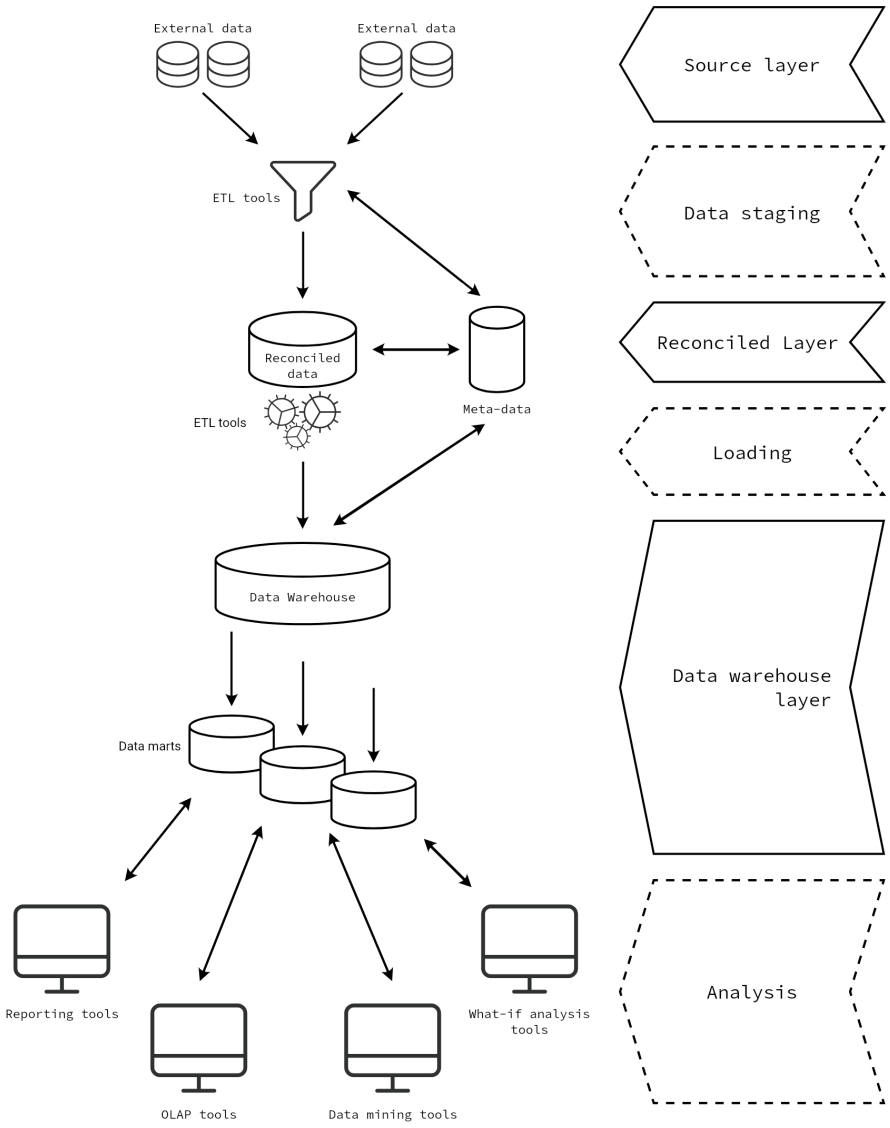

Figura 2.1: Architettura multilivello del DWH con livello di *staging*, livello centrale e livello di presentazione/analisi

Nella progettazione concettuale, Golfarelli e Rizzi introducono il DFM, un formalismo che consente di rappresentare gerarchie, aggregazioni, dipendenze e proprietà delle misure in modo rigoroso e indipendente dalla tecnologia adottata [5].

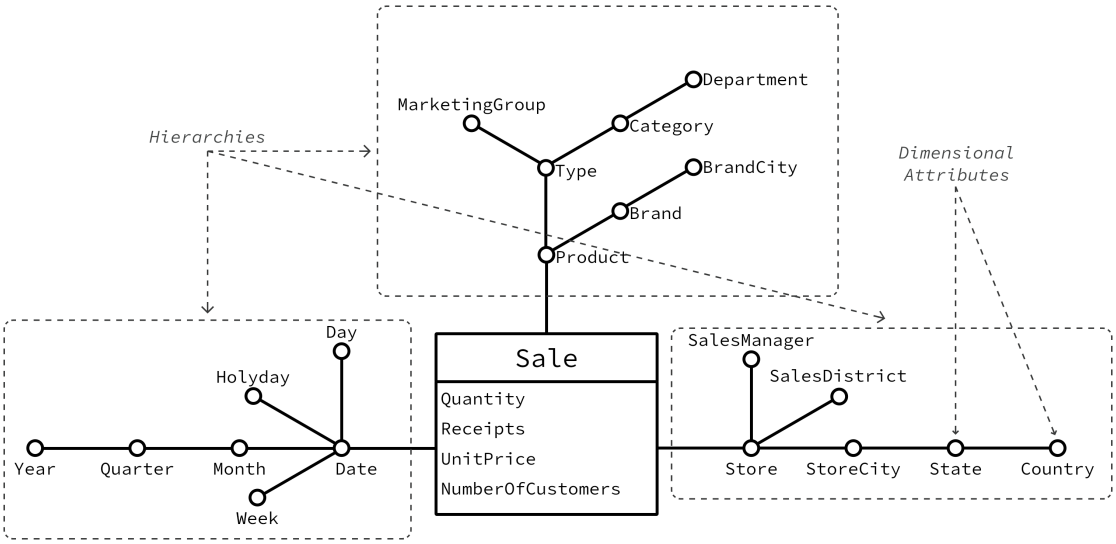

Figura 2.2: Esempio di DFM con fatto, misure, dimensioni e gerarchie

Questo livello concettuale risulta essenziale per garantire che le esigenze informative degli utenti finali vengano tradotte correttamente in strutture analitiche. In sintesi, la letteratura converge sulla distinzione tra:

- un modello concettuale, orientato ai fenomeni decisionali e descritto tramite fatti, misure e dimensioni;
- un modello logico, che formalizza tali concetti in schemi multidimensionali coerenti e ottimizzati;
- un'architettura fisica, che implementa il DWH su una piattaforma specifica.

Questi tre livelli costituiscono la base teorica per la progettazione e lo sviluppo di qualsiasi processo ETL e di qualsiasi flusso di integrazione dati, inclusi gli scenari trattati nel caso applicativo sviluppato in questa tesi.

2.2 Data Warehouse *on-premise* vs. *cloud-native*

L'evoluzione delle tecnologie di gestione dei dati ha trasformato in modo significativo l'architettura dei sistemi di DWH, introducendo un passaggio graduale dai modelli *on-premise* tradizionali verso soluzioni *cloud-native*. Le architetture *on-premise*, che hanno dominato per decenni, si basano su infrastrutture fisiche interne all'organizzazione e richiedono ingenti investimenti in hardware, manutenzione, aggiornamenti e personale specializzato.

Khawaja Ubaid Ur Rehman e Sajid Mahmood [6], in uno studio del 2018 sull'analisi comparativa tra DWH tradizionali e *cloud-native*, evidenziano che tali sistemi garantiscono alle aziende un elevato grado di controllo sull'intero ciclo di vita dei dati e sulla configurazione delle risorse, ma comportano costi operativi elevati e limitazioni nell'adattarsi rapidamente a esigenze variabili di carico e capacità elaborativa. Il progressivo spostamento verso il cloud ha introdotto un modello alternativo, caratterizzato da una diversa gestione della scalabilità, del provisioning delle risorse e dell'infrastruttura. Darko Golec, Ivan Strugar e Drago Belak sottolineano che il principale vantaggio delle soluzioni *cloud-native* risiede nella loro elasticità: le risorse possono essere incrementate o ridotte dinamicamente in risposta al volume dei dati e alla complessità delle elaborazioni, risparmiando all'azienda il costo fisso di mantenere un'infrastruttura sovradimensionata [7]. Inoltre, il modello pay-per-use consente una gestione più efficiente dei costi, spostando la spesa da capitale a costo operativo e riducendo notevolmente il tempo di implementazione di nuove funzionalità.

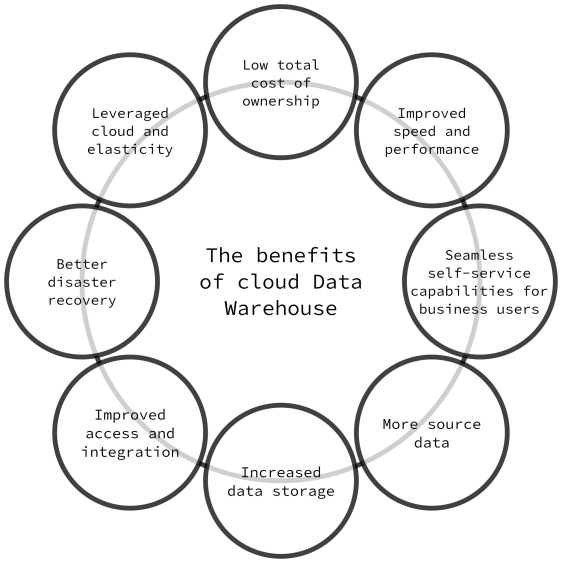

Figura 2.3: Principali benefici del DWH in cloud in termini di costo, prestazioni ed elasticità

Dal punto di vista architetturale, i DWH *cloud-native* introducono funzionalità che non trovano un corrispettivo diretto nei sistemi *on-premise* tradizionali. Come osservato da Bhushan Fadnis [8] nel 2024, piattaforme moderne come BigQuery, Snowflake o Redshift adottano un paradigma MPP completamente gestito, in cui lo storage è disaccoppiato dalla potenza di calcolo. Tale separazione consente di ottimizzare in modo indipendente le prestazioni e i costi, superando il limite strutturale delle architetture *on-premise*, in cui la scalabilità verticale rappresenta

spesso un collo di bottiglia. La possibilità di sfruttare cluster distribuiti, nodi elastici e meccanismi automatici di ottimizzazione rende le soluzioni *cloud-native* particolarmente adatte a scenari caratterizzati da grandi volumi di dati, carichi analitici intensivi e variazioni dinamiche della domanda computazionale. Nonostante queste differenze operative, le basi concettuali del data warehousing rimangono sostanzialmente invariate.

Dheeraj Kumar Bansal [9], in un contributo del 2025, richiama l'attenzione sul fatto che la progettazione logica dei DWH continua a fondarsi sugli stessi principi di integrazione, qualità del dato, persistenza storica e modellazione multidimensionale introdotti nelle architetture tradizionali. Il cloud introduce nuove possibilità tecniche, ma non modifica la necessità di definire con chiarezza le gerarchie informative, le relazioni tra fatti e dimensioni, e il ruolo dei diversi livelli architetturali. Le differenze riguardano soprattutto il livello fisico e l'organizzazione delle risorse: nei sistemi *cloud-native* il carico amministrativo è trasferito quasi completamente al provider, mentre l'organizzazione può concentrarsi sulla progettazione dei modelli e sulla governance del dato.

In conclusione, la distinzione tra architetture *on-premise* e *cloud-native* non riguarda tanto i principi fondamentali del data warehousing, quanto le modalità con cui tali principi vengono implementati. I sistemi *on-premise* restano preferibili in contesti che richiedono pieno controllo dell'infrastruttura, specifici requisiti di sicurezza o vincoli normativi stringenti, mentre le architetture *cloud-native* risultano più adatte a organizzazioni che privilegiano flessibilità, scalabilità e riduzione dei costi operativi. Il panorama contemporaneo mostra comunque una tendenza verso approcci ibridi, in cui soluzioni cloud si integrano con sistemi esistenti, con l'obiettivo di combinare stabilità e modernizzazione.

Tabella 2.1: Confronto tra DWH *on-premise* e *cloud-native*

on-premise	***cloud-native***
Pianificazione enorme	Pianificazione non necessaria
Rigido e può portare a over-provisioning e spreco di risorse	Flessibile e cresce automaticamente quando aumenta il bisogno
Subisce rallentamenti nelle query quando il volume di dati aumenta	Non subisce rallentamenti nelle query quando il volume di dati aumenta
Non si riduce quando è sottoutilizzato	Si riduce automaticamente quando è sottoutilizzato
Il criterio per scegliere un DWH *on-premise* è avere un grande volume di dati ma non aumentare il numero di utenti	Il criterio per scegliere un DWH *cloud-native* è la capacità dinamica ed elastica di gestire sia il volume di dati sia l'aumento del numero di utenti
Non è possibile aumentare o diminuire le risorse in qualsiasi momento; sono necessarie ore o giorni per configurare hardware, software e infrastruttura, e non può gestire un numero crescente di utenti	Può scalare verso l'alto o verso il basso in qualsiasi momento; non richiede ore o giorni per configurare hardware, software e infrastruttura e gestisce facilmente un numero crescente di utenti
Costoso e non semplice da scalare quando i dati aumentano	Facile ed economico da dimensionare e scalare
Non può gestire tipologie di dati diverse, processa solo dati strutturati	È in grado di gestire tipologie di dati diverse, strutturati e semi-strutturati
Obbliga ad acquistare insieme capacità di calcolo e capacità di storage	Permette di ridimensionare separatamente i cluster di calcolo

2.3 Modellazione operativa e multidimensionale: fatti, dimensioni, snapshot e SCD

Nel contesto della modellazione dimensionale, la scelta tra *star schema* e *Snowflake schema* riveste un ruolo centrale, poiché determina il grado di normalizzazione dei dati e, di conseguenza, l'efficienza delle analisi e delle operazioni di interrogazione.

Lo *star schema* rappresenta il modello multidimensionale nella sua forma più denormalizzata: una tabella dei fatti al centro e un insieme di tabelle dimensionali direttamente collegate ad essa, ciascuna contenente attributi descrittivi organizzati in un'unica struttura. Tale configurazione facilita la comprensione del modello, semplifica la scrittura delle query e migliora le performance delle operazioni di analisi, poiché minimizza la necessità di join tra tabelle. La struttura lineare delle dimensioni rende inoltre particolarmente efficiente l'esecuzione di query OLAP, che beneficiano della riduzione della complessità relazionale.

Lo *Snowflake schema*, al contrario, si basa su un maggiore livello di normalizzazione, nel quale le dimensioni vengono scomposte in più tabelle collegate tra loro, ciascuna delle quali rappresenta un livello della gerarchia di attributi. Tale approccio consente di ridurre la ridondanza dei dati e di migliorare l'integrità informativa, poiché gli attributi condivisi tra più dimensioni vengono mantenuti in un unico punto del modello, tuttavia, l'aumento del numero di tabelle e la conseguente necessità di join multipli può avere un impatto negativo sulle prestazioni delle query analitiche, rendendo così lo schema meno adatto a scenari caratterizzati da volumi elevati di interrogazioni complesse.

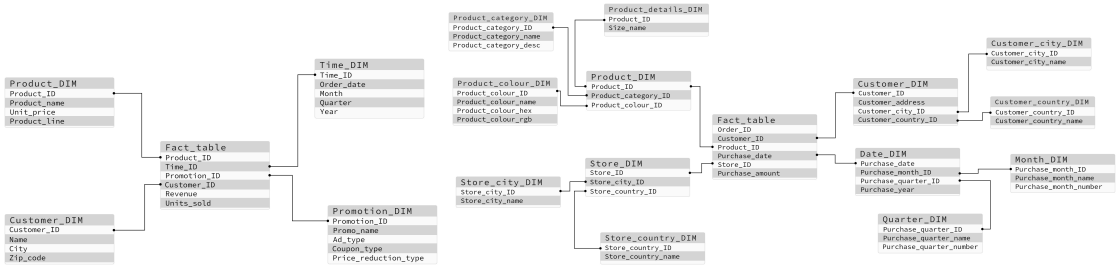

Figura 2.4: Esempio di *star schema* (sinistra) e *Snowflake schema* (destra)

La letteratura comparativa evidenzia come i due modelli rispondano a esigenze differenti. Studi di analisi strutturale mostrano che lo *star schema* è generalmente preferibile quando la semplicità, la velocità di interrogazione e l'intuitività del modello rappresentano vincoli prioritari [10], mentre lo *Snowflake schema* trova maggiore applicazione in contesti in cui la riduzione della ridondanza e la normalizzazione delle gerarchie assumono importanza strategica. In particolare, in ambienti

caratterizzati da un'elevata variabilità delle dimensioni e da una necessità di aggiornamento frequente delle gerarchie, la normalizzazione offerta dallo *Snowflake schema* può facilitare la manutenzione e l'evoluzione del modello.

Nonostante le differenze strutturali, entrambi gli schemi si collocano all'interno dello stesso paradigma multidimensionale e condividono l'obiettivo di supportare in modo efficace le attività decisionali. Diversi ricercatori del campo sottolineano che la scelta tra i due modelli dipende meno da considerazioni teoriche e più da vincoli operativi, strumenti di interrogazione utilizzati e caratteristiche del contesto aziendale; le soluzioni moderne tendono inoltre a combinare elementi di entrambi gli approcci, adottando modelli ibridi che ottimizzano sia la qualità dei dati sia le performance analitiche [11], a testimonianza della maturità delle pratiche di data warehousing contemporanee, nelle quali la modellazione non è più guidata da dogmi architetturali, ma da esigenze di business e criteri di efficienza.

La modellazione dimensionale rappresenta uno dei pilastri fondamentali del data warehousing e costituisce il punto di incontro tra i requisiti decisionali e la struttura logica del database analitico. Ralph Kimball e Margy Ross [12] nel 2013 già definirono la modellazione dimensionale come un approccio volto a organizzare i dati secondo una logica che rifletta i processi aziendali, strutturando le informazioni attraverso tabelle dei fatti e tabelle dimensionali, concepite per ottimizzare sia la comprensibilità da parte degli utenti sia le performance delle interrogazioni analitiche; lo schema multidimensionale consente infatti di rappresentare fenomeni complessi rendendoli navigabili lungo diverse prospettive decisionali, mantenendo al contempo un modello semplice, stabile e orientato alla business analysis.

Le tabelle dei fatti costituiscono il nucleo del modello e rappresentano gli eventi misurabili di un processo, definiti a un livello di granularità specifico. La scelta della granularità del fatto, ossia del livello minimo di dettaglio a cui l'informazione viene registrata, rappresenta una scelta progettuale cruciale, poiché influenza direttamente la possibilità di aggregare, analizzare e derivare misure lungo le diverse dimensioni del modello.

Nella pratica, viene spesso operata una distinzione tra fatti transazionali, fatti snapshot e fatti accumulativi: i primi catturano eventi puntuali, i secondi forniscono una fotografia periodica dello stato del sistema e i terzi registrano l'evoluzione di processi che si sviluppano nel tempo, aggiornandosi fino al loro completamento. Questa distinzione consente di adattare la modellazione al comportamento specifico del processo aziendale osservato. Parallelamente, le dimensioni rappresentano gli assi di analisi e forniscono il contesto descrittivo delle misure presenti nei fatti. Esse includono attributi quali date, clienti, prodotti, sedi e categorie gerarchiche, organizzate in strutture che consentono di navigare attraverso diversi livelli di aggregazione.

La letteratura riconosce l'importanza delle gerarchie dimensionali nel supportare operazioni OLAP quali *roll-up* e *drill-down*, e ne analizza la complessità attraverso

modelli che gestiscono gerarchie incomplete, non bilanciate o irregolari. La flessibilità delle dimensioni è inoltre garantita da soluzioni come le SCD, proposte in modo sistematico da Kimball e Ross [12], che permettono di gestire l'evoluzione dei valori descrittivi mantenendo coerenza storica e tracciabilità delle modifiche.

Tabella 2.2: Confronto tra i principali tipi di Slowly Changing Dimension (SCD)

SCD Type	**Descrizione**
Type 0 (*Fixed dimension*)	I valori della dimensione restano fissi e non vengono aggiornati.
Type 1 (*Overwrite*)	I valori vengono sovrascritti con quelli nuovi, senza conservare lo storico delle versioni precedenti.
Type 2 (*Add New Row*)	Ogni modifica genera una nuova riga nella dimensione, mantenendo lo storico completo tipicamente attraverso l'utilizzo di surrogate key e campi di validità/versione.
Type 3 (*Add New Attribute*)	La modifica viene gestita aggiungendo nuovi attributi/colonne (ad es. *current_value*, *previous_value*), mantenendo lo storico limitato a poche versioni.
Type 4 (*History Table*)	La tabella principale contiene solo lo stato corrente, mentre lo storico viene mantenuto in una tabella separata.

L'approfondimento teorico proposto da Inmon [3] a metà degli anni Duemila contribuisce a chiarire il legame tra modellazione dimensionale e storicizzazione del dato, evidenziando come la separazione tra dati operativi e dati analitici richieda un disaccoppiamento fisico e logico che consenta di mantenere la persistenza delle informazioni e di ricostruire gli stati precedenti del sistema. Questa prospettiva si integra con il lavoro di Kimball e Ross [4], che individua nelle caratteristiche delle misure (additive, semi-additive e non additive) un aspetto cruciale per la progettazione dei processi di aggregazione, determinando il tipo di analisi che può essere effettuata nelle diverse gerarchie dimensionali.

Un ulteriore elemento centrale della modellazione dimensionale è rappresentato dalla gestione delle relazioni molti-a-molti tra fatti e dimensioni, che viene affrontata attraverso l'introduzione delle dimensioni bridge. Kimball e Ross mostrano come tali strutture consentano di rappresentare correttamente scenari in cui un singolo evento

è associato a più elementi dimensionali, preservando al contempo la semplicità dello schema e la correttezza delle aggregazioni. Le dimensioni bridge assumono un ruolo particolarmente rilevante nei contesti in cui le gerarchie sono multivalore o variabili nel tempo, e si integrano con le soluzioni proposte da altri studi per la rappresentazione concettuale delle dipendenze complesse.

Nel complesso, quindi, la modellazione dimensionale emerge come un approccio maturo e consolidato nella progettazione dei sistemi di DWH, in cui le strutture dei fatti, delle dimensioni, degli snapshot e delle bridge permettono di modellare fenomeni aziendali complessi in modo coerente, scalabile e orientato all'analisi. La letteratura evidenzia come l'efficacia di questo modello risieda nella sua capacità di coniugare rigore metodologico, semplicità operativa e stabilità nel tempo, rendendolo tuttora uno degli strumenti più diffusi e apprezzati nella costruzione di architetture analitiche moderne.

2.4 ETL ed ELT: evoluzione delle pipeline di integrazione dati

L'integrazione dei dati costituisce uno dei pilastri fondamentali dei sistemi di BI e DWH. Senza un processo strutturato che governi il flusso informativo dalle fonti operative ai livelli analitici, nessun sistema di reporting o analisi avanzata sarebbe in grado di offrire risultati coerenti, tracciabili e stabili nel tempo. In questo contesto si collocano le pipeline ETL e, più recentemente, i processi ELT, che rappresentano due paradigmi differenti per la gestione delle fasi di estrazione, trasformazione e caricamento dei dati [13, 14, 15]. La letteratura scientifica e industriale ha dedicato particolare attenzione a tali processi, considerandoli un elemento chiave per garantire qualità, integrità e fruibilità del patrimonio informativo aziendale.

L'ETL costituisce il modello tradizionale dei flussi di integrazione; in questa configurazione i dati vengono estratti dalle sorgenti, trasferiti in un ambiente intermedio e trasformati tramite logiche applicative definite all'interno dello strumento di integrazione [13]. Solo dopo la completa trasformazione vengono caricati nelle strutture dimensionali, nelle tabelle dei fatti o nei data mart. Kimball e Ross [4] descrivono l'ETL come l'area di preparazione e integrazione del DWH: un insieme di procedure non visibili all'utente finale, ma essenziali per garantire che le informazioni esposte nei livelli analitici siano pulite, consistenti e aggiornate.

La prima fase del processo è rappresentata dallo *staging*, un ambiente isolato in cui i dati vengono trasferiti senza trasformazioni invasive. In quest'area si applicano controlli preliminari di coerenza, si individuano eventuali anomalie e si prepara il dato per le fasi successive.

Dal punto di vista teorico, le pipeline ETL sono considerate componenti ad alta complessità all'interno dei sistemi informazionali. Panos Vassiliadis [16], in

un'analisi del 2009, le definisce come "uno degli artefatti software più complessi nei sistemi informazionali moderni", sottolineando quanto la loro progettazione richieda un livello significativo di rigore e controllo. Tale complessità deriva da molteplici fattori: eterogeneità delle sorgenti, necessità di armonizzare schemi differenti, frequenze di aggiornamento variabili, vincoli di qualità e tracciamento storico, oltre alla necessità di garantire performance adeguate in ambienti con volumi di dati crescenti. A tutto ciò si aggiunge la necessità di trovare un equilibrio tra esigenze spesso contrastanti: stabilità delle pipeline, robustezza dei controlli, efficienza dell'elaborazione e minimizzazione delle operazioni ridondanti, rendendo indispensabile un approccio metodologico rigoroso alla progettazione dei flussi [13].

Nel paradigma ETL, il fulcro del processo risiede nella trasformazione. Le regole di conversione, standardizzazione, deduplicazione, derivazione e aggregazione vengono tipicamente eseguite da un motore interno allo strumento ETL, il quale genera e orchestra le operazioni necessarie per ottenere dati coerenti con gli standard del DWH [13]. Questo modello, consolidatosi nel corso di diversi decenni, è diventato sinonimo stesso di integrazione dei dati; tuttavia, con la crescita esponenziale dei volumi informativi, l'aumento delle frequenze di aggiornamento e l'introduzione di nuove architetture di calcolo, sono emersi limiti strutturali del paradigma tradizionale.

Uno dei limiti più evidenti del paradigma ETL tradizionale risiede nel fatto che gran parte delle trasformazioni avviene al di fuori del DBMS di destinazione, comportando che il sistema di integrazione debba disporre di risorse computazionali sufficienti a eseguire operazioni complesse e che la rete debba sostenere il trasferimento di quantità significative di dati tra strumenti ETL e database. Nei contesti moderni, caratterizzati da elevata variabilità dei carichi e da volumi sempre più ingenti, questa architettura può diventare rapidamente un collo di bottiglia. La maggiore criticità risiede nel fatto che il database, tipicamente ottimizzato per calcoli paralleli, join complessi e operazioni massivamente distribuite, rimane inutilizzato proprio nelle fasi più onerose del processo di integrazione.

Queste limitazioni hanno favorito l'emergere del paradigma ELT (Extract, Load, Transform), il quale ribalta completamente l'ordine delle operazioni [16]. Nel modello ELT, i dati vengono estratti e caricati immediatamente nel sistema di destinazione, senza trasformazioni preliminari. Solo dopo aver raggiunto l'ambiente finale, essi vengono elaborati tramite query SQL, procedure interne o motori ottimizzati per l'analisi, come sistemi MPP o piattaforme *cloud-native*. Diversi autori hanno evidenziato come l'avvento dell'ELT rappresenti un cambiamento profondo nell'ingegneria dei dati, poiché sposta il baricentro delle elaborazioni verso il livello dati e sfrutta in modo nativo le capacità computazionali dell'infrastruttura di destinazione.

Il passaggio all'ELT risponde a esigenze sia tecniche sia architetturali, infatti,

dal punto di vista tecnico, il database moderno è spesso in grado di gestire operazioni complesse in tempi significativamente inferiori rispetto a un motore ETL esterno, grazie a ottimizzatori SQL avanzati, capacità di parallelismo e architetture distribuite; dal punto di vista architetturale invece, l'ELT si integra perfettamente con i DWH *cloud-native*, come BigQuery, Snowflake o Redshift, che prevedono una separazione tra le risorse di archiviazione da quelle di elaborazione, consentendo di scalare dinamicamente solo la componente necessaria ottimizzando così costi e prestazioni.

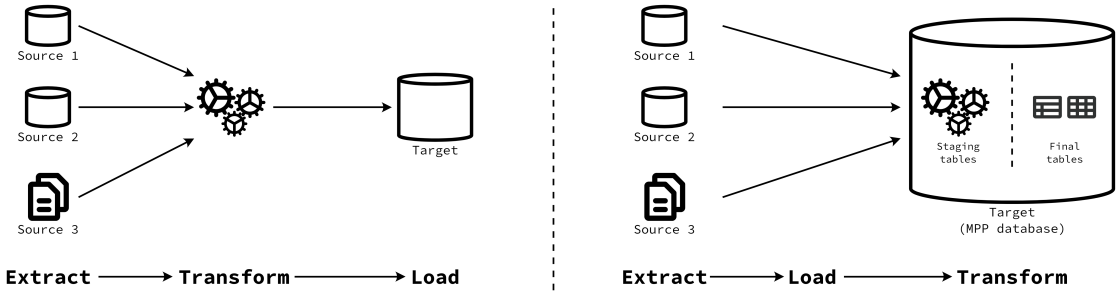

Figura 2.5: ETL vs. ELT

L'evoluzione dalle pipeline ETL a quelle ELT riflette un cambiamento più generale nella progettazione dei sistemi analitici, infatti nei modelli tradizionali l'attenzione era focalizzata sullo strumento di integrazione, mentre nelle architetture moderne il database assume un ruolo centrale e lo strumento di integrazione diventa principalmente un orchestratore: gestisce workflow, dipendenze, scheduling e monitoraggio, ma non esegue direttamente le operazioni più onerose; ciò comporta un cambiamento significativo anche nella gestione della qualità del dato, nella tracciabilità delle trasformazioni e nella gestione degli errori, che devono essere progettati per operare direttamente sul DBMS [16].

Un ulteriore elemento che distingue ETL ed ELT riguarda la modalità di gestione delle latenze informative: nei modelli basati su ETL, la finestra di elaborazione può risultare più ampia, poiché il dato deve attraversare diversi passaggi prima di raggiungere il DWH, mentre nei modelli ELT, al contrario, la possibilità di eseguire trasformazioni direttamente sul dato caricato consente in molti casi di ridurre le latenze e di realizzare integrazioni quasi in tempo reale, specialmente in presenza di architetture cloud elastiche; questo rafforza il ruolo dell'ELT nei sistemi moderni di data warehousing, dove la frequenza di aggiornamento rappresenta un requisito sempre più critico.

Nonostante l'evoluzione tecnologica, la letteratura sottolinea che i principi concettuali alla base dei flussi di integrazione rimangono invariati, di fatto rimane necessario garantire: qualità, consistenza, completezza e tracciabilità, gestire la storicizzazione dei dati, preservare le relazioni tra entità, armonizzare codifiche e

nomenclature provenienti da sorgenti eterogenee e assicurare la riproducibilità del processo. Sia l'ETL sia l'ELT, se progettati correttamente, costituiscono strumenti essenziali per raggiungere tali obiettivi.

In sintesi, l'evoluzione delle pipeline di integrazione rappresenta un percorso che riflette l'evoluzione stessa delle architetture dati nel quale l'ETL tradizionale rimane un modello utile in scenari applicativi o in contesti tecnologicamente omogenei, mentre l'ELT si afferma come paradigma preferenziale nei sistemi analitici moderni, specialmente quelli *cloud-native*. La comprensione delle differenze tra i due approcci, delle loro implicazioni progettuali e dei loro ambiti di applicazione costituisce un prerequisito fondamentale per affrontare in modo consapevole l'analisi degli strumenti di integrazione e dell'architettura multilivello adottata nel progetto trattato in questa tesi.

2.5 Strumenti di integrazione e architetture multilivello: SSIS, iPaaS e Oracle Data Integrator (ODI)

L'evoluzione delle pipeline di integrazione ha portato allo sviluppo di un ecosistema eterogeneo di strumenti e soluzioni, ciascuno pensato per affrontare specifiche esigenze architetturali, organizzative e tecnologiche. In ambito aziendale, la scelta dello strumento di integrazione non rappresenta una semplice decisione tecnica, ma incide in modo diretto e significativo sulle modalità di circolazione dell'informazione, sulla governance e sulla qualità del dato, nonché sulla capacità del sistema di BI di crescere e adattarsi nel tempo. Le soluzioni disponibili sul mercato si articolano in tre grandi categorie: strumenti ETL tradizionali, piattaforme di integrazione *cloud-native* (iPaaS) e framework ibridi basati su logica ELT, tra cui ODI, che assume un ruolo centrale nel progetto descritto in questa tesi.

Gli strumenti ETL tradizionali rappresentano la prima generazione di piattaforme dedicate all'integrazione dei dati e, tra questi, SSIS costituisce uno degli esempi più diffusi in contesti Microsoft. SSIS è concepito per orchestrare pipeline grafiche basate su data flow, all'interno delle quali si eseguono estrazioni, trasformazioni e caricamenti tramite un motore interno indipendente dal database di destinazione, risultando particolarmente efficace per estrazioni puntuali, sincronizzazioni periodiche e scenari applicativi *on-premise*. Tuttavia, presenta un limite strutturale: la maggior parte delle trasformazioni viene eseguita in memoria o mediante componenti dedicati, senza sfruttare pienamente la potenza computazionale del DBMS sottostante, aspetto che può diventare critico nel trattamento di volumi di dati elevati o in contesti che richiedono aggiornamenti frequenti. La letteratura di settore e i whitepaper Microsoft mettono in evidenza come SSIS sia una soluzione

solida ma orientata a scenari ETL classici, meno adatta a sistemi di analisi moderni basati su paradigmi ELT o architetture *cloud-native* [17, 18].

Una seconda categoria di strumenti è rappresentata dalle piattaforme di tipo iPaaS. Queste soluzioni, come evidenziato dalla letteratura accademica più recente, nascono per rispondere ad esigenze di integrazione distribuita, orchestrazione di API e sincronizzazione tra applicazioni cloud e *on-premise*. Gli iPaaS si basano su connettori preconfigurati, workflow visuali e meccanismi di governance centralizzata che semplificano processi complessi quali la gestione delle API REST, la mappatura dei dati e l'automazione dei flussi informativi tra sistemi eterogenei. Tali sistemi risultano molto efficaci in scenari di integrazione applicativa, ambienti multicloud e processi di *data synchronization*; tuttavia, la letteratura evidenzia come mostrino limitazioni significative nel contesto dei DWH tradizionali, in particolare, esse difficilmente supportano in modo nativo funzionalità avanzate quali la gestione delle SCD, la storicizzazione delle tabelle dei fatti, la gestione delle chiavi surrogate o la costruzione di livelli intermedi quali ODS e MDM. Inoltre, l'elaborazione di query complesse o trasformazioni massivamente parallele risulta generalmente meno efficiente rispetto a quanto garantito dai motori di elaborazione dei DMS ottimizzati per carichi analitici [19].

Accanto a queste categorie si collocano gli strumenti ibridi basati sul paradigma ELT, tra i quali ODI, che rappresenta una delle soluzioni più mature e diffuse in ambito enterprise; ODI adotta una filosofia progettuale profondamente diversa rispetto agli strumenti ETL tradizionali: la logica di trasformazione non viene eseguita all'interno del tool, ma è demandata al database di destinazione, sfruttandone direttamente le capacità di elaborazione. In questo contesto, ODI assume il ruolo di orchestratore intelligente, responsabile della generazione automatica di codice SQL ottimizzato e specifico per la piattaforma sottostante. La documentazione ufficiale Oracle descrive ODI come un sistema fondato su un approccio dichiarativo, in cui lo sviluppatore si limita a definire il risultato atteso, mentre il motore di integrazione determina autonomamente le modalità più efficienti per ottenerlo, inoltre, l'utilizzo dei KM consente di configurare e standardizzare operazioni complesse quali i caricamenti incrementali, la gestione delle chiavi surrogate, il rilevamento delle anomalie e la propagazione dei delta, favorendo riusabilità, governance e coerenza progettuale [20].

Queste caratteristiche rendono ODI particolarmente adatto all'implementazione di architetture di integrazione multilivello, una pratica ampiamente diffusa nei sistemi informativi aziendali reali. Nel progetto oggetto di questa tesi, ODI è stato utilizzato per costruire un framework di integrazione basato su tre livelli principali denominati L0, L1 e L2. Tale framework è stato progettato e progressivamente consolidato internamente da Var Group - Data Science Operations sulla base dell'esperienza maturata in diversi contesti progettuali, con l'obiettivo di strutturare in modo chiaro e ripetibile i processi di integrazione dei dati. Per questo motivo, i

livelli L0, L1 ed L2 non fanno riferimento ad un modello teorico formalizzato nella letteratura accademica, ma rappresentano una best practice interna all'azienda, adottata per garantire modularità, tracciabilità e manutenibilità dei flussi nel tempo; la loro struttura e le responsabilità specifiche di ciascun livello saranno descritte in dettaglio nel Capitolo 3.

Nel complesso, l'analisi comparativa degli strumenti e la descrizione dell'architettura multilivello adottata mostrano come il panorama dell'integrazione dati sia oggi caratterizzato da una crescente convergenza tra orchestrazione, governance, scalabilità e automazione, all'interno della quale la scelta dello strumento e la definizione del framework architetturale assumono un ruolo centrale nel garantire che i flussi di integrazione risultino solidi, adattabili e pienamente allineati alle esigenze analitiche dell'organizzazione.

2.6 Data integration da sistemi eterogenei: database, file e API REST

L'integrazione dei dati provenienti da fonti eterogenee rappresenta una delle sfide più complesse e rilevanti nella costruzione di sistemi informativi analitici moderni, in quanto le organizzazioni contemporanee raccolgono e generano informazioni attraverso applicazioni operative, strumenti gestionali, servizi web e dispositivi distribuiti, dando origine a un ecosistema di dati caratterizzato da formati, strutture, frequenze di aggiornamento e modelli informativi profondamente diversi tra loro [15, 14]. In questo contesto emerge la necessità di adottare architetture e metodologie di integrazione in grado di garantire coerenza, completezza e affidabilità del patrimonio informativo indipendentemente dalla natura e dall'origine delle singole sorgenti.

La letteratura scientifica evidenzia come l'eterogeneità delle sorgenti informative possa manifestarsi secondo differenti dimensioni, in particolare viene comunemente distinta un'eterogeneità di tipo strutturale, legata alle differenze nei modelli dei dati e nelle rappresentazioni sintattiche e un'eterogeneità di tipo semantico, che riguarda invece la coerenza dei significati, delle codifiche e delle regole di business associate agli attributi provenienti da sistemi diversi. Nel 2020, Giuseppe Fusco e Lerina Aversano [21] sottolineano come tale distinzione sia centrale nei processi di integrazione, poiché le attività di mappatura, trasformazione e riconciliazione devono affrontare congiuntamente sia i disallineamenti di natura tecnica sia le differenze concettuali e interpretative tra le sorgenti.

La complessità dell'integrazione aumenta ulteriormente quando i sistemi coinvolti appartengono a livelli organizzativi differenti o operano secondo logiche funzionali autonome, poiché l'integrazione eterogenea attraversa diversi livelli di un sistema aziendale, spaziando dai contesti più prossimi al controllo operativo fino ai sistemi gestionali e analitici di supporto alle decisioni. Come evidenziato da Horák e i suoi

collaboratori [22], l'integrazione non può essere ricondotta esclusivamente ad un problema di formati o protocolli di comunicazione, ma deve essere interpretata come un'attività che richiede una comprensione approfondita dei vincoli logici, temporali e operativi che caratterizzano ciascun livello della catena informativa.

Il tema dell'integrazione tramite web service e API REST assume un ruolo di particolare rilevanza nei sistemi informativi contemporanei, poiché la crescente diffusione di applicazioni cloud-based e di architetture orientate ai servizi ha profondamente trasformato le modalità con cui i dati vengono esposti, scambiati e consumati. Muhammad Intizar Ali [23] nel 2011 analizza l'integrazione di dati provenienti da sorgenti web distribuite, mettendo in luce specifiche criticità legate all'assenza di formati unificati e alla variabilità delle strutture semi-strutturate, tipicamente rappresentate in JSON o XML, rendendo necessari approcci dedicati di interpretazione e trasformazione dei dati. La natura semi-strutturata di tali flussi introduce infatti sfide tipiche del web data integration, quali la gestione di schemi impliciti, l'evoluzione dinamica delle API, la necessità di strumenti di analisi robusti e la riconciliazione di informazioni ottenute tramite invocazioni asincrone o attraverso endpoint multipli. Questa prospettiva risulta particolarmente significativa nei contesti in cui una o più sorgenti informative espongono i dati esclusivamente tramite API REST, in assenza di strutture relazionali tradizionali.

Le fonti accademiche sottolineano inoltre come l'integrazione da file costituisca una parte essenziale dei flussi informativi eterogenei, poiché formati quali CSV, Excel o JSON costituiscono spesso output generati da applicazioni legacy, strumenti operativi o sistemi che non espongono interfacce programmatiche strutturate. Paraskevas Koukaras [24], in uno studio pubblicato nel 2025, ha osservato che tali file presentano livelli di struttura variabili e richiedono l'adozione di procedure di *schema inference*, *data profiling* e validazione semantica al fine di garantire una corretta interpretazione del contenuto informativo. L'integrazione da file non può pertanto essere considerata un aspetto residuale, ma un elemento stabile e ricorrente nei processi informativi aziendali, soprattutto nei contesti in cui i sistemi non dispongono di meccanismi di interoperabilità moderni o quando vengono utilizzati formati di interscambio manuale per evitare forme di accoppiamento diretto tra applicazioni.

Uno dei contributi più rilevanti emersi dalla letteratura riguarda il ruolo centrale della semantica nella risoluzione delle eterogeneità informative. Ancora Fusco e Aversano [21] evidenziano come le problematiche di integrazione non possano essere risolte mediante la sola uniformazione dei formati o delle strutture sintattiche, poiché il nodo principale risiede nei significati attribuiti agli stessi attributi nei diversi sistemi. L'unificazione dei modelli informativi richiede pertanto l'adozione di tecniche di schema matching, l'impiego di ontologie condivise e l'applicazione di processi di entity resolution finalizzati ad identificare e riconciliare entità rappresentate in modo differente all'interno di sorgenti multiple, un passaggio fondamentale

quando vengono integrate applicazioni che trattano informazioni simili secondo terminologie, codifiche o tassonomie non allineate, come avviene frequentemente nel caso di strumenti di gestione operativa eterogenei o di applicazioni cloud-based sviluppate in modo indipendente.

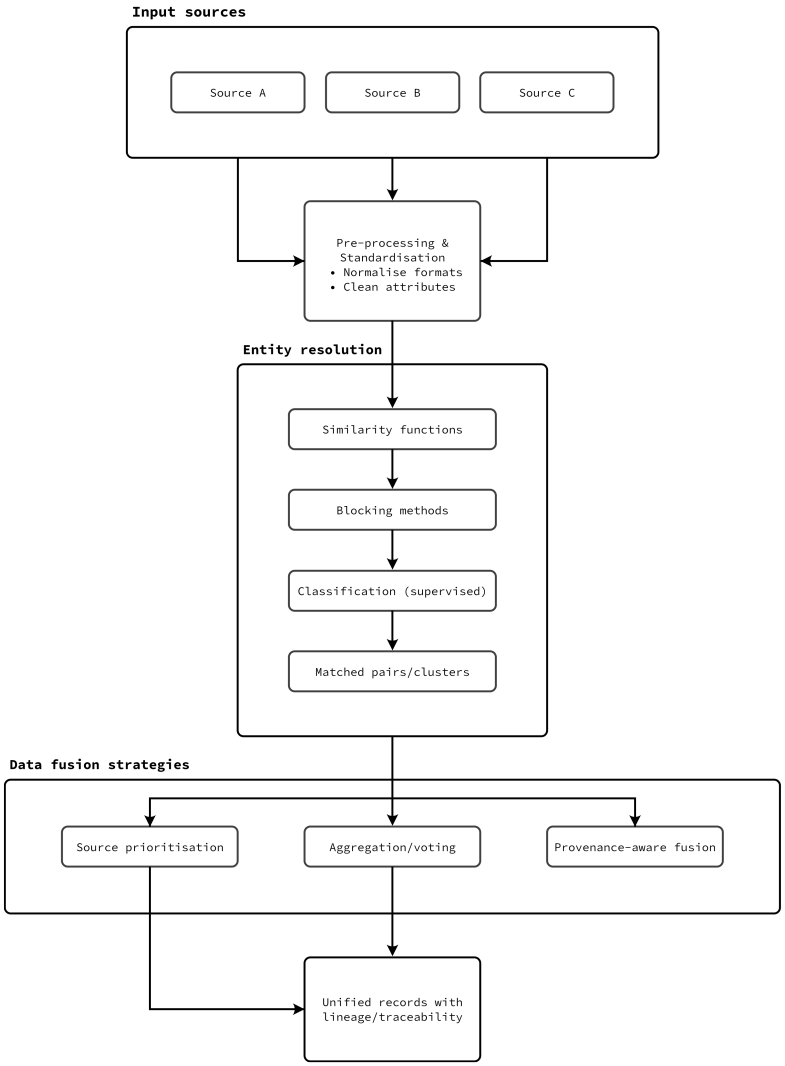

Figura 2.6: Fasi di *pre-processing*, *entity resolution* e fusione dei dati da sorgenti multiple.

Un aspetto spesso trascurato nei processi di integrazione è la necessità di considerare il livello di controllo da cui i dati hanno origine; l'integrazione tra sistemi che operano su livelli differenti, come quello operativo, gestionale e analitico, comporta delle differenze significative in termini di granularità temporale, frequenza di aggiornamento, significato dei valori e stabilità degli attributi. I dati provenienti

dai sistemi operativi risultano generalmente più granulari, variabili e orientati all'esecuzione delle attività, mentre quelli generati dai sistemi gestionali tendono ad essere maggiormente aggregati, strutturati e finalizzati alla pianificazione, ai quali si affiancano i dati esposti tramite API applicative, che possono seguire logiche proprie e non necessariamente allineate ai modelli relazionali o analitici tradizionali. Tale eterogeneità rende essenziale la definizione di un modello di integrazione in grado di gestire esplicitamente queste differenze, garantendo che le informazioni provenienti da sorgenti diverse possano essere confrontate, aggregate e utilizzate in modo coerente all'interno dei sistemi di analisi.

Un ulteriore contributo rilevante emerso dalla letteratura riguarda la distinzione che Fusco e Aversano [21] propongono tra modelli di integrazione basati su data warehousing e modelli fondati su approcci di tipo federato, nell'ambito dei quali l'approccio DWH presuppone la centralizzazione dei dati a seguito delle attività di trasformazione e integrazione, mentre l'integrazione federata mantiene le sorgenti distribuite, demandando le operazioni di trasformazione e conciliazione al momento dell'esecuzione delle query. Sebbene l'approccio federato risulti particolarmente adatto a scenari dinamici o a contesti nei quali non è possibile effettuare estrazioni massive dei dati, la maggior parte dei sistemi informativi aziendali continua a privilegiare soluzioni basate su DWH. Le architetture più recenti tendono tuttavia a combinare elementi di entrambi i modelli, dando origine a soluzioni ibride come Data Lake, Lakehouse e piattaforme integrate, in grado di supportare sia l'integrazione centralizzata sia la gestione di flussi semi-strutturati.

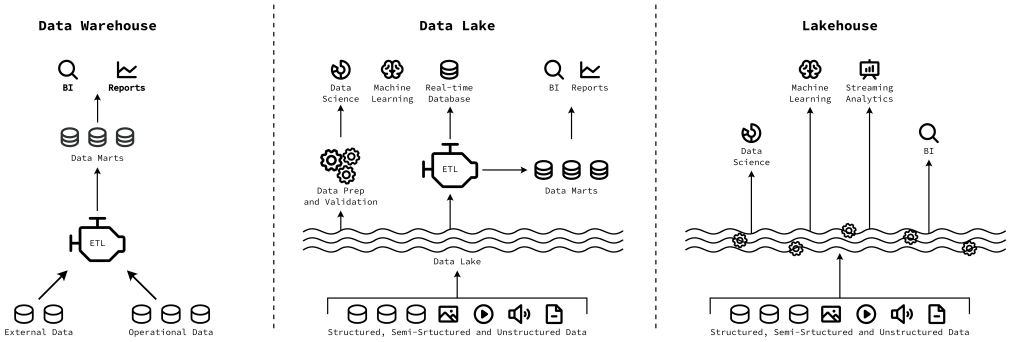

Figura 2.7: Confronto concettuale tra architetture di DWH, Data Lake e Lakehouse.

In questo quadro teorico, l'integrazione di dati provenienti da sistemi eterogenei quali database relazionali, file semi-strutturati e API REST non rappresenta un'eccezione, ma costituisce la condizione ordinaria dei sistemi informativi moderni, nei quali convivono strumenti di ticketing, sistemi di asset management e piattaforme di gestione dei progetti basati su modelli informativi e strutture differenti. Le analisi presenti in letteratura mostrano come la capacità di gestire formati diversi,

modelli semantici eterogenei e livelli logici distinti rappresenti una competenza fondamentale per garantire la coerenza del patrimonio informativo e la sua effettiva disponibilità a supporto dei processi decisionali; in tale contesto risulta evidente come l'integrazione dei dati da sistemi eterogenei richieda una combinazione di tecniche, modelli e competenze che va ben oltre la semplice trasformazione dei formati, poiché le differenze strutturali, semantiche e funzionali tra database, file e API REST rendono necessaria una progettazione attenta dei processi di integrazione, fondata su principi solidi e su una profonda comprensione delle caratteristiche delle sorgenti. Si può quindi arrivare alla conclusione che la gestione efficace di tali complessità costituisce un prerequisito essenziale per la costruzione di architetture analitiche stabili, scalabili e affidabili.

2.7 Best practice per la costruzione di un framework ETL/ELT moderno

La progettazione di un framework ETL/ELT moderno richiede l'interazione di principi metodologici consolidati e l'adozione di tecniche avanzate di gestione e orchestrazione dei dati, soprattutto alla luce della crescente eterogeneità delle sorgenti informative, dell'aumento dei volumi trattati e della diffusione di architetture cloud e ibride, fattori che hanno reso necessario ripensare i tradizionali processi di acquisizione dei dati, trasformazione e pubblicazione. In questo scenario si assiste al superamento di approcci statici a favore di pipeline evolutive, scalabili e governate, rispetto alle quali la letteratura più recente offre un quadro articolato di pratiche considerate ottimali per la costruzione di framework robusti, riutilizzabili e resilienti. Questa impostazione è coerente con le rassegne più recenti sui sistemi analitici eterogenei, in particolare con il contributo di Paraskevas Koukaras [24], che interpreta le pipeline ETL/ELT come componenti dinamiche di un ecosistema distribuito, progettate per adattarsi a contesti infrastrutturali e informativi in continua evoluzione.

Un contributo teorico di particolare rilievo è rappresentato dai modelli ETLT ed ELTL [25], che estendono gli approcci tradizionali introducendo una distribuzione più consapevole delle fasi di trasformazione tra sistemi di *staging*, database analitici e motori di orchestrazione; tali modelli mettono in evidenza come le pipeline moderne non possano più essere interpretate come semplici sequenze lineari di operazioni, ma debbano essere concepite come architetture adattative, in grado di sfruttare risorse computazionali eterogenee e di integrare controlli di qualità, arricchimenti semantici e meccanismi di validazione lungo l'intero ciclo di vita del dato; questa visione trova un fondamento teorico negli studi di Xiufeng Liu [26] sull'ottimizzazione dei dataflow ETL, nei quali l'autore dimostra come la riorganizzazione delle trasformazioni e il loro spostamento verso le fasi post-load

consentano di migliorare sia l'efficienza sia la flessibilità delle pipeline; da ciò si può assumere il ruolo centrale che l'introduzione di trasformazioni post-load può ricoprire poiché consentono di ricalcolare porzioni di dato con granularità variabile, migliorando sia la resilienza complessiva del sistema sia la sua capacità di adattamento ai cambiamenti delle sorgenti informative.

L'evoluzione verso architetture cloud-based ha ulteriormente influenzato la definizione delle best practices in ambito ETL/ELT, favorendo una più netta separazione tra i livelli di acquisizione dei dati e quelli di trasformazione, insieme all'adozione di meccanismi di scalabilità elastica e di gestione dinamica del throughput. Le analisi comparative sui framework Big Data condotte da Wissem Inoubli e collaboratori [27], successivamente sistematizzate da Koukaras [24], evidenziano come tale separazione rappresenti una condizione abilitante per la gestione efficiente di carichi eterogenei e distribuiti.In questo scenario, i sistemi di acquisizione dei dati moderni sono progettati per supportare modalità asincrone e parallele di acquisizione dei dati, rendendo possibile l'integrazione efficiente di sorgenti eterogenee quali database transazionali, file system distribuiti, flussi streaming e servizi API, mantenendo contemporaneamente coerenza e ripetibilità delle operazioni di caricamento; la definizione di un'acquisizione dei dati layer autonomo assume quindi, alla luce delle riflessioni appena fatte, un ruolo centrale poiché consente di isolare la variabilità dei formati e delle frequenza di aggiornamento rispetto ai moduli deputati alla trasformazione, creando al contempo un punto di controllo centralizzato per la gestione di errori, ritardi e anomalie nella disponibilità dei dati [28].

Un ulteriore ambito di rilevanza riguarda l'ottimizzazione delle prestazioni nei processi ETL/ELT, aspetto che la letteratura riconduce principalmente alle scelte architetturali piuttosto che alla sola potenza computazionale o alla tecnologia adottata. L'efficienza delle pipeline dipende infatti in larga misura dall'organizzazione logica del processo di integrazione, all'interno del quale la separazione delle trasformazioni più onerose, la riduzione dei passaggi intermedi e l'utilizzo di motori SQL nativi per l'esecuzione push-down rappresentano principi fondamentali per garantire prestazioni sostenibili anche in presenza di carichi elevati, in tal senso, i risultati sperimentali presentati da Liu mostrano come l'ottimizzazione del dataflow e l'eliminazione di colli di bottiglia strutturali producano benefici più significativi rispetto al semplice incremento delle risorse hardware. Tali tecniche risultano particolarmente significative nei contesti di reingegnerizzazione di flussi legacy, nei quali pipeline storicamente monolitiche devono essere progressivamente trasformate in strutture modulari, osservabili e scalabili, in grado di adattarsi a requisiti analitici in continua evoluzione.

La gestione dell'eterogeneità e della qualità del dato costituisce uno degli elementi centrali nelle best practices moderne in ambito ETL/ELT, poiché un framework efficace non può limitarsi alla sola trasformazione dei dati, ma deve governarne

l'intero ciclo di vita, incorporando controlli sistematici di coerenza, validità, completezza e tracciabilità. Come sottolineato sia da Liu sia da Koukaras [26, 24], la qualità del dato non rappresenta un requisito accessorio, ma una proprietà emergente dell'intera architettura di integrazione, è quindi in contesti caratterizzati dalla presenza di sorgenti semanticamente e strutturalmente diverse che la riconciliazione delle informazioni richiede l'adozione di strategie fondate su metadati espliciti, registri delle entità, *Mapping* tra livelli di astrazione differenti e processi di normalizzazione progressiva, così da garantire un modello di integrazione stabile, coerente e sostenibile nel tempo.

All'interno di questo quadro, le problematiche connesse ai dati semi-strutturati e all'integrazione tramite API REST rappresentano un caso particolarmente significativo di tale complessità, già messo in evidenza da Ali [23] che, nel 2011, ha mostrato come l'integrazione di sorgenti distribuite tipiche degli ecosistemi web introduca ulteriori sfide legate alla variabilità dei formati, all'assenza di schemi rigidi, alla dipendenza dal contesto applicativo e alla necessità di meccanismi di interpretazione semantica. Le best practices in questo ambito suggeriscono pertanto l'estrazione e la standardizzazione degli schemi impliciti, la definizione di *Mapping* concettuali verso modelli operativi coerenti e l'adozione di componenti di astrazione, come *wrapper* o *adapter*, capaci di isolare le instabilità delle interfacce API, un'impostazione che trova ulteriore conferma nelle rassegne più recenti, tra cui quella di Koukaras, che sottolineano l'importanza del monitoraggio continuo e del versioning controllato delle API.

Altra dimensione fondamentale nell'ambito delle best practices moderne è rappresentata dalla governance del dato, poiché nei contesti analitici eterogenei la qualità e l'affidabilità dell'informazione dipendono in larga misura dalla presenza di un sistema strutturato di gestione dei metadati, del lineage e del versioning degli schemi. Un framework ETL/ELT moderno deve pertanto essere accompagnato da un catalogo dei metadati esplicito, capace di documentare in modo sistematico le regole di trasformazione, le dipendenze tra i diversi livelli del sistema, le politiche di storicizzazione e, come evidenziato dalle analisi di Koukaras [24] sulla manutenibilità dei sistemi analitici complessi, abilitando analisi di impatto e una gestione controllata delle modifiche evolutive; tale infrastruttura non può infatti essere considerata un elemento accessorio, ma costituisce il fondamento della manutenibilità e della scalabilità delle pipeline, poiché garantisce l'allineamento continuo del sistema alle regole di business e ne supporta l'evoluzione nel tempo.

In stretta continuità con questi aspetti, il tema della resilienza emerge come una componente essenziale delle architetture di integrazione moderne, nelle quali i framework più evoluti sono progettati per gestire automaticamente fallimenti parziali, ripetere in modo selettivo le porzioni di flusso interessate da errori e tracciare gli stati intermedi del processo di elaborazione. Le analisi di Liu [26] e le rassegne sui framework distribuiti di Inoubli [27] e i suoi collaboratori mostrano

come l'adozione di livelli intermedi multipli, quali aree di *staging*, ODS e zone pre-pubblicazione, risulti particolarmente efficaci nel garantire riproducibilità e recuperabilità, poiché consentono di catturare istantanee del dato e isolarne l'evoluzione nel tempo, in coerenza con le più recenti strategie di acquisizione dei dati cloud-based che prevedono l'impiego di buffer persistenti, sistemi di checkpoint e meccanismi di sincronizzazione distribuiti.

In conclusione, le best practice individuate nella letteratura contemporanea convergono verso una visione del framework ETL/ELT come sistema complesso, modulare e governato, nel quale l'obiettivo non è unicamente la trasformazione dei dati, ma include la garanzia di qualità del dato e la trasparenza, riproducibilità e capacità evolutiva dei processi di integrazione. Questi principi costituiscono la base teorica indispensabile per la progettazione di pipeline preformanti e sostenibili e trovano applicazione concreta nei progetti di reingegnerizzazione dei flussi di integrazione, in particolare in contesti caratterizzati da sorgenti eterogenee e dalla necessità di consolidare le informazioni in un modello operativo coerente e affidabile.

2.8 Ricerche accademiche e casi di studio sulla reingegnerizzazione dei flussi ETL

Il tema della reingegnerizzazione dei flussi ETL ha acquisito una crescente rilevanza negli ultimi anni, parallelamente all'evoluzione delle architetture analitiche e alla transizione verso sistemi sempre più flessibili, scalabili e orientati all'automazione, in contrapposizione ai processi ETL storici che tendono a manifestare limiti evidenti in termini di manutenibilità, prestazioni, capacità di integrazione con nuove sorgenti informative e livello di governance dei metadati, essendo spesso progettati e implementati in contesti tecnologici profondamente diversi da quelli attuali. La letteratura contemporanea è concorde nel riconoscere come la modernizzazione dei flussi di integrazione non rappresenti un semplice aggiornamento tecnico, ma bensì una vera e propria trasformazione metodologica che coinvolge l'intero ciclo di vita del dato, dalla fase di acquisizione fino alla pubblicazione e all'utilizzo analitico delle informazioni.

Nel 2023, Simitsis, Vassiliadis e Skiadopulos [29] conducono una revisione semantica che costituisce uno dei contributi più influenti per comprendere l'evoluzione dei processi ETL negli ultimi venticinque anni, ripercorrendo lo sviluppo delle tecniche di integrazione dati a partire dagli anni Novanta, mostrando come l'ETL sia passato progressivamente da semplici script di caricamento a sistemi sempre più articolati e complessi comprensivi di aree di *staging* dedicate, gestione della storicizzazione delle dimensioni, meccanismi di orchestrazione e monitoraggio delle esecuzioni, fino ad arrivare a funzionalità avanzate di ottimizzazione automatica delle trasformazioni. L'importanza di questa analisi risiede nella capacità di rendere

comprensibili le ragioni per cui oggi molti sistemi legacy non risultano adatti a supportare le esigenze moderne, in ragione di una logica spesso monolitica, di una scarsa modularità e di una documentazione non sempre adeguata, elementi che risultano difficilmente conciliabili con requisiti quali la scalabilità orizzontale, l'integrazione tramite API REST o la gestione di dati semi-strutturati. La necessità di reingegnerizzazione emerge quindi come una conseguenza diretta del divario strutturale tra le architetture storiche e le esigenze moderne dei sistemi analitici.

Il contributo di Alan Willie [30], e dei suoi collaboratori, del 2023 si inserisce in questa prospettiva, concentrandosi sull'ottimizzazione dei workflow ETL in ambienti caratterizzati da grandi volumi di dati e da requisiti stringenti in termini di throughput e latenza, evidenziando come i processi tradizionali risultino spesso incapaci di sostenere alti carichi e proponendo un set di strategie che includono la parallelizzazione delle trasformazioni, la riduzione dei colli di bottiglia nelle operazioni di input/output, la gestione esplicita delle dipendenze tra attività e l'impiego di meccanismi di caching condiviso. Sebbene lo studio sia focalizzato su scenari particolarmente intensivi, i principi esposti risultano applicabili anche alla reingegnerizzazione di flussi di dati aziendali di scala medio piccola, poiché le cause di inefficienza individuate, come ad esempio trasformazioni duplicate, controlli incoerenti, gestione frammentata degli errori e scarsa tracciabilità dei flussi, tendono a ripresentarsi indipendentemente dal volume dei dati trattati.

Nel 2022 Christian Haase, Timo Röseler, Mattias Seidel [31] introducono un framework METL (Message ETL), che propone un'integrazione dati progettata per combinare gestione dinamica delle mappature, CDC e componenti di *streaming* allo scopo di superare i limiti di rigidità tipici dei processi ETL tradizionali e favorire un approccio evolutivo all'integrazione dei dati. L'elemento distintivo della proposta METL risiede nella separazione esplicita tra logica di mappatura e logica di orchestrazione, una scelta architetturale che consente di modificare il comportamento dei flussi di dati senza intervenire direttamente sul codice, facilitando la manutenzione evolutiva e mitigando il rischio che le modifiche apportate compromettano componenti già operative. Tale esigenza si riflette anche nei contesti aziendali in cui l'introduzione di nuove sorgenti, come nel nostro caso Webgate tramite API REST che approfondiremo nei prossimi capitoli, accanto a sistemi esistenti, richiede architetture capaci di accogliere estensioni progressive senza compromettere le funzionalità operative.

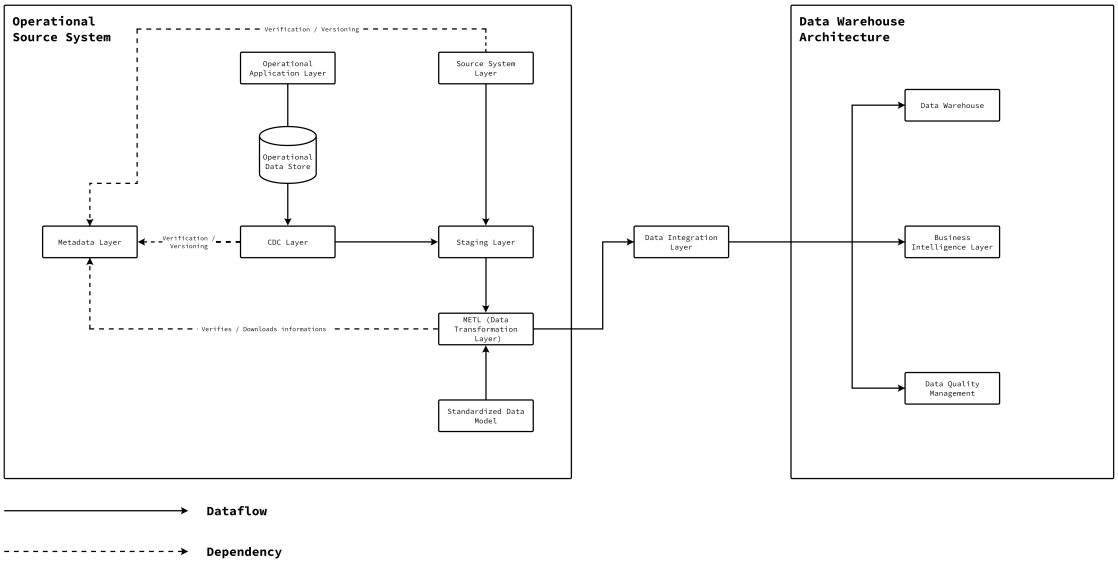

Figura 2.8: Architettura del framework METL basato su messaggi Kafka, con separazione tra sistema operativo FX e architettura analitica EOS

A completare la precedente prospettiva, Deepak Chanda [32] nel 2024 analizza in modo critico le principali debolezze dei flussi ETL tradizionali sottolineando come molti di questi flussi risultino difficili da evolvere a causa dell'assenza di uno strato di governance centralizzato, della mancanza di gestione e tracciamento delle versioni dei metadati, dell'assenza di tracciamento dello stato delle esecuzioni e della carenza di meccanismi di recupero automatico in caso di errore. Lo studio mostra come l'introduzione di pratiche moderne, quali la definizione formale dei metadati, l'adozione di regole di trasformazione dichiarative, l'implementazione di logging strutturato e l'automazione dei workflow di recupero, conduca ad un incremento significativo dell'affidabilità e della trasparenza dei processi di integrazione e ci fa intendere che il principale beneficio della reingegnerizzazione di flussi tradizionali non risieda esclusivamente nel miglioramento delle prestazioni, ma soprattutto nella possibilità di rendere i flussi prevedibili, verificabili e modificabili nel tempo.

A completare il quadro teorico presentato, il lavoro di Chiara Rucco, Motaz Saad e Antonella Longo [25] del 2025 propone una classificazione dei pattern di integrazione moderni e analizza il modo in cui i flussi più avanzati combinano trasformazioni *batch*, *micro-batch* e *real time* all'interno di strutture ibride, consentendo di interpretare la reingegnerizzazione non come una semplice riscrittura dei flussi esistenti, ma come l'adozione di un paradigma architetturale più coerente con le esigenze informative contemporanee. Gli autori sostengono che l'impiego di pattern formalizzati, come l'ETLT o l'ELTL, permetta di ottenere flussi più semplici da monitorare, più robusti nel tempo e meglio allineati ai requisiti di integrazione di sorgenti applicative esposte tramite API REST.

La letteratura che abbiamo analizzato in questo paragrafo converge tutta su un punto chiave: la reingegnerizzazione dei flussi ETL deve essere intesa come un processo strutturale di modernizzazione architetturale, e non come un semplice intervento tecnico. È stato dimostrato dai contributi più rilevanti come gli approcci incrementali, basati sulla progressiva introduzione di framework modulari, sulla separazione dei livelli concettuali e sulla gestione centralizzata dei metadati, consentono di evolvere i sistemi di integrazione senza compromettere l'operatività esistente; infine si conferma che il valore di tali interventi risiede nella capacità di rendere i flussi più trasparenti, governabili e sostenibili nel tempo, fornendo una solida base teorica per le scelte progettuali adottate nel percorso di tirocinio.

2.9 Dai principi teorici alle scelte architetturali: sintesi e transizione alla reingegnerizzazione

L'analisi dello stato dell'arte condotta nei paragrafi precedenti evidenzia come la progettazione dei sistemi di DWH moderni abbia fondamenti teorici consolidati rispetto ai quali le scelte architetturali concrete rappresentano risposte strutturate a esigenze specifiche di integrazione, governance e scalabilità. La distinzione tra modelli concettuali, logici e fisici descritta da Golfarelli e Rizzi [5] non rappresenta un'astrazione accademica ma la base metodologica su cui costruire qualsiasi framework di integrazione dati, ogni livello architetturale deve infatti preservare e tradurre in strutture eseguibili la semantica del dominio informativo, garantendo nel contempo che le trasformazioni applicate siano tracciate, controllabili ed adattabili nel tempo.

Le considerazioni relative alla modellazione dimensionale dei dati e all'organizzazione degli stessi, come la decisione nella scelta tra *star schema* e *Snowflake-schema*, incidono direttamente sulla manutenibilità del sistema informativo, sulle prestazioni in fase di interrogazione e sulla fruibilità delle informazioni da parte degli utenti finali. Gli stessi studi portano alla luce come l'evoluzione verso architetture *cloud-native*, l'introduzione di paradigmi di elasticità, *pay-per-use* e la separazione tra potenza computazionale e di memorizzazione dati, estendano ulteriormente le possibilità progettuali, favorendo la realizzazione di architetture a strati modulari e scalabili, meno vincolate da rigidità infrastrutturali.

Il riferimento all'integrazione di fonti eterogenee, basate su database relazionali e chiamate API REST, sottolinea l'importanza di distinguere chiaramente le fasi di acquisizione, normalizzazione e consolidamento del dato riflettendo il principio di separazione delle responsabilità applicato al ciclo di vita informativo, in cui ogni livello architetturale assolve ad una funzione, contribuendo alla riduzione delle interdipendenze e rendendo il sistema più controllabile e coerente nel tempo.

All'interno del quadro teorico riportato in questo capitolo, si colloca la proposta progettuale oggetto della presente tesi. Il framework multilivello adottato da Var Group - Data Science Operations rappresenta una coerente applicazione dei principi di separazione dei livelli, controllo della qualità, tracciabilità e modellazione dimensionale discussi in precedenza riflettendo una chiara distinzione tra acquisizione, consolidamento e pubblicazione del dato, secondo una logica di progressiva trasformazione informativa.

La reingegnerizzazione che verrà affrontata nei capitoli successivi costituisce dunque un'operazione di allineamento strutturale tra l'architettura effettivamente implementata e i principi teorici che ne guidano la progettazione. Il capitolo 3 analizzerà l'assetto attuale (AS-IS), identificandone le criticità ed evidenziando come queste derivino da una progressiva divergenza rispetto al framework teorico di riferimento, dovuta ad evoluzioni incrementali non guidate da una visione architetturale omogenea. La definizione della configurazione obiettivo (TO-BE) consente di ristabilire tale coerenza riconducendo ogni scelta progettuale ai principi delineati, con l'obiettivo di realizzare un sistema informativo robusto e allineato alle best practice viste in letteratura per quanto riguarda integrazione dati e governance.

Tabella 2.3: Collegamento tra concetti teorici e architettura TO-BE

Concetto Teorico	**Livello**	**Implementazione Concreta**
Separazione OLTP/OLAP [3]	L0	Area di *staging* isolata dalle sorgenti operative che replica i dati senza applicare logiche di business, preservando l'indipendenza tra sistemi operazionali e analitici
Modelli concettuali, logici e fisici [5]	L0 + L1 + L2	L0 mantiene il modello fisico fedele alla sorgente; L1 struttura i dati in modello logico consolidato; L2 implementa il modello concettuale dimensionale per l'analisi
DFM [5]	L1 $\rightarrow$ L2	Formalizzazione rigorosa di fatti, misure, dimensioni e gerarchie, traducibili in strutture multidimensionali coerenti con i requisiti analitici

Continua nella pagina successiva

Continua dalla pagina precedente

Concetto Teorico	**Livello**	**Implementazione Concreta**
Star schema [4, 10]	L2	Modellazione denormalizzata con tabella dei fatti centrale collegata direttamente a dimensioni semplici, ottimizzata per interrogazioni OLAP e fruibilità dell'utente finale
Modellazione dimensionale [4]	L1 + L2	Organizzazione dei dati secondo processi aziendali, con tabelle dei fatti a granularità specifica e dimensioni che forniscono contesto descrittivo
SCD [4]	L1	Gestione dell'evoluzione temporale degli attributi dimensionali mediante Type 2 (Add New Row), mantenendo coerenza storica e tracciabilità delle modifiche
Eterogeneità strutturale [21]	L0	Normalizzazione preliminare e trasformazione dei metadati che uniformano rappresentazioni sintattiche di database relazionali, file system e API REST in formato intermedio coerente
Eterogeneità semantica [21]	L1	Riconciliazione delle codifiche, mappatura dei significati, applicazione delle regole di business attraverso verifiche di integrità referenziale e conformità alle logiche aziendali
ETL tradizionale [13, 16]	L0 $\rightarrow$ L1	Estrazione, trasformazione in ambiente intermedio e caricamento strutturato, con controlli di coerenza preliminari e gestione della qualità

Continua nella pagina successiva

Continua dalla pagina precedente

Concetto Teorico	**Livello**	**Implementazione Concreta**
ELT (paradigma moderno) [16]	L1 → L2	Caricamento immediato nel database di destinazione seguito da trasformazioni eseguite nativamente nel DBMS, sfruttandone capacità di parallelismo e ottimizzazione
Data integration da fonti eterogenee [21, 23, 22]	L0	Framework unificato che gestisce acquisizioni da database relazionali, file semi-strutturati e API REST, normalizzando formati e schemi impliciti
Best practices ETL/ELT moderno [25, 32, 24]	L0 + L1 + L2	Governance centralizzata dei metadati, logging strutturato, separazione modularità, tracciabilità completa del lineage, recovery automatico e meccanismi di resilienza
Reingegnerizzazione architetturale [29, 31, 25]	Globale	Progressiva modernizzazione mediante introduzione di framework multilivello, separazione delle responsabilità, governance centralizzata e integrazione evolutiva di nuove sorgenti

Capitolo 3

Approccio metodologico all'architettura di integrazione dati: AS-IS e TO-BE

In questo capitolo verrà analizzata l'architettura di integrazione dati a supporto del DWH aziendale, descrivendone l'assetto attuale (AS-IS) e il modello obiettivo (TO-BE) previsto dal progetto di reingegnerizzazione. L'architettura in uso è nata per supportare esigenze di reporting e controllo interno in un contesto in cui i sistemi informativi si sono evoluti progressivamente nel tempo, dando luogo a flussi e componenti sviluppati in momenti differenti e secondo logiche non omogenee. L'analisi ha una duplice finalità: da un lato inquadrare le criticità dell'architettura AS-IS e i limiti degli attuali processi di integrazione, dall'altro introdurre l'architettura TO-BE e il framework multi-livello che dovrà guidare la standardizzazione dei flussi, la reingegnerizzazione delle sorgenti esistenti e l'integrazione delle nuove fonti dati, inclusa la gestione di dati semi-strutturati e l'adeguamento al modello dati al layer di pubblicazione.

3.1 Architettura attuale (AS-IS)

L'architettura attuale (AS-IS) del sistema di integrazione dati adottata da Var Group - Data Science Operations è il risultato di un'evoluzione incrementale, in cui sono state introdotte nel tempo diverse soluzioni allo scopo di rispondere ad esigenze operative e di reporting via via emergenti. L'assenza di una progettazione unitaria orientata sin dall'inizio alla standardizzazione e alla governance del dato ha portato ad un impianto complessivamente funzionante, ma caratterizzato da un'elevata eterogeneità tecnologica e da una ridotta flessibilità rispetto ad evoluzioni successive.

Le principali sorgenti informative coinvolte nell'architettura dati attuale sono OS Ticket e SAM, due sistemi centrali per la gestione delle attività operative e amministrative dell'azienda, le informazioni provenienti da entrambi i sistemi sono quindi complementari per l'analisi delle performance operative ed economiche:

- OS Ticket: è il sistema di ticketing interno utilizzato per tracciare le attività di assistenza e manutenzione, in particolare nell'ambito dei contratti AMS, da cui vengono ricavate informazioni su richieste, tempi di risoluzione e ore rendicontate;
- SAM: è il gestionale aziendale che raccoglie dati relativi ad ordini, commesse, sottocommesse, progetti, ore lavorate e fatturazioni.

Dal punto di vista tecnologico, l'alimentazione del DWH avviene attraverso due approcci distinti: una parte dei flussi è implementata in ODI, che estrae i dati da OS Ticket e li carica nelle tabelle di destinazione; le informazioni provenienti da SAM sono invece trattate principalmente tramite procedure PL/SQL eseguite direttamente sul database, le quali gestiscono lettura, trasformazione e caricamento dei dati adottando logiche di tipo *delete-insert* e ricarichi completi delle tabelle. Questa convivenza di flussi ODI e procedure PL/SQL introduce frammentazione nei processi di integrazione e complica le attività di manutenzione e controllo.

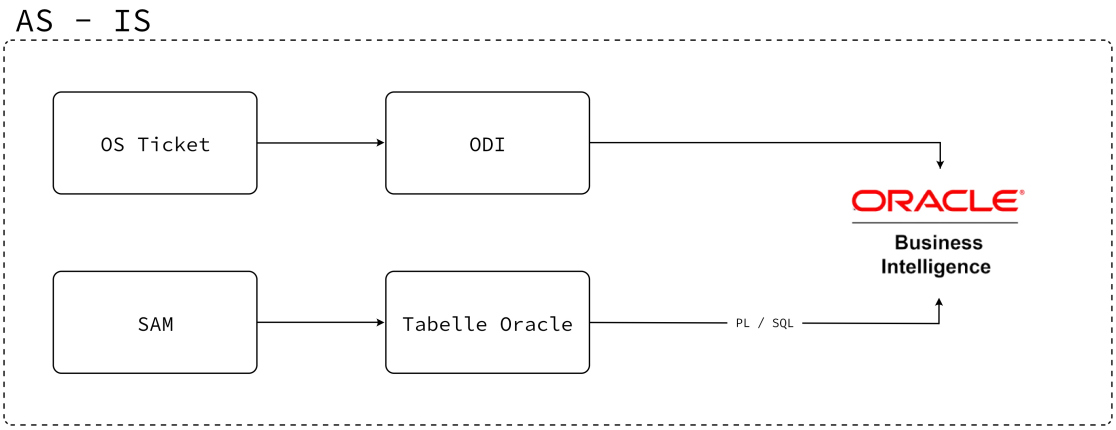

Figura 3.1: Architettura AS-IS del sistema di integrazione dati

Il modello dati su cui si basa l'attuale soluzione è altamente normalizzato, concepito originariamente per *Oracle Business Intelligence* e organizzato secondo uno *Snowflake schema*. Tale impostazione, adeguata al contesto tecnologico iniziale, oggi mostra limiti in termini di flessibilità, semplicità di utilizzo e compatibilità con strumenti di analisi più moderni, come Power BI, verso cui l'azienda si sta orientando, inoltre, la prospettiva di migrare verso Power BI rende necessario adottare un modello di pubblicazione basato su *star schema*, maggiormente ottimizzato per questo strumento rispetto allo *Snowflake schema*. L'elevato numero di tabelle dimensionali e il ricorso a molteplici join rendono infatti onerose sia le attività di manutenzione sia l'estensione del perimetro informativo e l'integrazione di nuove sorgenti.

Nel complesso, quindi, l'architettura AS-IS garantisce le esigenze di analisi correnti, ma evidenzia già alcuni limiti in termini di coerenza architetturale, standardizzazione dei processi e capacità di evoluzione, che saranno approfonditi nel paragrafo successivo.

3.2 Criticità dell'architettura AS-IS e limiti dei processi di integrazione

L'analisi dell'architettura attuale mette in luce diverse criticità di natura strutturale e metodologica, che ne riducono l'efficacia complessiva e la capacità di supportare in modo affidabile le esigenze analitiche dell'azienda. Tali criticità non sono riconducibili a singoli errori puntuali, ma piuttosto sono il risultato dell'assenza di un framework di integrazione formalizzato e della crescita incrementale del sistema,

che nel tempo ha privilegiato soluzioni ad hoc rispetto ad una progettazione organica di lungo periodo.

Una prima problematica che è stata osservata sin da subito riguarda la qualità del dato, fortemente influenzata sia dalle caratteristiche delle sorgenti operative sia dalle modalità con cui le informazioni vengono acquisite e trattenute. In particolare, il sistema SAM espone dati critici per l'analisi aziendale tramite tabelle in cui la totalità dei campi è memorizzata come `VARCHAR`, indipendentemente dal significato semantico, ne consegue che date, valori numerici, importi economici ed indicatori logici vengono gestiti come stringhe testuali, il che provoca una perdita di tipizzazione. Questa scelta ostacola l'applicazione di controlli formali, rende più complessa l'individuazione di anomalie e aumenta il rischio di errori nelle fasi di trasformazione e aggregazione, oltre a penalizzare le performance delle query analitiche e delle elaborazioni successive.

Ulteriori complessità derivano dall'integrazione di sorgenti eterogenee, in particolare SAM e OS Ticket, originariamente non progettate per comunicare tra di loro all'interno dell'architettura di integrazione, in quanto le informazioni fornite dai due sistemi differiscono per granularità, frequenza di aggiornamento e struttura logica, rendendo necessarie riconciliazioni gestite tramite regole specifiche e spesso non standardizzate. Questo approccio incrementa il rischio di incoerenze nei dati consolidati, duplicazioni e discrepanze nei valori analizzati, ad esempio in relazione alle ore rendicontate o alle anagrafiche condivise.

Dal punto di vista dei processi di integrazione, una criticità rilevante è l'assenza di una gestione strutturata degli aggiornamenti incrementali per la sorgente SAM, infatti questa implementa procedure mediante PL/SQL, adottando strategie di caricamento di tipo *delete-insert* o *truncate-insert*,equivalenti a un *full reload* della tabella di destinazione, poiché ne ricostruiscono integralmente il contenuto a ogni esecuzione. Pur essendo semplici da implementare, tali strategie non consentono di valorizzare concetti fondamentali quali il delta, la storicizzazione delle modifiche e la tracciabilità delle variazioni nel tempo, oltre a comportare un utilizzo inefficiente delle risorse computazionali e a limitare la scalabilità del sistema a fronte dell'aumento del volume dei dati.

Altro limite è legato alla mancanza di standard condivisi nella progettazione dei flussi ETL in quanto le pipeline presentano convenzioni di nomenclatura non omogenee, modalità differenti di gestione delle trasformazioni e una struttura eterogenea delle tabelle intermedie. Questa frammentazione rende onerose le attività di manutenzione, ostacola il riuso delle componenti e rende più complesso l'inserimento di nuovi sviluppatori sui flussi esistenti, soprattutto in assenza di una documentazione completa e sistematica.

Infine, i processi attuali risultano poco manutenibili e controllabili, mancando meccanismi sistematici di logging, audit e gestione delle anomalie che consentano di tracciare in modo puntuale l'esito delle esecuzioni e di individuare rapidamente

eventuali errori nei dati caricati. In presenza di risultati inattesi, l'analisi delle cause richiede spesso verifiche manuali, con conseguente aumento dei tempi di diagnosi e dei costi operativi.

3.3 Introduzione dell'architettura TO-BE e del framework di integrazione

L'analisi del contesto attuale ha evidenziato la necessità di definire un nuovo assetto per l'integrazione dei dati, capace di superare le debolezze riscontrate senza interrompere la continuità operativa e senza disperdere il capitale informativo già disponibile. L'intervento di reingegnerizzazione si propone quindi di riprogettare in modo coerente il funzionamento del DWH, adottando un impianto più ordinato, scalabile e allineato ai principi adottati nei moderni progetti di integrazione dati.

La configurazione obiettivo (TO -BE) è stata pensata, in fase di discussione del progetto, come un'evoluzione strutturata della soluzione esistente, orientata all'introduzione di un impianto più ordinato, scalabile e governabile, in linea con i principi adottati nei moderni progetti di integrazione dati. In questo contesto le sorgenti storiche SAM e OS Ticket continuano a costituire un elemento centrale dell'ecosistema informativo, soprattutto per la disponibilità di dati storici e per la continuità delle analisi; al contempo, la nuova architettura è stata progettata per supportare l'integrazione di ulteriori sorgenti, nel nostro caso Webgate, che espone i dati tramite chiamate API REST, la cui integrazione verrà descritta più dettagliatamente nel paragrafo 3.4.

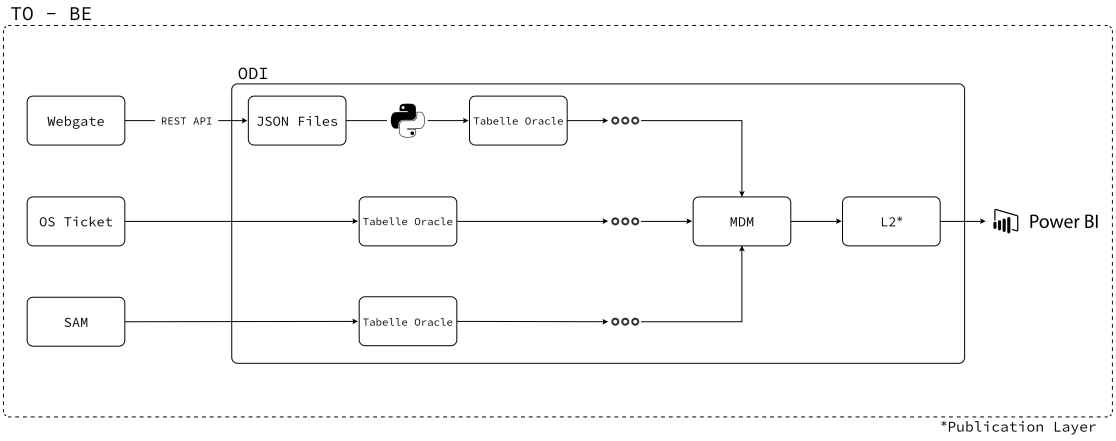

Figura 3.2: Architettura TO-BE del sistema di integrazione dati

Elemento centrale della proposta è l'adozione di un'architettura a strati, in cui ogni livello assume un ruolo specifico all'interno del ciclo di vita del dato. La

scomposizione in livelli consente di distinguere nettamente le fasi di acquisizione, consolidamento, controllo e messa a disposizione per le analisi, riducendo la dipendenza tra i componenti e rendendo più leggibile il percorso seguito dalle informazioni. Questo approccio facilita inoltre le attività quotidiane di manutenzione, il monitoraggio dei processi e il lavoro congiunto tra le diverse figure coinvolte nella gestione del sistema.

Il framework multilivello sviluppato da Var Group - Data Science Operations si basa sulla costruzione di tre livelli fondamentali: L0, L1 ed L2.

Il livello L0 costituisce il punto di ingresso del dato nel sistema analitico e svolge la funzione di area di *staging* e di gestione del delta. In questa fase i dati provenienti dalle sorgenti operative vengono acquisiti in modo aderente alle strutture di origine, senza applicare trasformazioni semantiche o logiche di business, così da preservare l'integrità del dato sorgente. Operativamente, il caricamento può avvenire tramite strategie di *truncate–insert* o in modalità incrementale, a seconda delle caratteristiche della sorgente e della disponibilità di informazioni sulle variazioni. Accanto alla semplice copia, L0 include meccanismi per l'individuazione delle modifiche rispetto al ciclo precedente, basati su timestamp di aggiornamento, indicatori di ultima modifica oppure, in mancanza di tali attributi, su confronti differenziali tra versioni successive della *staging*. In questo modo il livello L0 fornisce ai livelli a valle un insieme di record già filtrato sulle sole variazioni rilevanti, riducendo il volume di dati da processare e migliorando le prestazioni complessive dei flussi.

Il livello L1 rappresenta il nucleo del consolidamento informativo e concentra la maggior parte delle trasformazioni strutturate. È organizzato in una serie di sottofasi che riflettono sia una distinzione concettuale sia una concreta articolazione dei processi ETL. La fase OK applica controlli di qualità strutturali e di contenuto, verificando integrità referenziale, completezza dei campi obbligatori, coerenza delle codifiche e rispetto delle regole di business; solo i record che superano tali verifiche proseguono lungo il flusso. La sottofase ODS funge da area di consolidamento e storicizzazione, in cui i dati vengono integrati tramite operazioni di *merge* che consentono di tracciare le evoluzioni nel tempo e di mantenere una base informativa completa e coerente. La componente di MDM gestisce l'armonizzazione delle anagrafiche condivise tra più sorgenti, come clienti, commesse o fornitori,definendo regole di riconciliazione e priorità tra le fonti, risolvendo duplicazioni e generando surrogate key stabili e consistenti. Infine, la fase OUT prepara i dati alla pubblicazione, applicando arricchimenti e calcoli derivati, completando le normalizzazioni necessarie e producendo strutture pronte per il caricamento nel livello finale, mantenendo al contempo versioni temporali utili per esigenze di audit, rollback e ricostruzione storica.

Il livello L2 costituisce la fase di pubblicazione definitiva verso gli strumenti di analisi e reportistica. Poiché le strutture prodotte dalla fase OUT possono

contenere più versioni temporali dello stesso record, in L2 vengono applicate logiche di selezione della versione corrente, ad esempio tramite funzioni analitiche come `ROW_NUMBER()`, che permettono di ordinare i record in base ai cicli di caricamento e isolare l'immagine più recente del dato:

```
1     ROW_NUMBER() OVER (
2       PARTITION BY <CHIAVE>
3       ORDER BY JOBID_L0 DESC, FLG_NEG ASC, INS_TIME DESC
4     )
```

Listing 3.1: `ROW_NUMBER()`

In questo livello le informazioni vengono modellate in funzione delle esigenze di reporting e delle caratteristiche della piattaforma analitica, garantendo semplicità di utilizzo, coerenza semantica e prestazioni adeguate per le interrogazioni di business, e schermando completamente gli utenti finali dalla complessità dei processi di integrazione a monte.

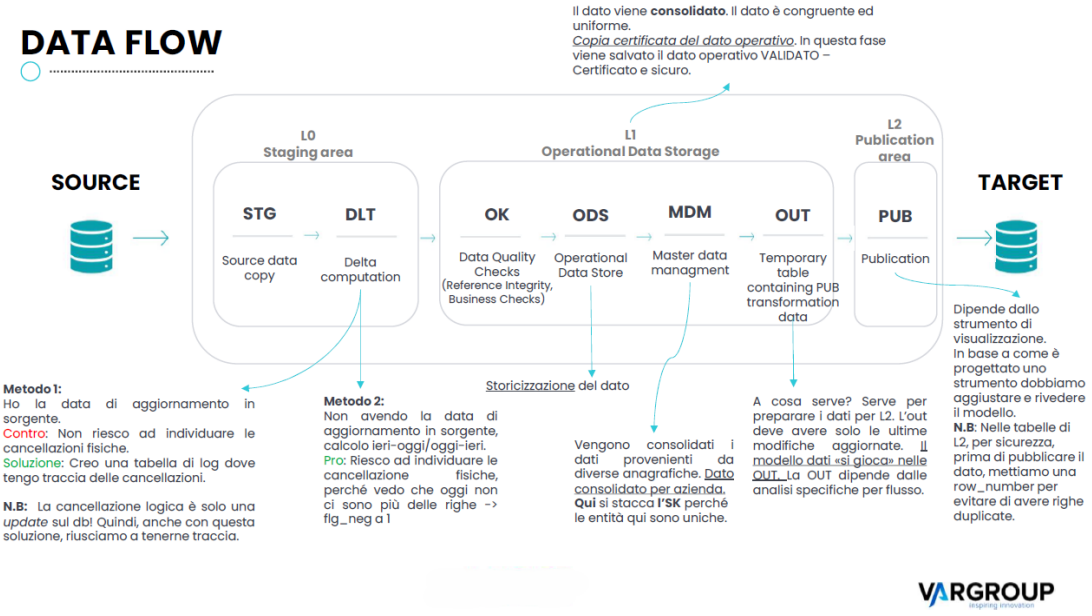

Figura 3.3: Framework Var Group - Data Science Operations

Dal punto di vista tecnologico, la soluzione futura prevede di utilizzare ODI come piattaforma unica per la gestione dei processi ETL, eliminando la frammentazione dovuta alla coesistenza di strumenti e approcci differenti. Tutte le sorgenti, sia storiche sia di nuova introduzione, vengono convogliate in questo framework, all'interno del quale vengono applicate in modo uniforme le logiche di caricamento, controllo e pubblicazione. Nell'ambito della tesi, l'attenzione è focalizzata in particolare sulla

progettazione e realizzazione dei livelli L0 ed L1, che rappresentano il cuore della trasformazione architetturale; il livello L2 verrà completato in una fase successiva, dopo aver definito insieme ai colleghi responsabili della pubblicazione del dato e della reportistica finale le esigenze lato front-end per le analisi realizzate tramite Power BI.

3.4 Integrazione della nuova sorgente informativa, gestione dei dati semi-strutturati e adeguamento al layer di produzione

L'architettura obiettivo non ha come unico scopo razionalizzare i flussi esistenti, ma è pensata anche per accogliere nuove sorgenti in modo strutturato e coerente con il framework di integrazione definito. In questo scenario, l'introduzione di Webgate assume un ruolo particolarmente rilevante, perché modifica in maniera significativa le modalità di acquisizione del dato rispetto ai sistemi tradizionali presenti nel modello attuale.

A differenza di OS Ticket e SAM, che espongono basi dati relazionali interrogabili direttamente a livello di database, Webgate rende disponibili le informazioni tramite servizi REST, richiedendo quindi l'uso di chiamate API per estrarre i dataset di interesse. I dati vengono restituiti in formato JSON, con una struttura semi-strutturata e potenzialmente soggetta a variazioni nel tempo, introducendo un ulteriore livello di complessità nella fase di integrazione, che deve gestire non solo il contenuto informativo, ma anche l'evoluzione dei tracciati e la trasformazione verso un modello relazionale.

All'interno del framework obiettivo, l'integrazione di Webgate viene comunque ricondotta alle stesse logiche architetturali adottate per le sorgenti storiche SAM e OS Ticket, infatti, il processo inizia con una chiamata di autenticazione, necessaria per ottenere il token di accesso alle API, seguita da una o più richieste di tipo GET per scaricare i dataset richiesti: le risposte JSON ottenute dalle API sono considerate a tutti gli effetti sorgenti del livello L0, dove vengono memorizzate in forma il più possibile aderente alla struttura originale, così da preservare il contenuto informativo e garantire la tracciabilità delle estrazioni.

La trasformazione dei dati semi-strutturati in tabelle relazionali avviene in una fase successiva, in cui i file JSON vengono elaborati e mappati su strutture compatibili con il database di destinazione, in questo modo una sorgente eterogenea viene ricondotta a un modello coerente con il resto dell'architettura, rendendo possibile applicare le stesse logiche di controllo, storicizzazione e consolidamento previste per le altre fonti. Una volta normalizzati, i dati di Webgate seguono lo stesso percorso dei flussi tradizionali, attraversando i livelli L0 ed L1 per la gestione degli aggiornamenti incrementali, l'applicazione delle regole di qualità

e l'integrazione con le informazioni provenienti da OS Ticket, SAM e dalle altre sorgenti.

L'inserimento di Webgate nel framework obiettivo mette in evidenza uno dei vantaggi principali dell'architettura proposta: la separazione tra le modalità di acquisizione del dato e le logiche di integrazione e consolidamento. Anche quando le tecnologie e i formati di origine differiscono in modo sostanziale, il framework consente di uniformare il trattamento delle informazioni a partire dallo *staging*, riducendo l'impatto della complessità tecnica sul resto del sistema. L'architettura obiettivo risulta così predisposta all'evoluzione futura del sistema informativo, permettendo l'integrazione di nuove sorgenti, anche non relazionali, senza dover ricorrere a soluzioni ad hoc o a ulteriori frammentazioni dei processi ETL. L'approccio adottato garantisce così coerenza architetturale, riutilizzabilità delle componenti e maggiore governabilità del dato, aspetti essenziali per sostenere la crescita e l'evoluzione delle esigenze analitiche aziendali.

La reingegnerizzazione dell'architettura di integrazione richiede anche una revisione del modello dati analitico, che costituisce il punto di contatto tra i processi di integrazione e gli strumenti di reporting. Il modello attuale è fortemente influenzato da Oracle Business Intelligence ed è basato su uno schema a fiocco di neve (*Snowflake*), caratterizzato da un'elevata normalizzazione delle dimensioni e da numerose tabelle interconnesse che, pur essendo in linea con le architetture di BI tradizionali, nella pratica mostra limiti significativi in termini di flessibilità, manutenibilità e prestazioni, soprattutto in relazione a strumenti di data visualization più moderni. Nel modello obiettivo è previsto il passaggio a Power BI come strumento principale di analisi e visualizzazione, il che rende necessaria una revisione sostanziale del modello di pubblicazione, in quanto Power BI lavora in modo più efficace con strutture semplici e facilmente leggibili, basate su schemi a stella (*star schema*). In tale configurazione le tabelle dei fatti sono collegate direttamente a un numero limitato di dimensioni, riducendo la profondità delle relazioni e semplificando i join necessari per il calcolo delle misure.

All'interno del framework obiettivo, il redesign del modello dati si completa nel passaggio dal livello L1 al livello L2, che rappresenta l'area di pubblicazione. L1 contiene già fatti e dimensioni definiti a partire da dati consolidati, coerenti e storicizzati, ma non è pensato per essere interrogato direttamente dagli strumenti di reporting, mentre L2 ha il compito di riorganizzare e specializzare queste strutture in un modello orientato alla fruizione analitica, modellando le relazioni e i layout in funzione delle esigenze di business e delle caratteristiche della piattaforma di visualizzazione adottata. In questa fase vengono selezionati gli attributi rilevanti, aggregate le informazioni provenienti da più sorgenti e ridotta la complessità complessiva del modello, così da renderlo più leggibile e performante per gli strumenti di analisi.

Una differenza importante rispetto all'architettura attuale riguarda la gestione

delle dimensioni, infatti, nel modello obiettivo, molte informazioni anagrafiche che in precedenza erano distribuite su più tabelle normalizzate vengono accorpate in dimensioni più ricche, che contengono attributi già pronti per l'analisi. Questo approccio migliora la comprensibilità del modello, riduce il numero di relazioni da gestire e facilita il lavoro degli utenti di business, che possono costruire report e dashboard senza doversi confrontare con una rete complessa di join. Parallelamente, la progettazione del layer L2 tiene conto delle esigenze di performance e di qualità del dato, introducendo controlli per evitare duplicazioni e garantire l'univocità delle entità pubblicate.

Il redesign del modello dati non è quindi un semplice adeguamento tecnico, ma un passaggio progettuale centrale nell'evoluzione dell'architettura complessiva del DWH. La definizione del modello di pubblicazione diventa il punto in cui convergono scelte architetturali, processi di integrazione ed esigenze analitiche, determinando la qualità e l'efficacia delle analisi di business. In questa prospettiva, il livello L2 rappresenta il livello in cui il framework di integrazione rende pienamente visibile il proprio valore agli utilizzatori del sistema informativo, offrendo un modello dati direttamente fruibile dagli strumenti di analisi.

Nel complesso, l'adozione di un modello dati orientato allo schema a stella, inserito in un'architettura a livelli ben definiti, consente di superare i limiti del modello attuale e di predisporre un assetto più coerente con le esigenze analitiche presenti e future. Il redesign del layer di pubblicazione completa il percorso di evoluzione architetturale, rendendo il DWH più semplice da utilizzare, più manutenibile e maggiormente allineato agli obiettivi di supporto alle decisioni aziendali.

3.5 Considerazioni di sintesi sull'evoluzione architetturale

L'analisi dell'architettura attuale ha evidenziato come l'evoluzione incrementale del sistema di integrazione dati abbia consentito di soddisfare le esigenze correnti di analisi e reportistica, al prezzo però di una forte eterogeneità tecnologica, di una limitata standardizzazione dei processi e di una scarsa attenzione alla qualità e alla tracciabilità del dato nel tempo. Le criticità emerse, a partire dalla tipizzazione debole delle sorgenti, alla mancanza di gestione strutturata del delta, fino alla frammentazione dei flussi ETL, rendono complessa la manutenzione, ostacolano l'estensione del perimetro informativo e limitano la capacità del DWH di adattarsi a nuove fonti e nuovi strumenti analitici.

La definizione dell'architettura obiettivo e del framework multilivello L0 – L1 – L2 rappresenta la risposta strutturata a queste debolezze: la chiara separazione tra *staging*, consolidamento e pubblicazione consente di uniformare i meccanismi di

caricamento, controllo e storicizzazione, riducendo la dipendenza tra componenti e migliorando la governabilità complessiva del sistema. L'integrazione di sorgenti eterogenee, come le basi dati relazionali tradizionali e i servizi API REST che restituiscono dati in formato JSON, viene ricondotta a un percorso architetturale comune, che rende possibile applicare le stesse logiche di qualità, arricchimento e riconciliazione anche in presenza di formati e tecnologie differenti.

In questo quadro, la revisione del modello dati analitico e l'adozione di uno *star schema* nel livello di pubblicazione assumono un ruolo centrale: da un lato permettono di superare i limiti dello *Snowflake schema* originariamente pensato per Oracle Business Intelligence, dall'altro allineano il DWH alle esigenze di strumenti come Power BI, semplificando il lavoro degli utenti di business e migliorando le prestazioni delle interrogazioni. Il livello L2 diventa così il punto di emersione del valore del framework di integrazione, offrendo un modello dati più leggibile, più stabile nel tempo e maggiormente coerente con gli obiettivi di supporto alle decisioni aziendali.

Capitolo 4

Implementazione del framework ETL in ODI

4.1 Livello L0 - *Staging Area*

4.1.1 Fase di preparazione all'implementazione in ODI

La fase di preparazione all'implementazione in ODI del livello L0 è nata dall'esigenza di costruire un'infrastruttura solida, in grado non solo di replicare i dati dei sistemi sorgente, ma anche di farlo in modo controllato, tracciabile e facilmente ri-eseguibile. In questa prospettiva, il livello L0 rappresenta il punto di ingresso dell'intero flusso informativo: qui le informazioni vengono raccolte dalla sorgente nella loro forma grezza, per poi essere organizzate e predisposte all'elaborazione nei livelli successivi (L1 e L2), dove saranno poi rese disponibili per la fruizione analitica.

Nel progetto che ho seguito, le principali sorgenti di dati sono tre: SAM, OSTicket e Webgate; per quest'ultimo il percorso è leggermente più articolato, i dati infatti vengono estratti tramite chiamate API REST di tipo GET, memorizzati in file JSON in forma semi-strutturata e quindi trasformati da un processo Python, che produce strutture coerenti con il modello del DWH, pronte per essere caricate su tabelle Oracle tramite ODI. Le sorgenti SAM e OSTicket seguono invece modalità di caricamento strutturate e convergono anch'esse alla medesima architettura di *staging*, così da uniformare la gestione dei dati fin dal primo livello.

Dal punto di vista operativo, il lavoro è iniziato definendo e creando le tabelle necessarie direttamente sul database Oracle utilizzando SQL Developer. Per ogni sorgente e per ciascuna entità replicata sono state definite in maniera omogenea tre tipologie di tabelle: STG, DLT e DLT_HIS. Le tabelle di STG costituiscono l'area di acquisizione del dato per il singolo ciclo di estrazione: l'obiettivo in questa fase è preservare il contenuto informativo senza introdurre trasformazioni

non strettamente necessarie, in modo da mantenere una copia quanto più fedele possibile delle informazioni provenienti dalle sorgenti.

```
1     CREATE TABLE "L0"."SAM_STG_EXT_ORDINI_FORN_VG"
2        (
3         "COD_COLLEGAMENTO" VARCHAR2(400 CHAR),
4       "ANNO" VARCHAR2(400 CHAR),
5       "CAUSALE" VARCHAR2(400 CHAR),
6       "COD_TIPO" VARCHAR2(400 CHAR),
7       "DOC_NUMERO" VARCHAR2(400 CHAR),
8       "DOC_RIGA" VARCHAR2(400 CHAR),
9       "TIPO_DOC" VARCHAR2(400 CHAR),
10      "COD_FORNITORE" VARCHAR2(400 CHAR),
11      "RAG_SOC" VARCHAR2(400 CHAR),
12      "DATA_DOC" VARCHAR2(400 CHAR),
13      "COD_ARTICOLO" VARCHAR2(400 CHAR),
14      "DESC_ARTICOLO" VARCHAR2(400 CHAR),
15      "UNITA_MISURA" VARCHAR2(400 CHAR),
16      "UNITA_PREZZO" VARCHAR2(400 CHAR),
17      "QTA" VARCHAR2(400 CHAR),
18      "IMPORT_NETTO" VARCHAR2(400 CHAR),
19      "NUMERO_FATTURA" VARCHAR2(400 CHAR),
20      "DATA_FATTURA" VARCHAR2(400 CHAR),
21      "JOBID_L0" NUMBER NOT NULL ENABLE,
22      "INS_TIME" DATE DEFAULT SYSDATE
23       ) SEGMENT CREATION IMMEDIATE
24      PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
25      NOCOMPRESS  NOLOGGING
26      TABLESPACE "TBS_L0"
27      PARTITION BY RANGE ("JOBID_L0") INTERVAL (1000000)
28      (PARTITION "STORICO"  VALUES LESS THAN (20190101000000)
    SEGMENT CREATION IMMEDIATE
29      PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
30      NOCOMPRESS NOLOGGING
31      TABLESPACE "TBS_L0" );
```

Listing 4.1: Tabella STG esempio[1]

Le tabelle DLT (e DLT_HIS) sono invece dedicate al delta (variazione) tra estrazioni consecutive e servono ad isolare i record nuovi o modificati rispetto allo snapshot precedente. Il delta viene determinato confrontando due insiemi di dati relativi ad esecuzioni successive, che per semplicità indicheremo come OGGI, riferito ai record acquisiti nel ciclo corrente, e IERI, riferito ai record presenti nel ciclo immediatamente precedente, e applicando operazioni di differenza insiemistica

[1]Nelle tabelle STG i campi informativi sono stati lasciati come `VARCHAR2(400 CHAR)` per preservare la massima aderenza alla sorgente; le conversioni di tipo e i controlli di qualità verranno effettuati nei livelli successivi.

che generano due insiemi logici: OGGI_IERI, che raccoglie i record presenti oggi ma non ieri, e IERI_OGGI, che contiene i record presenti ieri ma non più presenti oggi. A valle di questo passaggio, i due insiemi vengono ricombinati in un'unica struttura DLT, alla quale viene associato un indicatore di "segno" che consente, nei livelli successivi di interpretare correttamente le azioni da applicare ai dati, quali inserimento, aggiornamento e cancellazione logica.

```
1     CREATE TABLE "L0"."SAM_DLT_EXT_ORDINI_FORN_VG"
2        (
3         "COD_COLLEGAMENTO" VARCHAR2(400 CHAR),
4       "ANNO" VARCHAR2(400 CHAR),
5       "CAUSALE" VARCHAR2(400 CHAR),
6       "COD_TIPO" VARCHAR2(400 CHAR),
7       "DOC_NUMERO" VARCHAR2(400 CHAR),
8       "DOC_RIGA" VARCHAR2(400 CHAR),
9       "TIPO_DOC" VARCHAR2(400 CHAR),
10      "COD_FORNITORE" VARCHAR2(400 CHAR),
11      "RAG_SOC" VARCHAR2(400 CHAR),
12      "DATA_DOC" VARCHAR2(400 CHAR),
13      "COD_ARTICOLO" VARCHAR2(400 CHAR),
14      "DESC_ARTICOLO" VARCHAR2(400 CHAR),
15      "UNITA_MISURA" VARCHAR2(400 CHAR),
16      "UNITA_PREZZO" VARCHAR2(400 CHAR),
17      "QTA" VARCHAR2(400 CHAR),
18      "IMPORT_NETTO" VARCHAR2(400 CHAR),
19      "NUMERO_FATTURA" VARCHAR2(400 CHAR),
20      "DATA_FATTURA" VARCHAR2(400 CHAR),
21      "JOBID_L0" NUMBER NOT NULL ENABLE,
22      "FLG_NEG" NUMBER NOT NULL ENABLE,
23      "INS_TIME" DATE DEFAULT SYSDATE
24       ) SEGMENT CREATION IMMEDIATE
25      PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
26      NOCOMPRESS  NOLOGGING
27      TABLESPACE "TBS_L0"
28      PARTITION BY RANGE ("JOBID_L0") INTERVAL (1000000)
29      (PARTITION "STORICO"  VALUES LESS THAN (20190101000000)
    SEGMENT CREATION IMMEDIATE
30      PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
31      NOCOMPRESS NOLOGGING
32      TABLESPACE "TBS_L0" );
```

Listing 4.2: Tabella DLT esempio

Per rendere questa architettura realmente governabile è stato necessario introdurre fin da subito, nelle STG, dei campi tecnici e convenzioni di tracciamento coerenti con le logiche di orchestrazione e con i requisiti di audit. Tra questi, il campo `JOBID_L0` identifica in modo univoco ogni ciclo di carico, utilizzando generalmente un timestamp nel formato `"YYYYMMDDHH24MISS"` e permette di associare

ogni record a una specifica esecuzione dell'ETL, abilitando controlli di avanzamento, ripartenze e rielaborazioni mirate. Il campo `INS_TIME` registra il momento esatto in cui il record viene inserito nel database, fornendo un ulteriore appiglio per analisi di diagnostica o per verifiche puntuali. Infine il campo `FLG_NEG` sintetizza il tipo di variazione associata al record: un valore 0 identifica un'informazione da inserire o attivare, mentre un valore 1 indica un record eliminato. Dal punto di vista operativo, le tabelle intermedie OGGI_IERI e IERI_OGGI vengono volutamente mantenute prive di questi campi tecnici; `JOBID_L0` e `INS_TIME` vengono reintrodotti solo nella fase finale di costruzione della DLT, insieme al `FLG_NEG`, così da evitare che le informazioni di tracciamento interferiscano con il calcolo del delta.

In parallelo alla definizione delle strutture di replica, è stata predisposta la componente di storicizzazione tramite l'utilizzo delle tabelle di DLT_HIS che estendono il livello L0 con una memoria storica dei delta associati ai diversi cicli di estrazione e, a parità di struttura rispetto alle DLT, sono tipicamente partizionate per `JOBID_L0`. Questo accorgimento consente di ricostruire l'evoluzione del dato nel tempo e di ricalcolare, se necessario, i livelli successivi senza dover tornare alla sorgente, che spesso espone solo lo stato corrente dell'informazione. In pratica, la `DLT_HIS` diventa una sorta di tracciato dei cambiamenti, indispensabile quando si devono analizzare o correggere comportamenti del sistema su finestre temporali passate.

Una volta completata la preparazione delle strutture fisiche, l'attenzione si è spostata sulla configurazione dell'ambiente ODI, ponendo l'accento sulla distinzione tra architettura fisica e logica. Nella sezione *Topology* di ODI è stato creato un *Data Server Oracle* contenente i parametri di connessione al database aziendale, che rappresenta il punto di accesso concreto su cui ODI esegue le proprie operazioni. All'interno di questo *Data Server* è stato definito il *Physical Schema* del livello L0, cioè lo schema che ospita le tabelle STG, DLT e DLT_HIS; a questo è stato associato un *Logical Schema* dedicato al livello L0, che viene utilizzato nei *Mapping* come riferimento astratto alle strutture fisiche. Il collegamento tra schema logico e fisico è stato gestito tramite i contesti in modo che il medesimo oggetto logico possa essere associato a schemi diversi a seconda dell'ambiente, principalmente si riconoscono quattro ambienti: *Development*, *Global*, *Production* e *Test*. Questa separazione consente di progettare i flussi senza vincolarli a un'istanza specifica del database, semplificando sia i rilasci che le migrazioni tra ambienti.

Figura 4.1: Configurazione della Topology in ODI: architettura fisica, contesti di esecuzione e architettura logica

Completata la *Topology*, è stato creato il modello per il livello L0 in ODI. In questa fase sono stati configurati i parametri richiesti dallo strumento, quali nome del modello, codice, tecnologia Oracle, schema logico di riferimento e gruppo di azione, e quindi si è proceduto con la *reverse engineering* delle tabelle già presenti sul database. L'operazione ha permesso di importare nella repository di ODI i metadati strutturali, come colonne, tipi di dato e chiavi tecniche, garantendo l'allineamento tra ciò che è definito nel database e ciò che viene utilizzato nel *Mapping*. Questo approccio garantisce una riduzione del rischio di inconsistenze, facilitando la manutenzione evolutiva, in quanto grazie alla *reverse engineering* è possibile aggiornare il modello ODI lavorando solo sulle tabelle interessate.

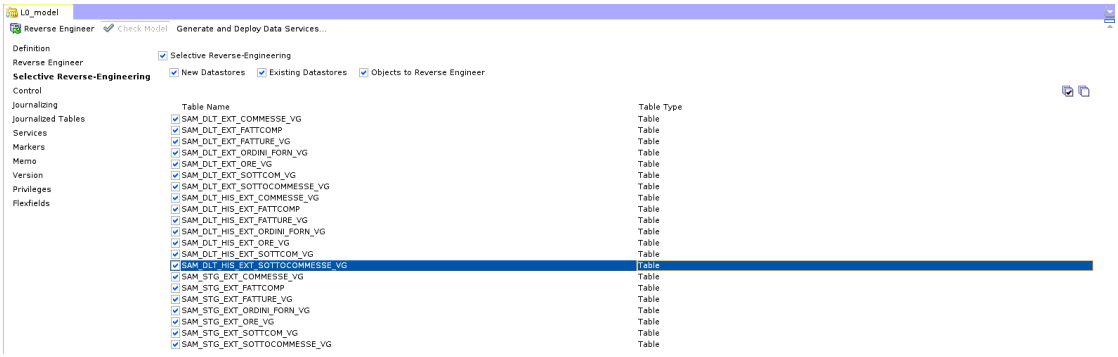

Figura 4.2: Reverse engineering selettivo delle tabelle STG, DLT e DLT_HIS nel modello ODI del livello L0 per la sorgente SAM

Infine, rimanendo coerenti con le best practice operative in ODI, la preparazione dell'ambiente L0 ha tenuto conto anche delle modalità di esecuzione in esercizio: una volta definiti i *Mapping*, l'esecuzione non avviene lanciandoli direttamente dall'ambiente di sviluppo, bensì passando attraverso la generazione di scenari e il lancio di questi. Questo approccio consente di ottenere un oggetto eseguibile, versionabile e distribuibile, adatto all'utilizzo in contesti di test e produzione, e facilita l'integrazione dei flussi con i meccanismi di orchestrazione, come *Load Plan*, *Package* e *scheduler*, risultando in un processo più governabile, replicabile e tracciabile.

4.1.1.1 Metadati e processo di logging

Parallelamente alla configurazione dello schema L0, è stata predisposta la componente di metadati di processo nello schema dedicato `ODI_ADMIN`, che rappresenta il fulcro delle logiche di controllo e di sincronizzazione del framework aziendale. In questo schema trovano posto le seguenti tabelle, insieme alle procedure e funzioni di supporto:

- `FLOW_MANAGER`: viene utilizzata per tracciare il ciclo di vita dei processi ETL registrando per ogni esecuzione l'istante di avvio, l'istante di fine e lo stato finale, distinguendo tra esecuzioni completate correttamente, concluse in errore o ancora in corso;

- `TABLE_MANAGER`: svolge un ruolo specifico nel livello L0, registrando per ogni tabella gestita in modalità incrementale il punto di avanzamento dell'ultima lettura e le principali informazioni di sintesi, come il numero di righe replicate. In questo modo formalizza il punto di sincronia del processo, che può essere espresso come valore temporale massimo, ad esempio attraverso l'utilizzo di timestamp, oppure come `JOBID` di riferimento aggiornato alla fine di ogni

replica. Tale meccanismo è particolarmente rilevante nei casi di CDC manuale, perché consente ad ogni nuovo ciclo di riprendere la lettura dal punto corretto senza perdita o duplicazione di dati;

- `METADATA_MANAGER`: è impiegata per esternalizzare parametri e configurazioni utilizzati dai flussi, accessibili a runtime, ad esempio tramite variabili ODI, in modo da rendere il sistema più flessibile e meno dipendente da modifiche applicative.

4.1.1.2 Variabili di orchestrazione e sincronia del framework ODI

Le variabili in ODI rappresentano un elemento fondamentale del framework, perché permettono di rendere i flussi parametrici e governabili, evitando che informazioni critiche vengano codificate direttamente all'interno di *Mapping* e *Package*. Pur essendo state introdotte durante la fase di preparazione precedente all'effettiva ingegnerizzazione del livello L0, queste variabili sono state progettate per avere valenza trasversale e vengono riutilizzate lungo l'intero percorso che inizia con L0 e finisce con L2, avendo un ruolo di supporto per l'orchestrazione, per la tracciabilità e per la gestione del punto di sincronia nei caricamenti incrementali.

In questa prospettiva è stata definita una serie di variabili di progetto in ODI con l'obiettivo di disaccoppiare la logica di controllo dal codice implementativo, valorizzate tramite query SQL o funzioni applicative presenti sul database e sono richiamate nei *Package* principali allo scopo di guidare l'esecuzione dei flussi e aggiornare in modo coerente i metadati di processo.

In questo contesto, le variabili possono essere raggruppate in due insiemi distinti: variabili trasversali di orchestrazione e sincronia, comuni a tutti i livelli e utilizzate per parametrizzare i *Package* e governare l'esecuzione dei flussi, e variabili specifiche di livello, impiegate per tracciare le elaborazioni e marcare i dati all'interno dei singoli layer (L0, L1 e L2).

Le variabili trasversali:

- `GRP_NAME`: viene utilizzata per rappresentare il gruppo logico del processo, seguendo una convenzione di tipo livello_sorgente (e.g. L0_SAM), e risulta utile sia per mantenere coerenza nella nomenclatura, sia per collegare correttamente i processi alle relative informazioni di logging e sincronia;

- `IDENTITY`: distingue la natura del flusso, indicando se si tratta di *ANAGRAFICHE* o *FATTI*, facilitando l'applicazione di regole di orchestrazione differenziate, ad esempio in termini di dipendenze o di priorità di esecuzione, contribuisce a rendere più leggibile la struttura complessiva del progetto;

- `NUM_LEVEL`: identifica il livello architetturale in cui sta operando il *Package* (0 per L0, 1 per L1 e 2 per L2), permettendo di riutilizzare gli stessi meccanismi in

più livelli senza duplicare il codice e mantenendo un comportamento coerente lungo tutta la pipeline;

- `TABLE_NAME`: viene impiegata nei *Package*, in particolare quelli dedicati alla gestione del delta per le coppie STG - DLT, per indicare la tabella oggetto dell'elaborazione, consentendo il riutilizzo di strutture analoghe di *Package* su entità diverse, variando semplicemente il valore della variabile;
- `STATUS`: sintetizza l'esito del processo distinguendo tra esecuzioni concluse correttamente, ancora in corso o terminate con errore, in coerenza con i codici utilizzati nel sistema di logging;
- `JOBID`: è una variabile popolata come identificativo univoco del run corrente, generando un timestamp numerico nel formato `"YYYYMMDDHH24MISS"`;

```
1         SELECT TO_NUMBER(TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'))
2         FROM DUAL;
3
```

Listing 4.3: Definizione variabile `JOBID`

- `LAST_JOBID`: permette di ottenere l'ultimo `JOBID` valido e disponibile per il flusso considerato;

```
1         SELECT MMBI_DATI.GET_LAST_JOBID('#MMBI_DATI.IDENTITY',
      #MMBI_DATI.NUM_LEVEL, '#MMBI_DATI.GRP_NAME')
2         FROM DUAL
3
```

Listing 4.4: Definizione variabile `LAST_JOBID`

- `LAST_DATE_READ`: recupera, per uno specifico scenario e contesto, l'ultima data di lettura valida, invocando una funzione che utilizza le informazioni registrate nelle tabelle dei metadati per restituire il corretto punto di avanzamento.

```
1         SELECT MMBI_DATI.GET_LAST_DATE_READ('#MMBI_DATI.
    IDENTITY', #MMBI_DATI.NUM_LEVEL, '#MMBI_DATI.GRP_NAME', '#
    MMBI_DATI.TABLE_NAME')
2         FROM DUAL;
3
```

Listing 4.5: Definizione variabile `LAST_DATE_READ`

Le variabili specifiche di livello:

- `JOBID_L`n: utilizzate per marcare i record e descrivere l'avanzamento delle elaborazioni nei diversi strati dell'architettura, consentendo di isolare i dati

prodotti da una specifica esecuzione, effettuare eventuali controlli e predisporre ripartenze controllate o rilanci selettivi senza dover ripercorrere l'intera catena di caricamento.

- `NUM_ROWS`: utilizzata nel livello L0 allo scopo di verificare il numero di righe caricate per uno specifico `JOBID_L0` distinguendo poi tra due casistiche: se il conteggio è maggiore o uguale ad uno, il flusso può proseguire aggiornando il punto di sincronia ed eseguendo le elaborazioni successive, altrimenti l'esecuzione può essere interrotta o gestita come anomalia.

```
1         SELECT COUNT(1)
2         FROM <%=snpRef.getSchemaName( "MMBI_DATI-L0", "D" )
      %>.#DATASCIENCEBI.TABLE_NAME
3         WHERE JOBID_L0 = #JOBIDNCEBI.JOBID_L0;
4
```

Listing 4.6: Definizione variabile `NUM_ROWS`

- `NULL_`*`PARAM_NAME`*: sono variabili utilizzate come valori di default per la gestione dei campi nulli nella fase di OK nel livello L1. Non sono variabili codificate direttamente nel *Mapping*, ma vengono lette dalla tabella `METADATA_MANAGER`, così che, nel caso fosse necessario in futuro modificare i valori di default, non sarebbe richiesto di intervenire manualmente su ogni *Mapping* o procedura, ma sarebbe sufficiente aggiornare il valore corrispondente nel DB. In particolare, il framework utilizza le seguenti:
  - `NULL_CHAR` = `Z`
  - `NULL_DATE` = `19010101000000`
  - `NULL_NUMBER` = `0`
  - `NULL_JOBID` = `19000101000000`
  - `NULL_VARCHAR` = `ND`
  - `NULL_DESC` = `Not Defined`

  In generale la query di definizione di una qualsiasi di queste variabili è:

```
1         SELECT PARAM_VALUE
2         FROM METADATA_MANAGER
3         WHERE PARAM_NAME = '<NULL_PARAM_NAME>';
4
```

Listing 4.7: Query generica per il recupero dei default da `METADATA_MANAGER`

4.1.1.3 Procedure di gestione del ciclo di vita dei processi ETL

Nel contesto del framework ODI sviluppato, le *Procedure* rappresentano un altro tassello fondamentale della componente di orchestrazione e controllo, complementare alla variabili appena descritte. In PL/SQL, una procedura è un blocco di codice noto come *Stored Procedure*, compilato, memorizzato e reso disponibile all'interno del database, richiamabile da qualsiasi contesto che disponga dei privilegi necessari. La procedura, a differenza della funzione, non restituisce un valore, ma esegue un'azione e modifica lo stato del sistema. Questo approccio permette di centralizzare la logica di controllo in un unico punto, riducendo di conseguenza il rischio di comportamenti inconsistenti tra i diversi flussi.

Nel progetto sono state definite quattro *Procedure*, tutte operanti sulla tabella `FLOW_MANAGER`:

- `START_PROCESS`: ha il compito di registrare l'avvio di un processo ETL all'interno della tabella `FLOW_MANAGER` attraverso l'inserimento di un record di controllo. `START_PROCESS` viene invocata subito dopo la definizione delle variabili all'interno dei *Package* ODI, prima che qualsiasi operazione di trasformazione o caricamento abbia luogo, e garantisce che l'esecuzione sia immediatamente visibile al livello di logging anche in caso di interruzione anomala. La procedura riceve in ingresso il gruppo logico del processo, il livello architetturale, il tipo di flusso, il `JOBID` sorgente e quello target, e costruisce dinamicamente un'istruzione `INSERT` tramite `EXECUTE IMMEDIATE`, che permette di eseguire durante l'esecuzione del flusso istruzioni il cui testo è costruito come stringa. I valori costanti interni, come `C_STATUS_LOAD = 1` (processo in corso) e `C_LOAD_TO_LOAD = 1` (il livello successivo deve ancora caricare), rappresentano gli stati iniziali del record appena inserito, mentre `NVL(P_SRC_JOBID, C_SRC_JOBID)` garantisce che, in assenza di un `JOBID` sorgente valido, venga utilizzato il valore tecnico di default `19000101000000`, coerente con il valore `NULL_JOBID` definito in `METADATA_MANAGER`.

```
1         CREATE OR REPLACE PROCEDURE START_PROCESS(
2             P_IDENTITY    VARCHAR2,
3             P_NUM_LEVEL   NUMBER,
4             P_GRP_NAME    VARCHAR2,
5             P_SRC_JOBID   NUMBER,
6             P_TRG_JOBID   NUMBER
7         ) IS
8             C_STATUS_LOAD    NUMBER := 1;
9             C_LOAD_TO_LOAD   NUMBER := 1;
10            C_SRC_JOBID      NUMBER := 19000101000000;
11            V_STMT           VARCHAR2(4000);
12        BEGIN
13            V_STMT := 'INSERT INTO FLOW_MANAGER (
14                            IDENTITY, NUM_LEVEL, GRP_NAME,
```

```
15                              SRC_JOBID, TRG_JOBID, STATUS,
16                              LOAD, START_DATE, END_DATE
17                          ) VALUES (
18                              '''||P_IDENTITY||''','
19                              ||P_NUM_LEVEL||','
20                              ||''''||P_GRP_NAME||''','
21                              ||NVL(P_SRC_JOBID, C_SRC_JOBID)||','
22                              ||P_TRG_JOBID||','
23                              ||C_STATUS_LOAD||','
24                              ||C_LOAD_TO_LOAD||','
25                              ||'SYSDATE, NULL)';
26              EXECUTE IMMEDIATE V_STMT;
27          END;
28
```

Listing 4.8: Definizione della procedura `START_PROCESS`

In ODI, la chiamata alla procedura avviene tramite un blocco PL/SQL all'interno di un oggetto di tipo *Procedure* definito all'interno del progetto ODI. All'interno di tale oggetto viene richiamata la *Stored Procedure* dal database, passando i valori delle variabili di progetto tramite il costrutto `odiRef.getOption()`, permettendo di leggere le opzioni dichiarate nella *Procedure* ODI e valorizzate dal *Package* in fase di esecuzione.

```
1       BEGIN
2           START_PROCESS(
3               '<%=odiRef.getOption("IDENTITY")%>',
4               <%=odiRef.getOption("NUM_LEVEL")%>,
5               '<%=odiRef.getOption("GRP_NAME")%>',
6               NULL,
7               <%=odiRef.getOption("JOBID")%>
8           );
9       END;
10
```

Listing 4.9: Richiamo di `START_PROCESS` nella *Procedure* ODI

- `END_PROCESS`: viene invocata al termine di un'esecuzione completata con successo e si occupa di aggiornare lo stato del record corrispondente nella tabella `FLOW_MANAGER`, portando il campo `STATUS` al valore `0` e valorizzando il campo `END_DATE` con la data corrente. La procedura implementa inoltre una logica di propagazione verticale tra i livelli, infatti, quando il processo terminato appartiene al livello L1 o L2 (ovvero `P_NUM_LEVEL = 1` o `P_NUM_LEVEL = 2`), viene eseguito un aggiornamento preventivo sul livello immediatamente precedente, impostando il campo `LOAD` a `0`, che segnala un livello già caricato, per tutti i record con stato diverso da `-3`, cioè non in errore. Questo meccanismo garantisce la coerenza della catena di dipendenze, segnalando che il

livello a monte ha già prodotto i dati necessari e che non è richiesto un nuovo caricamento, infine, per i processi di livello L2 o superiore, il campo `LOAD` del record corrente viene portato a `5` (`C_LOAD_NOT_LOADABLE`), indicando che quel run è da considerarsi esaurito e non ricaricabile.

```
1         CREATE OR REPLACE PROCEDURE END_PROCESS(
2             P_IDENTITY    VARCHAR2,
3             P_NUM_LEVEL   NUMBER,
4             P_GRP_NAME    VARCHAR2,
5             P_TRG_JOBID   NUMBER
6         ) IS
7             C_STATUS_DONE         NUMBER := 0;
8             C_LOAD_LOADED         NUMBER := 0;
9             C_LOAD_NOT_LOADABLE   NUMBER := 5;
10            V_STMT                VARCHAR2(4000);
11        BEGIN
12            IF P_NUM_LEVEL = 1 OR P_NUM_LEVEL = 2 THEN
13                V_STMT:='UPDATE FLOW_MANAGER
14                         SET LOAD = '||C_LOAD_LOADED||'
15                         WHERE IDENTITY = '''||P_IDENTITY||'''
16                         AND NUM_LEVEL = '||P_NUM_LEVEL||' - 1
17                         AND STATUS != -3
18                         AND GRP_NAME = '''||P_GRP_NAME||'''';
19                EXECUTE IMMEDIATE V_STMT;
20            END IF;
21            V_STMT :='UPDATE FLOW_MANAGER
22                      SET STATUS = '||C_STATUS_DONE||',
23                      END_DATE = SYSDATE
24                      WHERE IDENTITY = '''||P_IDENTITY||'''
25                      AND NUM_LEVEL = '||P_NUM_LEVEL||'
26                      AND GRP_NAME = '''||P_GRP_NAME||'''
27                      AND TRG_JOBID = '||P_TRG_JOBID;
28            EXECUTE IMMEDIATE V_STMT;
29            IF P_NUM_LEVEL >= 2 THEN
30                V_STMT:='UPDATE FLOW_MANAGER
31                         SET LOAD = '||C_LOAD_NOT_LOADABLE||'
32                         WHERE IDENTITY = '''||P_IDENTITY||'''
33                         AND NUM_LEVEL = '||P_NUM_LEVEL||'
34                         AND GRP_NAME = '''||P_GRP_NAME||'''
35                         AND TRG_JOBID = '||P_TRG_JOBID;
36                EXECUTE IMMEDIATE V_STMT;
37            END IF;
38        END;
39
```

Listing 4.10: Definizione della procedura `END_PROCESS`

In ODI:

```
1     BEGIN
2     END_PROCESS ( '<%=odiRef.getOption("IDENTITY")%>'
3                 , <%=odiRef.getOption("NUM_LEVEL")%>
4                 , '<%=odiRef.getOption("GRP_NAME")%>'
5                 , <%=odiRef.getOption("JOBID")%>
6                 ) ;
7     END ;
8
```

Listing 4.11: Richiamo di `END_PROCESS` nella *Procedure* ODI

- `END_PROCESS_ERROR`: gestisce il caso in cui l'esecuzione del processo ETL si concluda con un errore, aggiornando il record corrispondente nella `FLOW_MANAGER` impostando il campo `STATUS` a `-3`, corrispondente a `C_STATUS_ERROR`, valorizzando `END_DATE` e portando il campo `LOAD` a `5`, corrispondente a `C_LOAD_NOT_LOADABLE`. A differenza della `END_PROCESS`, non aggiorna il livello precedente della catena in quanto un processo in errore non viene considerato come un punto di avanzamento valido. Il processo di marcatura esplicita del record permette di evitare che i dati incompleti o parzialmente caricati vengano propagati ai livelli superiori, proteggendo l'integrità della pipeline.

```
1         CREATE OR REPLACE PROCEDURE END_PROCESS_ERROR(
2             P_IDENTITY    VARCHAR2,
3             P_NUM_LEVEL   NUMBER,
4             P_GRP_NAME    VARCHAR2,
5             P_TRG_JOBID   NUMBER
6         ) IS
7             C_STATUS_ERROR        NUMBER := -3;
8             C_LOAD_NOT_LOADABLE   NUMBER := 5;
9             V_STMT                VARCHAR2(4000);
10        BEGIN
11            V_STMT := 'UPDATE FLOW_MANAGER
12                       SET LOAD    = '||C_LOAD_NOT_LOADABLE||',
13                           STATUS = '||C_STATUS_ERROR||',
14                           END_DATE = SYSDATE
15                       WHERE IDENTITY  = '''||P_IDENTITY||'''
16                       AND NUM_LEVEL   = '||P_NUM_LEVEL||'
17                       AND GRP_NAME    = '''||P_GRP_NAME||'''
18                       AND TRG_JOBID   = '||P_TRG_JOBID;
19            EXECUTE IMMEDIATE V_STMT;
20        END;
21
```

Listing 4.12: Definizione della procedura `END_PROCESS_ERROR`

In ODI:

```
1     BEGIN
```

```
2     END_PROCESS_ERROR ( '<%=odiRef.getOption("IDENTITY")%>'
3                       , <%=odiRef.getOption("NUM_LEVEL")%>
4                       , '<%=odiRef.getOption("GRP_NAME")%>'
5                       , <%=odiRef.getOption("JOBID")%>
6                       ) ;
7     END ;
8
```

Listing 4.13: Richiamo di `END_PROCESS_ERROR` nella *Procedure* ODI

- `END_PROCESS_TABLE`: opera ad un livello di granularità più fine rispetto alle precedenti procedures, infatti è dedicata alla registrazione dell'esito del caricamento di una singola tabella, aggiornando la `TABLE_MANAGER` con le informazioni relative al numero di righe elaborate e al punto di sincronia raggiunto. Il suo scopo è quello di formalizzare il punto di avanzamento della lettura incrementale e viene invocata nei *Package* di gestione del delta minimizzando i rischi di perdita o duplicazione dei dati.

```
1         CREATE OR REPLACE PROCEDURE "END_PROCESS_TABLE" (
2             P_IDENTITY     VARCHAR2,
3             P_GRP_NAME     VARCHAR2,
4             P_JOBID        NUMBER,
5             P_TABLE_NAME   VARCHAR2,
6             P_NUM_ROWS     NUMBER
7         )IS
8             V_STMT VARCHAR2(4000) ;
9         BEGIN
10            V_STMT := 'INSERT INTO TABLE_MANAGER
11                (
12                    IDENTITY
13                   ,GRP_NAME
14                   ,TABLE_NAME
15                   ,JOBID
16                   ,NUM_ROWS
17                   ,LOAD_DATE
18                )
19            VALUES
20                (
21                    '''||P_IDENTITY||'''
22                   ,'''||P_GRP_NAME||'''
23                   ,'''||P_TABLE_NAME||'''
24                   , '||P_JOBID||'
25                   ,'||P_NUM_ROWS||'
26                   ,SYSDATE
27            )' ;
28        EXECUTE IMMEDIATE V_STMT ;
29        END ;
30
```

Listing 4.14: Definizione della procedura `END_PROCESS_TABLE`

In ODI:

```
1    BEGIN
2   END_PROCESS_TABLE(
3             P_IDENTITY=>'<%=snpRef.getUserExit("IDENTITY")%>'
4           , P_GRP_NAME=>'<%=snpRef.getUserExit("GRP_NAME")%>'
5           , P_JOBID=> <%=snpRef.getUserExit("JOBID")%>
6           , P_TABLE_NAME=>'<%=snpRef.getUserExit("TABLE_NAME")
    %>'
7           , P_NUM_ROWS=> <%=snpRef.getUserExit("NUM_ROWS")%>
8                ) ;
9    END ;
10
```

Listing 4.15: Richiamo di `END_PROCESS_TABLE` nella *Procedure* ODI

4.1.2 Implementazione in ODI

Una volta completata la configurazione dell'ambiente ODI e predisposte le strutture fisiche e logiche del livello L0, è stato possibile procedere con la costruzione vera e propria dei *Mapping* ETL e del sistema di orchestrazione che li gestisce. Il percorso ha seguito una sequenza ben definita e gerarchica, articolata su più livelli di astrazione: prima i *Mapping* di *staging* e delta, poi i *Package* di esecuzione di base, poi i *Load Plan* che li aggregano e infine i *Package* di coordinamento generale; questa architettura a strati è replicata in parallelo sia per le anagrafiche (dimensioni), che per i fatti.

4.1.2.1 Mappatura delle tabelle STG

La costruzione del *Mapping* di STG è stata avviata a partire dalle tabelle sorgente e di *staging* già modellate in ODI in fase di preparazione, adottando una logica implementativa che consiste nel collegamento diretto tra le due entità nell'interfaccia grafica di ODI, tracciando le corrispondenze colonna-colonna. Un elemento centrale nella costruzione del *Mapping* di STG è la valorizzazione del campo tecnico `JOBID_L0`: nella sezione *expression* della tabella target, al campo tecnico viene assegnata la variabile `JOBID_L0`, definita in fase di preparazione, che al momento del lancio dello scenario del *Mapping* viene sostituita con il valore assegnato all'esecuzione, garantendo l'identificabilità di ogni batch.

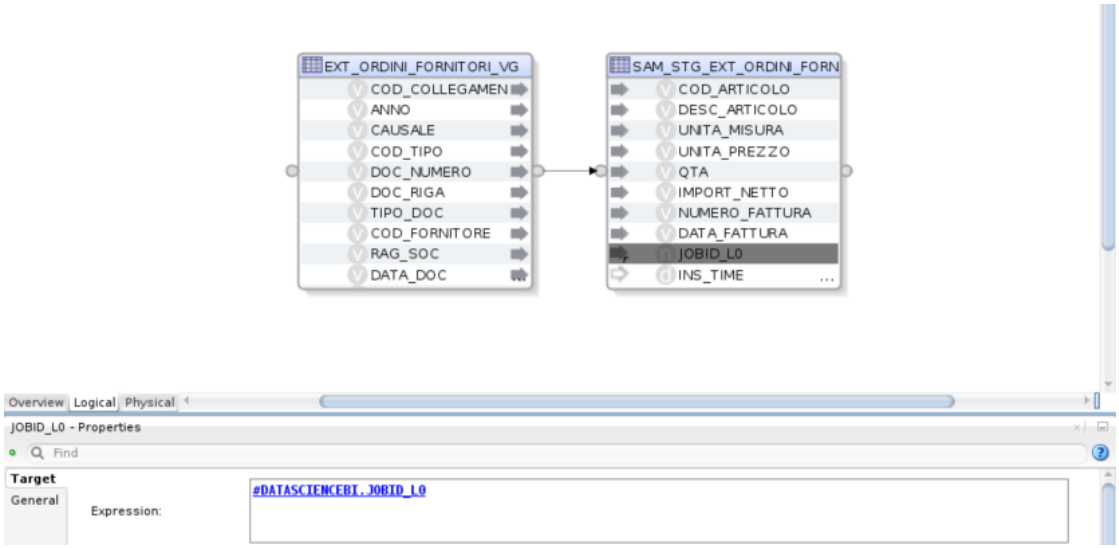

Figura 4.3: *Mapping* STG

4.1.2.2 Mappatura delle tabelle DLT

Completata la replica in *staging*, si è passati alla costruzione del *Mapping* DLT, in cui l'obiettivo è implementare un meccanismo di CDC calcolando la differenza insiemistica tra lo stato del dataset corrente (OGGI) e quello precedente (IERI), individuando i record entrati ed usciti dall'insieme informativo. Dal punto di vista implementativo, il *Mapping* DLT è stato costruito in ODI a partire da quattro istanze della medesima tabella STG, identiche, interrogate con filtri differenti così da ricostruire due istantanee temporalmente distinte del dataset: due filtri selezionano i record con `JOBID_L0` pari alla variabile `JOBID_L0` (OGGI), mentre gli altri due selezionano i record con `JOBID_L0` pari alla variabile `LAST_JOBID` (IERI). Le condizioni implementate nelle componenti FILTER di ODI sono definite come segue:

- OGGI: `<TABLE_NAME>.JOBID_L0 = #<PROGETTO>.JOBID_L0`
- IERI: `<TABLE_NAME>.JOBID_L0 = #<PROGETTO>.LAST_JOBID`

Le uscite di questi due filtri vengono poi convogliate in due componenti SET configurati in modalità MINUS. Il SET OGGI_IERI calcola la differenza tra il flusso odierno e quello precedente, restituendo tutti i record presenti nell'esecuzione corrente ma assenti in quella precedente, conservando quindi solo dati nuovi o aggiornati ed assegnando loro un `FLG_NEG = 0`.

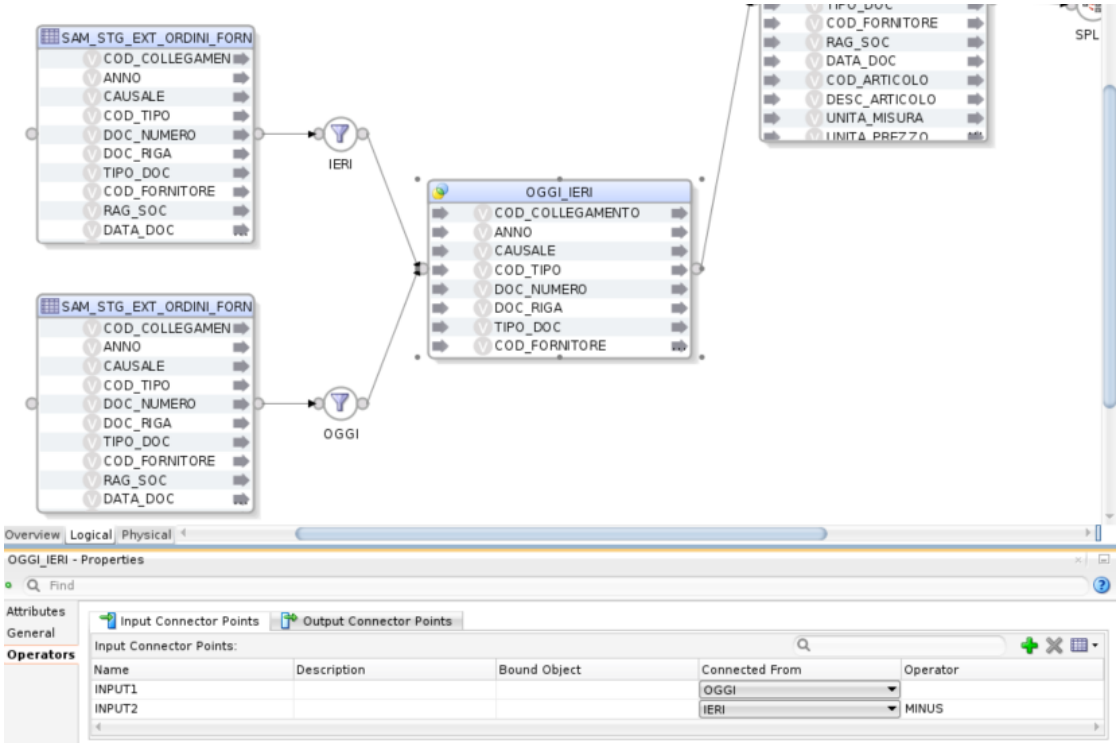

Figura 4.4: Componente SET OGGI_IERI

Il SET IERI_OGGI, invece, calcola la differenza tra il flusso precedente e quello attuale restituendo tutti i record che erano presenti nell'esecuzione precedente ma che non compaiono in quella attuale, mantenendo solo i dati eliminati o modificati dalla sorgente assegnando loro un `FLG_NEG = 1`.

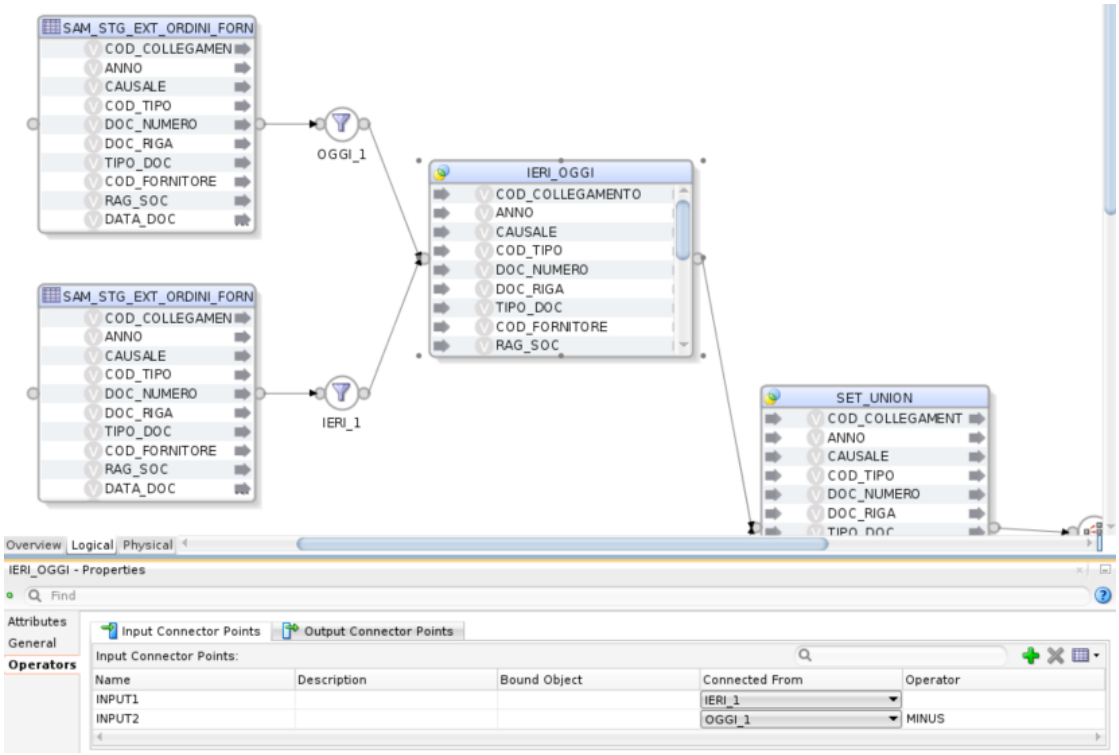

Figura 4.5: Componente SET IERI_OGGI

In entrambe le componenti SET le colonne tecniche `JOBID_L0` e `INS_TIME` vengono escluse dal confronto in quanto la loro inclusione renderebbe ogni record sempre diverso da un'esecuzione all'altra, rendendo il delta semanticamente non significativo. La logica implementata può essere formalizzata come segue:

Tabella 4.1: Logica del calcolo del delta al livello L0 e significato del campo `FLG_NEG`

Operazione insiemistica	**Significato**	**Valore `FLG_NEG`**
OGGI - IERI (OGGI_IERI)	Record presenti oggi ma non ieri: nuovi o aggiornati	0
IERI - OGGI (IERI_OGGI)	Record presenti ieri ma non oggi: eliminati o modificati	1

I due flussi vengono poi ricondotti ad un unico flusso utilizzando un terzo componente SET configurato come UNION, dove il record viene completato con l'informazione `FLG_NEG`, riconoscendo in base al ramo di provenienza se il dato è

entrato o è uscito. Il flusso finale della UNION viene distribuito da un componente SPLIT verso le tabelle di destinazione DLT e DLT_HIS, in cui tornano i campi tecnici e viene assegnato il JOBID_L0 attraverso la variabile di progetto `#<PROGETTO>.JOBID_L0`.

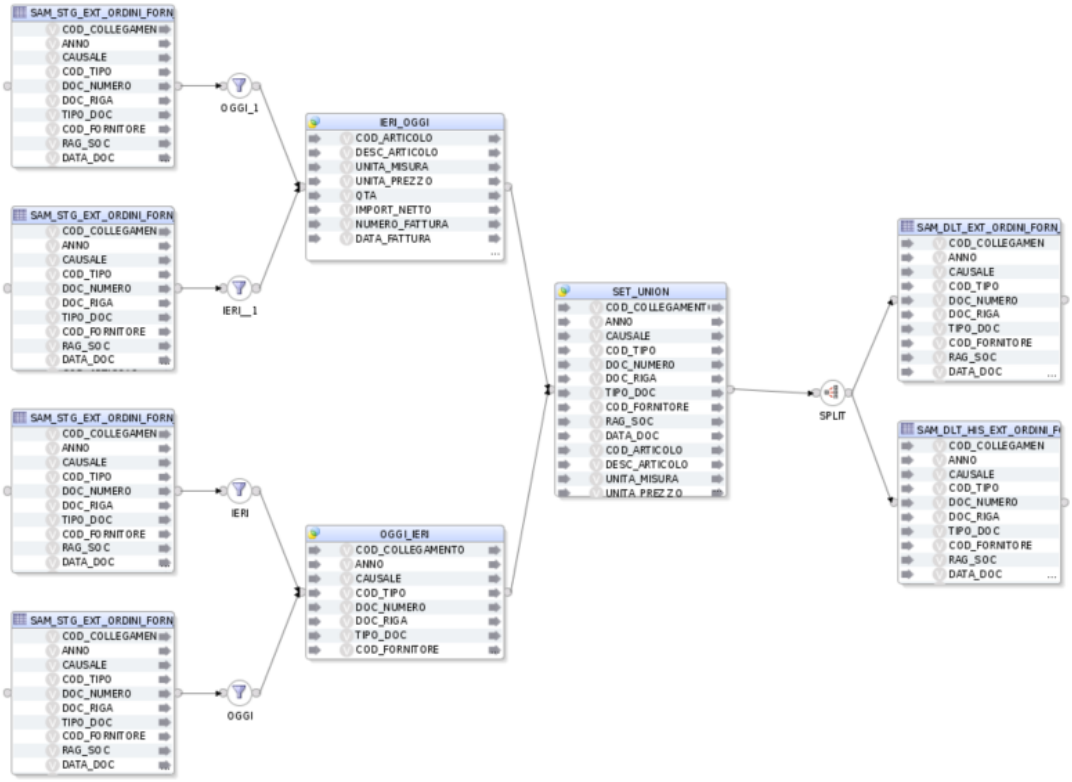

Figura 4.6: *Mapping* DLT

4.1.2.3 Orchestrazione delle mappature per il livello L0

Dopo aver eseguito sia gli scenari dei *Mapping* di tutte le STG e DLT e verificato che siano tutti andati a buon fine, si è proceduto al loro incapsulamento all'interno di strutture di orchestrazione più ampie.

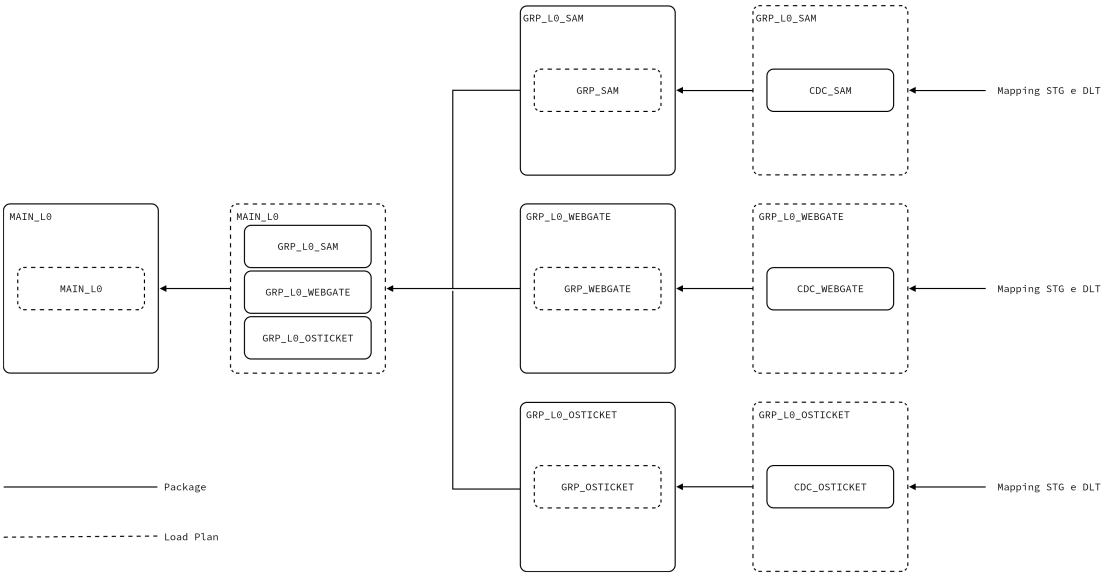

Figura 4.7: Diagramma gerarchico dell'orchestrazione L0: *Package* (linee continue) e *Load Plan* (linee tratteggiate)

Si è cominciato dai *Package* CDC, che rappresentano l'unità elementare di esecuzione del flusso per ogni singola tabella. In ODI, il *Package* è definito come una sequenza ordinata di passi, ognuno dei quali può corrispondere ad un'esecuzione di variabile, ad una procedura o al lancio di uno scenario; la connessione tra i passi può seguire la transizione di esito *ok* in caso di successo, o *ko* in caso di errore, consentendo una gestione strutturata del flusso di controllo.

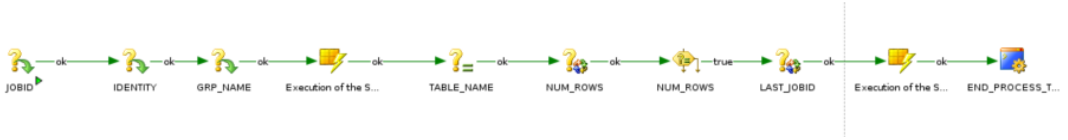

Figura 4.8: Sequenza di esecuzione del *Package* CDC per una coppia di *Mapping* STG–DLT

Il *Package* si apre con la dichiarazione delle variabili di contesto: `JOBID`, `IDENTITY` e `GRP_NAME`, configurate con il tipo *Declare Variable* in modo da non essere calcolate localmente, ma ereditare il valore assegnato dal livello padre dell'orchestrazione, garantendo coerenza nella propagazione dei parametri esecutivi. A seguire viene eseguito lo scenario del *Mapping* STG dell'entità presa in considerazione, che popola la tabella di *staging* con i dati della sorgente, dopodiché viene valorizzata la variabile `TABLE_NAME` tramite un passo di tipo *Set Variable*, in cui

viene impostato il nome fisico della tabella corrente. Successivamente viene aggiornata la variabile `NUM_ROWS` con un passo di tipo *Refresh Variable*, che interroga la tabella di gestione dei metadati per sapere quante righe sono state caricate nella STG durante questa esecuzione, e viene quindi valutata tramite un passo *Evaluate Variable*: nel caso in cui il valore sia maggiore o uguale a uno, la transizione è *true* e il flusso prosegue, altrimenti, quindi nel caso in cui non siano stati caricati record, il flusso si interrompe, evitando di eseguire il calcolo del delta su una tabella vuota. In caso di esito positivo, viene aggiornata la variabile `LAST_JOBID` tramite un ulteriore *Refresh Variable*, che recupera l'identificativo dell'ultima esecuzione andata a buon fine, rendendo disponibile questo valore per i filtri IERI del *Mapping* DLT; vengono quindi eseguiti lo scenario DLT, che calcola e scrive il delta sulle tabelle DLT e DLT_HIS, e infine la procedura `END_PROCESS_TABLE`, che registra nella tabella dei metadati l'avanzamento e l'esito dell'esecuzione per questa specifica tabella.

Una volta costruito il *Package* CDC per ogni coppia STG–DLT, e quindi per ogni tabella della sorgente, il passo successivo consiste nell'aggregare le esecuzioni all'interno di un *Load Plan* intermedio: `LO_ANAGRAFICHE_<SORGENTE>`, per le anagrafiche (dimensioni) e `LO_FATTI_<SORGENTE>` per i fatti. In ODI, un *Load Plan* è un oggetto di orchestrazione che consente di eseguire scenari in sequenza o in parallelo, offrendo funzionalità avanzate di ripartenza e gestione degli errori; in questo caso risponde alla necessità di separare il livello di esecuzione tabellare dal coordinamento per sorgente, inserendo in serie gli scenari dei *Package* CDC di tutte le tabelle (anagrafiche e fatti separatamente), in modo che l'esecuzione proceda tabella per tabella in modo ordinato e controllato.

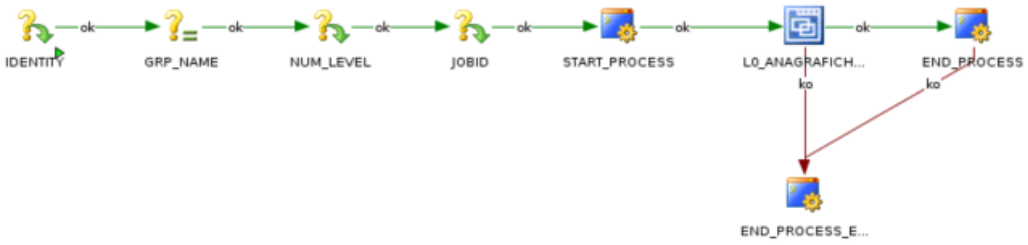

Figura 4.9: Sequenza di esecuzione del *Package* GRP per le tabelle anagrafiche in L0

Ottenuti i *Load Plan* intermedi, si passa alla costruzione dei *Package* GRP, uno per sorgente e per dominio logico, che rappresentano il livello di coordinamento dell'intero gruppo di tabelle di una sorgente. La sequenza di passi in un *Package* GRP si apre con la dichiarazione della variabile `IDENTITY` di tipo *Declare Variable*, seguita dalla variabile `GRP_NAME` configurata con tipo *Set Variable* e valorizzata con

l'identificativo del gruppo, in generale `L0_<SORGENTE>`, successivamente avvengono le dichiarazioni di `NUM_LEVEL` e `JOBID` come variabili di tipo *Declare Variable*. Il cuore operativo del *Package* GRP è racchiuso tra le due *Procedure* `START_PROCESS`, che registra l'apertura del processo nel sistema di metadati, ed `END_PROCESS`, che ne registra la chiusura con successo; tra le due viene inserito il *Load Plan* intermedio costruito in precedenza, che eseguirò tutti i *Package* CDC della sorgente. La gestione degli errori viene garantita collegando sia il *Load Plan* che la procedura `END_PROCESS` alla procedura `END_PROCESS_ERROR` tramite transizioni *ko*, in modo che qualunque fallimento venga tracciato e registrato correttamente prima che il flusso si interrompa.

Completati tutti i *Package* GRP, si procede alla costruzione del *Load Plan* di livello superiore, denominato `L0_ANAGRAFICHE`, per le dimensioni, e `L0_FATTI` per i fatti. In questo *Load Plan* vengono inseriti in serie gli scenari dei *Package* GRP di tutte le sorgenti del dominio considerato, garantendo che l'esecuzione proceda nell'ordine corretto attraverso l'intera catena delle sorgenti.

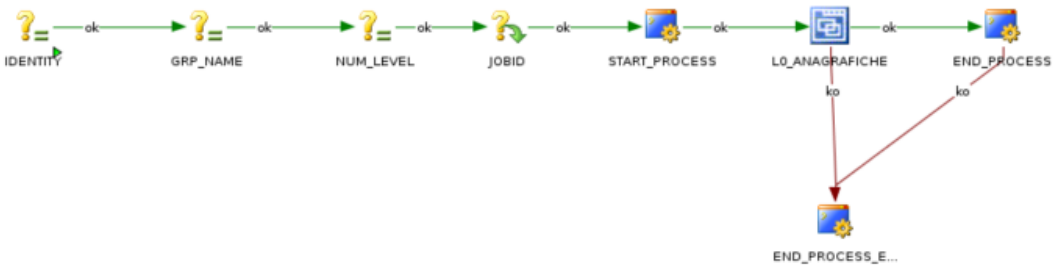

Figura 4.10: Sequenza di esecuzione del *Package* MAIN per le tabelle anagrafiche in L0

Al vertice della gerarchia di orchestrazione si alloca infine il *Package* `MAIN_L0`, che funge da punti di ingresso dell'orchestrazione dell'intero livello L0. A differenza dei livelli sottostanti, dove le variabili vengono ereditate dall'alto, qui le variabili chiave vengono inizializzate per la prima volta. In particolare, `IDENTITY`, `GRP_NAME` e `NUM_LEVEL` sono configurate come *Set Variable*: `IDENTITY` assume il valore `ANAGRAFICHE` o `FATTI` in funzione del dominio logico, `GRP_NAME` è valorizzata con `ANAGRAFICHE_Ln` o `FATTI_Ln` per identificare dominio e livello del framework, mentre `NUM_LEVEL` viene impostata a `0`, `1` o `2` a seconda del livello del framework. La variabile `JOBID` viene dichiarata senza valorizzazione esplicita, mettendola in tipo *Declare Variable*, poiché il suo valore sarà definito nella *Procedure* `START_PROCESS` che segue immediatamente. Infine, dopo l'esecuzione del *Load Plan*, il flusso si chiude con la *Procedure* `END_PROCESS`, con la medesima gestione degli errori già descritta per il *Package* GRP, attraverso la transizione *ko* verso `END_PROCESS_ERROR`.

La struttura risultante garantisce un sistema di orchestrazione in cui ogni livello sa esattamente cosa deve eseguire, come propagare le variabili verso il basso e come gestire i propri errori, rendendo il framework facilmente estendibile a nuove sorgenti o domini senza dover riprogettare l'intera catena di esecuzione.

4.2 Livello L1 - Operational Data Storage

4.2.1 Fase di preparazione all'implementazione in ODI

La fase di preparazione all'implementazione in ODI per il livello L1 si distingue da quella del livello L0 per una maggiore attenzione alla struttura semantica dei dati. Mentre L0 era orientato alla replica fedele e al tracciamento delle variazioni tra l'esecuzione presente e quella precedente, l'L1 ha l'obiettivo di consolidare l'informazione proveniente dal layer di *staging* in una forma certificata, consistente e aggiornabile nel tempo.

A differenza di quanto avvenuto nel livello L0, nella preparazione di L1 non è stato necessario introdurre nuove variabili di orchestrazione né predisporre ulteriori *Procedure* o funzioni, pertanto, il lavoro preparatorio, si è concentrato esclusivamente sulla definizione e sulla creazione delle strutture fisiche necessarie al livello, ovvero le tabelle di OK, ODS, MDM e OUT.

Le tabelle di OK rappresentano un'area intermedia all'interno di L1, popolata dai record che hanno superato con esito positivo i controlli di qualità applicati sui dati provenienti dalla DLT, e il loro compito è quello di selezionare, per ciascuna chiave logica, il record valido del ciclo corrente attraverso una logica di deduplicazione basata sul costrutto `ROW_NUMBER()`. Queste presentano un campo `JOBID_L1`, che identifica il ciclo di caricamento che ha generato i record presenti, riprendendo invece tutto il resto delle informazioni direttamente dalla tabella DLT, inoltre non è previsto alcun vincolo di chiave primaria, infatti la deduplicazione e la selezione del record corretto vengono gestite interamente a livello di *Mapping* in ODI, rendendo superflua, e potenzialmente limitante, la dichiarazione di un constraint fisico in questa fase.

```
1     CREATE TABLE "L1"."SAM_OK_EXT_ORDINI_FORN_VG"
2        ("JOBID_L1" NUMBER NOT NULL ENABLE,
3       "COD_COLLEGAMENTO" VARCHAR2(400 CHAR),
4       "ANNO" VARCHAR2(400 CHAR),
5       "CAUSALE" VARCHAR2(400 CHAR),
6       "COD_TIPO" VARCHAR2(400 CHAR),
7       "DOC_NUMERO" VARCHAR2(400 CHAR),
8       "DOC_RIGA" VARCHAR2(400 CHAR),
9       "TIPO_DOC" VARCHAR2(400 CHAR),
10      "COD_FORNITORE" VARCHAR2(400 CHAR),
11      "RAG_SOC" VARCHAR2(400 CHAR),
```

```
12        "DATA_DOC" VARCHAR2(400 CHAR),
13        "COD_ARTICOLO" VARCHAR2(400 CHAR),
14        "DESC_ARTICOLO" VARCHAR2(400 CHAR),
15        "UNITA_MISURA" VARCHAR2(400 CHAR),
16        "UNITA_PREZZO" VARCHAR2(400 CHAR),
17        "QTA" VARCHAR2(400 CHAR),
18        "IMPORT_NETTO" VARCHAR2(400 CHAR),
19        "NUMERO_FATTURA" VARCHAR2(400 CHAR),
20        "DATA_FATTURA" VARCHAR2(400 CHAR),
21        "FLG_NEG" NUMBER NOT NULL ENABLE,
22        "INS_TIME" DATE DEFAULT SYSDATE
23         ) SEGMENT CREATION IMMEDIATE
24        PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS
      NOLOGGING
25        STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
      2147483645
26        PCTINCREASE 0
27        BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
      DEFAULT)
28        TABLESPACE "TBS_L1"
29        PARTITION BY RANGE ("JOBID_L1") INTERVAL (1000000)
30       (PARTITION "STORICO"  VALUES LESS THAN (20190101000000)
      SEGMENT CREATION IMMEDIATE
31        PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
32       NOCOMPRESS NOLOGGING
33        STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
      2147483645
34        PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
35        BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
      DEFAULT)
36        TABLESPACE "TBS_L1" ) ;
```

Listing 4.16: Tabella OK esempio

Le tabelle ODS invece costituiscono la destinazione di consolidamento del livello L1 e rappresentano la vista aggiornata e certificata del dato operativo aziendale. Rispetto alle tabelle di OK, le ODS non sono progettate per una scrittura monodirezionale, infatti ogni record è identificato univocamente attraverso le proprie chiavi naturali e può essere sia inserito per la prima volta che aggiornato nei cicli di elaborazione successivi. Nelle ODS il campo `JOBID_L1` viene sostituito da due campi: `JOBID_L1_INS`, che registra il ciclo in cui il record è stato inserito per la prima volta, e `JOBID_L1_UPD`, che viene aggiornato ad ogni ciclo successivo in cui il record subisce una modifica. In questo modo il dato acquisisce una dimensione temporale completa attraverso la quale non è più sufficiente sapere che un record esiste, ma diventa possibile, e necessario, sapere quando è entrato nel sistema e quando è stato modificato l'ultima volta, informazioni che nei livelli superiori del framework costituiscono il presupposto delle logiche di evoluzione e

di controllo del dato. La seconda differenza rispetto alle OK è nell'introduzione del campo `UPD_TIME`, affiancato al già presente `INS_TIME`, che viene aggiornato ad ogni ciclo in cui viene modificato il dato, permettendo di datare con esattezza l'ultimo intervento effettuato. Infine, la differenza più rilevante rispetto alle tabelle di OK, risiede nella presenza di un blocco `CONSTRAINT` in coda alla DDL, attraverso il quale viene dichiarata esplicitamente la chiave primaria della tabella a partire dalle chiavi naturali dell'entità, individuate per ciascuna tabella tramite un'analisi condotta su SQL Developer, poiché non esiste un meccanismo automatico che determini quale combinazione di colonne garantisca l'univocità del record nella sorgente. Questa dichiarazione non è un dettaglio formale; sarà proprio il constraint di chiave primaria a guidare ODI nel determinare, ciclo per ciclo, se un record in ingresso debba essere inserito ex novo oppure utilizzato per aggiornare un record già esistente nella tabella, abilitando la logica *Incremental Update* che caratterizza il comportamento delle ODS.

```
1     CREATE TABLE "L1"."SAM_ODS_EXT_ORDINI_FORN_VG"
2        ("JOBID_L1_INS" NUMBER NOT NULL ENABLE,
3       "JOBID_L1_UPD" NUMBER NOT NULL ENABLE,
4       "COD_COLLEGAMENTO" VARCHAR2(400 CHAR),
5       "ANNO" VARCHAR2(400 CHAR),
6       "CAUSALE" VARCHAR2(400 CHAR),
7       "COD_TIPO" VARCHAR2(400 CHAR),
8       "DOC_NUMERO" VARCHAR2(400 CHAR),
9       "DOC_RIGA" VARCHAR2(400 CHAR),
10      "TIPO_DOC" VARCHAR2(400 CHAR),
11      "COD_FORNITORE" VARCHAR2(400 CHAR),
12      "RAG_SOC" VARCHAR2(400 CHAR),
13      "DATA_DOC" VARCHAR2(400 CHAR),
14      "COD_ARTICOLO" VARCHAR2(400 CHAR),
15      "DESC_ARTICOLO" VARCHAR2(400 CHAR),
16      "UNITA_MISURA" VARCHAR2(400 CHAR),
17      "UNITA_PREZZO" VARCHAR2(400 CHAR),
18      "QTA" VARCHAR2(400 CHAR),
19      "IMPORT_NETTO" VARCHAR2(400 CHAR),
20      "NUMERO_FATTURA" VARCHAR2(400 CHAR),
21      "DATA_FATTURA" VARCHAR2(400 CHAR),
22      "FLG_NEG" NUMBER NOT NULL ENABLE,
23      "INS_TIME" DATE DEFAULT SYSDATE,
24      "UPD_TIME" DATE DEFAULT SYSDATE,
25       CONSTRAINT "PK_SAM_ORDINI_FORN" PRIMARY KEY ("DOC_NUMERO",
    "DOC_RIGA", "DATA_DOC", "DATA_FATTURA")
26      USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
    STATISTICS
27      STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
    2147483645
28      PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
```

```
29        BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
      DEFAULT)
30        TABLESPACE "TBS_L1"  ENABLE
31         ) SEGMENT CREATION IMMEDIATE
32        PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
33        NOCOMPRESS NOLOGGING
34        STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
      2147483645
35        PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
36        BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
      DEFAULT)
37        TABLESPACE "TBS_L1" ;
```

Listing 4.17: Tabella ODS esempio

Le tabelle MDM rappresentano il punto del framework in cui i dati anagrafici, fino a questo momento esistenti separatamente nelle diverse sorgenti, vengono ricondotti in un'unica rappresentazione coerente. In questo progetto le fonti coinvolte nell'MDM sono solo SAM e Webgate, in quanto OS Ticket, a differenza delle altre due, è composta esclusivamente da tabelle dei fatti e non espone quindi alcuna anagrafica da riconciliare o consolidare. SAM e Webgate descrivono le stesse entità di business con strutture, codici e denominazioni parzialmente diverse, ed è proprio nell'MDM che vengono riconciliate per produrre un unico record master per ciascuna entità, indipendentemente da quante sorgenti la descrivano. La progettazione delle tabelle MDM prende avvio da un'analisi comparativa delle dimensioni esposte dai due sistemi, volta a individuare quali campi, pur provenendo da sistemi diversi, rappresentassero lo stesso contenuto informativo. Per i campi in comune è stata adottata una regola di priorità esplicita, infatti la denominazione utilizzata è quella corrispondente al campo proveniente dalla sorgente Webgate, che costituisce la fonte di riferimento primaria per gli attributi condivisi: questa priorità non si limita alla nomenclatura dei campi, ma si estende anche alla definizione della chiave primaria della tabella, che viene mutuata direttamente da quella della dimensione WebGate, garantendo così che il modello risultante sia deterministico ed evitando che la stessa informazione venga trattata con nomi o semantiche diverse a seconda del ciclo di elaborazione. Rispetto alle strutture viste per le tabelle di OK e ODS, le tabelle MDM introducono due elementi strutturali del tutto nuovi. Il primo è la SK, una chiave artificiale, detta anche surrogata, numerica che identifica in modo univoco l'entità master consolidata, indipendentemente dal sistema di origine che l'ha generata. La SK risponde alla necessità di un identificatore univoco stabile, poiché spesso le chiavi naturali delle sorgenti sono composte da più campi e possono variare nel tempo o essere incompatibili tra sistemi diversi, rendendo le operazioni di join nei livelli superiori più onerose e fragili. La sostituzione delle chiavi naturali con un intero progressivo semplifica il modello, accelera le query e garantisce stabilità nel tempo in quanto, qualora una chiave naturale dovesse

cambiare in una sorgente, la chiave surrogata della MDM rimarrebbe invariata, preservando la coerenza storica del dato. Dal punto di vista implementativo, la SK non viene generata internamente a ODI, ma è affidata ad una *Sequence* definita direttamente sul database.

```
1     CREATE SEQUENCE L1.MDM_<DIMENSIONE>_SEQ
2     INCREMENT BY 1
3     START WITH 1
4     NOCACHE
5     NOCYCLE;
```

Listing 4.18: Query generica per la creazione di una sequence

La *Sequence* viene poi collegata in ODI attraverso la creazione di un oggetto dedicato che richiama quella definita sul database, e il suo valore viene richiamato nel *Mapping* esclusivamente al momento dell'inserimento, poiché la SK, una volta assegnata, non deve mai essere aggiornata, essa infatti rappresenta l'identità stabile dell'entità master e modificarla equivarrebbe a perdere la continuità storica del record. Il secondo elemento distintivo è il campo `FLG_SRC`, che traccia la provenienza di ciascun record master indicando se il dato è alimentato esclusivamente da Webgate, esclusivamente da SAM oppure da entrambe le sorgenti contemporaneamente. Questo campo viene valorizzato attraverso una funzione personalizzata definita direttamente in ODI, la cui logica è espressa tramite un'istruzione `CASE WHEN` che valuta la presenza o l'assenza del valore convenzionale `'ND'`, ovvero il marcatore del campo '*Not Defined*' già introdotto nel livello L0, sui campi provenienti dalle due sorgenti.

```
1 -- Syntax: MDM_FLG_SRC($(Parametro1), $(Parametro2))
2 -- Implementazione Oracle:
3 CASE
4     WHEN ($(Parametro1) != 'ND' AND $(Parametro2) != 'ND') THEN '
    WG_SAM'
5     WHEN ($(Parametro2) != 'ND') THEN 'SAM'
6     WHEN ($(Parametro1) != 'ND') THEN 'WG'
7     ELSE 'WG_SAM'
8 END
```

Listing 4.19: Definizione della funzione ODI `MDM_FLG_SRC` per la valorizzazione del campo `FLG_SRC`

Dalla definizione della funzione possiamo osservare che `Parametro1` corrisponde al campo proveniente da Webgate e `Parametro2` al campo corrispondente proveniente da SAM, se entrambi sono valorizzati il record è alimentato da entrambe le sorgenti (`WG_SAM`), se solo uno dei due è presente il flag assume il valore della sorgente attiva, mentre nel caso residuale in cui entrambi risultino non disponibili viene comunque assegnato `WG_SAM` come valore di default conservativo. Il campo `FLG_SRC` non incide sulla logica di consolidamento del dato, ma costituisce uno strumento di

tracciabilità indispensabile dal momento che permette di sapere, per ciascuna entità master, da quale combinazione di sorgenti proviene il suo contenuto informativo, fondamentale sia in fase di verifica che in caso di anomalie o disallineamenti tra i sistemi di origine.

```
1     CREATE TABLE "L1"."MDM_SALESMEN"
2        ("JOBID_L1_INS" NUMBER NOT NULL ENABLE,
3       "JOBID_L1_UPD" NUMBER NOT NULL ENABLE,
4       "SALESMEN_SK" NUMBER NOT NULL ENABLE,
5       "LASTSTATUSCHANGE" DATE,
6       "ACTIVE" VARCHAR2(400 CHAR),
7       "ID" VARCHAR2(400 CHAR),
8       "DESCRIPTION" VARCHAR2(400 CHAR),
9       "TERRITORYBUSINESSMANAGERID" VARCHAR2(400 CHAR),
10      "REGIONID" VARCHAR2(400 CHAR),
11      "AGENCYID" VARCHAR2(400 CHAR),
12      "AREAID" VARCHAR2(400 CHAR),
13      "SALESMANAGERID" VARCHAR2(400 CHAR),
14      "EMAILADDRESS" VARCHAR2(400 CHAR),
15      "FLG_NEG" NUMBER NOT NULL ENABLE,
16      "FLG_SRC" VARCHAR2(400 CHAR),
17      "INS_TIME" DATE DEFAULT SYSDATE,
18      "UPD_TIME" DATE DEFAULT SYSDATE,
19       CONSTRAINT "MDM_SALESMEN" PRIMARY KEY ("ID")
20      USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
    STATISTICS
21      STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
    2147483645
22      PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
23      BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
    DEFAULT)
24      TABLESPACE "TBS_L1"  ENABLE
25       ) SEGMENT CREATION IMMEDIATE
26      PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
27     NOCOMPRESS LOGGING
28      STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
    2147483645
29      PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
30      BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
    DEFAULT)
31      TABLESPACE "TBS_L1" ;
```

Listing 4.20: Tabella MDM esempio

Le tabelle OUT rappresentano l'ultimo stadio della pipeline nel livello L1, prima della pubblicazione in L2 e, a differenza delle tabelle trattate fino ad ora, non sono tabelle di consolidamento del dato, ma di preparazione del modello dati per il livello di pubblicazione. Il loro obiettivo principale è estrapolare, per ogni entità, le chiavi surrogate delle tabelle MDM collegate tramite operazioni di join,

producendo una struttura che il tool di visualizzazione, che nel nostro caso sarà Power BI, possa interrogare direttamente per chiave surrogata, senza dover risalire alle chiavi naturali. La costruzione delle tabelle OUT sul database avviene, rispetto alle MDM, ripristinando un singolo campo `JOBID_L1` in sostituzione della coppia `JOBID_L1_INS` e `JOBID_L1_UPD`, viene poi rimosso il campo `UPD_TIME` mantenendo solo `INS_TIME`, viene eliminato il blocco `CONSTRAINT` di chiave primaria, e la SK della MDM viene rinominata in PK, a segnalare che, nelle OUT, quella colonna assume il ruolo di identificatore fisico della riga senza però essere vincolata da un constraint dichiarato. L'elemento caratteristico della costruzione delle OUT è l'espansione dei campi che contengono identificativi di entità esterne, infatti, ogni campo della MDM che rappresenta l'ID di un'altra entità viene sostituito da un gruppo di tre colonne correlate: la SK dell'entità referenziata, con il tipo `NUMBER`, il suo identificativo naturale, con il tipo `VARCHAR2` e un attributo descrittivo, solitamente la denominazione dell'entità stessa. Questa espansione rende la OUT autosufficiente per il livello di pubblicazione, mettendo già a disposizione tutte le informazioni necessarie, senza obbligare i livelli a valle a risalire la catena delle chiavi naturali attraverso join aggiuntivi. Nel caso in cui una tabella MDM non contenga identificativi di altre entità esterne, la costruzione della OUT rimane invariata nelle regole di trasformazione sopra descritte, ma non viene effettuata alcuna espansione poiché non vi sono relazioni esterne da materializzare.

```
1     CREATE TABLE "L1"."OUT_SALESMEN"
2         ("JOBID_L1" NUMBER NOT NULL ENABLE,
3       "SALESMEN_PK" NUMBER NOT NULL ENABLE,
4       "LASTSTATUSCHANGE" DATE,
5       "ACTIVE" VARCHAR2(400 CHAR),
6       "ID" VARCHAR2(400 CHAR),
7       "DESCRIPTION" VARCHAR2(400 CHAR),
8       "TERRITORYBUSINESSMANAGER_SK" NUMBER,
9       "TERRITORYBUSINESSMANAGER_ID" VARCHAR2(400 CHAR),
10      "TERRITORYBUSINESSMANAGER_DESCRIPTION" VARCHAR2(400 CHAR),
11      "REGION_SK" NUMBER,
12      "REGION_ID" VARCHAR2(400 CHAR),
13      "REGION_DESCRIPTION" VARCHAR2(400 CHAR),
14      "AGENCY_SK" NUMBER,
15      "AGENCY_ID" VARCHAR2(400 CHAR),
16      "AGENCY_DESCRIPTION" VARCHAR2(400 CHAR),
17      "AREAMANAGER_SK" NUMBER,
18      "AREAMANAGER_ID" VARCHAR2(400 CHAR),
19      "AREAMANAGER_DESCRIPTION" VARCHAR2(400 CHAR),
20      "SALESMANAGER_SK" NUMBER,
21      "SALESMANAGER_ID" VARCHAR2(400 CHAR),
22      "SALESMANAGER_DESCRIPTION" VARCHAR2(400 CHAR),
23      "EMAILADDRESS" VARCHAR2(400 CHAR),
24      "FLG_NEG" NUMBER NOT NULL ENABLE,
25      "FLG_SRC" VARCHAR2(400 CHAR),
```

```
26        "INS_TIME" DATE DEFAULT SYSDATE
27         ) SEGMENT CREATION IMMEDIATE
28        PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
29       NOCOMPRESS LOGGING
30        STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
      2147483645
31        PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
32        BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
      DEFAULT)
33        TABLESPACE "TBS_L1" ;
```

Listing 4.21: Tabella OUT esempio

4.2.2 Implementazione in ODI

Con le strutture fisiche predisposte e il modello ODI configurato per il livello L1, è stato possibile procedere con la costruzione dei *Mapping* che compongono la pipeline di elaborazione di cui fanno parte il *Mapping* OK, responsabile della selezione qualificata del dato dal delta, il *Mapping* ODS, che consolida il dato certificato nella sua forma stabile e storicizzata, il *Mapping* MDM, che riconcilia le anagrafiche provenienti da sorgenti diverse in un'unica rappresentazione aziendale e infine il *Mapping* OUT, che prepara il dato arricchito delle chiavi surrogate per la pubblicazione nel livello L2.

L'architettura di orchestrazione ricalca quella progettata nel livello L0, con la differenza che nei *Load Plan* di `GRP`, non viene richiamato un *Package* CDC, ma direttamente gli scenari compilati dei *Mapping* ed è divisa in tre blocchi: uno per i *Mapping* OK e ODS, declinato per sorgente, uno per i *Mapping* MDM e uno per i *Mapping* OUT. Questa struttura si replica in parallelo sia per le dimensioni che per i fatti, con l'unica eccezione che per le tabelle dei fatti il blocco MDM non è previsto, in quanto questo riguarda solo le entità anagrafiche e non ha applicazione sulle tabelle dei fatti.

4.2.2.1 Mappatura delle tabelle OK

La costruzione del *Mapping* di OK consiste nel trasferimento di dati dalla tabella DLT, utilizzata come sorgente dal livello L0, alla tabella corrispondente di OK. Il primo componente del *Mapping* è un filtro al quale è stata applicata la condizione `<TABELLA_DLT>.JOBID_L0 > #<PROGETTO>.LAST_DATE_READ`, garantendo che vengano processati esclusivamente i record generati da esecuzioni successive all'ultima già elaborata a livello L1, rendendo il *Mapping* incrementale per costruzione. L'uscita di questo primo filtro viene connessa ad un componente *Expression*, che in ODI agisce come area di trasformazione intermedia in cui è possibile ridefinire il set di colonne del flusso. In questa componente vengono rimossi i campi tecnici

JOBID_L0 e INS_TIME, che non devono partecipare alla logica di deduplicazione, e viene aggiunto il campo calcolato ROWS di tipo numerico, posizionato come prima colonna del flusso.

Prima di poter valorizzare questo campo è però necessario conoscere le chiavi naturali della tabella, perché sono proprio queste a definire la granularità della partizione su cui la funzione analitica opera. L'individuazione delle chiavi naturali avviene su SQL Developer attraverso l'utilizzo di tre query progressive, tutte focalizzate sull'ultima esecuzione disponibile nella tabella STG, essendo questa una fedele trasposizione dei dati direttamente dalla sorgente. La prima query stabilisce il conteggio di riferimento, ovvero il numero totale di righe distinte, prendendo in considerazione solo i dati elaborati nell'ultima esecuzione.

```
1    SELECT COUNT(*)
2    FROM <TABELLA_STG>
3    WHERE JOBID_L0 = (SELECT MAX(JOBID_L0) FROM <TABELLA_STG>);
```

Listing 4.22: Prima query per l'identificazione delle chiavi naturali

La seconda query raggruppa le righe per tutti i campi informativi, escludendo JOBID_L0 e INS_TIME, contando quante combinazioni risultino univoche, con l'obiettivo di ridurre progressivamente il set di campi fino a trovare la combinazione minima la cui cardinalità distinta pareggi il conteggio di riferimento: i campi che rimangono in quel punto costituiscono le chiavi naturali della tabella.

```
1    SELECT COUNT(*)
2    FROM (
3            SELECT
4                <TUTTI I CAMPI ESCLUSI JOBID_L0 E INS_TIME>,
5                COUNT(*)
6            FROM <TABELLA_STG>
7            WHERE JOBID_L0 = (SELECT MAX(JOBID_L0) FROM <
     TABELLA_STG>)
8            GROUP BY <TUTTI I CAMPI ESCLUSI JOBID_L0 E INS_TIME>
9            HAVING COUNT(*)=1
10       );
```

Listing 4.23: Seconda query per l'identificazione delle chiavi naturali

Come prova del nove, la terza query verifica che con la combinazione identificata non esistano righe duplicate, restituendo un insieme vuoto. Va sottolineato che questa terza interrogazione può essere utilizzata come criterio discriminante solo quando effettivamente restituisce righe: se la tabella non presenta mai duplicati nemmeno con tutti i campi inclusi, il risultato sarà sempre vuoto indipendentemente

dalla combinazione scelta, e in quel caso ci si affida esclusivamente al confronto tra le prime due query.

```
1     SELECT
2         <TUTTI I CAMPI ESCLUSI JOBID_L0 E INS_TIME>,
3         COUNT(*)
4     FROM <TABELLA_STG>
5     WHERE JOBID_L0 = (SELECT MAX(JOBID_L0) FROM <TABELLA_STG>)
6     GROUP BY <TUTTI I CAMPI ESCLUSI JOBID_L0 E INS_TIME>
7     HAVING COUNT(*)>1;
```

Listing 4.24: Terza query per l'identificazione delle chiavi naturali

Identificate le chiavi, il campo `ROWS` nell'*Expression* viene valorizzato con la funzione analitica `ROW_NUMBER()`, che numera i record a partire da uno all'interno di ogni partizione definita dalle chiavi naturali. L'ordinamento interno alla partizione segue tre criteri in cascata: `JOBID_L0` discendente, in modo da portare in cima il record più recente, `FLG_NEG` ascendente, per privilegiare un inserimento rispetto ad una cancellazione, e infine `INS_TIME` discendente come criterio finale di ordinamento. Il campo `ROWS` viene poi passato ad un secondo filtro, dove viene imposto che il suo valore sia pari ad `1`, che mantiene esclusivamente il record di testa per ogni chiave, eliminando versioni superate o duplicati dello stesso dato.

```
1     ROW_NUMBER() OVER(
2         PARTITION BY <TABELLA_DLT>.<CHIAVE>
3         ORDER BY
4             <TABELLA_DLT>.JOBID_L0 DESC,
5             <TABELLA_DLT>.FLG_NEG ASC,
6             <TABELLA_DLT>.INS_TIME DESC
7     )
```

Listing 4.25: Definizione `ROW_NUMBER` nel contesto del progetto

Il secondo filtro viene infine connesso alla tabella OK come destinazione, nel cui campo `JOBID_L1` viene inserita la variabile `#<PROGETTO>.JOBID_L1`, mentre `INS_TIME` rimane deliberatamente vuoto in quanto la tabella è stata creata con il valore di default alla data corrente, e Oracle provvede autonomamente alla sua valorizzazione al momento dell'inserimento fisico. Per ogni colonna informativa della tabella target viene poi configurata nell'*Expression* una funzione `NVL` che sostituisce eventuali valori nulli con un valore convenzionale letto a runtime dalla tabella `METADATA_MANAGER` tramite le variabili di progetto:

- `NULL_CHAR` per i singoli caratteri;
- `NULL_DATE` per le date;
- `NULL_NUMBER` per i campi numerici;

- `NULL_JOBID` per i campi tecnici identificativi;
- `NULL_VARCHAR` per i codici e i campi alfanumerici;
- `NULL_DESC` per i campi alfanumerici lunghi, come ad esempio descrizioni testuali.

Per la sorgente SAM, su un sottoinsieme specifico di campi definiti nella specifica applicativa, la funzione `NVL` da sola non è sufficiente, infatti, prima di applicare la gestione del nullo, è necessario eseguire delle trasformazioni preliminari di formato. Questa scelta è coerente con il principio architetturale del framework, per cui il livello L0 replica il dato senza alterarlo e il livello L1 si fa carico di renderlo corretto e compatibile con il modello del DWH. I campi soggetti a queste trasformazioni aggiuntive sono identificati puntualmente nella specifica applicativa della sorgente e le tipologie di conversione che ricorrono sono quattro:

```
1     -- 1.Trasformazione del campo data come numero di giorni dall'
      origine 30-12-1899:
2          NVL(
3          to_date('30-12-1899', 'dd-mm-yyyy') + to_number(<
      TABELLA_DLT>.<CAMPO_DATA>),
4          to_date('#<PROGETTO>.NULL_DATA', 'YYYYMMDDHH24MISS')
5          )
6     -- 2.Trasformazione del campo data come numero di giorni dall'
      origine 30-12-1899 con gestione del valore zero come assenza
      del dato:
7          NVL(
8          CASE
9               WHEN <TABELLA_DLT>.<CAMPO_DATA> = 0 THEN NULL
10              ELSE to_date('30-12-1899', 'dd-mm-yyyy') + to_number(<
      TABELLA_DLT>.<CAMPO_DATA>)
11         END,
12         to_date('#<PROGETTO>.NULL_DATA', 'YYYYMMDDHH24MISS')
13         )
14    -- 3.Trasformazione del campo numerico con separatore decimale
       non standard:
15         NVL(
16         TO_NUMBER(
17              <TABELLA_DLT>.<CAMPO_NUMERICO>,
18              '9999999999D999',
19              'NLS_NUMERIC_CHARACTERS = ''.,'' '
20              ),
21         '#<PROGETTO>.NULL_NUMBER'
22         )
23    -- 4.Trasformazione del campo data come stringa numerico in
      formato \texttt{YYYYMMDDHH24MISS} con gestione del valore zero
      come assenza del dato:
24         NVL(
```

```
25          CASE
26              WHEN <TABELLA_DLT>.<CAMPO_DATA> <> 0
27              THEN TO_DATE(<TABELLA_DLT>.<CAMPO_DATA>, '
    YYYYMMDDHH24MISS')
28              ELSE NULL
29          END,
30          TO_DATE('#<PROGETTO>.NULL_DATA', 'YYYYMMDDHH24MISS')
31          )
```

Listing 4.26: Tipologie di trasformazione applicate ai campi SAM nel *Mapping* OK

La prima tipologia di conversione riguarda i campi data memorizzati come numero intero che rappresenta i giorni trascorsi a partire dal 30 dicembre 1899. In questo caso, la data reale si ottiene dalla somma del valore numerico alla data base e racchiudendo il risultato in una funzione `NVL`, utilizzata per sostituire eventuali valori nulli con il valore default della variabile `NULL_DATE`. La seconda si applica agli stessi campi data in formato numerico, con la differenza che il valore `0` non rappresenta il primo gennaio 1900 bensì l'assenza stessa del dato: una conversione diretta genererebbe una data formalmente valida ma semanticamente errata, per questo motivo viene introdotta un `CASE WHEN` che intercetta questo caso specifico e restituisce un valore `NULL` che viene poi incapsulato in una `NVL` che lo sostituisce con il valore convenzionale. La terza riguarda invece i campi numerici esposti come stringhe con la virgola come separatore decimale, al posto del punto previsto dalla configurazione predefinita di Oracle; questa incompatibilità viene risolta tramite la funzione `TO_NUMBER` che, in combinazione con il parametro `NLS_NUMERIC_CHARACTERS`, consente di indicare esplicitamente il carattere da interpretare come separatore decimale. La quarta, e ultima tipologia, comprende i campi data esposti come stringa numerica nel formato `YYYYMMDDHH24MISS`, nei quali il valore `0` rappresenta nuovamente l'assenza del dato e viene gestito con la medesima logica `CASE WHEN` già adottata per la seconda tipologia.

Completata la configurazione del *Mapping*, nella tab *Physical* viene verificato che l'IKM applicato alla tabella OK sia quello proprietario sviluppato internamente dall'azienda, che estende il comportamento del modulo standard di Oracle aggiungendo logiche di controllo e gestione del flusso specifiche del framework aziendale.

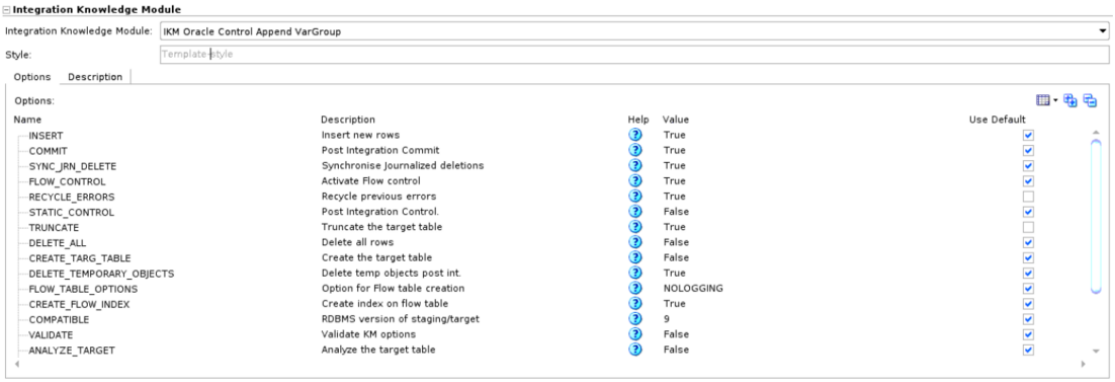

Figura 4.11: Configurazione IKM aziendale

Come si osserva dalla configurazione, il parametro `TRUNCATE` è impostato a `TRUE` e `DELETE_ALL` a `FALSE` garantendo che il contenuto della tabella venga azzerato tramite un'operazione di troncamento prima di ogni inserimento, coerentemente con la natura di tabella di transito della OK, che deve contenere esclusivamente i record del ciclo corrente. L'IKM opera internamente attraverso due tabelle temporanee generate da ODI:

- `I$`: funge da area di lavoro intermedia in cui i record vengono materialmente scritti e trasformati prima di essere spostati nella tabella target;
- `E$`: raccoglie i record che non hanno superato i controlli di integrità o che hanno generato errori durante il processo di integrazione, consentendo di isolare le anomalie senza interrompere il flusso e rendendole disponibili per analisi successive.

Infine viene generato lo scenario del *Mapping* e lo si lancia in ambiente di test, valorizzando `JOBID_L1` con un timestamp nel formato `YYYYMMDDHH24MISS` e `LAST_DATE_READ` con il valore `0` per la prima esecuzione, come avevamo già visto per il lancio degli scenari nel livello L0.

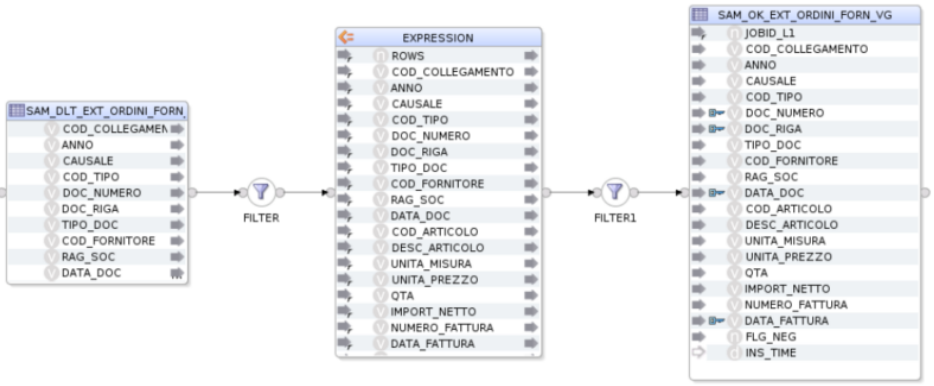

Figura 4.12: *Mapping* OK

4.2.2.2 Mappatura delle tabelle ODS

Il *Mapping* ODS è strutturalmente più lineare rispetto al *Mapping* OK, nell'ODS infatti la tabella OK viene collegata direttamente alla tabella ODS, senza utilizzare filtri o componenti di trasformazione intermedi, poiché la selezione e la qualificazione del dato sono già avvenute nella fase precedente.

Nelle colonne `JOBID_L1_INS` e `JOBID_L1_UPD` viene inserita la variabile di livello `#<PROGETTO>.JOBID_L1` e i campi `INS_TIME` e `UPD_TIME` vengono lasciati vuoti per la stessa ragione illustrata per il *Mapping* OK: entrambi sono definiti con il valore di default `SYSDATE` nella DDL della tabella.

La particolarità del *Mapping* ODS risiede nelle proprietà del target nella tab *Logical*, infatti questa è di tipo *Incremental Update* e come chiave di aggiornamento seleziona il constraint di chiave primaria dichiarato in fase di creazione della tabella. Questa impostazione istruisce ODI a comportarsi in modo differenziato a seconda che il record in ingresso esista già o meno: in *Incremental Update* se la chiave naturale è già presente in ODS, viene eseguito un aggiornamento dei campi non chiave, se invece viene in ingresso arriva un record nuovo, viene eseguito un inserimento.

Data questa proprietà della tabella target, è fondamentale impedire l'*Update* dei campi `JOBID_L1_INS`, `INS_TIME` e da tutti i campi che costituiscono la chiave primaria, facendo in modo che, nel caso di aggiornamento di un record già presente, questi campi non vengano sovrascritti, mantenendo sia la traccia del momento originale di inserimento sia la stabilità della chiave di accesso.

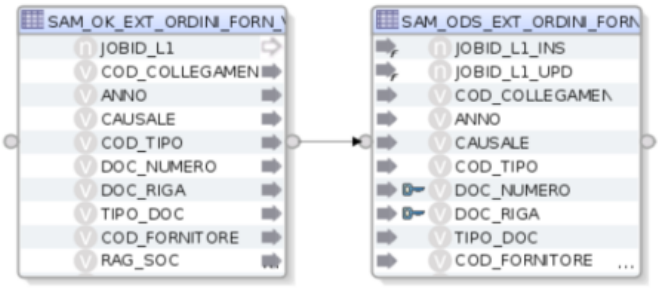

Figura 4.13: *Mapping* ODS

Come verifica finale, nella tab *Physical* si controlla che l'IKM sia `IKM Oracle Merge GLOBAL`, che traduce la logica di *Incremental Update* in un'istruzione SQL `MERGE`, che in un'unica operazione gestisce inserimenti e aggiornamenti.

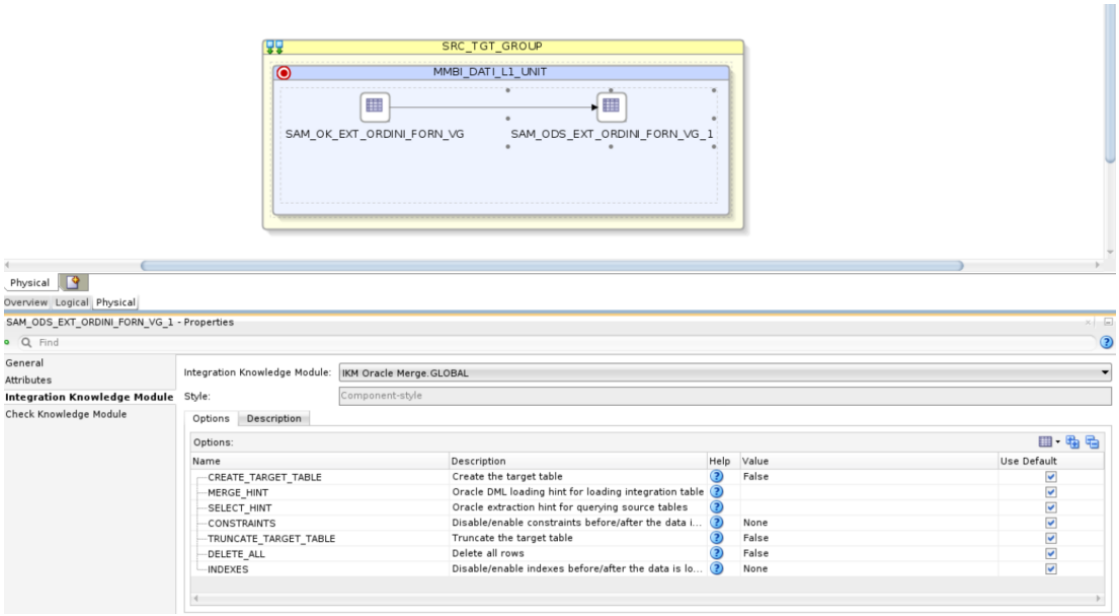

Figura 4.14: Tab *Physical* ODS

4.2.2.3 Mappatura delle tabelle MDM

La costruzione del *Mapping* MDM si distingue dalle fasi precedenti per la sua natura fortemente specifica rispetto alle esigenze di integrazione, infatti non esiste un pattern unico e replicabile su tutte le dimensioni, ma ogni *Mapping* viene

progettato in funzione della struttura delle sorgenti coinvolte, della loro omogeneità e del fatto che una dimensione provenga da una sola sorgente o dalla riconciliazione di tabelle provenienti da sorgenti diverse. Nello svolgimento di questo progetto si riscontrano tre fattispecie, con complessità crescente.

Il caso più semplice è quello delle dimensioni alimentate da un'unica sorgente, prendiamo come esempio la dimensione `BUSINESSUNITS`: la tabella ODS sorgente viene collegata a un filtro con condizione `<TABELLA_ODS>.JOBID_L1_UPD > #<PROGETTO>.LAST_DATE_READ`, che seleziona esclusivamente i record aggiornati dall'ultima elaborazione, successivamente il filtro viene connesso a un componente *Expression* in cui vengono rimossi tutti i campi tecnici ad eccezione del `FLG_NEG`, e viene aggiunto il campo `FLG_SRC`, valorizzato tramite la funzione ODI `MDM_FLG_SRC`.

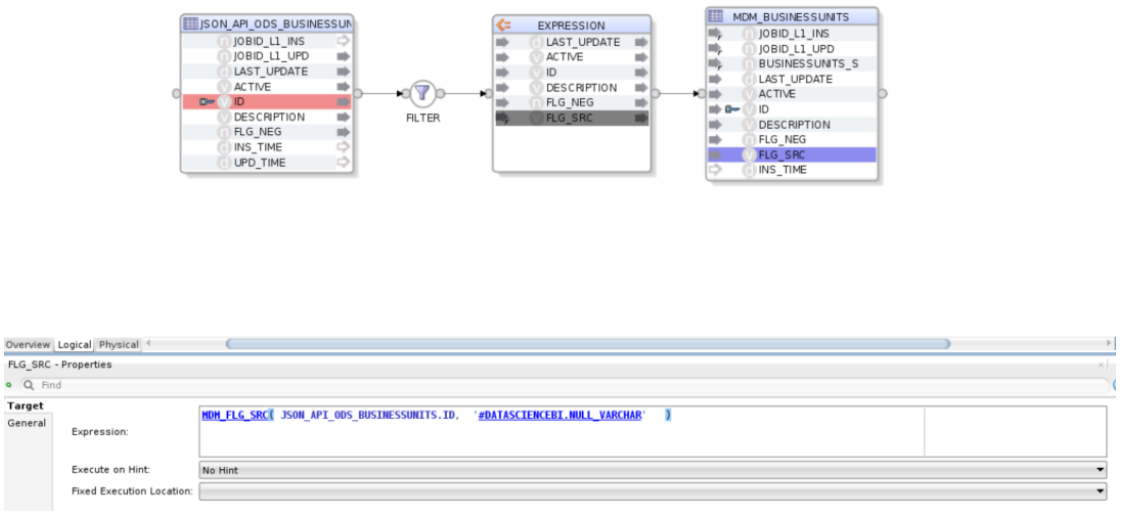

Figura 4.15: *Mapping* `MDM_BUSINESSUNITS` con dettaglio sulla funzione `MDM_FLG_SRC`

Il secondo parametro rappresenta il valore convenzionale della sorgente assente: poiché la chiave è tipicamente di tipo `VARCHAR`, si utilizza `NULL_VARCHAR`; nel caso in cui fosse numerica sarebbe necessario convertirla tramite `TO_CHAR` per garantire l'omogeneità dei tipi in ingresso alla funzione. L'*Expression* viene poi connessa alla tabella MDM target, in cui `JOBID_L1_INS` e `JOBID_L1_UPD` vengono valorizzati con la variabile `#<PROGETTO>.JOBID_L1`, la chiave surrogata viene valorizzata richiamando la *Sequence* definita in fase di preparazione, mentre `INS_TIME` e `UPD_TIME` rimangono non valorizzati, affidandosi al valore di default definito al momento di creazione della tabella sul database.

Il secondo caso è quello delle dimensioni che richiedono la riconciliazione di due sorgenti con strutture omogenee, prendiamo come esempio la dimensione

`CUSTOMERS`, dove la tabella `JSON_API_ODS_CUSTOMERS` proveniente da Webgate e la tabella `ODS_DIM_CLIENTE` proveniente da SAM, descrivono la stessa entità anagrafica: le due tabelle vengono messe in join tramite le rispettive chiavi naturali, utilizzando una *left join* che dà priorità alla sorgente Webgate, coerentemente con quanto stabilito nella fase di progettazione delle strutture fisiche, il join viene poi collegato a un filtro la cui condizione, in presenza di più sorgenti, non può più essere una semplice verifica sul singolo `JOBID_L1_UPD`, ma deve tenere conto dell'aggiornamento più recente tra tutte le tabelle coinvolte, tramite la funzione GREATEST, questa condizione garantisce che il record venga processato non appena almeno una delle due sorgenti abbia subito una variazione, indipendentemente da quale delle due sia stata aggiornata per ultima. A valle del filtro si trova l'*Expression* con la medesima struttura già descritta, dove la funzione `MDM_FLG_SRC` riceve in questo caso le chiavi di entrambe le tabelle ODS come parametri, permettendo di determinare da quale combinazione di sorgenti provenga l'informazione consolidata. La tabella MDM target viene infine configurata con le stesse modalità illustrate per il caso precedente, valorizzando `JOBID_L1_INS`, `JOBID_L1_UPD` e la chiave surrogata tramite la rispettiva *Sequence*.

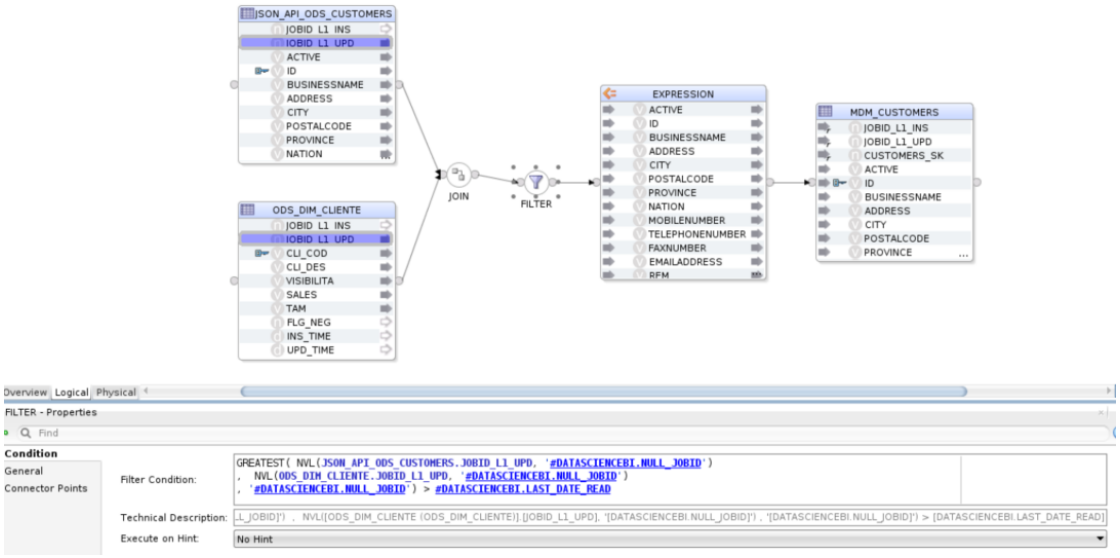

Figura 4.16: *Mapping* `MDM_CUSTOMERS` con dettaglio sulla condizione di filtro

Il terzo caso, il più complesso, è quello di dimensioni in cui la corrispondenza tra sorgenti non è uno a uno ma uno a molti, come nel caso di `WORKORDERS`, dove la dimensione Webgate `JSON_API_ODS_WORKORDERS` rappresenta indistintamente sia commesse che sottocommesse, mentre la sorgente SAM le espone su due tabelle separate, `ODS_DIM_COMMESSE` e `ODS_DIM_SOTTOCOMMESSE`. La costruzione del

Mapping richiede in questo caso due join successivi: il primo è una *full outer join* tra `ODS_DIM_COMMESSE` e `ODS_DIM_SOTTOCOMMESSE` sulla chiave comune `COM_ID`, a valle della quale viene inserito un componente *Expression* in cui ai campi presenti in entrambe le tabelle viene applicata una funzione `NVL` per garantire che l'informazione venga recuperata da qualunque delle due tabelle la esponga, il secondo join mette in relazione il risultato di questa prima elaborazione con la tabella Webgate, ma con una complicazione aggiuntiva, poiché i due sistemi codificano gli identificativi di commessa e sottocommessa in formati numerici incompatibili tra loro. Webgate utilizza un formato compatto come `YYNNNNNNN` per le commesse e `YYNNNNNNNA` per le sottocommesse, dove le prime due cifre rappresentano le ultime due cifre dell'anno, le successive cifre il numero di commessa con *zero-padding*, e la lettera finale identifica la sottocommessa in ordine alfabetico a partire da A, mentre SAM utilizza invece un formato esteso come `YYYYNNNNNOO` per le commesse e `YYYYNNNNNO1` per le sottocommesse, con l'anno esteso a quattro cifre, il numero commessa e un suffisso numerico a due cifre che indica l'ordine della sottocommessa a partire da `01`. poiché la priorità è data alla sorgente Webgate, viene trasformato il formato SAM per renderlo compatibile con quello Webgate.

```
1     JSON_API_ODS_WORKORDERS.ID = SUBSTR(NVL(ODS_DIM_SOTTOCOMMESSE.
    SOTTOC_ID, ODS_DIM_COMMESSE.COM_ID), 3, 2)
2     || LPAD(SUBSTR(NVL(ODS_DIM_SOTTOCOMMESSE.SOTTOC_ID,
    ODS_DIM_COMMESSE.COM_ID), 5, 4), 6, '0')
3     || CASE
4         WHEN SUBSTR(NVL(ODS_DIM_SOTTOCOMMESSE.SOTTOC_ID,
    ODS_DIM_COMMESSE.COM_ID), -2) = '00' THEN ''
5         ELSE CHR(64 + TO_NUMBER(SUBSTR(NVL(ODS_DIM_SOTTOCOMMESSE.
    SOTTOC_ID, ODS_DIM_COMMESSE.COM_ID), -2)))
6     END
```

Listing 4.27: Espressione di join per l'allineamento dei formati degli identificativi tra Webgate e SAM nel *Mapping* `MDM_WORKORDERS`

La logica applicata prevede l'estrazione dell'anno in formato breve, il numero di commessa con *zero-padding* a sei cifre e converte il suffisso numerico SAM nella lettera alfabetica corrispondente tramite la funzione `CHR`, trattando il valore `00` come assenza di sottocommessa. Il filtro successivo estende la condizione `GREATEST` a tutte e tre le sorgenti coinvolte nel processo, e l'*Expression* finale valorizza `FLG_SRC` con la funzione `MDM_FLG_SRC` passando come parametri la chiave Webgate e quella SAM ricostruita nell'*Expression* intermedia. Infine, nelle stesse modalità dei casi descritti precedentemente, viene configurata la tabella MDM target.

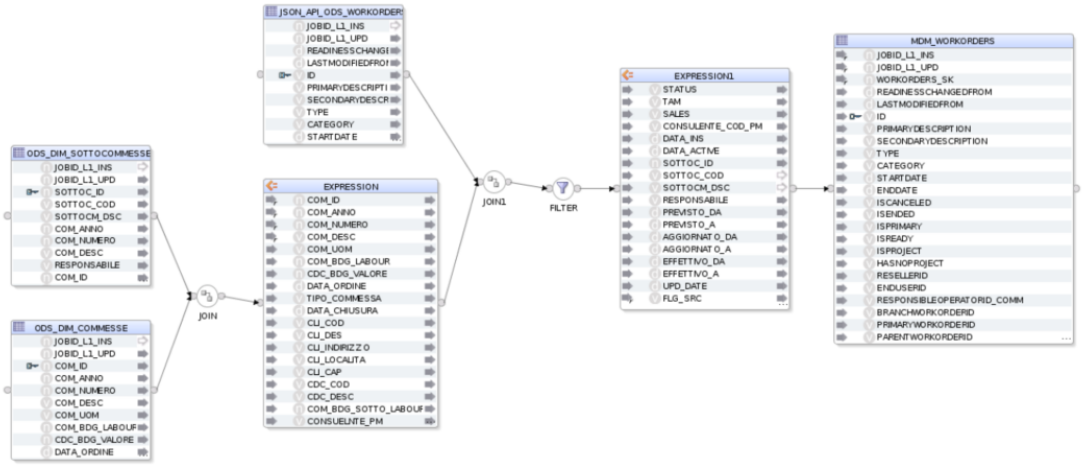

Figura 4.17: *Mapping* `MDM_WORKORDERS`

Ad accomunare tutte le tabelle è la configurazione della tabella target nella tab *Logical*, in cui viene impostato come tipo di integrazione *Incremental Update*, selezionando come *Update Key* la chiave primaria della tabella MDM, che per i principi di priorità è la chiave naturale della tabella ODS proveniente dalla sorgente Webgate. Inoltre vengono rimossi dallo status di *Update* i campi in inserimento, ovvero `JOBID_L1_INS` e `INS_TIME`, le chiavi naturali e dalla chiave surrogata. Questa ultima è di fondamentale importanza in quanto una volta assegnata ad un record, la chiave surrogata non deve essere mai aggiornata nei cicli successivi, anche se il record subisce modifiche nei suoi attributi informativi, questo perché ai livelli superiori del framework si smette di far riferimento alle chiavi primarie ma si richiama l'entità anagrafica tramite la sua chiave surrogata. Se questa venisse rigenerata in occasione di un aggiornamento, il record assumerebbe un nuovo identificativo e tutti i collegamenti esistenti con i livelli superiori perderebbero la propria referenza, rendendo di fatto invisibile la connessione tra i fatti storici e l'entità aggiornata.

Infine, come facevamo già per i *Mapping* ODS, si verifica che nella tab *Physical* l'IKM sia configurato in modalità *Merge*, si genera quindi lo scenario e lo si lancia in ambiente di test con `JOBID_L1` nel formato `YYYYMMDDHH24MISS` e `LAST_DATE_READ` impostato a `0`.

4.2.2.4 Mappatura delle tabelle OUT

La costruzione del *Mapping* OUT comincia dall'inserire all'interno del canvas la tabella MDM dell'entità di riferimento e, prima di costruire qualsiasi flusso, è necessario analizzarne la struttura per identificare quanti e quali identificativi di entità

esterne siano presenti al suo interno. Ogni identificativo esterno corrisponderà ad un componente *Join* nel *Mapping*, quindi il numero di join da inserire è determinato direttamente dal numero di entità esterne referenziate. Tutti i join vengono poi collegati in cascata mantenendo come tabella di guida la MDM dell'entità di riferimento in modo da preservarne la priorità lungo tutto il flusso.

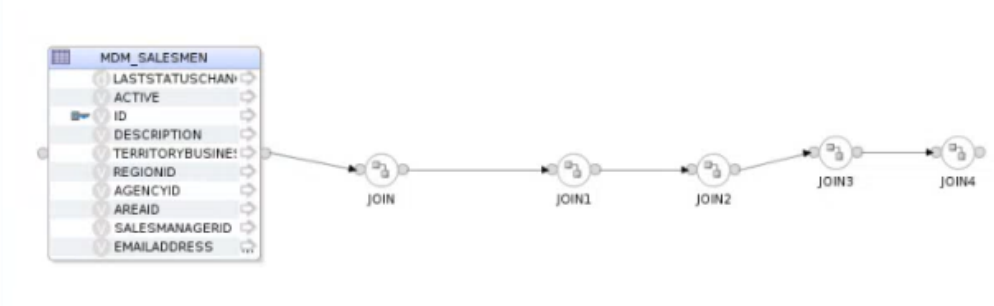

Figura 4.18: Catena di componenti *Join* collegata alla MDM principale

Per ciascun join viene portata nel canvas la tabella MDM dell'entità esterna corrispondente, e la condizione di join viene definita mettendo in relazione l'identificativo dell'entità esterna presente nella MDM di riferimento con l'identificativo corrispondente nella MDM esterna. A valle di tutti i join viene inserito un filtro con una condizione `GREATEST` che aggrega i `JOBID_L1_UPD` di tutte le tabelle MDM partecipanti al *Mapping*.

```
1     GREATEST(
2          NVL(<TABELLA_MDM_RIFERIMENTO>.JOBID_L1_UPD, '#<PROGETTO>.
      NULL_JOBID'),
3          NVL(<TABELLA_MDM_ENTITÀ_ESTERNA_1>.JOBID_L1_UPD, '#<
      PROGETTO>.NULL_JOBID'),
4          ...,
5          NVL(<TABELLA_MDM_ENTITÀ_ESTERNA_N>.JOBID_L1_UPD, '#<
      PROGETTO>.NULL_JOBID')
6     ) > #<PROGETTO>.LAST_DATE_READ
```

Listing 4.28: Condiione `GREATEST` nel *Mapping* OUT

Questa condizione è necessaria perché in presenza di più tabelle sorgente un aggiornamento rilevante per l'entità di riferimento potrebbe provenire da una qualsiasi delle tabelle coinvolte nel *Mapping*, non necessariamente dalla MDM principale. Senza il `GREATEST`, il *Mapping* processerebbe soltanto le variazioni registrate nella tabella di riferimento, perdendo le modifiche introdotte nalle entità esterne che contribuiscono alla composizione del record, con esso invece il record viene sempre aggiornato non appena almeno una delle tabelle partecipanti abbia subito una variazione dall'ultima elaborazione. L'`NVL` invece, serve a gestire i casi

in cui il valore sia `NULL` poiché questo renderebbe il confronto con `LAST_DATE_READ` impossibile, di fatto escludendo il record dal flusso; sostituendo il valore nullo con il valore convenzionale `NULL_JOBID`, che per costruzione è una data molto remota nel passato sicuramente sempre inferiore a qualsiasi valore attribuito alla variabile `LAST_DATE_READ`, si garantisce che la funzione `GREATEST` produca sempre un risultato confrontabile e che la presenza di valori nulli non alteri il comportamento del filtro.

Il flusso così filtrato viene poi connesso alla tabella OUT target e, successivamente, vengono valorizzati i campi: `JOBID_L1` con la variabile `#<PROGETTO>.JOBID_L1`, la chiave primaria della OUT con la SK della tabella MDM principale, i campi espansi con i valori presi dalle rispettive tabelle MDM e infine il campo `INS_TIME` viene lasciato vuoto in quanto valorizzato di default a `SYSDATE` al momento della creazione della tabella.

Successivamente si verifica, all'interno della tab *Physical*, che l'IKM selezionato sia `IKM Oracle Insert GLOBAL` e che il parametro `TRUNCATE_TARGET_TABLE` sia impostato a `TRUE`, questo perché, a differenza delle ODS e delle MDM, che lavorano in modalità *merge*, preservando la storia del record, la OUT viene completamente ricreata ad ogni esecuzione.

Infine, a conclusione dell'implementazione di OUT, si genera lo scenario e lo si lancia in ambiente di test valorizzando, come negli altri casi, `JOBID_L1` nel formato `YYYYMMDDHH24MISS` e `LAST_DATE_READ` con il valore `0`.

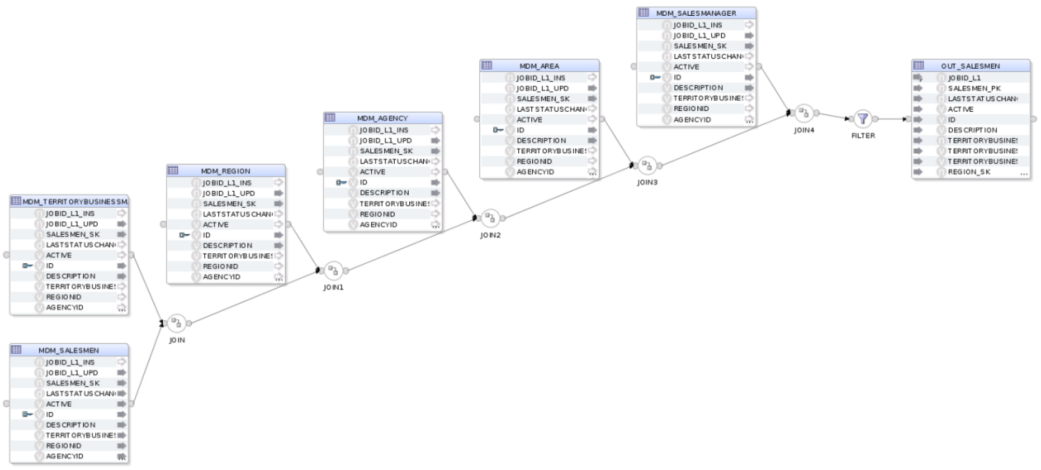

Figura 4.19: *Mapping* OUT

4.2.2.5 Orchestrazione delle mappature per il livello L1

Dopo aver costruito e verificato tutti i *Mapping* OK, ODS, MDM e OUT, si è proceduto alla loro integrazione all'interno delle strutture di orchestrazione, seguendo una logica gerarchica analoga a quella adottata per il livello L0. La differenza principale rispetto al livello precedente risiede nell'assenza del livello CDC, infatti, non essendo previsto alcun calcolo del delta in L1, gli scenari dei *Mapping* vengono direttamente richiamati all'interno dei *Load Plan*.

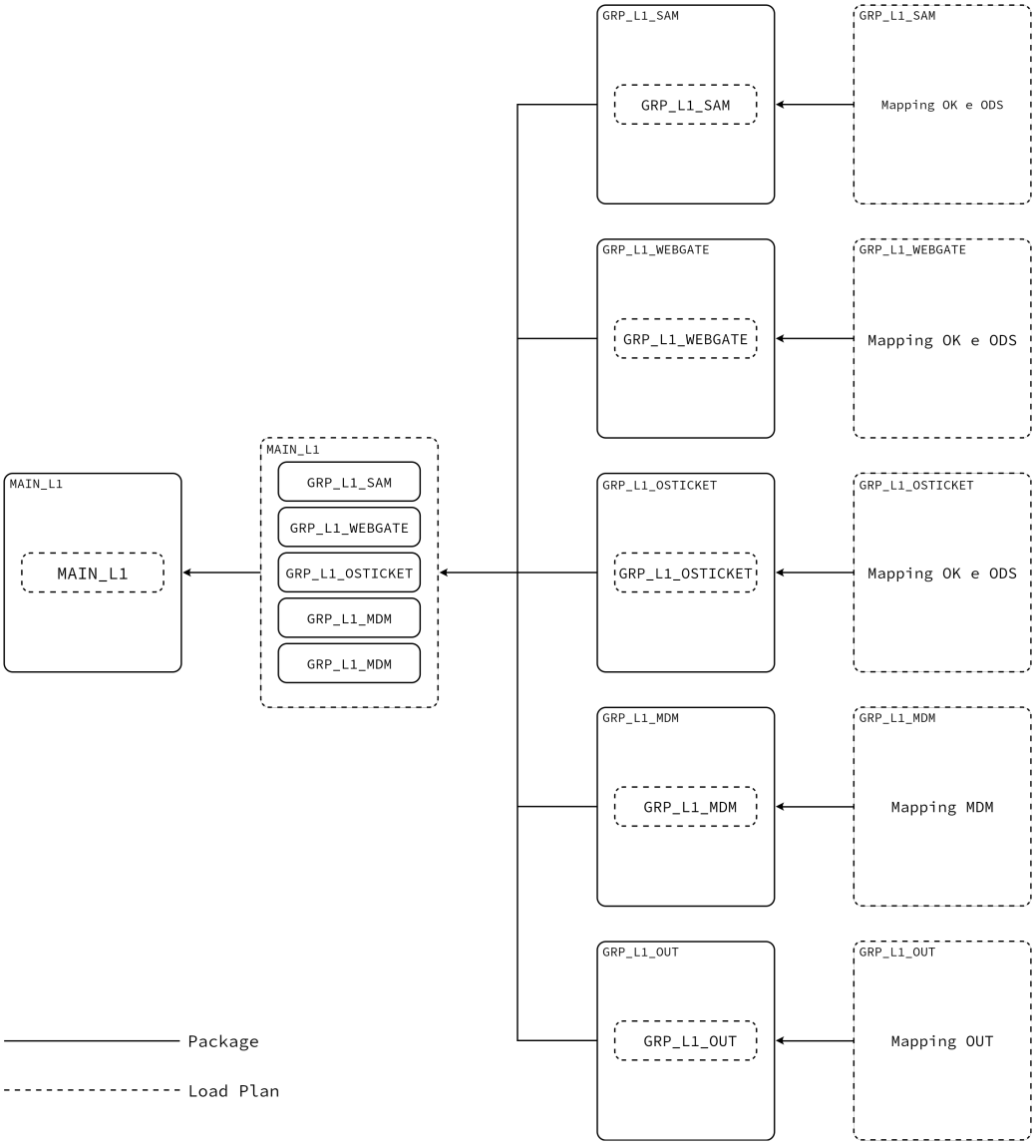

Figura 4.20: Diagramma gerarchico dell'orchestrazione L1: *Package* (linee continue) e *Load Plan* (linee tratteggiate)

Il primo livello della gerarchia è costituito dai *Package* GRP, uno per ciascuna sorgente e per ciascun dominio logico. Strutturalmente, questi *Package* sono identici a quelli già visti in L0: si apre con la dichiarazione delle variabili `IDENTITY`, di tipo *Declare Variable* e `GRP_NAME`,configurata come *Set Variable* valorizzata con l'identificativo del gruppo `L1_<SORGENTE>`, a cui seguono `NUM_LEVEL` e `JOBID`, entrambe con tipo *Declare Variable*, quindi viene eseguita la procedura `START_PROCESS`, seguita dal *Load Plan* intermedio di competenza e infine la procedura `END_PROCESS`, con la medesima gestione degli errori tramite transizione *ko* verso la procedura `END_PROCESS_ERROR`. Per i *Mapping* OK e ODS esistono tre *Package* di gruppo distinti per sorgente `GRP_L1_<SORGENTE>`ciascuno dei quali richiama il rispettivo *Load Plan* intermedio: `L1_ANAGRAFICHE_<SORGENTE>` per le dimensioni, e le corrispondenti versioni `L1_FATTI_<SORGENTE>` per le tabelle dei fatti. Per i *Mapping* MDM il *Package* di riferimento è `GRP_L1_MDM`, che richiama il *Load Plan* `L1_MDM` contenente in serie tutti i *Mapping* di riconciliazione anagrafica; analogamente per i *Mapping* OUT il *Package* `GRP_L1_OUT` richiama il *Load Plan* `L1_OUT`. Come già precisato precedentemente, il blocco MDM è previsto esclusivamente per le anagrafiche in quanto la riconciliazione anagrafica non ha applicazione sulle tabelle dei fatti.

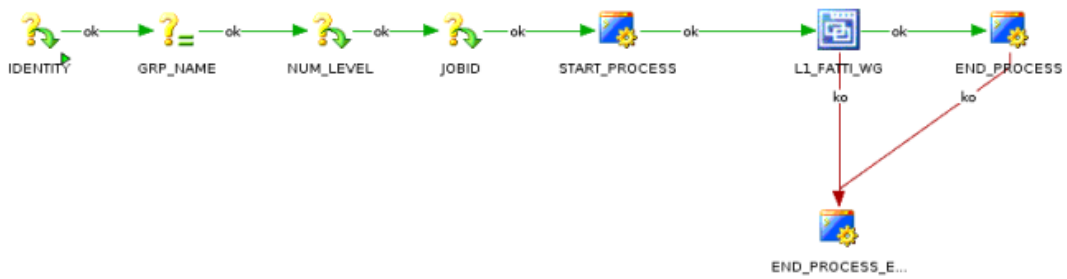

Figura 4.21: Sequenza di esecuzione del *Package* GRP per le tabelle dei fatti in L1

Completati i *Package* GRP, si procede alla costruzione dei *Load Plan* del livello superiore, quali `L1_ANAGRAFICHE` e `L1_FATTI`. Questi *Load Plan* aggregano in serie i *Package* `GRP_L1_<SORGENTE>` assieme ai blocchi `GRP_L1_MDM` (solo per il *Load Plan* `L1_ANAGRAFICHE`) e `GRP_L1_OUT`.

Al vertice della gerarchia si colloca il *Package* `MAIN_L1`, che funge da punto di ingresso dell'intera orchestrazione del livello. La sua struttura è identica a quella già esplorata del `MAIN_L0`: la variabile `IDENTITY` assume sempre il valore `ANAGRAFICHE` o `FATTI` a seconda del dominio logico in cui ci troviamo, `GRP_NAME` assume ancora un valore composto dal dominio logico, concatenato al livello del framework, quindi `ANAGRAFICHE_L1` o `FATTI_L1`, mentre, `NUM_LEVEL` viene impostato ad `1`. La

variabile `JOBID` viene dichiarata senza valorizzazione esplicita ed impostata con tipo *Declare Variable*, poiché il suo valore sarà definito dalla procedura `START_PROCESS` che segue immediatamente. Il flusso si chiude, dopo l'esecuzione del *Load Plan* `L1_<DOMINIO_LOGICO>`, con la procedura `END_PROCESS`, con la medesima gestione degli errori già descritta per il *Package* GRP, attraverso la transizione *ko* verso `END_PROCESS_ERROR`.

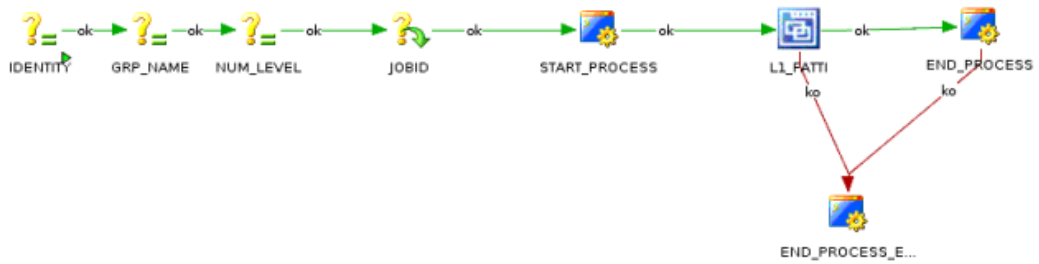

Figura 4.22: Sequenza di esecuzione del *Package* MAIN per le tabelle dei fatti in L1

4.3 Livello L2 - Publication Area

4.3.1 Fase di preparazione all'implementazione in ODI

Il livello L2 rappresenta il punto terminale della pipeline di integrazione descritta dal framework aziendale, e costituisce la superficie esposta agli strumenti di analisi e reportistica aziendale. In questo livello le informazioni vengono modellate in funzione delle esigenze informative e delle caratteristiche della piattaforma analitica che verrà utilizzata, adottando nel nostro caso uno *Star schema* che favorisce la migrazione da Oracle BI, che predilige un modello normalizzato di tipo *Snowflake schema*, a Power BI, che riesce a lavorare al massimo delle proprie capacità partendo da un modello denormalizzato con dimensioni direttamente collegabili alle tabelle dei fatti, schermando gli utenti finali dalla complessità dei processi di integrazione a monte. A differenza del Livello L0, articolato nelle tabelle STG, DLT e DLT_HIS, e di L1, che elabora i dati attraverso le tabelle OK, ODS, MDM e OUT, il livello L2 è composto da un'unica tipologia di tabella, la PUB, e non è stato necessario introdurre nuove variabili di orchestrazione, *Procedure*, funzioni o sequenze, in quanto tutte le componenti di controllo e tracciamento definite nei livelli precedenti risultano sufficienti anche per questo livello. La fase preparatoria si è pertanto focalizzata esclusivamente sulla definizione e sulla creazione delle tabelle PUB all'interno del database.

Le tabelle PUB ereditano dalla corrispondente tabella OUT del livello L1 l'intero contenuto informativo, inclusa l'espansione dei campi delle entità esterne prodotta dalla catena di join costruita nel *Mapping* precedente. La chiave primaria della PUB coincide con quella della tabella OUT, corrispondente a sua volta alla chiave surrogata assegnata in fase di MDM, così da permettere allo strumento di visualizzazione l'interrogazione delle tabelle direttamente tramite SK che, essendo un dato di tipo numerico, rende le tabelle PUB più stabili e gli eventuali join più efficienti. Il vincolo di chiave primaria viene dichiarato esplicitamente al momento della definizione sul database della tabella secondo la convenzione `PK_D_<ENTITÀ>` per le dimensioni e `PK_F_<TABELLA>` per le tabelle dei fatti, coerentemente con le rispettive convenzioni di nomenclatura `D_<ENTITÀ>` e `F_<TABELLA>`.

Rispetto alla struttura della tabella OUT, le PUB introducono alcune differenze significative, tra cui la sostituzione del campo `JOBID_L1` con la coppia `JOBID_L2_INS` e `JOBID_L2_UPD`, che replicano la stessa logica adattata nelle ODS del livello L1 per cui il primo traccia il ciclo di caricamento in cui il record è stato inserito per la prima volta nella tabella pubblicata, il secondo viene aggiornato ad ogni ciclo successivo in cui il record subisce una modifica. Analogamente ritorna il campo `UPD_TIME`, affiancato ad `INS_TIME`, valorizzato di default con `SYSDATE`, che consente di datare l'ultimo aggiornamento effettuato sul record. Infine i record `FLG_NEG` e `FLG_SRC`, introdotti rispettivamente nei livelli L0 ed L1, vengono mantenuti invariati, preservando lungo tutto il flusso le informazioni sul tipo di variazione e sulla provenienza del dato.

```
1     CREATE TABLE "L2"."D_SALESMEN"
2        (  "JOBID_L2_INS" NUMBER NOT NULL ENABLE,
3       "JOBID_L2_UPD" NUMBER NOT NULL ENABLE,
4       "SALESMEN_PK" NUMBER NOT NULL ENABLE,
5       "LASTSTATUSCHANGE" DATE,
6       "ACTIVE" VARCHAR2(400 CHAR),
7       "ID" VARCHAR2(400 CHAR),
8       "DESCRIPTION" VARCHAR2(400 CHAR),
9       "TERRITORYBUSINESSMANAGER_SK" NUMBER,
10      "TERRITORYBUSINESSMANAGER_ID" VARCHAR2(400 CHAR),
11      "TERRITORYBUSINESSMANAGER_DESCRIPTION" VARCHAR2(400 CHAR),
12      "REGION_SK" NUMBER,
13      "REGION_ID" VARCHAR2(400 CHAR),
14      "REGION_DESCRIPTION" VARCHAR2(400 CHAR),
15      "AGENCY_SK" NUMBER,
16      "AGENCY_ID" VARCHAR2(400 CHAR),
17      "AGENCY_DESCRIPTION" VARCHAR2(400 CHAR),
18      "AREAMANAGER_SK" NUMBER,
19      "AREAMANAGER_ID" VARCHAR2(400 CHAR),
20      "AREAMANAGER_DESCRIPTION" VARCHAR2(400 CHAR),
21      "SALESMANAGER_SK" NUMBER,
22      "SALESMANAGER_ID" VARCHAR2(400 CHAR),
```

```
23        "SALESMANAGER_DESCRIPTION" VARCHAR2(400 CHAR),
24        "EMAILADDRESS" VARCHAR2(400 CHAR),
25        "FLG_NEG" NUMBER NOT NULL ENABLE,
26        "FLG_SRC" VARCHAR2(400 CHAR),
27        "INS_TIME" DATE DEFAULT SYSDATE,
28        "UPD_TIME" DATE DEFAULT SYSDATE,
29         CONSTRAINT "PK_D_SALESMEN" PRIMARY KEY ("SALESMEN_PK")
30       USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     STATISTICS
31       STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     2147483645
32       PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
33       BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     DEFAULT)
34       TABLESPACE "TBS_L2"  ENABLE
35        ) SEGMENT CREATION IMMEDIATE
36       PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
37      NOCOMPRESS LOGGING
38       STORAGE(INITIAL 81920 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     2147483645
39       PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
40       BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     DEFAULT)
41       TABLESPACE "TBS_L2" ;
```

Listing 4.29: Tabella PUB esempio

4.3.2 Implementazione in ODI

4.3.2.1 Mappatura delle tabelle PUB

La costruzione del *Mapping* per l'integrazione nel flusso delle tabelle PUB si limita a selezionare l'immagine più aggiornata di ogni entità presente nella OUT e a pubblicarla nella tabella di destinazione.

Il piano di lavoro si apre con la tabella OUT come unica sorgente, collegata ad un filtro che seleziona esclusivamente i record generati da esecuzioni successive all'ultima già elaborata in questo livello attraverso la condizione `<TABELLA_OUT>.JOBID_L1 > #<PROGETTO>.LAST_DATE_READ`. Il filtro viene poi connesso ad un componente *Expression* in cui non viene mappata automaticamente tutta la sorgente, ma viene aggiunto manualmente un unico attributo `ROWS`, valorizzato tramite una funzione `ROW_NUMBER()` che numera i record all'interno di ogni partizione definita dalla chiave primaria della OUT, dal più recente, ordinando i record per `JOBID_L1` e `INS_TIME` entrambi decrescenti.

```
1     ROW_NUMBER() OVER (
2         PARTITION BY <TABELLA_OUT>.<PK>
```

```
3           ORDER BY
4               <TABELLA_OUT>.JOBID_L1 DESC,
5               <TABELLA_OUT>.INS_TIME DESC
6       )
```

Listing 4.30: Espressione `ROW_NUMBER()` nel *Mapping* PUB

L'*Expression* viene poi connessa ad un secondo filtro con condizione `ROWS = 1`, che mantiene esclusivamente il record di testa per ogni chiave, eliminando i record precedenti della stessa entità. Infine questo filtro viene collegato alla tabella PUB di destinazione.

Nella tabella target, le colonne `JOBID_L2_INS` e `JOBID_L2_UPD` vengono valorizzate con la variabile `#<PROGETTO>.JOBID_L2`, mentre rimangono vuote le espressioni dei campi `INS_TIME` e `UPD_TIME` in modo da essere affidati alle valorizzazioni di default a `SYSDATE`. Successivamente viene impedito che i campi in inserimento, vale a dire `JOBID_L2_INS` e `INS_TIME`, insieme alla chiave primaria, non vengano sovrascritti in caso di aggiornamento di un record già presente, preservando la traccia del momento originale di inserimento e la stabilità della chiave.

Nella tab *Logical* la tabella target viene configurata con tipo di integrazione *Incremental Update* e chiave di aggiornamento la PK della tabella PUB, di conseguenza si va a fare un controllo nella tab *Physical* per confermare che l'IKM sia `IKM Oracle Merge GLOBAL`, che traduce la logica di *Incremental Update* in un'istruzione SQL `MERGE`.

Come passaggio finale, si genera quindi lo scenario e lo si lancia in ambiente di test valorizzando `JOBID_L2` nel formato `YYYYMMDDHH24MISS` e `LAST_DATE_READ` con il valore `0`.

Figura 4.23: *Mapping* PUB

4.3.2.2 Orchestrazione delle mappature per il livello L2

In seguito alla costruzione e alla verifica dei *Mapping* di PUB, avviene la progettazione delle strutture di orchestrazione; a questo livello la logica gerarchica che

caratterizza le strutture di coordinamento dei processi è la medesima di quella adottata per il livello L1.

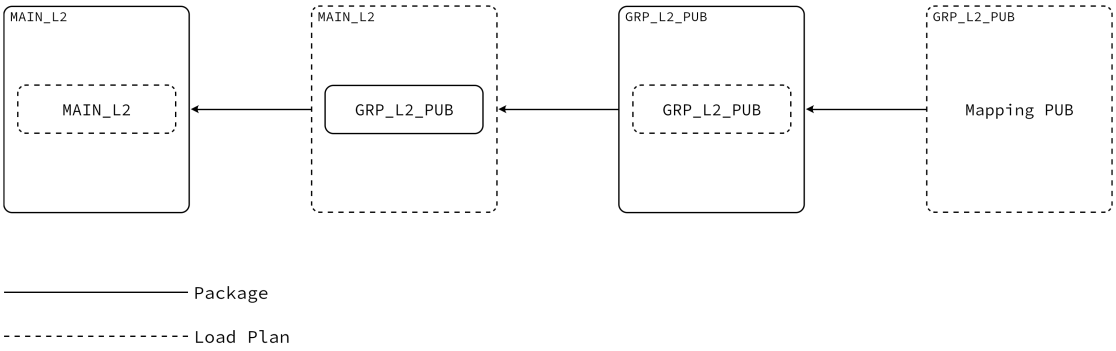

Figura 4.24: Diagramma gerarchico dell'orchestrazione L2: *Package* (linee continue) e *Load Plan* (linee tratteggiate)

Alla base della gerarchia si collocano i *Package* GRP, uno per ciascun dominio logico, la cui struttura riprende fedelmente l'impostazione già adottata nei livelli precedenti: il flusso si apre con la dichiarazione delle variabili `IDENTITY`, di tipo *Declare Variable*, e `GRP_NAME`, configurata come *Set Variable* e valorizzata con l'identificativo del gruppo `L2_PUB`, seguite da `NUM_LEVEL` e `JOBID`, entrambe di tipo *Declare Variable*; di seguito viene poi eseguita la procedura `START_PROCESS`, seguita dal *Load Plan* intermedio di competenza e, in chiusura, dalla procedura `END_PROCESS`, con la gestione degli errori già incontrata per i *Package* GRP degli altri livelli, attraverso la transizione *ko* verso la procedura `END_PROCESS_ERROR`. A differenza del livello L1, in cui i *Package* GRP che trattavano le OK e le ODS, erano distinti per ciascuna sorgente, in L2 il dato ha già attraversato le fasi di riconciliazione anagrafica in MDM e di arricchimento in OUT, risultando completamente unificato: i *Package* GRP in L2 sono quindi definiti unicamente per dominio logico, `GRP_L2_ANAGRAFICHE` e `GRP_L2_FATTI`, ciascuno dei due richiama il rispettivo *Load Plan* intermedio `L2_ANAGRAFICHE_PUB` e `L2_FATTI_PUB`.

Figura 4.25: Sequenza di esecuzione del *Package* GRP per le tabelle anagrafiche in L2

Una volta completata la definizione dei *Package* GRP, si procede alla costruzione dei *Load Plan* del livello superiore `L2_ANAGRAFICHE` e `L2_FATTI`, i quali aggregano in sequenza i rispettivi *Package*.

La gerarchia di orchestrazione si conclude con il *Package* `MAIN_L2`, punto di ingresso dell'intera orchestrazione del livello, la cui struttura ricalca quella già descritta per `MAIN_L0` e `MAIN_L1`, in cui `IDENTITY` viene valorizzata in base al dominio logico in `ANAGRAFICHE` o `FATTI`, `GRP_NAME` rispettivamente in `ANAGRAFICHE_L2` e `FATTI_L2`, e `NUM_LEVEL` viene impostato a `2` coerentemente con il livello del framework in cui ci troviamo. La variabile `JOBID` viene poi dichiarata senza una valorizzazione esplicita essendo il valore assegnato al passo successivo dalla procedura `START_PROCESS`. Il flusso si conclude con l'esecuzione del *Load Plan* `L2_<DOMINIO_LOGICO>` seguito dalla procedura `END_PROCESS`, mantenendo la stessa gestione degli errori dei livelli precedenti con la transizione *ko* in direzione della procedura `END_PROCESS_ERROR`.

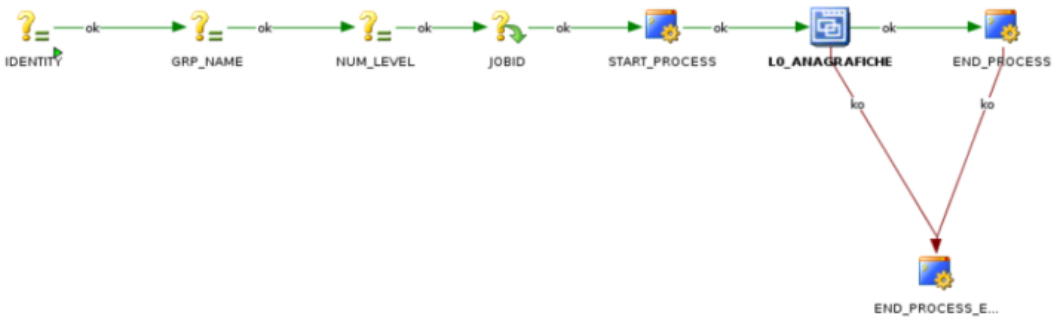

Figura 4.26: Sequenza di esecuzione del *Package* MAIN per le tabelle anagrafiche in L2

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Risultati ottenuti e conclusioni progettuali

Il progetto di reingegnerizzazione descritto nel corso di questa tesi ha prodotto, seguendo il framework sviluppato dall'azienda ospitante, un flusso di integrazione dati standardizzato, modulare e governabile, che supera in modo strutturale le criticità emerse nell'analisi dell'architettura AS-IS. L'intervento si è reso necessario in un contesto di crescente complessità dei flussi informativi aziendali, in cui l'introduzione di nuove sorgenti dati e l'adozione di strumenti analitici più evoluti avevano evidenziato i limiti di un'architettura non più adeguata alle esigenze di manutenibilità e coerenza nel lungo periodo.

Tra i contributi del livello L0, la gestione dei delta ricopre un ruolo centrale. I *Mapping* DLT, infatti, basandosi sulla differenza insiemistica OGGI_IERI e IERI_OGGI, hanno sostituito le strategie di *full reload* dell'architettura precedente con un meccanismo di CDC, capace di isolare le variazioni tra cicli successivi, riducendo significativamente l'onerosità del processo.

Il livello L1 è concettualmente il livello più denso del framework: l'introduzione della fase OK in questo livello ha reso esplicito un punto di controllo nel flusso di integrazione, in cui vengono applicati controlli di integrità referenziale, gestione strutturata dei valori nulli tramite valori convenzionali letti a tempo di esecuzione dalla tabella `METADATA_MANAGER`, conversioni di tipo specifiche per le criticità della sorgente SAM e processi di deduplicazione fondati su chiavi naturali identificate analiticamente. Tali controlli, precedentemente assenti o affidati a logiche implicite all'interno di procedure PL/SQL, sono ora codificati all'interno di *Mapping* ODI versionabili, rendendo il comportamento verificabile e riproducibile, e di conseguenza affidabile. È a valle di questa catena di controlli che il dato viene promosso alla ODS, dove assume il ruolo di riferimento stabile per la riconciliazione

anagrafica, operazione che avviene in MDM. Queste hanno fatto fronte ad una criticità introdotta dalla nuova struttura stessa: l'integrazione della sorgente Webgate, assente nell'architettura precedente, ha portato nel sistema entità aziendali già rappresentate in SAM, rendendo necessario un meccanismo esplicito di allineamento. La generazione di chiavi surrogate stabili tramite sequence Oracle e le regole di priorità esplicite tra le due sorgenti hanno prodotto una rappresentazione anagrafica unificata su cui possono fondarsi le successive analisi di business.

Sul piano della tracciabilità, le strutture nello schema `ADMIN`, in particolare `FLOW_MANAGER`, `TABLE_MANAGER` e `METADATA_MANAGER`, insieme alle *Procedure* `START_PROCESS`, `END_PROCESS`, `END_PROCESS_ERROR` e `END_PROCESS_TABLE`, formano l'infrastruttura di controllo dell'intero framework. Questo punto di monitoraggio centralizzato ha già consentito, nella fase di sviluppo, di gestire test e ripartenze in modo ordinato; il suo valore pieno si dispiegherà con l'esercizio continuativo in produzione.

Il livello L2 rappresenta infine la superficie esposta agli strumenti di analisi e reportistica. È in questo livello che il dato, già consolidato e arricchito nei livelli precedenti, viene modellato secondo le esigenze della piattaforma analitica di destinazione: le tabelle PUB ereditano dalle OUT il contenuto informativo completo, inclusa l'espansione delle chiavi surrogate prodotta dalla catena di join costruita in L1, e selezionano tramite una funzione `ROW_NUMBER()` l'immagine più aggiornata di ogni entità, pubblicandola in uno star schema ottimizzato per Power BI. In questo livello il framework rende pienamente visibile il proprio valore agli utenti finali, schermandoli dalla complessità dei processi a monte e offrendo un modello dati direttamente interrogabile per chiave surrogata.

È tuttavia importante riconoscere, con onestà intellettuale, i limiti del lavoro svolto. Il livello L2, sebbene progettato nelle sue strutture fisiche e implementato nei relativi *Mapping* ODI, non è stato ancora collegato agli strumenti di analisi finali: la migrazione verso Power BI richiede un confronto ancora in corso con i colleghi responsabili della reportistica, al fine di definire con precisione le esigenze di modellazione del layer di pubblicazione.

Tabella 5.1: Confronto strutturale tra i layer della pipeline ETL

Caratteristica	STG	DLT	OK	ODS	MDM	OUT	PUB
Livello	L0	L0	L1	L1	L1	L1	L2
JOBID	`JOBID_L0`	`JOBID_L0`	`JOBID_L1`	`JOBID_L1_INS`, `JOBID_L1_UPD`	`JOBID_L1_INS`, `JOBID_L1_UPD`	`JOBID_L1`	`JOBID_L2_INS`, `JOBID_L2_UPD`
Timestamp	`INS_TIME`	`INS_TIME`	`INS_TIME`	`INS_TIME`, `UPD_TIME`	`INS_TIME`, `UPD_TIME`	`INS_TIME`	`INS_TIME`, `UPD_TIME`
Constraint PK	Assente	Assente	Assente	Presente	Presente	Assente	Presente
Chiave surrogata	Assente	Assente	Assente	Assente	Presente (SK generata)	Presente (SK da join MDM)	Presente (SK chiave di accesso)
`FLG_NEG`	Assente	Presente	Presente	Presente	Presente	Presente	Presente
`FLG_SRC`	Assente	Assente	Assente	Assente	Presente	Presente	Presente
Scopo principale	Replica sorgente	Calcolo delta	Selezione dato valido	Consolidamento e storicizzazione	Armonizzazione anagrafica e SK	Preparazione per pubblicazione	Esposizione agli strumenti analitici

DLT e DLT_HIS condividono struttura identica e sono alimentate dallo stesso *Mapping* tramite componente SPLIT, per questo motivo nella tabella si fa riferimento solamente a DLT.

Guardando al percorso nel suo insieme, ciò che emerge con maggiore evidenza non è il valore o la complessità dei singoli componenti, ma piuttosto la coerenza del disegno architetturale che li integra. La progettazione di un sistema di integrazione robusto richiede infatti, prima ancora delle competenze tecniche necessarie alla sua implementazione, una chiara visione di come l'informazione debba muoversi, trasformarsi e rendersi disponibile all'interno di un'organizzazione. Le scelte architetturali descritte in questa tesi sono state orientate da tale impostazione, fondata su principi teorici consolidati e su un'analisi concreta delle esigenze operative reali. In questa prospettiva, un sistema informativo ben progettato non è soltanto quello in grado di risolvere le esigenze presenti, ma soprattutto quello strutturato per sostenere nel tempo l'evoluzione futura dei processi informativi.

5.2 Sviluppi futuri

Lo svolgimento di questa tesi ha coperto l'implementazione e la validazione dei livelli L0, L1 e L2 del framework, ciascuno dotato della propria struttura di orchestrazione interna. Il passo successivo naturale, che rappresenta al contempo il completamento architetturale del progetto e il prerequisito per il suo utilizzo in produzione, è la costruzione di un *Package* di orchestrazione unificato che coordini l'esecuzione dell'intera pipeline, orchestrando in sequenza i livelli L0, L1 e L2 all'interno di un unico punto di ingresso controllato. Attualmente i tre livelli sono eseguibili in modo indipendente, ma l'introduzione di un orchestratore globale consentirebbe di gestire in modo centralizzato le dipendenze tra i livelli, garantendo che L1 venga avviato solo al completamento di L0, e che L2 segua a sua volta soltanto dopo la conclusione validata di L1. Questo meccanismo di dipendenza verticale è già parzialmente formalizzato nella logica della procedura `END_PROCESS`, che aggiorna il campo `LOAD` del livello precedente al termine di ogni esecuzione, ma richiede di essere completato a livello di orchestrazione ODI con un load plan o un *Package* di coordinamento che legga questi stati e governi l'avanzamento dell'intera catena.

Una volta completata l'orchestrazione unificata, il secondo sviluppo previsto è il rilascio della pipeline in un contesto di produzione. Questo passaggio, apparentemente tecnico, comporta in realtà una serie di attività di validazione che vanno ben oltre la sola messa in esercizio degli scenari ODI: è necessario definire le finestre di schedulazione appropriate per ciascun livello, verificare che i tempi di esecuzione siano compatibili con le esigenze operative, configurare correttamente i contesti di produzione nella Topology di ODI e stabilire le procedure di monitoraggio continuativo dei processi tramite le tabelle di metadati già predisposte.

È proprio in questa fase che ci si aspetta l'emersione di micro-problematiche che, per loro natura, non possono essere anticipate in un ambiente di sviluppo o di test, come anomalie nei dati sorgente non riscontrate durante la fase di

sviluppo, casi limite nelle logiche di deduplicazione o nelle regole di qualità della fase OK, disallineamenti temporali tra i cicli di aggiornamento delle diverse sorgenti o comportamenti inattesi nelle logiche di `GREATEST` nei *Mapping* MDM e OUT in presenza di combinazioni di dati reali. La gestione di questi aggiustamenti costituisce parte integrante del ciclo di vita di qualsiasi sistema di integrazione dati: il framework è stato progettato con sufficiente modularità da consentire interventi puntuali su singoli *Mapping* o *Procedure* senza dover modificare l'intera catena, ma solo l'esecuzione continuativa in produzione potrà rivelare con precisione quali componenti richiedono affinamenti e in quale direzione.

Bibliografia

[1] I3P – Incubatore del Politecnico di Torino. *Mediamente Consulting*. 2026. URL: https://www.i3p.it/startup/mediamente-consulting (cit. a p. 1).

[2] Var Group. *Chi siamo*. 2026. URL: https://datascience.vargroup.com/en/Chi-siamo (cit. a p. 1).

[3] W. H. Inmon. *Building the Data Warehouse*. 3ª ed. New York, NY: John Wiley & Sons, 2002, p. 432. ISBN: 978-0-471-08130-2 (cit. alle pp. 5, 6, 14, 31).

[4] Ralph Kimball e Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. 3ª ed. Indianapolis, IN: Wiley, 2013, p. 608. ISBN: 978-1-118-53080-1 (cit. alle pp. 6, 14, 15, 32).

[5] Matteo Golfarelli e Stefano Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. 1ª ed. New York, NY: McGraw-Hill, 2009, p. 480. ISBN: 978-0-07-161039-1 (cit. alle pp. 6, 7, 30, 31).

[6] Khawaja Ubaid Ur Rehman, Umair Ahmad e Sajid Mahmood. «A Comparative Analysis of Traditional and Cloud Data Warehouse». In: *VAWKUM Transactions on Computer Sciences* 15.1 (2018), pp. 34–40. ISSN: 2308-8168 (cit. a p. 9).

[7] Darko Golec, Ivan Strugar e Drago Belak. «The Benefits of Enterprise Data Warehouse Implementation in Cloud vs. On-premises». In: *ENTRENOVA – Enterprise in Theory and Practice* 7.1 (2021), pp. 66–74. DOI: 10.54820/DMZS9230 (cit. a p. 9).

[8] Bhushan Fadnis. «Evolving Data Warehouse Architectures from On-Premises to Cloud». In: *International Journal of Science and Research (IJSR)* 13.4 (2024), pp. 1832–1834. DOI: 10.21275/SR24428084024 (cit. a p. 9).

[9] Dheeraj Kumar Bansal. «Enterprise Data Warehouses: Types, Benefits, and Considerations». In: *International Research Journal of Modernization in Engineering Technology and Science (IRJMETS)* 7.3 (2025), pp. 1588–1599. DOI: 10.56726/IRJMETS68496 (cit. a p. 10).

[10] Khalid Ibrahim Mohammed. «Data Warehouse Design and Implementation Based on Star Schema vs. Snowflake Schema». In: *International Journal of Academic Research in Business and Social Sciences* 9.14 (2019), pp. 25–38. DOI: 10.6007/IJARBSS/v9-i14/6502 (cit. alle pp. 12, 32).

[11] Georgia Garani e Sven Helmer. «Integrating Star and Snowflake Schemas in Data Warehouses». In: *International Journal of Data Warehousing and Mining* 8.4 (2012), pp. 1–27. DOI: 10.4018/jdwm.2012100102 (cit. a p. 13).

[12] Ralph Kimball e Margy Ross. *Dimensional Modeling Techniques*. White Paper. Kimball Group, set. 2013. URL: https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/ (cit. alle pp. 13, 14).

[13] Alkis Simitsis, Panos Vassiliadis e Timos Sellis. «Optimizing ETL Processes in Data Warehouses». In: *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*. Tokyo, Japan: IEEE, 2005, pp. 564–575. DOI: 10.1109/ICDE.2005.97 (cit. alle pp. 15, 16, 32).

[14] Papa Senghane Diouf, Aliou Boly e Samba Ndiaye. «Performance of the ETL Processes in Terms of Volume and Velocity in the Cloud: State of the Art». In: *2017 IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. Bahrain: IEEE, 2017, pp. 1–5. DOI: 10.1109/ICETAS.2017.8277875 (cit. alle pp. 15, 20).

[15] Papa Senghane Diouf, Aliou Boly e Samba Ndiaye. «Variety of Data in the ETL Processes in the Cloud: State of the Art». In: *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*. Bangkok, Thailand: IEEE, 2018, pp. 1–5. DOI: 10.1109/ICIRD.2018.8376308 (cit. alle pp. 15, 20).

[16] Panos Vassiliadis. «A Survey of Extract-Transform-Load Technology». In: *International Journal of Data Warehousing and Mining* 5.3 (2009), pp. 1–27. DOI: 10.4018/jdwm.2009070101 (cit. alle pp. 15–17, 32, 33).

[17] Microsoft Corporation. *SQL Server Integration Services Documentation*. Microsoft Learn Documentation. Microsoft. 2024. URL: https://learn.microsoft.com/en-us/sql/integration-services/ (cit. a p. 19).

[18] Ranjith Katragadda, Sreenivas Sremath Tirumala e David Nandigam. «ETL Tools for Data Warehousing: An Empirical Study of Open Source Talend Studio versus Microsoft SSIS». In: *Proceedings of the World Congress on Computer Applications and Information Systems (WCCAIS 2015)*. Hammamet, Tunisia: IEEE, 2015 (cit. a p. 19).

[19] Oracle Corporation. *Oracle Integration Cloud Documentation*. Oracle. 2024. URL: https://docs.oracle.com/en/cloud/paas/integration-cloud/ (cit. a p. 19).

[20] Oracle Corporation. *Understanding Oracle Data Integrator 14c (14.1.2.0.0)*. Fusion Middleware Documentation Guide G10248-01. Oracle. 2024. URL: https://docs.oracle.com/en/middleware/fusion-middleware/data-integrator/14c/ (cit. a p. 19).

[21] Giuseppe Fusco e Lerina Aversano. «An Approach for Semantic Integration of Heterogeneous Data Sources». In: *PeerJ Computer Science* 6 (2020), e254. DOI: 10.7717/peerj-cs.254 (cit. alle pp. 20, 21, 23, 32, 33).

[22] Tibor Horak, Peter Strelec, Michal Kebisek, Pavol Tanuska e Andrea Vaclavova. «Data Integration from Heterogeneous Control Levels for the Purposes of Analysis within Industry 4.0 Concept». In: *Sensors* 22.24 (2022), p. 9860. DOI: 10.3390/s22249860 (cit. alle pp. 21, 33).

[23] Muhammad Intizar Ali. «Integration of Distributed Heterogeneous Web Data Sources». In: *Journal of Web Engineering* 10.1 (2011), pp. 27–54 (cit. alle pp. 21, 26, 33).

[24] Paraskevas Koukaras. «Data Integration and Storage Strategies in Heterogeneous Analytical Systems: Architectures, Methods, and Interoperability Challenges». In: *Information* 16.11 (2025), p. 932. DOI: 10.3390/info16110932 (cit. alle pp. 21, 24–26, 33).

[25] Chiara Rucco, Motaz Saad e Antonella Longo. *Formalizing ETLT and ELTL Design Patterns and Proposing Enhanced Variants: A Systematic Framework for Modern Data Engineering*. 2025. DOI: 10.48550/arXiv.2511.03393. arXiv: 2511.03393 [cs.DB]. URL: https://arxiv.org/abs/2511.03393 (cit. alle pp. 24, 29, 33).

[26] Xiufeng Liu. *Optimizing ETL Dataflow Using Shared Caching and Parallelization Methods*. 2014. DOI: 10.48550/arXiv.1409.1639. arXiv: 1409.1639 [cs.DB]. URL: https://arxiv.org/abs/1409.1639 (cit. alle pp. 24, 26).

[27] Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri e Engelbert Mephu Nguifo. «An Experimental Survey on Big Data Frameworks». In: *Future Generation Computer Systems* 86 (2018), pp. 546–564. DOI: 10.1016/j.future.2018.04.032 (cit. alle pp. 25, 26).

[28] Chiara Rucco, Antonella Longo e Motaz Saad. *Efficient Data Ingestion in Cloud-based Architecture: a Data Engineering Design Pattern Proposal*. 2025. DOI: 10.48550/arXiv.2503.16079. arXiv: 2503.16079 [cs.DB]. URL: https://arxiv.org/abs/2503.16079 (cit. a p. 25).

[29] Alkis Simitsis, Spiros Skiadopoulos e Panos Vassiliadis. «The History, Present, and Future of ETL Technology». In: *Proceedings of the 25th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP 2023)*. Test-of-Time Award – Invited Talk. 2023. URL: https://ceur-ws.org/Vol-3379/ (cit. alle pp. 27, 33).

[30] Alan Willie, Dong Alan e Catherine Benz. *Best Practices for Optimizing ETL Workflows in Data Warehousing*. Set. 2023. URL: https://www.researchgate.net/publication/387746068_Best_Practices_for_Optimizing_ETL_Workflows_in_Data_Warehousing (cit. a p. 28).

[31] Christian Haase, Timo Roseler e Mattias Seidel. *METL: A Modern ETL Pipeline with a Dynamic Mapping Matrix*. 2022. DOI: 10.48550/arXiv.2203.10289. arXiv: 2203.10289 [cs.DB]. URL: https://arxiv.org/abs/2203.10289 (cit. alle pp. 28, 33).

[32] Deepak Chanda. «Automated ETL Pipelines for Modern Data Warehousing: Architectures, Challenges, and Emerging Solutions». In: *The Eastasouth Journal of Information System and Computer Science* 1.3 (apr. 2024), pp. 209–212. DOI: 10.58812/esiscs.v1i03 (cit. alle pp. 29, 33).