ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

PROGETTAZIONE E PROTOTIPAZIONE DI UN FRAMEWORK PER CONVERSATIONAL OLAP

Elaborato in

DATA MINING

Relatore

Prof. MATTEO GOLFARELLI

Presentata da

SARA SINTONI

Co-relatore

Dott. MATTEO FRANCIA

Seconda Sessione di Laurea

Anno Accademico 2018 – 2019

PAROLE CHIAVE

Conversational

Business Intelligence

OLAP

Indice

Sommario

I sistemi conversazionali sono quelli che interagiscono con gli utenti utilizzando il linguaggio umano, definito naturale. Il vantaggio principale è la loro facilità di utilizzo: i sistemi diventano accessibili anche a persone non esperte di informatica. Quando questi sistemi consentono agli utenti di accedere alle informazioni memorizzate nei database si parla di una vera e propria democratizzazione di accesso ai dati. Conversational OLAP vuole sottolineare che il nostro framework si pone all'intersezione tra due grandi aree di lavoro: i sistemi "conversazionali" e le interrogazioni OLAP. L'obiettivo della tesi è la progettazione e la prototipazione di un sistema per rendere più semplice l'accesso alle informazioni nei data warehouse, consentendo agli utenti di eseguire delle vere e proprie sessioni di analisi utilizzando un approccio conversazionale. Rispetto alle classiche interfacce in grado di tradurre frasi in linguaggio naturale in query SQL, il nostro sistema dovrà essere in grado di sostenere una vera e propria conversazione con l'utente: facendogli domande e interpretando le sue risposte; proprio come un assistente digitale. Questo aspetto consentirà non solo di utilizzare i feedback dell'utente per migliorare le capacità di comprensione del sistema, ma anche di poter eseguire vere e proprie sessioni di analisi OLAP. Queste ultime si differenziano dalle classiche interrogazioni SQL proprio per la loro interattività. Il nostro sistema dovrà quindi essere in grado di riconoscere per ogni nuovo inserimento se si tratta di una nuova query

o di una modifica ai risultati precedenti, e nel caso modificare la query precedentemente eseguita. La query dovrà poi essere eseguita sul data warehouse e i risultati verranno mostrati all'utente. Nel documento è presentata l'architettura funzionale progettata per la costruzione di un framework per raggiungere i risultati descritti.

Introduzione

La comprensione del linguaggio naturale da parte dei dispositivi informatici è oggetto di ricerche sin dagli anni '50. Numerose sono le sfaccettature e le applicazioni di questi sistemi negli anni, tra cui iniziano a comparire le interfacce per accedere alle informazioni memorizzate nei database in grado di comprendere le richieste degli utenti in linguaggio naturale: si parla di una vera e propria democratizzazione di accesso ai dati, rendendoli disponibili anche a tutte quelle persone che non conoscono i linguaggi artificiali per l'interrogazione dei database. Il nostro framework nasce dall'idea di estendere queste interfacce all'ambito della Business Intelligence, e in particolare alle interrogazioni sui data warehouse.

"Conversational OLAP" vuole sottolineare che il nostro framework si pone all'intersezione tra due grandi aree di lavoro: i sistemi "conversazionali" e le interrogazioni OLAP; per questo motivo verrà chiamato C-BI. Non vuole solo essere in grado di comprendere le query dell'utente in linguaggio naturale, ma sostenere una vera e propria conversazione con esso: facendogli domande e interpretando le sue risposte; proprio come un assistente digitale. Questo aspetto consentirà non solo di utilizzare i feedback dell'utente per migliorare le capacità di comprensione del sistema, ma anche di poter eseguire vere e proprie sessioni di analisi OLAP. Queste ultime si differenziano dalle classiche interrogazioni SQL proprio per la loro interattività: la sessione di analisi è,

per definizione, non prevedibile e costruita dall'utente in modo interattivo, producendo una serie di interrogazione sui dati, ognuna delle quali eseguita per differenza rispetto all'interrogazione precedente. Il nostro sistema dovrà quindi memorizzare i risultati di una interrogazione e comprendere, per ogni nuovo inserimento, se si tratta di una modifica ai dati appena visualizzati (e in questo caso modificare la query memorizzata) o se si tratta di una nuova interrogazione e di conseguenza eliminare ciò che ha in memoria e ripartire con la fase di comprensione da zero. Indipendentemente dal tipo di inserimento, la query dovrà essere poi eseguita realmente sul data warehouse, e i risultati dovranno essere mostrati all'utente.

Un limite di molte delle interfacce presentate in letteratura è la poca generalizzazione possibile: sono in grado di riconoscere ed eseguire query su specifici database. C-BI vuole invece essere un framework generalizzabile: in grado di comprendere ed eseguire query su qualsiasi cubo.

L'obbiettivo della tesi è la progettazione e la prototipazione di un sistema che raggiunga gli obiettivi appena elencati: occorre quindi un'analisi approfondita delle interfacce già presentate in letteratura, per conoscere quali sono le principale tecniche utilizzate per affrontare queste problematiche e i loro eventuali limiti, per valutare quali approcci è possibile riutilizzare parzialmente e quali invece hanno portato a risultati fallimentari. Come risultato della fase di progettazione sarà presentata nel documento l'architettura funzionale su cui si baserà il nostro framework; per quanto riguarda la prototipazione è stata sviluppata una porzione del sistema e una semplice applicazione web che tramite un'interfaccia grafica consente all'utente di inserire una nuova frase nel sistema e mostrare i risultati di analisi di essa raggiunti fin'ora.

Il documento è così composto:

1. Nel Capitolo 1 viene fatta una presentazione del problema, definendo

in modo dettagliato gli obiettivi del nostro framework e quali vantaggi presenterà rispetto ai principali strumenti di analisi OLAP.

2. Nel Capitolo 2 vengono date le nozioni di base necessarie poi a comprendere come funziona il sistema e perché sono state fatte alcune scelte. Prima di tutto viene introdotto il tema della Business Intelligence e in particolare dei data warehouse, spiegando i concetti fondamentali che sono utilizzati nella progettazione di C-BI. Vengono poi presentati i risultati delle ricerche in merito alle interfacce per l'accesso ai dati memorizzati nei database, con una spiegazione dettagliate di alcune di esse.

3. Nel Capitolo 3 è possibile vedere nel dettaglio l'architettura funzionale di quello che sarà il nostro framework, frutto del lavoro di progettazione; con la presentazione dettagliata dei moduli che già sono stati implementati.

4. Nel Capitolo 4 sono presenti i risultati ottenuti eseguendo i test sulla porzione di sistema sviluppato, con una presentazione delle metriche utilizzate per valutarlo.

Capitolo 1

Conversational OLAP

Al giorno d'oggi una delle maggiori tendenze di ricerca in ambito informatico è la democratizzazione di accesso, visualizzazione e analisi di dati, per evitare che gli utenti finali debbano necessariamente avere competenze informatiche per accedere ai vari servizi che consentono di fare queste operazioni. Gli assistenti digitali presenti in tutti gli smartphone (Siri, Cortana, Google Now/Home...), e i vari altoparlanti intelligenti (Amazon Echo, Alexa...), così come i servizi di *auto machine learning* [27] sono un esempio di successo di queste ricerche, ormai in uso comune sia nelle aziende [28] che nelle nostre vite di tutti i giorni.

In particolare, le interfacce per l'analisi del linguaggio naturale, scritto o parlato, verso i database hanno aperto nuovi opportunità per l'esplorazione e l'analisi dei dati [29]. In letteratura sono stati proposti numerosi contributi, i primi sistemi risalgono agli anni '70, come ad esempio LUNAR [8] o LADDER [9], molto limitati e poco generalizzabili; fino ad arrivare al giorno d'oggi con interfacce sempre più efficaci: NaLIR [15], ATHENA [14], ANALYZA [40] o TEMPLAR [41].

Tuttavia, i sistemi basati su linguaggio naturale sono complessi a causa

dell'estrema variabilità di questo linguaggio: basti pensare a sinonimi (parole diverse che hanno lo stesso significato), omonimi (parole uguali con significati diversi), diverse coniugazioni dello stesso verbo, etc. Inoltre, se si parla di testo scritto entrano in gioco tutta un'altra serie di complessità data dai possibili modi di scrivere la stessa parola (U.S.A. e USA sono entrambi corretti e hanno lo stesso significato, così come Emilia-Romagna e Emilia Romagna), gli accenti che spesso non vengono inseriti, la questione del separatore di token (Emilia-Romagna è una parola o sono due termini separati?), gli errori di battitura; senza considerare la complessità nel gestire date (10/10/19, Ott. 10, 2019, 10-10-2019, etc.) e numeri. Entrano in gioco poi complessità legate alla vera e propria comprensione del testo: spesso parlando si fa riferimento a un contesto che il computer non può conoscere, è possibile utilizzare sarcasmo, estremamente difficile da comprendere per un dispositivo. Infine, tutto ciò che è legato alla comprensione del linguaggio naturale è strettamente legato alla lingua che si si sta prendendo in considerazione, richiedendo uno sforzo non indifferente per adattare ciò che è stato sviluppato ad altri linguaggi. Ad esempio lo stesso algoritmo può dare ottimi risultati se applicato a parole di una certa lingua ma non altrettanto buoni se utilizzato in un altro linguaggio. È necessario poi considerare che ci sono obiettivi più semplici da raggiungere, in cui non è necessaria una piena comprensione del testo inserito, come ad esempio l'analisi del sentimento di un testo (positivo o negativo) o classificazione di testi; mentre in C-BI si avranno più che altro frasi brevi, e non testi complessi, ma che devono essere comprese completamente.

L'idea alla base di C-BI è l'estensione di queste interfacce per accedere ai database relazionali a sistemi di Business Intelligence, progettando un sistema appositamente sviluppato per questo contesto, in grado di utilizzare la conoscenza per migliorare la comprensione di del linguaggio naturale. La Business

Intelligence (abbreviata BI) è un insieme di strumenti e procedure utilizzati dalle aziende per comprendere i propri dati, in modo da sfruttarli in modo ottimale durante il processo decisionale. In particolare si parlerà dello strumento di BI chiamato "data warehouse": un grande contenitore in cui integrare tutti i dati operazionali dell'azienda, per poi consentirne la consultazione anche a persone non esperte di informatica. La principale modalità di fruizione delle informazioni contenute al suo interno sono le analisi OLAP: interrogazioni multidimensionali interattive che richiedono la scansione di moltissimi dati per calcolare dati numerici di sintesi. Si parla di vere e proprie "sessioni" di analisi OLAP perché queste vengono costruite attivamente dagli utenti, eseguendo una sequenza di interrogazioni formulate per differenza rispetto alla precedente. Per spiegazione più dettagliate relativamente a BI, data warehouse e interrogazioni OLAP vedere la Sezione 2.1.

Lo scopo del nostro framewrok è quindi poter eseguire sessioni di analisi OLAP su i data warehouse, consentendo agli utenti di esprimere le proprie richieste in linguaggio naturale , in modo testuale o tramite comando vocale.

I data warehouse, e in particolare OLAP, nascono per definizione per semplificare l'accesso ai dati, ma "il carattere estemporaneo delle sessioni di lavoro, l'approfondita conoscenza dei dati richiesta, la complessità delle interrogazioni formulabili e l'orientamento verso utenti tipicamente non esperti di informatica rendono cruciale il ruolo dello strumento utilizzato, la cui interfaccia deve necessariamente presentare ottime caratteristiche di flessibilità, facilità d'uso ed efficacia." [16] Il sistema utilizzato per accedere ai dati diventa quindi un elemento fondamentale per l'effettiva utilità di questi strumenti. Sul mercato esistono diverse interfacce che consentono di eseguire sessioni di analisi OLAP: IBM Cognos[1], Microsoft Analysis Services[2], Micro Strategy Intelligence Ser-

[1] `https://www.ibm.com/products/cognos-analytics`

[2] `https://azure.microsoft.com/en-us/services/analysis-services/`

ver[3], etc. Tutte queste interfacce sono di tipo *"point-and-click"*: interfacce in cui un'azione viene selezionata posizionando un cursore sopra la sua rappresentazione sul display usando un dispositivo di puntamento (ad esempio il mouse), e viene quindi avviata facendo clic [38]. Quello che consentono di fare è selezionare quali misure mostrare e quale operatore di aggregazione utilizzare, come raggruppare i dati, e qualche impostazione di visualizzazione (quali valori usare nelle righe e nelle colonne...) per poi tradurre il tutto in modo automatico in una query SQL e mostrare i risultati. Quello che vuole offrire in più C-BI è la possibilità di dialogare con con un assistente digitale per eseguire sessioni di analisi OLAP; questo, oltre a ridurre la complessità di esecuzione di una query, consente all'utente di inserire dei feedback per raffinare e correggere le query errate e può essere utilizzato in scenari in cui interfacce *"hand free"* (interfacce con cui gli utenti possono interagire senza l'utilizzo delle mani e dei classici dispositivi di input: mouse e tastiera) sono obbligatorie.

L'obbiettivo della tesi è quindi la progettazione e la prototipazione di un sistema che sia in grado di:

1. Comprendere la richiesta dell'utente (sia che se sottoposta al sistema in formato testuale che tramite comando vocale);

2. Comunicare con l'utente in caso di incertezza su come interpretare la frase;

3. Eseguire la query sul data warehouse e mostrare all'utente il risultato;

4. Consentire all'utente di raffinare il risultato visualizzato senza riscrivere interamente la query ma solo esplicitando le modifiche (ovvero comprendere non solo una singola query ma tutta la sessione di analisi).

[3] `https://community.microstrategy.com/s/?language=en_US`

Un altro requisito del sistema è la generalità: il sistema deve essere in grado di eseguire query su qualsiasi data warehouse con una fase di configurazione iniziale manuale, per ogni nuovo data warehouse, non esageratamente complessa.

Una porzione del sistema è già stato progettato nel dettaglio e anche sviluppato, si tratta del cuore del sistema: ovvero i moduli in grado di comprendere la query intera; mentre l'aspetto più "conversazionale" legato all'interazione in linguaggio naturale con l'utente è stato solamente progettato ad alto livello ma non ancora sviluppato. Per alcune funzionalità richieste dal sistema si è deciso di affidarsi a API e/o strumenti presenti sul mercato.

Capitolo 2

Stato dell'arte

C-BI può essere classificato come una *natural language interface* (NLI) su sistemi di Business Intelligence. Si colloca quindi all'intersezione tra due grandi aree di lavoro: la Business Intelligence e i sistemi di *query answering*. Nella Sezione 2.1 viene presentata la Business Intelligence: cos'è e alcuni dei principali concetti che vengono poi utilizzati in C-BI. Segue poi la Sezione 2.2 in cui vengono presentati i Chatbot e il loro funzionamento. Infine, nella Sezione 2.3, viene presentato il *query answering* con un focus particolare sul *question answering* su dati strutturati ovvero le *Natural Language Interfaces to Database* di cui verranno analizzate alcune delle tecniche maggiormente utilizzate e due delle maggiori soluzioni presentate in letteratura.

2.1 Modellazione multidimensionale e OLAP

La Business Intelligence è un insieme di strumenti e procedure che consentono a un'azienda di trasformare i propri dati di business in informazioni utili al processo decisionale, da rendere disponibili alla persona giusta e nel formato idoneo [16]. La sua importanza sta crescendo sempre di più negli ultimi an-

ni grazie all'aumento del volume dei dati operazionali (ovvero i dati generati da operazioni svolte all'interno dei processi gestionali) che rende i computer l'unico strumento possibile per analizzare questi dati.

Lo strumento di business intelligence più diffuso è il *data warehouse*: una collezione di dati di supporto per il processo decisionale che presenta le seguenti caratteristiche:

- Orientato ai soggetti di interesse: in quanto si incentra sui concetti di interesse dell'azienda.
- Integrato e consistente: poiché il data warehouse integra i dati provenienti da diverse sorgenti (diversi database o addirittura diversi sistemi informativi che possono essere anche esterni all'azienda).
- Rappresentativo dell'evoluzione temporale e non volatile: i data warehouse devono consentire analisi che spazino sulla prospettiva di alcuni anni. I dati operazionali vengono quindi caricati dalle sorgenti a intervalli regolari nel nuovo contenitore e mai cancellati.

Per comprendere meglio l'utilità di questi strumenti si pensi a una grande azienda con tante filiali: in ognuna di queste filiali avvengono ogni giorno numerose operazioni di operatività standard che generano un elevato numero di dati operazionali che vengono poi memorizzati nei singoli database delle filiali. Se i dirigenti dell'azienda volessero fare analisi a livello globale sarebbe necessario:

1. Accedere ai diversi database, cosa che il dirigente non ha le capacità di fare quindi bisogna far intervenire gli amministratori dei database nelle varie filiali.
2. I dati dei vari database devono essere integrati, con tutte le difficoltà che ci posso essere se sono stati progettati in modo indipendente.

3. Infine, da questa enorme quantità di dati di dettaglio devono essere calcolati dei valori di sintesi che riassumano l'andamento dell'azienda.

Il data warehouse ha l'obiettivo di semplificare queste operazioni, fungendo da unico contenitore in cui caricare tutti i dati operazionali provenienti dalle diverse sorgenti, in un formato tale che anche i non esperti informatici siano in grado di consultarli e così poter fare valutazioni finalizzate alla pianificazione e al processo decisionale. Il manager sarà quindi in grado, lui stesso, di accederci per avere a propria disposizione i dati di sintesi. Il data warehouse non vuole, però, sostituire i vecchi database, ma l'idea è quella di separare l'elaborazione legata alle transazioni (OLTP, *On-Line Transactional Processing*) da quella di tipo analitico (OLAP, *On-line Analytical Processing*). Le prime sono effettuate per supportare l'operatività quotidiana dell'azienda, sono quindi sempre le stesse che vengono ripetute in modo frequente e possono continuare ad essere effettuate sui database originali, come avviene già. Le altre, invece, effettuano un'analisi multidimensionale che richiede la scansione di un'enorme quantità di record per calcolare un insieme di dati numerici di sintesi che quantifichino le prestazioni dell'azienda; caratteristica principale di queste interrogazioni è l'interattività, che le rende sempre diverse tra loro e quindi non prevedibili. Queste ultime, sono quelle che saranno fatte sui data warehouse.

Il modello fondamentale per la rappresentazione e l'interrogazione dei dati nei data warehouse è il modello multidimensionale, molto utilizzato grazie alla sua semplicità e intuitività anche per utenti non esperti di informatica. Utilizzando questo modello è possibile vedere gli oggetti che influenzano il processo decisionale, ovvero i fatti di interesse, come dei cubi multidimensionali. Un data warehouse è quindi l'insieme di uno o più cubi, a seconda di quanti sono i fatti di interesse per l'azienda. Ogni occorrenza del fatto genera un evento, caratterizzato quantitativamente da misure numeriche. Questo corri-

sponde quindi a una cella del cubo che contiene un valore per ognuna di queste misure.

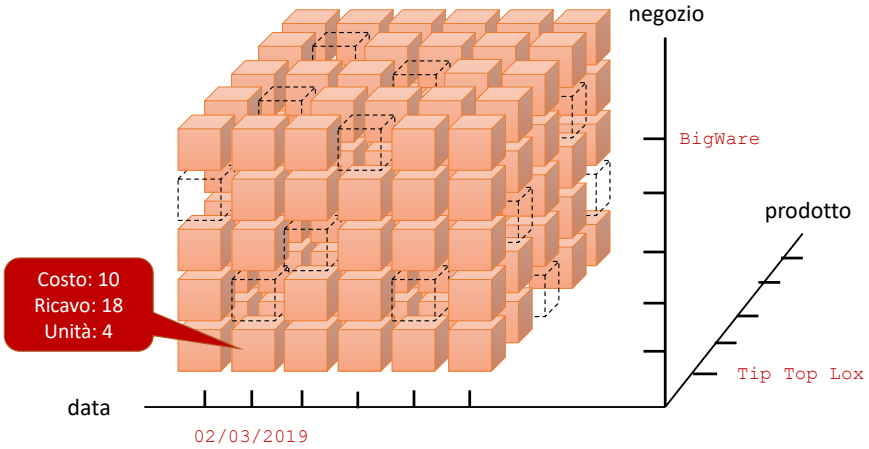


Figura 2.1: Esempio di un cubo che modella le vendite in una catena di negozi. Nel negozio "BigWare" il 02/03/2019 sono state vendute 4 unità del prodotto "Tip Top Lox" per un ricavo di 18 euro avendo sostenuto un costo di 10 euro.

Gli assi del cubo rappresentano le dimensioni di analisi del fatto, utilizzate per definire diverse prospettive per l'identificazione degli eventi. Gli eventi che genera un'azienda sono tantissimi, troppi per poter essere analizzati singolarmente, quindi ogni dimensione può essere la radice una gerarchia di livelli di aggregazione che ne raggruppa i valori in diversi modi. Una gerarchia è un albero direzionato i cui nodi sono attributi dimensionali (i livelli della gerarchia) e i cui archi modellano associazioni molti-a-uno tra coppie di attributi dimensionali. Gli attributi possono essere suddivisi in due macro-categorie: (1) categorici, definiscono delle caratteristiche qualitative dell'oggetto e solitamente sono discreti o binari ovvero hanno un numero finito di valori che possono assumere; (2) numerici che invece identificano caratteristiche quantitative e sono spesso continui, ovvero possono assumere valori reali e quindi infiniti. Nella

Figura 2.1 è mostrato graficamente un cubo di esempio che potrebbe essere la versione semplificata del data warehouse mostrato nel DFM nella Figura 3.1. Il fatto descritto è la vendita in una catena di negozi, le dimensioni di analisi sono: prodotto, negozio e data; un evento corrisponde alla vendita di un certo prodotto in un certo negozio un certo giorno ed è descritto da 3 misure: costo, ricavo e unità vendute. I cubetti "trasparenti" rappresentano la sparsità delle informazioni memorizzati nei cubi: non tutti i prodotti possono essere venduti tutti i giorni in tutti i negozi.

Una modellazione grafica di un cubo OLAP è presente nella Figura 3.1. Il modello utilizzato si chiama DFM (*Dimensional Fact Model*) [17] ed è un modello concettuale specificatamente concepito per fungere da supporto alla progettazione di data warehouse. La sua natura essenzialmente grafica lo rende comprensibile anche non esperti di informatica, in questo modo può essere utilizzato non solo come supporto alla progettazione ma anche come strumento per facilitare la comunicazione tra progettista e utente finale. La rappresentazione concettuale generata dal DFM consiste in un insieme di "schemi di fatto" che modellano gli elementi fondamentali di un cubo OLAP: fatti, misure, dimensioni e gerarchie. Lo schema di fatto mostra quindi il fatto al centro, in un quadrato con il nome del fatto più tutte le misure. Le dimensioni e tutti gli attributi dimensionali sono rappresentati da circoletti collegati tramite linee; le dimensioni sono collegate al fatto, le linee che collegano due attributi denotano gli archi delle gerarchie esprimendo le dipendenze funzionali. Nell'esempio in Figura 3.1 esiste una dipendenza tra PRODUCT_SUBCATEGORY e PRODUCT: il prodotto appartiene a una sola sottocategoria e una sottocategoria identifica un insieme di prodotti.

Le interrogazioni OLAP sono la principale modalità di fruizione delle informazione contenute in un data warehouse. Consentono a utenti, le cui necessità

di analisi non siano facilmente identificabili a priori di analizzare ed esplorare interattivamente i dati sulla base del modello multidimensionale. Un'altra differenza con le classiche interrogazioni transazionali è che in questo caso si può parlare di vere e proprie "sessioni OLAP" che consistono in un percorso di navigazione che riflette il procedimento di analisi di uno o più fatti di interesse sotto diversi aspetti e a diversi livelli di dettaglio. Questo percorso si concretizza in una sequenza di interrogazioni che spesso vengono formulate per differenza rispetto all'interrogazione precedente in quanto ogni passo della sessione di analisi è scandito dall'applicazione di un operatore OLAP che trasforma l'ultima interrogazione formulata in una nuova interrogazione. Esistono diversi operatori OLAP che consentono di: aumentare o ridurre il livello di aggregazione dei dati, eliminando o introducendo livelli di dettaglio in una delle gerarchie (*Roll-up / Drill-down*); altri operatori vengono utilizzati per visualizzare solo una porzione del cubo, fissando un valore o un range di valori per una o più dimensioni (*Slicing* e Selezione); esistono poi operatori che consentono di collegare cubi correlati tra loro (*Drill-across*) oppure di passare dai dati multidimensionali aggregati ai dati operazionali presenti nelle sorgenti (*Drill-through*); infine *pivoting* è un operatore utilizzato per modificare solamente la visualizzazione dei dati, invertendo le righe con le colonne.

In particolare, le query che saranno riconosciute da C-BI sono le query GPSJ [23] (*Generalized Projection, Selection, Join*) che sono composte da: join, predicato di selezione e aggregazione. Una query è quindi composta da:

1. MC: *measure clause*, unico elemento fondamentale poiché una GPSJ potrebbe aggregare totalmente una singola misura senza selezioni. È l'insieme delle misure il cui valore verrà restituito dalla query più eventuali operatori di aggregazione. Quando i risultati vengono aggregati bisogna infatti definire come aggregare le misure presenti nel risultato, nel caso

più semplice i valori vengono semplicemente sommati, ma ci sono alcune misure che vengono definite "non additive" su alcune dimensioni o su tutte; ovvero sommare i valori delle misure quando si aggrega su una certa dimensione porta ad avere un risultato privo di senso. Ad esempio, se è presente una misura che rappresenta il costo medio di un acquisto, probabilmente non ha senso calcolare la somma di questi valori medi, piuttosto conviene calcolarne nuovamente la media.

2. GBC: *group-by set*, ovvero un sottoinsieme dei livelli nelle gerarchie del cubo. Può non essere presente nessun attributo e in questo caso vengono mostrati i dati al massimo livello di dettaglio.

3. SC: *selection clause* composta da un insieme di clausole booleane sul valore degli attributi. Anche questo può essere vuoto.

Esistono diversi approcci per l'implementazione logica dei data warehouse, il più utilizzato è ROLAP: *Relational OLAP*, ovvero implementazione del cubo su un classico DBMS relazionale. In questo caso ci sono tante tabelle quante sono le dimensioni (chiamate "dimension table"), avente ognuna una chiave surrogata e un attributo per ogni attributo dimensionale della gerarchia che parte da quella dimensione; infine c'è una tabella, chiamata "fact table" che contiene un attributo per ogni misura del fatto e le chiavi importate di tutte le dimension table.

2.2 Chatbot

Vengono definiti "bot" quei software che possono eseguire dei task in modo automatico; quando utilizzati in piattaforme di messaggistica vengono chiamati "chatterbot" o più comunemente "chatbot". Il loro obiettivo è quello di

simulare conversazioni umane; abilitando la comunicazione attraverso comandi vocali, chat testuali o interfacce grafiche; consentendo così agli utenti di interagire con i dispositivi digitali utilizzando il linguaggio naturale. Basano il loro funzionamento su frasi di esempio che che vengono utilizzate per addestrate il sistema ad estrarre dall'input dell'utente: (1) intenti (cosa vuole raggiungere) e (2) entità (parametri per completare l'intento). In generale, Klüwer [1] riassume i passi di elaborazione di un chatbot in:

1. Pulizia dell'input: rimozione e sostituzione dei caratteri e dei termini inutili (smile, abbreviazioni, etc.).

2. Applicazione di un algoritmo di *pattern-matching* per verificare se uno o più pattern sono presenti nella frase in input.

3. Generazione delle risposte sulla base dei pattern presenti.

Il loro funzionamento si basa sulla ricerca di pattern all'interno delle frasi in input. I pattern sono dei modelli, definiti da stringhe facoltativamente arricchite con la sintassi delle espressioni regolari che possono essere ricercate all'interno della frase in input. Ad alcuni pattern sono associate più risposte, per garantire un minimo di variabilità, ad altri una unica; in ogni caso viene scelta una risposta e mostrata all'utente.

Anche se il nome venne introdotto solo successivamente, da Mauldin nell'omonimo sistema CHATTERBOT [2], il primo software "conversazionale" appartenente a questa categoria viene considerato ELIZA [3], sviluppato nel 1966 con l'obiettivo di simulare uno psicoterapeuta. Da allora, questi sistemi stanno comparendo in diverse aree di applicazione e in ogni dispositivo digitale. Alcuni hanno un puro scopo di intrattenimento, simulando semplici chiacchierate, molti chatbot sono stati creati nell'ambito dell'assistenza a svolgere determinati compiti, come per esempio per rispondere a domande in un sito

di e-commerce [4]; oppure con lo scopo di rispondere a domande in base a una conoscenza sottostante memorizzata in un database o in un'ontologia [5].

I vantaggi nell'utilizzo dei chatbot sono numerosi: possono assistere un numero infinito di persone contemporaneamente, 24 ore su 24, ogni giorno; migliorando anche l'efficienza. Inoltre i chatbot che si basano sull'intelligenza artificiale sono in grado di migliorare le loro conoscenze sfruttando le interazioni con gli utenti che li utilizzano.

Come analizzato in [6] esistono diversi framework che consentono lo sviluppo di questi chatbot che hanno poteri espressivi e funzionalità differenti ma il funzionamento di base è per tutti lo stesso: ogni intent che si vuole riconoscere necessita di frasi di training e per ogni entità è necessario inserire l'elenco dei possibili valori. Gli intenti sono quindi finiti e predefiniti in quanto questi chatbot sono orientati ai task.

Il motivo principale per cui C-BI è un sistema inadatto a essere sviluppato sfruttando questa tecnologia è che nel nostro caso l'intento è uno solo: eseguire la query; e la difficoltà principale è riconoscere il ruolo dei vari termini all'interno della frase per inserirli nel posto giusto nella query. Questo compito non può essere svolto sfruttando i concetti di entità statiche di questi framework; poiché il ruolo di un termine della frase nella query dipende non solo dal suo "tipo" (dimensione, misura, valore di un attributo, etc.), cosa che potrebbe comunque essere trovata usando i chatbot, ma anche dalle parole che sono nel suo intorno. Questo, rende C-BI più simile a un sistema di question answering su dati strutturati, di conseguenza inadatto ad essere sviluppato utilizzando i vari framewrok per lo sviluppo di chatbot.

2.3 Query answering

Query answering o anche *question answering* è una disciplina informatica che ha come obiettivo la creazione di sistemi in grado di rispondere automaticamente a domande poste dagli utenti in linguaggio naturale. I principali approcci di risoluzione sono due: *information retrieval-based* e *knowledge-based* [18]. L'*Information Retrieval (IR)* è l'insieme di metodi e algoritmi che, data una query posta dall'utente, cerca in un insieme di documenti non strutturati quali di questi sono rilevanti. Nel primo approccio, quindi, l'obiettivo è rispondere alla domanda cercando brevi porzioni di testo sul web o in altre raccolte di documenti. In questo caso la query da estrarre dalla richiesta dell'utente comprende solamente le *keyword* da dare in input al sistema di IR per cercare i documenti rilevanti; essendo questi sistemi molto generici solitamente, questa fase viene fatta sfruttando solo la semantica lessicale. Parlando invece di *knowledge-based question answering* si fa riferimento alla situazione in cui le informazioni utilizzate per rispondere alle domande sono memorizzate in forma strutturata; si può trattare di un vero e proprio database relazionale completo o un database strutturato più semplice che può essere composto, ad esempio, da semplici triple RDF (triple composte da un predicato con due argomenti che esprime una semplice relazione o una proposizione). In questi casi la richiesta dell'utente deve essere trasformata in una rappresentazione logica utilizzata poi per recuperare i dati dalla sorgente. Il dominio del discorso è conosciuto e limitato, e la meta-conoscenza (i nomi utilizzati nello schema del database) può essere sfruttata per ridurre lo spazio di ricerca e interpretare la richiesta dell'utente in modo migliore [21].

Tra i sistemi appartenenti alla seconda classe quelli di maggiore interesse per il nostro studio sono quelli che si occupano di comprendere query da eseguire su database strutturati, chiamati anche NLIDB (*Natural Language*

Interfaces to Databases); il loro scopo è quello di consentire agli utenti di accedere a informazioni memorizzate in un database esprimendo la richiesta in linguaggio naturale.

Questi sistemi hanno numerosi vantaggi:

+ Non richiedono la conoscenza da parte dell'utente di linguaggi specifici con cui vengono solitamente interrogati i database (come SQL).

+ Richiedono una conoscenza del dominio del discorso ma non la conoscenza della struttura fisica esatta dei dati memorizzati, ovvero i nomi delle tabelle, le relazioni tra loro e i nomi dei loro campi.

+ Sono semplici da utilizzare.

+ Non hanno bisogno di una fase iniziale di addestramento.

I primi tre in particolare sono caratteristiche fondamentali in quanto solitamente la persona che ha necessità di accedere ai dati non è un tecnico informatico. Hanno, però, anche alcuni limiti:

- Lavorano con un insieme limitato di linguaggi naturali.

- La copertura linguistica non è ovvia.

- L'utente da per scontato che il sistema sia "intelligente" e potrebbe fare assunzioni errate.

- La configurazione iniziale può essere tediosa.

È un ambito da anni oggetto di ricerca e in letteratura esistono già diverse soluzioni valide, in quanto le informazioni stanno assumendo un ruolo sempre più centrale nelle nostre vite, e i database sono una delle maggiori sorgenti [7]. Le prime interfacce risalgono all'inizio degli anni '70, tra cui il più conosciuto

è LUNAR [8], un sistema sistema in grado di rispondere a domande relative ai campioni di rocce trovate sulla luna che basa il suo funzionamento su due database: uno per le analisi chimiche e il secondo per i riferimenti bibliografici. Un altro esempio è LADDER [9], sviluppato per rispondere a domande espresse in linguaggio naturale relative alle navi della Marina americana. Basa il suo funzionamento su un'architettura a tre livelli: INLAND *(Informal Natural Langage Access to Navy Data)* che trasforma le domande e produce query, IDA *(Intelligent Data Access)* che crea le risposte per l'utente e infine FAM *(File Access Managemer)* che ha il compito di trovare la posizione dei file gestirne l'acceso nel database distribuito. Entrambi questi sistemi, quindi, sono pensati e progettati relativamente a uno specifico scenario di utilizzo su particolari database, per cui poco generalizzabli. I primi risultati interessanti, punto di partenza nelle ricerche attuali sono stati ottenuti negli anni '90, come riassunto nel documento [10].

In generale, il problema di accedere a database usando il linguaggio naturale può essere scomposto in due sottoparti: [7]

I. Componente linguistico: trasforma la query in linguaggio naturale in una query formale e genera un output in linguaggio naturale.

II. Componente database: esegue le tradizionali funzioni di gestione dei database.

Tutti i sistemi sviluppati in questo ambito possono essere suddivisi in quattro macro-aree in base alla loro architettura: [10]

1. Sistemi basati su *pattern matching*.

   Molto simili ai chatbot, vengono definiti un insieme di pattern e una serie di azione associate a questi. Il pattern viene poi ricercato all'interno della frase e, se presente, viene eseguita l'azione associata. Sono semplici da

implementare ed è semplice aggiungere o rimuovere feature ma il loro potere espressivo risulta essere molto limitato. SAVVY [30] è un esempio di sistema appartenente a questa classe.

2. Sistemi *syntax-based*.

   Basano la loro conoscenza su una grammatica che esprime la possibile struttura sintattica delle domande poste dell'utente, grazie a questa viene creato il *parse tree* sintattico della domanda posta dall'utente e poi trasformato direttamente in SQL. Fornisce informazioni dettagliate sulla struttura della frase ma non è semplice definire come i nodi devono essere mappati negli elementi della query poiché possono esserci dei mapping multipli o mapping inesistenti. LUNAR [8] è un esempio di sistema *syntax-based*.

3. Sistemi basati su *semantic grammar*.

   Nei sistemi di questo tipo la conoscenza alla base è una grammatica che esprime non solo la possibile struttura sintattica delle domande poste, ma anche quella semantica; anche in questo caso viene utilizzata per creare il *parse tree* sintattico della domanda posta dall'utente che viene poi trasformato direttamente in SQL. Rispetto ai precedenti l'ambiguità è ridotta ma richiede una conoscenza degli elementi del dominio che rende i sistemi poco generalizzabili. Appartengono a questa categoria LADDER [9] e PLANES [31].

4. Sistemi basati su un linguaggio di rappresentazione intermedio.

   Il linguaggio naturale viene prima trasformato in una query logica scritta in un particolare linguaggio interno che esprime la richiesta dell'utente in concetti di alto livello, indipendenti dalla struttura del database. Solo in

una seconda fase viene trasformata in un linguaggio per essere eseguita sul database. MASQUE/SQL [32] appartiene a questa classe.

L'approccio più classico a questo problema è l'utilizzo di *keyword*, e di sistemi appartenenti a questa categoria ne sono stati sviluppati diversi [11, 12, 13], ma il loro potere espressivo risulta molto limitato come dimostrato in [14]. Sono stati quindi pensati dei sistemi che superino l'idea delle *keyword*, cercando di comprendere la semantica della query, ovvero l'intento dell'utente.

2.3.1 Principali tecnologie utilizzate

Prima di entrare del dettaglio di due delle principali interfacce testuali di accesso ai database occorre fare un'introduzione alle tecnologie utilizzate da questi sistemi, che sono principalmente tre: grammatiche formali, Natural Language Processing e ontologie.

Grammatiche

Le grammatiche vengono utilizzate per descrivere un linguaggio formale tramite una serie di regole, anche ricorsive, che descrivono come formare le stringhe che vi appartengono partendo dall'alfabeto della lingua in base alla sintassi. La notazione tecnica utilizzata per definire queste grammatiche è definita *Backus-Naur form* [19]. La grammatiche più conosciute sono generative, ovvero le regole vengono utilizzate per generare tutte le possibile stringhe della lingua (che possono anche essere infinite), per farlo si comincia con una stringa composta dal solo simbolo iniziale (predefinito nella grammatica) e si applicano successivamente le regole per riscriverla. Quelle, però, utilizzate maggiormente nei sistemi di question answering sono grammatiche analitiche che funzionano da parser: data una stringa in input vengono utilizzate le regole per verificare se appartiene o meno al linguaggio definito dalla grammatica. Il risultato di

questa operazione è la generazione di un **parse tree**: un albero sintattico che mostra la struttura sintattica della stringa.

Natural Language Processing

Ovvero "elaborazione del linguaggio naturale". È un problema che si trova nell'intersezione tra linguistica, informatica e intelligenza artificiale. Riguarda le interazioni tra i computer e i linguaggi umani, appunto "naturali"; con l'obiettivo di programmare i computer in modo che siano in grado di comprendere anche questo tipo di linguaggio apprendendo algoritmi e modelli matematici del linguaggio umano. È una disciplina ampiamente studiata sin dagli anni '40 che trova applicazione in diversi ambiti, quello di maggior interesse parlando di NLIDB è la comprensione di un testo scritto in linguaggio naturale. Ora verranno spiegati alcuni concetti e alcune tecniche di NLP che verranno poi utilizzate in C-BI.

Pulizia dell'input Quando viene analizzato un testo inserito dall'utente esistono diverse tecniche necessarie per avere un input "pulito".

- Rimozione delle *"Stop Words"*: ovvero rimozione dal testo in input di tutte quelle parole che sono molto presenti all'interno di quasi tutti i testi ma hanno uno scarso contenuto semantico (congiunzioni, preposizioni, articoli, etc.), definite, appunto, "stop words". Sul web esistono numerosi elenchi di stop words generali che possono essere estesi con i termini di scarso interesse dello specifico dominio di interesse del problema.

- Normalizzazione dei token: nel linguaggio naturale succede spesso che alcune parole, anche se scritte in modo diverso, hanno lo stesso significato. Questo può valere sia per diverse versioni della stessa parola (USA, U.S.A. e United States of America anche se sono scritte in modo diverso

hanno lo stesso significato), sia per le diverse coniugazioni della stessa parola, etc. Le varie forme della stessa parola devono quindi essere ricondotte alla stessa versione, ai fine della comprensione da parte del dispositivo. Esistono molte tecniche che cercano di risolvere questo problema, quella utilizzata all'interno di C-BI è la "lemmatizzazione" che riduce le diverse forme flesse della stessa parole nel singolo termine del dizionario (ad esempio tutti i verbi vengono ricondotti all'infinito).

Sinonimi Spesso analizzando il linguaggio naturale è necessario considerare non solo differenze su come due parole sono scritte ma anche sul loro significato, per fare questo è possibile utilizzare i sinonimi. Per far fronte a questo problema viene spesso utilizzato WordNet [20]: un grafo direzionato dove ogni nodo è una parola inglese e ogni arco una relazione direzionata tra due parole, tra cui anche la relazione di sinonimia.

N-gram È una tecnica in cui data una sequenza ordinata di elementi (in questo caso le parole della frase) vengono analizzate tutte le sue sottosequenze di n elementi, gli n-grammi appunto. Viene quindi fatta scorrere una finestra lunga N sulla frase in input e prese in considerazione tutti i possibili gruppi di N parole vicine. In alcuni casi vengono analizzati solo gruppi composti da N parole; quando, ad esempio, si sta ricercando qualcosa di specifico di cui si conosce la lunghezza. Altre volte, invece, se si vuole dare una maggiore robustezza al sistema, viene fatta un'analisi prendendo i gruppi di parole da 1 a N. È un tecnica molto utilizzata per creare i modelli di linguaggio, ovvero i sistemi che assegnano una certa probabilità ad una parola in presenza di altre parole; ma sono anche molto utilizzati nel riconoscimento vocale e della scrittura a mano, e nei sistemi di correzione ortografica e traduzione automatica.

Parse tree linguistico Un altro concetto legato al natural language processing molto utilizzato in questo ambito sono i parse tree linguistici; spesso quando si usa il termine *parse tree* si fa riferimento a questo tipo di alberi. Il parse tree linguistico viene generato da una frase in linguaggio naturale applicandoci sopra una serie di regole per una specifica lingua. La generazione di questi parse tree è un problema molto noto in letteratura che richiede numerose conoscenze di linguistica. Fortunatamente sul web esistono diverse librerie in grado di svolgere questo compito con ottimi risultati, la più famosa è lo Stanford Parser [1] che è in grado anche di comprendere se ci sono gruppi di parole che devono essere considerati come unica *phrase* e quali parole sono soggetto o oggetto di un verbo e infine le relazioni grammaticali tra le varie parole.

Ontologia

Nel contesto informatico un'ontologia definisce un insieme di primitive rappresentative con cui modellare un dominio di conoscenza o un discorso. La primitive di solito sono: classi (o insiemi), attributi (o proprietà) e relazioni (o relazioni tra membri delle classi). La definizione delle primitive include informazioni sul loro significato e vincoli relativi alla loro applicazione. Molti dei sistemi NLIDB, come per esempio ATHENA [14], utilizzano ontologie come base della loro conoscenza; questa descriva le entità semantiche del dominio e le relazioni tra loro.

2.3.2 Principali soluzioni

Come analizzato, la letteratura è piena di interfacce in grado di comprendere più o meno bene query SQL espresse in linguaggio naturale, anche molto complesse [36]. Anche in ambito business intelligence sono stati sviluppati si-

[1] `https://nlp.stanford.edu/software/lex-parser.shtml`

stemi in grado di suggerire query o sessioni OLAP con l'obiettivo di migliorare l'esplorazione dei dati sulla base delle precedenti operazioni degli utenti [33], anche in contesti non standard [34]. Dal punto di conversazionale sono stati effettuati degli studi per l'esposizione vocale di risultati di sessioni OLAP e raffinamento delle query [35], non ci sono dei risultati in grado di eseguire delle vere e proprie sessioni di analisi OLAP.

In conclusone, non esiste nessun sistema sviluppato appositamente per comprendere le query GPSJ ed eseguire sessioni di analisi OLAP tramite un approccio conversazionale.

Ora entreremo nel dettaglio di due delle interfacce per database relazionali maggiormente efficienti: NaLIR [15] e ATHENA [14].

NaLIR Ha come focus principale l'iterazione con l'utente in quanto l'idea di F. Li e H. V. Jagadish è che se l'utente esprime una query complessa e riceve come risultato un unico valore (solitamente numerico) non ha modo di verificare se è realmente il risultato che desiderava. Inoltre, in caso di fallimento della query, l'utente dovrebbe riscrivere la stessa richiesta senza sapere cosa il sistema non ha compreso e cosa sì, rischiando di riformulare una frase incomprensibile dal sistema. L'idea è quindi, in ogni situazione di non assoluta certezza, di chiedere all'utente qual'è la scelta giusta, così come chiedere conferma prima di fare qualsiasi assunzione.

Il sistema quindi parte creando il *parse tree* linguistico della frase in input che viene poi trasformato in un *"Query Tree"*, un albero corretto che alla fine può essere tradotto in modo algoritmico in una query SQL. La correttezza dell'albero viene verificata sfruttando una grammatica per i parse tree sintatticamente validi. Come base di conoscenza utilizza sia il database stesso su cui sono memorizzati i dati (per poter accedere ai valori degli attributi), sia un grafo pesato che esplicita la struttura del database, mostrandone gli attributi

e le relazioni. Si cerca di trasformare ogni nodo del *parse tree* in uno nodo del *query tree* a seconda che venga trovato un match con:

- Keyword legata alla struttura della query:
  - Quantificatori: "all" , "any", etc.
  - Operatori: >, <, =, etc.
  - Funzioni di aggregazione: "avg" , "max", etc.
  - Logici: "and", "or", "not", etc.
- Valori memorizzati nel database o valori di nomi e relazioni del database.

Vista l'improbabilità di riuscire nella trasformazione completa subito (a causa di errori nella generazione del query tree o di informazioni implicite nel linguaggio parlato che possono essere assunte conoscendo il dominio), viene poi attuata una fase iterativa in cui si cerca di aggiustare l'albero, spostandone alcune parti. Quando viene trovato l'albero migliore, anche facendo richieste all'utente, vengono poi inseriti i nodi impliciti; chiedendo conferma per ognuno. Al termine si avrà un query tree corretto che può essere tradotto in una query SQL in modo algoritmico, il cui risultato verrà mostrato all'utente.

ATHENA Sviluppato successivamente a NaLIR, può essere visto come un estensione del primo. Innanzi tutto basa la sua conoscenza oltre che sul database, su un'ontologia che descrive le entità semantiche e le relazioni tra esse, dando così una descrizione standard del dominio dell'applicazione. Viene, inoltre, memorizzato il mapping tra gli elementi nell'ontologia e la loro implementazione fisica nel database. Per tutti questi termini, inoltre, vengono memorizzati anche i loro sinonimi, sfruttando WordNet [20]. In questo modo, il sistema ha a disposizione maggiori informazioni semantiche da utilizzare per le disambiguazioni, in modo da ridurre le richieste all'utente. Infatti, seppure

anche in questo caso in caso di incertezza viene chiesto all'utente di scegliere la giusta alternativa, l'idea è quella di cercare di ridurre al minimo le interazioni.

Prima di tutto, sfruttando le *"evidence"*, vengono mappati i token della query in: elementi appartenenti allo schema del database, valori degli attributi del database, numeri e date. Per ogni token possono esserci più mapping possibili per cui si cerca di disambiguare i risultati sfruttando anche lo schema dell'ontologia, creando così un *"interpretation tree"*, che infine viene tradotto in una query. Al termine, insieme al risultato della query generata, viene mostrata anche la query con il secondo punteggio più alto.

Principali differenze

In conclusione, sebbene l'argomento delle interfacce testuali verso i database sia da anni al centro di ricerche, C-BI si differenzia dalle varie soluzioni presenti in letteratura poiché offre:

1. Traduzione efficace e efficiente di frasi in linguaggio naturale in query GPSJ;

2. Formalizzazione e implementazione di sessioni OLAP in stile conversazione umana;

3. Applicazione in tempo reale durante la traduzione di vincoli di cardinalità alla generazione di query.

Capitolo 3

Un sistema per la conversational OLAP

In questo capitolo entreremo del dettaglio del funzionamento di C-BI. Nella Sezione 3.1 verrà mostrata l'architettura funzionale del sistema: risultato della fase di progettazione. Nelle sezioni successive verranno descritti i vari moduli in maniera più dettagliata, soffermandosi maggiormente sulla porzione di sistema già sviluppata.

Prima di proseguire, nella Figura 3.1 è presentato, utilizzando il DFM, il cubo che verrà utilizzato nel resto del capitolo per gli esempi. Attualmente il sistema è in grado di riconoscere solamente frasi espresse in lingua inglese. Per questo motivo i termini e i valori del cubo sono in inglese, così come tutti gli altri termini che verranno utilizzati per la comprensione della frase; anche le frasi di esempio saranno in lingua inglese.

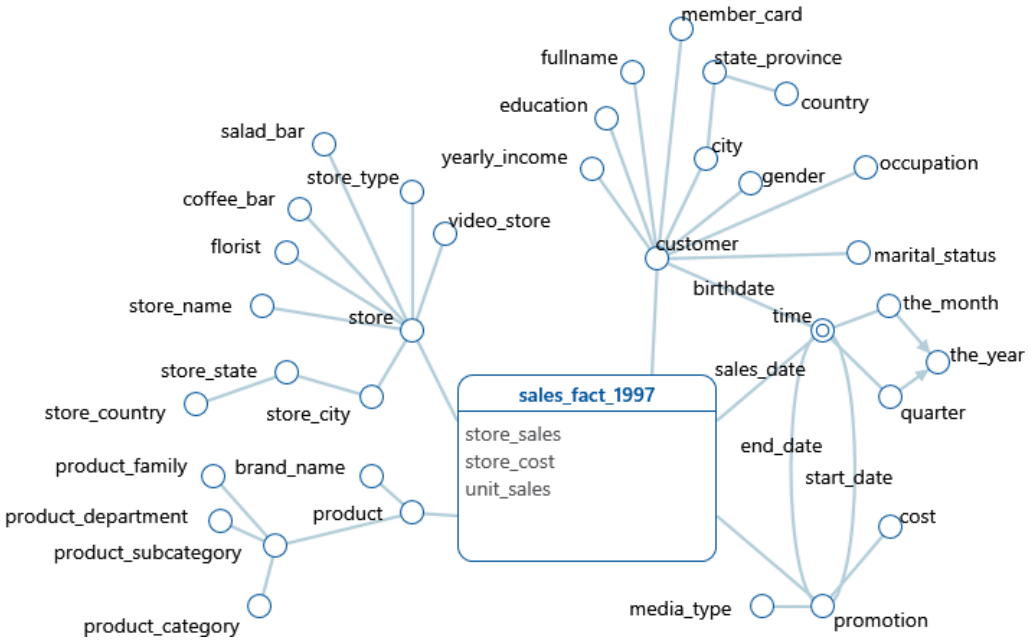


Figura 3.1: DFM di foodmart: il fatto è sales_fact_1997 ed ha 5 dimensioni in cui hanno origine rispettivamente 5 gerarchie: product, store, customer, time e promotion.

3.1 Architettura funzionale

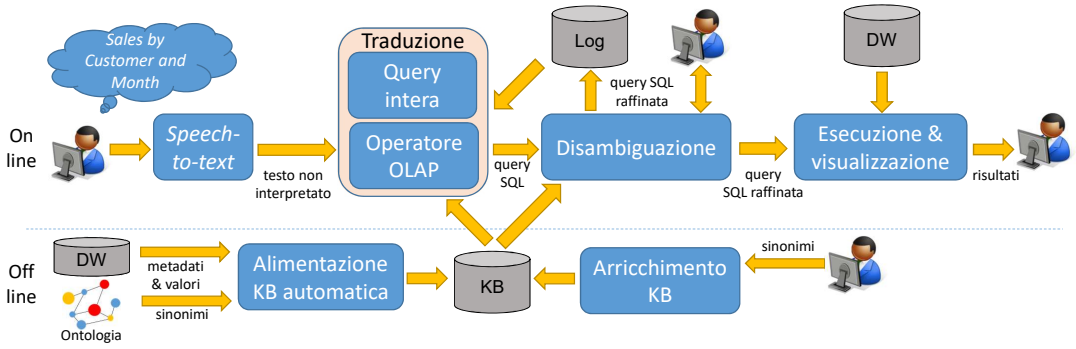


Figura 3.2: Architettura funzionale di C-BI

Come è possibile vedere dalla Figura 3.2 il sistema si compone di due parti: una *off-line* e una *on-line*.

Durante la fase off-line, il modulo Alimentazione KB automatica si occupa di estrarre in modo automatico dal data warehouse tutti i dati (cioè i membri) e metadati (nomi delle misure e degli attributi, strutture gerarchiche, operatori di aggregazione). Queste informazioni vengono poi memorizzate in un nuovo database che chiameremo *Knowledge Base* (KB), che verrà utilizzato dal sistema come base di conoscenza per comprendere la richiesta dell'utente e per creare l'SQL corrispondente. Questa fase viene eseguita ogni volta che viene aggiunto un nuovo data warehouse e ogni volta che vengono aggiornati i valori caricati nei cubi. Per migliorare il supporto del sistema oltre ai termini del DW è possibile utilizzare i sinonimi di questi: maggiore è il numero di sinonimi, più ampia è la porzione di linguaggio compresa dal framework; ad esempio per il concetto di "store" potrebbe essere inserito il sinonimo "shop". I sinonimi possono essere estratti in modo automatico da ontologie di *open data* (letteralmente "dati aperti", ovvero dati liberamente accessibili a tutti) o possono essere inseriti manualmente dall'utente, sfruttando il modulo Arricchimento KB, quando il dominio applicativo presenta un vocabolario non standard. Oltre ai termini del data warehouse, in KB sono memorizzati anche alcuni termini standard di SQL (ad esempio: "sum", "by", etc.) e anche alcuni termini del linguaggio parlato ("tell me", "show me", etc.); questi sono indipendenti dal dominio, devono quindi essere caricati in modo manuale ma è sufficiente farlo una sola volta, indipendentemente dal caricamento dei data warehouse.

Durante la fase *on-line* avviene realmente la traduzione della frase in linguaggio naturale in query SQL e viene di conseguenza eseguita ogni volta che una nuova frase viene introdotta nel sistema. Se la richiesta viene espressa

tramite comando vocale la prima operazione da fare è la sua traduzione in testo scritto, effettuata dal modulo Speech-to-text. Questo aspetto non è stato oggetto della ricerca ma l'idea è quella di affidarsi alle API già presenti, come le Google API [1]. Una volta che la frase è in formato testuale viene data in input al modulo Traduzione. Questo è il componente principale del framework ed è diviso in due sotto-moduli:

1. Query intera: con il compito di riconoscere quando viene inserita una nuova query, da tradurre interamente, rimuovendo ciò che prima era in memoria; solitamente avviene all'avvio di una nuova sessione OLAP.

2. Operatore OLAP: entra in gioco quando l'utente inserisce un operatore OLAP in una sessione di analisi con l'obiettivo di raffinare il risultato già ottenuto precedentemente modificando l'ultima query fatta. Apparentemente comprendere un singolo operatore è più semplice rispetto a comprendere una frase intera, ma richiede di avere in memoria la query precedente e capire quale parte deve essere modificata.

A causa di ambiguità nel linguaggio naturale, errori dell'utente o del sistema alcune parti del testo potrebbero non essere comprese. Il modulo Disambiguazione ha il compito di risolvere queste incomprensioni interagendo con l'utente in linguaggio naturale; le incomprensioni e le possibili soluzioni devono essere quindi espresse in un linguaggio comprensibile dall'utente e le risposte devono poi essere analizzate e tradotte in modo da essere utilizzabili per raffinare l'elaborazione della query effettuata fino a quel momento. Le incertezze possono riguardare:

[1] `https://cloud.google.com/speech-to-text/`

- Incertezza su come una parte del testo deve essere interpretata (ad esempio ci sono più interpretazioni possibili di una parola, o vengono restituite più query possibili dalla fase precedente);
- Inconsistenze nel predicato di selezione (tra attributo e valore o tra operatore utilizzato e tipo di attributo);
- Inconsistenza tra operatore di aggregazione e misura.
- Identificazione di un elemento rilevante nel testo senza riuscire ad assegnarli un ruolo all'interno della query.

Una volta ottenuta un'interpretazione soddisfacente la query viene eseguita sul data warehouse e il risultato viene mostrato all'utente tramite il modulo di Esecuzione & visualizzazione. Questo modulo potrebbe utilizzare uno degli strumenti standard di visualizzazione di query OLAP o potrebbe implementare un approccio *voice-based* [24] per creare una soluzione conversazionale end-to-end.

Entriamo ora nel dettaglio dei moduli principali, dalla discussione sono esclusi il modulo di *speech-2-text* e esecuzione & visualizzazione in quanto sono integrabili al sistema tramite API esterne.

3.2 Base di conoscenza (Knowledge Base - KB)

Nel database KB sono memorizzate, in modo principalmente automatico tutte le informazioni necessarie per supportare la frase di traduzione: ovvero i valori e lo schema stesso dei cubi.

Le informazioni possono essere divise in due classi: entità e struttura.

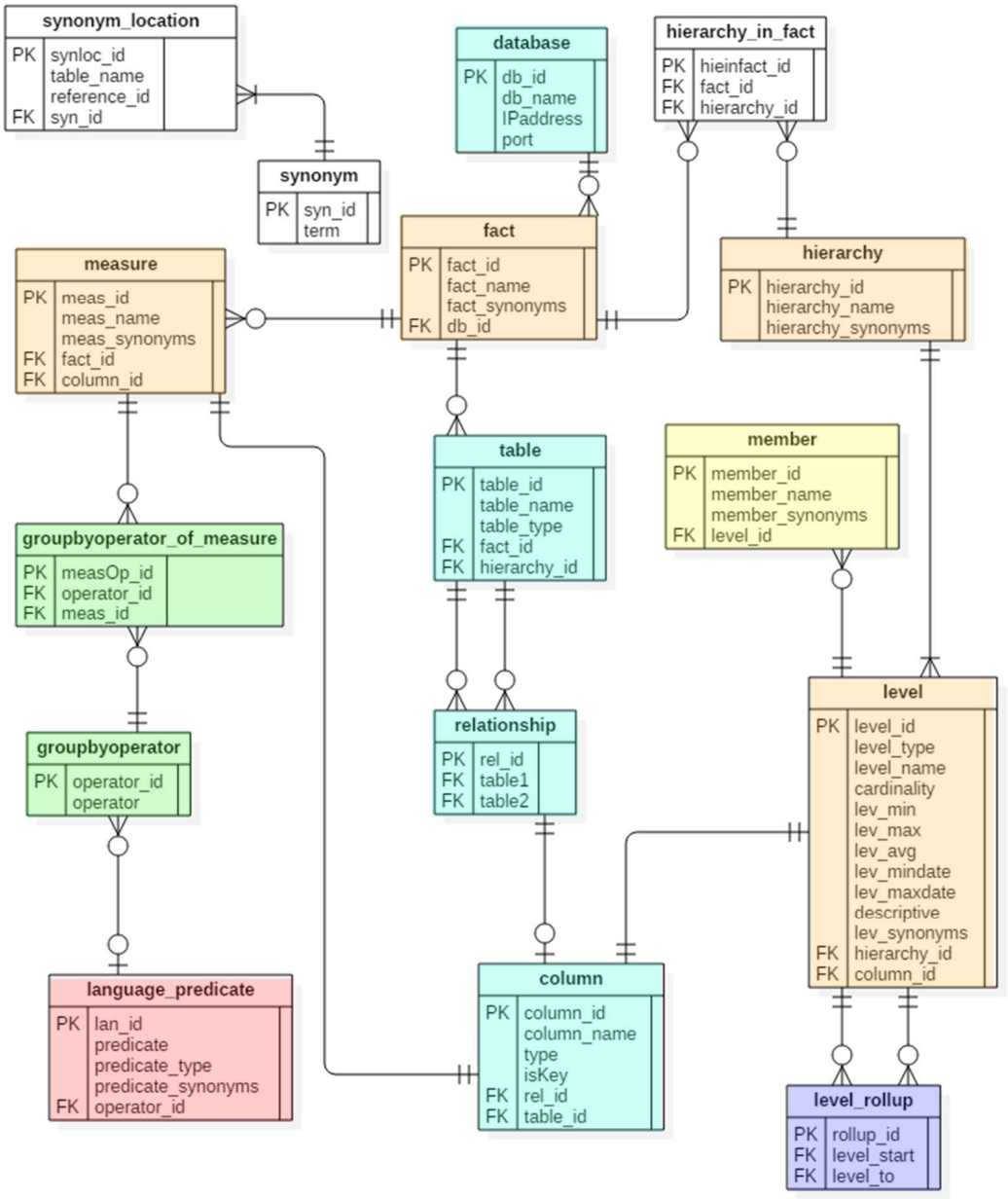


Figura 3.3: Schema E-R database relazionale che implementa *KB*, compatibile QB4OLAP [25].

- *Entità*: sono i valori che vengono identificati nella frase inserita in linguaggio naturale. Si dividono in:
  - **Nomi elementi DW**: ovvero misure, dimensioni, attributi dimensionali e nome del fatto. Le tabelle che memorizzano queste informazioni sono mostrate in Figura 3.3 di colore arancione.
  - **Valori elementi DW**: per ogni attributo categorico vengono memorizzati tutti i suoi possibili valori; nella tabella colorata di giallo.

  Per ognuna di queste entità possono essere memorizzati più sinonimi per poter gestire il gergo parlato e le differenti sfumature del testo.

- *Struttura DW*: consentono di fare i controlli sulla correttezza delle interpretazioni date alla frase (punti 1 e 2) e, alla fine, permette di generare la query in linguaggio SQL (punto 3). Include:
  - **Strutture gerarchiche**: esprime le relazioni di roll-up che possono essere eseguite tra coppie di attributi. La tabella in cui sono esplicitate queste informazioni è LEVEL_ROLLUP (in blu in Figura 3.3).
  - **Operatori di aggregazione**: ("sum", "average", "max", "min" e "count"): utilizzati per aggregare le misure, non tutti sono applicabili a tutte le misure, per cui viene memorizzato anche per ogni misura quali operatori di aggregazione è possibile applicargli ed eventualmente qual'è quello di default. Le informazioni sono memorizzate in GROUPBYOPERATOR e GROUPBYOPERATOR_OF_MEASURE (in verde in Figura 3.3).
  - **Tabelle DB**: memorizza la struttura del database fisico in cui risiede il data warehouse, includendo nomi di tabelle e attributi, chiavi

primarie ed esterne e relazioni tra loro; le tabelle interessate sono quelle di colore azzurro.

Tutte queste informazioni sono riferite direttamente a un data warehouse, ogni volta che si vuole inserire il supporto a eseguire query su un nuovo cubo queste informazioni devono essere aggiunte, il modulo di Alimentazione KB automatica si occupa di caricare in modo automatico queste informazioni, per rendere questa fase il meno impegnativa possibile.

Oltre a queste informazioni, sono anche memorizzati alcuni termini indipendenti dal contesto, che possono essere caricati un'unica volta e sono utilizzabili per ogni data warehouse. Sono:

- **Operatori utilizzabili nelle query**: all'interno delle query possono essere utilizzati diversi operatori: (1) *logici* ("and", "or" e "not"): utilizzati per concatenare i diversi predicati di selezione; (2) *matematici* ($>$, $<$, $=$, $\neq$) utilizzati all'interno dei predicati di selezione.

- **Parole chiave del linguaggio**: vengono infine memorizzate una serie di parole che vengono utilizzate nel linguaggio parlato che possono essere sfruttate per comprendere il ruolo dei componenti successivi nella frase; ad esempio: *"by"*, *"grouped by"* e *"for each"* indicano che l'attributo che li seguirà nella frase andrà inserito nella *group by clause*.

Sono memorizzate in un'unica tabella, visibile nel disegno colorata di rosso, con l'attributo PREDICATE_TYPE per differenziare il tipo di termine.

3.3 Alimentazione KB automatica

Il modulo Alimentazione KB automatica si occupa del caricamento automatico dei dati e dei metadati del cubo su KB. Occorre precisare che questa fase

di caricamento automatico può essere fatta solo su un data warehouse implementato utilizzando l'approccio ROLAP: in cui i dati sono quindi memorizzati in un database relazionale. Per un corretto funzionamento di questo modulo il cubo deve essere implementato nel modo corretto e devono essere dichiarate, per ognuna delle tabelle, le chiavi primarie ed esterne. Questo caricamento è stato reso il più possibile automatico sfruttando JDBC[2], API Java di connessione a DB relazionali. Questo modulo si occupa di leggere i metadati (nomi di tabelle, colonne, etc.) e i dati (tutti i valori delle tabelle) del database in cui è implementato il cubo e memorizzarli nel nuovo database: KB.

Innanzi tutto è necessario conoscere il nome del database in cui risiede il cubo, il suo indirizzo IP e il numero di porta; infine possedere delle credenziali di accesso a quest'ultimo; questi dati vengono anche inseriti nella tabella DATABASE. Conoscendo questi dati è possibile connettersi al database e leggerne dati e metadati. Il nome della *fact table* viene specificato dall'utente e memorizzato sia nella tabella FACT che nella tabella TABLE. Dopodiché partendo da questa tabella vengono lette tutte le sue colonne. Quelle che non sono chiavi esterne sono misure, vengono quindi memorizzate come tali nella tabella MEASURE e in COLUMN. Anche le chiavi vengono memorizzate in COLUMN ma vengono anche utilizzate come collegamento per raggiungere le tabelle a cui si riferiscono. Ognuna di queste è l'origine di una nuova gerarchia, e quindi memorizzata in HIERARCHY, specificando la connessione tra la gerarchia e il fatto nella tabella HIERARCHY_IN_FACT; vengono, poi, aggiunte anch'esse a TABLE e nella tabella RELATIONSHIP vengono memorizzate quali tabelle possono essere raggiunte tramite un "join" e grazie a quali colonne. Tutte le loro colonne che non sono chiavi esterne sono livelli della gerarchia e come tali memorizzati in LEVEL. A questo punto per le colonne testuali, che si riferiscono ad attributi

[2] `https://www.oracle.com/it/database/technologies/appdev/jdbc.html`

categorici, vengono inseriti tutti i possibili valori in MEMBER e ne viene memorizzata la cardinalità. Per i valori numerici, invece, viene calcolata subito la cardinalità. Questo perché se è maggiore di una certa soglia, viene comunque considerato come categorico, ad esempio un codice numerico; se invece è minore della soglia vengono calcolati massimo, minimo e media. Anche per le date vengono calcolati massimo e minimo ma non la media. Per le chiavi primarie di queste tabelle viene memorizzato come sinonimo anche il nome della gerarchia, questo perché quando viene inserito il nome della gerarchia quello che si vuole fare è raggruppare per la sua chiave, ad esempio nella frase *"unit sold by store"*, "store" deve essere trasformato nel'id della *dimension table* corrispondente, per poter generare la corretta clausola di group by. Anche una *dimension table* può avere chiavi esterne, anche in questo caso queste vengono aggiunte a COLUMN e utilizzate per raggiungere la tabella di riferimento. Si prosegue in questo modo iterativo raggiungendo e aggiungendo tutte le tabelle, con l'unica differenza rispetto alla prima iterazione che queste nuove tabelle non danno il via a nuove gerarchie, ma aggiungono semplicemente livelli alle gerarchie esistenti.

Dato un termine, il modulo di alimentazione automatica supporta la possibilità di trovare e memorizzare tutti i suoi sinonimi sfruttando WordNet [20]; dopo una prima analisi è stato deciso di non utilizzare questi sinonimi in modo automatico, ma eventualmente lasciare la possibilità di inserirli in modo manuale.

Come è possibile vedere dall'E-R in Figura 3.3 tutti i sinonimi sono memorizzati in un'unica tabella, ed è presente una seconda tabella, in relazione molti-a-uno con la prima, in cui sono memorizzati i riferimenti agli elementi del DB a cui si riferisce il sinonimo. Nell'ultima fase di caricamento automatico, eseguibile anche singolarmente in caso in cui vengano inseriti dei si-

nonimi manualmente, vengono letti tutti i sinonimi e i termini dalle tabelle per FACT, HIERARCHY, LEVEL, MEASURE e LANGUAGE_PREDICATE e vengono salvati tutti nella tabella SYNONYM mentre i riferimenti alla loro posizione nel db (che servirà poi per risalire al tipo di elemento) vengono memorizzati in SYNONYM_LOCATION.

3.4 Arricchimento KB

Le tabelle in cui sono memorizzati i termini legati alle operazioni OLAP e al linguaggio parlato non possono essere riempite in modo automatico. Queste informazioni sono memorizzate nella tabella LANGUAGE_PREDICATE in cui vengono memorizzate alcune parole chiave del linguaggio parlato che sono rilevanti ai fini di riconoscere porzioni di frase più alcuni operatori logici e matematici o di aggregazione; queste ultime in particolare sono memorizzate anche nella tabella GROUPBYOPERATOR mentre nella tabella GROUPBYOPERATOR_OF_MEASURE viene memorizzata l'applicabilità degli operatori a tutte le misure. Come già detto, infatti, non tutti gli operatori sono applicabili su tutte le misure, per questo viene memorizzata questa informazione che verrà poi utilizzata per controllare la correttezza o meno di una eventuale interpretazione della frase in input. Queste tabelle devono quindi essere popolate in modo manuale. Le prime due, a meno che non ci siano cambiamenti particolari, è sufficiente che vengano riempite solo una volta, anche se vengono inseriti nuovi data warehouse; mentre l'ultima tabella è dipendente dal contesto e deve essere modificata ogni volta che un nuovo data warehouse, e di conseguenza le sue misure, vengono caricati.

L'ultima tabella da riempire in modo manuale è LEVEL_ROLLUP, che contiene in ogni sua istanza una coppia di *id* di livelli che rappresenta che dal primo livello è possibile fare *roll-up* sul secondo. Questa tabella verrà poi

utilizzata nel modulo OLAP operator per verificare la correttezza dell'interpretazione. Anche questa contiene informazioni legate al data warehouse quindi deve essere aggiornata ogni volta che ne viene caricato uno nuovo.

In questa fase possono anche essere inseriti in modo manuale nuovi sinonimi per i termini già memorizzati nel database.

3.5 Traduzione: Query intera

Il modulo di Traduzione è il più complesso in quanto si occupa concretamente della traduzione della frase. Può essere scomposto in due sotto-parti: query intera e operatore OLAP. In questa sezione verrà analizzata nel dettaglio la parte di query intera, la cui architettura dettagliata è mostrata in Figura 3.4. Quello ricevuto in input in questa fase è la frase in in linguaggio naturale in forma scritta.

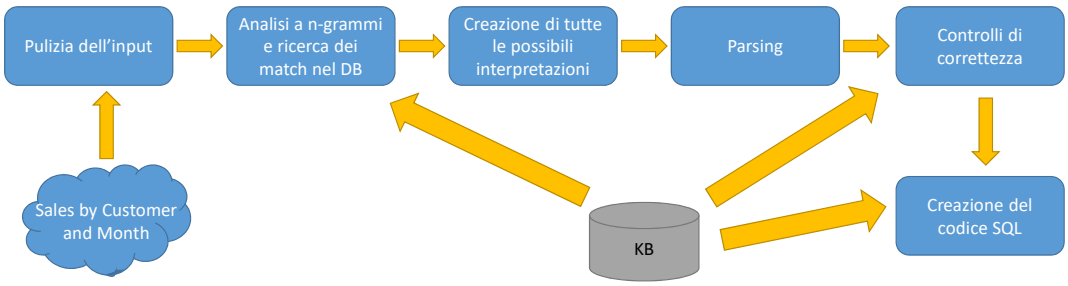
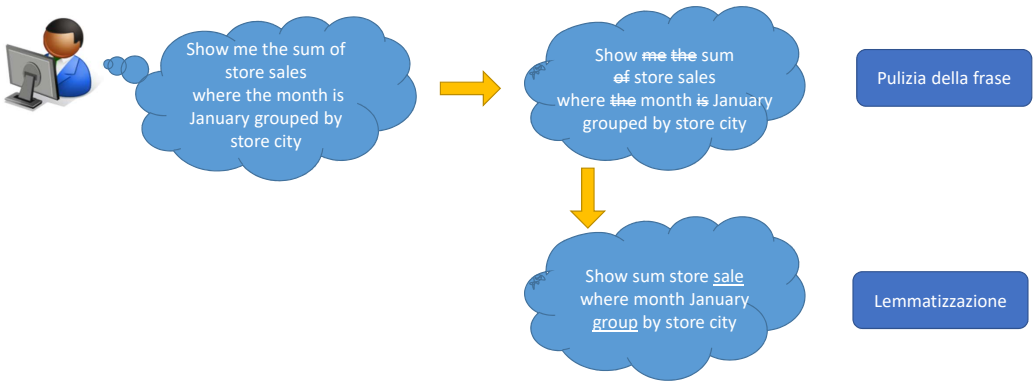


Figura 3.4: Architettura dettagliata del sotto-modulo Query intera all'interno del modulo Traduzione

È la fase on-line che viene quindi eseguita ogni volta che una nuova frase è data in input al sistema, di conseguenza diventa necessario fare anche considerazioni legate all'efficienza.

3.5.1 Pulizia dell'input



La prima operazione eseguita sull'input è la pulizia. Per questa fase è stata utilizzata la libreria `Stanford CoreNLP` [3], una libreria gratuita che implementa in diversi linguaggi di programmazione lo Stanford Parser. Questa mette a disposizione diverse funzionalità, quelle utilizzate all'interno del framework sono:

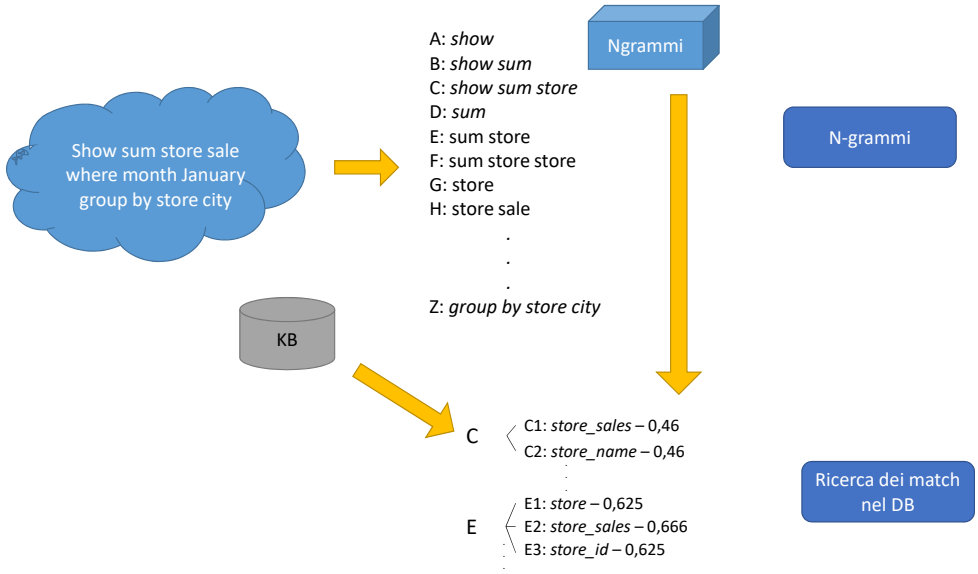
1. `Tokenize`: che si occupa di dividere la frase in token, ovvero singole parole.
2. `Ner`: *Named Entity Recognition*, che ha il compito di etichettare particolari tipi di entità che vengono riconosciute nel testo: nomi di persone, società, date, città, numeri, etc.
3. `Lemma`: che traduce ogni token nel suo lemma.

La frase viene quindi data in input alla libreria e su di essa vengono eseguite le prime due operazioni, memorizzando temporaneamente quali termini sono stati etichettati dal `Ner`. A questo punto tra i token viene verificato se ce n'è qualcuno categorizzabile come stop word, e in questo caso eliminato. Nel sistema è stata utilizzata una lista di stop word trovata online, alla quale sono

[3] https://stanfordnlp.github.io/CoreNLP/

state rimosse le parole in realtà utili nel nostro contesto come “by” o “for”, che solitamente non aggiungono informazioni rilevanti in una frase in linguaggio naturale ma possono essere fondamentali per riconoscere porzioni di query SQL. Infine dalle parole rimaste viene ricavato il lemma, in modo da ridurre le varie forme flesse della stessa parole in un unico termine.

3.5.2 Ricerca dei match nel DB



Una volta ricavati i lemmi dei soli token potenzialmente interessanti vengono ricercati i match tra questi token e gli elementi in KB. Per limitare il numero di chiamate al database i sinonimi vengono letti una sola volta e mantenuti in memoria.

In quanto nel database alcune delle entità sono composte da più di una parola (ad esempio la misura “store sales”) si è deciso di utilizzare la tecnica di analisi del testo che utilizza gli N-grammi; in questo modo verranno ricercati i match non solo tra singole parole ma anche tra sottosequenze di parole

vicine nella frase. In particolare si è deciso di analizzare gli n-grammi da 1 a ngramSize, dove ngramSize è uno dei parametri, con valore di default 3. Si è deciso di fare una distinzione tra i match con i membri (i valori degli attributi categorici) e tutti gli altri valori; è stata presa questa decisione in quanto i membri memorizzati in KB sono molti di più rispetto agli altri valori (caricando i valori di foodmart ci sono 12315 membri contro 62 termini tra fatto, misure, livelli, nomi delle gerarchie e termini del linguaggio) ma in percentuale meno presenti all'interno della frase. Inoltre, visto che uno dei principali obbiettivi di C-BI è proprio astrarre dall'implementazione fisica del database è necessario fornire una certa flessibilità sui nomi di gerarchie, livelli e misure. L'idea è quindi di utilizzare una soglia maggiore per i membri (thrMember) memorizzandone anche un numero inferiore (nSynMemeber), con l'idea che se l'utente vuole specificare un particolare valore di un livello il nome deve essere inserito con un certo livello di correttezza; e utilizzare, invece, una soglia minore per i match dell'altro tipo (thrMetadata) memorizzandone invece un numero maggiore (nSynMeta). Un valore sensato per questi ultimi parametri potrebbe essere nSynMemeber 1 con thrMember 0.9, per avere appunto al massimo uno di questi valori; invece, nSynMeta potrebbe essere 5 con thrMetadata 0.4, molto inferiore quindi.

Per ogni n-gramma del testo vengono ricercati gli elementi simili nel database, analizzandone uno per volta. Per ogni coppia di elementi (l'n-gramma e l'elemento in KB) definiti genericamente T e W lunghi rispettivamente l e m:

1. Se almeno uno dei due è composto da più di una parola viene calcolato il miglior match possibile tra le parole delle due entità per rendere il calcolo della similarità indipendente dall'ordine in cui i token si trovano nella frase e nel database; ad esempio "costs for store" e "store cost" (il cui esempio dettagliato è visibile nella Figura 3.5) devono avere una

similarità molto elevata, cosa che non avrei se confrontassi la prima parola della prima entità con la prima della seconda e così via ("costs" con "store" e "for" con "cost"). Questo problema può essere ricondotto allo *stable marriage problem* [42], il cui obiettivo è quello di trovare il miglior accoppiamento possibile tra n uomini e m donne, considerando le preferenze espresse da ogni persona relativamente ai membri del genere opposto. Il risultato non viene definito stabile se ci sono almeno un uomo e una donna che non sono accoppiati ma ognuno dei due preferisce l'altro al proprio partner assegnato. In questo caso uomini e donne sono tutte le parole che formano le due entità, e la preferenza è data dalla similarità tra le singole parole. Nel dettaglio:

(a) Per ogni possibile coppia di parole t e w appartenenti alle due entità viene calcolata la similarità nel modo seguente:

$$Sim(< t >, < w >) = 1 - \frac{LevDistance(t, w)}{\max(|t|, |w|)}$$

Viene utilizzata la distanza di Levenshtein [26] normalizzata, questa tra due stringhe A e B è il numero minimo di modifiche elementari che consentono di trasformare la A nella B. Per modifica elementare si intende: la cancellazione di un carattere, la sostituzione di un carattere con un altro o l'inserimento di un carattere.

(b) Viene così definita la lista di preferenze tra le parole della seconda entità per ogni parola della prima e viceversa.

(c) Viene eseguito l'algoritmo per la risoluzione dello *stable marriage problem* utilizzando le preferenze precedentemente calcolate.

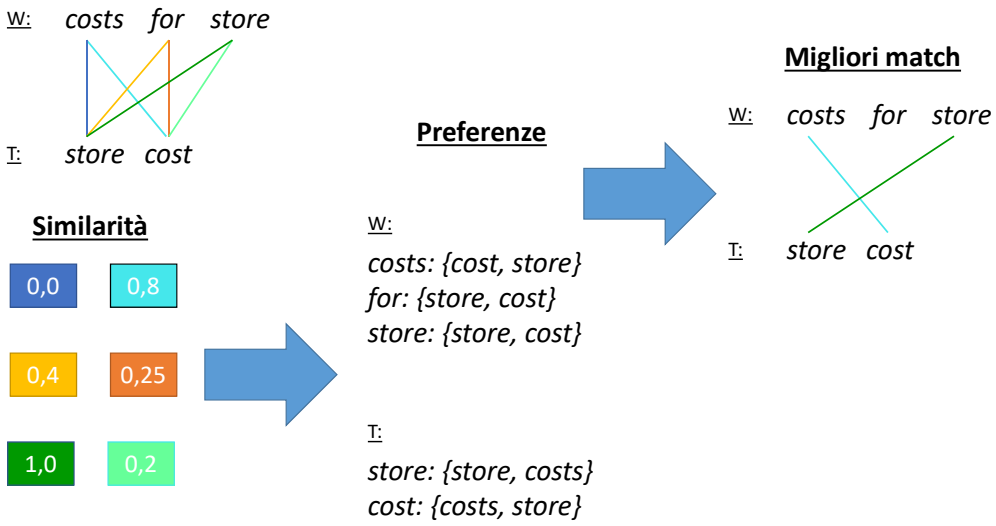


Figura 3.5: Esempio di ricerca del match migliore tra un ipotetico n-gramma "costs for store" e il termine del database "store cost" (misura).

Al termine di questi passaggi si avrà il miglior accoppiamento possibile tra le parole del primo termine e quelle del secondo termine. Se entrambi i termini sono composti da una sola parola questa fase viene saltata e viene calcolata direttamente la similarità tra le due parole.

2. Una volta ottenuto il miglior match possibile tra le parole viene calcolata la similarità complessiva dei due termini T e W lunghi rispettivamente l e m dove $l \leqslant m$, definita come:

$$Sim(< t_1, ..., t_l >, < w_1, ..., w_m >) = \max_{D \in Disp(l,m)} \frac{\sum_{i=1}^{l} sim(t_i, w_{D(i)}) \cdot \max(|t_i|, |w_{D(i)}|)}{\sum_{i=1}^{l} \max(|t_i|, |w_{D(i)}|) + \sum_{i \in \hat{D}} |w_i|}$$

dove $D \in Disp(l, m)$ è una disposizione di l elementi su $\{1, ..., m\}$ (ovvero gli indici delle singole parole di W) che massimizza la sommatoria (come spiegato nel punto precedente); $\hat{D}$ è il sottoinsieme dei valori in $\{1, ..., m\}$

non presente in D. Nell'esempio in Figura 3.5 sarebbe $D = \{3, 1\}$ mentre $\hat{D} = \{2\}$ poiché il primo elemento di T va associato al terzo elemento di W, e il secondo di T al primo di W. La similarità tra le parole viene quindi pesata in base alla loro lunghezza (data da $\max(|t_i|, |w_{D(i)}|)$, inoltre viene penalizzata la similarità tra sequenze di lunghezza differente che implicano che alcune parole non trovino nessun match ($\sum_{i \in \hat{D}} |w_i|$). Nell'esempio in Figura 3.5 si avrebbe:

$$Sim(< t_1, t_2 >, < w_1, w_2, w_3 >) =$$

$$\frac{(sim(t_1, w_3) \cdot \max(|t_1|, |w_3|)) + (sim(t_2, w_1) \cdot \max(|t_2|, |w_1|))}{\max(|t_1|, |w_3|) + \max(|t_2|, |w_1|) + |w_2|} =$$

$$\frac{sim(store, store)|store| + sim(cost, costs)|costs|}{|store| + |costs| + |for|} = \frac{1.0 \cdot 5 + 0.8 \cdot 5}{5 + 5 + 3} \simeq 0.7$$

Una volta calcolati tutti questi valori vengono memorizzati per ogni n-gramma i primi nSynMember match con dei membri con la similarità più alta, solo se maggiore di thrMember, e lo stesso vale per i primi nSynMeta match con i metadati, con una similarità maggiore di thrMetadata.

Alla fine di questa fase non esisterà più il concetto di frase ma semplicemente verranno prese in considerazione le entità di KB per cui è stato trovato almeno un match con un n-gramma. Ci sono però alcune informazioni che potrebbero non essere presenti nel DB ma essere comunque utili nella query: ovvero le date e i numeri, in quanto potrebbero essere inseriti come valori nei predicati di selezione per i livelli numerici anche se il valore stesso non è presente nel DB. Ad esempio la frase *"Get the sum of store sales only for product sold with a promotion costing more than 35 euro"* è corretta anche se nessuna promozione ha esattamente questo costo per cui questo valore non è presente nel

database. Per questo motivo anche questo tipo di informazioni (riconosciute e etichettate dalla libreria citata nella sezione precedente) vengono memorizzate come se fossero un match.

Per ogni n-gramma si avrà in output da questa fase una lista con al massimo nSynMeta + nSynMember entità di KB. Ognuna di queste entità viene memorizzata insieme alla similarità (per non doverla ricalcolare ogni volta) e viene mantenuto anche il collegamento con l'n-gramma, in modo da poter risalire alla posizione delle parole all'interno della frase (startPos, endPos).

Una volta trovati tutti match si passa alla prima fase di pulizia dei risultati. Vengono prese, partendo da quelle collegate a un numero maggiore di token, tutte le entità con una similarità maggiore di una certa soglia, che definiamo ngramSymThr; dopodiché vengono rimosse dalla lista di entità tutte quelle contenute nelle prime. Un'entità $E1$ è contenuta in un'altra $E2$ se startPos di $E1$ è maggiore o uguale a startPos di $E2$ e se endPos di $E1$ è minore o uguale a endPos di $E2$. Se usassi come soglia 0.6, ad esempio, dal risultato in Figura 3.6 l'entità con più token maggiore della soglia sarebbe K, di conseguenza le entità B, C, D, G e H verrebbero eliminate; ma non verrebbero eliminate F e I anche se contengono alcuni dei token appartenenti a K. Il valore di questa soglia deve essere molto alto per non correre il rischio di eliminare informazioni utili, il valore di default è, infatti, 0.9.

Esempio 1. *Nella frase "Show me the sum of store sales for the month of January grouped by store city" utilizzando gli n-grammi fino a 4, come soglie 0.8 e 0.4 e un massimo di 3 match sia per i membri che per metadati vengono trovati inizialmente 89 entità. Aggiungendo questa limitazione, con una soglia di 0.9, il numero di entità viene ridotto a 68.*

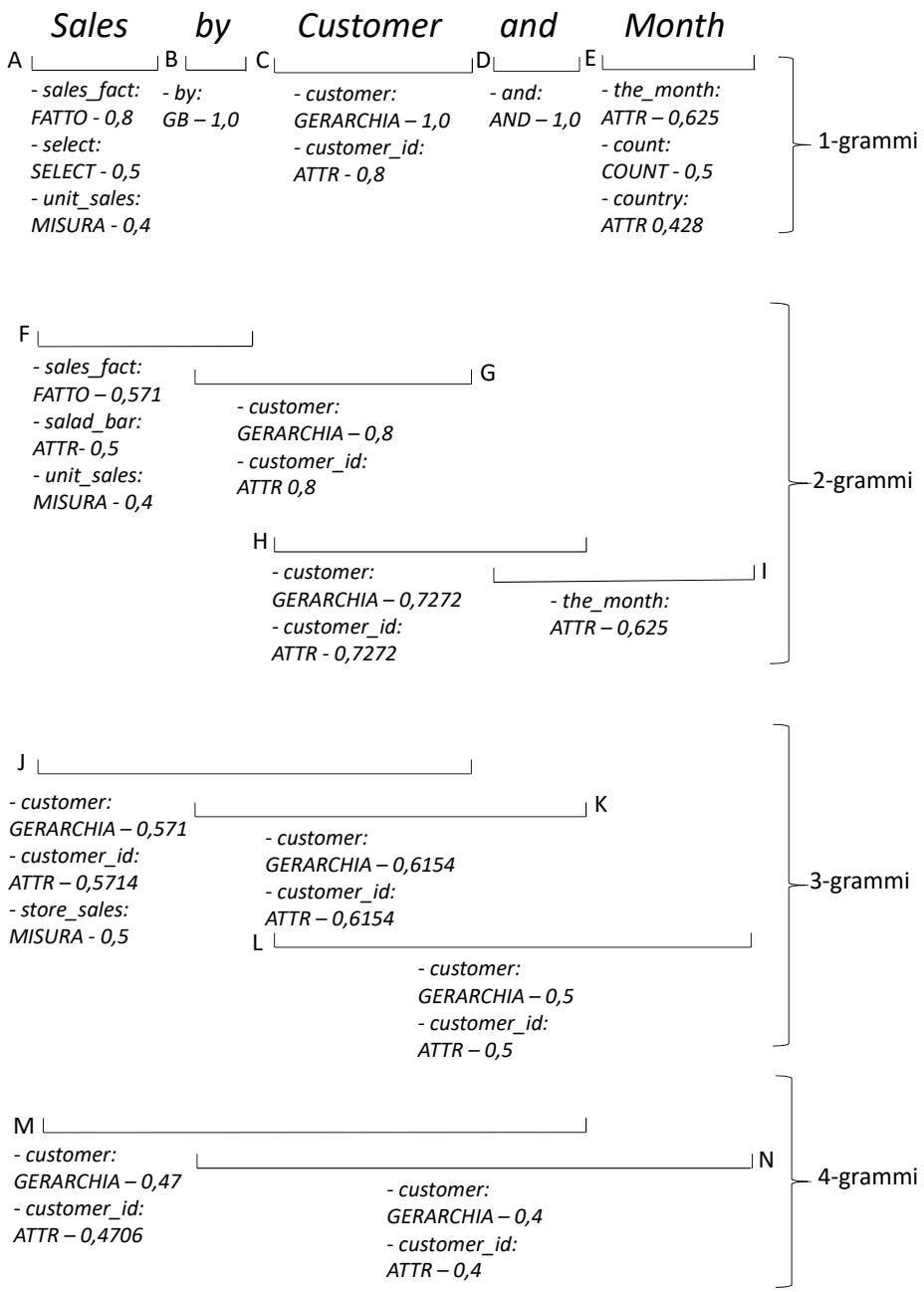
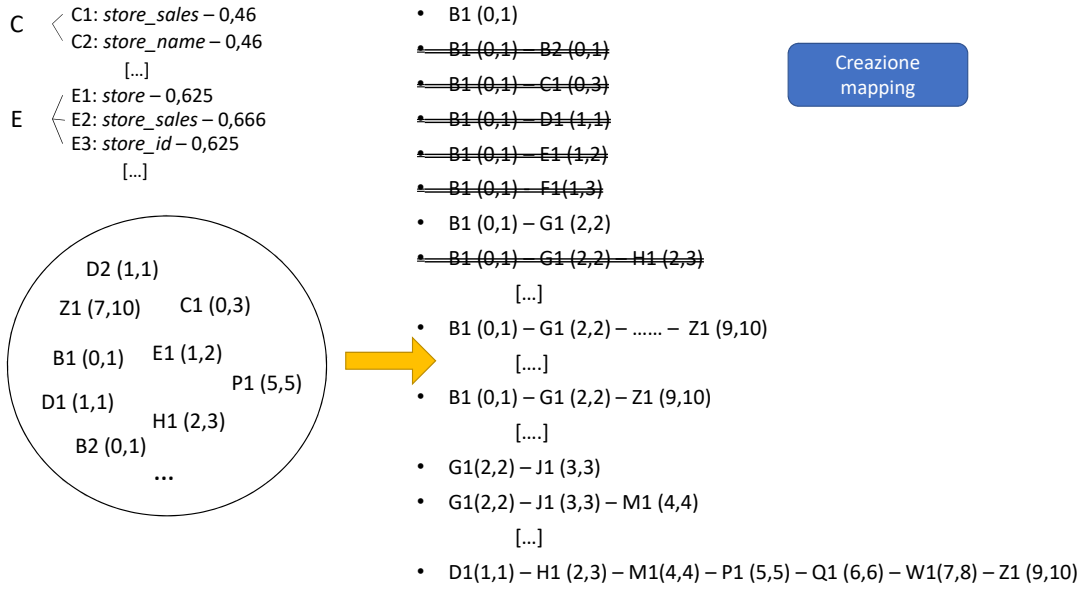


Figura 3.6: Risultato fase di ricerca match per la frase *"Sales by Customer and Month"* con ngramSize $= 4$, thrMember $= 0.8$, thrMetadata $= 0.4$, nSynMemeber $= 3$ e nSynMeta $= 3$.

3.5.3 Creazione dei mapping



Una volta trovate singolarmente tutte le entità presenti nel testo bisogna "ricomporre la frase" unendo i vari componenti. Il risultato di questa fase restituisce i *"mapping"*: una lista di entità del DB con associate le varie informazioni citate prima (n-grammi da cui sono state generate e similarità con essi); le entità all'interno della lista sono ordinate proprio in base alla posizione nella frase dell'n-gramma a loro associato.

Per farlo, dato l'insieme di tutte le entità trovate bisogna calcolarne il *power set* (ovvero creare tutti i suoi sottoinsiemi possibili) con un vincolo: non possono esserci 2 entità i cui n-grammi associati si sovrappongono, ovvero si riferiscono ad uno stesso token della frase. Il power set di un insieme composto da n elementi ha 2^n elementi, eliminando tutti quelli che non rispettano questo vincolo la cardinalità si riduce in maniera notevole. Nella situazione descritta in Esempio 1, con 68 entità il power set conterrebbe quasi 1 triliardo (10^{21}) di

mapping, ma mantenendo solo quelli che rispettano il vincolo ne rimangono 260 mila.

Avere un numero di mapping molto elevato aumenta la robustezza del sistema ma incrementa i costi computazionali. Dal punto di vista semantico la migliore interpretazione è quella che consente a tutte le entità del testo di essere incluse nella query, e questo significa che tutte le entità sono correttamente mappate e analizzate attraverso la grammatica. Alla fine di questa fase non è possibile prevedere se il mapping sarà completamente corretto dal punto di vista della grammatica ma è evidente che maggiore è la cardinalità del mapping, ovvero maggiore è il numero di entità del testo inserite al suo interno, maggiore è la sua probabilità di determinare un'interpretazione ottimale. È possibile inserire quindi due nuovi vincoli che limitano il numero di mapping ma mantengono quelli con una maggiore probabilità di essere corretti:

– Distanza massima tra due n-grammi: la creazione del power set non considera la lontananza degli n-grammi nella frase; ad esempio, uno dei mapping creati conterrà l'entità trovata dalla prima parola della frase e l'entità trovata dal'ultima, nell'esempio in Figura 3.6 il mapping <A,E> sarà sicuramente generato. Questa funzionalità consente di dare una maggiore flessibilità al sistema in quanto potrebbero esserci alcuni termini che non sono rilevanti nella composizione della query ma riescono comunque ad avere un match con qualche elemento del database maggiore della soglia. L'idea è quella che possono esserci delle parole ignorate dal sistema, ma tra due entità consecutive nel mapping non possono esserci più di thrNgramDist parole della frase originale non prese in considerazione; in quanto nella nostra interpretazione la vicinanza viene considerata come relazione tra le entità e in questa situazione verrebbe considerata una relazione che nella frase iniziale non esiste.

– Percentuale di frase considerata: come generalizzazione del punto precedente il mapping deve contenere almeno il thrCoverage % delle parole presenti nella frase iniziale (escluse le stop word). Anche questo vincolo è stato inserito in quanto un interpretazione che ignora troppi termini sarà anche questa un'interpretazione sbagliata.

Aggiungendo questi due vincoli nella frase in Esempio 1 con thrNgramDist 4 e thrCoverage 0.4 si escludono altri 10000 mapping, alzando thrCoverage a 0.6 il numero di mapping rimanenti cala di altri 30000.

Anche in questa fase, una volta generati tutti i possibili mapping, è possibile eliminarne alcuni. Dalla fase di ricerca dei match potrebbe accadere che due n-grammi diversi ma parzialmente sovrapposti trovano un match sufficientemente alto con una stessa entità; ciò porterà ad avere dei mapping che, valutando solo l'entità e non la posizione nella frase, sono in realtà uguali. Facendo riferimento all'esempio in Figura 3.6 i mapping contenente <C,D,E>, <G,D,E> e <I,D,E> porteranno tutti allo stesso mapping contenente: $< Attr, \text{“}and\text{”}, Attr >$. Siccome ciò che verrà utilizzato nelle fasi successive è l'entità e la similarità con l'n-gramma collegato, ciò che viene fatto è manetenere solo il mapping la cui somma delle similarità ha un punteggio maggiore.

Algoritmo 1 Creazione dei mapping

Require: $\mathcal{DB}$: insieme dei match

1: $\mathcal{DB} \leftarrow sort(\mathcal{DB})$ ▷ ordino i match in base alla posizione
2: **while** $\mathcal{DB} \neq \varnothing$ **do**
3: $m \leftarrow head(\mathcal{DB})$
4: $\mathcal{DB} \leftarrow \mathcal{DB} \setminus \{m\}$
5: $CalcolaValidita(\{m\}, \mathcal{DB})$

Nell'Algoritmo 2 è spiegata la procedura ricorsiva con cui i mapping vengono creati, mentre nel 1 come viene avviata la ricorsione.

Algoritmo 2 Funzione ricorsiva

Require: $thrNgramDist$: distanza massima tra due n-grammi vicini nel match, $\mathcal{M}$: insieme dei mapping validi, $thrCoverage$: percentuale minima di token in un mapping, $lenght$: lunghezza della frase

1: **function** CalcolaValidita(Mapp, Missed) ▷ Mapp - il mapping generato fin'ora (lista di match), Missed - la lista di match non ancora verificati.
2: $soglia \leftarrow lenght * thrCoverage$
3: $last \leftarrow tail(Mapp)$ ▷ ultimo match in mapping
4: **while** $Missed \neq \varnothing$ **do**
5: $m \leftarrow head(Missed)$
6: $dist \leftarrow startPos(m) - endPos(last)$ ▷ startPos(m) $\geqslant$ endPos(last)
7: **if** $dist > 0$ **then** ▷ i due non si sovrappongono
8: **if** $dist < thrNgramDist$ **then**
9: $perc \leftarrow ntoken(Mapp) + ntoken(m)$
10: **if** $perc > soglia$ **then** ▷ il mapping è corretto
11: $Mapp \leftarrow Mapp \cup \{m\}$
12: $\mathcal{M} \leftarrow \mathcal{M} \cup \{Mapp\}$
13: **if** $perc + lenght - endPos(m) \geqslant soglia$ **then**
14: $Missed \leftarrow Missed \setminus \{m\}$
15: $CalcolaValidita(Mapp, Missed)$

Nel dettaglio: viene presa ogni entità singolarmente, dopodiché viene avviata una fase iterativa in cui dato un mapping (all'inizio composto da un singolo elemento) e la lista di tutte le entità ancora da provare si prova ad aggiungere ognuno degli elementi mancanti. Per poter inserire le entità in ordine

all'interno del mapping, in modo da rendere i controlli più veloci, la lista di entità ancora da controllare deve essere ordinata in base alla posizione degli n-grammi associati. Se la distanza tra l'n-gramma del nuovo elemento e l'ultimo già nel mapping è maggiore di 0 (significa che i due non sono sovrapposti) e minore di thrNgramDist si passa al secondo controllo in cui si verifica la percentuale di frase coperta, se i controlli sono positivi il mapping viene aggiunto alla lista dei mapping corretti e si prosegue con la fase iterativa; se il secondo controllo da esito negativo ma c'è la possibilità che aggiungendo delle entità al mapping si raggiunga la soglia richiesta (numero di token attualmente contenuti + lunghezza totale della frase - token della frase i cui match che devono ancora essere analizzati) si prosegue con l'iterazione, altrimenti l'iterazione su quel mapping termina.

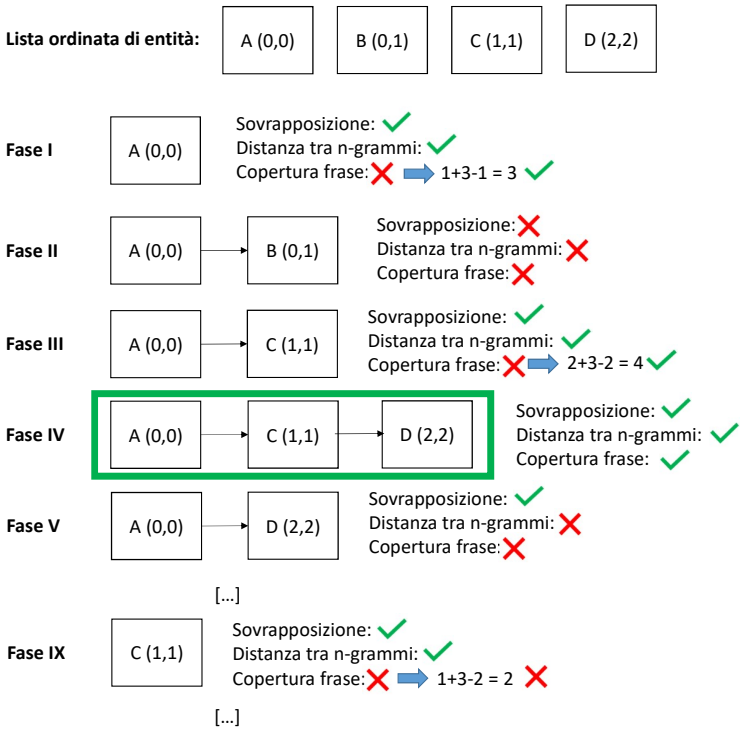
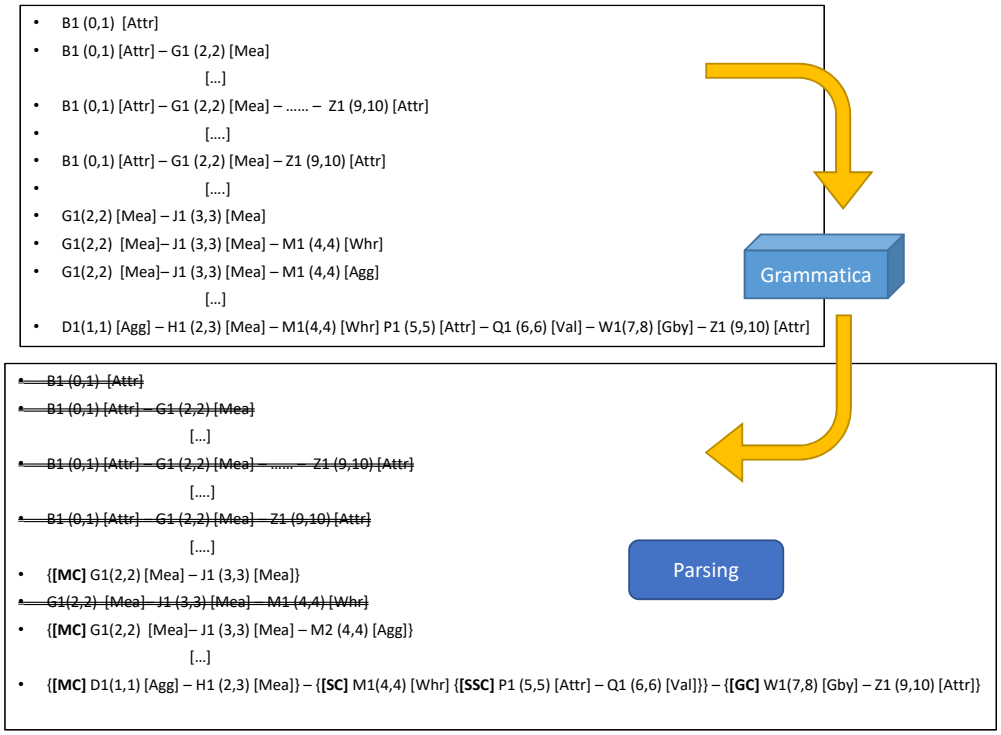


Figura 3.7: Esempio di creazione dei mapping con thrNgramDist 1 e thrCoverage = 100% L'unico valido generato al termine è quello con il bordo verde.

3.5.4 Parsing

L'obiettivo di questa fase è, dati tutti i mapping generati, cercare quali di questi contengono le corrette strutture sintattiche (le clausole) per costruire una query: le misure (MC), il group by set (GBC) e i predicati di selezione (SC).



Questa operazione può essere effettuata sfruttando il concetto di grammatica e parsing, generando per ogni mapping un parse tree. La grammatica per le query GPSJ è mostrata nella Figura 3.8 ed è di tipo LL(1) [37]. Di conseguenza, si tratta di una grammatica non ambigua (per ogni mapping in input può essere generato un solo parse tree) e può essere analizzata con un parser LL(1) con una complessità lineare [37], quest'ultimo è automaticamente generabile partendo dalla grammatica stessa. In realtà la grammatica mostrata in Figura 3.8 non soddisfa i vincoli delle grammatiche LL(1) per ragioni di

leggibilità. È possibile, comunque, in modo algoritmico trasformare le regole nella loro versione LL(1), ma come risultato si avranno regole molto più difficili da leggere e comprendere. L'output di un parser data una frase input è un parse tree che contiene tutte le sue entità se la frase è interamente corretta, se così non è viene prodotto un parse tree contenente solo la porzione di frase compresa. È un requisito molto importante per il nostro sistema, ciò che vogliamo in output, in caso di una frase non del tutto comprensibile, è qualcosa che mostri all'utente quale porzione è stata compresa (e come) e cosa invece no, e non solo una risposta negativa.

$$
\begin{aligned}
\langle \text{GPSJ} \rangle ::= {} & \langle \text{MC} \rangle \langle \text{GC} \rangle \langle \text{SC} \rangle \mid \langle \text{MC} \rangle \langle \text{SC} \rangle \langle \text{GC} \rangle \mid \langle \text{SC} \rangle \langle \text{GC} \rangle \langle \text{MC} \rangle \\
& \mid \langle \text{SC} \rangle \langle \text{MC} \rangle \langle \text{GC} \rangle \mid \langle \text{GC} \rangle \langle \text{SC} \rangle \langle \text{MC} \rangle \mid \langle \text{GC} \rangle \langle \text{MC} \rangle \langle \text{SC} \rangle \\
& \mid \langle \text{MC} \rangle \langle \text{SC} \rangle \mid \langle \text{MC} \rangle \langle \text{GC} \rangle \mid \langle \text{SC} \rangle \langle \text{MC} \rangle \mid \langle \text{GC} \rangle \langle \text{MC} \rangle \mid \langle \text{MC} \rangle \\
\langle \text{MC} \rangle ::= {} & (\langle \text{Agg} \rangle \langle \text{Mea} \rangle \mid \langle \text{Mea} \rangle \langle \text{Agg} \rangle \mid \langle \text{Mea} \rangle) + \\
\langle \text{GC} \rangle ::= {} & \langle \text{Gby} \rangle \langle \text{Attr} \rangle + \\
\langle \text{SC} \rangle ::= {} & \langle \text{Whr} \rangle \langle \text{SCO} \rangle \\
\langle \text{SCO} \rangle ::= {} & \langle \text{SCA} \rangle \text{“or”} \langle \text{SCO} \rangle \mid \langle \text{SCA} \rangle \\
\langle \text{SCA} \rangle ::= {} & \langle \text{SCN} \rangle \text{“and”} \langle \text{SCA} \rangle \mid \langle \text{SCN} \rangle \\
\langle \text{SCN} \rangle ::= {} & \text{“}\textit{not}\text{”} \langle \text{SSC} \rangle \mid \langle \text{SSC} \rangle \\
\langle \text{SSC} \rangle ::= {} & \langle \text{Attr} \rangle \langle \text{Cop} \rangle \langle \text{Val} \rangle \mid \langle \text{Attr} \rangle \langle \text{Val} \rangle \mid \langle \text{Val} \rangle \langle \text{Cop} \rangle \langle \text{Attr} \rangle \\
& \mid \langle \text{Val} \rangle \langle \text{Attr} \rangle \mid \langle \text{Val} \rangle
\end{aligned}
$$

Figura 3.8: Grammatica in *Backus-Naur form* [19] delle query GPSJ.

Per effettuare questa fase ed eseguire il parsing dei vari mapping, prima di tutto le entità che li compongono devono essere suddivise nelle categorie

sintattiche elencate nella Tabella 3.1.

Cat. sintattica	**Entità di riferimento**	**Esempi di sinonimi**
Int	select	return, show
Whr	where	such that
Gby	group by	by
Cop	$=, <>, >, <, \geq, \leq$	equal to, greater than
Agg	sum, count, avg	total, amount, medium
Mea	*Misure e fatti*	dipendente dal dominio
Att	*Attributi*	dipendente dal dominio
Val	*Valori attr. categorici*	
	Date e numeri	dipendente dal dominio

Tabella 3.1: Categorie sintattiche.

Come già detto il numero di entità contenute in un mapping è ciò che utilizziamo, in questa fase, come indicatore della bontà del mapping. Per questo motivo, i mapping vengono ordinati in base a questo valore. A questo punto è stato implementato un parser che provando ad applicare iterativamente le regole della grammatica al mapping, genera (se esiste) il parse tree del mapping, che può contenere alcune o tutte le sue entità. Come in tutte le grammatiche esiste un concetto di "ordinamento" tra le regole, e questo fa sì che questa fase generi al più un parse tree dato un mapping poiché quando una regola è applicata a un insieme di entità non si prova neanche ad applicargli quelle successive. L'ordinamento tra regole della stessa derivazione è stato fatto in base alla lunghezza della regola: ogni volta che si hanno due regole $A ::= X$ e $A ::= Y$ in cui X e Y sono due sottoinsiemi delle possibili categorie sintattiche e $Y \subset X$ se viene applicata la prima regola sicuramente verrà

applicata anche la seconda, ma questo genererebbe due parse tree in cui il secondo sicuramente ha un punteggio inferiore al primo. Ad esempio se è possibile applicare la regola ⟨MC⟩ ::= ⟨Mea⟩⟨Agg⟩ sarà sicuramente possibile applicare anche ⟨MC⟩ ::= ⟨Mea⟩ ma la prima è preferibile. È stato fatto poi un ordinamento tra gli operatori booleani nelle ⟨SC⟩: "or" ha la precedenza, seguito poi da "and" e infine "not".

Verranno applicate quindi tutte le possibili regole al mapping finché non si riesce a chiudere il parse tree, ovevero quando si riesce ad applicare una regola per ⟨GPSJ⟩, oppure finché non si riescono ad applicare più regole; vengono però mantenuti solo i parse tree chiusi, gli altri non vengono proprio memorizzati, considerando i mapping sbagliati.

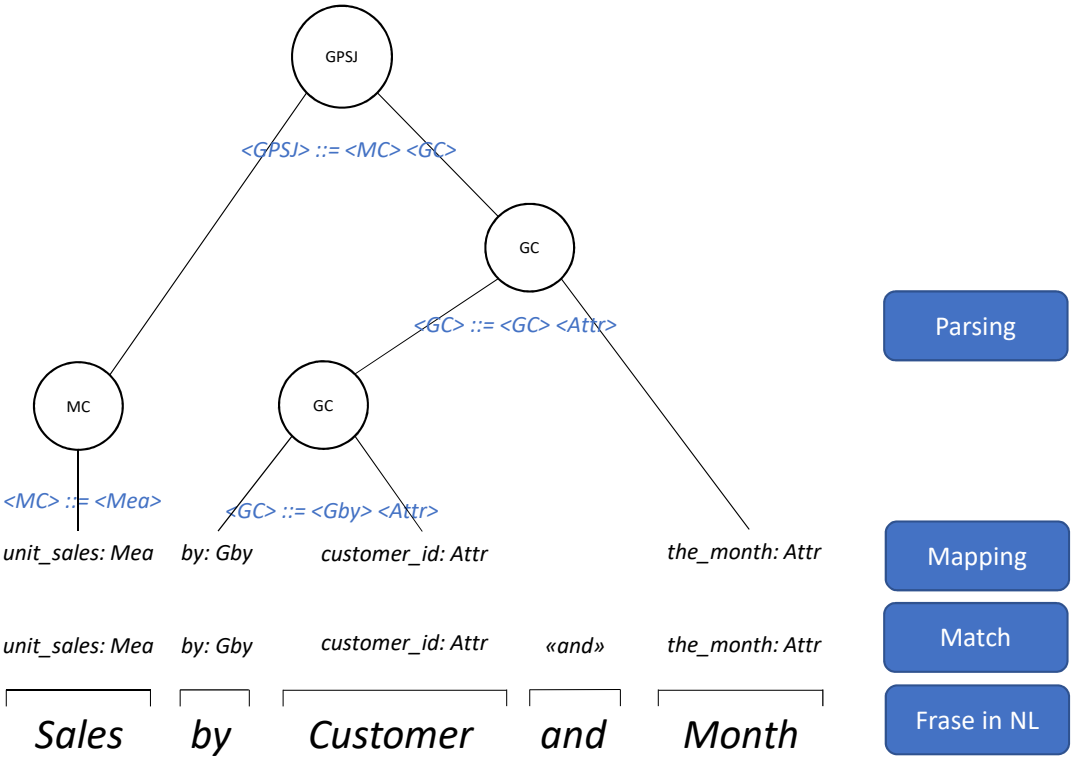


Figura 3.9: Esempio grafico di analisi completa della frase in Figura 3.6 (nella fase di ricerca dei match e mapping sono stati solo i match e i mapping corretti).

In Algoritmo 3 è mostrato il funzionamento del parser dato un mapping. È una versione semplificata di un parser: crea uno stack di interpretazioni (in cui verranno inserire tutte le fasi di elaborazione parziale del mapping), ogni volta (finché ce ne sono) viene preso il primo e il parser prova ad applicargli tutte le regole. Le regole sono di tipo $el ::= X$ dove X è un sottoinsieme di n elementi di tutte le categorie sintattiche. Per ogni regola (analizzate in base all'ordine in cui sono state definite) vengono prese tutte le sottosequenze s di entità del mapping lunghe n, e si verifica se è applicabile; ovvero se le entità in s sono della stessa categoria indicata nella parte destra della regola. Se una regola r è applicabile viene generata una nuova interpretazione uguale a quella analizzata ma rimuovendo tutti gli elementi in s e sostituendoli con l'elemento ritornato dalla regola; dopodiché questa nuova interpretazione viene messa in fondo allo stack. Quando non è più possibile applicare nessuna regola a un mapping, questo viene aggiunto alla lista finale, e si prosegue analizzando un nuovo elemento dello stack. Evidentemente questo approccio non genera un'unica interpretazione; al termine di questa fase occorre quindi fare una selezione dei risultati, in modo che venga memorizzato al più un parse tree per ogni mapping. Per fare questa operazione vengono innanzi tutto selezionati i soli parse tree chiusi, anche se questi contengono solo una parte delle entità del mapping; tra questi viene selezionato quello con il punteggio maggiore. A questo punto occorre definire come viene calcolato il punteggio di una parse tree. Abbiamo deciso di utilizzare non solo il numero di entità contenute nel parse tree, ovvero il numero di foglie, ma questo valore pesato per la similarità media dei match contenuti nell'albero; in modo da avere una compromesso tra il numero di entità utilizzate ma anche l'accuratezza con cui queste entità sono riconosciute. Calcolare questo valore equivale a sommare la similarità delle sue entità.

Algoritmo 3 Parser

Require: *mapp*: mapping (lista di entità), $\mathcal{R}$: lista di regole

1: $\mathcal{S} \leftarrow \{mapp\}$ ▷ creo uno stack che contiene solo il mapping
2: $\mathcal{T} \leftarrow \varnothing$ ▷ stack di traduzioni
3: **while** $\mathcal{S} \neq \varnothing$ **do** ▷ finché lo spazio di ricerca non è vuoto
4: $c \leftarrow false$
5: $s \leftarrow \mathcal{S}.pop$ ▷ prendo il primo mapping dallo stack e lo rimuovo (ogni mapping è una lista di entità)
6: **for all** r in $\mathcal{R}$ **do**
7: **for** $i = 0$ to $|s|$ **do**
8: $size \leftarrow nelements(r)$ ▷ numero di elementi nella parte destra della regola
9: $p \leftarrow sublist(s, i, size)$ ▷ prendo $size$ elementi di s partendo dalla posizione i
10: **if** $match(r, p)$ **then** ▷ è possibile applicare la regola alla porzione di frase
11: $tmp \leftarrow s$ ▷ creo una copia di s
12: $tmp \leftarrow tmp \setminus p$ ▷ rimuovo dal mapping le entità a cui ho applicato la regola
13: $tmp \leftarrow tmp \cup returned(r)$ ▷ al nuovo mapping aggiungo la derivazione della regola applicata
14: $\mathcal{S} \leftarrow \mathcal{S}.push(tmp)$ ▷ aggiungo il nuovo elemento allo stack
15: $c \leftarrow true$
16: $c \leftarrow c + s - 1$
17: **if** c is $true$ **then**
18: $\mathcal{T} \leftarrow \mathcal{T} \cup \{s\}$ ▷ non posso più applicare regole
return $\mathcal{T}$

È possibile ottimizzare il numero di parse tree generati, utilizzando il concetto di punteggio definito sopra. Definiamo k come il numero di parse tree che si vogliono memorizzare e analizziamo i mapping in ordine decrescente di numero di entità al loro interno. Quando da un mapping viene generato un parse tree viene aggiunto alla lista dei parse tree finali, se questa contiene più di k risultati viene eliminato quello con un punteggio minore PT_M. A questo punto, ogni mapping per cui la somma delle sue entità è minore di PT_M non verrà neppure analizzato, in quanto anche se venisse costruito un parse tree contenente tutte le sue entità il punteggio sarebbe minore. Al termine, si avranno quindi i k parse tree con un punteggio migliore.

La porzione di codice attualmente sviluppata termina qui, come output attualmente vengono forniti i k mapping con il punteggio appena definito maggiore. Non è detto che il parse tree contenga tutte le entità del mapping, in quel caso viene restituito sia il parse tree generato che l'insieme di entità che non ne fanno parte.

3.5.5 Controlli di correttezza

La fase di parsing verifica l'appartenenza del mapping alla grammatica delle GPSJ, ma questo non garantisce l'eseguibilità poiché possono esserci: elementi impliciti, errori e ambiguità.

Gli elementi impliciti sono dati da porzioni di query che possono essere considerate scontate, di conseguenza, nel linguaggio parlato queste informazioni non vengono specificate, in alcuni casi il sistema è in grado di riconoscerle e inserire l'informazione che manca, sfruttando le informazioni memorizzate in KB. In questo caso l'utente non deve eseguire nessuna azione ma i nodi impliciti vengono inseriti nel parse tree ottenuto dalla fase precedente. È possibile dover aggiungere:

- Operatore di aggregazione implicito: se in una ⟨MC⟩ viene riconosciuta solo una misura ma questa ha un solo operatore applicabile e/o è stato definito un operatore di default per la misura (informazioni recuperabili in KB), questo è applicabile in modo implicito con certezza. Il nuovo nodo viene aggiunto all'albero ed etichettato come nodo implicito, semplicemente per far notare all'utente l'operazione fatta.

- Operatore implicito in una SC se viene applicata la regola ⟨SSC⟩ ::= ⟨Attr⟩⟨Val⟩ o ⟨SSC⟩ ::= ⟨Val⟩⟨Attr⟩ l'operatore di uguaglianza deve essere inserito in modo implicito. Anche in questo caso il nodo viene aggiunto ed etichettato come tale.

- Attributo implicito in una SC se viene applicata la regola ⟨SSC⟩ ::= ⟨Val⟩ oltre all'operatore di uguaglianza da inserire in modo implicito è necessario inserire anche l'attributo a cui appartiene il valore, risalendo a questa informazione in KB. Anche qui i due nuovi nodi sono inseriti e segnalati all'utente.

Una situazione diversa si ha in caso di errori e ambiguità, in cui il sistema non sa come risolvere il problema. In questo caso i nodi che generano questi errori verranno etichettati come tali e daranno il via a una interazione con l'utente nella fase di Disambiguazione.

1. Errori: nel parse tree possono essere riconosciuti degli elementi che consentono di creare una query ma che dal punto di vista semantico sono in realtà sbagliati. Sono riconoscibili verificando le informazioni memorizzate in KB e possono essere:

   - Inconsistenza tra attributo - valore: una ⟨SC⟩ contiene un valore non valido per lo specifico attributo. Il valore potrebbe essere fuori dal dominio dell'attributo (per quanto riguarda gli attributi categorici

dei quali vengono salvati tutti i membri) o incompatibile con il dominio dell'attributo (per gli attributi di tipo numerico ci deve essere un valore numerico, per gli attributi di tipo data ci deve essere una data, etc.).

- Inconsistenza tra misura e operatore di aggregazione: non è possibile applicare l'operatore di aggregazione riconosciuto in una ⟨MC⟩ sulla misura.

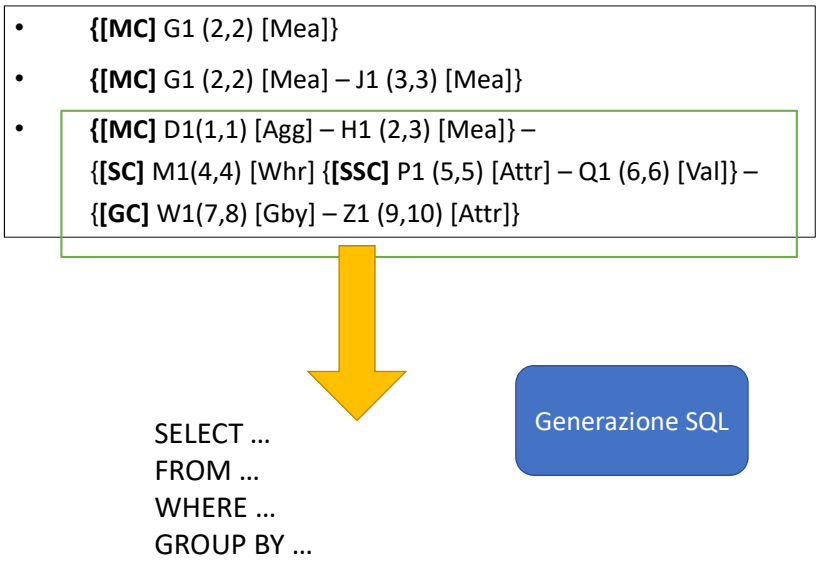
2. Ambiguità: nella frase possono essere riconosciute delle entità che generano delle situazioni di ambiguità; in cui, quindi, c'è più di una operazione eseguibile e il sistema non ha sufficienti informazioni per sapere quale di queste eseguire.

   - Ambiguità rispetto all'operatore di aggregazione in una MC una ⟨MC⟩ potrebbe essere composta solamente da una misura (⟨MC⟩ ::= ⟨Mea⟩), questo perché l'operatore potrebbe essere considerato implicito nel linguaggio parlato. Se alla misura riconosciuta nella frase è possibile applicare più di un operatore questa situazione crea un'ambiguità che deve essere risolta chiedendo all'utente quale degli operatori utilizzare.
   - Ambiguità a causa di sinonimi e omonimi: come è possibile vede nell'E-R in Figura 3.3 la relazione tra la tabella SYNONYM e SYNONYM_LOCATION è di tipo uno-a-molti, ciò che significa che un termine può essere sinonimo di più oggetti del database. Ciò non è un problema se gli oggetti appartengono a categorie differenti poiché porteranno ad avere dei parse tree differenti, se invece gli oggetti appartengono alla stessa categoria i parse tree generati (dai mapping uguali per tutte le entità a parte quella che genera l'ambiguità)

sono esattamente identici e non c'è modo di sapere qual'è il migliore dei due; in questo caso occorre chiedere all'utente quale delle interpretazioni è corretta. Ad esempio nel nostro DW di esempio il termine "Salem" è presente sia come valore dell'attributo "city" nella gerarchia "customer" che dell'attributo "store_city".

3.5.6 Generazione SQL

Supponiamo quindi di arrivare ad avere una serie di parse tree corretti sia dal punto di vista sintattico che dal punto di vista semantico, ordinati secondo un criterio di correttezza decrescente, per cui i primi k sono più rilevanti. Anche se la fase di disambiguazione non è ancora stata sviluppata questa porzione di sistema è già eseguibile.



A questo punto l'operazione di generazione del codice SQL è un'operazione algoritmica molto semplice. I tre nodi principali che ci possono essere nel parse tree sono tre:

– ⟨MC⟩: è la clausola più semplice da gestire. Arrivati a questa fase tutte le ⟨MC⟩ conterranno altre ⟨MC⟩ oppure misura e operatore (eventualmente aggiunto in fase di controlli), quello che deve essere fatto è scorrere l'albero e memorizzare dopo *"SELECT"* i vari nodi nella forma di *operatore(misura)*. Tutte le misure sono memorizzate nella *fact table* per cui non è necessario memorizzare nient'altro.

– ⟨SC⟩: anche in questo caso i nodi degli alberi che arrivano a questa fase sono completi (attributo, operatore, valore) e corretti; quello che c'è da fare è semplicemente concatenare il valore dei vari nodi dopo la clausola di *"WHERE"* e memorizzare in una lista gli attributi riconosciuti per poi andare a recuperare le tabelle fisiche in cui sono memorizzati. Il termine di *where* (Whr) viene ignorato.

– ⟨GC⟩: composta, oltre che dal termine di *group by* (Gby, che in questa fase non viene considerato), da una lista di attributi; a questo punto è sufficiente memorizzare tutti i loro nomi dopo *"GROUP BY"*. Vengono anche memorizzati nella lista citata al punto precedente gli attributi.

Dopo aver generato le varie clausole è il momento di riempire il "FROM". Viene quindi inserito innanzi tutto il nome della *fact table*, dopodiché viene analizzata la lista con gli attributi. Per ognuno di essi è possibile risalire alla tabella in cui è memorizzato, informazione presente in KB passando dalla tabella LEVEL (in cui sono salvati i dettagli del livello) alla tabella COLUMN (in cui invece sono salvati i dettagli della colonna fisica in cui è memorizzato il livello, compresa l'informazione della tabella a cui appartiene). Conoscendo le due tabelle è possibile quindi scorrere la tabella RELATIOSHIP che contiene le chiavi su cui fare il join tra *fact table* e la *dimension table* in questione. Attualmente il sistema è in grado di ricostruire solo percorsi di join in cui tutte le *dimension*

table sono direttamente collegate alla *fact table*; per allargare il supporto del sistema occorrerebbe inserire qui un algoritmo per trovare il cammino più corto per arrivare alla *dimension table* secondaria in cui è memorizzato l'attributo.

A questo punto, l'SQL è generato e la query è pronta per essere eseguita sul DW per poi mostrare i risultati.

3.6 Traduzione: operatore OLAP

È il modulo più legato all'aspetto conversazionale e riguarda il raffinamento dei risultati ottenuti con la prima query intera. Non è ancora stata fatta una progettazione dettagliata del modulo né la sua implementazione, ma si potrà riutilizzare una buona parte del modulo progettato per la comprensione della query intera: la pulizia dell'input, l'analisi del testo a n-grammi e la ricerca dei match in KB fino alla creazione dei mapping. Quello che dovrà essere modificato è la grammatica, che in questo momento accetta solo frasi la cui struttura è associabile a una query GPSJ completa. Sarà necessario aggiungere alla fase precedente il salvataggio in Log delle query intera, e aggiungere un modulo che si occuperà di comprendere quale porzione modificare e/o cosa aggiungere.

3.7 Disambiguazione

Il modulo di Disambiguazione non è ancora stato sviluppato in modo dettagliato ed è ancora interamente da sviluppare. L'idea è quella di ridurre al minimo le interazione con l'utente, ma di sfruttarlo in caso di errori e ambiguità, definiti nel paragrafo precedente.

È stato deciso di mostrare all'utente in questa fase solo il parse tree con il punteggio maggiore e le eventuali entità del mapping da cui è stato generato

che, secondo la grammatica, non appartengono alla query GPSJ. Questa scelta è stata fatta per due motivi:

1. Proporre all'utente più di una alternativa potrebbe creare confusione e rendere difficile la contestualizzazione delle domande di disambiguazione.

2. Il parse tree con il punteggio migliore potrebbe non rappresentare la corretta derivazione della frase, ma può essere utilizzato come base per generarla. Questa può essere migliorata in questa fase aggiungendo o rimuovendo clausole attraverso ulteriori iterazioni che verranno gestite dal modulo di Operatore OLAP.

In caso di errori sarà necessario suggerire all'utente delle possibili soluzioni; mentre nel caso di ambiguità le possibili alternative devono essere espresse all'utente. Da ognuna delle annotazioni generate nella fase di controllo di correttezza verranno quindi generate delle domande, anche utilizzando le informazioni memorizzate in KB, che consentiranno all'utente di correggere o eventualmente eliminare le clausole corrispondenti. Ad esempio nel caso di inconsistenza tra misura m e operatore $o1$ potrebbe essere generato un messaggio del tipo "L'operatore $o1$ non è applicabile alla misura m, sceglierne uno tra: $o2$ e $o3$". Dove $o2$ e $o3$ sono i due operatori applicabili alla misura, recuperati da KB.

Un approccio simile sarà utilizzato per le clausole/entità non connesse al parse tree: verrà chiesto all'utente se gli elementi devono essere mantenuti o meno, in caso di risposta affermativa il sistema dovrà essere in grado di aggiungere correttamente i nodi al parse tree, a seconda del tipo di nodo.

3.8 Interfaccia grafica

Attualmente è stata sviluppata un'interfaccia grafica per visualizzare i risultati raggiunti fin'ora, l'obbiettivo non è quello di mostrare i risultati della query (che sarà l'obiettivo finale del framework) ma piuttosto quello di mostrare come lavora il sistema. L'interfaccia consente l'inserimento in formato testuale di una query in linguaggio naturale e, una volta completata l'interpretazione mostra i primi k parse tree generati, consentendo di scegliere il valore di k. Viene mostrato anche il DFM del cubo di foodmart, che attualmente è l'unico caricato in KB per rendere il tutto più comprensibile. Attualmente il DFM è stato disegnato a parte ed è solamente un immagine statica, ma l'idea è quella di generarlo automaticamente in base al cubo a cui si riferirà la query, quando verranno caricati più cubi.

L'interfaccia consente anche di inserire l'interpretazione corretta della query sotto forma delle 3 componenti principali. Queste, devono però essere inserite nel formato esatto:

- MC: *operatore misura, operatore misura, etc.* L'operatore deve essere scritto nel modo esatto ("sum", "avg", "min", "max" o "count") e anche la misura deve essere scritta esattamente com'è nel cubo.

- GBC: *attributo, attributo, etc.* Non è necessario il termine di group by ma gli attributi devono essere scritti esattamente come sono dichiaratati nel cubo.

- SC: *attibuto operatore valore "and/or" attributo operatore valore etc.* Anche in questo caso tutti i valori devono essere scritti nel formato giusto.

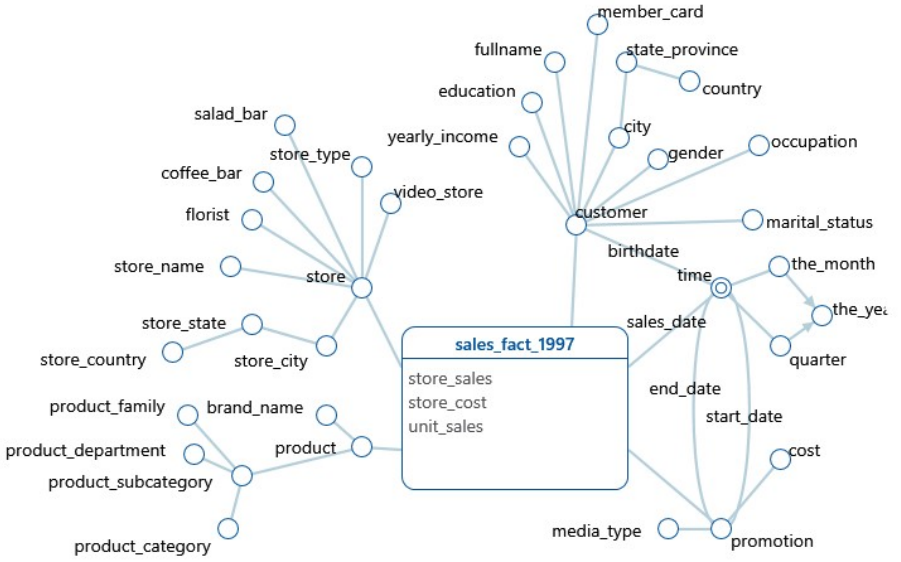
Sono regole molto stringenti ma questa deve essere l'interpretazione corretta definita dall'utente, non si può lasciare l'interpretazione al sistema.

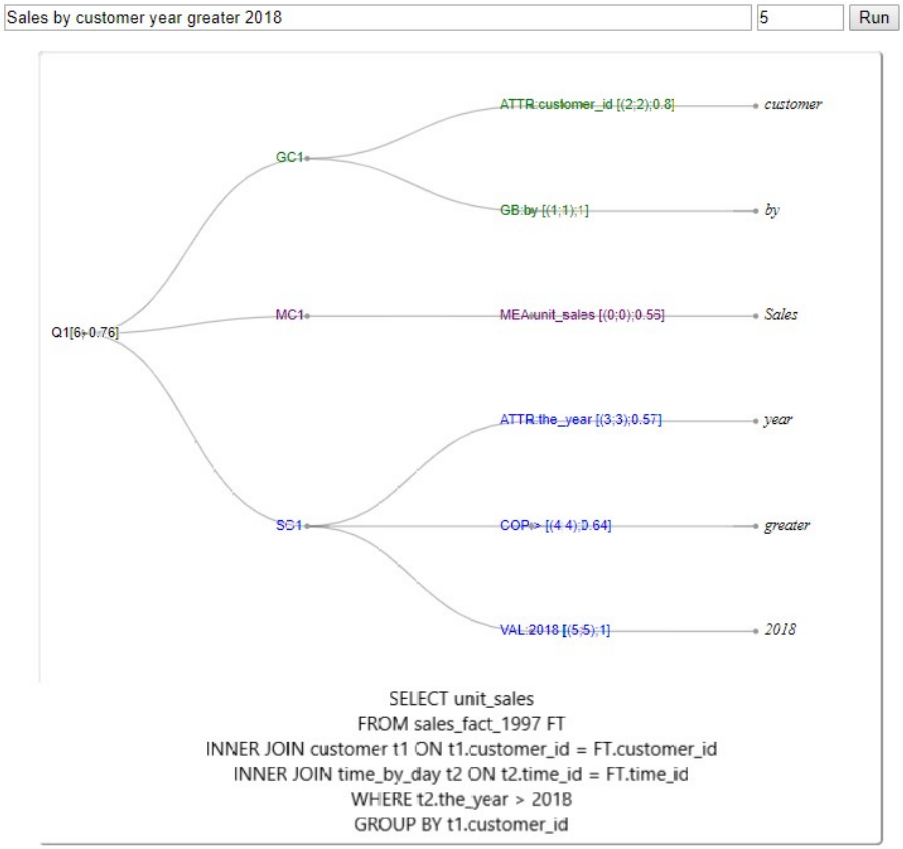
Attualmente si tratta di una applicazione web che comunica con il sistema tramite chiamate chiamate GET. L'interfaccia quindi è stata sviluppata in HTML, CSS e jQuery, in particolare per il disegno degli alberi è stata utilizzata la libreria D3[4]. A livello di applicazione, di conseguenza, una volta che sono stati generati i primi k parse tree devono essere comunicati all'applicazione web in un formato idoneo, in modo che D3 sia in grado di disegnarlo.

Una prima idea per implementare l'interazione con l'utente è quella di estendere questa interfaccia, rendendola interattiva. Prima di tutto anche il DFM deve esser disegnato in modo dinamico, in modo che sia utilizzabile anche in fase di disambiguazione. Anche i prase tree disegnati dovranno essere interattivi, in modo da segnalare all'utente eventuali errori e ambiguità, per consentire agli utenti di inserire le proprie decisioni.

Nella figura sottostante è mostrato uno screenshot dell'applicazione in cui viene mostrato solo il primo dei parse tree calcolati. A destra di ogni riquadro è possibile vedere l'n-gramma della frase iniziale con cui è stato trovato il match con l'entità di KB visibile nel livello precedente insieme alla posizione dell'n-gramma e la similarità. Ci sono poi una serie di livelli che mostrano il parse tree generato dall'elaborazione del mapping, fino alla radice che qui è definita "Q". Vengono mostrate anche le entità (eventualmente raggruppate in clausole) riconosciute ma non appartenenti al parse tree; queste non saranno connesse alla radice ma comunque visibili nel disegno. Infine, viene mostrato il codice SQL del parse tree.

[4] `https://d3js.org/`


member_card
fullname
state_province
country
education
salad_bar
city
yearly_income
occupation
store_type
gender
coffee_bar
video_store
florist
customer
marital_status
birthdate
store_name
store
time
the_month
store_state
sales_date
sales_fact_1997
store_country
store_city
quarter
store_sales
store_cost
unit_sales
end_date
product_family
brand_name
start_date
product_department
product
cost
product_subcategory
media_type
promotion
product_category


Sales by customer year greater 2018
5
Run
ATTR:customer_id [(2;2);0.8]
customer
GC1
GB:by [(1;1);1]
by
Q1[6;0.76]
MC1
MEA:unit_sales [(0;0);0.55]
Sales
ATTR:the_year [(3;3);0.57]
year
SC1
COP:> [(4;4);0.64]
greater
VAL:2018 [(5;5);1]
2018
SELECT unit_sales
FROM sales_fact_1997 FT
INNER JOIN customer t1 ON t1.customer_id = FT.customer_id
INNER JOIN time_by_day t2 ON t2.time_id = FT.time_id
WHERE t2.the_year > 2018
GROUP BY t1.customer_id

Capitolo 4

Test e visualizzazioni

In questo capitolo verranno analizzati i risultati dei test effettuati su C-BI al variare di alcuni dei parametri (riassunti in Tabella 4.1). I test effettuati possono essere divisi in due categorie: test di efficacia e test di efficienza. I primi hanno il compito di valutare quanto il sistema è in grado di svolgere il compito per cui è stato progettato, mentre i test appartenenti alla seconda categoria valutano quante risorse utilizza per farlo.

Per eseguire i test occorre avere un dataset di query in linguaggio naturale; inoltre, per poter valutare l'efficacia del sistema occorre conoscere anche la loro corretta interpretazione. L'interpretazione delle query deve essere calcolata dal sistema, quindi è necessario anche avere a disposizione il cubo a cui le query si riferiscono e caricarlo su KB. Inoltre, occorre ricordare che tra l'insieme di query possibili quelle riconosciute da C-BI sono quelle di tipo GPSJ. Non avevamo a disposizione questo tipo di informazioni, per cui ne abbiamo creato uno contenente 30 query in lingua inglese riferite a foodamrt (Figura 3.3).

Nome parametro	**Significato**	**Default**
ngramSize	Lunghezza massima n-grammi	3
thrMember	Soglia minima per i match con i membri	0.9
thrMetadata	Soglia minima per i match con i metadati	0.4
nSynMemeber	Numero massimo di sinonimi tra i membri	1
nSynMeta	Numero massimo di sinonimi tra i metadati	5
ngramSymThr	Soglia per cui vengono eliminati i match dei sotto elementi dell'n-gramma	0.9
thrNgramDist	Distanza massima tra due n-grammi considerati consecutivi	3
thrCoverage	Percentuale massima di token non considerati in un mapping	0.6

Tabella 4.1: Riassunto dei principali parametri del sistema

4.1 Test di efficacia

I test di efficacia sono legati a quanto il sistema è in grado di comprendere una query espressa in linguaggio naturale. Per effettuare questi test occorre avere un dataset di query in linguaggio naturale a cui è associata la loro corretta interpretazione in SQL. Abbiamo deciso di non calcolare la similarità dal codice SQL ma rispetto ai parse tree.

Validatore

È stato sviluppato un modulo in grado di calcolare l'accuratezza di un'interpretazione del sistema in base alla sua interpretazione corretta. Conoscendo questi due dati verrà analizzata la frase in linguaggio naturale tramite i moduli

descritti precedentemente, al termine verrà preso il parse tree con il punteggio più alto, che ora definiremo I e questo verrà confrontato con il parse tree corretto: C. Come risultato si avrà un valore numerico che definisce quanto questi due sono simili. Abbiamo deciso di inserire le interpretazioni corrette in linguaggio naturale suddivise nei tre principali componenti di una GPSJ: MC, SC e GBC. A questo punto per la generazione di C queste vengono analizzate:

- Deve essere presente una MC composta da una misura e da un operatore o da un insieme di MC composte a loro volta da misura e operatore.
- Se è presente una GBC viene verificato che sia composta da attributi.
- Se è presente una SC viene controllata la sua correttezza sia sintattica (attributo, operatore, valore) che semantica.

Come già detto nella Sezione 3.8 nelle giuste interpretazioni i nomi degli attributi e delle misure devono essere inserite esattamente come sono memorizzate nel cubo, se così non è questa fase fallisce. Se invece è tutto corretto viene generato il mapping, che viene poi dato in input al modulo di parsing definito prima, in modo da avere il parse tree corretto. Al termine di queste due operazioni si avranno i due parse tree.

Per calcolare la differenza tra i due alberi è stata utilizzata la distanza di Zhang e Shasha [39]. Questa formula, presi due alberi conta quanti nodi sono stati aggiunti, modificati o rimossi dall'uno rispetto all'altro. È importante considerare che nei nodi derivati non viene controllata l'effettiva uguaglianza (ovvero che i figli siano uguali) ma semplicemente che siano dello stesso tipo. Questa misura è una distanza, la similarità di due parse tree p_1 e p_2 con rispettivamente $|p_1|$ e $|p_2|$ foglie è quindi definita come

$$Sim(P_1, P_2) = 1 - \frac{ZhangShashaDist(P_1, P_2)}{\max(|P_1|, |P_2|)}$$

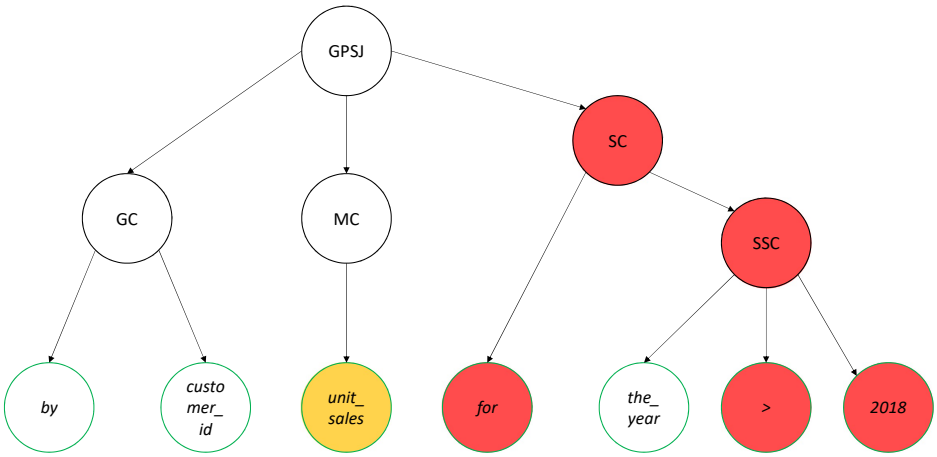


Figura 4.1: P_1 con $|P_1| = 12$

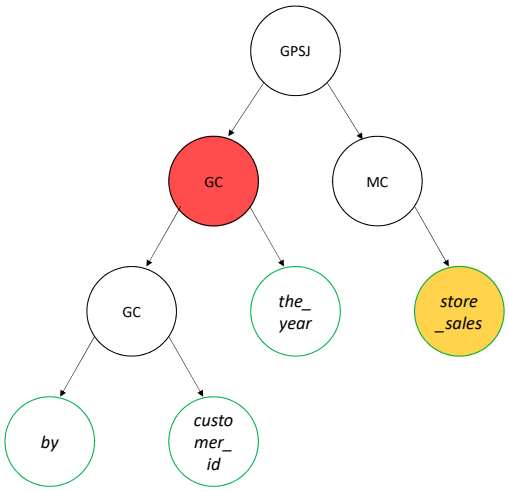


Figura 4.2: P_2 con $|P_2| = 8$

Dati, ad esempio i due parse tree in Figura 4.1 e 4.2 la loro distanza è 7:

1. I 6 nodi rossi sono quelli che non sono presenti in nessun modo nell'altro albero;

2. I due nodi arancioni sono il rispettivo l'uno dell'altro, ma avendo un valore diverso contano come 1 differenza.

Notare che invece il nodo con la MC, sebbene sia diverso (ovvero ha dei figli diversi) viene considerato comunque presente in entrambi gli alberi quindi non viene contato come differenza.

La similarità tra i due alberi sarà quindi $= 1 - \frac{7}{12} \simeq 0.4$

Test

Definito questo componente per eseguire il test è sufficiente calcolare questo punteggio per ogni query del dataset. Per ogni query in linguaggio naturale vengono calcolati i primi k parse tree e il parse tree corretto.

In questo caso i parametri vengono mantenuti tutti con il loro valore di deafult all'infuori del numero massimo di entità memorizzato: fissato a 1 il numero di entità tra i membri vengono analizzate le prestazioni del sistema con un diverso numero di entità tra i metadati: 1 e 5.

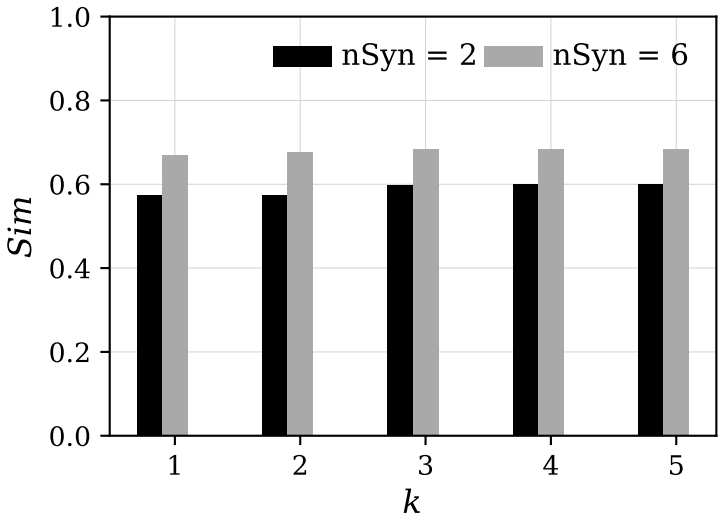


Figura 4.3: Risultati test similarità massima al variare del numero di risultati presi in considerazione (k) e del numero di sinonimi.

Per generare il grafico in Figura 4.3 sono stati generati i primi k parse tree di ogni query, al variare di k; per ognuno di essi è stata calcolata la similarità con il parse tree corretto ed è stato preso il valore maggiore; infine, è stata calcolata la media di tutte le query di questo valore.

Dal grafico è possibile notare che il numero di sinonimi influenza il punteggio, 2 sinonimi sono troppo pochi per gestire le possibili variazioni dei nomi degli elementi con i nomi inseriti dagli utenti. Al contrario, k non influenza, se non in minima parte, i risultati. Ciò significa che, quasi sempre, il parse tree generato dal nostro sistema con un punteggio maggiore è quello che ha anche una maggiore similarità con quello corretto.

4.2 Test di efficienza

I test di efficienza sono invece mirati a valutare le risorse che impiega il sistema per svolgere il suo compito. In questo caso non viene presa in considerazione l'interpretazione corretta ma ciò che viene fatto è analizzare la varie fasi di produzione del parse tree.

4.2.1 Tempo

Il primo test eseguito in merito all'efficienza è legato ai tempi di esecuzione. Il grafico mostra il tempo medio impiegato nell'elaborazione di tutti i mapping contenente un diverso numero di entità (valori sull'asse X). Anche in questo caso l'unico parametro modificato è il numero di sinonimi tra i metadati che sono 1 e 5. Come è ovvio prendere in considerazione un maggior numero di sinonimi porta ad avere un numero maggiore di mapping della stessa lunghezza, e di conseguenza ad avere tempi di esecuzione maggiori; si rimane comunque nell'ordine dei secondi.

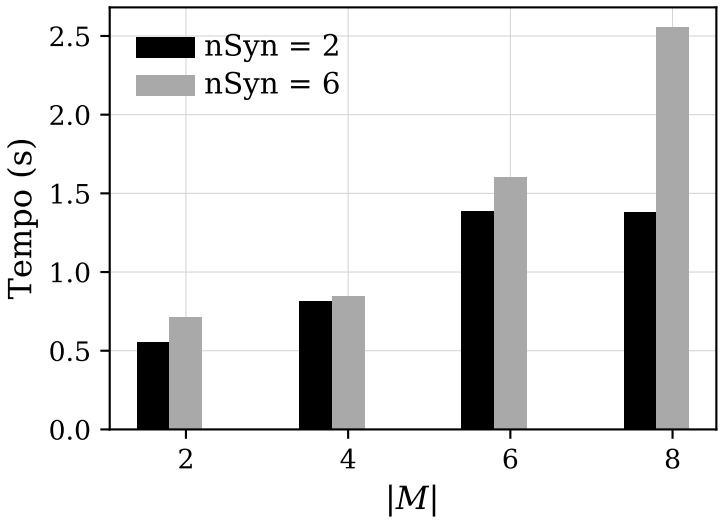


Figura 4.4: Tempi di esecuzione del parsing in base al numero di entità nel mapping

4.2.2 Pruning

L'ultimo test vuole mostrare gli effetti del pruning sui mapping generati e analizzati dal parser in base al numero di entità contenute nel mapping. Fare *"pruning"* (letteralmente "potatura") significa eliminare un sottoinsieme di risultati dall'insieme complessivo sulla base di una determinata regola.

Per l'esecuzione di questo test vengono mantenuti i valori di default per tutti i parametri, fissando il numero di sinonimi massimo a 6: 1 per i membri e 5 per i metadati.

Come è già stato detto il numero di mapping è molto elevato viene generato, seppur con qualche limitazione, calcolando il power set dell'insieme di entità trovate. Il numero maggiore di mapping si ha, ovviamente, in caso di nessun pruning; ovvero vengono generati tutti i mapping che rispettano i vincoli di: non sovrapposizione di entità, distanza massima e copertura minima della

frase. Maggiore è il numero di entità della frase (valori nell'asse X), maggiore è il numero di mapping analizzati. La prima operazione di pruning si occupa di rimuove i mapping uguali, ovvero contenenti le stesse entità ma collegate a n-grammi differenti. Già questa operazione porta ad avere una riduzione notevole, indipendentemente dalla lunghezza del mapping. Infine, la linea più bassa conta il numero di mapping realmente analizzati dal parser. Questo numero è ancora inferiore per via dell'ottimizzazione che viene fatta in fase di generazione dei parse tree in base al punteggio del peggiore dei k parse tree (tutti quelli con un potenziale punteggio inferiore non vengono calcolati). In conclusione, l'aumento di entità porta a un incremento nel numero di mapping, ma rispetto a tutti i mapping generati quelli che vengono davvero analizzati sono meno della metà.

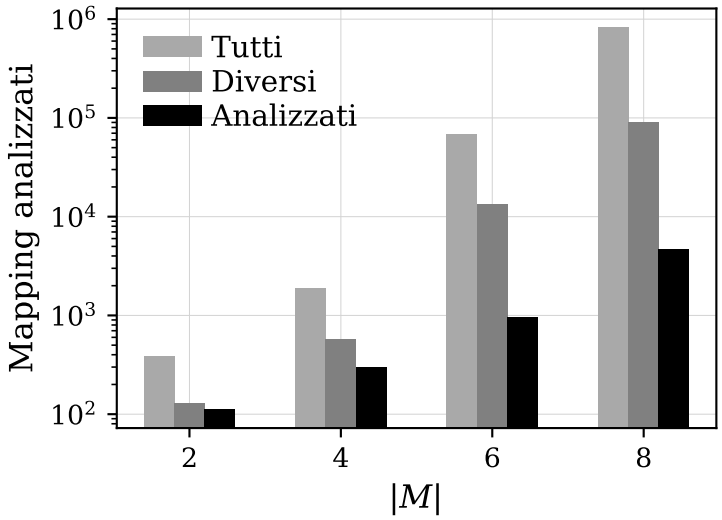


Figura 4.5: Numero di mapping analizzati in fase di parsing al variare della modalità di generazione dei mapping stessi.

Conclusioni

Nella tesi viene presentato C-BI: il nostro framewrok progettato per eseguire sessioni di analisi OLAP comunicando con l'utente utilizzando il linguaggio naturale. La progettazione di questo sistema ha toccato due grandi aree di interesse: i sistemi conversazionali e la Business Intelligence. Per quanto riguarda i primi, in particolar modo, è stata effettuata un'approfondita ricerca in merito agli assistenti personali e alle interfacce che consentono agli utenti di eseguire query SQL su database relazionali, due tematiche molto in voga attualmente in letteratura.

I risultati in termini di efficacia possono essere migliorati apportando alcune migliorie alla porzione di sistema già implementato:

- Utilizzo di sinonimi: attualmente il sistema carica solamente il termine stesso e nessun tipo di sinonimo. Lasciando all'utente un minimo grado di libertà legato più che altro a coniugazioni o errori di battitura l'utente dovrà comunque inserire il termine esattamente com'è memorizzato nel cubo. Ovviamente ciò limita molto le prestazioni del sistema, potrebbero quindi essere inseriti nuovi sinonimi per i termini già presenti in modo manuale o automatico.

- Estensione della grammatica: attualmente la grammatica non riconosce tutti i possibili componenti di una query GPSJ. Manca il concetto di

COUNT, che può essere associato al fatto o a un attributo, e anche il BETWEEN nei vari predicati di selezione.

Per quanto riguarda i principali sviluppi futuri, l'obiettivo principale è completare il progetto con lo sviluppo dei moduli o sottomoduli che attualmente sono stati solamente progettati ma non ancora sviluppati:

1. Controlli di correttezza: sebbene siano già stati definiti quali controlli fare una volta generati i parse tree, questa porzione di sistema non è ancora stata sviluppata.

2. Raffinamento dei risultati: al momento il sistema ricerca in ogni suo inserimento una intera query GPSJ; dovrà quindi essere implementato il salvataggio della query generata, così come il modulo di Operatore OLAP che si occupa concretamente di comprendere la frase inserita e di modificare la query precedente nel modo adeguato. Infine, deve essere sviluppato l'aspetto conversazionale di questo approccio iterativo.

3. Disambiguazione: tutta la porzione del sistema relativa all'aspetto conversazionale ancora non è stata sviluppata. Manca sia l'aspetto legato all'esposizione delle incertezze in linguaggio naturale, che la parte di interpretazione della risposta.

Per sviluppare Conversational nella sua interezza i punti mancanti sono quelli sopracitati; possono esserci poi una serie di migliorie da effettuare per migliorare le prestazioni del sistema.

– Innanzi tutto è possibile aggiungere il supporto ad altre lingue, con la consapevolezza che il supporto di una lingua dipende dalla lingua del cubo. Per farlo è necessario adattare i termini inseriti manualmente, sia quelli legati al linguaggio parlato che i nomi degli operatori e i sinonimi

di essi. Potrebbero essere poi necessarie modifiche per adattare la fase di pulizia dell'input (come la lemmatizzazione e la rimozione delle stop word).

– Sebbene l'implementazione ROLAP di un cubo sia la modalità maggiormente scelta, non è l'unica alternativa possibile. Si potrebbe quindi pensare di estendere la fase di caricamento del DW in KB anche a cubi memorizzati in altri modi (MOLAP o HOLAP), tuttavia, non esistendo standard sul mercato per quanto riguarda questi approcci ed essendo scarsamente utilizzati questa fase potrebbe risultare molto faticosa e non altrettanto utile.

Ringraziamenti

Grazie al prof. Golfarelli che mi ha dato l'opportunità di fare un progetto di tesi stimolante e interessante, seguendomi in modo paziente. Grazie a Matteo Francia, aiuto fondamentale per arrivare al termine di questo progetto; che insieme a Enrico Gallinucci mi hanno fatto conoscere un'università in cui c'è gente che ama ciò che fa, senza competizioni in cui non viene mai negato l'aiuto a nessuno. Grazie perché insieme all'Anna, spalla femminile e compagna di risate, e Nicola, con il quale abbiamo condiviso ormai diversi banchi e scrivanie, hanno reso questi mesi di lavoro meno pesanti, anche ad Agosto in una stanza senza finestre.

Grazie a chi ha condiviso con me questo lungo cammino per arrivare alla laurea, perché anche la peggior salita è meno faticosa se affrontata tra chiacchiere, risate e ansia condivisa.

Grazie ai miei genitori che mi hanno sempre permesso coltivare le mie passioni e i miei studi, senza farmi mai mancare nulla, ma insegnandomi a non dare nulla per scontato. Grazie a tutta la mia enorme famiglia che continua a crescere, perché so di avere l'enorme fortuna di potermi rivolgere sempre qualcuno in caso di bisogno.

Grazie agli amici di tutta la vita, dai quali ho imparato che non importa a quanti kilometri di distanza ti porterà la vita e quanto le strade si possano separare, quando si è insieme è sempre come se non cu si fosse mai separati.

Grazie alla lanterna, che mi ha insegnato che non ci sono ostacoli insormontabili, ma solo se provi a superarli accompagnato da maestri saggi e pazienti e se lo fai in compagnia di persone che puoi considerare una famiglia.

Grazie agli amici dell'AC, relazioni speciali, costruite a cuore aperto, senza la paura di dire davvero ciò che si pensa; per le montagne scalate, i bagni in mare, i canti con la chitarra, le notti con poche ore di sonno, i sorrisi dei bambini e le domande dei giovanissimi, momenti in cui mi sento davvero nel posto giusto al momento giusto.

Grazie agli amici del gruppo dal nome improbabile, perché di anni ne passati diversi da quando abbiamo lasciato i banchi di scuola, ma l'amicizia resta.

Bibliografia

[1] T. Klüwer. *From chatbots to dialog systems.* In Conversational agents and natural language interaction: Techniques and Effective Practices (pp. 1-22). IGI Global, 2011.

[2] M. L. Mauldin. *Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition.* In AAAI (Vol. 94, pp. 16-21), 1994.

[3] J. Weizenbau. *ELIZA—a computer program for the study of natural language communication between man and machine.*
Communications of the ACM, 9(1), 36-45, 1966.

[4] A. Vegesna *Ontology based Chatbot (For Ecommerce Website).*
Vol. 179, no. 14, pp. 51–55, 2018.

[5] H. Al-Zubaide and A. A. Issa. *OntBot: Ontology Based Chatbot.*
Proc. IEEE of 2011 Fourth International Symposium on Innovation in Information & Communication Technology (ISIICT), 2011, pp. 7-12.

[6] Michael Mctear.
Conversational Modelling for ChatBots: Current Approaches and Future Directions. Technical report, Ulster University, Ireland, 2018.

[7] M. Tyagi, *Natural Language Interface to Databases: A Survey* International Journal of Science and Research (IJSR), Volume 3 Issue 5, pp. 1443-1445, May, 2014.

[8] W.A. Woods, R.M. Kaplan, and B.N. Webber. *The Lunar Sciences Natural Language Information System: Final Report.* BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.

[9] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. *Developing a Natural Language Interface to Complex Data.* ACM Transactions on Database Systems, 3(2):105–147, 1978

[10] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. *Natural Language Interfaces to Databases - An Introduction.* Natural Language Engineering, vol 1, part 1, pages 29–81, 1995.

[11] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. *Keyword Searching and Browsing in Databases using BANKS.* Proc. ICDE, 2002.

[12] S. Tata and G. M. Lohman. *SQAK: Doing More with Keywords.* ACM SIGMOD, 2008.

[13] A.-M. Popescu, O. Etzioni, and H. Kautz. *Towards a Theory of Natural Language Interfaces to Databases.* IUI, 2003.

[14] D Saha, A Floratou, K Sankaranarayanan. *ATHENA: An OntologyDriven System for Natural Language Querying over Relational Data Stores.* Proc. VLDB Endowment, 9(12):1209-1220, 2016.

[15] F. Li and H. V. Jagadish.
Constructing an Interactive Natural Language Interface for Relational Databases. Proceedings of the VLDB Endowment, 8(1):73–84, 2014.

[16] M. Golfarelli, S. Rizzi. *Data Warehouse: Teoria e pratica della progettazione.* New York City: McGraw-Hill, 2006.

[17] M. Golfarelli, D. Maio, S. Rizzi. *The dimensional fact model: A conceptual model for data warehouses.* International Journal of Cooperative Information Systems, 7(02n03), 215-247, 1998.

[18] D. Jurafsky, J. H. Martin. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* 2018.

[19] J. W. Backus. *The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference.* Proceedings of the International Comference on Information Processing, 1959.

[20] I. Feinerer and K. Hornik. *WordNet Interface* 2016.

[21] N. Stratica, I. Kosseim, B.C. Desai. *NLIDB Templates for Semantic Parsing.* In: Applications of Natural Language to Databases (NLDB 2003), Germany, pp. 235–241, 2003.

[22] V. I. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals.* In Soviet physics doklady (Vol. 10, No. 8, pp. 707-710), 1966.

[23] A. Gupta, V. Harinarayan, D. Quass. *Aggregate-query processing in data warehousing environments.* In Proceedings of the 21th International Conference on Very Large Data Bases, VLDB '95, pages 358–369, 1995.

[24] I. Trummer, Y. Wang, S. Mahankali. *A holistic approach for query evaluation andresult vocalization in voice-based OLAP.* In Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019., pages 936–953, 2019.

[25] L. Etcheverry, A. A Vaisman. *Qb4olap: a new vocabulary for olap cubes on the semantic web.* In Proceedings of the Third International Conference on Consuming Linked Data, volume 905, pages 27–38, 2012.

[26] V. I. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals.* In Soviet physics doklady, Vol. 10, No. 8, pp. 707-710, 1966.

[27] R. Elshawi, M. Maher, S. Sakr. *Automated Machine Learning: State-of-The-Art and Open Challenges.* arXiv preprint arXiv:1906.02287, 2019.

[28] Hype cycle for artificial intelligence, 2018.
`http://www.gartner.com/en/documents/3883863/hype-cycle-for-artificial-intelligence-2018.`
Visualizzato: 21/06/2019.

[29] F. Li, H. V. Jagadish. *Understanding natural language queries over relational databases.* SIGMOD Record, 45(1):6–13, 2016.

[30] T. Johnson. *Natural Language Computing: The Commercial Applications.* Ovum Ltd., London, 1985.

[31] D.L. Waltz. *An English Language Question Answering System for a Large Relational Database.* Communications of the ACM, 21(7):526–539, 1978.

[32] P. Auxerre. *MASQUE Modular Answering System for Queries in English - Programmer's Manual.* Technical Report AIAI/SR/11, Artificial Intelligence Applications Institute, University of Edinburgh, March 1986.

[33] J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, S. Rizzi. *A collaborative filtering approach for recommending OLAP sessions.* Decision Support Systems, 69:20–30, 2015.

[34] M. Francia, M. Golfarelli, S. Rizzi. *Augmented Business Intelligence.* In DOLAP, 2019.

[35] I. Gur, S. Yavuz, Y. Su, X. Yan. *Dialsql: Dialogue based structured query generation.* In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers, pages 1339–1349, 2018.

[36] J. Sen, F. Ozcan, A. Quamar, G. Stager, A. R. Mittal, M. Jammi, C. Lei, D. Saha, K. Sankaranarayanan. *Natural language querying of complex business intelligence queries.* In Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019., pages 1997–2000, 2019.

[37] J. C. Beatty. *On the relationship between ll(1) and lr(1) grammars.* J. ACM, 29(4):1007–1022, October 1982.

[38] Point And Click Interface, definizione. `https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/point-and-click-interface`. Visualizzato: 23/09/2019.

[39] K. Zhang, R. Statman, D. Shasha. *On the editing distance between unordered labeled trees.* Information processing letters, 42(3), 133-139. 1992.

[40] K. Dhamdhere, K. S. McCurley, R. Nahmias, M. Sundararajan, Q. Yan. *Analyza: Exploring data with conversation.* In Proceedings of the 22nd International Conference on Intelligent User Interfaces, IUI 2017, Limassol, Cyprus, March 13-16, 2017, pages 493–504, 2017.

[41] C. Baik, H. V. Jagadish, Y. Li. *Bridging the semantic gap with SQL query logs in natural language interfaces to databases.* CoRR, abs/1902.00031, 2019.

[42] D. Gale, L. S. Shapley. *College admissions and the stability of marriage.* The American Mathematical Monthly, 69(1), 9-15, 1962.