ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Compressione e Vocalizzazione di Risultati Multidimensionali nel Paradigma OLAP

Tesi di laurea in
Big Data

Relatore
Dott. Enrico Gallinucci

Correlatore
Dott. Matteo Francia

Candidato
Tommaso Bombardi

Terza Sessione di Laurea
Anno Accademico 2019-2020

Sommario

Si definiscono conversazionali i sistemi che consentono all'utente di interagire utilizzando il linguaggio naturale. Negli ultimi anni essi sono stati oggetto di numerose ricerche in ambito informatico, basti pensare che strumenti quali assistenti vocali e chatbot sono oramai di uso comune. Quando questi sistemi vengono sfruttati per l'analisi dati si parla di una vera e propria democratizzazione dell'accesso ai dati, perché le informazioni diventano accessibili anche ad utenti che non hanno competenze tecniche. Il problema da affrontare nello sviluppo dei sistemi conversazionali può essere suddiviso in due parti complementari: la costruzione di interfacce in grado di tradurre un comando espresso in linguaggio naturale in un'operazione da svolgere su un sistema informatico e la presentazione dei risultati ottenuti con un output dello stesso tipo. Nello scenario applicativo dei database sono stati realizzati vari framework per la comprensione del linguaggio naturale, mentre solo di recente è stato introdotto e esplorato il problema della data vocalization. Esso è finalizzato alla presentazione dell'output in formato vocale ed è caratterizzato da esigenze differenti rispetto alla data visualization. La tesi presentata si colloca nell'ambito della Conversational Business Intelligence e si pone l'obiettivo di studiare e implementare una tecnica per la vocalizzazione di risultati di query OLAP. Questa dovrà poi essere integrata in un framework esistente, per ottenere un sistema conversazionale end-to-end in cui input e output sono espressi in formato vocale.

A chi non si accontenta mai

Indice

Elenco delle figure

Capitolo 1

Introduzione

Una delle principali tematiche su cui si è concentrata la ricerca informatica negli ultimi anni è la democratizzazione dell'accesso ai dati. In uno scenario tradizionale l'utente, per accedere a servizi di visualizzazione e analisi dati, deve necessariamente avere competenze informatiche. Superare questo limite offrirebbe un grande numero di nuove possibilità, specialmente in un contesto come quello attuale, caratterizzato da una crescita continua della mole di dati a disposizione.

Nell'ambito della democratizzazione dell'accesso ai dati un ruolo centrale è quello ricoperto dai sistemi conversazionali, che consentono agli utenti un'interazione espressa in termini umani e, quindi, in linguaggio naturale. La progettazione e lo sviluppo di questo tipo di sistemi sono attività che richiedono un duplice sforzo, poiché è necessario sia comprendere il linguaggio umano sia presentare i dati rendendo l'esperienza dell'utente quanto più naturale possibile.

Gli esempi di successo delle ricerche in ambito conversazionale sono vari, basti pensare agli assistenti vocali integrati negli smartphone o nei computer (quali Siri, Google Assistant e Cortana) e a quelli indipendenti (come ad esempio Amazon Alexa e Google Home). Si tratta di strumenti che sono oramai di uso comune sia in ambienti privati sia in ambito aziendale, e che diventano essenziali laddove l'interazione con l'utente sia obbligatoriamente hands-free [13].

Come è facile intuire, i sistemi conversazionali condividono le aree applicative toccate dai vari paradigmi per l'accesso e l'analisi dei dati. Un'intersezione molto interessante è quella con la Business Intelligence, termine con cui si fa riferimento

agli strumenti e alle procedure che consentono ad un'azienda di trasformare i propri dati di business in informazioni utili al processo decisionale. Mentre una classica interrogazione di un data warehouse prevede una query espressa formalmente (ad esempio in linguaggio SQL) e un risultato in formato tabulare, in un contesto conversazionale sia l'input sia l'output devono essere in linguaggio naturale.

In questo ambito, si collocano varie attività di ricerca che hanno portato alla realizzazione di sistemi in grado di comprendere le richieste dell'utente espresse in linguaggio naturale, eseguire query OLAP e mostrare i risultati [38, 39].

Uno dei framework conversazionali più recenti è COOL [12], che consente di trasformare le interrogazioni dell'utente in SQL, eseguirle su data warehouse e visualizzare i risultati. COOL è stato concepito mantenendo il focus sull'interattività, la principale caratteristica delle sessioni OLAP, tanto che con l'utente viene instaurata una vera e propria conversazione. Questa consente al sistema di risolvere eventuali ambiguità prodotte in fase di comprensione, rivolgendosi direttamente all'utente. Inoltre il framework non permette soltanto di esprimere un'interrogazione inizializzandola da zero, ma offre anche la possibilità di farlo modificando la precedente (di cui si tiene traccia). Un'altra importante caratteristica è la genericità, perché il sistema è in grado di operare su qualsiasi data warehouse.

COOL e gli altri framework conversazionali esistenti hanno raggiunto un obiettivo significativo, ossia la costruzione di interfacce conversazionali attraverso le quali è possibile interrogare un data warehouse. Alcune recenti attività di ricerca si sono concentrate sul problema complementare, studiando e proponendo tecniche per la trasformazione del risultato di una query in output vocale [46, 47, 48]. La combinazione di queste metodologie con le interfacce disponibili apre ad uno scenario "end-to-end" nuovo e molto promettente. Esso può essere infatti definito completo dal punto di vista conversazionale perché, dato un input espresso in linguaggio naturale, prevede la restituzione di un output dello stesso tipo.

Questa tesi si pone quindi l'obiettivo di studiare ed implementare una tecnica per la vocalizzazione dei risultati di query OLAP, in modo da poter estendere il framework COOL e offrire ad esso la possibilità di restituire vocalmente l'output ottenuto. L'elaborato presentato è composto dai seguenti capitoli:

- Nel Capitolo 2 è riportata una panoramica delle tematiche trattate da questo lavoro di tesi. In particolare, vengono date le nozioni di base necessarie ad introdurre la Business Intelligence, facendo riferimento al modello multidimensionale e alle query OLAP, e i sistemi conversazionali. Per quanto riguarda questi ultimi, dopo una panoramica generale, si illustra il problema della data vocalization, si evidenziano le esigenze legate ad esso e si riportano i risultati delle attività di ricerca svolte in questo ambito.

- Il Capitolo 3 è dedicato alla presentazione dell'approccio scelto per la valutazione e la vocalizzazione di risultati di query OLAP e contiene tutte le nozioni necessarie a comprendere il suo funzionamento.

- Nel Capitolo 4 sono riportati i dettagli relativi all'implementazione della metodologia scelta, con particolare riferimento a quanto fatto per automatizzare l'approccio, rendendolo utilizzabile in un qualsiasi data warehouse, e per ottimizzarlo, al fine di migliorare i risultati ottenuti con il metodo originale. Infine, viene descritta la fase di integrazione che ha consentito l'utilizzo della nuova implementazione nel framework conversazionale esistente.

- Nel Capitolo 5 sono riportati i risultati ottenuti eseguendo test di efficacia e di efficienza sull'implementazione realizzata, accompagnati dalla presentazione dei data mart usati e delle metriche scelte. Ai risultati raggiunti si aggiunge una loro interpretazione, in cui essi sono valutati sia in termini assoluti sia in rapporto a quanto ottenuto con l'approccio originale.

- Il Capitolo 6, inserito a completamento dell'elaborato, è dedicato alle conclusioni e ai possibili sviluppi futuri.

Capitolo 2

Conversational business intelligence

Con il termine *Conversational Business Intelligence* si fa riferimento ad un'estensione della Business Intelligence tradizionale, arricchita da una componente conversazionale. Quest'ultima offre all'utente la possibilità di effettuare sessioni di analisi utilizzando il linguaggio naturale per interagire con il sistema. Sebbene rappresenti l'intersezione tra due tematiche per cui è disponibile un'ampia letteratura, si tratta di un ambito di ricerca ancora relativamente poco esplorato.

Questo secondo capitolo fornisce una panoramica generale dell'argomento, necessaria per comprendere il lavoro presentato. Nella Sezione 2.1 sono riportate le nozioni di base necessarie per introdurre la Business Intelligence e, in particolare, vengono presentati concetti fondamentali quali *data warehouse*, *modello multidimensionale* e *query OLAP*. La Sezione 2.2 contiene invece un'introduzione ai sistemi conversazionali, in cui si approfondiscono le tematiche di *query answering* e *data vocalization* e si riportano i risultati di alcune attività di ricerca significative.

2.1 Data warehousing

La Business Intelligence è un insieme di strumenti e procedure che consentono ad un'azienda di trasformare i propri dati di business in informazioni utili al processo decisionale [34]. Negli ultimi anni, con l'aumento del volume di dati operazionali generati da processi gestionali, la sua importanza è cresciuta costantemente.

Tra gli strumenti di Business Intelligence il più diffuso è il data warehouse, che è definito come una collezione di dati di supporto al processo decisionale [16]. Esso presenta una serie di caratteristiche specifiche, in particolare:

- È orientato ai soggetti d'interesse, perché è modellato sulla base dei concetti d'interesse dell'azienda, quali clienti, prodotti, vendite e ordini. Questo è il primo aspetto che differenzia i data warehouse dai databse operazionali, organizzati invece in base alle applicazioni del dominio aziendale.
- È integrato e consistente, in quanto si appoggia a fonti di dati eterogenee che possono provenire sia da aree aziendali sia da sistemi informativi esterni.
- È rappresentativo dell'evoluzione temporale e non volatile, poiché deve permettere analisi che spazino sulla prospettiva di alcuni anni. Per questo motivo, il data warehouse viene aggiornato a intervalli regolari a partire dalle sorgenti dati operazionali ed è in continua crescita.

Un tipico scenario di utilizzo è quello di una grande azienda composta da diverse filiali, ognuna delle quali dotata di un proprio database in cui avvengono giornalmente operazioni elementari che generano una grande quantità di dati operazionali. Nel caso in cui una figura dirigenziale abbia intenzione di svolgere un'analisi per valutare l'andamento globale dell'impresa, dovrebbe innanzitutto rivolgersi ai tecnici incaricati della gestione dei database nelle singole filiali. Questi ultimi dovrebbero quindi occuparsi della fase di integrazione, con probabili difficoltà aggiuntive da affrontare in caso di database indipendenti, per poi calcolare dei valori di sintesi in grado di riassumere la situazione dell'azienda.

Questo modus operandi non è né efficiente né efficace, perché porta ad un inutile consumo di tempo e di risorse e non sempre consente di ottenere il risultato desiderato. Il data warehouse è uno strumento che nasce con l'obiettivo di semplificare ed ottimizzare queste operazioni e funge da "contenitore" unico in cui caricare tutti i dati operazionali provenienti da diverse sorgenti, in un formato adatto a consentire la consultazione anche ai non esperti di informatica.

L'idea alla base del data warehose non è quella di sostituire i database tradizionali, che soddisfano esigenze completamente differenti, ma è quella di separare le

elaborazioni legate alle transazioni (OLTP, *On-Line Transactional Processing*) da quelle di tipo analitico (OLAP, *On-Line Analytical Processing*). Mentre le prime continuano a supportare l'operatività delle aziende, le seconde effettuano un'analisi che richiede la scansione di un'enorme quantità di record per calcolare un insieme di dati di sintesi. I data warehose supportano le elaborazioni OLAP.

2.1.1 Modello multidimensionale

Il paradigma di rappresentazione dei dati adottato dagli strumenti di data warehousing è il modello multidimensionale, scelto perché intrinsecamente semplice e intuitivo anche per gli utenti non esperti di informatica. Queste sue caratteristiche derivano, a loro volta, dalla grande diffusione di applicazioni che prevedono l'utilizzo di fogli elettronici come strumenti di produttività individuale [16].

Alla base del modello multidimensionale vi è la constatazione che le entità in grado di influenzare il processo decisionale sono *fatti* del mondo aziendale, come ad esempio vendite, spedizioni o interventi. Le occorrenze di un fatto corrispondono agli *eventi* accaduti e sono rilevanti le *misure* che li descrivono quantitativamente, quali incasso delle vendite, quantità spedita o durata degli interventi.

Gli eventi che accadono nell'azienda sono troppo numerosi per poter essere analizzati singolarmente e vengono perciò collocati in uno spazio n-dimensionale i cui assi, chiamati *dimensioni* di analisi, definiscono delle prospettive per la loro identificazione. Ad esempio, gli eventi di vendita possono essere rappresentati nello spazio tridimensionale definito da prodotto, negozio e data. Solitamente ogni dimensione è associata ad 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 delle associazioni molti a uno tra coppie di attributi dimensionali.

Il concetto di dimensione ha dato origine alla diffusissima metafora del *cubo* per la rappresentazione di dati multidimensionali, secondo cui gli eventi equivalgono a celle di un cubo i cui spigoli corrispondono alle dimensioni di analisi. Ogni cella è caratterizzata da un valore per ciascuna misura. Un data warehouse è quindi costituito da un insieme di cubi, uno per ogni fatto d'interesse dell'azienda.

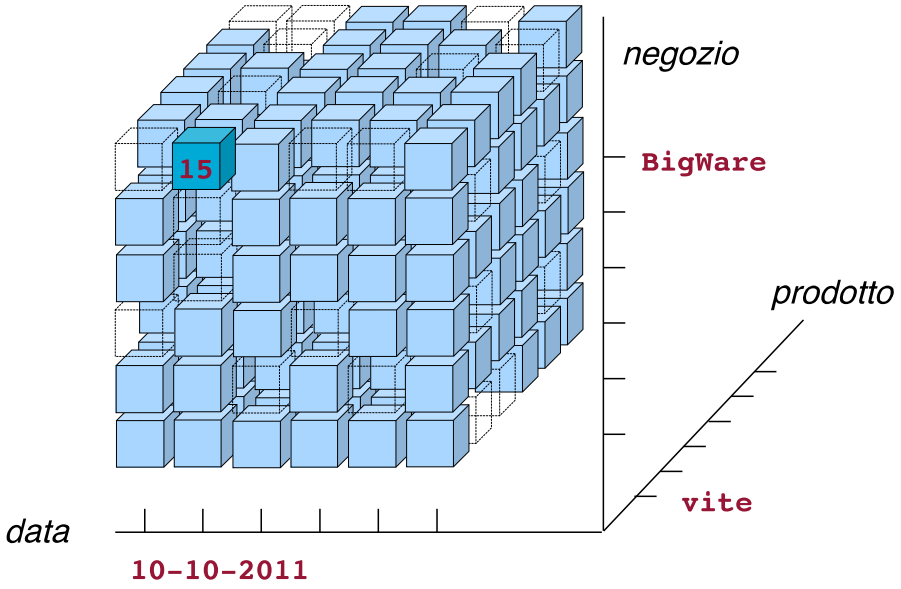

Figura 2.1: Esempio di cubo che rappresenta le vendite [16]

Nella Figura 2.1 è presentato un esempio di cubo multidimensionale, in cui il fatto d'interesse è la vendita in una catena di negozi, la misura considerata è la quantità di unità vendute e le dimensioni di analisi sono il prodotto, la data e il negozio. L'immagine mostra chiaramente che, considerando il prodotto "vite", la data "10-10-2011" e il negozio "BigWare", sono state vendute 15 unità. Le celle trasparenti evidenziano la sparsità del cubo, perché indicano assenza di informazioni. Ognuna di esse identifica infatti un prodotto, una data e un negozio che, combinati tra loro, non hanno fatto registrare nessuna entità venduta.

La struttura di un cubo multidimensionale può essere rappresentata con il DFM (*Dimensional Fact Model*), un modello concettuale di tipo grafico che può essere considerato come una specializzazione del modello multidimensionale per applicazioni di data warehousing [14]. Il DFM nasce per supportare efficacemente la progettazione di data warehouse, per creare un ambiente in cui le interrogazioni possano essere formulate in modo intuitivo e per rendere possibile la comunicazione tra progettista e utente finale. Quest'ultimo punto è particolarmente importante, perché il formalismo grafico proposto è semplice, intuitivo, comprensibile anche a figure non esperte di informatica e, quindi, adatto a contesti applicativi reali.

La rappresentazione concettuale generata dal DFM è composta da una serie di "schemi di fatto", che modellano gli elementi fondamentali di un cubo multidimensionale. In particolare ogni schema mostra in posizione centrale il fatto, accompagnato da tutte le sue misure, a cui sono collegati gli attributi che costituiscono le radici delle varie gerarchie dimensionali. Un collegamento tra due attributi dimensionali indica la presenza di una dipendenza funzionale. Nella Figura 2.2 è riportato il DFM che modella il cubo delle vendite visto in precedenza.

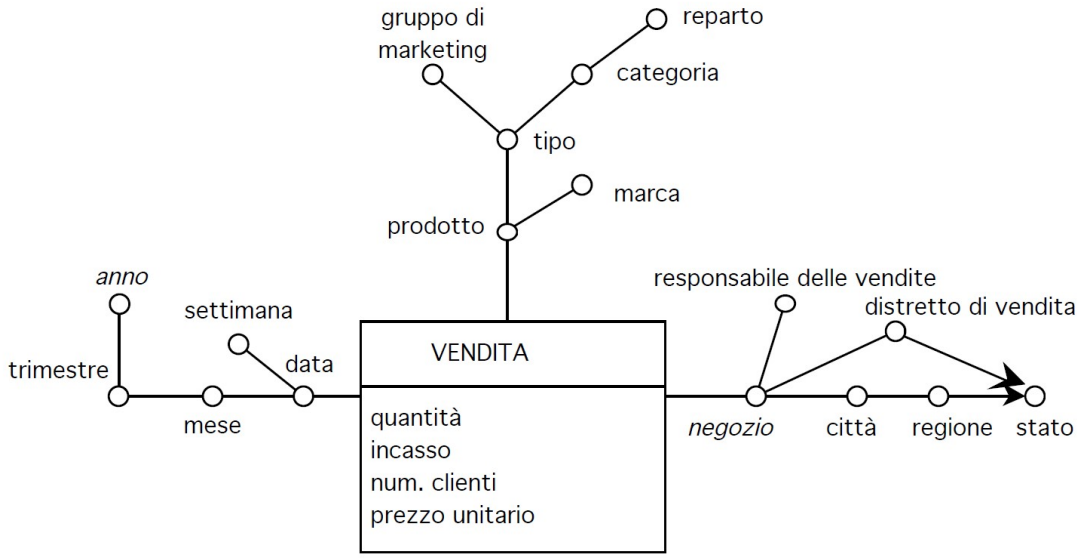

Figura 2.2: Esempio di DFM che modella le vendite [16]

Nella progettazione di data warehouse, mentre il modello concettuale universalmente adottato è quello multidimensionale, dal punto di vista logico è possibile rappresentare i dati con modelli differenti: ROLAP, MOLAP e HOLAP [16].

Attualmente i sistemi più diffusi sono quelli ROLAP, ossia *Relational On-Line Analytical Processing*, in cui viene usato il modello relazionale per la memorizzazione di dati multidimensionali. Si tratta di una scelta apparentemente forzata, ma giustificata dal fatto che il modello relazionale costituisce uno standard universalmente riconosciuto nel mondo dei database e dall'evoluzione subita dai DBMS relazionali negli ultimi anni. Essi sono infatti diventati strumenti sempre più raffinati e ottimizzati, al contrario di altri sistemi indubbiamente meno maturi.

Nei sistemi ROLAP la memorizzazione dei dati è basata sul cosiddetto *schema a stella* e sulle sue varianti. Esso è costituito da una serie di DT (*dimension table*), caratterizzate da una chiave primaria e da attributi che descrivono le dimensioni a diversi livelli di aggregazione. Per ogni fatto è presente una FT (*fact table*), che importa le chiavi delle varie DT e include un attributo per ogni misura.

2.1.2 Interrogazioni OLAP

Le interrogazioni OLAP costituiscono la principale modalità di fruizione delle informazioni contenute nei data warehouse. A differenza degli strumenti di reportistica, che consentono di accedere periodicamente a informazioni strutturate in modo quasi invariabile, le interrogazioni OLAP permettono agli utenti di esplorare interattivamente i dati sulla base del modello multidimensionale. Si tratta di una modalità di fruizione adatta ad utenti con necessità di analisi non identificabili a priori, che possono formulare attivamente una serie di interrogazioni [6].

Il carattere estemporaneo delle interazioni previste in un contesto OLAP, l'approfondita conoscenza dei dati richiesta, la complessità delle interrogazioni formulabili e l'orientamento verso utenti non esperti di informatica rendono cruciale il ruolo dello strumento usato, la cui interfaccia deve necessariamente presentare caratteristiche quali flessibilità, facilità d'uso ed efficacia.

A differenza delle classiche elaborazioni transazionali, nel contesto OLAP si parla di vere e proprie *sessioni*. Esse consistono in un percorso di navigazione che riflette il procedimento di analisi di uno o più fatti d'interesse, sotto diversi aspetti e a diversi livelli di dettaglio. Questo percorso si concretizza in una sequenza di interrogazioni che spesso non sono espresse direttamente, ma come differenza rispetto alla precedente. Ad ogni "passo" della sessione si applica un operatore in grado di trasformare l'ultima interrogazione in una nuova. Nel paradigma OLAP sono disponibili diversi operatori che consentono, ad esempio, di aumentare o ridurre il livello di aggregazione dei dati eliminando o introducendo livelli di dettaglio in una delle gerarchie (roll-up, drill-down), di visualizzare solo una porzione del cubo fissando un valore o un range di valori in una delle dimensioni (slicing, selezione) oppure di collegare cubi dimensionali correlati (drill-across).

Il risultato delle interrogazioni è multidimensionale e, date le scarse capacità umane di ragionare in uno spazio caratterizzato da un numero di dimensioni superiore a 3, gli strumenti OLAP solitamente usano il formato tabulare per rappresentare i dati. Per mantenere il focus sulle dimensioni, queste sono evidenziate con intestazioni multiple, colori o altro. Nella Figura 2.3 è riportato il risultato di una query sul cubo delle vendite visto in precedenza, in cui si calcola l'incasso in dollari al livello di aggregazione definito da regione del cliente e trimestre.

Metrics	Dollar Sales					
Customer Region / Quarter	North-East	Mid-Atlantic	South-East	Central	South	North-West
Q1 1997	$ 1.526	$ 1.249	$ 978	$ 1.885	$ 3.650	$ 4.855
Q2 1997	$ 2.979	$ 1.415	$ 2.664	$ 1.582	$ 1.130	$ 1.906
Q3 1997	$ 3.016	$ 1.772	$ 5.076	$ 2.050	$ 1.443	$ 2.311
Q4 1997	$ 2.711	$ 5.030	$ 2.025	$ 1.961	$ 3.601	$ 2.112
Q1 1998	$ 5.288	$ 3.399	$ 4.935	$ 9.174	$ 3.601	$ 9.484
Q2 1998	$ 1.773	$ 3.658	$ 1.862	$ 1.213	$ 1.115	$ 4.635
Q3 1998	$ 1.818	$ 5.402	$ 1.709	$ 2.485	$ 1.704	$ 3.632
Q4 1998	$ 2.051	$ 2.968	$ 4.664	$ 5.917	$ 1.775	$ 3.162

Figura 2.3: Esempio di risultato di una query OLAP [16]

Nelle sessioni di analisi OLAP, la classe di interrogazioni più comune è rappresentata dalle query GPSJ (*Generalized Projection, Selection, Join*) [18]. Il loro nome deriva dalle operazioni che un'interrogazione di questo tipo esegue sulle tabelle relazionali presenti in uno schema a stella. In particolare, una query GPSJ corrisponde ad una proiezione generalizzata sul risultato di una selezione, a sua volta fatta sul join tra una o più tabelle (ossia fact table e dimension table) [15]. Nell'algebra relazionale una query GPSJ su uno schema a stella può essere definita dalla formula seguente, in cui Pred costituisce una congiunzione di predicati sugli attributi delle DT, G è un insieme di attributi delle DT ed M è un insieme di misure della FT accompagnate dai rispettivi operatori di aggregazione.

$$q = \pi_{G,M}(\sigma_{Pred}(FT \bowtie DT_1 \bowtie ... \bowtie DT_n))$$

Gli operatori di selezione e join non richiedono un approfondimento, perché universalmente noti nell'ambito dei database, ma non si può dire lo stesso delle proiezioni generalizzate. Esse costituiscono un'estensione delle proiezioni tradizionali, eliminano i duplicati replicando il funzionamento della clausola SQL distinct e sono in grado di catturare i concetti di aggregazione e group by [18].

Le interrogazioni GPSJ prevedono una semantica in grado di impedire all'utente di ottenere risultati ambigui, che possono portare ad un'interpretazione errata nel contesto analitico. In particolare, le query GPSJ sono costituite da:

- MC (*measure clause*), ossia l'insieme delle misure il cui valore deve essere restituito dall'interrogazione. Esse devono essere specificate e sono accompagnate da operatori di aggregazione, che definiscono come aggregare le misure presenti nel risultato tenendo conto anche delle loro caratteristiche (come, ad esempio, la non additività di una misura su una o più dimensioni).

- GBC (*group by set*), che corrisponde ad un sottoinsieme dei livelli nelle gerarchie dimensionali del cubo e definisce la granularità dell'aggregazione. Il group by set può anche non comprendere nessun attributo e, in tal caso, i dati vengono mostrati al massimo livello di dettaglio.

- SC (*selection clause*), composta da un insieme di clausole booleane applicate al valore degli attributi. Come il group by set, anch'essa può essere omessa.

2.2 Sistemi conversazionali

Negli ultimi anni, l'applicazione di algoritmi e tecniche di intelligenza artificiale ha portato grandi benefici nei domini applicativi più vari. La mole di dati sfruttata da questi strumenti è cresciuta costantemente, soprattutto con l'avvento delle tecnologie *Big Data*, tanto che è stato riconosciuto come il numero di esperti informatici in grado di usare strumenti di intelligenza artificiale non possa scalare in modo tale da coprire tutte le richieste del mercato [10]. Questa constatazione sottolinea l'importanza della democratizzazione dell'accesso, che renderebbe possibile un uso "pervasivo" degli strumenti finalizzati ad estrarre valore dai dati.

In questo contesto un ruolo centrale è quello ricoperto dai sistemi conversazionali, che consentono agli utenti di interagire usando il linguaggio naturale. Recentemente le aziende hanno investito molto in questo ambito, sia in termini di ricerca sia in termini di sviluppo, e sono stati infatti prodotti strumenti oramai diventati di uso comune sia in ambienti privati sia in ambito aziendale [2].

I principali esempi di successo delle ricerche in ambito conversazionale sono assistenti vocali e chatbot. Per quanto riguarda i primi, si tratta di agenti software in grado di interpretare il linguaggio umano e di rispondere con un output in formato vocale. Essi possono essere sia integrati in altri dispositivi, quali smartphone o computer, sia indipendenti. Gli assistenti vocali più diffusi attualmente sono quelli sviluppati dalle più grandi aziende informatiche, ossia Apple Siri, Google Assistant, Amazon Alexa e Microsoft Cortana [23]. I chatbot sono invece software in grado di eseguire task in modo automatico e vengono utilizzati in piattaforme di messaggistica. Il loro obiettivo è quello di simulare conversazioni umane, permettendo la comunicazione con comandi vocali, chat testuali o interfacce grafiche. Spesso, i chatbot usano l'intelligenza artificiale per migliorare le proprie conoscenze sfruttando le interazioni con gli utenti. I più diffusi si appoggiano a servizi di messaggistica istantanea quali Telegram, WhatsApp e Messenger [8].

Il problema da affrontare nello sviluppo dei sistemi conversazionali, ossia la realizzazione di un'interazione basata sul linguaggio naturale, è intrinsecamente complesso e può essere suddiviso in due sottoparti complementari tra loro:

- La costruzione delle *Natural Language Interface* (NLI), in grado di tradurre un comando espresso in linguaggio naturale in un'operazione da svolgere su un sistema informatico. Facendo riferimento all'area applicativa dei database, questi sistemi consentono ad utenti non esperti di esplicitare query complesse senza conoscere il linguaggio SQL [1]. Le ricerche nell'ambito del *query answering* sono state le più significative per lo sviluppo delle NLI.

- La presentazione dei risultati ottenuti in linguaggio naturale, una volta eseguita sul sistema informatico l'operazione richiesta dall'utente. Questo secondo "sotto-problema", meno affrontato in letteratura rispetto al precedente, consente di creare una soluzione conversazionale end-to-end: sia l'input

dato dall'utente sia l'output generato dal sistema sono espressi in linguaggio naturale. A seconda della tipologia di strumento conversazionale realizzato, potrebbe essere emesso in formato vocale il risultato tradotto (assistenti vocali) o potrebbe essere visualizzato il suo contenuto (chatbot).

In base al sistema informatico interrogato, la presentazione dei risultati in linguaggio naturale può essere più o meno complessa. Nel caso specifico dell'interrogazione di database, l'output deve essere in grado di riassumere risultati abitualmente presentati in formato tabulare. In questo scenario è stato definito il problema della *data vocalization*, che nasce per coprire gli scenari applicativi in cui i metodi di *data visualization* non sono efficaci.

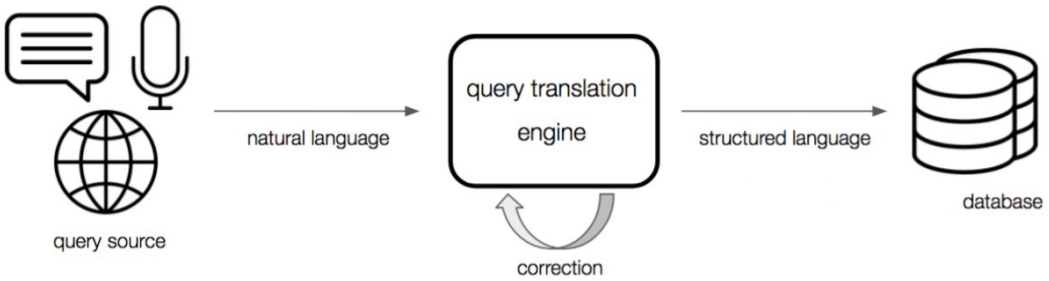

Figura 2.4: Funzionamento di una NLI nell'ambito dei database [42]

2.2.1 Query answering

Con il termine *query answering*, si fa riferimento ad una disciplina informatica che ha come obiettivo la realizzazione di sistemi capaci di rispondere automaticamente a domande poste in linguaggio naturale [43]. Essi possono fare affidamento su due metodologie differenti per rispondere alle interrogazioni, ossia l'approccio *information retrieval based* e l'approccio *knowledge based* [19].

L'information retrieval (IR) è l'insieme delle tecniche, dei sistemi e dei servizi che si occupano della ricerca, della manipolazione e della rappresentazione di dati digitali, espressi in linguaggio naturale e archiviati in vaste raccolte di documenti. L'approccio di query answering che sfrutta l'information retrieval prevede quindi

che, data una query formulata dall'utente, essa venga risolta cercando gli elementi più rilevanti all'interno di un insieme di documenti non strutturati. In particolare, dalla richiesta dell'utente si estrae un'interrogazione costituita dalle *keyword* da dare in input al sistema di IR per cercare documenti rilevanti. La risposta è quindi ottenuta raccogliendo brevi porzioni di testo dal web o da altre raccolte.

Alternativamente i sistemi di query answering possono sfruttare le cosiddette knowledge base, in cui le informazioni sono memorizzate in forma strutturata. Si può trattare sia di database completi sia di banche dati più semplici, come ad esempio database composti da semplici triple RDF (*Resource Description Framework*, linguaggio che prevede la memorizzazione di concetti sotto forma di preposizioni che legano un soggetto, un oggetto e un predicato). In questi strumenti la richiesta è trasformata in una rappresentazione logica, a sua volta utilizzata per recuperare i dati dalla sorgente. Il dominio del discorso è noto e limitato e la meta-conoscenza, ossia i nomi usati nello schema del database, può essere sfruttata per ridurre lo spazio di ricerca e interpretare in modo migliore la richiesta [41].

Nello scenario applicativo di riferimento, in cui l'obiettivo è la creazione di un framework conversazionale per la risoluzione di query OLAP, l'approccio di maggiore interesse al query answering è quello che si serve di knowledge base contenenti dati strutturati. Gli strumenti di questo tipo sono indicati dall'acronimo NLIDB (Natural Language Interfaces to Databases) [49] ed offrono i seguenti vantaggi:

- Non richiedono all'utente la conoscenza dei linguaggi specifici, come SQL, con cui vengono interrogati i database.
- Sono semplici da utilizzare. Questo punto e il precedente sono molto importanti nell'ottica della democratizzazione dell'accesso, perché consentono la fruizione dei dati anche ad utenti non esperti di informatica.
- Richiedono al sistema la conoscenza del dominio del discorso ma non quella della struttura esatta dei dati memorizzati, ovvero i nomi delle tabelle, le relazioni tra esse e i nomi dei loro campi.
- Non necessitano di una fase iniziale di addestramento.

Al contempo, questi sistemi presentano alcuni limiti:

- Lavorano con un insieme limitato di linguaggi naturali.
- Non sempre garantiscono la copertura linguistica.
- Possono richiedere una configurazione iniziale impegnativa.
- Possono essere fatte assunzioni errate dall'utente sul loro funzionamento.

Gli strumenti NLIDB sono da anni oggetto di ricerca, tanto che in letteratura esistono già diverse soluzioni valide [1], e le principali tecnologie sfruttate da questi sistemi sono *grammatiche formali*, *natural language processing* e *ontologie*.

Le grammatiche formali sono usate per descrivere linguaggi tramite una serie di regole, che indicano come modellare le stringhe che vi appartengono a partire da un alfabeto. Nei sistemi di query answering le grammatiche più diffuse sono quelle analitiche, che fungono da parser e verificano se una stringa data in input appartiene al linguaggio definito dalla grammatica. Per quanto riguarda il natural language processing (NLP), si tratta della disciplina che si pone l'obiettivo di analizzare, rappresentare e quindi "comprendere" il linguaggio naturale. Gli strumenti e le tecniche di NLP, come ad esempio pulizia del testo e ricerca di sinonimi, sono fondamentali nello sviluppo di sistemi NLIDB. Le ontologie sono invece definite come un insieme di primitive (classi, attributi e relazioni) con cui modellare un dominio e, nei sistemi di query answering, sono usate come base di conoscenza per descrivere le entità semantiche del dominio e le relazioni tra esse [4, 49].

Il framework conversazionale COOL, che si vuole estendere in questo lavoro di tesi, può essere classificato come sistema NLIDB. Esso è infatti in grado di comprendere le richieste dell'utente espresse in linguaggio naturale, di tradurle efficacemente in query GPSJ esplicitate in linguaggio SQL, di eseguirle su data warehouse e di mostrare il risultato. Le peculiarità dell'approccio usato da questo framework sono l'automatizzazione, poiché vengono sfruttati i metadati per semplificare le interazioni uomo-macchina, e la capacità di formalizzare sessioni OLAP quanto più possibile simili a conversazioni umane, consentita dal fatto che il sistema mantiene in memoria le interrogazioni precedenti [11, 12].

2.2.2 Data vocalization

Il problema della presentazione dei dati agli utenti è sempre stato centrale per le ricerche nell'ambito dei database, che si sono però concentrate quasi completamente su interfacce di tipo visuale. Nello sviluppo di soluzioni software sono infatti usate frequentemente delle tecniche di *data visualization*, disciplina che comprende un insieme di metodologie e strumenti utili a rappresentare graficamente i dati e ad esplorarli in maniera interattiva [7]. Essi sono particolarmente efficaci nei sistemi di supporto alle decisioni perché permettono, a chi li osserva, di interpretare con facilità i dati e di cogliere rapidamente le informazioni più significative.

Tuttavia, esistono scenari applicativi in cui un output visuale non costituisce una soluzione adatta. Si consideri, ad esempio, un data scientist ipovedente che ha intenzione di eseguire un'analisi su un determinato dataset. Egli non sarebbe in grado di interagire con un classico sistema che restituisce un output visuale, ma potrebbe ottenere le informazioni di cui ha bisogno interrogando uno strumento che prevede un output vocale (come un assistente smart). In particolare una sequenza di query, più o meno lunga a seconda del contenuto informativo desiderato, consentirebbe al data scientist di esplorare i vari aspetti dei dati [45].

I benefici portati da soluzioni in grado di presentare vocalmente i dati non sono però limitati all'accessibilità, ossia alla capacità dei sistemi informatici di erogare servizi e fornire informazioni utilizzabili anche da coloro che, a causa di disabilità, necessitano di tecnologie aggiuntive o configurazioni particolari.

Al giorno d'oggi la comunicazione tra utenti e computer si sta infatti spostando sempre più verso interfacce conversazionali, come dimostrato dalla diffusione di strumenti oramai comuni quali gli assistenti vocali [23]. Analizzando il comportamento di un utente medio, i sistemi che presentano visualmente i risultati potrebbero essere preferiti per analisi di ampio raggio, poiché offrono informazioni più dettagliate. Al contrario, per problemi specifici, un utente medio potrebbe scegliere un'interfaccia vocale, più semplice e rapida da utilizzare.

A valle di queste osservazioni è stato formalizzato il problema della *data vocalization*, che è complementare alla data visualization e che nasce per presentare i dati tramite un output vocale e nel modo più efficiente possibile [48].

Prima di analizzare nel dettaglio la data vocalization e le sue peculiarità, è necessario puntualizzare che essa si focalizza sulla vocalizzazione dell'output e va perciò distinta dalla cosiddetta *data sonification* [20]. Quest'ultima si occupa infatti della trasformazione di dati in linguaggio naturale, ma prevede che il testo prodotto venga restituito in formato visuale. Facendo riferimento agli scenari applicativi, le tecniche di data vocalization sono tipicamente usate negli assistenti vocali mentre quelle di data sonification vengono sfruttate dai chatbot.

Le maggiori differenze tra testi in formato vocale e visuale sono le seguenti [36]:

- Mentre la velocità di lettura di un testo presentato in forma scritta è stabilita dall'utente, che può procedere lentamente per comprendere il significato, in caso di output vocale il controllo è completamente affidato al sistema.

- Un testo emesso vocalmente può essere ascoltato soltanto una volta dall'utente, mentre un output visuale può essere riletto una o più volte.

- All'interno di un testo riportato in forma scritta l'utente può selezionare le parti più significative, ma questo non è possibile in caso di vocalizzazione.

Gli strumenti di data vocalization sono però più richiesti rispetto a quelli di data sonification, sia perché coprono un maggior numero di scenari applicativi (come quelli visti in precedenza) sia perché sono preferiti dagli utenti. Quest'ultima affermazione, non oggettiva come la prima, è supportata da uno studio di usabilità eseguito all'interno di un lavoro di ricerca che approfondisce la data vocalization. In questa analisi, i tre quarti dei possibili utenti hanno infatti espresso una preferenza per la vocalizzazione, il 23% si è dichiarato imparziale e soltanto il 2% ha scelto i sistemi che restituiscono output testuale in formato visuale [46].

La data sonification può essere vista come un caso particolare di data visualization, in cui l'output è costituito da un testo in forma scritta. Nella Figura 2.5, che segue, sono confrontati i principali aspetti (controllo, persistenza e trasmissione) delle tecniche di data visualization e data vocalization. Le caratteristiche sottolineate sono coerenti con le differenze viste in precedenza [22, 45].

Criterion	Visualization	Vocalization
Delivery	*One-Shot*	*Gradual*
Persistency	*Durable*	*Fleeting*
Control	*User*	*System*

Figura 2.5: Confrontro tra data visualization e data vocalization [45]

Controllo

L'output visuale offre maggiore controllo all'utente, che può scegliere quale parte (del testo, del grafico, ...) studiare. In caso di output di grandi dimensioni, è possibile identificare le parti più rilevanti in modo da concentrarsi su esse. Al contempo, le parti più complesse possono essere valutate ripetutamente dall'utente. Tutto ciò non è possibile in uno scenario di visualizzazione, perché il sistema controlla sia le informazioni trasmesse sia la modalità di comunicazione e l'utente si affida ad esso per selezionare le parti più significative dell'output.

Persistenza

L'output visuale è duraturo, in quanto rimane a disposizione dell'utente e gli consente di studiarlo a lungo. Al contrario l'output vocale è provvisorio, perché ogni sua parte è rivelata soltanto per un breve istante e l'utente deve memorizzarla (almeno parzialmente) per essere in grado di sfruttarla in modo efficace.

La transitorietà dell'output vocale costituisce un limite, che è evidente se si considera la scarsa capacità umana di memoria a breve termine. Il carico cognitivo generato dalla frase emessa vocalmente non deve eccedere la capacità mnemonica dell'utente medio. Quest'ultima è espressa in termini di *memory span*, termine con cui si indica la più lunga lista di concetti che una persona, subito dopo l'acquisizione, è in grado di ricordare nell'ordine corretto [32].

Trasmissione

Per quanto riguarda la trasmissione, la principale differenza è costituita dal fatto che l'output visuale è emesso in un'unica soluzione mentre quello vocale viene rivelato gradualmente. L'utente quindi, in un dato istante, non ha a disposizione informazioni diverse da quelle contenute nella frase corrente.

Dato che gli strumenti di data vocalization sono risultato di ricerche nell'ambito dei sistemi conversazionali, è importante che garantiscano una comunicazione interattiva con l'utente. Le risposte devono quindi essere generate con un ritardo quasi impercettibile, inferiore alla soglia (fissata a 500 ms) che consente ad un'analisi dati interattiva di essere considerata conveniente [33, 40].

Le peculiarità della data vocalization definiscono vincoli specifici che devono essere rispettati. In particolare, formalizzando e riassumendo le osservazioni fatte, in un sistema che restituisce un output vocale sono previsti i seguenti requisiti:

- Il risultato deve essere restituito rapidamente, rispettando la soglia prevista per una comunicazione interattiva.
- L'output deve avere una struttura semplice, in grado di limitare il carico cognitivo generato sull'utente in ascolto.
- Il risultato deve avere una lunghezza ridotta, in modo da essere efficace nonostante la scarsa memoria a breve termine degli utenti.
- L'output deve comunicare il massimo contenuto informativo possibile.

La data vocalization è quindi definita come un problema di ottimizzazione, il cui obiettivo è quello di massimizzare la quantità di informazioni trasmesse all'utente rispettando i vincoli di tempo e complessità di linguaggio [48].

Nell'ambito dell'accesso a database attraverso interfacce vocali, i primi contributi, come ad esempio il sistema EchoQuery [30], si sono focalizzati sulla comprensione dell'input dell'utente e prevedono una semplice lettura dell'output. Essi consentono di ottenere risultati soddisfacenti in caso di dataset di dimensione molto ridotta, ma non permettono di scalare all'aumentare della mole di dati. Inoltre,

risultati troppo complessi o troppo lunghi non consentono di rispettare i requisiti formalizzati per la data vocalization. Partendo da queste premesse sono stati presentati i primi sistemi di data vocalization, che si basano su alcune intuizioni progettuali direttamente correlate alle caratteristiche dell'output vocale [45]:

- Il risultato espresso vocalmente deve focalizzarsi su tendenze di alto livello e il sistema deve selezionare attentamente le informazioni da trasmettere. Infatti, rappresentare i dati ad un elevato livello di astrazione permette al sistema di generare risposte non eccessivamente complesse.
- Generare il risultato completo, caratterizzato da alta precisione, è superfluo perché questo non può essere emesso vocalmente.
- Dato che l'output viene restituito gradualmente, è possibile sovrapporre una computazione eseguita in background con l'emissione incrementale del risultato. La scelta di un approccio concorrente offre un vantaggio significativo, ossia un tempo maggiore per calcolare l'output.

Gli approcci alla data vocalization presenti in letteratura consentono rispettivamente di restituire, tramite output vocale, dati relazionali [48], serie temporali [46] e risultati di query OLAP [47]. Trattandosi di casistiche completamente differenti tra loro, ogni approccio è pensato in maniera specifica per il paradigma considerato ma tiene conto dei requisiti definiti per la data vocalization. Nel contesto identificato dall'obiettivo della tesi, l'unico approccio applicabile è quello finalizzato alla vocalizzazione dei risultati multidimensionali ottenuti con query OLAP. Si è quindi scelto di approfondirlo, in modo da comprenderne il funzionamento, implementarlo e integrarlo nel framework conversazionale COOL [12].

Capitolo 3

Vocalizzazione di risultati multidimensionali

L'approccio scelto per la vocalizzazione di risultati OLAP è definito *olistico*, perché combina la valutazione delle query e la trasformazione dell'output in formato vocale per minimizzare gli overhead computazionali e massimizzare la qualità del risultato. Prima di procedere all'implementazione esso è stato studiato in modo approfondito, al fine di comprendere nel dettaglio il suo funzionamento.

Il terzo capitolo presenta i vari aspetti dell'approccio e fa riferimento a quanto proposto da uno specifico lavoro di ricerca [47]. In particolare, nella Sezione 3.1 viene formalizzato il problema della data vocalization finalizzato alla traduzione di risultati OLAP. La Sezione 3.2 è dedicata alla modellazione dell'output vocale, a cui si fa riferimento col termine *discorso*, ed evidenzia sia gli aspetti sintattici sia quelli semantici. Nella Sezione 3.3 è invece trattato in modo specifico l'algoritmo di valutazione e vocalizzazione, responsabile della generazione del risultato.

3.1 Formalizzazione del problema

L'approccio olistico presentato in questo capitolo nasce con l'obiettivo di valutare e vocalizzare i risultati di query GPSJ e, quindi, definite tramite measure clause, group by set e selection clause. Il numero di interrogazioni OLAP che possono essere risolte è però limitato da due vincoli che queste devono rispettare, entrambi strettamente legati ai requisiti definiti per la vocalizzazione:

- Sono accettate unicamente le query che analizzano una sola misura, accompagnata da un operatore di aggregazione. Non si tratta di un vero e proprio limite dell'approccio, che potrebbe essere esteso per supportare anche queste interrogazioni, ma di una scelta fatta a livello concettuale. Dato che l'output vocale deve rispettare il requisito di semplicità, consentire interrogazioni più complesse sarebbe controproducente ai fini della generazione del risultato.

- Mentre una generica query GPSJ supporta gli operatori di aggregazione *avg*, *sum*, *count*, *min* e *max*, l'approccio olistico non accetta interrogazioni che richiedono di calcolare il minimo o il massimo. Questo perché il metodo di valutazione si basa su un meccanismo di campionamento, che non consente di approssimare efficacemente le funzioni min e max [35].

Ad esempio, dato un cubo multidimensionale contenente informazioni relative ai dipendenti di una grande azienda, una query ammissibile è quella che richiede di calcolare il salario medio a metà carriera (measure clause) raggruppando i dati per regione di provenienza e salario iniziale del lavoratore (group by set).

L'approccio olistico prevede una modellazione delle interrogazioni più esplicita ma ugualmente espressiva. In particolare, una generica query q è identificata da una funzione di aggregazione $q.fct$, da una colonna di aggregazione $q.col$ e da un insieme di aggregati $q.aggs$. Le prime due corrispondono al contenuto della measure clause. Gli aggregati sono invece determinati combinando tutti i possibili valori degli attributi dimensionali presenti nel group by set, ad esclusione di quelli che non rispettano le condizioni esplicitate dalla selection clause.

Facendo riferimento alla query precedente, $q.fct$ corrisponde alla media, $q.col$ al salario a metà carriera e $q.aggs$ contiene un aggregato per ogni combinazione di regione e salario iniziale (ad esempio, Northeast e salario pari almeno a 50mila). Si può notare che gli aggregati corrispondono a "celle" del cubo multidimensionale restituito e sono caratterizzati da diverse condizioni di filtro sui dati. Esse equivalgono a congiunzioni di più condizioni atomiche, ognuna delle quali è in grado di limitare i possibili valori di un attributo. Le colonne considerate rappresentano le dimensioni e il loro dominio di valori è strutturato in gerarchie, caratterizzate da uno o più livelli che permettono di porre restrizioni a varie granularità.

Nell'ambito della stessa query aggregati diversi sono associati a condizioni mutualmente esclusive, poiché ogni record del dataset è rilevante al più per un aggregato. In tal caso si dice che la tupla è nel suo *scope*, dato che soddisfa la condizione collegata. Il risultato di una query OLAP è quindi in grado di assegnare, ad ogni elemento di $q.aggs$, il valore aggregato che riassume i record del suo scope.

Nel contesto specifico della data vocalization, una soluzione ottimale è rappresentata da un output vocale in grado di descrivere il risultato quanto meglio possibile e di rispettare in vincoli di tempo e complessità [47]. In particolare, l'approccio olistico genera uno spazio di ricerca contenente i possibili discorsi (candidati) e si pone l'obiettivo di trovare quello in grado di massimizzare il contenuto informativo trasmesso. Per misurarlo, è stato svolto uno studio dell'utente che ha permesso di associare ad ogni output vocale una distribuzione di probabilità in grado di modellare l'opinione che l'ascoltatore ha del risultato della query.

Indicando con $\beta(a, t)$ la percezione dell'utente in merito ad un aggregato a dopo aver ascoltato il discorso t, modellata come distribuzione di probabilità, la qualità dell'output vocale t corrisponde alla media delle probabilità che gli utenti assegnano ai reali risultati della query dopo aver ascoltato il discorso t.

$$quality(t) = \sum_{a \in q.aggs} P(a(D)|\beta(a,t))/|q.aggs|$$

È quindi possibile definire la vocalizzazione di risultati di interrogazioni OLAP come un problema di ottimizzazione in cui, data una query e uno spazio di ricerca T contenente una serie di discorsi candidati, l'obiettivo è massimizzare il valore della metrica di qualità appena introdotta e trovare l'output vocale $t \in T$ tale che:

$$\forall t' \in T : quality(t) \geq quality(t')$$

3.2 Modellazione dell'output vocale

Nelle tecniche di data vocalization, un aspetto fondamentale è la definizione della struttura e delle caratteristiche dell'output vocale. Questa sezione è dedicata alla presentazione della sintassi usata dall'approccio per la generazione dei discorsi (ed esplicitata con una grammatica) e della semantica ad essa associata.

La sintassi dell'output vocale deve essere in grado di integrare sia la prospettiva dell'utente, rendendo un discorso comprensibile, sia quella del sistema, facilitando la generazione del risultato in formato vocale. La formalizzazione del problema ha mostrato la necessità di modellare il modo in cui il discorso influenza la percezione che l'utente ha dei dati. Questa esigenza è soddisfatta dalla semantica, che definisce come ragiona un ascoltatore sottoposto ad un determinato output.

3.2.1 Sintassi del discorso

Per essere adatta a rappresentare un output espresso in formato vocale, una grammatica dovrebbe possedere determinate caratteristiche:

- **Concisione**
  L'output vocale deve essere molto conciso per rispettare i vincoli di memoria a breve termine degli utenti [32]. I risultati delle query OLAP potrebbero infatti fare riferimento ad un numero molto elevato di aggregati, troppi per essere descritti singolarmente dal discorso. La grammatica destinata ad output vocali deve quindi essere caratterizzata da mezzi di astrazione adatti a riassumere informazioni relative a molti aggregati.

- **Precisione**
  In caso di risultati di piccole dimensioni o di vincoli di tempo particolarmente rilassati, un sistema di vocalizzazione deve essere in grado di trarne vantaggio. La grammatica, dunque, necessita anche di mezzi di astrazione in grado di trasmettere dettagli dei risultati a grana fine.

- **Estensibilità**
  L'emissione graduale dell'output vocale può rappresentare un'opportunità, perché permette di sovrapporre una computazione eseguita in background con la restituzione del risultato, e per sfruttarla la grammatica deve essere estendibile. In tal caso essa consente infatti di perfezionare il risultato in modo incrementale, aggiungendo nuove frasi all'output vocale ed evitando di creare contraddizioni con le precedenti affermazioni.

- **Semplicità**
  La complessità della grammatica del discorso influisce sullo spazio di ricerca per la vocalizzazione dei risultati. Una dimensione eccessiva di quest'ultimo porta ad un notevole overhead computazionale, inappropriato in uno scenario applicativo in cui è richiesta l'interattività. Di conseguenza, è necessario scegliere una grammatica semplice e limitare lo spazio di ricerca.

<Speech>::=<Preamble><Baseline><Refinement>*
<Preamble>::=Considering (<Pred>|(<Pred>)+ and <Pred>).
[Results are broken down by (<Level>|<Level>+ and <Level>).]
<Baseline>::=<Value> is the <Aggregate>.
<Refinement>::=Values <Change> for (<Pred>|(<Pred>)+ and <Pred>).
<Change>::=(increase|decrease) by <Quantifier>
<Pred>::=<Dimension Context> <Member>

Figura 3.1: Sintassi del discorso rappresentata in EBNF [47]

Nella Figura 3.1 è riportata interamente la grammatica definita dall'approccio olistico. Essa soddisfa le proprietà elencate e introduce le seguenti derivazioni:

- *Speech*, che combina *Preamble*, *Baseline* e un numero variabile di *Refinement*.
- *Preamble*, in cui è esplicitato il contenuto delle clausole di selezione (con i predicati rappresentati da *Pred*) e del group by set (grazie ai livelli *Level*).
- *Baseline*, che associa uno specifico valore *Value* ad un aggregato *Aggregate*.
- *Refinement*, in cui è indicata una variazione *Change* del valore aggregato nell'ambito dello scope definito da uno o più predicati *Pred*.
- *Pred*, che assegna ad un attributo dimensionale un membro della gerarchia.

Ad esempio, data la query precedente che richiede di calcolare il salario medio a metà carriera in relazione a regione e salario iniziale, un possibile discorso generato dalla grammatica è *"Si considerano i laureati provenienti da qualsiasi college e con un salario iniziale pari a qualsiasi cifra. I risultati sono raggruppati per regione*

e salario iniziale approssimato. 90mila è la media dei salari a metà carriera. Il valore aumenta del 5% per i laureati provenienti dal Northeast. Il valore aumenta del 20% per i laureati con un salario iniziale pari almeno a 50mila.".

Si può notare che ogni output vocale inizia con un *preambolo*, contenente il contesto del risultato descritto. La presenza di questo elemento fornisce un duplice vantaggio: aiuta gli utenti a tenere traccia della posizione nelle gerarchie dimensionali e offre al sistema più tempo per calcolare la parte restante del discorso. Questo perché il preambolo può essere generato direttamente a partire dalla query formulata dall'utente, e la sua creazione non richiede l'accesso ai dati.

Il preambolo è seguito da una *baseline*, a cui fanno riferimento tutte le affermazioni successive. Essa assegna alla misura considerata un valore aggregato quanto più possibile a grana grossa, poiché riassume tutti i risultati della query.

A sua volta la baseline può essere accompagnata da un numero arbitrario di *refinement*, che fanno riferimento a sottoinsiemi di dati e sono caratterizzati da uno o più predicati e da un indicatore del cambiamento. I predicati definiscono lo scope assegnando uno o più attributi dimensionali a valori specifici (membri della gerarchia), mentre l'indicatore descrive una variazione relativa nel valore aggregato atteso. Questo cambiamento è espresso sia in rapporto al valore della baseline sia in rapporto a quello dei refinement il cui scope include quello corrente.

La grammatica utilizzata permette di stabilire il livello di astrazione in modo flessibile, scegliendo il numero di predicati da inserire nei refinement, e sono quindi supportate descrizioni sia a grana grossa sia specifiche. Inoltre, è possibile estendere un discorso aggiungendo nuove informazioni. È importante sottolineare che questa proprietà deriva anche dalla natura relativa (e non assoluta) dei refinement, che garantisce la mancanza di contraddizioni con i loro predecessori.

L'esempio riportato mostra chiaramente che, per la generazione del discorso, sono usati sia i nomi dei membri delle gerarchie dimensionali (quali "North East" o la radice dell'albero "qualsiasi college"), sia quelli dei livelli delle gerarchie (ad esempio "regione"), sia i modelli di contesto delle dimensioni (come "laureati provenienti da"). I nomi dei livelli delle dimensioni e i loro modelli di contesto, visti nella grammatica, costituiscono quindi metadati necessari all'approccio olistico.

3.2.2 Semantica del discorso

Per scegliere il miglior output vocale, è necessario predire la reazione degli utenti all'ascolto dei discorsi candidati. Si tratta di un'operazione resa difficoltosa dai limiti della vocalizzazione, perché l'output deve essere conciso e non è in grado di coprire tutti i dettagli del risultato. L'obiettivo è quindi quello di capire come gli utenti, intuitivamente, concepiscano le informazioni mancanti.

È stata quindi modellata una distribuzione di probabilità che rappresenta la percezione dell'utente medio riguardo al risultato della query dopo aver ascoltato l'output vocale. Per poter mappare i discorsi nelle opinioni corrispondenti, è fondamentale fare delle assunzioni sul comportamento dell'utente medio.

Le ipotesi su cui si basa la semantica definita nell'approccio proposto derivano da assunzioni a cui si fa riferimento in precedenti attività di ricerca sul paradigma OLAP [31, 37] e da considerazioni di buonsenso. È stato inoltre svolto uno studio dell'utente finalizzato a verificare la loro presenza nel comportamento umano [47]. In particolare, le ipotesi di riferimento per la semantica sono le seguenti:

- **Simmetria**
  Data una media, l'utente assume che le deviazioni in entrambe le direzioni (valori minori e valori maggiori) siano equamente probabili.

- **Concentrazione**
  Dato un valore medio, l'utente considera le piccole variazioni più probabili di quelle di entità maggiore. La distribuzione di probabilità da essi immaginata è perciò unimodale, perché caratterizzata da un solo punto di massimo.

- **Composizione**
  Dati due refinement con scope sovrapposti, l'utente integra la composizione dei loro effetti nel proprio modello mentale.

- **Uniformità**
  L'utente sceglie il risultato più uniforme che è consistente con il discorso. In accordo con il MEP (*Maximum Entropy Principle*, usato per distribuzioni di dati parzialmente noti), il disordine si può misurare tramite l'entropia [17].

A valle delle considerazioni fatte, si può affermare che la distribuzione immaginata dall'utente in caso di risultati parzialmente noti è simmetrica e unimodale. Inoltre, come confermato dallo studio dell'utente, questa distribuzione è abbastanza vicina ad una normale avente deviazione standard proporzionale alla media [3]. Essa è coerente con tutte le osservazioni fatte e, essendo universalmente nota e facilmente calcolabile, è stata scelta come base per il modello mentale. Ciò non significa che non esistano altre distribuzioni in grado di approssimare efficacemente la percezione dell'utente ma, semplicemente, che la normale può farlo.

Formalmente, la semantica β di un discorso t è una funzione che assegna ad un aggregato a una distribuzione normale corrispondente alla percezione dell'utente.

$$\beta(a,t) = N(M(a,t),\sigma)$$

La media della distribuzione equivale al valore restituito dalla funzione $M(a,t)$. La deviazione standard σ è invece una costante ottenuta moltiplicando l'aggregazione di tutti i dati presenti nello scope della query, ossia il valore della baseline, per un fattore compreso tra 0 e 1, che rappresenta un parametro.

$M(a,t)$ è la funzione che calcola il valore medio della distribuzione normale e, se t è composto solamente da una baseline, per ogni aggregato a restituisce il valore contenuto in essa. Questo perché l'utente, come sostenuto da varie ricerche in ambito OLAP [37], tende ad immaginare distribuzioni uniformi.

Se invece il discorso t è formato da una baseline seguita da uno o più refinement, anch'essi influenzano il risultato restituito da $M(a,t)$. In particolare, dato un refinement r che segue un prefisso x e descrive una variazione relativa Δ (incremento o decremento), se l'aggregato a ricade all'interno dello scope definito dai predicati di r, esso viene applicato direttamente al risultato ottenuto in precedenza:

$$M(a,t) = M(a,x) * (1+\Delta)$$

Dato che la baseline è espressa in termini assoluti e i refinement sono caratterizzati da una natura relativa, questi ultimi sono anche in grado di modificare la percezione degli aggregati che non appartengono al loro scope. Intuitivamente, se un refinement aumenta il valore atteso per qualche aggregato, deve diminuire la stima

fatta per gli altri (e viceversa). Questa osservazione è coerente con la simmetria della distribuzione considerata. Quindi se r fa riferimento a m aggregati su n, per ogni aggregato a che non ricade nello scope di r, la media della distribuzione è:

$$M(a,t) = M(a,x) * (1 - m * \Delta/(n-m))$$

Si prenda in considerazione, ad esempio, l'output vocale *"Si considerano i laureati provenienti da qualsiasi college e con un salario iniziale pari a qualsiasi cifra. I risultati sono raggruppati per regione. 80mila è la media dei salari a metà carriera. Il valore aumenta del 50% per i laureati provenienti dal Northeast."* in cui il preambolo è seguito dalla baseline e da un unico refinement [47].

I dati sono raggruppati per regione e, dato che i possibili valori di questo attributo dimensionale sono Northeast, Midwest, West e South, il risultato comprende quattro aggregati. Scegliendo 0.5 come fattore moltiplicativo per calcolare la deviazione standard, si ha $\sigma = 40000$. La distribuzione normale ottenuta è quindi $N(120000, \sigma)$ per l'aggregato caratterizzato dalla regione Northeast e $N(66667, \sigma)$ per tutti gli altri. Il valore medio di quest'ultima distribuzione si calcola applicando alla stima precedente un decremento pari al 17% (50%/3). Questo garantisce che la media di tutti gli aggregati produca un risultato coerente con la baseline.

Aggregato	$M(a)$	$\beta(a)$
<Regione = Northeast>	120000	$N(120000, 40000)$
<Regione = Midwest>	66667	$N(66667, 40000)$
<Regione = West>	66667	$N(66667, 40000)$
<Regione = South>	66667	$N(66667, 40000)$

Figura 3.2: Esempio di semantica di un output vocale

3.3 Algoritmo di valutazione e vocalizzazione

L'approccio scelto per la vocalizzazione di risultati multidimensionali è definito olistico, perché si basa sulla combinazione di attività apparentemente separate quali valutazione delle query e trasformazione dell'output in formato vocale.

La valutazione di query OLAP su dataset di grandi dimensioni è computazionalmente onerosa. Al contempo, lo spazio di ricerca definito per l'output vocale è abbastanza ampio e il calcolo della qualità di un discorso (applicando il modello mentale dell'utente illustrato nella Sezione 3.2.2) è costoso. Malgrado queste difficoltà, l'obiettivo rimane coerente con quanto definito nei requisiti e le risposte devono essere generate con un ritardo quasi impercettibile (e inferiore alla soglia di interattività di 500 ms [33, 40]). Per farlo, vengono sfruttate le seguenti idee:

- **Campionamento**
  A causa dei vincoli di concisione, lo spazio di ricerca considerato contiene tendenzialmente output vocali a grana grossa e la trasmissione di risultati OLAP precisi risulta essere quasi sempre impossibile. Di conseguenza l'approccio, invece di valutare interamente la query, analizza piccoli campioni di dati e li utilizza per stimare la qualità dei vari discorsi candidati. Inoltre il modello di comportamento dell'utente non viene istanziato completamente, ma solo per aspetti del risultato OLAP selezionati casualmente.

- **Prioritizzazione**
  Lo spazio di ricerca contenente i discorsi candidati è ampio e, a causa dei vincoli temporali stringenti, non si può ottenere una stima di qualità precisa per ogni possibile output vocale. Perciò l'approccio olistico è iterativo e, ad ogni passo, sceglie un discorso e perfeziona la stima della sua qualità sulla base del contenuto informativo di un campione. La tecnica di prioritizzazione usata per stabilire l'ordine in cui vengono valutati i vari output vocali appartiene alla famiglia della MCTS (*Monte Carlo Tree Search*) [27].

- **Pipelining**
  Una peculiarità dell'output vocale è la sua trasmissione sequenziale. Essa è sfruttata dall'approccio olistico alternando l'attività combinata di valutazione della query e generazione del discorso all'emissione dell'output: mentre viene letta la frase corrente, in background è determinato il seguito migliore. In questo modo è possibile ottenere rapidamente affermazioni di alto livello e, durante la loro riproduzione, generare osservazioni più specifiche.

Algoritmo 1: Valutazione e vocalizzazione di una query

```
input: query q
// Genera ed emette il preambolo;
t ⟵ SG.preamble(q);
VO.start(t);
// Inizializza l'albero di ricerca;
root ⟵ ST.newNode(−, t);
ST.expand(q, root);
// Itera fino alla conclusione del discorso;
finished ⟵ false;
while ¬finished do
    // Itera finché non termina l'output vocale corrente;
    while VO.isPlaying() do
        ST.sample(q, root);
    end
    if root.isLeaf() then
        finished ⟵ true;
    else
        // Sceglie ed emette la prossima frase del discorso;
        root ⟵ arg max_{c∈root.children} c.reward/c.visits;
        VO.start(root.lastSentence);
    end
end
```

La metodologia proposta concretizza le tre idee che sono state riportate ed è descritta, ad alto livello, dall'Algoritmo 1. Esso utilizza varie funzioni ausiliarie, raggruppate in tre categorie a cui si fa riferimento con gli acronimi: speech generation (SG), voice output (VO) e search tree operations (ST). Dopo aver fissato le preferenze dell'utente espresse in termini di lunghezza del discorso, ossia numero di refinement, la procedura è in grado di accettare in input query aventi le caratteristiche formalizzate nella Sezione 3.1. Viene dunque generato, frase dopo frase, l'output vocale, mentre la computazione prosegue in background.

Inizialmente l'algoritmo genera il preambolo, riassumendo la query data in input, e avvia il suo output vocale chiamando la funzione $VO.start$. Questa è asincrona, perché ritorna non appena viene sottoposta una frase da emettere in formato vocale. Perciò, mentre viene letto il preambolo, l'algoritmo si occupa della generazione di un albero che contiene tutti i discorsi candidati e che rappresenta lo spazio di ricerca. Questa operazione è svolta dalle funzioni $ST.newNode$ e $ST.expand$, la cui implementazione è approfondita nella Sezione 3.3.1.

Dopo aver inizializzato l'albero di ricerca, l'algoritmo procede iterativamente fino a trovare l'ultima frase dell'output vocale (in accordo con il numero di refinement definito dalle preferenze dell'utente). Nella prima parte di ogni iterazione sono perfezionate le stime di qualità dei discorsi candidati, mentre l'output viene emesso. In particolare, si sfrutta la funzione $VO.isPlaying$ per determinare se la lettura della frase precedente è ancora in corso e il metodo $ST.sample$ per raffinare il calcolo della qualità dei candidati. Quest'ultimo è discusso dettagliatamente nel prosieguo e, ad un alto livello di astrazione, si può dire che ad ogni sua invocazione seleziona un output vocale interessante e verifica come questo è in grado di descrivere un campione di dati appartenente allo scope della query.

Una volta terminata l'emissione della frase corrente del discorso, la seconda parte dell'iterazione prevede che si decida se aggiungere o meno (sulla base del numero di refinement scelto) un'ulteriore frase all'output vocale. In caso affermativo, l'algoritmo seleziona il contenuto del figlio più promettente del nodo corrente. Ogni operazione di campionamento associa al nodo considerato un valore di *reward*, in grado di catturare l'efficacia con cui il discorso riesce a descrivere il campione usato. Il più promettente tra i figli presenti nell'albero è quello che massimizza la media dei valori di *reward* ottenuti, calcolata come rapporto tra la ricompensa totale raggiunta e il numero di visite (e quindi valutazioni) ricevute.

È importante notare che la stima della qualità dei discorsi candidati non viene re-inizializzata per ogni frase dell'output vocale ma, trattandosi dello stesso albero di ricerca, tutte le valutazioni fatte in precedenza rimangono disponibili e consentono di evitare l'esecuzione di attività computazionali ridondanti.

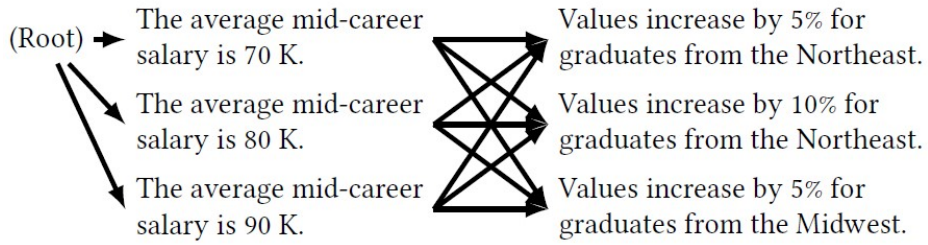

Figura 3.3: Esempio di albero di ricerca dei discorsi [47]

La Figura 3.3 rappresenta un semplice esempio di albero di ricerca contenente i discorsi candidati, in cui ogni nodo è associato ad una frase e ogni percorso corrisponde ad un possibile output vocale (quello ottenuto concatenando il contenuto di tutti i nodi attraversati). I livelli della struttura sono associati a diversi aspetti del dataset, ossia alle dimensioni utilizzate nel raggruppamento, e l'altezza dell'albero è limitata dal numero di refinement previsti. Nell'esempio riportato in figura l'altezza corrisponde all'estensione orizzontale e, in particolare, è uguale a due perché la baseline è seguita solamente da un refinement [47].

Si consideri ora una query che richiede di calcolare il salario medio dei dipendenti di un'azienda raggruppando i dati per regione e per salario iniziale. Innanzitutto, l'algoritmo calcola il preambolo *"Si considerano i laureati provenienti da qualsiasi college e con un salario iniziale pari a qualsiasi cifra. I risultati sono raggruppati per regione e salario iniziale approssimato."* e avvia il suo output vocale.

Viene quindi inizializzato l'albero di ricerca, mostrato nella Figura 3.3, ed è avviata l'attività di campionamento a partire dalla sua radice (*root*). Ad ogni nodo è associata una stima di qualità e le valutazioni sono progressivamente raffinate, confrontando i discorsi candidati con dei campioni. Supponendo che al termine dell'emissione del preambolo il nodo più promettente contenga la frase *"80mila è la media dei salari a metà carriera"*, questa viene emessa vocalmente.

La procedura è poi ripetuta scegliendo come radice il nodo selezionato. In questo caso, si assuma che le nuove iterazioni assegnino una maggiore *reward* al nodo contente la frase *"Il valore aumenta del 5% per i laureati provenienti dal Northeast"*. Essa è scelta come prossima parte dell'output vocale e inoltre l'algoritmo, avendo raggiunto una foglia dell'albero di ricerca, termina le proprie iterazioni.

3.3.1 Esplorazione dell'albero di ricerca

La generazione dell'albero di ricerca contenente i discorsi candidati costituisce un'operazione di pre-processing necessaria per poter applicare l'approccio e, nell'Algoritmo 1, è svolta dalle funzioni $ST.newNode$ e $ST.expand$.

In particolare, alla radice sono collegate le possibili baseline calcolate in seguito all'esecuzione di alcune statistiche sul dataset di riferimento. Ad ogni baseline vengono associati tutti i refinement ammissibili che, per mantenere alto il livello di astrazione, sono caratterizzati da un unico predicato. Dato l'insieme di attributi dimensionali corrispondente al group by set della query, ogni livello dell'albero è associato ad uno di essi (ad esclusione del primo, destinato alle baseline). A ciascun nodo del livello precedente sono infatti collegati, come figli, i nodi che rappresentano i refinement generati combinando tutti i possibili valori dell'attributo con i fattori, definiti a priori, che quantificano una variazione relativa.

È importante ricordare che i livelli dell'albero sono limitati dal numero di refinement scelto. Al contempo, quest'ultimo deve rimanere al di sotto di una certa soglia a causa delle scarse capacità mnemoniche degli utenti. Perciò la complessità dell'albero, il cui numero di nodi è nell'ordine di grandezza di $O(m^k)$ se si indica con m il massimo numero di figli di un nodo e con k il numero dei livelli dell'albero, è accettabile perché limitata da un valore di k non troppo elevato.

I nodi dell'albero generato contengono informazioni utili a descrivere il discorso associato, a catturare la struttura dello spazio di ricerca e ad applicare l'algoritmo di prioritizzazione. In particolare, in un nodo sono presenti i seguenti campi:

- *text*, contenente l'output vocale completo rappresentato dal nodo.
- *lastSentence*, che esplicita l'ultima frase del discorso a cui il nodo è associato.
- *children*, necessario perché mantiene i collegamenti a tutti i nodi figli e consente di capire se si tratta di una foglia ($children = \emptyset$).
- *reward*, corrispondente alla somma delle ricompense accumulate nelle operazioni di campionamento che seguono un percorso contenente il nodo.
- *visits*, che contiene il numero di visite ricevute nei vari campionamenti.

Algoritmo 2: Campionamento dell'albero di ricerca

input: query q
input: root node n
$path \longleftarrow \{n\}$;
// Itera fino a raggiungere una foglia dell'albero;
while $\neg n.isLeaf()$ **do**
  // Concede la priorità ai figli non visitati;
  $S \longleftarrow \{c : c \in n.children \ \& \ c.visits = 0\}$;
  if $S = \emptyset$ **then**
    // Massimizza la formula dell'algoritmo UCT;
    $S \longleftarrow \arg\max_{c \in n.children} \frac{c.reward}{c.visits} \sqrt{\frac{2 * log(n.visits)}{c.visits}}$;
  end
  // Sceglie il nodo che ottimizza il tradeoff exploration-exploitation
  $maxUctChild \longleftarrow randomElement \in S$;
  // Aggiunge il nodo più promettente al discorso;
  $path \longleftarrow path \cup \{maxUctChild\}$;
end
// Valuta il discorso contenuto nella foglia;
$r \longleftarrow evaluateSpeech(q, n.text)$;
// Aggiorna le statistiche nel percorso selezionato;
for $n \in path$ **do**
  $n.reward \longleftarrow n.reward + r$;
  $n.visits \longleftarrow n.visits + 1$;
end

Una volta generato lo spazio di ricerca, l'Algoritmo 1 inizia le proprie iterazioni e sono fondamentali le attività di campionamento eseguite da *ST.sample*. Queste permettono di stimare la qualità dei discorsi candidati, pur avendo a disposizione solo dei campioni appartenenti a una distribuzione di probabilità la cui media corrisponde al reale valore di qualità. Esso non può essere calcolato perché il risultato completo della query non è noto. Per far fronte a questa situazione, è necessario un algoritmo di ricerca in grado di gestire valori incerti.

Tra le tecniche che soddisfano questo requisito, alcune delle più popolari sono quelle appartenenti alla famiglia della MCTS [5]. Esse basano la risoluzione di problemi sulla valutazione di soluzioni derivate da un albero di ricerca e sull'esecuzione di una serie di operazioni di campionamento. Più precisamente l'approccio olistico usa una possibile variante dell'algoritmo UCT (*Upper Confidence Tree*), che è caratterizzato dalla propria strategia di prioritizzazione [27].

In particolare, UCT sfrutta i campi *reward* e *visits* per determinare i valori di confidenza da assegnare ai nodi e utilizza quanto ottenuto per prioritizzarli. La strategia usata dall'algoritmo bilancia i criteri di *exploration*, secondo cui devono essere raccolte informazioni sui nodi visitati raramente, e *exploitation*, che prevede il raffinamento delle stime per i nodi più promettenti. Il principio di exploration è motivato dal fatto che i valori di confidenza sono più attendibili se calcolati in base al confronto con più campioni e favorisce i nodi meno visitati. Il criterio di exploitation nasce invece dall'intuizione secondo cui, se nei campionamenti precedenti un nodo ha ricevuto una ricompensa maggiore rispetto ad un altro, è probabile che esso sia più interessante ai fini della rappresentazione del risultato.

L'Algoritmo 2 illustra ad alto livello un'operazione di campionamento eseguita sull'albero di ricerca, che nell'Algoritmo 1 è implementata dalla funzione *ST.sample*. In una prima fase, viene applicato l'algoritmo UCT per selezionare un discorso candidato tra quelli presenti nell'albero di ricerca. Successivamente, l'output vocale è valutato sulla base del confronto con un campione di dati. Il valore restituito dalla stima fatta viene assegnato a tutti i nodi nel percorso che identifica il discorso e, per ognuno di essi, è registrata una nuova visita.

È importante notare la differenza tra la formula utilizzata per selezionare le frasi da restituire nell'Algoritmo 1 e quella usata nell'Algoritmo 2 per scegliere quali nodi esplorare. In quest'ultimo caso, raccogliere più informazioni è utile poiché permette di fare scelte migliori nelle iterazioni future. Perciò la formula UCT contiene anche un termine (il secondo) che concretizza il principio di exploration. Al contrario, l'algoritmo principale non può permettersi esplorazioni aggiuntive e sceglie il nodo più promettente: quello che massimizza la ricompensa media.

3.3.2 Valutazione dei discorsi candidati

Come già evidenziato, i discorsi candidati sono valutati in base al confronto con alcuni campioni estratti dal risultato della query. Le tecniche di campionamento vengono usate quando è troppo costoso, o persino impossibile, sfruttare tutti i dati. Tuttavia, a differenza delle computazioni tradizionali, quelle basate su campionamento sono intrinsecamente incerte, perché non permettono di calcolare risultati esatti. Nell'ambito dello studio del paradigma OLAP, caratterizzato da analisi che coinvolgono una grande quantità di dati, sono presenti vari contributi che sfruttano il campionamento per accelerare l'elaborazione delle query [25, 26, 29].

Queste attività di ricerca sono però finalizzate alla presentazione visuale dei risultati. In questo contesto, la struttura della visualizzazione viene stabilita a priori e implica dei vincoli che complicano il campionamento (ad esempio, la necessità di informazioni relative a sotto-popolazioni rare). Le peculiarità della data vocalization, ossia concisione e output sequenziale, rendono più flessibile il campionamento. In caso di output vocale, è infatti possibile raccogliere i campioni in un primo momento e usarli poi per scegliere gli elementi del discorso.

Algoritmo 3: Valutazione di un discorso

input: query q
input: speech t
output: reward r
// Seleziona casualmente un aggregato rappresentato nella cache;
$E \longleftarrow \{e : e \in q.aggs \ \ \& \ \ CA.size(e) > 0\}$;
$a \longleftarrow randomElement \in E$;
// Estrae un campione dalla cache;
$V \longleftarrow CA.resample(a)$;
// Stima il valore dell'aggregato sulla base del campione;
$e \longleftarrow estimateValue(a, V)$;
// Calcola la distribuzione corrispondente al modello dell'utente;
$b \longleftarrow \beta(a, t)$;
// Restituisce la probabilità ottenuta per il valore stimato;
return $P(e|b)$

L'Algoritmo 3 descrive, ad un alto livello di astrazione, la procedura *evaluateSpeech* utilizzata dall'Algoritmo 2. In particolare, dati in input una query e un discorso candidato, l'obiettivo è valutare quanto esso sia in grado di descrivere il risultato dell'interrogazione. L'output è costituito infatti da un valore di confidenza compreso tra 0 e 1, che rappresenta la qualità del discorso candidato.

L'approccio proposto basa il proprio funzionamento sulla presenza di una cache, popolata in fase di configurazione del sistema, che consente di ottenere campioni del risultato a partire da una query data. La scelta di eseguire computazioni in memoria è motivata dallo scenario applicativo, in cui l'interattività rappresenta un requisito fondamentale e il sistema deve garantire risposte rapide.

In particolare, l'Algoritmo 3 seleziona casualmente un aggregato del risultato che è rappresentato almeno da un elemento all'interno della cache. Questo perché, se in memoria non è disponibile nessuna informazione sul valore di un aggregato, non è possibile valutare un discorso in base ad esso. La funzione $CA.size$ consente di controllare questo aspetto, poiché restituisce il numero di tuple presenti nella cache che appartengono allo scope dell'aggregato passato come argomento. La scelta fatta si basa sull'utilizzo di una distribuzione casuale uniforme, in modo che tutti gli aggregati del risultato siano considerati egualmente importanti.

Successivamente l'algoritmo, con la funzione $CA.resample$, recupera un campione associato all'aggregato selezionato. Affinché questo procedimento sia efficiente, al momento della sottomissione di una query la cache indicizza i record in accordo con gli aggregati presenti nel suo risultato. Questa operazione può essere completata con una scansione sequenziale della cache e non causa alcun overhead, perché è eseguita durante la lettura del preambolo. Il campione estratto dalla cache permette di stimare il valore dell'aggregato, che dipende sia dalla funzione di aggregazione prevista dalla query sia dalla dimensione del campione.

Una volta selezionato un aggregato e stimato il suo valore sulla base del contenuto del campione considerato, viene utilizzata la semantica esplicitata nella Sezione 3.2.2 per generare la distribuzione di probabilità in grado di modellare la percezione dell'utente dopo aver ascoltato il discorso accettato dall'algoritmo. È importante notare che non viene generato il modello probabilistico completo, su-

perfluo dato che l'output vocale viene confrontato solamente con un campione, ma è sufficiente calcolare quello relativo all'aggregato selezionato. Il modello ottenuto è in grado di assegnare una probabilità alla stima ricavata dalla cache e il suo valore corrisponde alla confidenza restituita dall'Algoritmo 3.

Come visto nella Sezione 3.2.2, la distribuzione di probabilità corrispondente al modello mentale dell'utente è una normale avente deviazione standard proporzionale alla media. Si tratta di una distribuzione continua e perciò permette di determinare la probabilità per un intervallo di valori, ma non per un singolo punto [3]. Quindi, data una stima ottenuta grazie alla cache e il corrispondente modello mentale, non può essere calcolata la probabilità di quell'esatto valore.

L'algoritmo procede dunque ad un'approssimazione della stima disponibile, che è mappata in un intervallo centrato nel valore stimato e con ampiezza predefinita. Quest'ultima è costante nella valutazione di una query, poiché ampiezze diverse favorirebbero alcuni discorsi candidati, e viene calcolata moltiplicando il valore della baseline per un fattore compreso tra 0 e 1 (parametro dell'algoritmo).

Ad esempio, si supponga di selezionare l'aggregato a cui è associata la condizione *"I laureati provenienti dal Midwest con un salario iniziale pari almeno a 50mila"*. Analizzando il discorso considerato e applicando il modello mentale dell'utente, è stata ottenuta la distribuzione di probabilità $N(82K, 40K)$. L'operazione di campionamento ha invece portato ad ottenere una stima pari a 90K. Perciò, se si considera l'ampiezza dell'intervallo fissata a 10K, la confidenza si ottiene calcolando la probabilità che il valore sia compreso tra 85K e 95K nella distribuzione normale $N(82K, 40K)$. Il risultato ottenuto è pari circa a 0.1 [47].

$$F = N(82K, 40K)$$

$$P(85K < X \leq 95K) = P(X \leq 95K) - P(X \leq 85K) = F(95K) - F(85K) = 0.098$$

I valori di confidenza ottenuti consentono all'Algoritmo 1 di calcolare la qualità dei discorsi candidati, che può essere molto utile perché permette di offrire agli utenti un'informazione sul grado di incertezza che caratterizza la soluzione proposta. È possibile, ad esempio, generare un messaggio di warning se la confidenza è inferiore ad una soglia prestabilita. Alternativamente, il valore che rappresenta la qualità può essere restituito insieme al corrispondente output vocale.

Capitolo 4

Sviluppo di un sistema di vocalizzazione

Dopo essere stato adeguatamente approfondito, l'approccio olistico per la vocalizzazione di risultati OLAP è stato implementato. In particolare, è stata realizzata un'applicazione Java in grado, dopo una rapida fase di configurazione, di collegarsi ad un cubo multidimensionale memorizzato in un database relazionale e di eseguire query OLAP. Il sistema sviluppato è in grado di accettare interrogazioni esplicitate in linguaggio SQL, corrispondenti alle query GPSJ che possono essere risolte con l'approccio olistico (Sezione 3.1). L'output prodotto è espresso in linguaggio naturale e viene emesso sia in formato vocale sia in formato visuale.

Nelle Sezioni 4.1 e 4.2 sono presentati gli aspetti implementativi del sistema, con particolare riferimento alle principali attività che esso svolge: inizializzazione e risoluzione di una query data. La Sezione 4.3 riassume invece quanto fatto per automatizzare l'approccio, rendendolo utilizzabile in un qualsiasi data warehouse, e per ottimizzarlo, al fine di migliorare i risultati raggiunti dal metodo originale. Il capitolo è completato dalla Sezione 4.4, che descrive l'integrazione dell'applicazione nel framework conversazionale COOL [12] e mostra il sistema completo.

4.1 Inizializzazione del sistema

Come visto in precedenza, per poter utilizzare l'approccio olistico è necessario avere a disposizione una serie di informazioni relative al cubo multidimensionale:

- I nomi che identificano, all'interno del database, la fact table di riferimento e le dimension table ad essa associate.
- Per ogni misura presente nella fact table, il nome della colonna del database e il nome usato per indicarla nell'output vocale.
- Per ciascuna dimension table, il nome utilizzato per fare riferimento ad essa nel discorso e i livelli della gerarchia dimensionale.
- Per ogni livello di una gerarchia, il nome della colonna del database, il nome usato per rappresentarlo nell'output vocale e il contesto utile a definire predicati relativi a questo livello. Come esplicitato dalla sintassi del discorso nella Figura 3.1, quest'ultimo è la parte della frase che introduce i vari membri della gerarchia nella definizione di un predicato.

Lavori recenti svolti nell'ambito del paradigma OLAP [11, 12] sono in grado di estrarre automaticamente dal data warehouse matadati quali nomi delle misure, nomi degli attributi e strutture gerarchiche. Tuttavia, in questo caso il sistema deve avere a disposizione delle parti dell'output vocale e esse potrebbero essere estratte solamente applicando tecniche di NLP ai dati disponibili. Considerando che non si tratta di un aspetto centrale ai fini dell'obiettivo della tesi, si è scelto di non approfondirlo e di utilizzare un file di configurazione per memorizzare queste informazioni. Affinché il sistema possa essere eseguito su un determinato cubo multidimensionale, devono quindi essere specificati i metadati necessari.

Una volta configurato il sistema, è possibile procedere con l'inizializzazione della cache. Nella Sezione 3.3.2 viene sottolineato come l'approccio olistico basi la propria strategia di valutazione sulla presenza di un meccanismo di caching, che permette un accesso molto rapido ai dati ed è perciò adatto ad uno scenario applicativo in cui l'interattività costituisce un requisito fondamentale.

All'interno della cache sono presenti tutte le informazioni di cui il sistema ha bisogno, ossia quelle contenute nella fact table e nelle dimension table che identificano il cubo multidimensionale. È importante evidenziare che sono eseguite query sul data warehouse soltanto per popolare la cache, e quindi in fase iniziale, perché tutte le computazioni successive vengono svolte in memoria.

4.1.1 Cache delle gerarchie dimensionali

Per quanto riguarda le dimension table a cui fa riferimento il cubo multidimensionale, il meccanismo di caching implementato memorizza ognuna di esse attraverso una struttura ad albero. Questa codifica tutti i possibili membri della gerarchia dimensionale, suddividendoli in livelli associati ad attributi diversi ed esplicitando le dipendenze funzionali presenti attraverso relazioni padre-figlio tra i nodi. In particolare i livelli della gerarchia, partendo dalla radice fino a raggiungere le foglie dell'albero, corrispondono a granularità di aggregazione sempre più fini.

Si consideri, ad esempio, un cubo multidimensionale che modella le vendite e una gerarchia dimensionale prodotto in cui sono presenti gli attributi categoria, sottocategoria e prodotto. Essi corrispondo a livelli di aggregazione differenti e sono caratterizzati dalle dipendenze funzionali $prodotto \rightarrow sottocategoria \rightarrow categoria$. L'albero corrispondente contiene una radice che fa riferimento a tutti i prodotti, a cui è associato un figlio per ogni possibile categoria. A ciascuna categoria sono quindi legate le sottocategorie in grado di specializzarla, a loro volta associate a tutti i prodotti che vi appartengono. Di conseguenza, nella struttura ad albero ogni percorso radice-foglia rappresenta un certo prodotto, descritto dalla propria sottocategoria, a sua volta collegata alla categoria che la contiene.

La struttura ad albero introdotta è necessaria per il funzionamento del sistema di vocalizzazione, dato che consente di svolgere operazioni fondamentali quali:

- Parsing della query SQL accettata in input, perché l'albero codifica tutti i membri delle gerarchie dimensionali e permette quindi di capire se eventuali clausole di slezione applicate al valore degli attributi sono corrette o meno.

- Calcolo dei possibili valori degli attributi presenti nel group by set della query, utile per determinare sia gli aggregati del risultato sia i refinement relativi ad un certo attributo. In particolare, se un attributo compare nel group by set ma non sono presenti clausole di selezione relative alla stessa dimensione, sono rilevanti tutti i nodi del livello identificato dall'attributo. Al contrario, in presenza di una clausola di selezione, sono considerati soltanto i nodi discendenti dal membro della gerarchia indicato nella clausola.

Facendo riferimento all'esempio precedente, se l'attributo sottocategoria fosse nel group by set ma non venisse inserita nessuna clausola di selezione relativa alla dimensione prodotto, i possibili valori sarebbero tutte le sottocategorie. Se invece fosse presente una clausola che assegna "Bevande" alla categoria, sarebbero considerate solo le sottocategorie che la specializzano.

Algoritmo 4: Cache delle gerarchie dimensionali

```
input: dimensionDb
input: dimensionSpoken
input: levelsDb
// Esegue una query in grado di ottenere le combinazioni di attributi;
q ⟵ "select distinct " + join(",", levelsDb) + " from " + dimensionDb;
result ⟵ executeQuery(q);
// Inizializza la radice della gerarchia dimensionale;
root ⟵ new Member("any " + dimensionSpoken, 0, −);
// Itera su tutte le possibili combinazioni di attributi dimensionali;
while result.next() do
    member ⟵ root;
    // Itera fino a raggiungere l'ultimo livello della gerarchia;
    for level ∈ levelsDb do
        name ⟵ result.get(level);
        // Controlla se il valore dell'attributo è presente nella gerarchia;
        if member.hasChild(name) then
            member ⟵ member.getChild(name);
        else
            // In caso negativo, aggiunge un nuovo membro alla gerarchia;
            next ⟵ new Member(name, member.level + 1, member);
            member.addChild(next);
            member ⟵ next;
        end
    end
end
```

L'Algoritmo 4 illustra, ad alto livello, la procedura implementata per la memorizzazione della dimension table. I parametri *dimensionDb* e *dimensionSpoken* corrispondo rispettivamente al nome della tabella nel databse e al nome usato per fare riferimento ad essa in un discorso. In *levelsDb* sono invece contenuti i nomi delle colonne della tabella che memorizzano gli attributi dimensionali corrispondenti ai livelli della gerarchia, ordinati in base alla granularità.

Si può notare che, tramite una query eseguita sulla dimension table, vengono recuperate tutte le combinazioni degli attributi presenti nella gerarchia. Queste sono poi sfruttate per popolare la struttura al albero, in cui ogni nodo rappresenta un membro della gerarchia e contiene un possibile valore dell'attributo.

Ad esempio, data la tabella corrispondente alla gerarchia prodotto caratterizzata dagli attributi dimensionali *prodotto* $\rightarrow$ *sottocategoria* $\rightarrow$ *categoria*, viene creata la radice dell'albero che indica tutti i membri. I record della dimension table permettono quindi di associare ad essa tutti i nodi che fanno riferimento alle categorie, e questa procedura si ripete per sottocategorie e singoli prodotti.

4.1.2 Cache del fatto d'interesse

La memorizzazione delle gerarchie è necessaria al sistema per poter operare all'interno di uno spazio multidimensionale. Al contempo, affinché una query possa essere eseguita, la cache deve contenere dei campioni di dati associati al risultato ed è perciò necessario mantenere in memoria il contenuto della fact table.

L'implementazione proposta prevede una scansione sequenziale di tutte le righe di questa tabella. Grazie ad un confronto con la cache delle gerarchie dimensionali, ad ogni record sono associate le corrispondenti coordinate. Queste sono espresse tramite un insieme di coppie, in cui un membro della gerarchia è associato ad una dimensione, e permettono di identificare la "cella" del cubo a cui il record fa riferimento. Per ogni cella della tabella contenente una misura, viene quindi aggiunto un elemento alla cache. Esso contiene le coordinate dimensionali, il valore corrispondente e un contatore che mantiene il numero dei record originali (inizialmente è sempre uguale a 1, dato che non viene eseguita alcuna aggregazione). Nell'Algoritmo 5, che segue, è illustrato il procedimento descritto.

È importante notare che nella cache è mantenuto il massimo livello di dettaglio, ossia quello dei singoli elementi del cubo multidimensionale. Questo perché la granularità dei risultati dipende dalla singola query, che non può essere nota a priori (in fase di inizializzazione), e al contempo il sistema deve essere in grado di risolvere efficacemente anche interrogazioni che richiedono risultati specifici.

Algoritmo 5: Cache del fatto d'interesse

input: $factDb$
input: $dimensions$
input: $measures$
// Inizializza la cache;
$cache \longleftarrow \{\}$;
// Esegue una query in grado di recuperare tutti i record della fact table;
$dim \longleftarrow join(\text{“,”}, dimensions.map(d \rightarrow d.lastLevelDb))$;
$meas \longleftarrow join(\text{“,”}, measures.map(m \rightarrow m.measureDb))$;
$q \longleftarrow \text{“}select\text{ ”} + dim + \text{“,”} + meas + \text{“ }from\text{ ”} + factDb$;
$result \longleftarrow executeQuery(q)$;
// Itera su tutti i risultati ottenuti;
while $result.next()$ **do**
    $coordinates \longleftarrow \{\}$;
    // Memorizza le coordinate dimensionali di ogni record;
    for $dimension \in dimensions$ **do**
        $memberName \longleftarrow result.get(dimension.lastLevelDb)$;
        $member \longleftarrow dimension.getMember(memberName)$;
        $coordinates \longleftarrow coordinates \cup (dimension, member)$;
    end
    // Aggiunge un elemento alla cache per ogni valore delle misure;
    for $measure \in measures$ **do**
        $value \longleftarrow result.get(measure.measureDb)$;
        $cache \longleftarrow cache \cup new\ Entry(coordinates, measure, value, 1)$;
    end
end

Si consideri, ad esempio, il fatto d'interesse vendita associato alle dimensioni data, negozio e prodotto e descritto dalla misura costo. Per ogni record della fact table considerata, corrispondente ad un singolo evento, il meccanismo di caching determina le coordinate assegnando ad ogni dimensione il membro corrispondente della gerarchia (nel caso considerato la data precisa, il negozio specifico e il singolo prodotto). Viene dunque memorizzato un elemento della cache associato a queste coordinate, al valore della misura e al numero di record di riferimento (ossia 1).

Per ottimizzare l'estrazione dei campioni, l'algoritmo di valutazione e vocalizzazione prevede che, al momento della sottomissione di una query, la cache sia indicizzata in accordo con gli aggregati presenti nel risultato. L'obiettivo di questa operazione è ottenere campioni in grado di rappresentare adeguatamente un certo aggregato, perché supportati da informazioni relative a numerosi record.

Nella Sezione 3.1 è stato sottolineato che, in una certa query, aggregati diversi sono associati a condizioni mutualmente esclusive e quindi ogni record del dataset è rilevante al più per un aggregato. Lo stesso si può affermare anche per gli elementi della cache, dato che il meccanismo di memorizzazione implementato prevede una corrispondenza uno a uno tra questi e le celle presenti nella fact table.

Partendo dalla considerazione fatta, la tecnica di indicizzazione implementata assegna ad ogni aggregato del risultato tutti gli elementi della cache originale che appartengono al suo scope. Per verificare l'appartenenza viene eseguita la semplice procedura riportata nell'Algoritmo 6. In accordo con essa, un elemento appartiene allo scope di un aggregato se le loro misure corrispondono e se tutte le coordinate dell'elemento (ossia i membri della gerarchia che lo identificano) sono discendenti delle coordinate dell'aggregato corrispondenti alle stesse dimensioni.

Una volta collegati gli aggregati ai corrispondenti record della cache, questi sono raggruppati all'interno di un singolo elemento. Esso è caratterizzato dalle coordinate dell'aggregato, da un valore ottenuto sommando il contenuto degli elementi originali e da un contatore che indica il loro numero. Di conseguenza, per ogni aggregato la cache indicizzata mantiene un solo elemento: quello che riassume tutti i valori appartenenti allo scope. I record contenuti nella cache indicizzata corrispondono ai campioni utilizzati dall'algoritmo di vocalizzazione.

Algoritmo 6: Verifica di appartenenza allo scope

```
input: aggregate
input: entry
// Controlla la corrispondenza delle misure;
if aggregate.measure ≠ entry.measure then return false ;
// Controlla la corrispondenza delle coordinate;
for (dimension, m1) ∈ entry.coordinates do
    if dimension ∈ aggregate.coordinates.keys() then
        m2 ⟵ aggregate.coordinates.get(dimension);
        if ¬m1.isDescendantOf(m2) then return false ;
    end
end
```

L'Algoritmo 6 mostra la procedura che si occupa di verificare l'appartenenza di un elemento della cache allo scope definito da un certo aggregato. Si consideri, ad esempio, un singolo evento di vendita che fa riferimento in modo specifico a una data, a un negozio e a un prodotto e l'aggregato identificato da una categoria di prodotti e da una città. L'elemento della cache appartiene allo scope definito per l'aggregato solo se il prodotto appartiene alla categoria e il negozio è situato all'interno della città. Questo equivale ad affermare che i membri che identificano l'elemento devono essere discendenti dai membri corrispondenti dell'aggregato. È importante sottolineare che i controlli sulle coordinate dell'elemento della cache si limitano alle dimensioni che sono considerate nella definizione dell'aggregato. Per gli altri attributi, l'aggregato considera infatti tutti i possibili valori.

4.2 Risoluzione di una query

Una volta completata la fase di inizializzazione e popolata la cache, il sistema è in grado di eseguire query sul data warehouse. L'implementazione del metodo di risoluzione è coerente con quanto previsto dall'approccio olistico, fatta eccezione per alcune modifiche discusse dettagliatamente nella Sezione 4.3.

Il sistema accetta in input una stringa contenente la query da risolvere, espres-

sa in formato SQL. Viene quindi eseguito un semplice parsing dell'interrogazione, finalizzato a verificare la sua validità e ad estrarre i principali elementi della query: misura analizzata, operatore di aggregazione, group by set e clausole di selezione. La clausola "from" e il suo contenuto sono volontariamente ignorati, perché il cubo multidimensionale da interrogare deve essere scelto durante la configurazione del sistema e non in fase di interrogazione. Dato che l'approccio olistico permette di risolvere solamente query che analizzano un'unica misura, nel caso in cui vengano specificate più colonne di aggregazione (ognuna associata al corrispondente operatore) il sistema procede scegliendo l'ultima e ignorando le altre.

Avendo a disposizione l'interrogazione data in input, il sistema determina il preambolo e avvia l'output vocale. Sono quindi calcolati gli aggregati che compongono il risultato, combinando i possibili valori degli attributi dimensionali presenti nel group by set della query: ad ogni combinazione corrisponde un aggregato.

A causa del requisito di concisione, l'output vocale non è in grado (escludendo rari casi in cui i dati sono caratterizzati da distribuzioni particolarmente uniformi) di rappresentare efficacemente un elevato numero di aggregati. Si consideri ad esempio una tabella contenente 2500 righe, risultato non molto insolito nell'ambito delle sessioni OLAP. Un discorso composto da un numero limitato di frasi (3, 4, 5, ...) è caratterizzato da un contenuto informativo nettamente inferiore.

Si è scelto, per questo motivo, di aggiungere un ulteriore parametro all'implementazione: il massimo numero di aggregati che possono essere contenuti nel risultato della query. Il suo valore è stato fissato a 1500, ma può essere facilmente modificato. Se gli aggregati superano questa soglia il sistema non risolve l'interrogazione, ma restituisce un messaggio di warning che informa l'utente della granularità troppo fine e lo invita a formulare una query ad un livello più alto.

Dopo aver determinato gli aggregati presenti nel risultato, in accordo con essi il sistema indicizza gli elementi della cache per poter mettere a disposizione dell'algoritmo di valutazione e vocalizzazione i campioni necessari. Una volta completata l'indicizzazione, viene inizializzato l'albero di ricerca contenente i discorsi candidati. Queste operazioni sono eseguite in background durante l'emissione del preambolo e, di conseguenza, non inficiano l'interattività del sistema.

Avendo a disposizione lo spazio di ricerca e i campioni del risultato, l'approccio olistico può iniziare le proprie iterazioni finalizzate a determinare, frase dopo frase, il miglior output vocale. Come ampiamente discusso nella Sezione 3.3, l'algoritmo di riferimento si basa sull'esecuzione di computazioni in background durante l'emissione vocale del discorso. Per poter implementare una strategia di questo tipo, il sistema deve essere in grado di svolgere le seguenti operazioni:

- Vocalizzare un testo in formato scritto.
- Determinare se la lettura della frase corrente è in corso. Questo punto è particolarmente importante, perché consente all'approccio olistico di arrestare la stima della qualità dei discorsi candidati e di restituire la prima frase dell'output vocale più promettente.

Queste attività devono essere svolte dalla componente adibita all'emissione dell'output vocale e, dovendo integrare il sistema all'interno di un framework esistente, sarebbe stato irragionevole aggiungere questa funzionalità senza sapere quale parte dell'applicazione si occupa della presentazione del risultato. Se il sistema di vocalizzazione fosse invece autonomo, ad esempio se venisse eseguito all'interno di un assistente vocale, potrebbero essere sfruttate API comuni (come quelle offerte dal modulo Text to Speech[1]) per ottenere il comportamento desiderato.

Per testare l'applicazione prima dell'integrazione, è stata quindi sviluppata una semplice interfaccia a linea di comando che consente all'utente di sottomettere query in formato SQL. Il sistema visualizza le varie parti dell'output vocale non appena sono disponibili e simula la lettura della frase corrente attraverso un timeout, la cui durata è proporzionale alla lunghezza del discorso.

Nella Figura 4.1, che segue, sono riportati tre semplici esempi di funzionamento dell'interfaccia a riga di comando realizzata per lanciare il sistema. In particolare, all'avvio dell'applicazione sono eseguite automaticamente la fase di configurazione e quella di inizializzazione. Una volta completate, il sistema rimane in attesa di stringhe di input corrispondenti alle query in formato SQL e mostra all'utente le varie frasi dell'output vocale generato dall'approccio olistico.

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

```
Insert input query
select avg(unit_sales)
where product.product_category='breakfast foods' and store.store_city='los angeles'
group by customer.country, time_by_day.quarter, product.product_subcategory
Considering any customer, products of category Breakfast Foods, stores in Los Angeles a
Results are broken down by customer country, product subcategory and quarter.
Around 3.1 is the Average unit sales.
The value increases by 6.9 percent for products of subcategory Pancake Mix.
The value increases by 2.9 percent for dates of Q4.
The value increases by 0.74 percent for dates of Q1.
17:38:13 DEBUG [main] (VolapSession.java:95) - Evaluated query in 0 min, 27 s, 202 ms
Average relative error: 0.057153087179609076
Mental model quality: 0.31948738234522295
Vocalization time: 31680 ms

Insert input query
select avg(store_cost)
where time_by_day.quarter='q3'
group by product.product_category, time_by_day.the_month
Considering any customer, any product, any store and dates of Q3.
Results are broken down by product category and month.
Around 2.9 is the Average store cost.
The value increases by 340 percent for products of category Beer and Wine.
The value increases by 16 percent for products of category Canned Anchovies.
The value increases by 10 percent for products of category Plastic Products.
17:39:40 DEBUG [main] (VolapSession.java:95) - Evaluated query in 0 min, 24 s, 560 ms
Average relative error: 0.08970620547626823
Mental model quality: 0.26915960552210266
Vocalization time: 30960 ms

Insert input query
select sum(store_sales)
where customer.city='renton'
group by store.store_country, time_by_day.quarter
Considering customers from Renton, any product, any store and any date.
Results are broken down by store country and quarter.
Around 5200 is the Sum of store sales.
21 percent of the value comes from dates of Q1.
24 percent of the value comes from dates of Q2.
17 percent of the value comes from dates of Q3.
17:40:41 DEBUG [main] (VolapSession.java:95) - Evaluated query in 0 min, 20 s, 560 ms
Average relative error: 0.012441068026400956
Mental model quality: 0.3806790729480271
Vocalization time: 24640 ms
```

Figura 4.1: Esempio di query eseguite usando la CLI

4.3 Automatizzazione e ottimizzazione dell'approccio

Il sistema sviluppato è in grado, dopo la configurazione, di eseguire interrogazioni su un qualsiasi data warehouse. Per offrire questa possibilità, in fase di implementazione sono stati adottati alcuni accorgimenti utili a renderlo efficace indipendentemente dal cubo multidimensionale analizzato e dalla query data. Sono state apportate anche altre modifiche all'approccio originale, finalizzate ad ottimizzarlo e ad ottenere quindi un output vocale in grado di avvicinare quanto più possibile la percezione dell'utente al risultato reale. Questa sezione riassume i cambiamenti applicati, con particolare riferimento ai vantaggi che garantiscono.

4.3.1 Revisione sintattica e semantica

Come formalizzato nella Sezione 3.1, l'approccio olistico è in grado di risolvere query GPSJ caratterizzate da una sola misura e dal corrispondente operatore di aggregazione. Le funzioni supportate sono media, somma e count.

La sintassi definita per l'output vocale, ed esplicitata nella Figura 3.1, è adatta al caso in cui la query richieda di calcolare la media. Infatti i vari sottoinsiemi di dati associati ai refinement sono tipicamente caratterizzati, proprio per la definizione di media, da valori aggregati maggiori o minori rispetto a quello della baseline (che riassume tutti i risultati). La variazione relativa espressa dai refinement è quindi in grado di rappresentare efficacemente questo tipo di distribuzione.

Queste considerazioni non valgono però per gli operatori somma e count, perché i valori aggregati generati da essi per rappresentare sottoinsiemi di dati sono sempre minori (o uguali nel caso limite in cui il sottoinsieme corrisponde all'insieme dato) rispetto al valore aggregato contenuto nella baseline. Di conseguenza, nello scenario descritto una variazione relativa risulta essere poco efficace, perché corrisponde sempre ad un decremento e può trarre in inganno l'utente.

Si consideri ad esempio l'output *"700mila è la somma degli stipendi a metà carriera. Il valore diminuisce del 60% per i laureati provenienti dal Northeast."*. In accordo con il principio di uniformità, a fronte di una frase in cui è indicato

un decremento per la regione Northeast, l'utente può erroneamente assumere un valore maggiore per le altre regioni (Midwest, West e South). Questa intuizione non corrisponde però al reale valore dei dati, in cui il 40% della somma è costituito da dati relativi alla regione Northeast. Assumendo una distribuzione uniforme, la somma dei valori che fanno riferimento alle altre tre regioni sarebbe infatti pari al 20% del totale (ossia il restante 60% equamente suddiviso) e quindi minore.

A valle di queste osservazioni, nell'ambito dell'implementazione dell'approccio olistico è stata proposta e utilizzata una sintassi alternativa del discorso. Essa è destinata in modo specifico alla risoluzione di query in cui l'operatore di aggregazione è uguale a somma o count, mentre viene usata la sintassi originale quando è richiesto il calcolo della media. La Figura 4.2, riportata di seguito, contiene la grammatica EBNF che rappresenta la sintassi alternativa proposta.

<Speech>::=<Preamble><Baseline><Refinement>*
<Preamble>::=Considering (<Pred>|(<Pred>)+ and <Pred>).
[Results are broken down by (<Level>|<Level>+ and <Level>).]
<Baseline>::=<Value> is the <Aggregate>.
<Refinement>::=<Quantifier> of the value comes from
(<Pred>|(<Pred>)+ and <Pred>).
<Pred>::=<Dimension Context> <Member>

Figura 4.2: Sintassi alternativa rappresentata in EBNF

La sintassi proposta possiede tutte le caratteristiche necessarie per rappresentare un output vocale, ossia quelle esplicitate nella Sezione 3.2.1, e mantiene interamente la struttura della sintassi originale. Il preambolo è infatti seguito da una baseline, a sua volta accompagnata da un numero variabile di refinement. L'unica differenza è costituita proprio dai refinement, in cui il quantificatore non indica più uno scostamento espresso in termini relativi ma rappresenta la "porzione" del valore complessivo (ossia quello della baseline) presente all'interno dello scope definito dai predicati. È importante ricordare che i predicati definiscono lo scope assegnando uno o o più attributi dimensionali a specifici valori.

Facendo riferimento all'esempio precedente, con la sintassi alternativa l'output vocale non sarebbe *"700mila è la somma degli stipendi a metà carriera. Il valore diminuisce del 60% per i laureati provenienti dal Northeast."* ma *"700mila è la somma degli stipendi a metà carriera. Il 40% del valore è originato dai laureati provenienti dal Northeast."*. Il nuovo refinement è espresso in termini relativi, per non creare contraddizioni coi predecessori, e elimina l'ambiguità sottolineata.

Una volta definita questa sintassi alternativa, è stata modellata la semantica associata in accordo con le ipotesi di simmetria, concentrazione, composizione e uniformità (descritte nella Sezione 3.2.2 e supportate da uno studio dell'utente [47]). La semantica β di un discorso t è una funzione che assegna ad un aggregato a una distribuzione normale corrispondente alla percezione dell'utente.

$$\beta(a,t) = N(M(a,t),\sigma)$$

Se l'output vocale t è composto solo da una baseline, la funzione $M(a,t)$ calcola la media della normale suddividendo equamente il suo valore tra tutti i possibili aggregati. In particolare, indicando con *base* il valore della baseline e con n il numero di aggregati presenti nel risultato, $M(a,t)$ restituisce sempre il loro rapporto.

$$M(a,t) = base/n$$

Se invece il discorso t è formato da una baseline seguita da un refinement, anch'essa influenza il risultato restituito da $M(a,t)$. Si consideri un refinement r, che tramite i suoi predicati definisce un sottoinsieme di dati e che descrive il valore aggregato corrispondente con un fattore α. Esso è in grado di influenzare gli aggregati a che ricadono nello scope di r, il cui numero è indicato con m.

$$M(a,t) = base * \alpha/m$$

Inoltre, dato che la baseline è espressa in termini assoluti e i refinement sono caratterizzati da una natura relativa, questi ultimi sono anche in grado di modificare la percezione degli aggregati a che non appartengono al loro scope. Il ragionamento applicato è lo stesso del caso precedente, ma viene considerato il sottoinsieme complementare a quello definito dai predicati del refinement.

$$M(a,t) = base * (1-\alpha)/(n-m)$$

La strategia descritta si applica in modo speculare se sono presenti più refinement e, in questo caso, ognuno di essi influisce sulla semantica generata.

Ad esempio, si consideri l'output vocale *"800mila è la somma degli stipendi a metà carriera. Il 40% del valore è originato dai laureati provenienti dal Northeast."*. La distribuzione normale ha media uguale a 320mila (800∗0.4) per l'aggregato caratterizzato dalla regione Northeast e 160mila per gli aggregati relativi alle regioni Midwest, West e South ($800 * 0.6/3$). Se il fattore moltiplicativo scelto per calcolare la deviazione standard è 0.5, il valore di σ è 100mila ($800/4 * 0.5$).

4.3.2 Generazione dei discorsi candidati

L'approccio olistico, come ripetutamente evidenziato, basa il proprio funzionamento sulla presenza di uno spazio di ricerca contenente tutti i discorsi candidati. In particolare, si tratta di una struttura ad albero che codifica tutte le possibili combinazioni tra le baseline disponibili e i refinement utilizzabili.

Le baseline vengono calcolate in seguito all'esecuzione di alcune statistiche sul dataset. I refinement sono invece generati combinando i possibili valori degli attributi dimensionali presenti nel group by set della query con dei fattori, definiti a priori, che quantificano una variazione relativa. Tuttavia, non è chiaro come siano eseguite le statistiche e come siano scelti i fattori usati nei refinement.

Per implementare e automatizzare l'approccio, è stato necessario capire come generare efficacemente baseline e refinement. Query differenti possono richiedere l'accesso ai dati più vari e, quindi, essere caratterizzate da spazi di ricerca contenenti baseline e fattori usati nei refinement completamente diversi. Avendo l'obiettivo di realizzare un sistema in grado di eseguire query su qualsiasi data warehouse, l'uso di baseline o fattori predefiniti non costituisce una soluzione valida.

L'implementazione proposta genera i discorsi candidati sfruttando la cache, il cui funzionamento è stato discusso nella Sezione 4.1. In particolare l'indicizzazione della cache, eseguita in seguito alla sottomissione di una query, fa sì che venga memorizzato un elemento per ogni possibile aggregato. Dato che i refinement sono caratterizzati da un solo predicato, ogni elemento della cache non può appartenere allo scope di più refinement mutualmente esclusivi. Di conseguenza, una scansione

sequenziale della cache indicizzata consente di stimare sia il valore aggregato che riassume tutti i risultati della query (baseline) sia i fattori che indicano la variazione relativa associata ai vari attributi dimensionali (utili a generare i refinement).

Le caratteristiche di concisione e semplicità dell'output vocale non permettono di restituire il valore ottenuto alla massima precisione, ma esso deve essere approssimato. In particolare, sia nel valore della baseline sia in quello dei fattori usati nei refinement, viene mantenuto il numero di cifre significative indicato da un apposito parametro. Esso è stato posto a 2 nell'implementazione presentata, ma può essere facilmente modificato in fase di configurazione del sistema.

Ad esempio, si consideri una query che richiede di calcolare il salario medio dei laureati a metà carriera raggruppando i dati per regione di provenienza del lavoratore e per salario iniziale. Se i possibili valori dell'attributo dimensionale regione sono "Northeast", "Midwest", "West" e "South" e quelli del salario iniziale "Inferiore a 50mila" e "Almeno 50mila", la cache indicizzata potrebbe contenere gli elementi riportati nella Figura 4.3. Il valore di count è sempre pari a 1 per semplificare l'esempio, ma è importante ricordare che ogni elemento della cache indicizzata è solito aggregare i valori di numerosi record del dataset di riferimento.

Aggregato	Regione	Salario iniziale	Somma	Count
1	Northeast	Inferiore a 50mila	90 000	1
2	Midwest	Inferiore a 50mila	74 000	1
3	West	Inferiore a 50mila	53 000	1
4	South	Inferiore a 50mila	71 000	1
5	Northeast	Almeno 50mila	99 000	1
6	Midwest	Almeno 50mila	118 000	1
7	West	Almeno 50mila	105 000	1
8	South	Almeno 50mila	110 000	1

Figura 4.3: Esempio di contenuto della cache indicizzata

Una scansione sequenziale della cache indicizzata consente di capire che:

- La stima del valore della baseline corrisponde a 90mila e può essere ottenuta aggregando tutti gli elementi della cache (rapporto tra somma globale dei valori e conteggio complessivo, ossia 720000/8).

- Il refinement che associa il valore "Northeast" all'attributo dimensionale regione dovrebbe prevedere una variazione relativa pari ad un incremento del 5%, perché il valore ottenuto aggregando tutti i dati appartenenti al suo scope è uguale a 94500 (rapporto tra $90000 + 99000$ e 2).

- Allo stesso modo, il refinement che associa il membro "Almeno 50mila" alla gerarchia che rappresenta il salario iniziale dovrebbe indicare un incremento del 20% (valore aggregato pari a $(99000 + 118000 + 105000 + 110000)/4$).

La cache implementata offre quindi la possibilità di calcolare, sia per la baseline sia per tutti i possibili refinement, il valore associato ideale: quello che rappresenta i dati in modo "esatto", compatibilmente con l'approssimazione applicata.

Si considerino, ad esempio, tutti i refinement associati ad un determinato scope e quindi caratterizzati dal medesimo predicato. Per definizione del modello mentale, il refinement r in grado di rappresentare i dati nel modo migliore è quello che avvicina maggiormente la percezione dell'utente al risultato reale. Perciò r deve contenere l'informazione corretta relativa allo scope descritto, e la cache permette di ottenerla. Lo stesso ragionamento viene applicato anche alla baseline.

La versione originale dell'approccio olistico prevede più baseline e più refinement per ogni possibile valore di un attributo dimensionale. In accordo con l'osservazione precedente, nella versione implementata sono invece presenti rispettivamente una sola baseline e un solo refinement per membro della gerarchia: quelli calcolati attraverso la cache. Questo aspetto costituisce indubbiamente un vantaggio per l'algoritmo di valutazione e vocalizzazione, sia in termini di efficacia perché non sono inseriti nello spazio di ricerca frasi contenenti informazioni non ottimali, sia per quanto riguarda l'efficienza poiché il numero dei discorsi candidati diminuisce. La generazione dei possibili output vocali non causa un overhead significativo, dato che è eseguita durante l'emissione vocale del preambolo.

4.3.3 Generazione dell'albero di ricerca

Nell'albero di ricerca generato dall'approccio olistico, alla radice sono collegate tutte le possibili baseline. Ogni livello successivo è invece associato ad uno degli attributi dimensionali presenti nel group by set della query e contiene i nodi che rappresentano i refinement ottenuti combinando tutti i possibili valori dell'attributo con i fattori in grado di quantificare una variazione relativa.

Come spiegato nella Sezione 4.3.2, la prima differenza tra l'approccio originale e la versione implementata è costituita dal fatto che quest'ultima genera solamente una baseline e un refinement per ogni membro della gerarchia dimensionale. Essi vengono calcolati a partire dalla cache e, in questo modo, il sistema sviluppato è in grado di eseguire interrogazioni su un qualsiasi cubo multidimensionale.

La prima implementazione realizzata prevedeva una struttura dell'albero di ricerca coerente con quanto previsto dall'approccio ma, a seguito di alcune prove sperimentali, è emerso chiaramente un limite. Infatti, dato che ogni livello dell'albero è associato ad un certo attributo dimensionale, può essere emesso un solo refinement che descrive uno dei suoi valori. Al contempo, è necessario restituire un refinement per ogni attributo dimensionale presente nel group by set. Di conseguenza, se le variazioni presenti nei dati non sono distribuite tra le varie dimensioni ma si concentrano in alcune di esse, i refinement scelti dall'approccio non sono in grado di offrire all'utente il massimo contenuto informativo possibile.

Si consideri, ad esempio, la query che richiede di calcolare il salario medio raggruppando i dati per regione di provenienza del lavoratore e per salario iniziale. Assumendo che il discorso in grado di massimizzare la qualità percepita dall'utente contenga i refinement *"Il valore aumenta del 20% per i laureati provenienti dal Northeast"* e *"Il valore diminuisce del 30% per i laureati provenienti dal South"*, la strategia originale non sarebbe in grado di selezionare questo output.

L'implementazione presentata propone quindi un approccio alternativo, a cui si fa riferimento con il termine *completo*. Esso prevede che, in tutti i livelli dell'albero, a ciascun nodo venga collegato un figlio per ogni refinement relativo ad un possibile valore di un attributo dimensionale presente nel group by set della query. È però necessario escludere i refinement che compaiono nei nodi precedenti.

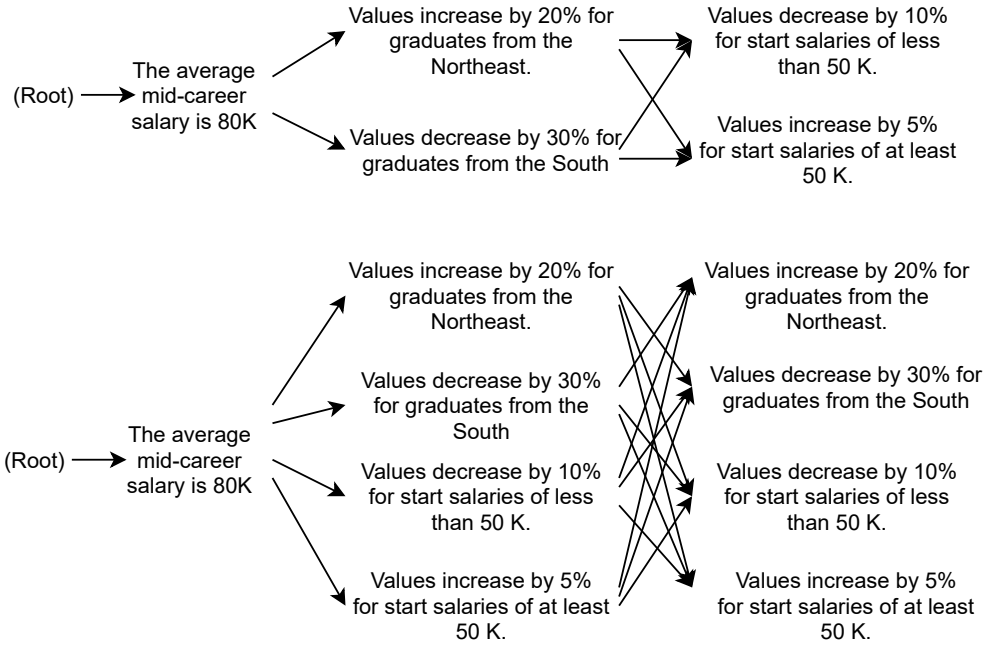

Figura 4.4: Esempio di albero di ricerca completo

La Figura 4.4 riporta il confronto tra l'albero di ricerca generato dall'approccio originale e quello ottenuto con la variante completa, nel caso in cui la query corrisponda a quella indicata nell'esempio precedente. Si può notare che il secondo albero è caratterizzato da una maggiore espressività, in quanto cattura tutti i discorsi codificati dal primo aggiungendone altri. Il secondo albero viene definito completo, perché rappresenta tutte le combinazioni tra i possibili refinement.

Nell'implementazione presentata è possibile, tramite un apposito parametro, scegliere se generare l'albero di ricerca coerente con l'approccio originale o quello completo. Questo ha permesso, come illustrato nel Capitolo 5, di confrontare le due strategie e di verificare i vantaggi ottenuti grazie ad un albero completo.

Gestione della complessità computazionale

Generare un albero completo, che considera tutte le possibili combinazioni di refinement, può consentire all'approccio olistico di ottenere risultati migliori. Uno spazio di ricerca di dimensione maggiore può però causare un overhead computazionale, non ammissibile in uno scenario applicativo che richiede interattività.

Fissato il numero di refinement a n e data una query caratterizzata da un group by set contenente n attributi dimensionali, ognuno dei quali prevede m possibili valori, il numero dei nodi foglia dell'albero generato dall'approccio olistico è pari a m^n. Considerando invece l'albero completo, il numero di foglie è $(m * n)^n$ perché ad ogni livello sono disponibili tutti i refinement (ad eccezione di quelli già selezionati). La dimensione dell'albero aumenta quindi di un fattore n^n che, pur essendo n limitato dalla scarsa memoria degli utenti [32], non è trascurabile.

È stata dunque adottata una tecnica euristica per contenere la dimensione dello spazio di ricerca, indipendentemente dal cubo multidimensionale interrogato e dalla query formulata. Secondo il principio di uniformità descritto nella Sezione 3.2.2, in assenza di informazioni l'utente tende ad assumere una distribuzione uniforme dei dati. Perciò i refinement più interessanti risultano essere quelli che indicano un maggiore scostamento dalla distribuzione uniforme precedentemente percepita. Ad esempio, data una query che calcola una media, i refinement più interessanti sono caratterizzati da una variazione maggiore in valore assoluto.

La strategia descritta è usata per ordinare tutti i refinement generati, vengono poi selezionati i primi N (parametro che indica il massimo numero di figli di un nodo dell'albero) ed essi, a loro volta, sono utilizzati per generare l'albero di ricerca completo. Quest'ultimo è caratterizzato da una complessità nell'ordine di grandezza di $O(m^k)$, che può essere controllata grazie a questa tecnica. Infatti, m e k corrispondono rispettivamente al massimo numero di figli di un nodo e al numero dei livelli dell'albero e sono entrambi parametri dell'approccio.

4.4 Integrazione con un framework conversazionale

Una volta completata l'implementazione del sistema di vocalizzazione, esso è stato integrato all'interno del framework conversazionale COOL [12] per ottenere una soluzione end-to-end. Questa è in grado di sfruttare simultaneamente i vantaggi portati da strumenti di NLIDB e di data vocalization e, perciò, riesce a rispondere ad un input espresso in linguaggio naturale con un output dello stesso tipo.

Attualmente COOL è reso disponibile tramite un'applicazione web, la cui interfaccia[2] consente all'utente di eseguire sessioni OLAP specificando le richieste in formato testuale o in formato vocale. È infatti possibile sia scrivere l'input in un apposito form, sia attivare il microfono e interagire col sistema parlando.

L'applicazione web è caratterizzata da un'architettura client-server, in cui le due componenti comunicano con chiamate GET e POST. In particolare, il client ha il compito di acquisire le richieste dell'utente e di inviarle al server. Questo mette a disposizione una API, che si occupa di re-indirizzare le richieste al modulo conversazionale destinato alla loro risoluzione. Il server restituisce quindi il risultato al client, codificandolo in formato JSON (*JavaScript Object Notation*).

Se l'utente sottopone al sistema una frase che non contiene ambiguità, essa viene trasformata nella query corrispondente e quest'ultima è eseguita sul data warehouse. L'interfaccia mostra il *parse tree* generato per la traduzione dell'input dell'utente, la query SQL corrispondente all'interrogazione inserita e una tabella contenente i primi n risultati ottenuti. L'obiettivo della fase di integrazione è aggiungere alle informazioni restituite l'output vocale, permettendo all'applicazione web sia di visualizzarlo nella sua interfaccia sia di emetterlo vocalmente.

Come discusso nella Sezione 3.3, l'approccio olistico si basa sull'esecuzione di attività in background durante l'emissione dell'output. Per implementare questo tipo di strategia è necessario vocalizzare un testo scritto, ma soprattutto comunicare al chiamante quando l'output vocale contente la frase corrente è terminato.

In un'applicazione client-server, l'emissione del testo in formato vocale è gestita dal client. Al contempo, la valutazione e la vocalizzazione delle query sono attività da eseguire lato server. Quindi, per poter "sovrapporre" efficacemente la computazione all'output vocale, è necessario implementare una comunicazione stateful tra client e server. In particolare, il client dovrebbe essere in grado di inviare al server sia la richiesta iniziale contenente la query sia una nuova richiesta per ogni frase dell'output vocale completamente emessa. Simultaneamente il server dovrebbe proseguire la propria attività, restituendo gradualmente al client le varie frasi e un'informazione aggiuntiva in caso di conclusione del discorso.

[2] `https://big.csr.unibo.it/projects/cool/`

Tuttavia, l'applicazione web che consente di utilizzare il sistema COOL prevede un'interazione stateless: ad ogni query formulata dall'utente corrisponde un output completo. Nell'ambito del lavoro di tesi svolto si è scelto di mantenere questo tipo di comunicazione, perché l'implementazione di un'interazione stateful sarebbe stata più dispendiosa e avrebbe comportato modifiche significative a parti del sistema sviluppate in altri lavori. È però importante ricordare che, per sviluppare un'applicazione web interattiva, la comunicazione stateful è necessaria.

Lato server, il sistema sviluppato è stato integrato nella Java Servlet realizzata per COOL (che utilizza Tomcat come server di riferimento). Il framework conversazionale, dopo aver tradotto una query dell'utente in SQL, richiama la componente di vocalizzazione e arricchisce l'oggetto JSON destinato al client con il risultato ottenuto. Per quanto riguarda la fase di inizializzazione, essa viene avviata quando il framework riceve la prima richiesta e, una volta completata questa procedura, la componente rimane attiva finché il server è in esecuzione.

Per quanto riguarda la componente client dell'applicazione web, sono stati apportati alcuni semplici cambiamenti. Agli elementi offerti dall'interfaccia originale è stata affiancata una rappresentazione dell'output vocale, dove sono riportati anche i valori restituiti da metriche in grado di descrivere il grado di incertezza della soluzione proposta (discusse in modo dettagliato nel Capitolo 5).

Oltre ad essere visualizzato, il discorso viene automaticamente emesso in formato vocale. Questo è reso possibile dall'utilizzo delle funzioni di *SpeechSynthesis* offerte dalle Web Speech API[3], che consentono di vocalizzare un testo scritto scegliendo la lingua e la velocità di lettura. Si tratta di una scelta coerente con quanto era stato fatto per comprendere un input vocale nell'applicazione originale, poiché essa è basata sul modulo *SpeechRecognition* delle medesime API.

La Figura 4.5 mostra un esempio di query eseguita tramite l'interfaccia web, focalizzandosi sul confronto tra il risultato reale e l'output restituito dal sistema di vocalizzazione. La tabella riportata è composta da 78 righe ma, per motivi di spazio, l'immagine è in grado di mostrare solo le prime. Al contrario, l'output vocale è conciso e si focalizza sulle principali caratteristiche del risultato.

[3] `https://wicg.github.io/speech-api/`

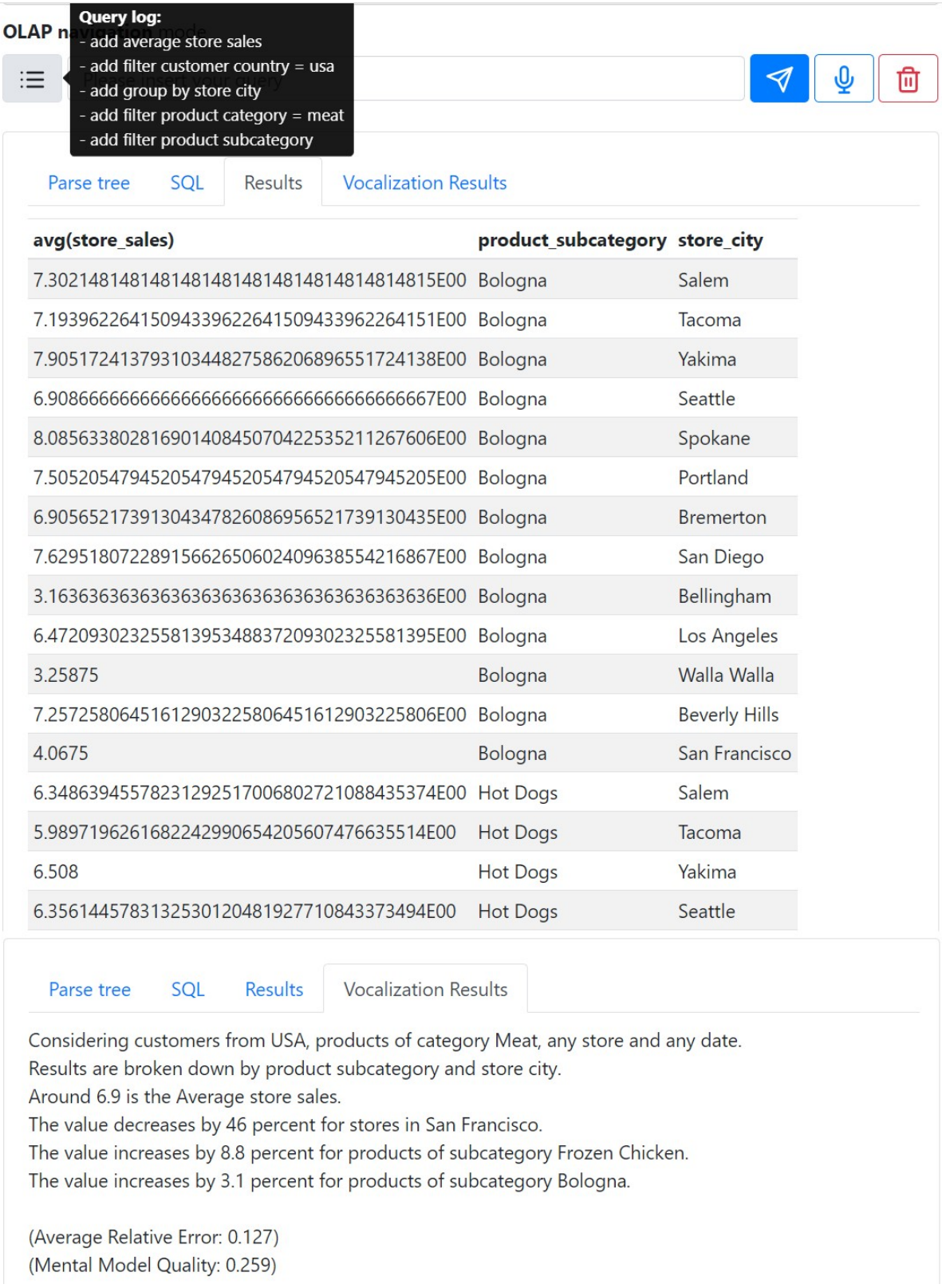

Figura 4.5: Esempio di query eseguita con l'interfaccia web

Capitolo 5

Valutazione dei risultati ottenuti

Dopo aver completato l'implementazione del sistema di vocalizzazione, sono state svolte alcune attività finalizzate a valutare il lavoro fatto. Esse possono essere suddivise in due categorie: test di efficacia e test di efficienza. I primi giudicano la capacità del sistema di raggiungere l'obiettivo prefissato, mentre i secondi valutano la quantità di risorse necessarie per farlo. Ove possibile, i risultati raggiunti dal sistema sono stati confrontati con quelli ottenuti dall'implementazione originale, realizzata nell'ambito del lavoro di ricerca preso come riferimento [47].

Questo capitolo è dedicato alla presentazione delle attività di valutazione svolte e dei risultati ottenuti. In particolare, nella Sezione 5.1 sono introdotti i data mart su cui è stato eseguito il sistema sviluppato e nella Sezione 5.2 sono approfondite le metriche di valutazione scelte. Le Sezioni 5.3 e 5.4 descrivono invece i test di efficacia e i test di efficienza e riportano i risultati ottenuti sui cubi analizzati.

5.1 Data mart di riferimento

Avendo integrato il sistema di vocalizzazione nel framework COOL, il primo cubo multidimensionale su cui è stata valutata l'implementazione realizzata è lo stesso usato per il testing di COOL [12]. Si tratta di *Foodmart*[1], un dataset contenente informazioni relative alle vendite di prodotti alimentari tra il 1997 e il 1998.

[1] `https://github.com/julianhyde/foodmart-data-mysql`

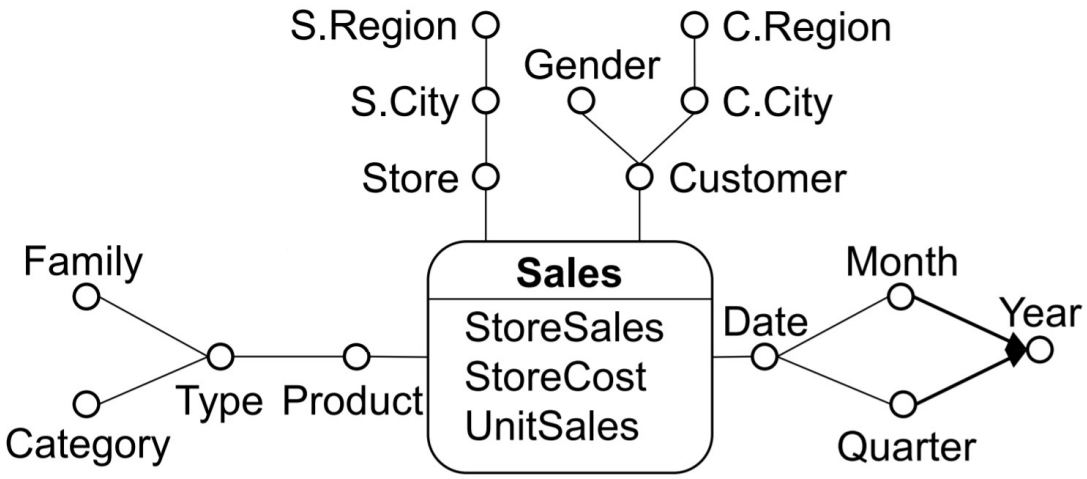

Figura 5.1: DFM semplificato relativo al cubo foodmart sales [12]

Nella Figura 5.1 è contenuta una versione semplificata del DFM che modella la struttura del cubo Foodmart Sales. Si può notare che il fatto è rappresentato dalla vendita e che gli eventi sono caratterizzati dalle misure "store sales", "store cost" e "unit sales". Le principali dimensioni di analisi sono prodotto, negozio, cliente e data e per ognuna di esse è definita la corrispondente gerarchia.

Foodmart consente di valutare l'efficacia e l'efficienza del sistema in termini assoluti. Tuttavia, come accennato in precedenza, i test sono stati svolti anche con l'obiettivo di confrontare i risultati raggiunti dal sistema proposto con quelli ottenuti dall'implementazione originale. Questo avrebbe infatti permesso di valutare la bontà delle scelte fatte per l'automatizzazione e l'ottimizzazione dell'approccio e di capire se esse hanno effettivamente portato a dei miglioramenti.

Perciò il sistema è stato eseguito anche sul cubo *Flight Delays*, scelto perché utilizzato per le valutazioni sperimentali svolte nell'ambito del lavoro di ricerca di riferimento [47]. Flight Delays contiene informazioni estratte dall'omonimo dataset[2], in cui sono descritti i voli in orario, in ritardo, cancellati e dirottati sulla base dei rapporti mensili sui viaggi aerei. I dati presenti sono stati raccolti dal dipartimento dei trasporti degli USA e sono relativi ai voli del 2015.

[2] `https://www.kaggle.com/usdot/flight-delays`

Per quanto riguarda l'analisi dei dati che descrivono i viaggi aerei, si fa riferimento allo schema di fatto riportato nella Figura 5.2. Esso è caratterizzato dalle dimensioni e dal fatto considerati nella valutazione dell'approccio olistico e, di conseguenza, consente di rappresentare ed eseguire le stesse query.

Ad ogni volo sono associate le misure "cancellation" e "deviation", che possono avere valore pari a 0 o a 1 e che indicano rispettivamente la cancellazione o il dirottamento. Un'operazione di aggregazione consente di calcolare, ad esempio, il numero di voli dirottati o la media di voli cancellati. Le dimensioni considerate sono tre: aeroporto di partenza (che prevede i livelli paese, regione, stato, città e aeroporto), data del volo (nella cui gerarchia sono presenti gli attributi dimensionali stagione e mese) e compagnia aerea (caratterizzata da un unico livello).

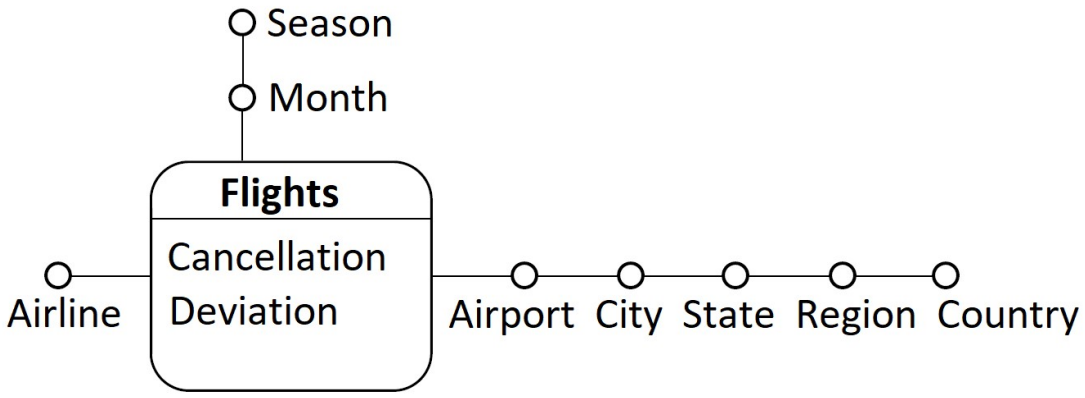

Figura 5.2: DFM relativo al cubo flight delays

La modellazione logica ROLAP corrispondente al DFM mostrato prevede una fact table "flights" e tre dimension table "airline", "airport" e "date". Non è però possibile ottenerle direttamente dai file CSV (*Comma Separated Values*) presenti nel dataset originale. Essi non contengono infatti gli attributi stagione e regione, utilizzati nel group by set e nelle clausole di selezione delle query eseguite per valutare l'approccio originale. Inoltre, non tutti i record contenuti in *flights.csv* fanno riferimento ad aeroporti e a compagnie aeree presenti nei file corrispondenti a queste dimensioni. Questo non è ammissibile perché, nell'ambito dei database relazionali, equivale ad una violazione del vincolo di integrità referenziale [9].

Per inizializzare il cubo multidimensionale Flight Delays, è stato quindi necessario eseguire una serie di operazioni di preprocessing sul dataset di partenza:

- Tramite una semplice scansione sequenziale dei file *airline.csv* e *airports.csv*, sono stati raccolti tutti i possibili valori degli identificativi di compagnie aeree e aeroporti. Questi hanno permesso di eliminare i record di *flights.csv* associati a id non noti. In questo modo, nella fact table ottenuta importando questo file è garantito il rispetto del vincolo di integrità referenziale.

- Per quanto riguarda la dimensione che fa riferimento alla data, nel dataset originale è presente soltanto il mese e quest'ultimo è contenuto direttamente dalla tabella principale *flights.csv*. Tuttavia, il valore della stagione meteorologica può essere calcolato deterministicamente in funzione del mese [44]. È stato quindi creato un nuovo file *date.csv* dove ogni mese è associato alla corrispondente stagione, in modo da arricchire il dataset.

- Tutti gli aeroporti presenti in *airports.csv* si trovano all'interno degli USA. Pur non avendo a disposizione la loro regione, i cui possibili valori sono "Northeast", "Midwest", "West", "South" e "United States Territories", essa può essere determinata univocamente a partire dalla stato. Così facendo, ad ogni record presente nel file è stata aggiunta la regione associata.

Le operazioni descritte sono state svolte in modo automatico grazie ad uno script Python. Una volta completate, i record presenti nei file *flights.csv*, *airline.csv*, *airport.csv* e *date.csv* corrispondono rispettivamente al contenuto della fact table e delle varie dimension table. È stato quindi possibile popolare le tabelle dello schema a stella, selezionando dai file le colonne interessanti per l'analisi e impostando i vari vincoli di chiave primaria (nelle DT e nella FT) e di chiave importata (per tutte le colonne della FT contenenti la chiave di una DT).

Il buon esito della fase di preprocessing è stato verificato eseguendo sul cubo la query che richiede di calcolare la probabilità di cancellazione di un volo raggruppando i risultati per regione e stagione. Il lavoro di riferimento riporta i risultati completi e questi corrispondono a quanto ottenuto interrogando il sistema con SQL Developer, un IDE che consente di eseguire query SQL su database Oracle.

5.2 Metriche di valutazione

Nelle attività sperimentali eseguite a supporto della presentazione dell'approccio olistico [47], l'efficacia del sistema è stata valutata utilizzando come metrica la qualità dei discorsi generati. Si ricorda che, indicando con $\beta(a,t)$ la distribuzione di probabilità con cui è modellata la percezione che l'utente ha di un aggregato a dopo aver ascoltato il discorso t, la qualità di t equivale alla media delle probabilità che l'utente assegna ai reali risultati della query in seguito all'output vocale t.

$$quality(t) = \sum_{a \in q.aggs} P(a(D)|\beta(a,t)) * \frac{1}{|q.aggs|}$$

La qualità è una misura efficace per valutare un discorso in termini relativi, ossia confrontandolo con un altro, ma non in termini assoluti. Ad esempio, si consideri un output vocale caratterizzato da un valore di qualità pari a 0.25. Per sapere "quanto" esso è in grado di approssimare correttamente il risultato, sarebbe necessario conoscere i parametri dell'algoritmo (fattore usato per calcolare la deviazione standard della normale e fattore che determina l'ampiezza dell'intervallo di approssimazione utile al calcolo della probabilità). Quindi, se si restituisce solo un valore di qualità, l'utente non riesce a capire se la soluzione è valida.

Si ipotizzi ora l'esecuzione di tre interrogazioni successive, che restituiscono output vocali caratterizzati rispettivamente da qualità pari a 0.22, 0.18 e 0.31. Confrontando i discorsi, l'utente è in grado di notare che l'ultima soluzione è più valida delle precedenti, ma non può affermare che lo sia in senso assoluto.

Un altro limite della misura di qualità è costituito dal fatto che il suo valore dipende direttamente dai parametri dell'algoritmo. Ad esempio, considerando lo stesso discorso e il medesimo risultato da descrivere, distribuzioni normali con deviazioni standard differenti restituiscono valori di qualità diversi.

A valle di queste considerazioni, la qualità è stata affiancata da un'altra metrica di valutazione dei risultati. Si è scelto di utilizzare l'ARE (*Average Relative Error*) che, data una serie di misure, è definito come la media dei rapporti tra gli errori assoluti e i valori reali (ossia la media degli errori relativi). A sua volta, l'errore assoluto equivale allo scostamento tra valore misurato e valore reale [28].

Facendo riferimento alla vocalizzazione di risultati OLAP e indicando con $M(a,t)$ la media della distribuzione di probabilità che modella la percezione dell'utente riguardo ad un aggregato a dopo aver ascoltato il discorso t, l'errore relativo medio che descrive l'output vocale t equivale alla media degli scostamenti relativi tra $M(a,t)$ e il valore reale $a(D)$ degli aggregati presenti nel risultato.

$$ARE(t) = \sum_{a \in q.aggs} \frac{||a(D) - M(a,t)||}{||a(D)||} * \frac{1}{|q.aggs|}$$

Ad esempio, si consideri una query che richiede di calcolare il salario medio dei lavoratori a metà carriera raggruppando i dati per regione di provenienza. I possibili valori dell'attributo regione sono Northeast, Midwest, West e South e, di conseguenza, il risultato comprende quattro aggregati. Ipotizzando che l'output vocale sia *"Si considerano i laureati provenienti da qualsiasi college e con un salario iniziale pari a qualsiasi cifra. I risultati sono raggruppati per regione. 80mila è la media dei salari a metà carriera. Il valore aumenta del 50% per i laureati provenienti dal Northeast."* e che i risultati reali siano quelli riportati nella colonna $a(D)$ della Figura 5.3, l'errore relativo medio è calcolato come mostrato.

a	Regione	$M(a)$	$a(D)$	$eA = \|\|a(D) - M(a,t)\|\|$	$eR = eA/\|\|a(D)\|\|$
1	Northeast	120000	115000	5000	0.043
2	Midwest	66667	71000	4333	0.061
3	West	66667	80000	13333	0.167
4	South	66667	56000	10667	0.190
					$ARE = 0.115$

Figura 5.3: Esempio di calcolo dell'errore relativo medio

Si può notare che, rispetto alla qualità, l'ARE presenta i seguenti vantaggi:

- Restituisce un valore indipendente dai parametri dell'algoritmo, ma che è influenzato solo dal risultato reale e dalla media della distribuzione predetta.
- Esprime il risultato in termini assoluti, mettendo a disposizione dell'utente una misura "canonica" con cui quantificare la bontà dell'approssimazione.

Inoltre, dato che l'implementazione della cache descritta nella Sezione 4.1 fa sì che venga memorizzato un unico elemento per ogni aggregato, se il dataset è mantenuto completamente in memoria il loro valore corrisponde a quello reale degli aggregati. In tal caso, è possibile calcolare l'errore relativo medio grazie ad un confronto tra i valori assegnati dalla cache agli aggregati e i corrispondenti valori medi della distribuzione normale modellata dal discorso considerato.

Per quanto riguarda invece le metriche usate per la valutazione dell'efficienza, sono stati eseguiti test finalizzati a quantificare sia il tempo necessario per l'inizializzazione del sistema sia il tempo impiegato per la risoluzione di una query.

5.3 Risultati dei test di efficacia

I test di efficacia eseguiti sul cubo Foodmart Sales hanno consentito di valutare i risultati ottenuti, in termini di qualità e di errore relativo medio, confrontando l'implementazione "base" dell'approccio con la sua variante "completa" (introdotta nella Sezione 4.3.3). In seguito ad alcune prove sperimentali, sono stati scelti i seguenti parametri per la configurazione dell'algoritmo di vocalizzazione:

- *nRefinements = 3*, corrispondente al massimo numero di refinement che possono essere contenuti in un output vocale.
- *nSignificantDigits = 2*, che indica il numero di cifre significative mantenute nell'approssimazione della baseline e dei fattori usati nei refinement.
- *standardDeviationFactor = 0.1*, ossia il fattore utilizzato per ottenere il valore della deviazione standard a partire da quello della baseline.
- *probabilityRangeFactor = 0.1*, corrispondente al fattore che consente di determinare l'intervallo di approssimazione usato nel calcolo della probabilità.
- *maxAggregates = 1500*, che indica il massimo numero di aggregati che possono essere contenuti nel risultato di una query accettata dal sistema.
- *maxChildren = 50*, ossia il massimo numero di figli di un qualsiasi nodo nell'albero di ricerca generato dall'algoritmo.

Nella sessione OLAP svolta su Foodmart sono state eseguite una serie di query finalizzate a valutare il costo medio in negozio (misura "store cost"). Le clausole di selezione e i group by set specificati considerano le dimensioni prodotto (con gli attributi categoria e sottocategoria), data (utilizzando i livelli della gerarchia trimestre e mese) e negozio (attributo dimensionale città del negozio).

La Figura 5.4, riportata di seguito, mostra i risultati ottenuti con le query eseguite. Il primo aspetto che emerge dai grafici è la correlazione negativa tra qualità e errore relativo medio, che conferma ciò che si può dedurre dalla loro definizione: un discorso con maggiore qualità riesce ad avvicinare di più la percezione dell'utente al risultato, minimizzando l'errore. Inoltre, i risultati evidenziano il miglioramento raggiunto grazie alla variante completa dell'approccio. Essa permette di raggiungere valori di qualità maggiori e errori minori in tutte le query, con differenze che variano in base a clausole di selezione e group by set.

Per quanto riguarda invece i test di efficacia svolti su Flight Delays, è stata replicata la sessione di interrogazioni eseguita nell'ambito della presentazione dell'approccio originale [47]. Questa ha reso possibile un confronto tra la versione realizzata e quella esistente. Affinché esso fosse veritiero, l'algoritmo è stato configurato con gli stessi parametri usati nell'implementazione originale (*nSignificantDigits = 1*, *standardDeviationFactor = 0.5* e *probabilityRangeFactor = 0.5*).

Su questo secondo cubo multidimensionale, le query eseguite si pongono l'obiettivo di calcolare la probabilità di cancellazione dei voli, corrispondente alla media di voli cancellati, considerando le dimensioni di analisi aeroporto (al livello di granularità definito dalla regione), data (con l'attributo dimensionale stagione) e compagnia aerea (con riferimento all'omonimo livello della gerarchia).

Nella Figura 5.5 sono presenti i grafici contenenti i risultati ottenuti grazie alle query eseguite su Flight Delays. In particolar modo, essi riportano un confronto tra l'efficacia dell'approccio originale, quella della versione base e quella della variante completa. Dato che l'errore relativo medio è stato introdotto in questo lavoro di tesi, per la valutazione dell'implementazione originale sono disponibili solamente i valori di qualità. Fa eccezione un'interrogazione specifica, per cui viene fornito anche l'output vocale corrispondente ed è quindi possibile calcolare l'errore.

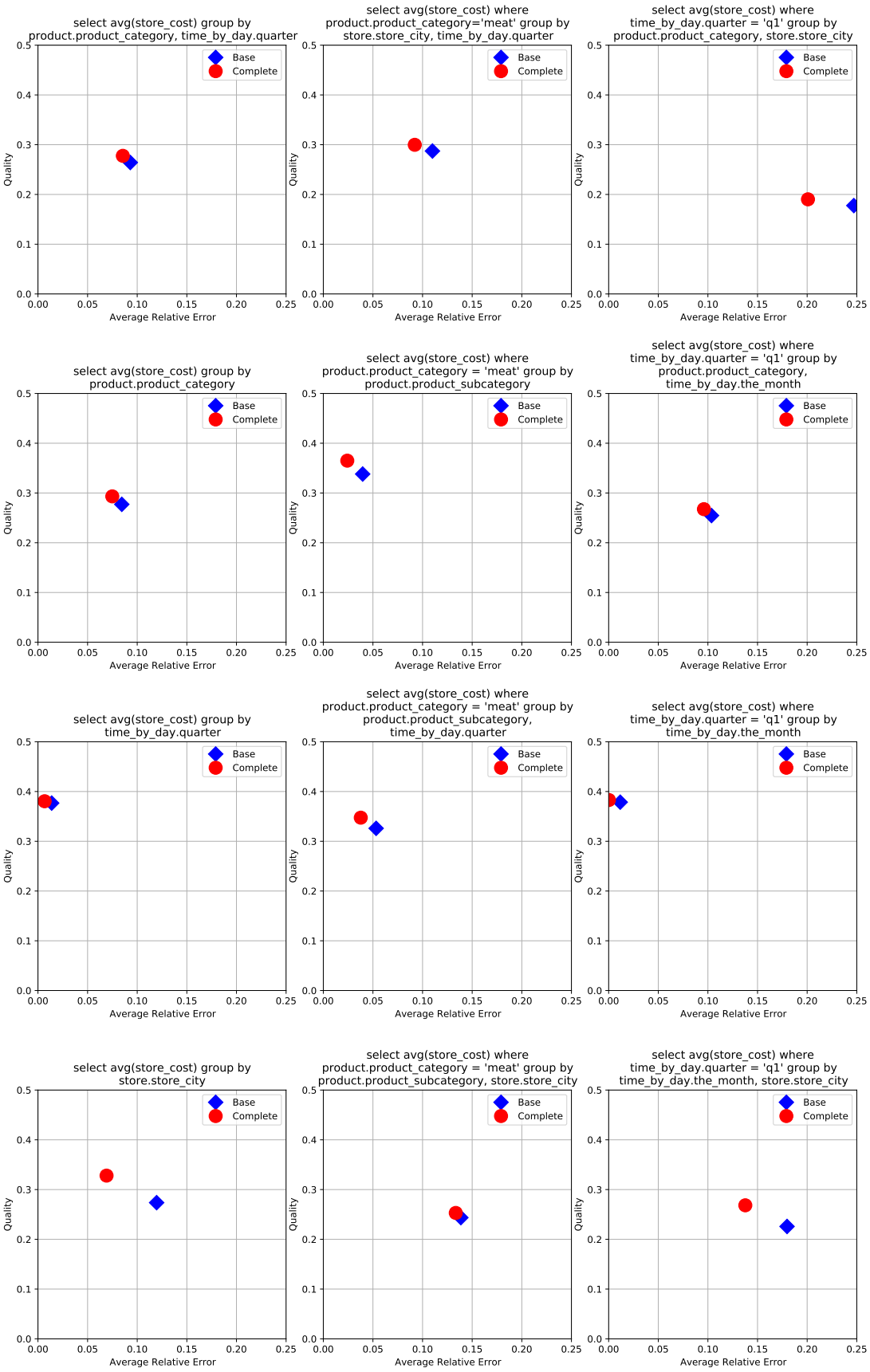

Figura 5.4: Test di efficacia sul cubo foodmart sales

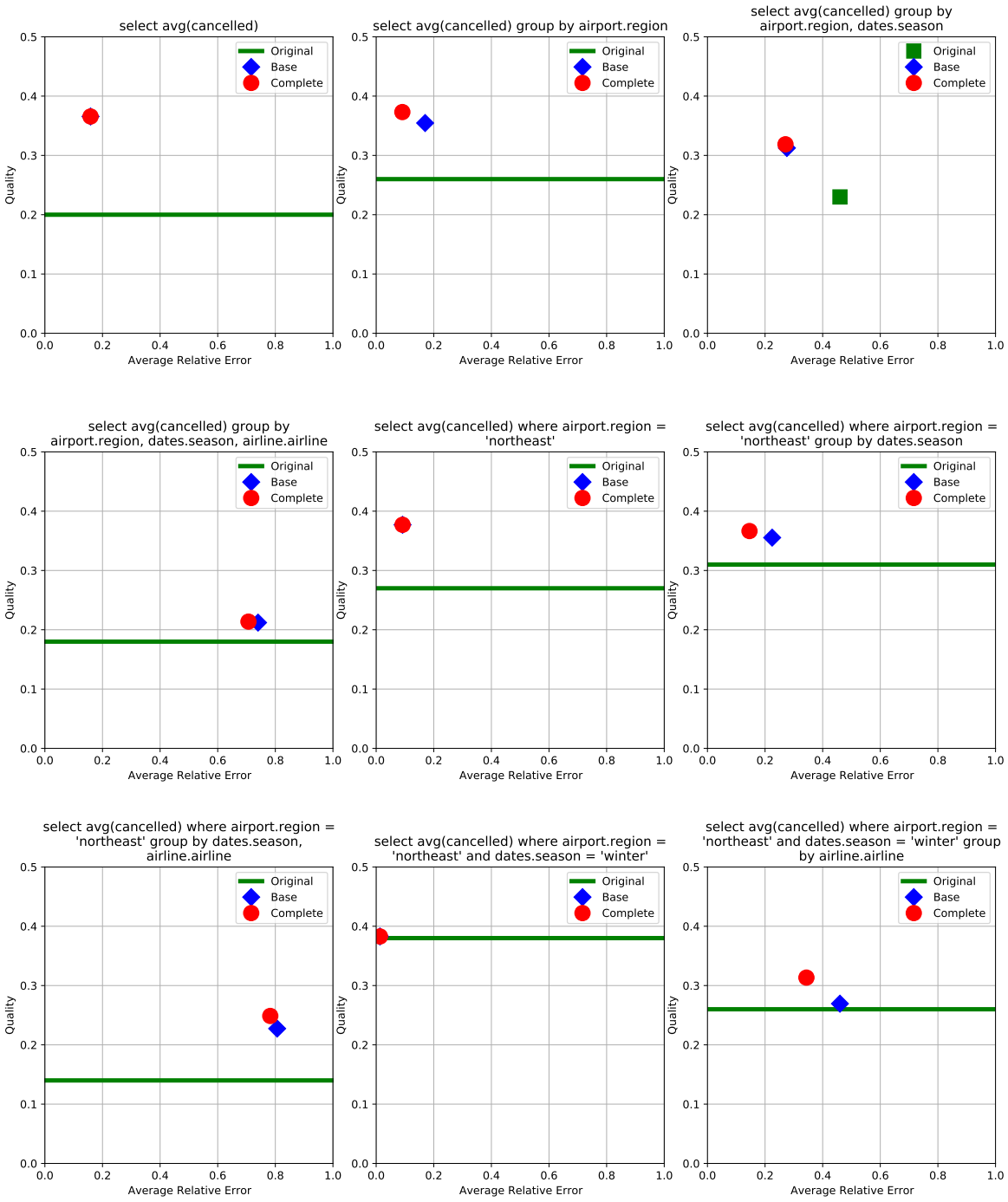

Figura 5.5: Test di efficacia sul cubo flight delays

Si consideri, ad esempio, la query "select avg(cancelled) group by airport.region, dates.season" eseguita sul cubo Flight Delays. Il grafico corrispondente ai suoi risultati, presente nella Figura 5.5, mostra i miglioramenti portati dalla nuova implementazione rispetto all'approccio olistico originale. La variante completa e quella base generano invece valori di qualità e di errore molto simili tra loro. Questo significa che i refinement migliori sono equamente distribuiti tra le dimensioni aeroporto e data e, perciò, non è evidente il vantaggio garantito dall'eliminazione del vincolo che associa ogni livello dell'albero soltanto a una dimensione.

Tutte le altre query eseguite su Flight Delays non riportano l'errore relativo medio ottenuto con l'implementazione originale, perché non sono disponibili gli output vocali associati. I risultati originali sono quindi rappresentati mediante una retta (che indica la qualità) e mostrano chiaramente il miglioramento garantito dalla generazione dei discorsi candidati basata sul contenuto della cache.

Ci si concentri infatti sulle query "select avg(cancelled)" e "select avg(cancelled) where airport.region ='northeast' and dates.season = 'winter'". Il loro group by set non comprende nessun attributo e, quindi, l'output vocale corrispondente è formato soltanto dalla baseline. L'implementazione base e quella completa ottengono infatti i medesimi risultati, mentre la versione originale restituisce la stessa qualità nel secondo caso e un valore decisamente inferiore nel primo. Quest'ultimo è dovuto alla selezione di un valore non ottimale come baseline, situazione che non si può verificare se si usa la cache per la generazione dell'albero di ricerca.

5.4 Risultati dei test di efficienza

I test di efficienza eseguiti sui cubi Foodmart Sales e Flight Delays analizzano i tempi di esecuzione del sistema. Per ottenere risultati coerenti con le attività di valutazione dell'efficacia svolte in precedenza, l'algoritmo di vocalizzazione è stato configurato utilizzando gli stessi parametri riportati nella Sezione 5.3.

Inoltre, tutti i test di efficienza sono stati svolti utilizzando il sistema in modo autonomo e quindi sfruttando l'interfaccia a riga di comando. Questo ha consentito di ottenere un'effettiva misura del tempo, in cui non sono considerate le eventuali latenze introdotte, ad esempio, da un ritardo nella comunicazione di rete.

La fase di inizializzazione, in cui si popola la cache, è onerosa e richiede circa 20 minuti per Foodmart Sales e quasi 30 per Flight Delays (contiene un maggior numero di record). Questi valori sono stati misurati accedendo in VPN ai server dell'università che contengono i database. I tempi potrebbero diminuire eseguendo la procedura nella rete universitaria, o adottando strutture di ottimizzazione quali indici e viste materializzate. È importante considerare che l'inizializzazione avviene soltanto all'avvio e quindi, in un contesto web o in un assistente vocale, l'overhead è accettabile perché il sistema è destinato a rimanere attivo a lungo.

Facendo riferimento all'esecuzione delle query, è stata valutata innanzitutto la latenza che, per rispettare la soglia di interattività, deve essere inferiore a 500 ms [33, 40]. Essa può corrispondere sia al tempo trascorso tra l'invio dell'input e l'inizio dell'emissione del discorso sia a quello che divide una frase dell'output vocale dal suo prosieguo. L'approccio olistico, per essere interattivo, interrompe e riprende ripetutamente le proprie iterazioni. Infatti, è stata sempre misurata una latenza trascurabile e impercettibile (pochi millisecondi, spesso circa 1).

Il tempo complessivo di risposta ad una query, misurato dall'invio dell'input dell'utente fino alla conclusione dell'output vocale, è stato sempre inferiore a 32 secondi (mantenendo il numero di refinement pari a 3). Si tratta di risultati in linea con il tempo impiegato dagli assistenti vocali per rispondere a interrogazioni complesse e in grado di rispettare i limiti di memoria dell'utente [32].

Nelle Figure 5.6 e 5.7 sono riportati i risultati dei test di efficienza eseguiti sui cubi Foodmart Sales e Flight Delays. Tendenzialmente, il tempo necessario aumenta di qualche secondo nell'implementazione completa. Questo perché la versione base prevede massimo un refinement per ogni attributo del group by set, e i raggruppamenti delle query eseguite considerano al più due attributi.

Ad esempio, si consideri la query su Foodmart "select avg(store_cost) group by store.store_city ". Con l'implementazione completa il tempo cresce di 5 secondi, ma l'errore diminuisce da 0.12 a 0.06 e aumenta il contenuto informativo trasmesso. Un'eccezione è costituita dalla query "select avg(cancelled) group byairport.region, dates.season, airline.airline" su Flight Delays, in cui le due versioni restituiscono tre refinement con tempistiche simili (e errore minore nella variante completa).

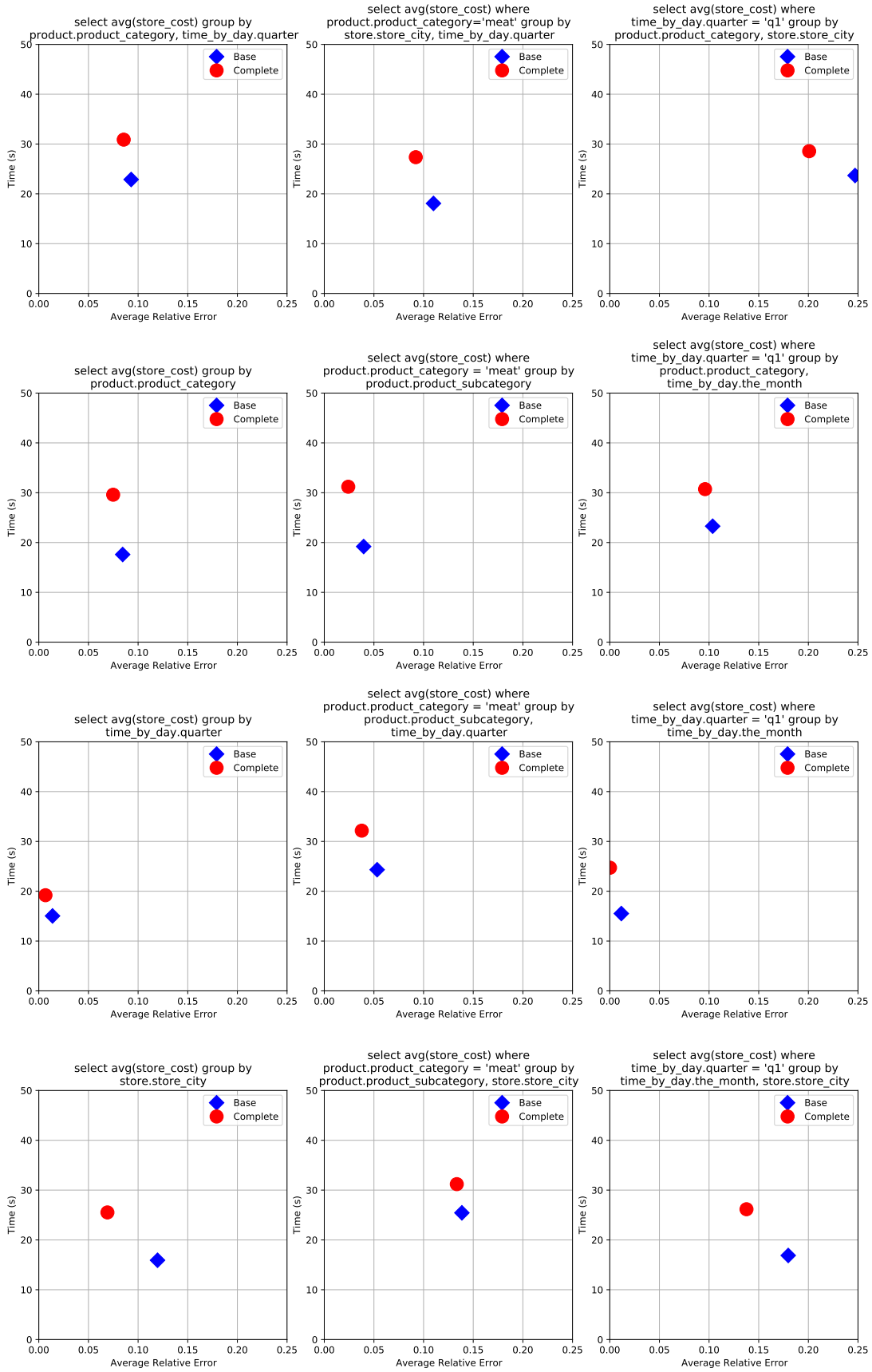

Figura 5.6: Test di efficienza sul cubo foodmart sales

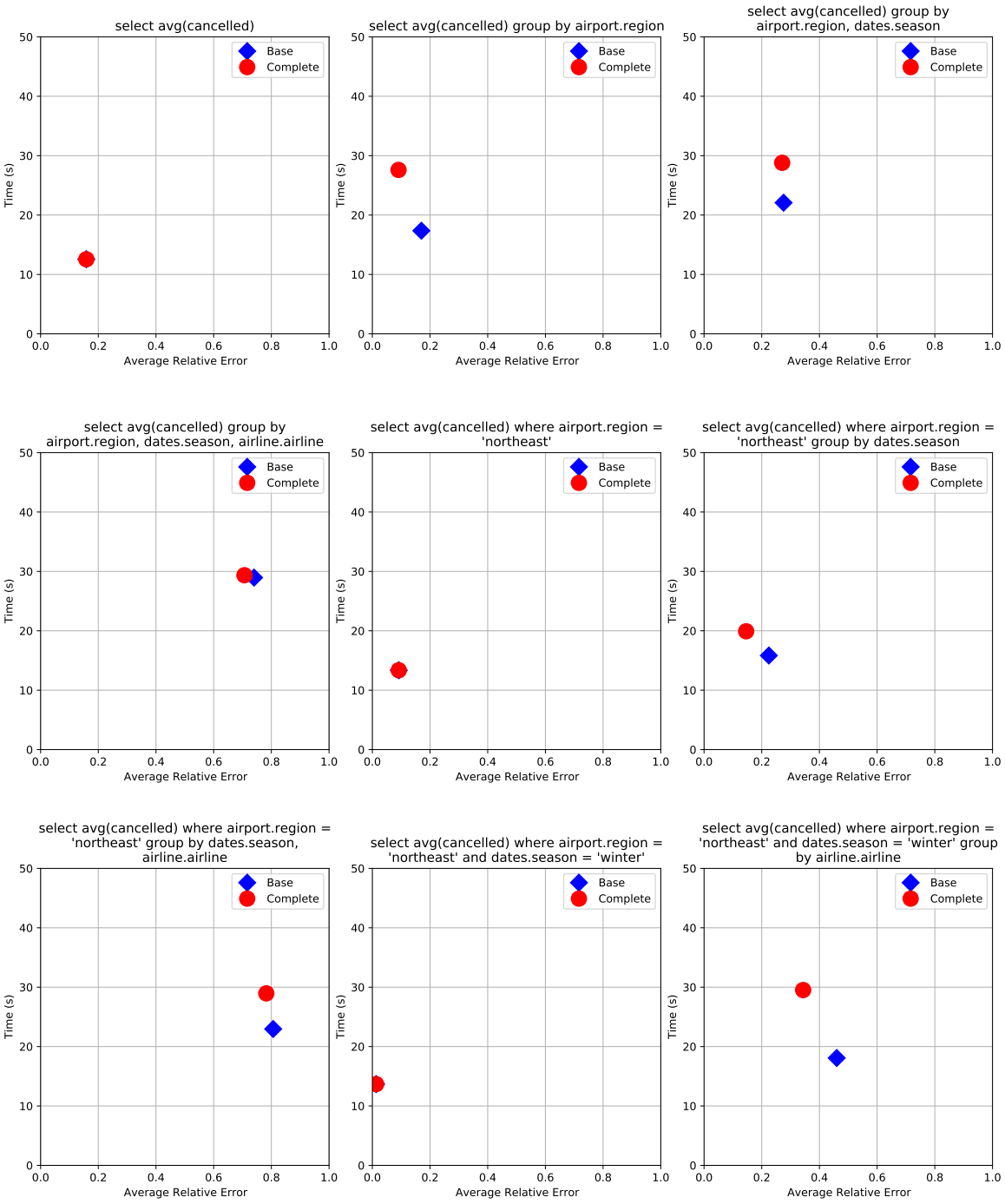

Figura 5.7: Test di efficienza sul cubo flight delays

Capitolo 6

Conclusioni e sviluppi futuri

Questa tesi si colloca nell'ambito della Conversational Business Intelligence e, in particolar modo, esplora il problema della data vocalization applicato ai risultati di query OLAP. L'obiettivo prefissato è infatti rappresentato dallo studio e dall'implementazione di una una tecnica per la vocalizzazione di risultati multidimensionali, attività finalizzate all'estensione del framework conversazionale COOL.

È stato quindi approfondito un approccio olistico alla vocalizzazione, che combina la valutazione delle query OLAP e la trasformazione dell'output in formato vocale con l'obiettivo di minimizzare gli overhead computazionali e di massimizzare il contenuto informativo trasmesso dal risultato. L'implementazione di questo metodo ha consentito di sviluppare un sistema in grado di:

- Essere eseguito su un qualsiasi cubo multidimensionale caratterizzato da una modellazione logica ROLAP, a valle di una rapida configurazione.
- Risolvere query GPSJ generiche, limitate soltanto dalla possibilità di analizzare un'unica misura e dal mancato supporto agli operatori min e max.
- Restituire output con una struttura semplice e una lunghezza ridotta, efficaci nonostante la scarsa capacità umana di memoria a breve termine.
- Garantire l'interattività nella comunicazione con l'utente, offrendo risposte caratterizzate da una latenza minima (ben inferiore alla soglia di 500 ms).

I test eseguiti sul sistema hanno evidenziato i buoni risultati ottenuti grazie al lavoro svolto, sia in termini di efficacia sia in termini di efficienza. Inoltre, gli accorgimenti adottati per automatizzare e ottimizzare l'approccio originale hanno permesso di migliorare l'efficacia dei risultati raggiunti (misurata da metriche in grado di quantificare il contenuto informativo trasmesso dall'output).

L'integrazione con il framework COOL, eseguita a conclusione del progetto presentato in questa tesi, ha consentito di combinare i vantaggi garantiti da un'interfaccia conversazionale e quelli derivati dalla vocalizzazione dei risultati. L'applicazione risultante costituisce infatti un esempio di soluzione conversazionale "end-to-end", in cui input e output sono espressi in linguaggio naturale.

Sebbene l'obiettivo della tesi possa essere considerato raggiunto, il contributo presentato può costituire un valido spunto per ulteriori lavori che si pongono l'obiettivo di esplorare delle soluzioni alternative o di migliorare l'approccio. In particolare, i principali sviluppi futuri individuati sono i seguenti:

- *Utilizzo di un algoritmo alternativo per la prioritizzazione*

  L'approccio olistico si basa sulla ricerca del miglior output vocale all'interno di uno spazio contenente tutti i possibili discorsi candidati. Quest'ultimo non può essere esplorato completamente, a causa dei vincoli di tempo stringenti, ma viene usata una tecnica di prioritizzazione degli output vocali.

  UCT [27] è l'algoritmo di ottimizzazione scelto dall'approccio olistico, e utilizzato in questa implementazione. Un possibile sviluppo è rappresentato dalla ricerca e dall'applicazione di un altro algoritmo per la prioritizzazione, che consentirebbe l'introduzione di una variante dell'approccio. Alternative valide sono rappresentate dagli stessi algoritmi appartenenti alla famiglia della MCTS configurati appositamente per il problema dato [24] e da tecniche di deep learning, come ad esempio l'algoritmo Deep Q-Learning [21], che è già stato utilizzato in alternativa alle tecniche della MCTS.

- *Scelta di una nuova metrica per la valutazione dell'output vocale*

  Gli algoritmi di prioritizzazione dei discorsi, come ad esempio UCT, sono accomunati da una caratteristica: assegnano a un output vocale un valore che

indica la sua "bontà". Nell'approccio olistico questo corrisponde alla qualità, misurata in base alla modellazione della percezione dell'utente riguardo al risultato. Potrebbe essere scelta una metrica differente da abbinare all'algoritmo di prioritizzazione, in modo da esplorare una soluzione alternativa. Un esempio è la misura restituita dall'errore relativo medio, già utilizzata nell'ambito della valutazione dell'efficacia dei risultati ottenuti.

- *Utilizzo di una tecnica per l'estrazione automatica dei metadati*

  Il funzionamento del sistema si basa sulla presenza di metadati in grado di descrivere la struttura del cubo multidimensionale e di rappresentare i suoi elementi nell'output vocale. La fase di configurazione potrebbe essere semplificata estraendo queste informazioni direttamente dal data warehouse. In letteratura esistono lavori lavori finalizzati a comprendere automaticamente la struttura di cubi multidimensionali [11, 12]. Per ottenere le varie parti del discorso utili alla generazione dell'output vocale, potrebbero invece essere applicate delle tecniche di NLP ai dati disponibili.

- *Applicazione di un metodo di campionamento a cubi di dimensione eccessiva*

  Il meccanismo di caching attuale prevede che venga mantenuto in memoria il contenuto di tutti i record della fact table e delle dimension table presenti nel cubo multidimensionale analizzato. Questo costituisce un evidente limite, poiché non consente l'utilizzo del sistema se lo spazio occupato dal cubo eccede la memoria disponibile nell'ambiente di esecuzione.

  Per migliorare questo aspetto, potrebbe essere scelto un metodo di campionamento e applicato ai record presenti nel cubo multidimensionale. In questo modo, se non fosse possibile mantenere interamente in memoria la fact table e le dimension table, il sistema potrebbe eseguire automaticamente il campionamento. È importante notare che, essendo l'approccio olistico basato su una serie di valutazioni successive di campioni, sarebbe possibile applicarlo allo stesso modo ed ottenere risultati altrettanto validi.

Bibliografia

[1] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28(5):793–819, 2019.

[2] Alok Aggarwal. The current hype cycle in artificial intelligence. *KDnuggets News*, 18, 2018.

[3] Mohammad Ahsanullah, BM Golam Kibria, and Mohammad Shakil. Normal distribution. In *Normal and Student´ st Distributions and Their Applications*, pages 7–50. Springer, 2014.

[4] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases-an introduction. *arXiv preprint cmp-lg/9503016*, 1995.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[6] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[7] Chun-houh Chen, Wolfgang Karl Härdle, and Antony Unwin. *Handbook of data visualization*. Springer Science & Business Media, 2007.

[8] Menal Dahiya. A tool of conversation: Chatbot. *International Journal of Computer Sciences and Engineering*, 5(5):158–161, 2017.

[9] CJ Date. Referential integrity. In *Proceedings of the seventh international conference on Very Large Data Bases-Volume 7*, pages 2–12, 1981.

[10] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

[11] Matteo Francia, Enrico Gallinucci, and Matteo Golfarelli. Towards conversational olap. In *DOLAP*, pages 6–15, 2020.

[12] Matteo Francia, Enrico Gallinucci, and Matteo Golfarelli. Cool: A framework for conversational olap. *Information Systems*, page 101752, 2021.

[13] Matteo Francia, Matteo Golfarelli, and Stefano Rizzi. A-bi+: a framework for augmented business intelligence. *Information Systems*, 92:101520, 2020.

[14] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(02n03):215–247, 1998.

[15] Matteo Golfarelli and Stefano Rizzi. View materialization for nested gpsj queries. In *DMDW*, page 10, 2000.

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

[17] Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The mathematical intelligencer*, 7(1):42–48, 1985.

[18] Ashish Gupta, Venky Harinarayan, and Dallan Quass. Aggregate-query processing in data warehousing environments. 1995.

[19] Sanda M Harabagiu and Vinay Chaudhri. Mining answers from text and knowledge bases. In *AAAI Spring Symposium Series, American*, 2002.

[20] Thomas Hermann, Andy Hunt, and John G Neuhoff. *The sonification handbook*. Logos Verlag Berlin, 2011.

[21] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[22] Wendy Holmes. *Speech synthesis and recognition*. CRC press, 2001.

[23] Matthew B Hoy. Alexa, siri, cortana, and more: an introduction to voice assistants. *Medical reference services quarterly*, 37(1):81–88, 2018.

[24] T. Imagawa and Tomoyuki Kaneko. Enhancements in monte carlo tree search algorithms for biased game trees. *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 43–50, 2015.

[25] Ruoming Jin, Leonid Glimcher, Chris Jermaine, and Gagan Agrawal. New sampling-based estimators for olap queries. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 18–18. IEEE, 2006.

[26] Shantanu Joshi and Christopher Jermaine. Materialized sample views for database approximation. *IEEE Transactions on Knowledge and Data Engineering*, 20(3):337–351, 2008.

[27] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[28] X Rong Li and Zhanlue Zhao. Relative error measures for evaluation of estimation algorithms. In *2005 7th International Conference on Information Fusion*, volume 1, pages 8–pp. IEEE, 2005.

[29] Xiaolei Li, Jiawei Han, Zhijun Yin, Jae-Gil Lee, and Yizhou Sun. Sampling cube: a framework for statistical olap over sampling data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 779–790, 2008.

[30] Gabriel Lyons, Vinh Tran, Carsten Binnig, Ugur Cetintemel, and Tim Kraska. Making the case for query-by-voice with echoquery. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2129–2132, 2016.

[31] Patrick Marcel, Rokia Missaoui, and Stefano Rizzi. Towards intensional answers to olap queries for analytical sessions. In *Proceedings of the fifteenth international workshop on Data warehousing and OLAP*, pages 49–56, 2012.

[32] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[33] Robert B Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, 1968.

[34] Solomon Negash and Paul Gray. Business intelligence. In *Handbook on decision support systems 2*, pages 175–193. Springer, 2008.

[35] Navneet Potti and Jignesh M Patel. Daq: a new paradigm for approximate query processing. *Proceedings of the VLDB Endowment*, 8(9):898–909, 2015.

[36] TV Raman. *Audio system for technical readings*. Springer, 1998.

[37] Sunita Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, volume 2000, pages 307–316. Citeseer, 2000.

[38] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment*, 13(12):2747–2759, 2020.

[39] Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan.

Natural language querying of complex business intelligence queries. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1997–2000, 2019.

[40] Ben Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[41] Niculae Stratica, Leila Kosseim, and Bipin C Desai. Nlidb templates for semantic parsing. *Natural language processing and information systems*, 2003.

[42] Christina Sun et al. *A natural language interface for querying graph databases*. PhD thesis, Massachusetts Institute of Technology, 2018.

[43] Virginia Teller. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. *Computational Linguistics*, 26(4):638–641, 2000.

[44] Piero Tempesti. *Il calendario e l'orologio*. Gremese editore, 2006.

[45] Immanuel Trummer. Data vocalization with cicerodb. In *CIDR*, 2019.

[46] Immanuel Trummer, Mark Bryan, and Ramya Narasimha. Vocalizing large time series efficiently. *Proceedings of the VLDB Endowment*, 11(11):1563–1575, 2018.

[47] Immanuel Trummer, Yicheng Wang, and Saketh Mahankali. A holistic approach for query evaluation andresult vocalization in voice-based olap. In *Proceedings of the 2019 International Conference on Management of Data*, pages 936–953, 2019.

[48] Immanuel Trummer, Jiancheng Zhu, and Mark Bryan. Data vocalization: optimizing voice output of relational data. *Proceedings of the VLDB Endowment*, 10(11):1574–1585, 2017.

[49] Manika Tyagi. Natural language interface to databases: A survey. *International Journal of Science and Research*, 3(5):1443–1445, 2014.