Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Scienze e Tecnologie Informatiche

UNO STRUMENTO PER LA GENERAZIONE PARAMETRICA DI CARICHI DI LAVORO OLAP

Tesi di Laurea in Sistemi Informativi Avanzati

Relatore:
Chiar.mo Prof.
Stefano Rizzi

Presentata da:
Luca Spadazzi

Sessione I
Anno Accademico 2012-2013

"L'università non farà mai più parte del mio futuro."

(Luca Spadazzi, 2005)

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

In informatica, OLAP (On-Line Analytical Processing) è un corpus di tecniche usato per rispondere rapidamente ad interrogazioni analitiche multidimensionali. OLAP fa parte della Business Intelligence, una disciplina che comprende anche reportistica, data mining e benchmarking.

Gli strumenti OLAP permettono agli utenti di analizzare interattivamente dati multidimensionali da diverse prospettive. I tre operatori OLAP principali sono *roll-up*, *drill-down* e *slice and dice*:

- l'operatore di roll-up serve ad aggregare i dati, accumulandoli su una o più dimensioni (vedi Cap. 1);
- l'operatore di drill-down, al contrario, esplode i dati aumentandone il livello di dettaglio;
- il terzo operatore seleziona un sottoinsieme dei dati (slice) e lo analizza sotto diversi aspetti (dice).

I database configurati per le analisi OLAP organizzano i dati secondo un modello multidimensionale, e possono essere usati per formulare interrogazioni (o *query*) complesse, con ridotti tempi di risposta. La mole di dati da elaborare e sintetizzare rende necessaria l'adozione di tecniche di ottimizzazione, per aumentare l'efficienza delle analisi effettuate (ad esempio, scegliere quante e quali viste materializzare).

Il carico di lavoro di una sessione OLAP non è prestabilito: dipende dalle caratteristiche del cubo OLAP, dalle esigenze dell'utente e dalla confidenza che quest'ultimo ha con il sistema di analisi. Tuttavia, se si conoscesse il

carico di lavoro (ad esempio, tramite il log delle query elaborate dal sistema) sarebbe possibile migliorare l'esperienza dell'utente applicando tecniche di *query formulation support* [4] (per completare automaticamente una query) o *query recommendation* [2] (per suggerire la prossima query in base alle sessioni create in precedenza).

In mancanza di un log (o, più in generale, di un carico di lavoro reale) c'è bisogno di uno strumento per generare carichi di lavoro OLAP realistici: un simile strumento sarebbe utile sia nel campo della ricerca, sia per i prodotti commerciali. L'obiettivo di questa tesi è documentare la creazione di uno strumento per la generazione parametrica di carichi di lavoro OLAP. Inoltre, si vuole rendere questo generatore liberamente disponibile alla comunità di ricerca scientifica, perché possa essere adattato alle esigenze individuali di chi fa ricerca nell'ambito della Business Intelligence. Di conseguenza, questo obiettivo ha portato alla scelta di formalismi adatti a rendere portabile il generatore.

Struttura della tesi

Questa tesi sarà così strutturata:

- il capitolo 1 esporrà il contesto di studio e una breve panoramica di strumenti OLAP e benchmark;
- nel capitolo 2 ci si soffermerà sui requisiti funzionali del progetto di tesi;
- analisi e progettazione saranno esaminate nel capitolo 3;
- il capitolo 4 si occuperà dell'implementazione del progetto;
- il capitolo 5 discuterà i risultati dei test effettuati sul progetto, inclusi alcuni indicatori della similarità tra sessioni;
- il capitolo 6, infine, riguarderà le conclusioni.

Capitolo 1

Background

1.1 Modello multidimensionale

Il modello multidimensionale di dati è parte integrante di OLAP, ed impone la semplicità: è composto da cubi, fatti, dimensioni, gerarchie e misure. La semplicità del modello deriva dal dover rappresentare entità di business del mondo reale: gli analisti sanno quali fatti sono di interesse, quali dimensioni rendono significativi i dati e con quali gerarchie sono organizzate le dimensioni.

- Un *cubo* organizza i fatti che hanno la stessa struttura (ovvero, le stesse dimensioni e misure). I fatti di un cubo possono essere facilmente analizzati e visualizzati insieme.
- Un *fatto* è un evento che accade dinamicamente (come una vendita o un ricovero) e di cui si vuole registrare l'occorrenza, oltre ad alcuni aspetti numerici. I fatti sono descritti da un certo numero di dimensioni.
- Una *dimensione* è una proprietà fondamentale del fatto, e ne rappresenta una coordinata (p.e. le dimensioni di un fatto 'Ricovero' potrebbero essere *ospedale*, *giorno*, *paziente*, *diagnosi*).
- Una *gerarchia* organizza i dati a diversi livelli di aggregazione. Gli analisti rilevano un fenomeno in un particolare livello, esplodono i dati

per trovarne le cause e aggregano per verificarne l'impatto su larga scala. Esempio di gerarchia: città ⇒ regione ⇒ nazione.

- Una *misura* è un aspetto quantitativo di un fatto: ad esempio, di una vendita si possono registrare l'incasso, la quantità venduta e il prezzo unitario.

La figura 1.1 mostra un esempio di *Dimensional Fact Model* [3]: il fatto 'Vendita' è caratterizzato dalle misure 'Quantità Venduta', 'Prezzo di Vendita' e 'Prezzo di Acquisto'. Il fatto è descritto da tre dimensioni, ognuna radice di una gerarchia. Nella dimensione 'Prodotto' c'è una ramificazione: gli attributi dimensionali 'Fornitore' e 'Tipo Prodotto' sono in relazione molti-a-molti tra loro. Nella dimensione temporale troviamo una convergenza: tra 'Giorno' e 'Anno' ci sono due percorsi orientati di dipendenze molti-a-uno. Infine, l'attributo 'Negozio' ha come figlio un attributo descrittivo, 'Indirizzo': sugli attributi descrittivi non ha senso aggregare.

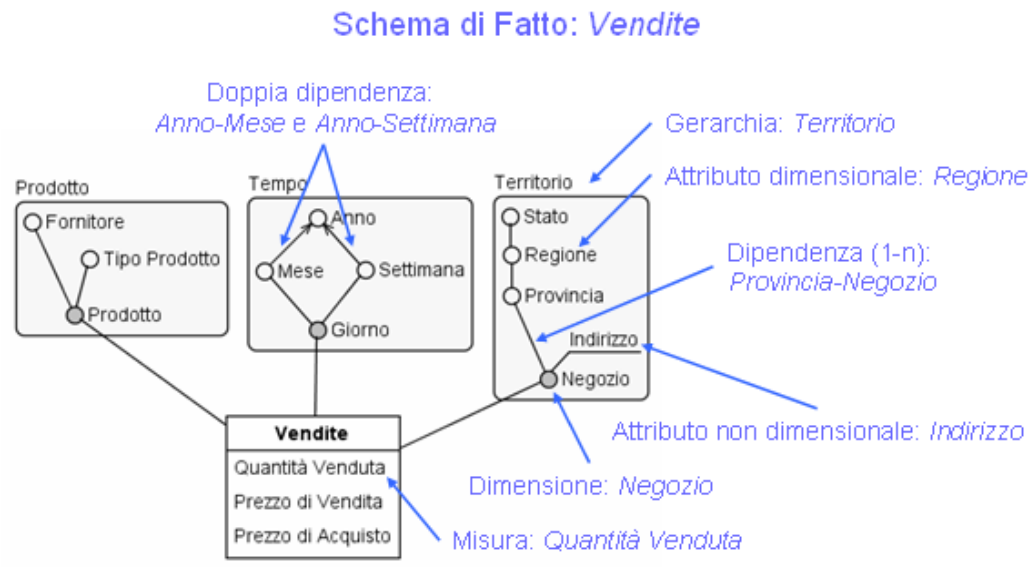

Figura 1.1: Il fatto 'Vendita' è descritto dalle dimensioni *Prodotto*, *Giorno* e *Negozio*; ogni dimensione è radice di una gerarchia.

1.2 Strumenti OLAP

Il primo prodotto ad elaborare query OLAP fu *Express*, rilasciato nel 1970 e acquisito da Oracle™ nel 1995. Il termine 'OLAP', tuttavia, risale al 1993, quando fu coniato da Edgar F. Codd (considerato il padre dei database relazionali). Il mercato OLAP conobbe una forte crescita verso la fine degli anni '90, con la nascita di decine di prodotti commerciali. Nel 1998, Microsoft™ rilasciò il suo primo server OLAP, Microsoft Analysis Services: questo ha portato una grande diffusione e adozione di tecnologie OLAP.

Le maggiori compagnie produttrici sono Microsoft (Microsoft Analysis Services), Oracle (Oracle Database OLAP Option, Essbase), IBM (Cognos TM1), SAP (SAP NetWeaver BW) e MicroStrategy (MicroStrategy Intelligence Server). Una sottosezione a parte è dedicata ad uno strumento OLAP open-source sviluppato da Pentaho: Pentaho Analysis Service - anche noto come *Mondrian*.

Mondrian è un server OLAP open-source, scritto in Java™: supporta il linguaggio di query MDX (*MultiDimensional eXpressions*) e le interfacce *olap4j* e *XML for Analysis*. Mondrian è in grado di elaborare velocemente le query sfruttando le tabelle aggregate di un RDBMS, traducendo dal linguaggio MDX al linguaggio SQL (*Structured Query Language*).

Mondrian utilizza un file di metadati XML per descrivere la struttura dello schema multidimensionale: troviamo i *cube* (fatti), le *dimension* (dimensioni), le *hierarchy* (gerarchie), i *level* (attributi dimensionali) e le *measure* (misure). Inoltre, i metadati specificano membri calcolati ed espressioni SQL per gestire attributi dimensionali e misure derivate (le suddette espressioni sono diversificate per interfacciarsi ai principali RDBMS - Access, Oracle, PostgreSQL ecc.).

Mondrian permette di esprimere costrutti avanzati come gerarchie condivise, attributi descrittivi e gerarchie ricorsive: altri costrutti, come archi multipli, attributi cross-dimensionali o convergenze [5], non sono direttamente esprimibili.

Nell'ottica di apertura verso il mondo open-source, il generatore di cari-

chi di lavoro OLAP sviluppato nel progetto di tesi accetta, per costruire lo schema multidimensionale, un file di metadati con specifica di Mondrian.

1.3 Benchmark: TPC

Transaction Processing Performance Council (TPC) [13] è un'azienda no-profit fondata nell'agosto 1998 per definire benchmark su database ed elaborazioni di transazioni, e per offrire alle industre dati obiettivi e verificabili sulle performance TPC. Attualmente i benchmark coprono carichi di lavoro OLTP (On-Line Transactional Processing), Web Commerce e DSS (*Decision Support System*).

Nell'aprile 1995 nacque TPC-D, il primo benchmark per il supporto alle decisioni. I benchmark esistenti consistevano di transazioni di leggero-medio peso, mentre TPC-D conteneva applicazioni per il supporto alle decisioni che richiedevano lunghe query su strutture dati grandi e complesse. Grazie al TPC-D, migliorarono i prodotti commerciali e aumentò la competizione: tuttavia, una delle lacune fu la mancata anticipazione di strutture aggregate (indici di join, viste materializzate ecc.), e la loro introduzione rese inutilizzabile il benchmark. Questo portò all'approvazione di altri due benchmark, TPC-H e TPC-R. Entrambi erano d'ausilio al supporto alle decisioni, e differivano nella conoscenza a priori del carico di lavoro assunto. Il TPC-H rappresentava un ambiente in cui i database administrator non sapevano quali query sarebbero state eseguite, e non era possibile ottimizzare il DBMS in questo senso; nel TPC-R, invece, questa informazione preliminare era presente, e grazie ad essa si potevano definire strutture aggregate. Un ulteriore benchmark, TPC-DS, fu sviluppato per unire gli aspetti positivi di TPC-H e TPC-R, e per essere più comprensibile in termini dei moderni DSS.

Il benchmark TPC-DS modella diversi aspetti - generalmente applicabili - di un DSS, incluse le query e la manutenzione dei dati. Il benchmark offre una valutazione rappresentativa delle performance del *System Under*

Test (sistema testato per la correttezza delle operazioni) come sistema di supporto alle decisioni general purpose.

I DSS illustrati da questo benchmark hanno le seguenti caratteristiche:

- esaminano grandi volumi di dati;
- rispondono a domande di business del mondo reale;
- eseguono query con complessità e requisiti operazionali variabili (ad hoc, di reportistica, OLAP, data mining);
- sono caratterizzati da grandi carichi di CPU e I/O;
- sono sincronizzati periodicamente con database sorgenti OLTP attraverso funzioni di manutenzione di database.

Un risultato del benchmark misura la *query throughput* (produttività delle query) e la performance di manutenzione dei dati per un dato hardware, sistema operativo e configurazione del DBMS, con un carico di lavoro di supporto alle decisioni controllato, complesso e multi-utente.

Capitolo 2

Requisiti funzionali

L'obiettivo del progetto è la creazione di uno strumento per la generazione di carichi di lavoro OLAP realistici: sessioni create casualmente - o con pochi vincoli - non simulano adeguatamente la navigazione di un cubo, e non possono essere usate per testare algoritmi di personalizzazione e *query recommendation*.

Il prodotto del generatore deve simulare un log di sessioni create in un'azienda, con tutte le query immesse ed elaborate. Il primo e fondamentale aspetto del generatore è la gestione dei profili: in un'azienda, gli utenti del sistema occupano diversi livelli della gerarchia e hanno obiettivi differenti, quindi il generatore deve creare sessioni parametrizzate per ogni profilo. Alcuni esempi di profilo potrebbero essere dirigente, responsabile di reparto, preside di facoltà o analista.

2.1 Input

L'utente del generatore, per creare le sessioni, deve fornire in input uno schema multidimensionale ed impostare i parametri globali e locali.

2.1.1 Schema multidimensionale

Per definire lo schema multidimensionale serve un file di metadati XML che descriva cubi, dimensioni, gerarchie e misure. Lo schema seguente riflette la semantica di Mondrian [7]:

```
<Schema>
        <Cube name="Sales">
                <Table name="sales_fact_1997"/>
                <Dimension name="Gender" foreignKey="customer_id">
                        <Hierarchy hasAll="true" allMemberName="All Genders"
                             primaryKey="customer_id">
                                <Table name="customer"/>
                                <Level name="Gender" column="gender"
                                    uniqueMembers="true"/>
                        </Hierarchy>
                </Dimension>
                <Dimension name="Time" foreignKey="time_id">
                        <Hierarchy hasAll="false" primaryKey="time_id">
                                <Table name="time_by_day"/>
                                <Level name="Year" column="the_year" type="
                                    Numeric" uniqueMembers="true"/>
                                <Level name="Quarter" column="quarter"
                                    uniqueMembers="false"/>
                                <Level name="Month" column="month_of_year"
                                    type="Numeric" uniqueMembers="false"/>
                        </Hierarchy>
                </Dimension>
                <Measure name="Unit Sales" column="unit_sales" aggregator="
                    sum" formatString="#,###"/>
                <Measure name="Store Sales" column="store_sales" aggregator=
                    "sum" formatString="#,###.##"/>
                <Measure name="Store Cost" column="store_cost" aggregator="
                    sum" formatString="#,###.00"/>
                <CalculatedMember name="Profit" dimension="Measures" formula
                    ="[Measures].[Store Sales] - [Measures].Store Cost]">
                <CalculatedMemberProperty name="FORMAT_STRING" value="
                    \$#,##0.00"/>
                </CalculatedMember>
        </Cube>
</Schema>
```

Il generatore deve effettuare un *parsing* del file di metadati per creare un cubo multidimensionale completo (con nomi di misure, dimensioni, gerarchie e livelli). Una delle dimensioni sarà impostata come la dimensione temporale,

sempre presente in uno schema di fatto. In un file di metadati XML è possibile definire più cubi (fatti), ognuno con le proprie caratteristiche: il generatore, durante il parsing dello schema, deve registrare i nomi dei cubi disponibili ed offrire all'utente una scelta.

Conoscere la struttura del cubo multidimensionale non è sufficiente per costruire un insieme di query: una parte fondamentale delle query OLAP è rappresentata dai predicati di selezione. Nel contesto di questo progetto, tutti i predicati creati avranno la seguente forma:

GERARCHIA.LIVELLO = ELEMENTO.

Mentre per *gerarchia* e *livello* basta elaborare il file di metadati, *elemento* richiede una conoscenza dei valori del livello della suddetta gerarchia nelle istanze dei fatti. Queste informazioni aggiuntive saranno estratte da alcune *dimension table*.

Nel modello logico ROLAP (Relational OLAP), le dimension table sono relazioni, corrispondenti alle dimensioni del cubo. Ogni tupla della relazione contiene una chiave primaria surrogata (generalmente un numero intero progressivo) e un valore testuale per ogni livello della gerarchia, con aggregazione crescente. Un esempio di tupla per la gerarchia 'CITTÀ':

112, BERGAMO, LOMBARDIA, ITALIA.

L'utente deve fornire al generatore un insieme di file CSV (Comma Separated Values), chiamati con i nomi delle gerarchie in maiuscolo: il generatore leggerà il file, scarterà il primo valore e - con gli altri - popolerà ogni livello della gerarchia in questione. Il valore testuale non sarà aggiunto se già presente tra i valori distinti. I livelli 'ALL' delle gerarchie (ovvero il massimo livello di aggregazione) rimarranno vuoti.

2.1.2 Parametri globali

I parametri globali che l'utente deve impostare riflettono alcune caratteristiche di tutte le sessioni da creare:

- numero dei profili: l'utente specifica per quanti profili vuole simulare un insieme di sessioni OLAP;

- numero massimo di misure: la quantità di misure distinte presenti, al massimo, in ogni query;

- cardinalità minima e massima delle *session-seed query*: le session-seed query, o *query iniziali*, sono le query OLAP con cui partiranno tutte le navigazioni. Per effettuare un'analisi, un decisore aziendale non scrive le query OLAP, ma sceglie da un pool ristretto di query predefinite e naviga secondo le sue esigenze. Le cardinalità delle query di questo pool devono essere limitate da due parametri: il decisore non ha interesse a cominciare con una query troppo vaga (100.000 tuple restituite) o troppo specifica (una singola tupla). Senza informazioni specifiche, si considera come cardinalità di una query la produttoria delle cardinalità dei livelli di gerarchia inclusi nel group-by set, considerando anche eventuali predicati di selezione;

- numero di *query sorprendenti*: le query sorprendenti sono particolari query che compaiono frequentemente all'interno di alcune sessioni, perché restituiscono dati particolarmente rilevanti (inaspettati o anomali). Nelle sessioni create con il template Explorative (descritto in seguito), il percorso delle query passerà, prima o poi, da una query sorprendente - o da una molto simile. Questo aspetto della generazione dei workload serve a dare un'idea di fenomeni inusuali ricorrenti nelle sessioni di analisi.

2.1.3 Parametri locali

Dopo aver scelto i parametri globali (ovvero, valevoli per tutti i profili), occorre impostare le caratteristiche di ogni profilo, che ne determinano la struttura e i contenuti delle sessioni. I parametri sono i seguenti:

- nome del profilo: utile per distinguerli, non ha un impatto effettivo nella generazione;

- numero di session-seed query: come già accennato, queste query rappresentano il punto di partenza di un insieme di sessioni, e ogni profilo deve crearle prima di generare le proprie sessioni;

- lunghezza minima e massima di sessione: questi due valori limitano il numero di query che formeranno una sessione, compresa la session-seed query (o query iniziale);

- numero di sessioni: questo parametro indica quante sessioni generare per ogni query iniziale;

- probabilità di *year prompt*: un valore appartenente all'intervallo [0, 1], che indica la percentuale di query iniziali vincolate da uno year prompt. Lo year prompt è un predicato di selezione che limita la dimensione temporale su un determinato anno. Questo predicato può essere eliminato durante l'evolversi di una sessione;

- presenza di *segregation attribute*: un flag (valore booleano) che indica se le query iniziali sono vincolate da un segregation attribute. Il segregation attribute è un particolare predicato di selezione che può applicarsi ad una qualsiasi gerarchia e ad un qualsiasi livello: simula una restrizione nella visibilità dei dati all'utente. Ad esempio, un preside di facoltà ha accesso solo ai dati della sua facoltà, il responsabile di un reparto 'vede' solo i dati relativi al suo reparto, ecc. Il segregation attribute non è eliminabile: di conseguenza, se è presente in una query iniziale, lo si ritroverà in tutte le sessioni generate partendo da quella query.

2.2 Generazione

Una volta impostati i parametri, il generatore creerà, per ogni profilo e per ogni query iniziale del profilo, un numero prestabilito di sessioni. Per esempio, supponiamo che l'utente imposti il generatore su tre profili con i seguenti parametri:

	Profilo 1	Profilo 2	Profilo 3
Numero di *session-seed query*:	4	5	3
Lunghezza minima sessione:	10	12	8
Lunghezza massima sessione:	20	18	16
Numero di sessioni:	15	25	20
Year prompt:	0,5	0	0,6
Segregation attribute:	false	true	true

Per il Profilo 1, il generatore creerà 60 sessioni (15 per ogni query iniziale), di lunghezza variabile tra le 10 e le 20 query. Delle 4 query iniziali, $4*0,5 = 2$ query avranno uno year prompt, e nessuna delle 4 query avrà un segregation attribute. Il Profilo 2 avrà 125 sessioni (25 per ogni query iniziale): tutte le query iniziali avranno un segregation attribute, ma nessuna avrà year prompt. Ragionamento simile vale per il Profilo 3: in questo caso ci sarà uno year prompt nel 60% di query iniziali (ovvero in $3 * 0,6 = 1,8 \approx 2$ delle 3 query).

Ogni sessione sarà caratterizzata da un modello, ovvero un *template*.

2.2.1 Template

Un template descrive come una sessione sarà costruita, e riflette i diversi modi con cui un utente può interagire con i dati multidimensionali, secondo i propri obiettivi.

Slice and Drill: questo template costruisce una sessione in cui si navigano le gerarchie verso il massimo dettaglio; lo si può vedere come un'esplosione iterativa di particolari dati. La sua utilità è dovuta al modo in cui interagi-

scono diversi front-end OLAP con i report (ovvero, abbinando gli operatori drill-down e slice-and-dice). Per esempio, un utente potrebbe visualizzare, per ogni nazione, le vendite annuali dei propri negozi: selezionando una nazione, visualizza le vendite per ogni regione; selezionando una regione, visualizza le vendite per ogni città, e così via. Il template simula l'applicazione di due operatori OLAP: la selezione di una 'fetta' del cubo (Slice) e l'esplosione verso un maggior dettaglio (Drill).

Just Slice: questo template costruisce una sessione in cui, iterativamente, si effettua uno Slice su tutti gli elementi del livello di una gerarchia. Ad esempio, un utente può voler vedere la durata media dei giorni di ricovero dei pazienti con diagnosi di trauma cranico, poi con diagnosi di intossicazione alimentare, poi con diagnosi di frattura, e così via. Per non creare sessioni troppo brevi, è possibile spostarsi ad un livello di maggior dettaglio e applicare di nuovo uno Slice.

Explorative: questo template, come suggerisce il nome, simula un'esplorazione estemporanea del cubo multidimensionale. L'utente può cambiare i predicati o le misure, e può spostarsi nelle gerarchie per aggregare o esplodere i dati. Durante la navigazione, tuttavia, queste sessioni tenderanno ad avvicinarsi ad una query sorprendente (descritta in precedenza), per dare un'idea di fenomeni interessanti nella casualità. Dopo aver raggiunto una query sorprendente (o una molto simile), la sessione prosegue con un'esplorazione casuale.

Goal Oriented: questo template simula una sessione con risultato predeterminato. Ad ogni query iniziale è assegnata una query finale: tutte le sessioni che cominciano con la i-esima query iniziale finiranno con la i-esima query finale, indipendentemente dalla lunghezza della sessione o dalla presenza di particolari predicati di selezione. Ogni query finale deve essere raggiungibile dalla query iniziale corrispondente.

2.3 Output

Nell'ottica della disponibilità del progetto (e per ragioni di portabilità), l'output sarà generato in formato XML, e conterrà le seguenti informazioni:

- data e ora;
- parametri globali;
- parametri locali di ogni profilo;
- sessioni di ogni profilo, contenenti:
  - nome del profilo;
  - numero progressivo;
  - template di sessione;
  - query di ogni sessione, contenenti:
    - elementi del group-by set;
    - misure;
    - predicati di selezione.

In aggiunta a questo file di output, è opportuno produrre un file testuale contenente solo le sessioni, per poter controllare agevolmente la correttezza semantica del carico di lavoro generato.

Capitolo 3

Analisi e progettazione

Durante la specifica dei requisiti, ci si è accordati su quali funzionalità deve avere il progetto per adempiere ai desiderata: quali input accetta, quale output produce, con quali criteri gestire i profili e generare le sessioni, ecc. La fase successiva è l'analisi dei requisiti, in cui si decide come strutturare il progetto in un formalismo adatto, quali concetti del dominio applicativo includere nel progetto e come questi concetti (o entità) interagiscono fra loro. Solo alla fine della fase di analisi si potrà procedere con una prima progettazione del sistema, in cui i concetti della fase precedente vengono raffinati per poter essere direttamente implementati. È stato utilizzato il formalismo del linguaggio UML® [14] (*Unified Modeling Language*).

3.1 Classi di analisi

Nella figura 3.1 si può vedere la struttura del diagramma delle classi, generato usando Visual Paradigm [15].

Generator: questa classe si occupa di popolare il cubo multidimensionale su cui fare analisi, generare i vari profili e creare un numero fissato di query sorprendenti. I parametri globali si possono desumere dagli attributi di questa classe (limiti di cardinalità delle query iniziali, numero massimo di misure) e dalle relazioni con le altre classi (numero di query sorprendenti,

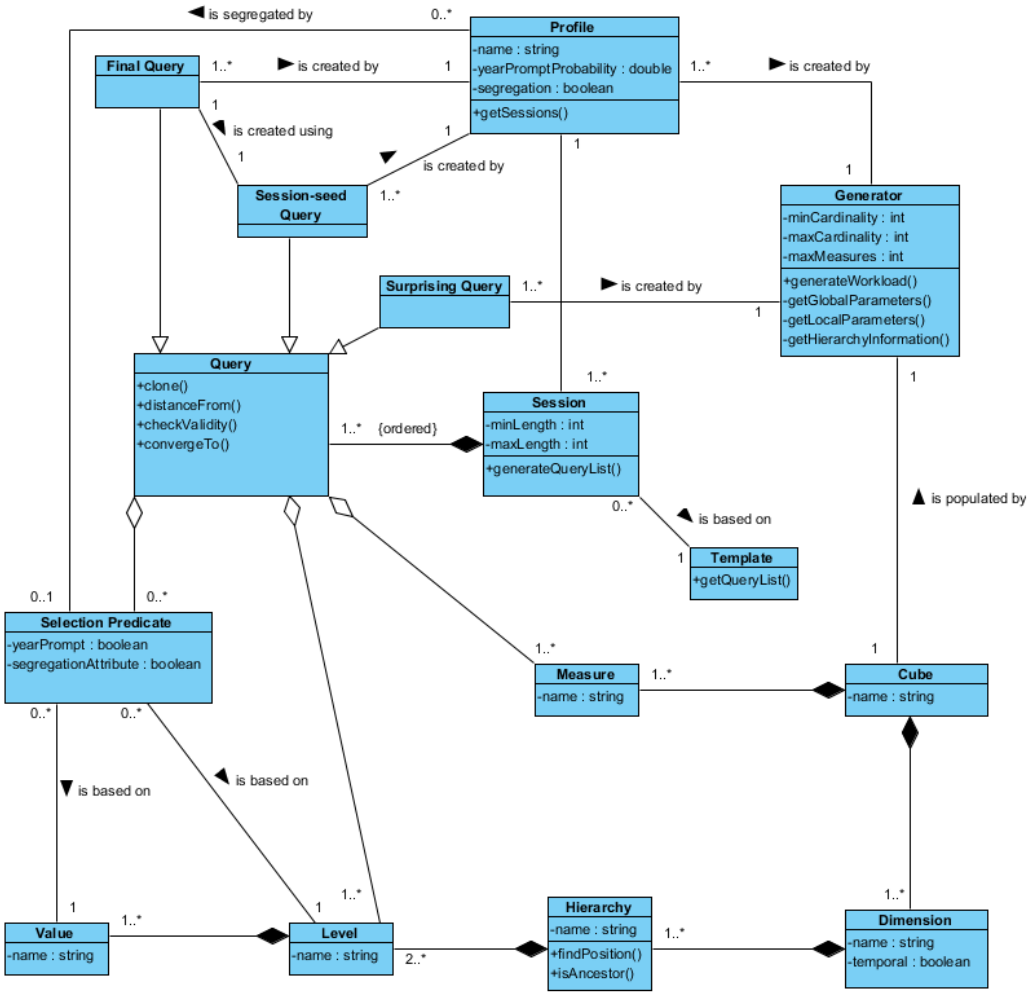

Figura 3.1: Diagramma delle classi di analisi.

numero di profili). La classe Generator offre la funzionalità di generazione del carico di lavoro.

Cube: questa classe rappresenta il cubo multidimensionale, contenente informazioni per la creazione di query. Un cubo contiene un certo numero di dimensioni e un insieme di misure.

Dimension: questa classe identifica l'entità omonima. Una dimensione possiede un nome ed una o più gerarchie, ed offre le funzionalità base per impostare e gestire le gerarchie: la presenza di più di una gerarchia solleva la

questione del *branching* (ramificazione), che sarà approfondito descrivendo la classe Query.

Hierarchy: questa classe fondamentale rappresenta una gerarchia. In base alle caratteristiche di una gerarchia è possibile aggregare o esplodere i dati, verificare inconsistenze nel group-by set di una query e scorrerne i livelli. Una gerarchia consta, per l'appunto, di un insieme di livelli.

Level: questa classe identifica un livello di una gerarchia. All'interno di una query, Level ricopre una doppia funzione: determina un elemento del group-by set, e può essere il cardine di un predicato di selezione che vincola la query. Un livello ha un nome e un insieme di valori testuali, necessari anch'essi ai predicati di selezione.

Value: questa classe termina la gerarchia di composizioni di un cubo multidimensionale. Come già accennato, Value è utile alla costruzione dei predicati di selezione.

Measure: questa classe rappresenta l'entità 'misura' di un cubo. In base ai requisiti di progetto, per gestire le misure è sufficiente conoscerne il nome: nonostante ciò, è stata creata una classe dedicata in previsione di future evoluzioni del software. Ad esempio, si potrebbero aggiungere funzionalità riguardo all'additività delle misure o alla loro caratterizzazione (misure di flusso, unitarie...).

Query: l'entità Query è di fondamentale importanza. Una query rappresenta l'unità base di una sessione, ed è composta da tre elementi:

- un group-by set (insieme non vuoto);
- un certo numero di misure (superiormente limitato da un parametro globale);
- un insieme di predicati di selezione (potenzialmente vuoto).

Per identificare gli elementi del group-by set è sufficiente un livello: come si nota dal diagramma, ogni livello appartiene ad una sola gerarchia, indipendentemente dal branching.

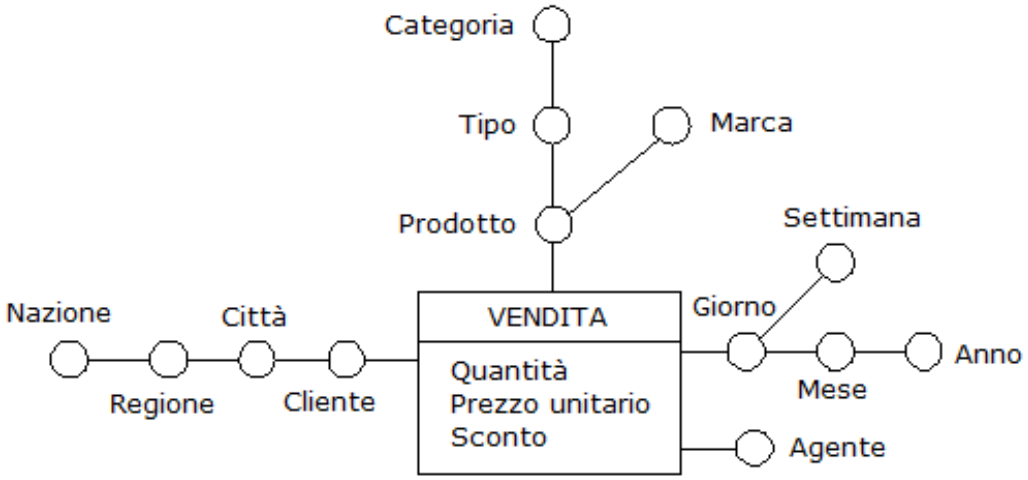

Figura 3.2: DFM per il fatto VENDITA. Nella dimensione del prodotto e in quella temporale c'è un branching.

Il branching appare comunemente negli schemi di fatto che modellano fenomeni del mondo reale: occorre quando, nell'albero di una gerarchia, esiste almeno un attributo dimensionale con più di un discendente; si parla, in questo caso, di gerarchie non lineari. Nella semantica di Mondrian, questa caratteristica è modellata definendo diverse gerarchie in una sola dimensione: gli attributi dimensionali comuni a più gerarchie indicano un percorso da esse condiviso. L'importanza del branching si riscontra allorché si costruisce una query, durante la scelta del group-by set. Affinché il group-by set sia sensato, non devono esistere dipendenze funzionali tra i suoi elementi: in una gerarchia, tra un nodo e un suo discendente esiste una relazione molti-a-uno, e in presenza di gerarchie che condividono alcuni attributi dimensionali occorre fare attenzione. Se due o più gerarchie partecipano ad un group-by set con livelli collegati tra loro da relazioni molti-a-uno c'è un'incoerenza: la rappresentazione dei dati ha poco senso. Per esempio, supponiamo che per il fatto VENDITA una ipotetica dimensione PRODOTTO sia costruita come in Figura 3.2.

Per ipotesi, vogliamo visualizzare gli incassi delle vendite secondo un

group-by set contenente (marca, categoria) (le gerarchie non presenti si considerano aggregate al massimo livello). L'output risultante assomiglierà a questo:

	AL	EL	AB	PU	IG	CA	UT	SP
DiTutto	287,00	340,52	0,00	1.025,00	0,00	188,00	212,00	0,00
Magnam	2.886,00	0,00	0,00	0,00	390,00	0,00	0,00	0,00
Piccol	1.683,30	291,00	1.024,00	336,00	204,00	0,00	442,00	380,00
ShockIt	0,00	2.556,00	0,00	0,00	0,00	0,00	1.402,00	0,00
CurrCurr	0,00	0,00	0,00	0,00	0,00	0,00	0,00	3.938,00
NonSoloPile	0,00	255,50	0,00	0,00	0,00	484,00	1.936,00	0,00
DeCorti	810,00	0,00	0,00	0,00	1.898,70	0,00	0,00	0,00

Legenda delle categorie: AL - Alimentari; EL - Elettronica; AB - Abbigliamento; PU - Pulizia; IG - Igiene personale; CA - Cancelleria; UT - Utensili; SP - Sport.

Dato che non esistono dipendenze funzionali tra MARCA e CATEGORIA, ai prodotti di una marca possono corrispondere più categorie, e viceversa. Se però scegliessimo CATEGORIA dalla prima gerarchia e PRODOTTO dalla seconda gerarchia, e visualizzassimo ancora gli incassi delle vendite, il risultato sarebbe simile al seguente:

	AL	EL	AB	PU	IG	CA	UT	SP
Maglia DolceBabbana	0,00	0,00	1.025,00	0,00	0,00	0,00	0,00	0,00
Pila CuraDell	0,00	976,00	0,00	0,00	0,00	0,00	0,00	0,00
Dentifricio Sbiancadent	0,00	0,00	0,00	0,00	436,00	0,00	0,00	0,00
Fogli PoSteat	0,00	0,00	0,00	0,00	0,00	557,00	0,00	0,00
Pasta DeBeppe	2.048,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Scarpe Mike	0,00	0,00	0,00	0,00	0,00	0,00	0,00	3.347,00
Detersivo ArmaBrillo	0,00	0,00	0,00	1.898,70	0,00	0,00	0,00	0,00
Trapano White&Deckest	0,00	0,00	0,00	0,00	0,00	0,00	533,30	0,00

Al massimo una cella di ogni riga conterrà un valore significativo: giacché PRODOTTO determina TIPO e TIPO determina CATEGORIA, ad ogni prodotto è associata una sola categoria. Il coefficiente di sparsità di questa matrice è pari a $(k-1)/k$, dove k è il numero di categorie: questo esempio mostra l'insensatezza di un group-by set con dipendenze funzionali tra i propri membri.

Le misure di una query appartengono all'insieme di misure del cubo multidimensionale: in questo caso, si può fare riferimento alla classe Measure sopracitata.

Il predicato di selezione (espresso nella classe *SelectionPredicate*) è un oggetto complesso, nonostante ai fini del progetto si considerino soltanto predicati di uguaglianza: un predicato consta di un livello di una gerarchia e un elemento preso dai valori distinti del livello in questione (come per gli elementi del group-by, la gerarchia non è necessaria). Oltre ai componenti multidimensionali, un predicato di selezione deve contenere due attributi booleani, per identificarlo come year prompt, segregation attribute o predicato normale.

Complessivamente, il group-by set, le misure e i predicati di selezione costituiscono gli attributi di una Query. Le sue funzionalità, tuttavia, riguarderanno principalmente quattro tipi di operazioni:

- gestione degli attributi di classe;
- controlli di consistenza e di selettività;
- convergenza verso un'altra query;
- manipolazione delle gerarchie.

Le operazioni di controllo verificano la consistenza del group-by set e la presenza di predicati di selezione su particolari gerarchie. La convergenza verso una particolare query è indispensabile per la creazione di sessioni Goal Oriented (in cui l'ultima query è predefinita) ed Explorative (in cui il percorso passa molto vicino - o attraversa - una query sorprendente); saranno necessarie operazioni di supporto sulla raggiungibilità di una query e sulla 'distanza' tra query. La manipolazione delle gerarchie occorre quando si applica un operatore di roll-up o drill-down su una gerarchia: per non creare query errate, servono controlli sull'attuale livello della gerarchia nella query corrente. Una query è incoerente se, ad esempio, contiene più di un predi-

cato di selezione su una stessa gerarchia, o se il suo group-by set contiene dipendenze funzionali tra i membri.

La classe Query è stata specializzata in *Session-seed Query* (query iniziale), *Final Query* (query finale) e *Surprising Query* (query sorprendente). Nonostante la specializzazione non comporti né attributi né operazioni aggiuntive, serve una distinzione:

- ogni profilo crea le proprie query iniziali, che rappresentano i punti di partenza delle proprie sessioni;
- i profili creano anche una query finale per ogni query iniziale, per essere usate nelle sessioni con template Goal Oriented;
- le query sorprendenti sono globali (valide per tutti i profili simulati) e vengono create prima della generazione dei profili, per essere usate nelle sessioni con template Explorative.

Profile: questa classe rappresenta l'entità omonima. Il profilo ha come attributi i parametri locali impostati durante la scelta dell'input (numero di sessioni e di query iniziali, limiti di lunghezza di sessione ecc.), mentre le operazioni sono le funzioni base per leggere gli attributi e la funzione centrale di creazione delle sessioni.

Session: questa classe indica una sessione generata in output. Una sessione contiene una lista (ordinata) di query e il nome del template che si è occupato della generazione della suddetta lista; ai fini della produzione dell'output, è ragionevole includere due attributi informativi che specifichino il numero progressivo della sessione e il nome del profilo per cui è stata creata.

Template: questa classe identifica un template. Il template si occupa della generazione di query che andranno a formare una sessione.

3.2 Classi di progettazione

Le classi di analisi corrispondono a concetti esistenti ed escludono ogni dettaglio implementativo: le classi di progettazione, invece costituiscono un

raffinamento delle classi di analisi, e il loro dettaglio è tale da poter immediatamente procedere con l'implementazione, dato che specificano esattamente come le classi adempiranno ai loro compiti.

Il processo di raffinamento ha portato alcune modifiche strutturali:

- la classe Value è stata eliminata: il suo collegamento con la classe SelectionPredicate si è spostato verso la classe contenitore Level;
- è stata aggiunta una classe GroupByElement, per gestire il collegamento tra Query e Level;
- il processo di creazione delle sessioni tramite i template ha suggerito l'adozione di una *abstract factory*, un design pattern creazionale (vedi Figura 3.3): l'istanziazione degli oggetti non dipende più dal profilo. Template è ora un'interfaccia, implementata dalle classi GoalOriented, JustSlice, SliceAndDrill ed Explorative.

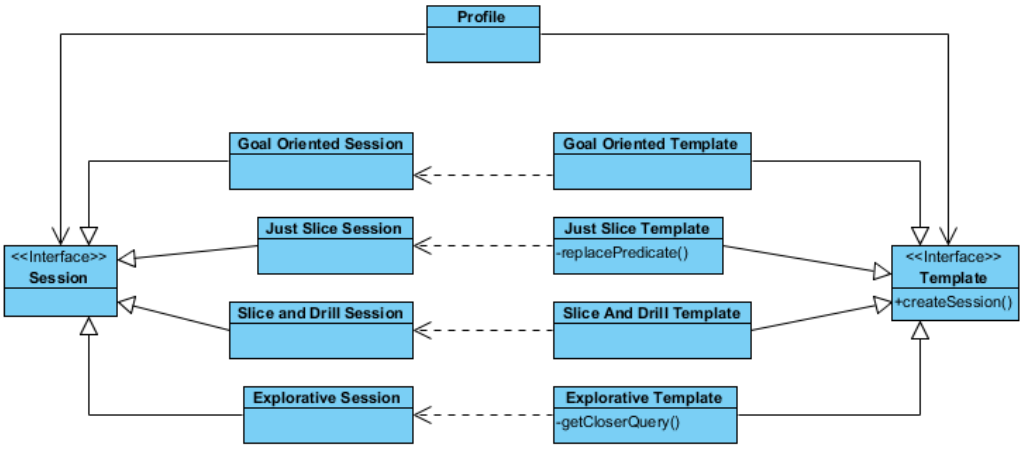

Figura 3.3: Viene usato un design pattern Abstract Factory per istanziare i template e generare famiglie di sessioni.

Per rispettare l'incapsulamento del paradigma ad oggetti, tutti gli attributi sono privati, ed accessibili solo tramite operazioni getter/setter. Si userà la

semantica di UML per descrivere attributi (**nomeAttributo : tipo**) e operazioni (**visibilità nomeOperazione (parametro : tipoParametro...) : tipoRestituito**).

GENERATOR

- Attributi:
  - maxCardinality : int - la cardinalità massima ammissibile per le query iniziali;
  - maxMeasures : int - il limite superiore al numero di misure per ogni query;
  - minCardinality : int - la cardinalità minima ammissibile per le query iniziali;
  - numberOfProfiles : int - il numero dei profili da simulare;
  - numberOfSurprisingQueries : int - il numero di query sorprendenti da creare;
  - surprisingQueries : Query[] - le query sorprendenti da usare nelle sessioni Explorative.
- Operazioni:
  - createSurprisingQueries(cube : Cube) : void - questa operazione genera le query sorprendenti basandosi sul cubo multidimensionale;
  + generateWorkload() : void - questa operazione invoca la creazione delle sessioni di tutti i profili da simulare, e produce i documenti in output;
  - getCube() : Cube - questa operazione costruisce il cubo multidimensionale usando il file di metadati;
  - getGlobalParameters() : void - questa operazione inizializza i parametri globali del generatore;
  - getHierarchyInformation(cube : Cube) : void - questa operazione popola i livelli delle gerarchie del cubo usando file testuali;
  - getLocalParameters() : Profile - questa operazione crea un profilo completo di parametri e lo restituisce.

CUBE

- Attributi:
  - dimensions : Dimension[] - l'insieme di dimensioni del cubo;
  - measures : Measure[] - le misure del cubo;
  - name : String - il nome del cubo;
- Operazioni:
  + addDimension(dimension : Dimension) : void - questa operazione aggiunge una dimensione al cubo;

+ addMeasure(measure : Measure) : void - questa operazione aggiunge una misura al cubo;
+ findHierarchy(name : String) : Hierarchy - questa operazione cerca e restituisce una gerarchia con lo stesso nome del parametro passato;
+ getDimensions() : Dimension[] - operazione getter per le dimensioni del cubo;
+ getHierarchies() : Hierarchy[] - questa operazione restituisce tutte le gerarchie del cubo;
+ getMeasures() : Measure[] - operazione getter per le misure del cubo
+ getName() : String - operazione getter per il nome del cubo;
+ getRandomMeasures(number : int) : Measure[] - questa operazione restituisce un numero fissato di misure distinte;
+ getTemporalDimension() : Dimension - operazione getter per la dimensione temporale.

DIMENSION

- Attributi:
  - hierarchies : Hierarchy[] - la lista di gerarchie appartenenti alla dimensione;
  - name : String - il nome della dimensione;
  - temporal : boolean - un flag che indica se la dimensione è temporale;
- Operazioni:
  + addHierarchy(hierarchy : Hierarchy) : void - questa operazione aggiunge una gerarchia alla lista di gerarchie della dimensione;
  + getHierarchies() : Hierarchy[] - operazione getter per le gerarchie della dimensione;
  + getHierarchy(number : int) : Hierarchy - operazione getter per una particolare gerarchia;
  + getName() : String - operazione getter per il nome della dimensione;
  + isTemporal() : boolean - operazione getter per il flag temporale della dimensione.

HIERARCHY

- Attributi:
  - levels : Level[] - lista dei livelli della gerarchia;
  - name : String - nome della gerarchia;
- Operazioni:
  + addLevel(level : Level) : void - questa operazione aggiunge un livello alla lista di livelli della gerarchia;
  + findPosition(name : String) : int - questa operazione cerca un livello con il nome fornito in input e ne restituisce la posizione;
  + getLevel(number : int) : Level - operazione getter per il livello presente in lista alla posizione data;

+ getLevel(name : String) : Level - operazione getter per il livello con il nome fornito in input;
+ getLevelCount() : int - operazione getter per il numero totale di livelli;
+ getLevels() : Level[] - operazione getter per la lista di livelli della gerarchia;
+ getName() : String - operazione getter per il nome della gerarchia;
+ getRandomLevel(maxIncluded : boolean) : Level - operazione getter per un livello casuale. Il parametro indica se si include o no il livello più aggregato;
+ getValidPredicatePosition(position : int) : int - questa operazione restituisce un valore accettabile per posizionare un predicato di selezione, considerando la posizione di una query sulla gerarchia;
+ isAncestor(firstLevel : String, secondLevel : String) : boolean - questa operazione verifica se due livelli sono collegati da una relazione molti-a-uno sulla gerarchia corrente;
+ isMaxLevel(name : String) : boolean - questa operazione verifica se un livello (identificato dal nome) rappresenta la massima aggregazione della gerarchia;
+ setLevelValues(values : String[]) : void - questa operazione popola tutti i livelli 'non-all' della gerarchia usando un insieme di stringhe.

LEVEL

- Attributi:
  - name : String - il nome del livello;
  - values : String[] - i valori distinti del livello;
- Operazioni:
  + addDistinctValue(name : String) : void - questa operazione aggiunge un valore alla lista di elementi del livello (se non già presente);
  + getName() : String - operazione getter per il nome del livello;
  + getRandomValue() : String - operazione getter per un valore casuale del livello;
  + getValues() : String[] - operazione getter per i valori distinti del livello.

MEASURE

- Attributi:
  - name : String - il nome della misura;
- Operazioni:
  + getName() : String - operazione getter per il nome della misura.

QUERY

- Attributi:

  - groupBySet : GroupByElement[] - la lista di elementi nel group-by set della query;
  - measures : Measure[] - la lista di misure della query;
  - predicates : SelectionPredicate[] - l'insieme di predicati di selezione della query;

- Operazioni:

  + addGroupByElement(hierarchy : String, level : String) : void - questa operazione crea un nuovo elemento e lo aggiunge al group-by set della query;
  + addMeasure(measure : Measure) : void - questa operazione aggiunge una misura all'insieme di misure della query;
  + addSelectionPredicate(predicate : SelectionPredicate) : void - questa operazione aggiunge un predicato di selezione a quelli esistenti nella query;
  + ascendHierarchy(hierarchy : Hierarchy) : void - questa operazione applica un roll-up sulla gerarchia indicata;
  - changeGroupBy(cube : Cube) : void - questa operazione cambia il group-by della query applicando un roll-up o un drill-down su una gerarchia su cui sia possibile spostarsi;
  - changeMeasures(cube : Cube) : void - questa operazione cambia l'insieme di misure della query, rimuovendone una e aggiungendone un'altra, diversa da quella eliminata e dalle misure già presenti;
  - changePredicates(cube : Cube) : void - questa operazione cambia l'insieme di predicati di selezione di una query, aggiungendone o eliminandone uno;
  + checkValidity(cube : Cube) : void - questa operazione verifica la correttezza del group-by set, e all'occorrenza 'nasconde' alcune gerarchie (una gerarchia nascosta può tornare visibile se tramite roll-up il group-by torna consistente);
  + clone(query : Query) : Query - questa operazione restituisce una copia esatta di una query;
  + containsPredicateOn(hierarchy : String) : boolean - questa operazione verifica se esiste un predicato di selezione su una determinata gerarchia;
  + convergeTo(query : Query, cube : Cube) : boolean - questa operazione 'avvicina' la query corrente ad un'altra data, cambiando misure, group-by set o predicati. L'operazione ritorna **true** se è stato possibile avvicinare la query corrente a quella data;
  + descendHierarchy(hierarchy : Hierarchy) : void - questa operazione applica un drill-down sulla gerarchia indicata;
  + distanceFrom(query : Query, cube : Cube) : int - questa operazione calcola la 'distanza' tra la query corrente ed un'altra data, in termini di diversità dei componenti della query (misure, predicati, group-by);
  + findLevel(hierarchy : String) : String - questa operazione cerca l'elemento del group-by relativo alla gerarchia indicata, e ne restituisce il livello;
  + getCardinality(cube : Cube) : int - questa operazione calcola la cardinalità di una query considerando tutti gli elementi visibili del group-by set, le cardinalità dei loro livelli ed eventuali predicati di selezione;
  + getGroupBySet() : GroupByElement[] - operazione getter per il group-by set della query;

+ getMeasures() : Measure[] - operazione getter per l'insieme di misure della query;
+ getPredicates() : SelectionPredicate[] - operazione getter per i predicati di selezione della query;
+ getSelectionPredicate(hierarchy : String) : SelectionPredicate - questa operazione cerca e restituisce il predicato di selezione impostato sulla gerarchia indicata;
+ getVisibility(hierarchy : String) : boolean - questa operazione verifica la visibilità di una gerarchia all'interno della query (nota: una gerarchia totalmente aggregata si considera ancora visibile);
– hierarchyMatch(query : Query, hierarchy : Hierarchy) : boolean - questa operazione controlla se la query corrente può convergere sulla query indicata sulla gerarchia data;
+ isAscendable(hierarchy : Hierarchy) : boolean - questa operazione controlla se è possibile effettuare un roll-up sulla gerarchia indicata, considerando il livello corrente della query sulla gerarchia e la presenza di predicati di selezione;
+ isDescendible(hierarchy : Hierarchy) : boolean - questa operazione controlla se è possibile effettuare un drill-down sulla gerarchia indicata, considerando il livello corrente della query sulla gerarchia;
+ isSegregated() : boolean - questa operazione verifica la presenza di un segregation attribute nella query;
+ isSegregatedOn(hierarchy : Hierarchy) : boolean - questa operazione controlla se esiste un segregation attribute sulla gerarchia indicata;
– measureMismatch(measures : Measure[]) : boolean - questa operazione verifica se le misure della query corrente e le misure della lista indicata non coincidono;
– predicatesMismatch(query : Query) : boolean - questa operazione valuta la possibilità di far combaciare i predicati di selezione della query corrente con quelli della query indicata. La presenza di segregation attribute può indicare un quasi-match: i predicati sono diversi, ma non è possibile trasformare completamente gli uni negli altri;
+ printQuery() : String - questa operazione costruisce una rappresentazione testuale di una query OLAP;
+ randomEvolution(cube : Cube) : Query - questa operazione restituisce una query ottenuta come evoluzione casuale della query corrente;
+ setLevel(hierarchy : String, level : String) : void - questa operazione imposta un determinato livello per la gerarchia data nella query corrente;
+ setMeasures(Measure[]) : void - operazione setter per l'insieme di misure della query;
– setVisibility(hierarchy : String, visibility : boolean) : void - questa operazione imposta come visibile o non visibile una gerarchia presente nel group-by set.

GroupByElement

- Attributi:
  – hierarchy : String - il nome della gerarchia di questo elemento del group-by;
  – level : String - il livello della gerarchia.

  - visible : boolean - un flag, impostato a **true** se la gerarchia deve essere nascosta;

- Operazioni:

  + getHierarchy() : String - operazione getter per il nome della gerarchia;
  + getLevel() : String - operazione getter per il nome del livello;
  + getVisible() : boolean - operazione getter per la visibilità dell'elemento del group-by;
  + setLevel(level : String) : void - operazione setter per il livello della gerarchia;
  + setVisible(flag : boolean) : void - operazione setter per la visibilità dell'elemento.

SelectionPredicate

- Attributi:

  - element : String - il valore su cui è impostato il predicato;
  - hierarchy : String - il nome della gerarchia vincolata dal predicato;
  - level : String - il nome del livello della gerarchia vincolata;
  - segregationAttribute : boolean - un flag, impostato a **true** se il predicato è un segregation attribute;
  - yearPrompt : boolean - un flag, impostato a **true** se il predicato è un year prompt;

- Operazioni:

  + getElement() : String - operazione getter per il valore del predicato;
  + getHierarchy() : String - operazione getter per la gerarchia vincolata;
  + getLevel() : String - operazione getter per il livello della gerarchia;
  + getPrompt() : boolean - operazione getter per il flag year prompt;
  + getSegregation() : boolean - operazione getter per il flag segregation attribute;
  + getSelectionPredicate() : String - questa operazione costruisce una stringa concatenando gli attributi testuali del predicato;
  + matchesWith(predicate : SelectionPredicate) : boolean - questa operazione valuta se il predicato indicato coincide totalmente col predicato corrente;

PROFILE

- Attributi:

  - finalQueries : Query[] - lista di query finali del profilo;
  - initialQueries : Query[] - lista di query session-seed del profilo;
  - maxSessionLength : int - limite superiore alla lunghezza di una sessione di questo profilo;
  - minSessionLength : int - limite inferiore alla lunghezza di una sessione di questo profilo;

  - segregationAttribute : boolean - un flag, indica se il profilo è vincolato da un segregation attribute;
  - template : Template - il template di creazione delle sessioni del profilo;
  - yearPromptPercentage : double - percentuale di query iniziali segnate da un year prompt;

- Operazioni:

  - chooseTemplate() : Template - questa operazione restituisce un template per la creazione di sessioni, scelto casualmente tra quattro template;
  - createFinalQueries(cube : Cube) : void - questa operazione crea le query finali applicando ripetute evoluzioni delle query iniziali;
  - createInitialQueries(cube : Cube, maxMeasures : int, minCardinality : int, maxCardinality : int) : void - questa operazione crea un numero fissato di query iniziali, secondo i vincoli imposti dai parametri globali e locali;
  + getMaxSessionLength() : int - operazione getter per la massima lunghezza di sessione;
  + getMinSessionLength() : int - operazione getter per la minima lunghezza di sessione;
  + getname() : String - operazione getter per il nome del profilo;
  + getNumberOfQueries() : int - operazione getter per il numero di query iniziali;
  + getSegregationAttribute() : boolean - operazione getter per il flag segregation attribute;
  + getSessions(cube : Cube, maxMeasures : int, minCardinality : int, maxCardinality : int, surprisingQueries : Query[]) : Session[] - questa operazione raccoglie tutte le sessioni create dai template invocati dal profilo, e le restituisce;
  + getYearPromptPercentage() : boolean - operazione getter per il flag year prompt.

SESSION

- Attributi:

  - profileName : String - il nome del profilo a cui appartiene questa sessione;
  - progressive : int - il numero progressivo della sessione corrente;
  - queryList : Query[] - la lista di query che forma la sessione;
  - templateName : il nome del template che ha creato questa sessione;

- Operazioni:

  + getProfileName() : String - operazione getter per il nome del profilo;
  + getProgressive() : int - operazione getter per il numero progressivo della sessione;
  + getQueryList() : Query[] - operazione getter per la lista di query;
  + getTemplateName() : String - operazione getter per il nome del template;
  + setQueryList(queries : Query[]) : void - operazione setter per la lista di query;

TEMPLATE (Template è un'interfaccia, quindi possiede soltanto operazioni pubbliche)

- Operazioni:

  + generateSession(profileName : String, progressive : int, minLength : int, maxLength : int, initialQuery : Query, cube : Cube, surprisingQueries : Query[], finalQuery : Query) : Session - questa operazione genera una sessione comprensiva di parametri locali e lista di query;
  + getName() : String - operazione getter per il nome del template;

Le classi seguenti realizzano l'interfaccia Template: di esse si riportano gli attributi e le operazioni non presenti in Template.

GoalOriented

- Attributi:

  – name : String='Goal Oriented' - il nome del template;

- Operazioni:

  – getQueryList(minLength : int, maxLength : int, initialQuery : Query, cube : Cube, surprisingQueries : Query[], finalQuery : Query) : Query[] - questa operazione costruisce la lista di query da includere nella sessione.

Explorative

- Attributi:

  – name : String='Explorative' - il nome del template;

- Operazioni:

  – getCloserQuery(initialQuery : Query, surprisingQueries : Query[]) : Query - questa operazione verifica qual è la query sorprendente più 'vicina' alla query iniziale, e la restituisce;
  – getQueryList(minLength : int, maxLength : int, initialQuery : Query, cube : Cube, surprisingQueries : Query[], finalQuery : Query) : Query[] - questa operazione costruisce la lista di query da includere nella sessione.

SliceAndDrill

- Attributi:

  – name : String='Slice and Drill' - il nome del template;

- Operazioni:

  – getQueryList(minLength : int, maxLength : int, initialQuery : Query, cube : Cube, surprisingQueries : Query[], finalQuery : Query) : Query[] - questa operazione costruisce la lista di query da includere nella sessione.

JustSlice

- Attributi:
  - name : String='Just Slice' - il nome del template;
- Operazioni:
  - getQueryList(minLength : int, maxLength : int, initialQuery : Query, cube : Cube, surprisingQueries : Query[], finalQuery : Query) : Query[] - questa operazione costruisce la lista di query da includere nella sessione;
  - replacePredicate(query : Query, hierarchy : Hierarchy, value : String) : void - questa operazione aggiorna con un nuovo elemento il predicato di selezione sulla gerarchia indicata della query data.

Capitolo 4

Il progetto

Dopo il raffinamento delle classi di analisi nelle classi di progettazione, è seguita la fase dell'implementazione. Il progetto è stato sviluppato in linguaggio Java [11], versione 1.7, su ambiente di sviluppo Eclipse Juno.

Tutte le classi di progettazione sono state trasformate in classi Java, con qualche aggiunta:

- una classe Start, che istanzia e invoca la classe Generator, da cui parte la creazione di sessioni;

- due classi, CSVParser [8] e CSVReader [9], di ausilio per il parsing dei file CSV contenenti le informazioni sulle dimension table;

- una classe XMLReader, che si occupa di estrarre le informazioni da un file di metadati XML. Le sue operazioni comprendono l'estrazione dei nomi dei cubi dai metadati - per permettere all'utente di decidere su quale cubo costruire le sessioni - e la costruzione fisica di un cubo multidimensionale. Questa classe ha richiesto le API SAX [12] (Simple API for XML), e l'uso di XPath [16] per navigare il file XML;

- una classe XMLWriter, che costruisce il file di output in linguaggio XML, secondo le specifiche mostrate nel Cap. 2.

L'interazione dell'utente con il generatore avviene secondo questo procedimento (mostrato nella Figura 4.1):

1. l'utente avvia il programma;
2. il generatore chiede all'utente quale file di metadati XML considerare;
3. l'utente inserisce il nome del file (deve essere fisicamente collocato nella stessa cartella dell'applicativo);
4. il generatore estrae i nomi dei cubi descritti nel file indicato e chiede all'utente quale utilizzare;
5. l'utente indica su quale cubo vuole fare analisi;
6. il generatore costruisce il cubo multidimensionale e lo popola usando i file CSV contenenti le informazioni sulle dimension table (i file devono essere fisicamente collocati nella stessa cartella dell'applicativo);
7. il generatore chiede all'utente di inserire i parametri globali;
8. il generatore chiede all'utente di inserire i parametri locali per ogni profilo da simulare (al passo precedente l'utente ha specificato quanti profili devono essere creati);
9. il generatore crea le query sorprendenti;
10. il generatore procede alla creazione delle sessioni di un profilo;
11. il generatore produce l'output formale in un file XML e un output di supporto in un file testuale.

La creazione delle sessioni segue questa sequenza:

- il profilo crea le query iniziali: devono avere una cardinalità compresa tra due valori e i predicati corretti (se le impostazioni di profilo prevedono una percentuale non nulla di year prompt o un segregation attribute);

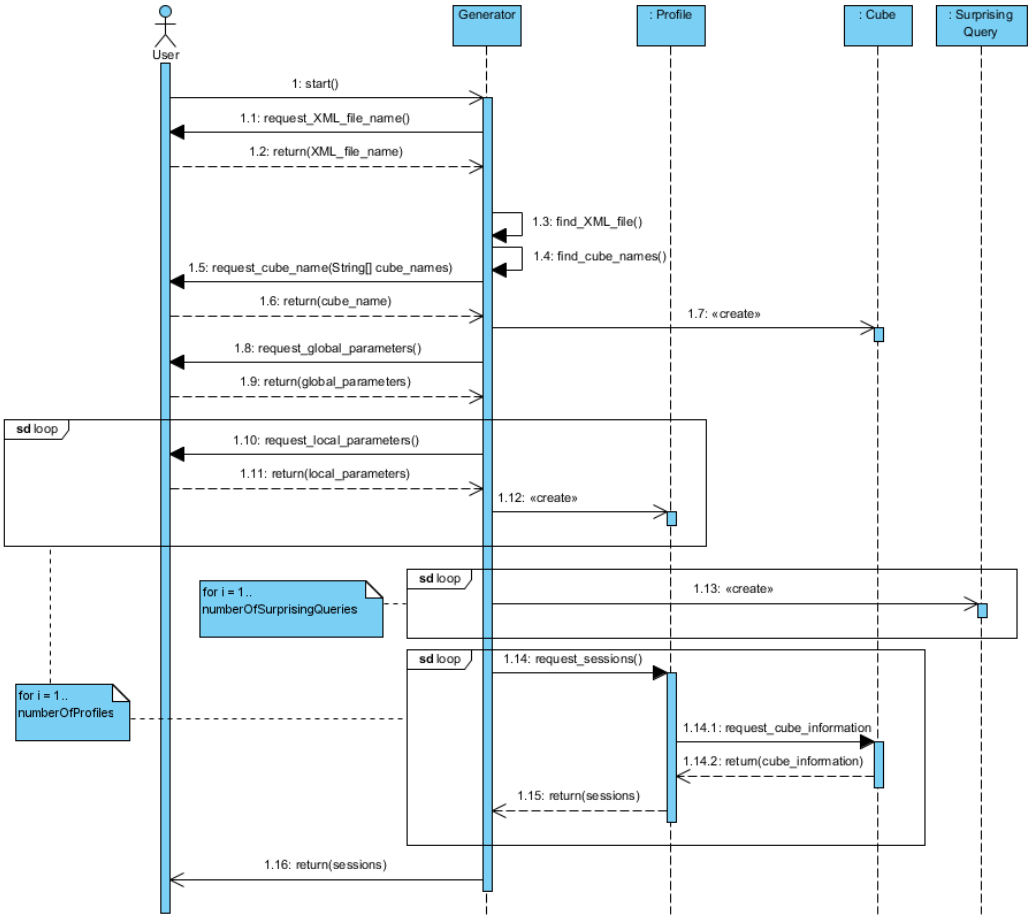

Figura 4.1: Diagramma di sequenza che specifica l'interazione tra l'utente e il generatore, e l'ordine di istanziazione di cubi e profili.

- vengono create le query finali: una query finale è ottenuta applicando un numero fissato di evoluzioni casuali alla query iniziale corrispondente (tutte le evoluzioni mantengono la query in uno stato consistente);
- per ogni query iniziale viene creato un numero fisso di sessioni, scegliendo casualmente un template e delegando ad esso la creazione della sessione e della sua lista di query.

I quattro template condividono la funzionalità di creazione della sessione, ma differiscono in come generano la lista di query:

- il template Explorative calcola quale query sorprendente è la più vicina alla query iniziale da utilizzare, aggiunge la query iniziale alla lista e crea query sempre più vicine alla query sorprendente scelta percorrendo un cammino minimo. Questo processo si ferma allorché si raggiunge la query sorprendente, o una molto simile (può capitare che, a causa di un segregation attribute, le query non possano coincidere). Se la lista risultante è troppo lunga, alcune query intermedie vengono eliminate: altrimenti, la lista prosegue con evoluzioni casuali della query sorprendente;
- il template Goal Oriented usa una funzionalità della classe Query (convergeTo) per costruire un percorso dalla query iniziale alla query finale; si noti che la query finale è stata creata in modo tale da poter essere raggiunta dalla corrispondente query iniziale. Se il percorso creato è troppo breve, vengono introdotte query intermedie, per indicare una deviazione;
- il template Slice and Drill controlla su quali gerarchie sia possibile applicare un operatore di drill-down fino in fondo (queste gerarchie devono avere un numero minimo di livelli e non essere vincolate da un segregation attribute): dopodiché, sceglie una delle suddette gerarchie, la esclude dal gruppo di gerarchie percorribili e applica iterativamente uno slicing e un drill-down. Una volta arrivato al massimo dettaglio

della gerarchia, il template ricomincia il procedimento su un'altra gerarchia, e così via. La lunghezza della sessione è vincolata dal numero dei livelli delle gerarchie su cui si possa fare drill-down: questo limite non è aggirabile;

- il template Just Slice controlla su quali gerarchie sia possibile effettuare uno slicing: non deve essere presente un segregation attribute sulla gerarchia, e deve avere un numero minimo di livelli (per poter applicare una sola volta un drill-down e continuare a procedere, in caso di sessioni troppo corte). Scelta una gerarchia, si applica iterativamente un predicato di selezione che limita un suo livello (non il massimo dettaglio), eventualmente sostituendo un predicato su questa gerarchia presente nella query iniziale: in questo caso, per non generare query ripetute, si memorizza il valore testuale del predicato di selezione che vincolava la query iniziale sulla gerarchia scelta, e lo si esclude dai futuri predicati di selezione.

4.1 Output

In seguito si riporta un esempio di query OLAP prodotta dal generatore (basata su database IPUMS, vedi Cap. 5):

```
1  <Query progressive="5">
2    <GroupBy>
3      <Element>
4        <Hierarchy Value="CITY"/>
5        <Level value="STATE"/>
6      </Element>
7      <Element>
8        <Hierarchy Value="RACE"/>
9        <Level value="RACEGROUP"/>
10     </Element>
11     <Element>
12       <Hierarchy Value="YEAR"/>
13       <Level value="YEAR"/>
14     </Element>
15     <Element>
16       <Hierarchy Value="SEX"/>
```

```
17        <Level value="SEX"/>
18      </Element>
19      <Element>
20        <Hierarchy Value="OCCUPATION"/>
21        <Level value="BRANCH"/>
22      </Element>
23    </GroupBy>
24    <Measures>
25      <Element value="SUMCOSTWATR"/>
26      <Element value="SUMPERWT"/>
27      <Element value="MAXINCTOT"/>
28    </Measures>
29    <SelectionPredicates>
30      <Element>
31        <Hierarchy value="CITY"/>
32        <Level value="REGION"/>
33        <Predicate value="East south central"/>
34        <YearPrompt value="false"/>
35        <SegregationAttribute value="true"/>
36      </Element>
37      <Element>
38        <Hierarchy value="YEAR"/>
39        <Level value="YEAR"/>
40        <Predicate value="2005"/>
41        <YearPrompt value="true"/>
42        <SegregationAttribute value="false"/>
43      </Element>
44      <Element>
45        <Hierarchy value="OCCUPATION"/>
46        <Level value="SUBCATEGORY"/>
47        <Predicate value="Professional and related occupations"/>
48        <YearPrompt value="false"/>
49        <SegregationAttribute value="false"/>
50      </Element>
51      <Element>
52        <Hierarchy value="RACE"/>
53        <Level value="MRN"/>
54        <Predicate value="2"/>
55        <YearPrompt value="false"/>
56        <SegregationAttribute value="false"/>
57      </Element>
58    </SelectionPredicates>
59  </Query>
```

La medesima query, in formato compresso, è la seguente:

CITY.STATE, RACE.RACEGROUP, YEAR.YEAR, SEX.SEX, OCCUPATION.BRANCH

SUMCOSTWATR, SUMPERWT, MAXINCTOT

CITY.REGION = East south central, YEAR.YEAR = 2005, OCCUPATION.SUBCATEGORY = Professional and related occupations, RACE.MRN = 2

La query consta di un group-by con cinque elementi ('nomeGerarchia' . 'nomeLivello'), tre misure e quattro predicati di selezione, tra cui un segregation attribute, un year prompt e due predicati normali. Per costruzione, il livello più dettagliato di una gerarchia si chiama come la gerarchia stessa, mentre il livello di massima aggregazione prende il nome di ALL.

Capitolo 5

Testing

La fase di testing è stata necessaria per valutare la correttezza sintattica e semantica delle sessioni generate. Una sessione può definirsi sintatticamente corretta se, ad esempio:

- tutte le query hanno i componenti indispensabili (un group-by set e un certo numero di misure);
- la lunghezza delle sessioni è contenuta nel range (*minSessionLength*, *maxSessionLength*);
- viene generato il corretto numero di query iniziali e di sessioni per query iniziale;
- la giusta percentuale di query iniziali contiene uno year prompt.

Gli indicatori di correttezza semantica, per quanto riguardino aspetti meno visibili, sono altrettanto fondamentali:

- il segregation attribute non viene mai eliminato;
- i template assolvono alla loro funzione creando una sessione di query consistenti;
- una query è sempre raggiungibile dalla precedente e la loro 'distanza' non è eccessiva;

- non esistono dipendenze funzionali all'interno del group-by set.

5.1 Database di test

Per verificare la correttezza sintattica e semantica delle sessioni create, sono stati effettuati test usando due file di metadati XML: IPUMS e FoodMart.

5.1.1 IPUMS

IPUMS-International [10] è un progetto volto a raccogliere e distribuire dati di censimento da tutto il mondo: l'intento è la disseminare gratuita di dati armonizzati, per scopi economici e sociali. Il progetto vede la collaborazione del Minnesota Population Center.

Lo schema di fatto di questo database prevede cinque gerarchie, così formate:

- CITY ⇒ STATE ⇒ REGION ⇒ ALLCITY
- RACE ⇒ RACEGROUP ⇒ MRN ⇒ ALLRACE
- YEAR ⇒ ALLYEAR
- SEX ⇒ ALLSEX
- OCCUPATION ⇒ BRANCH ⇒ SUBCATEGORY ⇒ CATEGORY ⇒ ALLOCCUPATION

La sigla MRN significa Major Race Number. Il livello più aggregato porta il nome della gerarchia con il prefisso ALL (per indicare l'aggregazione completa della gerarchia).

Sono presenti 25 misure, relative ai costi di energia elettrica, gas e acqua corrente. I nomi delle misure sono preceduti dal tipo di aggregatore (SUMCOSTGAS, MINCOSTWATR, AVGCOSTELEC...) con l'eccezione del conteggio delle occorrenze (EVENTCOUNT).

5.1.2 FoodMart

FoodMart è uno schema multidimensionale basato sull'omonimo dataset di test incluso nella distribuzione di Mondrian [6]. Il dataset è disponibile come database Microsoft Access™ o come script SQL con istruzioni *insert*.

Il file di metadati FoodMart presenta alcune dimensioni 'globali' (*Store*, *Time*, *Product*...) e altre dimensioni specifiche di ogni cubo. È possibile fare analisi su diversi cubi usando i metadati FoodMart: 'Sales', 'Warehouse', 'Store' ecc. Per effettuare i test, è stato usato il cubo 'Sales' (questo ha richiesto la creazione di file CSV appositi per rispondere alle caratteristiche delle dimensioni).

Le dimensioni usate sono le seguenti:

Store : Store Name ⇒ Store City ⇒ Store State ⇒ Store Country

Store Size in SQFT : Store Sqft

Store Type : Store Type

Time :

Time : Month ⇒ Quarter ⇒ Year

Weekly : Day ⇒ Week ⇒ Year

Product : Product Name ⇒ Brand Name ⇒ Product Subcategory ⇒ Product Category ⇒ Product Department ⇒ Product Family

Promotion Media : Media Type

Promotions : Promotion Name

Customers : Name ⇒ City ⇒ State Province ⇒ Country

Education Level : Education Level

Gender : Gender

Marital Status : Marital Status

Yearly Income : Yearly Income

Le misure di interesse per il cubo scelto sono Unit Sales, Store Cost, Store Sales, Sales Count, Customer Count e Promotion Sales.

Alcune note:

- sono stati omessi i nomi dei livelli di massima aggregazione, in quanto non specificati nello schema (si è usato il nome 'All' per tutte le gerarchie);
- la maggior parte delle dimensioni è degenere, ovvero costituita da un solo attributo dimensionale 'non-all': su questi attributi, tuttavia, ha senso applicare predicati di selezione (ad esempio, valutare gli acquisti di clienti maschi, o ad alto reddito);
- nella dimensione temporale sono presenti due gerarchie: queste hanno un attributo in comune ('Year'), ma non all'inizio delle gerarchie. Probabilmente nella gerarchia principale (che prende il nome dalla dimensione) è stato eliminato l'attributo 'Day' - il giorno determina il mese, che determina il trimestre, che determina l'anno. In questo caso ci troveremmo in una gerarchia ramificata, in cui i due rami collimano nell'attributo 'Year': si tratterebbe di una convergenza.

5.2 Distribuzione dei template

Questa serie di test valuta come le sessioni si distribuiscono fra i quattro template (Explorative, Goal Oriented, Just Slice e Slice and Drill). Per la legge empirica del caso (o legge dei grandi numeri), ci si aspetta che, al crescere del numero di sessioni prodotte, la scelta pseudocasuale del template da adoperare si avvicini sempre più ad una distribuzione uniforme.

Da questi test, schematizzati nelle tabelle 5.1, 5.2 e 5.3, si nota l'avvicinamento ad una distribuzione omogenea al crescere del numero di sessioni prodotte dal generatore.

Explorative	Goal Oriented	Just Slice	Slice and Drill
34	18	17	31
23	24	21	32
28	21	19	32
26	19	32	23
38	22	17	23
31	30	22	17
27	30	19	24
28	29	16	27
29	20	26	25
21	24	23	32
28,5	23,7	21,2	26,6
Media: 25		Varianza: 7,74	

Tabella 5.1: Distribuzione dei template in 100 sessioni.

Explorative	Goal Oriented	Just Slice	Slice and Drill
267	240	266	227
251	246	241	262
239	253	233	275
251	271	236	242
239	249	253	259
263	222	251	264
240	255	259	246
262	240	251	247
261	260	225	254
221	262	256	261
249,4	249,8	247,1	253,7
Media: 250		Varianza: 5,63	

Tabella 5.2: Distribuzione dei template in 1.000 sessioni.

Explorative	Goal Oriented	Just Slice	Slice and Drill
2438	2552	2503	2507
2523	2535	2448	2494
2500	2501	2416	2583
2501	2486	2553	2460
2434	2553	2549	2464
2532	2510	2461	2497
2525	2498	2475	2502
2526	2490	2461	2523
2507	2501	2533	2459
2492	2566	2475	2467
2497,8	2519,2	2487,4	2495,6
Media: 2500		Varianza: 137,90	

Tabella 5.3: Distribuzione dei template in 10.000 sessioni.

5.3 Template e tempistiche

Questi test riprendono i precedenti, e aggiungono il calcolo del tempo impiegato all'effettiva generazione delle sessioni. Sono escluse le fasi di creazione del cubo, popolazione delle gerarchie e generazione delle query sorprendenti: tuttavia, sono incluse tutte le fasi di ogni profilo, comprese la creazione delle query iniziali, la creazione delle query finali e l'effettiva generazione, da parte dei template, delle sessioni OLAP. Inoltre, si esclude la fase di scrittura del file di output XML: questa rappresenta il vero collo di bottiglia, giacché la scrittura del suddetto file porta, con 10.000 sessioni, ad un output di circa 6.000.000 di righe.

Explorative	Goal Oriented	Just Slice	Slice and Drill	Tempo (ms)
23	20	30	27	53,19
25	24	25	26	52,76
32	19	21	28	54,54
23	30	25	22	59,07
25	24	27	24	56,83
17	30	31	22	55,83
24	20	32	24	53,06
31	25	24	20	55,15
24	21	28	27	51,73
22	26	30	22	51,15
24,6	23,9	27,3	24,2	54,33
Media: 25		Varianza: 1,83		

Tabella 5.4: Tempi di esecuzione per la creazione di 100 sessioni.

Explorative	Goal Oriented	Just Slice	Slice and Drill	Tempo (ms)
230	249	257	264	154,58
237	242	258	263	153,37
256	252	256	236	135,50
261	266	233	240	141,66
263	259	244	234	152,78
249	245	253	253	149,32
242	247	260	251	147,71
256	246	254	244	156,08
248	244	256	252	172,44
232	237	269	249	152,13
247,4	248,7	254	249,9	151,56
Media: 250		Varianza: 6,12		

Tabella 5.5: Tempi di esecuzione per la creazione di 1.000 sessioni.

Explorative	Goal Oriented	Just Slice	Slice and Drill	Tempo (ms)
2543	2495	2489	2473	1519,71
2487	2580	2449	2484	1443,59
2438	2546	2517	2499	988,75
2602	2464	2509	2425	1241,57
2509	2512	2527	2452	1074,74
2544	2507	2475	2474	1408,81
2473	2439	2579	2509	650,82
2545	2386	2479	2570	683,29
2484	2490	2547	2479	1134,12
2564	2423	2473	2540	661,09
2518,9	2484,2	2504,4	2490,5	1080,65
Media: 2500		Varianza: 178,87		

Tabella 5.6: Tempi di esecuzione per la creazione di 10.000 sessioni.

5.4 Lunghezza delle sessioni

Questi test verificano l'aderenza ai parametri di lunghezza minima e massima di sessione. Per non creare sessioni con lunghezze troppo simili, i template scelgono una lunghezza casuale compresa nel range e producono una lista di query della lunghezza scelta. Non si sono verificati casi di sessioni troppo lunghe: una tale eventualità è prevenuta eliminando alcune query dalla sequenza (operazione ammissibile, giacché non ci sono vincoli sulla complessità del passaggio da una query alla successiva). Può capitare, tuttavia, che alcune sessioni siano più brevi della lunghezza minima: i template in cui accade sono Just Slice e Slice and Drill.

Nel template Just Slice, viene scelta una gerarchia percorribile (ovvero, non vincolata da un segregation attribute e con un numero minimo di livelli) e un suo livello intermedio: iterativamente, si aggiunge un predicato di selezione sul livello scelto della gerarchia, mantenendo la query aggregata sul livello direttamente più dettagliato. Se, una volta finiti i valori, la sessione è troppo corta, si applica un operatore di drill-down e si ricomincia; alla fine, dalla sessione saranno eliminate le query superflue. Il caso patologico sussiste allorché i livelli della gerarchia scelta hanno pochi valori distinti: all'aumentare dell'aggregazione la quantità dei suddetti valori diminuisce, quindi se la somma degli elementi dei due livelli interessati è minore della lunghezza minima di sessione, si ottiene una lista di query più breve della norma.

Nel template Slice and Drill, invece, la lunghezza di una sessione è superiormente limitata dal numero dei livelli delle gerarchie discendibili. Il template verifica quali gerarchie siano percorribili (senza segregation attribute e con un numero minimo di livelli), dopodiché ne sceglie una alla volta e la discende usando gli operatori slice e drill-down. Se la lista di query risultante è troppo lunga, alcune query in coda sono eliminate; tuttavia, non c'è rimedio ad una sessione troppo corta. Per percorrere una gerarchia, il template sposta la query al livello 'non-all' più aggregato, poi percorre tutti i livelli, fino al massimo dettaglio: alla fine del percorso, la query ritorna alla posizione originaria e si procede con un'altra gerarchia. Se la query, prima

della discesa di una gerarchia, è già al livello suddetto, non si crea una ripetizione: la sessione sarà ancora più breve. Riassumendo: una sessione creata con il template Slice and Drill sarà tanto più breve quanto minore sarà la somma dei livelli delle gerarchie percorribili.

I test sono stati eseguiti usando, come parametri locali, lunghezza minima 10 e lunghezza massima 20. Le Figure 5.1 ... 5.12 mostrano la distribuzione delle sessioni secondo la loro lunghezza; la tabella 5.7 ne riassume i risultati.

Le sessioni sono state generate usando entrambi i database, con o senza un segregation attribute. Rimuovendo il segregation attribute, la quantità di sessioni troppo brevi create su schema IPUMS diminuisce significativamente: giacché lo schema contiene cinque gerarchie, di lunghezza $\{5, 4, 4, 2, 2\}$, è molto probabile che il segregation attribute 'cada' su una gerarchia profonda. Se questo accade, la suddetta gerarchia non è percorribile dal template Slice and Drill, quindi la lunghezza massima della sessione diminuisce. Nella Figura 5.1 si nota il caso limite di una sessione costituita da una lista di cinque query: l'unica situazione in cui si può verificare è quando la query iniziale è vincolata da un segregation attribute sulla gerarchia OCCUPATION ed è aggregata al massimo livello 'non-all' sulle gerarchie CITY e RACE. In questo caso, non è possibile percorrere la gerarchia OCCUPATION (vincolata) né le gerarchie SEX e YEAR (numero insufficiente di livelli), quindi le uniche gerarchie percorribili sono RACE e CITY: se la query è aggregata sui livelli MRN e REGION, rispettivamente, non vengono generate query ripetute e il percorso di discesa è 'lungo' solo due query. Il suddetto percorso si ottiene ponendo un predicato di selezione a livello $k + 1$ e raggruppando a livello k, iterativamente: il predicato di selezione può essere posto solo in un livello intermedio, quindi su una gerarchia con quattro livelli ci sono due passaggi possibili.

Anche in assenza di segregation attribute, esiste un limite superiore alla lunghezza delle sessioni di tipo Slice and Drill. Nel caso favorevole in cui la query iniziale non sia aggregata al massimo livello 'non-all' delle gerarchie OCCUPATION, RACE e CITY, su ogni gerarchia vengono eseguiti i seguenti

passaggi base:

- spostare la query al penultimo livello di aggregazione;
- finché non si raggruppa al livello di massimo dettaglio, porre un predicato sul livello corrente e discendere la gerarchia.

Per ogni gerarchia, le query generate in questo modo sono pari al livello della gerarchia meno 1. Considerando la query iniziale, le sessioni Slice and Drill hanno una lunghezza massima pari a:

$$1 + (5 - 1) + (4 - 1) + (4 - 1) = 11.$$

Diversamente dai test sullo schema IPUMS, la presenza di segregation attribute non influisce sulla percentuale di sessioni troppo brevi: questo particolare predicato limita il numero di gerarchie percorribili, ma il numero di tali gerarchie (13) è comunque sufficiente a garantire la lunghezza minima delle sessioni create con il template Slice and Drill. Anche se una delle cinque gerarchie abbastanza profonde (Store, Time, Weekly, Product, Customers) fosse limitata da un segregation attribute, percorrendo le altre sarebbe possibile raggiungere la lunghezza minima di sessione.

Le sessioni generate per lo schema FoodMart presentano lunghezze anomale (più brevi di 10 query) esclusivamente nelle sessioni di tipo Just Slice. Anche se nella cernita vengono escluse le gerarchie non sufficientemente profonde, può capitare che la gerarchia scelta sia formata da livelli con pochi valori distinti: in questo caso, la lunghezza della sessione dipende dalla somma delle quantità di valori distinti di due livelli della gerarchia scelta.

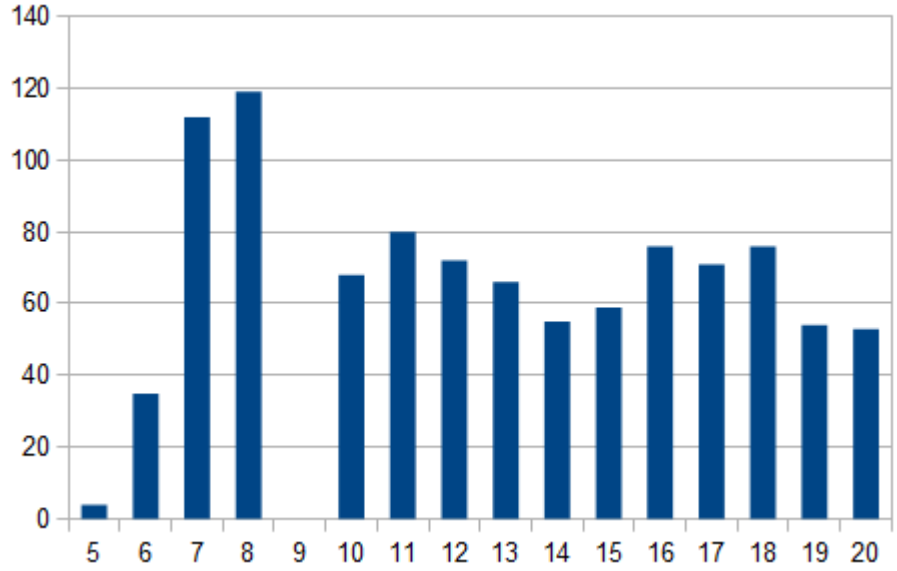

Figura 5.1: Distribuzione di 1.000 sessioni in base alla lunghezza, su schema IPUMS, in presenza di segregation attribute.

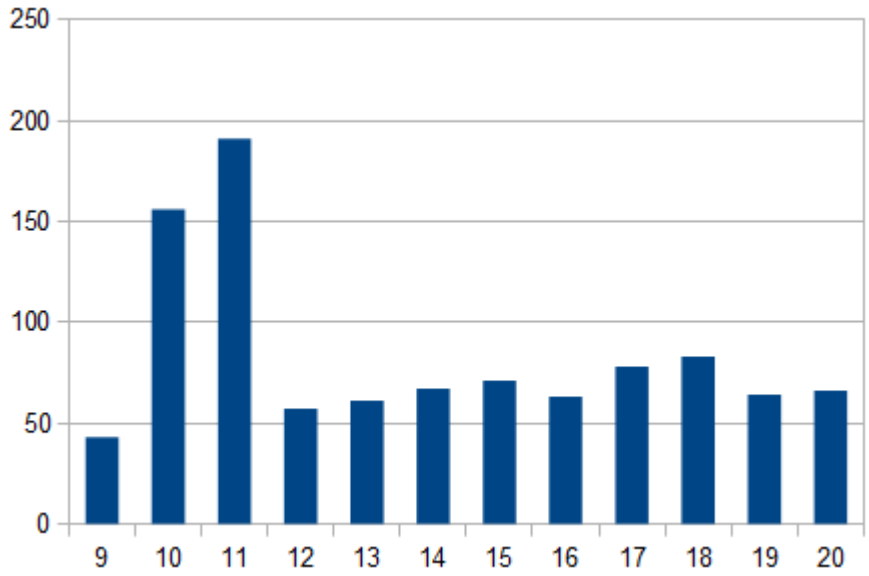

Figura 5.2: Distribuzione di 1.000 sessioni in base alla lunghezza, su schema IPUMS, in assenza di segregation attribute.

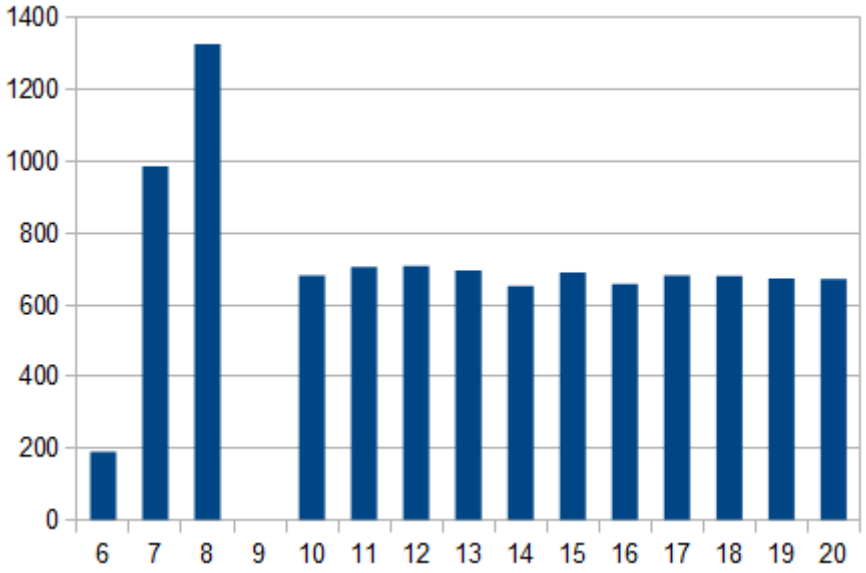

Figura 5.3: Distribuzione di 10.000 sessioni in base alla lunghezza, su schema IPUMS, in presenza di segregation attribute.

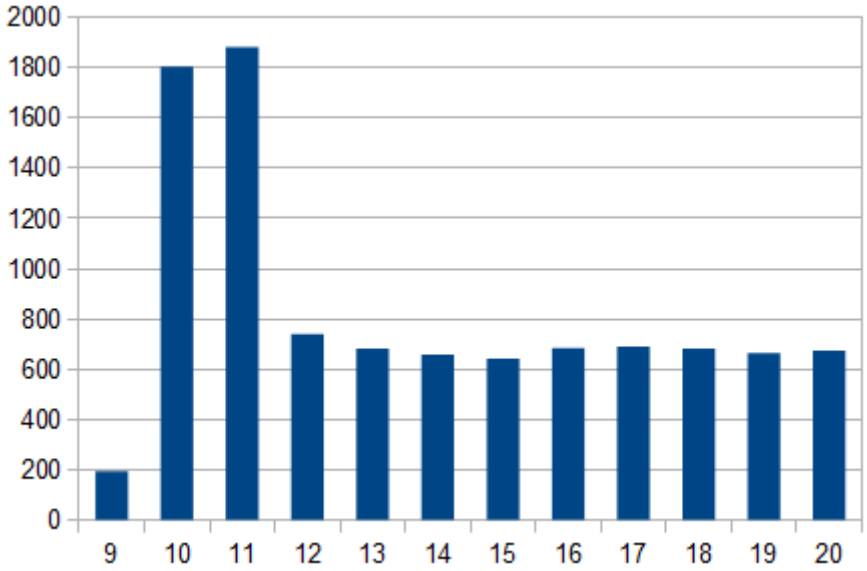

Figura 5.4: Distribuzione di 10.000 sessioni in base alla lunghezza, su schema IPUMS, in assenza di segregation attribute.

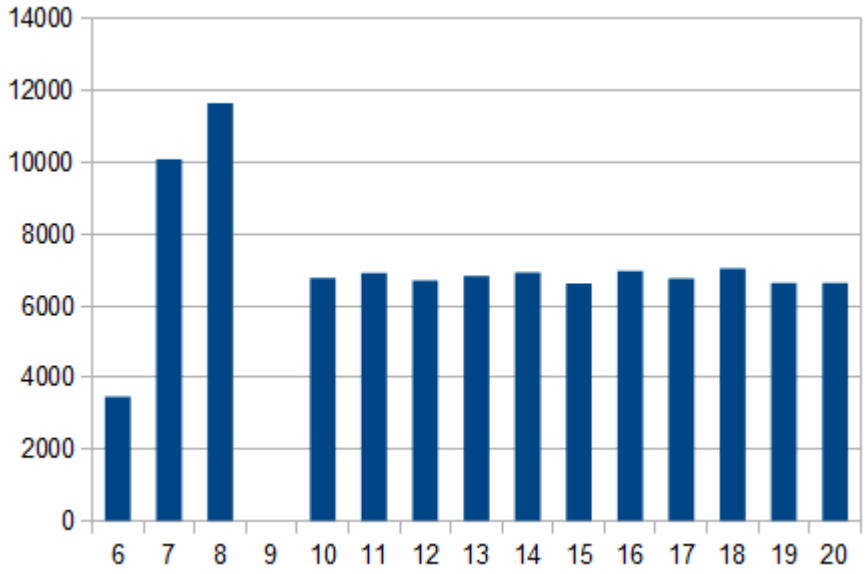

Figura 5.5: Distribuzione di 100.000 sessioni in base alla lunghezza, su schema IPUMS, in presenza di segregation attribute.

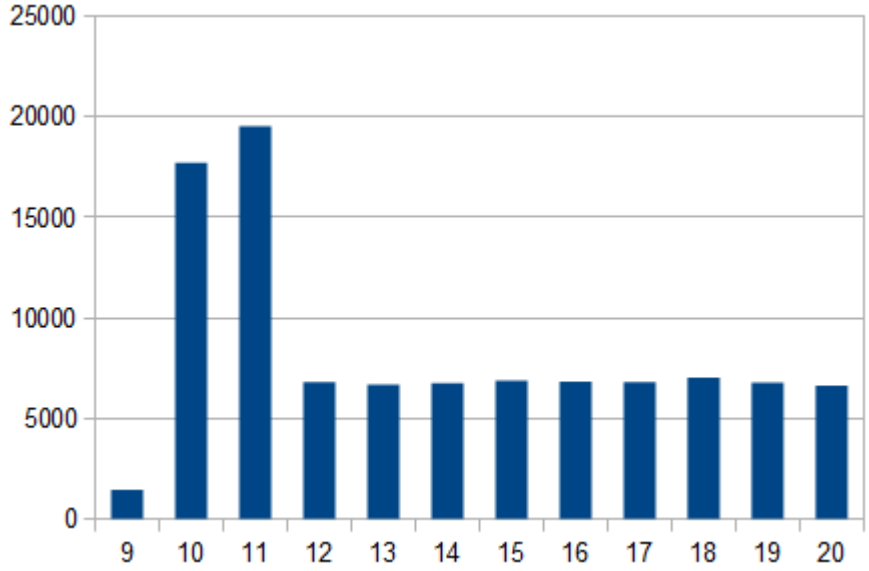

Figura 5.6: Distribuzione di 100.000 sessioni in base alla lunghezza, su schema IPUMS, in assenza di segregation attribute.

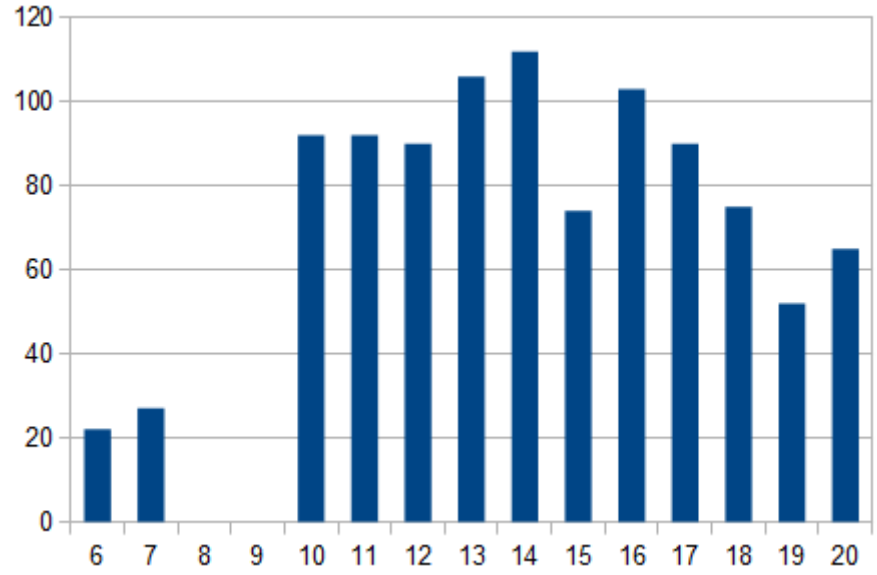

Figura 5.7: Distribuzione di 1.000 sessioni in base alla lunghezza, su schema FoodMart, in presenza di segregation attribute.

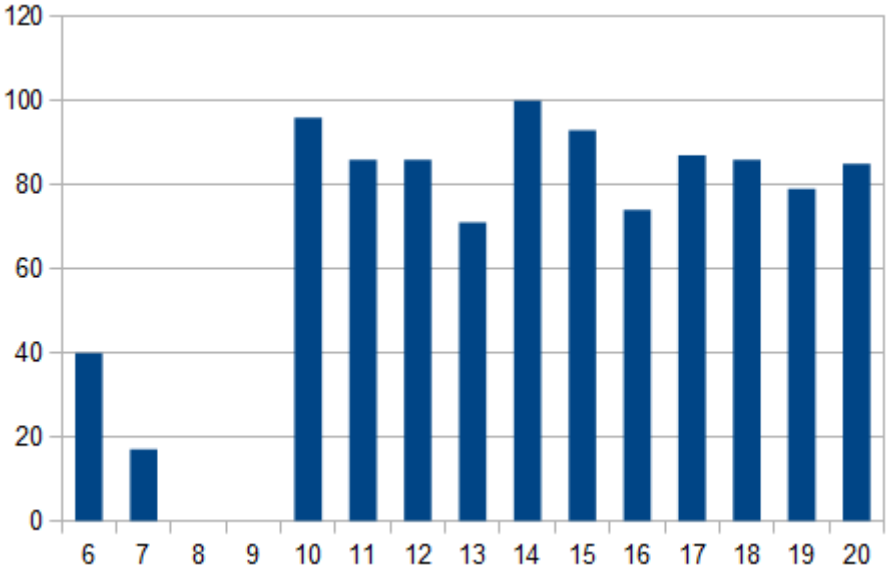

Figura 5.8: Distribuzione di 1.000 sessioni in base alla lunghezza, su schema FoodMart, in assenza di segregation attribute.

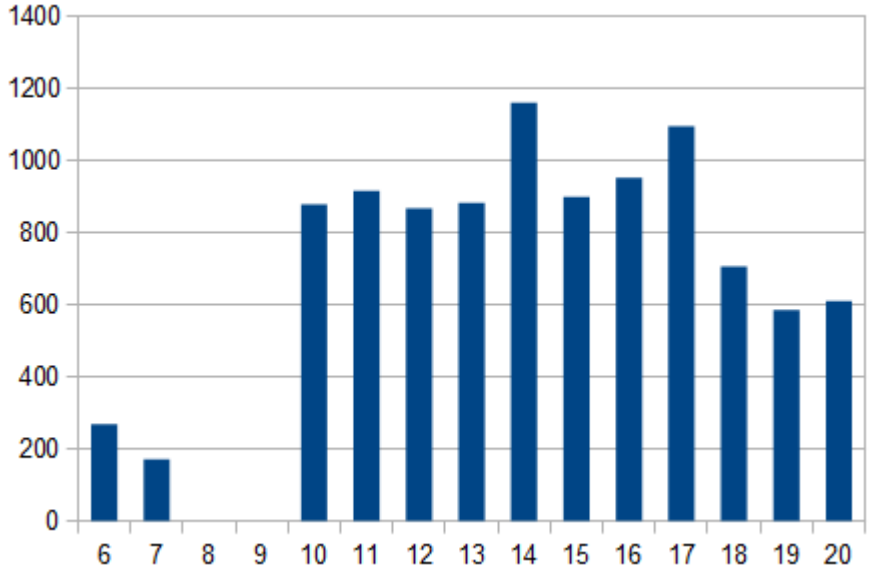

Figura 5.9: Distribuzione di 10.000 sessioni in base alla lunghezza, su schema FoodMart, in presenza di segregation attribute.

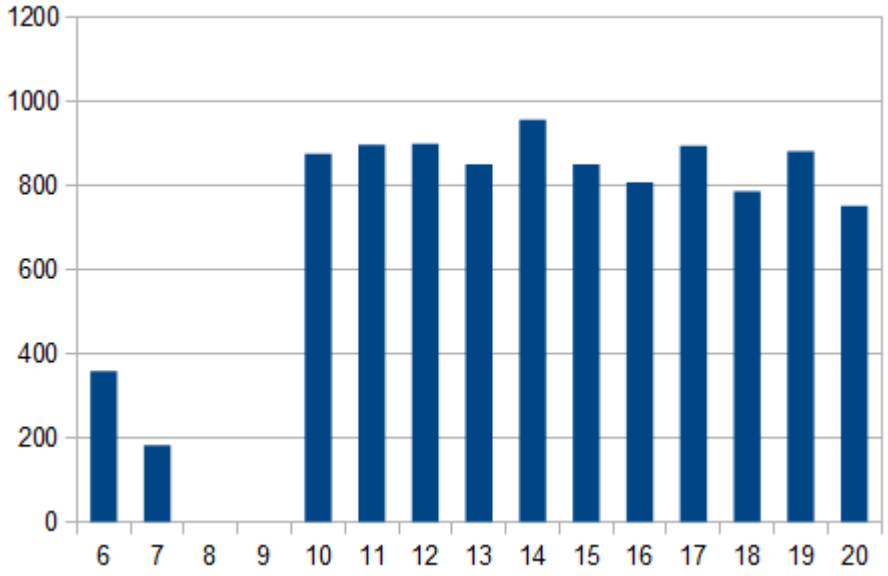

Figura 5.10: Distribuzione di 10.000 sessioni in base alla lunghezza, su schema FoodMart, in assenza di segregation attribute.

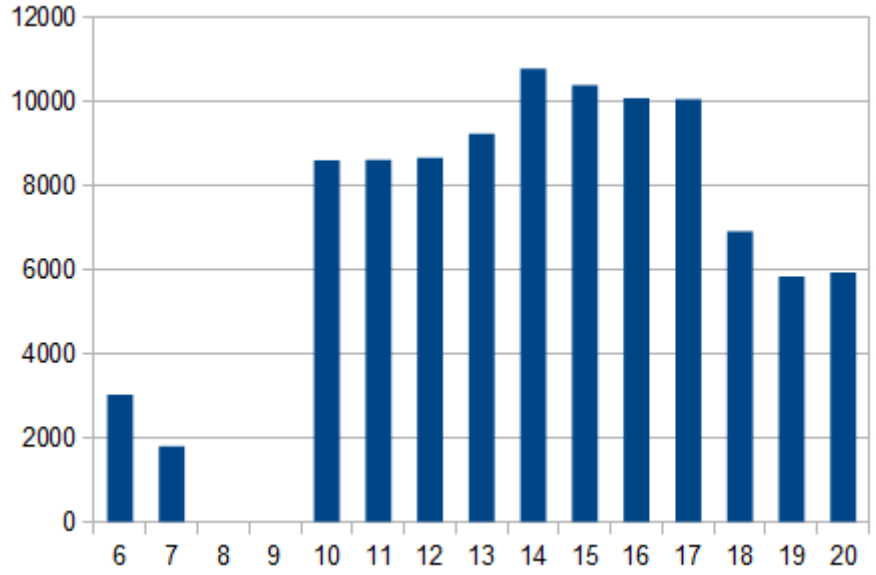

Figura 5.11: Distribuzione di 100.000 sessioni in base alla lunghezza, su schema FoodMart, in presenza di segregation attribute.

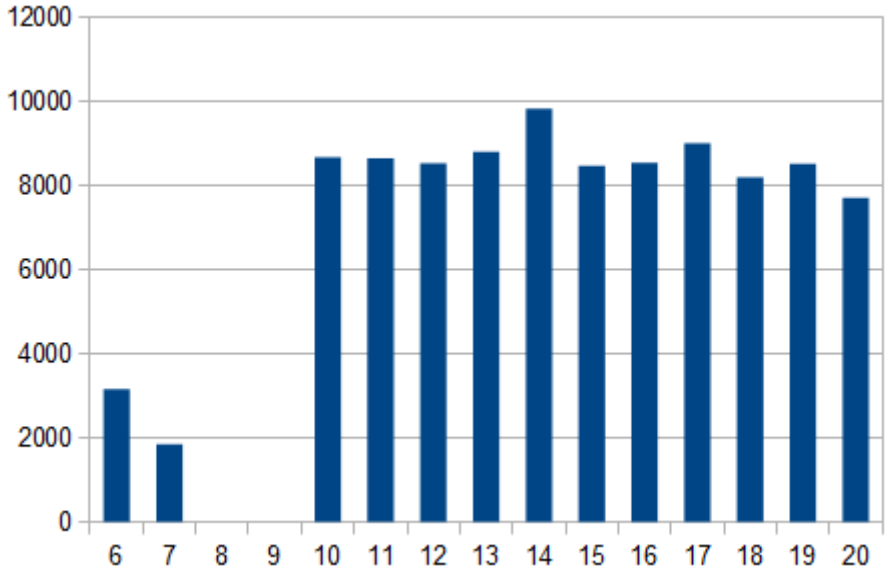

Figura 5.12: Distribuzione di 100.000 sessioni in base alla lunghezza, su schema FoodMart, in assenza di segregation attribute.

Schema	Sessioni generate	Con segregation		Senza segregation	
		In range	Fuori range	In range	Fuori range
IPUMS	1.000	73%	27%	95,7%	4,3%
	10.000	74,96%	25,04%	98,06%	1,94%
	100.000	74,81%	25,19%	98,53%	1,47%
FoodMart	1.000	95,1%	4,9%	94,3%	5,7%
	10.000	95,6%	4,4%	94,59%	5,41%
	100.000	95,17%	4,83%	94,99%	5,01%

Tabella 5.7: Percentuali di sessioni rientranti nel range fissato, in base a schema e presenza di segregation attribute.

5.5 Similarità tra sessioni

I test che seguiranno, effettuati su schema IPUMS, valutano la similarità di sessioni create usando un solo template: i risultati possono essere utili alla ricerca nell'ambito delle applicazioni di query recommendation [2] o di query formulation support [4]. In [1] si può trovare una panoramica esaustiva sulle possibili misure di similarità tra sessioni OLAP. Serve un approccio a due livelli: verificare la similarità tra query e la similarità tra sessioni. La similarità tra query è basata sulla combinazione di tre diversi coefficienti, relativi a group-by, misure e predicati; la similarità tra sessioni deve prendere in considerazione i loro elementi e l'ordine con cui si presentano.

Slice and Drill: il grafico di queste sessioni (riportato in Figura 5.13) mostra un'alta similarità, costante in diverse sottosequenze della lista di query. Lo schema IPUMS, come già descritto, contiene tre gerarchie su cui poter applicare il drill-down: a parità di query iniziale, esistono solo sei modi per percorrerle tutte (cambia il loro ordinamento). Questo fattore, unito all'invariabilità delle misure e dei predicati di selezione appartenenti a gerarchie non percorse, contribuisce a rendere uniformi le sessioni, quindi si può prevedere un'alta similarità.

Just Slice: la Figura 5.14 schematizza le sessioni create con template Just Slice. La similarità è alta anche in questo caso: i fattori invariabili sono le misure e i predicati di selezione presenti sulle gerarchie inalterate (e, diversamente dal template Slice and Drill, si percorre una sola gerarchia). A causa della poca profondità delle gerarchie, l'operatore di slice viene applicato frequentemente agli stessi livelli di gerarchia: il discrimine, in questo caso, è dato dall'ordine con cui i valori distinti vengono scelti per il predicato di selezione. La funzione di similarità tra query, tuttavia, registra predicati di selezione basati sullo stesso livello della gerarchia, quindi la diversità dei valori distinti passa in secondo piano.

Explorative: i test effettuati su questo template (schematizzati in Figura 5.15) mostrano un'oscillazione sulla similarità tra sessioni, con un picco nella zona centrale. Questo fenomeno è attribuibile al metodo di creazione delle

sessioni Explorative: dalla query iniziale si converge verso una query sorprendente (scelta come più 'vicina', in precedenza, tra le query sorprendenti), poi si evolve in modo casuale. Da query iniziali diverse tra loro è possibile convergere verso la stessa query sorprendente: a maggior ragione, da una stessa query iniziale si convergerà sempre verso la più vicina query sorprendente, e questo porta a delle similarità. Dopo il picco di media-sessione, c'è un calo di affinità: non solo le evoluzioni random delle query rendono imprevedibile il percorso della sessione, ma le sessioni hanno lunghezze diverse, e la similarità tende a diminuire. Infatti, a prescindere dalla quantità di query costituenti una sessione, tutte le sessioni passano vicino ad una query sorprendente, ma la restante parte della sessione può variare molto, sia in lunghezza sia in contenuti.

Goal Oriented: queste sessioni mostrano un coefficiente di similarità variabile, meno marcato rispetto alle sessioni create con il template Explorative (vedi Figura 5.16). Le sessioni Goal Oriented che partono dalla i-esima query iniziale terminano sempre nella i-esima query finale, indipendentemente dal numero di query intermedie: tuttavia, ad abbassare il coefficiente di similarità di tali sessioni sono la lunghezza (una sessione di 10 query ed una di 20 query, per quanto identiche agli estremi, sono molto differenti) e l'ordine con cui le query intermedie si succedono all'interno della sessione. Perdipiù, in presenza di sessioni troppo corte viene introdotto del 'rumore' (query che deviano dal cammino minimo verso la query finale), ulteriore fattore di diversità tra sessioni, seppur identiche agli estremi.

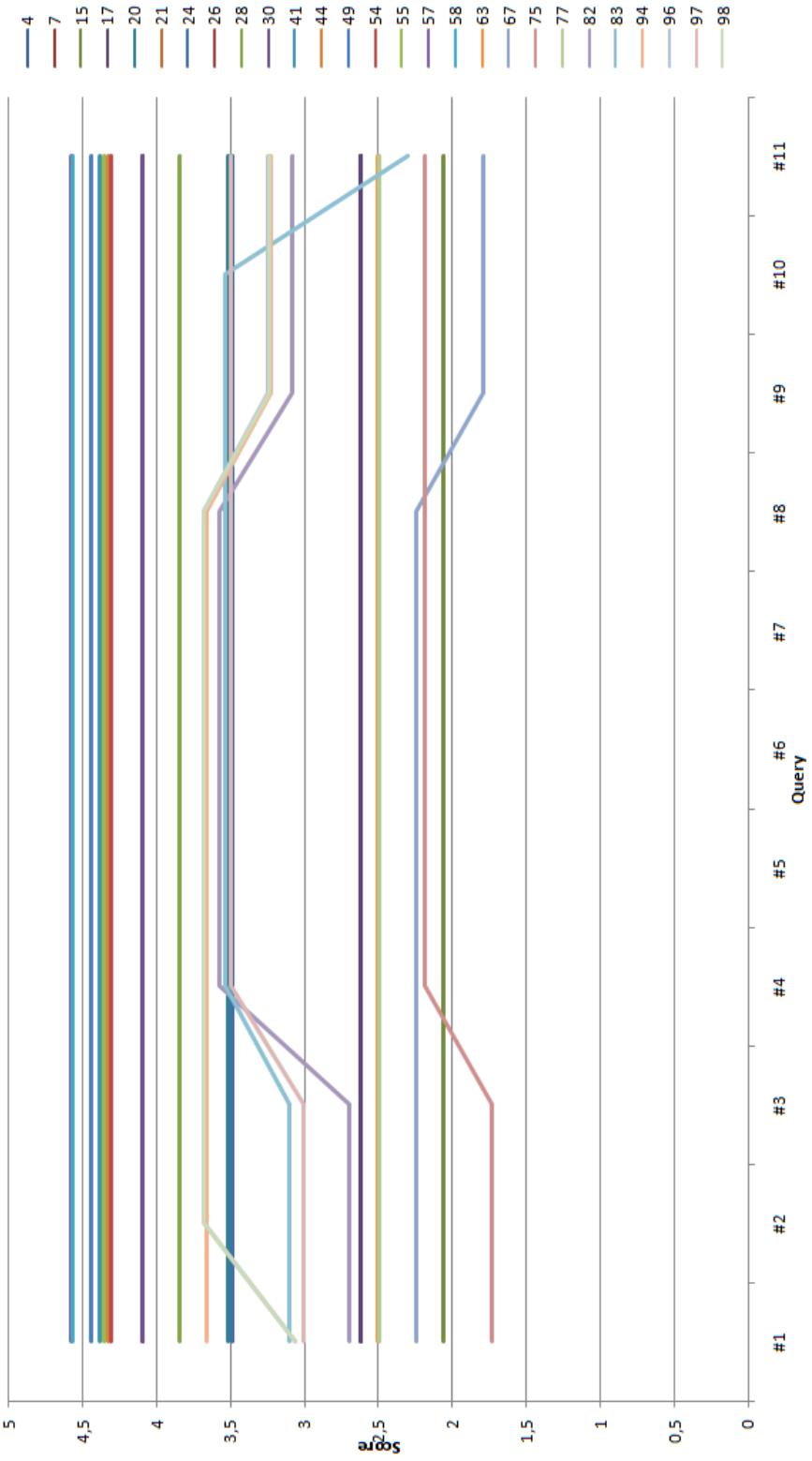

Figura 5.13: Coefficienti di similarità delle sessioni create con template Slice and Drill, lungo la lista di query.

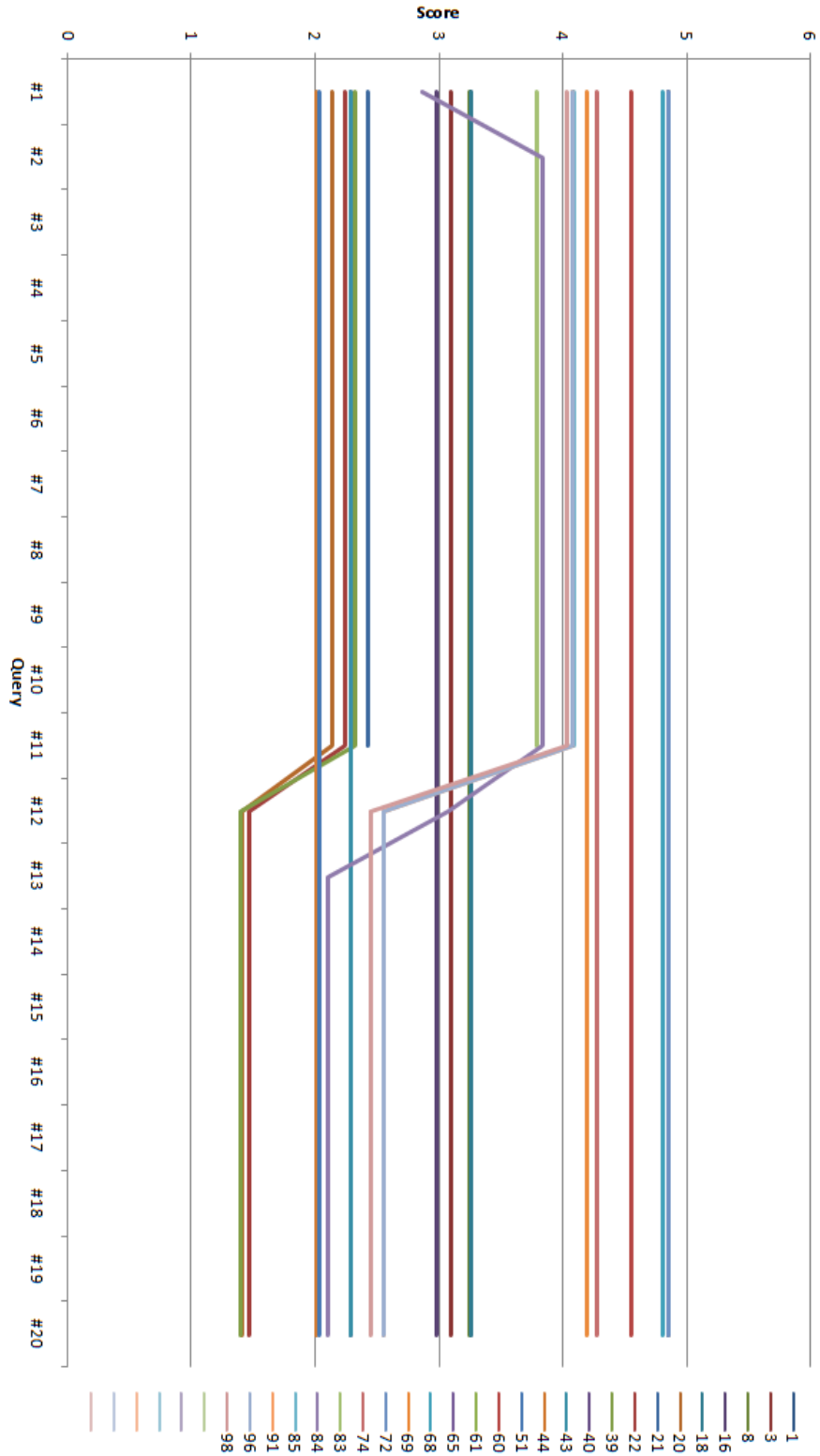

Figura 5.14: Coefficienti di similarità delle sessioni create con template Just Slice, lungo la lista di query.

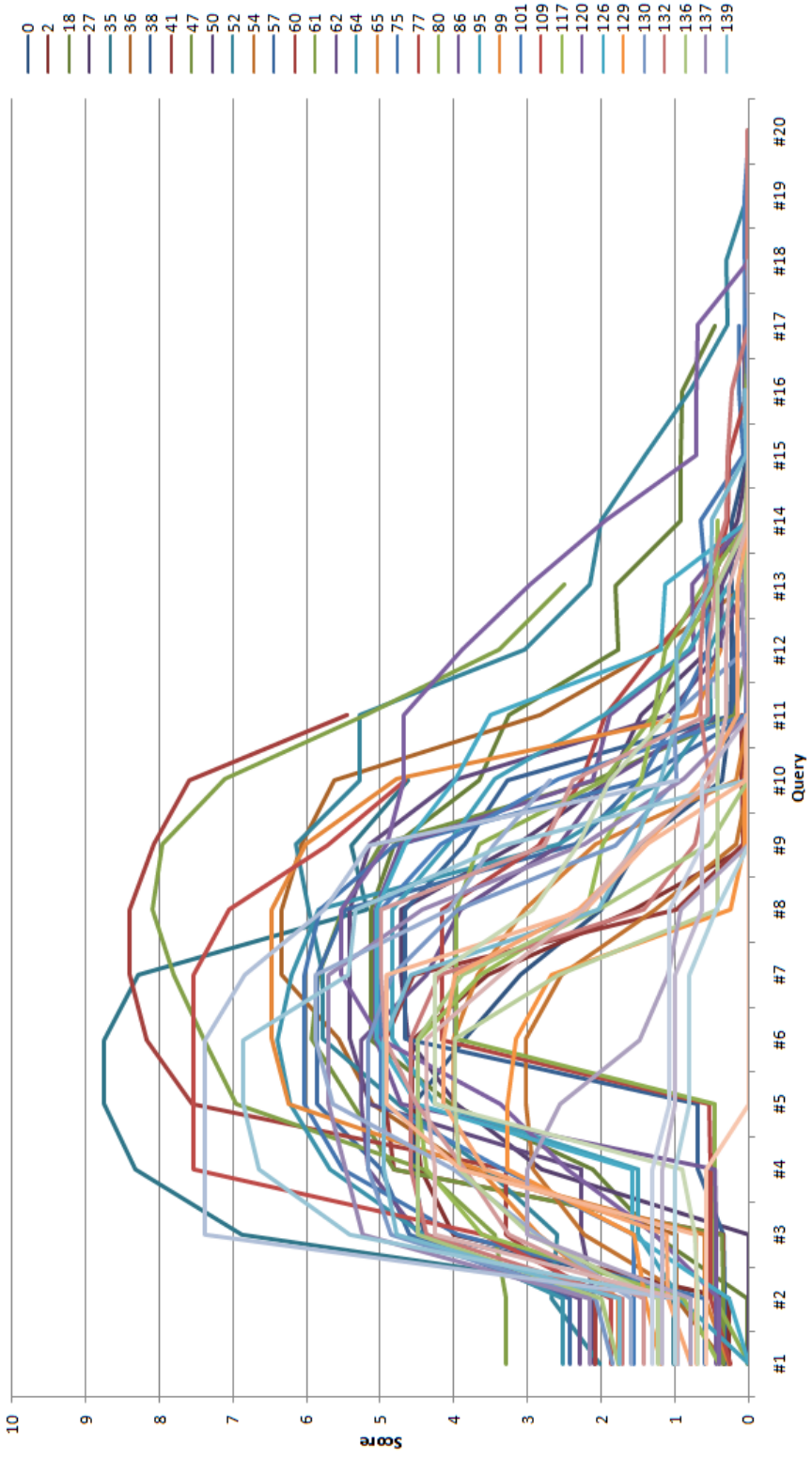

Figura 5.15: Coefficienti di similarità delle sessioni create con template Explorative, lungo la lista di query.

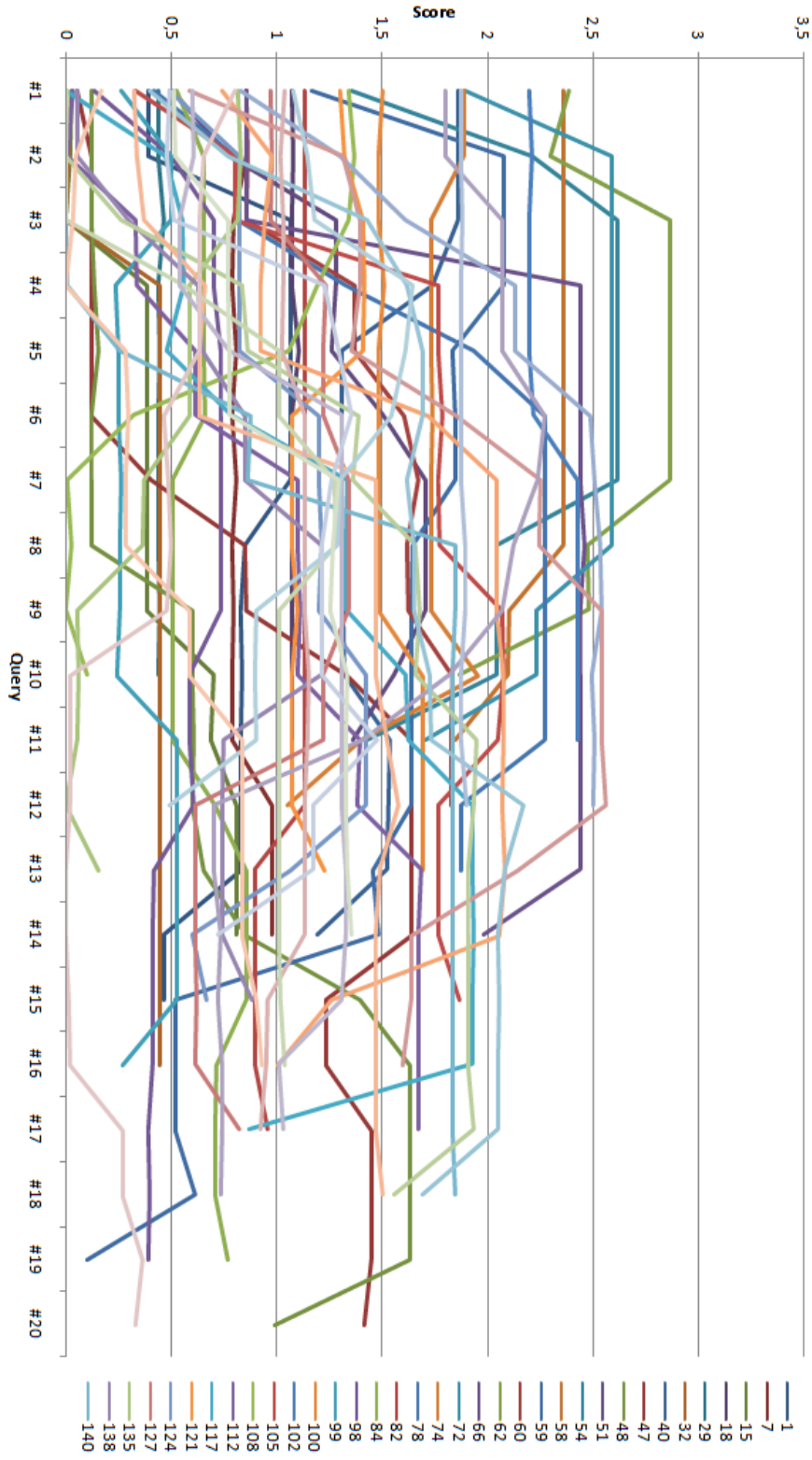

Figura 5.16: Coefficienti di similarità delle sessioni create con template Goal Oriented, lungo la lista di query.

Conclusioni

Questo progetto di tesi riguarda la progettazione e realizzazione di uno strumento per generare carichi di lavoro ('workload') OLAP in base a determinati parametri definiti dall'utente. L'utente può impostare un certo numero di profili: un profilo simula una figura all'interno di un'azienda, con una determinata visibilità dei dati e obiettivi più o meno prestabiliti. Per ogni profilo, il generatore produce una serie di sessioni OLAP: queste ultime sono sequenze ordinate di query OLAP, dove il passaggio tra una query e la successiva è arbitrariamente complesso. Il generatore fa riferimento a quattro 'template' di creazione delle sessioni, che simulano il percorso di analisi di un utente in base alle proprie esigenze. Il generatore utilizza un file di metadati XML per definire la struttura di un cubo multidimensionale, legge un determinato numero di file CSV per popolare il cubo e sintetizza le sessioni create in un altro file XML.

L'utilità di questo progetto risiede nella necessità di lavorare su carichi di lavoro realistici, per poter valutare la bontà degli algoritmi di ottimizzazione delle prestazioni OLAP (come la *query recommendation*). Sessioni OLAP casuali o con pochi controlli non rappresentano adeguatamente una simulazione di analisi del mondo reale.

Sia per non rendere il progetto troppo articolato, sia per i contenuti del file di metadati usato per definire un cubo (aderente alle specifiche del motore OLAP Mondrian - vedi Cap. 1), diverse funzionalità delle analisi OLAP non sono espresse: ad esempio, l'uso di costrutti avanzati del DFM, come convergenze, attributi cross-dimensionali ed archi multipli. Inoltre, non c'è

una caratterizzazione completa delle misure di un DFM, di cui non si esprime l'additività. C'è, tuttavia, la gestione del branching (gerarchie non lineari), con annessi rigorosi controlli sulla correttezza del group-by set delle query.

Un altro possibile fronte di miglioramento può essere l'aggiunta di una interfaccia grafica all'applicativo: l'utente potrebbe così impostare i parametri con una form, anziché da linea di comando. In più, per velocizzare il caricamento di profili usati spesso, sarebbe possibile memorizzarne i parametri (su un file XML, per esempio). Perdipiù, per raccogliere le informazioni sulle dimension table e il file di metadati non sarebbe necessario avere tutti i documenti nella cartella dell'applicativo: l'utente potrebbe scegliere il percorso in cui si trovano i suddetti file.

Bibliografia

[1] Aligon J., Golfarelli M., Marcel P., Rizzi S., Turricchia E. *Similarity measures for OLAP sessions*. Knowledge and Information Systems, 2013.

[2] Giacometti A., Marcel P., Negre E., Soulet A. *Query recommendations for OLAP discovery driven analysis*. Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP 2009, pp. 81-88.

[3] Golfarelli M., Maio D., Rizzi S. *The Dimensional Fact Model: a Conceptual Model for Data Warehouses*. International Journal of Cooperative Information Systems, vol. 07, issue 02n03, 1998.

[4] Khoussainova N., Kwon Y., Balazinska M. and Suciu D. (2010), *SnipSuggest: Context-aware autocompletion for SQL*, PVLDB 4(1) 2010, pp. 22-33.

[5] Wrembel R., Koncilia Christian. *Data Warehouses and OLAP - Concepts, Architectures and Solutions*, 2007.

[6] http://mondrian.pentaho.com/documentation/installation.php#2_Set_up_test_data

[7] http://mondrian.pentaho.com/documentation/schema.php#Cubes_and_dimensions

[8] https://source.at.northwestern.edu/svn/os/bbgint/trunk/source/common-src/au/com/bytecode/opencsv/CSVParser.java

[9] https://source.at.northwestern.edu/svn/os/bb-gint/trunk/source/common-src/au/com/bytecode/opencsv/CSVReader.java

[10] https://international.ipums.org/international/

[11] http://www.java.com/

[12] http://www.saxproject.org/

[13] http://www.tpc.org/

[14] http://www.uml.org/

[15] http://www.visual-paradigm.com/

[16] http://www.w3schools.com/xpath/