

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



*Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione*

Progetto per il corso di new generation database

Professoressa:
CLAUDIA DIAMANTINI

Studenti:
SCALELLA SIMONE
YIHANG ZHANG

ANNO ACCADEMICO 2022-2023

Il codice è disponibile al seguente indirizzo: https://github.com/Simone-Scalella/Progetto_NewGenerationDB

Nella cartella **image** sono presenti tutte le immagini ottenute durante la realizzazione del progetto.

Nella cartella **experiments-set-up-docker** sono contenuti tutti i file per avviare l'architettura.

Nella cartella **experiments** sono contenute le query utilizzate e i risultati ottenuti.

Nella cartella **experiments/obda-mixer** sono presenti i vecchi risultati ottenuti dagli autori e i nuovi risultati ottenuti durante il progetto.

Nella cartella **experiments/obda-mixer-ngdb** sono presenti i risultati ottenuti dalle nuove interrogazioni.

Nella cartella **experiments/obdf-resultAnalysis** sono presenti i risultati in formato csv.

Nelle cartelle chiamate **Templates** e **Templates+NomeSorgente**, dentro **experiments/obda-mixer** sono presenti le query sparql e quelle sql utilizzate.

Indice

Indice	ii
1 Introduzione	1
1.1 Obiettivi del progetto	1
1.2 Workflow	1
1.3 Sintesi dell'articolo	2
2 Strumenti e Metodi	3
2.1 Linguaggi e librerie	3
2.2 Strumenti software	3
2.3 Link all'articolo di riferimento	4
3 Sviluppo del progetto	5
3.1 Implementazione dell'architettura	5
3.1.1 Risoluzione dei problemi tecnici	5
3.1.2 Risoluzione dei problemi di docker	7
3.2 Esecuzione delle query	9
3.2.1 Utilizzo degli script per eseguire le query	10
3.2.2 Analisi dei risultati	12
3.3 Esecuzione delle nuove query	12
3.3.1 Reverse engineering degli schemi	13
3.3.2 Scrittura delle nuove interrogazioni	16
3.4 Esecuzione delle query modificate	16
3.4.1 Modifica delle query già esistenti	16
3.4.2 Analisi dei nuovi risultati	16
4 Risultati	18
5 Conclusioni e Sviluppi futuri	25
5.1 Sviluppi futuri	25
5.2 Conclusioni	25
Bibliografia	26

Elenco delle figure	27
Elenco delle tabelle	28

Capitolo 1

Introduzione

Nella seguente relazione verrà illustrato lo svolgimento di questo progetto.

1.1 Obiettivi del progetto

Questo progetto prevede diversi obiettivi, che sono lo studio e la comprensione dell'architettura **OBDF**, replicazione dei risultati proposti nell'articolo e scrittura di nuove query.

1.2 Workflow

Di seguito riportiamo il workflow seguito per la realizzazione di tale progetto.

1. **Implementazione dell'architettura:** Durante questa fase si è iniziato con lo studio dell'architettura presentata nell'articolo. Successivamente, utilizzando le linee guida presenti su github e varie documentazioni si è riusciti a risolvere diversi problemi e a implementare tutta l'architettura.
2. **Esecuzione delle query:** Durante questa fase si è proceduto a mandare in esecuzione le query proposte nell'articolo, risolvendo i problemi d'esecuzione che si sono presentati.
3. **Esecuzione delle nuove query:** Durante questa fase si è proceduto con un'operazione di reverse engineering e si è studiata l'architettura per poter realizzare delle nuove interrogazioni.
4. **Esecuzione delle query modificate:** Durante questa fase si è proceduto con la modifica di alcune delle interrogazioni proposte dagli autori dell'articolo per testare ulteriormente l'efficienza delle ottimizzazioni.

1.3 Sintesi dell'articolo

Il seguente progetto si basa sullo studio e la ricerca proposti nell'articolo **Ontology-based Data Federation – A Framework Proposal**.

Per comprendere a pieno il lavoro svolto in questo progetto è necessario fare un breve riassunto dell'articolo. Per ulteriori approfondimenti si invita a consultare l'articolo originale, il cui link viene riportato nel capitolo successivo.

In questo articolo viene formalmente introdotta la nozione di **federazione dei dati basata su ontologie** (OBDF) per denotare un framework che combina ontology-based-data-access (OBDA) con un livello di federazione dei dati in cui più fonti eterogenee sono virtualmente esposte come un singolo database relazionale. Vengono proposte nuove tecniche per rendere più efficiente la risposta alle query. Queste tecniche sono convalidate attraverso un'ampia valutazione sperimentale.

L'ottimizzazione delle query in **OBDF** si basa sul fatto che l'ontologia e le mappature contengono informazioni per guidare l'utilizzo dei suggerimenti (**hint**) sui dati.

La procedura di ottimizzazione si compone di due fasi, nella prima abbiamo una parte di precalcolo dei suggerimenti offline e nella seconda abbiamo un'ottimizzazione online della traduzione.

L'utilizzo dei suggerimenti permette di eliminare operazioni inutili e ridotti, che sono join vuoti, unioni ridondanti, sostituzione di join e join esterni con espressioni, infine, la sostituzione dei join federati nelle query sql in richieste su viste materializzate.

Questo lavoro introduce all'impostazione dei sistemi **OBDF** e studia il problema dell'ottimizzazione delle query.

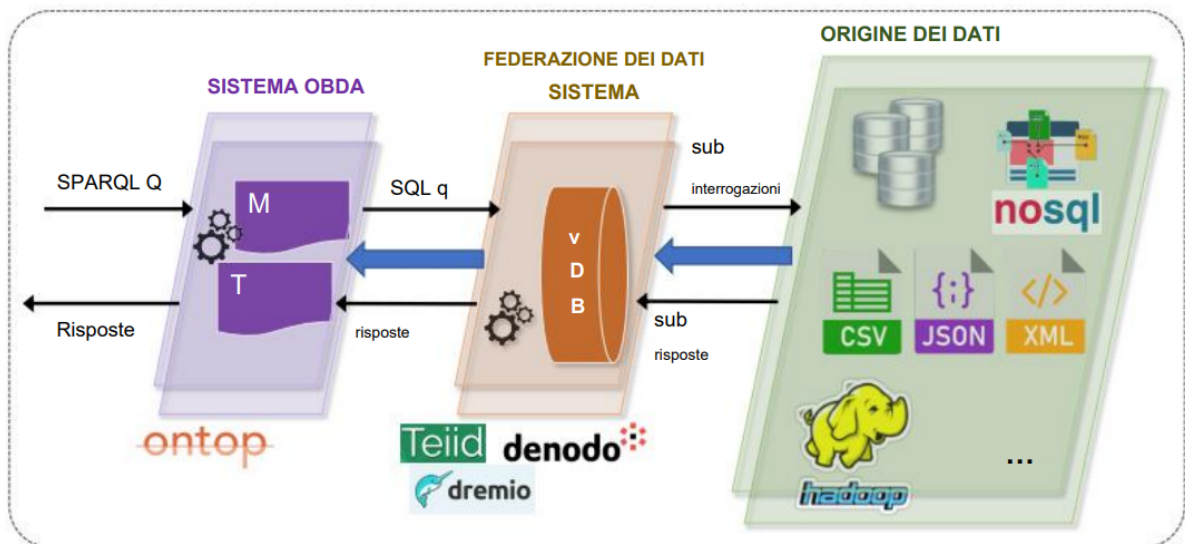


Figura 1.1: framework OBDF e procedura di risposta alle query.

Capitolo 2

Strumenti e Metodi

2.1 Linguaggi e librerie

Tutto il codice presente nel progetto è stato scritto in linguaggio Python[1], in linguaggio Java[2] e con comandi per la shell di Linux.

Per il linguaggio Python sono state utilizzate le seguenti librerie:

- **Pandas:** strumento per la manipolazione e l'analisi dei dati veloce e semplice da utilizzare.
- **Numpy:** Questa è una libreria open source per il linguaggio di programmazione Python, che aggiunge supporto a grandi matrici e array multidimensionali insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati.
- **Matplotlib:** Questa è una libreria per la creazione di grafici per il linguaggio di programmazione Python.
- **Math:** Questa è una libreria che fornisce l'accesso a diverse funzioni matematiche.

2.2 Strumenti software

Per lo sviluppo di questo progetto, sono stati utilizzati i seguenti strumenti software:

- versione Python 3.11[1], utilizzata per il progetto.
- Ubuntu[3], per eseguire gli script presenti nel progetto è necessario utilizzare il sottosistema Linux per Windows. Come sistema operativo è stato usato Ubuntu.
- JDK 19[2], è l'insieme del software e degli strumenti necessari per compilare, eseguire e testare applicazioni scritte nel linguaggio Java. Deve essere installato sul sistema operativo Ubuntu.

- Docker Desktop[4], necessario per poter fare il set up dell'architettura.

2.3 Link all'articolo di riferimento

Il seguente progetto si basa sull'articolo **Ontology-based Data Federation – A Framework Proposal**. Di seguito riportiamo i link all'articolo originale.

- Link all'articolo.
- Link per poter accedere a una versione più completa ed esplicativa dell'articolo
- Link per poter accedere ai precedenti risultati che hanno ottenuto gli autori dell'articolo.
- Link per poter accedere al progetto caricato su github dagli autori dell'articolo.

Capitolo 3

Sviluppo del progetto

Lo sviluppo del progetto è stato suddiviso in quattro fasi principali:

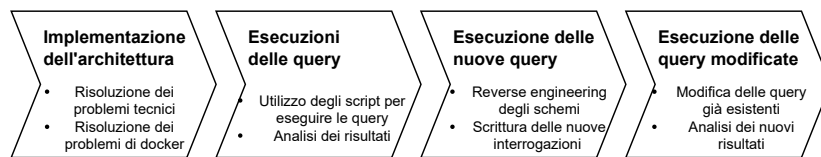


Figura 3.1: Flowchart

Di seguito approfondiamo la descrizione delle singole fasi.

3.1 Implementazione dell'architettura

3.1.1 Risoluzione dei problemi tecnici

La prima fase di questo progetto ha previsto un'operazione di fork del progetto caricato sul sito **github** dagli autori dell'articolo.

Si è proceduto, utilizzando le linee guida per l'installazione presenti nella repository del progetto, a eseguire le operazioni necessarie per implementare l'architettura.

Generazione dei dati e del file di configurazione

Come prima cosa si è aperto il terminale di windows, selezionando la directory **experiments-set-up-docker** come directory di lavoro. Successivamente, si è proceduto a eseguire il comando:

```
1 ./bin/generate-data scale
```

Per poter eseguire questi script su windows si è dovuti procedere a modificare l'estensione di tutti questi file in **nome_file.sh**, altrimenti, il sistema operativo non è in grado di eseguirli e dà errore.

L'esecuzione di questo comando ha portato alla risoluzione di una seconda problematica. Infatti, questo è un file.sh, e tutti i file **.sh** che erano presenti all'interno del progetto avevano come impostazione della sequenza di fine riga il valore **CTLF**, questo in windows genera un problema, in quanto non è in grado di leggere correttamente i token, cioè, i simboli presenti in questo file.

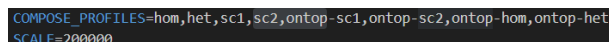
Utilizzando **VScode** si è proceduto a cambiare il valore dell'impostazione di tutti i file .sh in **LF**.

Al termine dell'esecuzione del comando vengono creati tutti i dati utilizzati dall'architettura. L'esecuzione di questo comando richiede molto tempo, inoltre, è stato necessario controllare i dati generati, in quanto alcuni elementi possono essere generati male e risultano corrotti.

Successivamente, si è proceduto con la creazione del documento di configurazione per docker, utilizzando il comando:

```
1 cp .env.example .env
```

Questo file è molto importante, al suo interno sono presenti due voci, nelle quali bisogna specificare i nomi dei container che devono essere istanziati e la dimensione del dataset con cui si vuole lavorare. Riportiamo un esempio nella seguente immagine.



```
COMPOSE_PROFILES=hom,hetsc1,sc2,ontop-sc1,ontop-sc2,ontop-hom,ontop-het
SCALE=200000
```

Figura 3.2: File di configurazione

Avvio di Docker

Per avviare l'applicazione, scaricando le immagini e i contenitori richiesti bisogna eseguire il comando:

```
1 docker-compose up
```

Nel docker file sono presenti dei profili, che vengono assegnati a uno o più container per gestirli. I profili hom ed het utilizzano diverse risorse, mentre, il profilo sc1 ed sc2 utilizzano una sola risorsa.

La differenza tra i profili hom ed het è che in hom le risorse utilizzate sono omogenee, cioè, implementate con lo stesso software, invece, in het le risorse sono eterogenee, cioè, implementate con software diversi.

Le operazioni di ottimizzazione sono applicate solo sulle query per i profili hom ed het, e vengono salvate nella directory **hom/het-opt** quelle ottimizzate, invece, quelle ottimizzate con viste materializzate sono salvate nella directory **hom/het-opt-matv**.

All'interno di docker sono presenti i seguenti container:

- Il container **Sc1** rappresenta una base di dati centralizzata. Al suo interno sono contenute diverse entità legate da varie relazioni.
- Il container **Sc2** rappresenta una base di dati centralizzata. Al suo interno sono contenute le stesse informazioni presenti in **Sc1**, cambia lo schema, quindi, ci sono entità diverse e relazioni diverse. Ad esempio l'entità **Product** presente in Sc1 viene rappresentata con due entità dentro Sc2, che sono **Product1** e **Product2**.
- Il container **Source-smatv** contiene le viste materializzate usate per fare le ottimizzazioni.
- I container **source-s1**, **source-s2**, **source-s3**, **source-s4**, **source-s5** e **source-s2p** contengono dei frammenti della base di dati Sc1. Infatti, al loro interno sono presenti solo due o tre entità legate da una o due relazioni.
- I container **teiid-hom** e **teiid-het** contengono al loro interno il software **Teiid**. Questo software permette di eseguire delle interrogazioni su database virtuali e di accedere ai dati in modo federato. Un database virtuale è un artefatto che combina una o più sorgenti di dati per garantire una facile integrazione dei dati. L'integrazione è incapsulata attraverso viste e procedure che Teiid elaborerà in modo ottimizzato rispetto alle fonti.
- I container **ontop-sc1**, **ontop-sc2**, **ontop-hom** e **ontop-het** contengono l'applicazione **ontop**, che rappresenta lo strato obda (ontology based data access) utilizzato dall'architettura.

Per eliminare i container si può utilizzare il seguente comando:

```
1 docker-compose down -v --remove-orphans --rmi all
```

3.1.2 Risoluzione dei problemi di docker

Per istanziare tutti i container si è dovuto risolvere un problema nel file **docker-compose**, in quanto alcuni path per utilizzare dei file erano sbagliati. Si è proceduto alla risoluzione del problema aggiornando i path.

Una volta istanziati tutti i container si è osservato un altro problema, in quanto tutti i container **ontop** si riavviavano continuamente.

Studiando il file **docker-compose** sono emersi una serie di errori. Si è osservato che questi container utilizzavano un file chiamato **entrypoint**, che non era presente nella directory di lavoro. Tale file veniva utilizzato in maniera analoga da altri container, quindi, per risolvere il problema si è proceduto a copiare il file mancante all'interno della cartella utilizzata dai container ontop.

Un altro problema che si è dovuto risolvere è stato quello relativo ai path di alcuni file dei container ontop dentro il file **docker-compose**, infatti, tali path erano sbagliati e facevano riferimento a delle directory che non esistevano.

Aggiornando i path si è riusciti a risolvere il problema dei container ontop. Infatti, dopo queste operazioni si osserva che i container smettevano di riavviarsi continuamente.

Esecuzione di una singola query

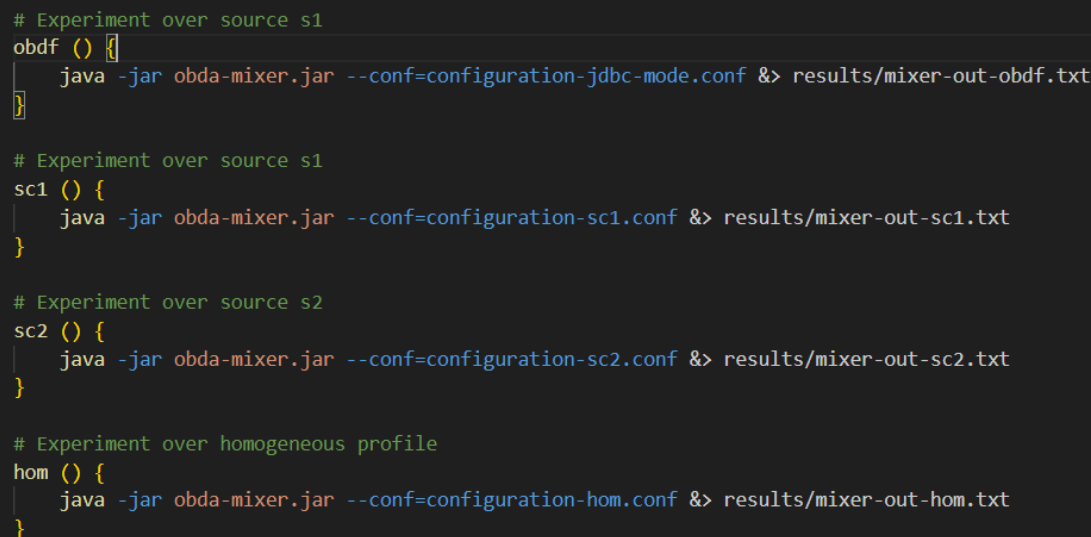
Una volta avviata l'architettura si proceduti a eseguire una singola query, per verificare che tutti i container funzionassero correttamente. Si proceduti con l'esecuzione di una query e l'esito è stato negativo, erano presenti dei problemi che non permettevano alla query di essere eseguita correttamente.

Per eseguire una query bisogna utilizzare il file **tester.sh** utilizzando il seguente comando:

```
1 sudo bash -c "source tester.sh; sc1"
```

Tale comando deve essere eseguito su terminale Ubuntu, utilizzando come directory di lavoro la stessa dov'è presente il file **tester.sh**. All'interno del comando, dopo il simbolo ';' viene passato come argomento il nome della funzione da mandare in esecuzione, infatti, al suo interno sono presenti tante funzioni quante sono le directory contenenti le query sql.

Di seguito riportiamo un'immagine che rappresenta una parte del codice presente nel file.



```
# Experiment over source s1
obdf () {
    java -jar obda-mixer.jar --conf=configuration-jdbc-mode.conf &> results/mixer-out-obdf.txt
}

# Experiment over source s1
sc1 () {
    java -jar obda-mixer.jar --conf=configuration-sc1.conf &> results/mixer-out-sc1.txt
}

# Experiment over source s2
sc2 () {
    java -jar obda-mixer.jar --conf=configuration-sc2.conf &> results/mixer-out-sc2.txt
}

# Experiment over homogeneous profile
hom () {
    java -jar obda-mixer.jar --conf=configuration-hom.conf &> results/mixer-out-hom.txt
}
```

Figura 3.3: tester.sh

Una volta lanciato il comando viene utilizzato uno specifico file di configurazione, chiamato **configuration-sc1.conf**, il nome di tale file varia a seconda della funziona che viene chiamata perchè lavora su una directory diversa. All'interno di questo file sono presenti molte informazioni, tra cui nome utente e password con cui collegarsi alla sorgente, gli indirizzi, la directory all'interno della quale trovare tutte le query sql da

eseguire, e altre informazioni.

Una volta lanciato il comando vengono eseguite tutte le query presenti nella directory specificata dal file di configurazione.

Un primo problema che si è dovuto risolvere era dovuto al fatto che all'interno dei file di configurazione, alcuni indirizzi delle risorse erano sbagliati, e questo causava errori durante l'esecuzione delle query. Sistemati gli indirizzi ci si è accorti che era presente una seconda problematica. All'interno della cartella mancavano dei file di configurazione utilizzati da alcune funzioni, che erano **configuration-sc2.conf** e **configuration-hom-opt-matv.conf**.

Per ricostruire tali file sono state utilizzate le informazioni presenti negli altri file di configurazioni che non cambiavano mai, mentre, per gli indirizzi delle sorgenti sono stati utilizzati quelli presenti nel file **docker-compose**.

Sistemati questi problemi l'esecuzione della funzione termina con esito positivo e le query generano dei risultati.

3.2 Esecuzione delle query

All'interno della repository dell'articolo sono presenti le query scritte nel linguaggio sparql dagli autori dell'articolo e le corrispondenti query già tradotte e ottimizzate in linguaggio sql.

Come prima cosa si è proceduto a verificare che le interrogazioni in sparql venissero tradotte correttamente nelle corrispondenti interrogazioni in sql. Questa verifica è stata fatta per i profili sc1, sc2, hom ed het.

Si è proceduto a utilizzare il terminale di Docker-desktop, che permette di comunicare con i container, per inserire all'interno del terminale dei container ontop le query da convertire. Il container produce il risultato che riportiamo nella seguente figura. La query in linguaggio sql è evidenziata.

```
2023-09-05 19:22:26 {"timestamp": "2023-09-05T17:22:26.565Z", "message": "query:all", "application": "heterogeneous",
  "payload": {"queryId": "58a7c34d-a9f7-456a-bd8f-f8c58f465416", "classesUsedInQuery": [], "propertiesUsedInQuery": [{"h
    http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/productId", "http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/
    v01/vocabulary/productFeature"], "tables": [{"ss1"."product1", "ss5"."product2", "ss1"."productfeatur
    eproduct1", "ss5"."productfeatureproduct2"], "reformulationDuration": 160, "reformulationCacheHit": false, "ht
    tpHeaders": {"referer": "http://localhost:13000/"}}, {"sparqlQuery": "PREFIX bsbm-inst: <http://www4.wiwiiss.fu-berlin.
    de/bizer/bsbm/v01/instances/>\nPREFIX bsbm: <http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>\nPREFIX
    rdfs: <http://www.w3.org/2000/01/rdf-schema#>\nPREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n\nSEL
    ECT ?product (count(?feature) as ?featureNum)\n\nWHERE {\n  ?product bsbm:productId ?id .\n  FILTER (?id <= 1000
    )\n  ?product bsbm:productFeature ?feature .\n  }\nGROUP BY ?product", "reformulatedQuery": "ans1(product,featureNu
    m)\n\nCONSTRUCT [product/RDF(http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFr
    omProducer/Product/)(INTEGERToString(nr2m23)),IRI, featureNum/RDF(BIGINTToString(v1,xsd:integer))]\n\nNATIVE [
    nr2m23, v1]\n\nSELECT v12.\"nr2m23\" AS \"nr2m23\", COUNT(*) AS \"v1\" \nFROM (SELECT DISTINCT v5.\"nr2m23\" AS \"n
    r2m23\", v10.\"productfeature2m2\" AS \"productfeature2m2\" \nFROM (SELECT v1.\"nr\" AS \"nr2m23\" \nFROM \"ss1\".
    \"product1\" v1 \nWHERE (v1.\"nr\" <= 1000) \nUNION ALL \nSELECT v3.\"nr\" AS \"nr2m23\" \nFROM \"ss5\". \"product2\"
    v3 \nWHERE (v3.\"nr\" <= 1000) \n) v5, (SELECT v6.\"product\" AS \"nr2m0\", v6.\"productfeature\" AS \"productfe
    ature2m2\" \nFROM \"ss1\". \"productfeatureproduct1\" v6 \nWHERE (v6.\"product\" <= 1000) \nUNION ALL \nSELECT v8.\"
    product\" AS \"nr2m0\", v8.\"productfeature\" AS \"productfeature2m2\" \nFROM \"ss5\". \"productfeatureproduct2\"
    v8 \nWHERE (v8.\"product\" <= 1000) \n) v10 \nWHERE v5.\"nr2m23\" = v10.\"nr2m0\" \n) v12 \nGROUP BY v12.\"nr2m23\" \n
    \n\", "executionBeforeUnblockingDuration": 1126, "executionAndFetchingDuration": 1254, "totalDuration": 1414, "resultCou
    nt": 1000}}
```

Figura 3.4: Traduzione della query 00 per il profilo het

Le interrogazioni prodotte da ontop partono dalla parola **Native**. Per poter essere utilizzate devono essere rimossi tutti i caratteri \ e bisogna sostituire i caratteri \n

```

2023-09-05 19:14:25 {"@timestamp":"2023-09-05T17:14:25.037Z","message":"query:all","application":"sc1","payload":
{"queryId":"5e6ad3c6-5383-43fc-8c5d-bf572c12a321","classesUsedInQuery":[],"propertiesUsedInQuery":["http://www4
.wiwiiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/productId","http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/v01/vocabu
lary/productFeature"],"tables":["productfeatureproduct"],"reformulationDuration":171,"reformulationCacheHit"
:false,"httpHeaders":{"referer":"http://localhost:13002/"},"sparqlQuery":"PREFIX bsbm-inst: <http://www4.wiwiiss.
fu-berlin.de/bizer/bsbm/v01/instances/>\nPREFIX bsbm: <http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/v01/vocabulary
/>\nPREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\nPREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-n
s#>\n\nSELECT ?product (count(?feature) as ?featureNum)\nWHERE {\n  ?product bsbm:productId ?id .\n  FILTER (?
id <= 1000 )\n  ?product bsbm:productFeature ?feature .\n }\nGROUP BY ?product\n","reformulatedQuery":"ansi(produ
ct,featureNum)\n\nCONSTRUCT [product, featureNum] [product/RDF(http://www4.wiwiiss.fu-berlin.de/bizer/bsbm/v01/inst
ances/dataFromProducer/Product[] (INTEGERToTEXT(nr1m23)),IRI), featureNum/RDF(BIGINTToTEXT(v1),xsd:integer)]\n
NATIVE [nr1m23, v1]\n\nSELECT v1.\"product\" AS \"nr1m23\", COUNT(*) AS \"v1\"\nFROM \"productfeatureproduct\" v1\
nWHERE (v1.\"product\" <= 1000)\nGROUP BY v1.\"product\"\n\n","executionBeforeUnblockingDuration":9,"executionAn
dFetchingDuration":167,"totalDuration":338,"resultCount":1000}}

```

Figura 3.5: Traduzione della query 00 per il profilo sc1

con la newline.

Una volta pulito il testo quello che si ottiene è l'interrogazione che deve essere salvata come file .sql. Al termine dell'operazione si è verificato che tutte le query, scritte in sparql, presenti nel progetto venivano tradotte correttamente nelle corrispondenti query sql.

3.2.1 Utilizzo degli script per eseguire le query

L'esecuzione di una funzione del file tester.sh genera due risultati. Il primo file si chiama **mixer-stats-** e dopo il trattino viene inserito il nome della funzione che è stata eseguita.

Al suo interno vengono riportati eventuali errori, tempo di esecuzione delle query e numero di risultati ottenuti.

Di seguito carichiamo un esempio di questo file.

Il secondo file si chiama **mixer-out-** e dopo il trattino viene inserito il nome della funzione che è stata eseguita. Al suo interno vengono riportati possibili errori che si verificano durante l'esecuzione delle query e vengono riportate le query che alla fine vengono eseguite sulle risorse. Infatti, nelle query che vengono mandate in esecuzione sono presenti dei placeholder, come ad esempio **\$1:product.nr:percent**, esso viene sostituito durante l'esecuzione della query dal valore corrispondente.

Di seguito riportiamo un esempio di file mixer-out.

Dopo che si è eseguita una funzione, è necessario eseguire il programma java **ResultToCSV.java** usando il comando:

```
1 java ResultToCSV
```

Tale programma calcola il tempo di esecuzione medio, per ogni query e per ogni profilo di lavoro, e lo inserisce all'interno di un file .csv, che sarà rinominato con la dimensione del dataset utilizzato.

Si è proceduto a realizzare dei nuovi file che rendessero più automatica l'esecuzione delle funzioni e del file ResultToCSV. Sono stati realizzati due nuovi file che sono **all-test.sh** e **ResultToCSVfull**.

Il primo file permette di chiamare in maniera sequenziale tutte le funzioni del file te-

```
[GLOBAL]
[main] [load-time] = 1
[thread#0]
[run#5] [num_results#05.sql] = 507
[run#5] [num_results#11.sql] = 12
[run#5] [num_results#00.sql] = 93
[run#5] [num_results#03.sql] = 70
[run#5] [num_results#02.sql] = 2009
[run#5] [num_results#07.sql] = 2
[run#5] [num_results#09.sql] = 32
[run#5] [num_results#12.sql] = 105
[run#5] [num_results#01.sql] = 16
[run#5] [num_results#06.sql] = 1
[run#5] [num_results#04.sql] = 17
[run#5] [num_results#08.sql] = 6
[run#5] [num_results#10.sql] = 6
[run#5] [rewriting_time#04.sql] = 0
[run#5] [resultset_traversal_time#11.sql] = 3310
[run#5] [mix_time#5] = 91201
[run#5] [execution_time#07.sql] = 9571
[run#5] [unfolding_time#10.sql] = 0
[run#5] [resultset_traversal_time#02.sql] = 19
[run#5] [execution_time#08.sql] = 4912
[run#5] [unfolding_time#03.sql] = 0
[run#5] [unfolding_time#12.sql] = 0
[run#5] [rewriting_time#05.sql] = 0
[run#5] [resultset_traversal_time#10.sql] = 0
[run#5] [execution_time#06.sql] = 6823
```

Figura 3.6: Esempio di file mixer-stat-sc1

```
17:19:38.152 [Thread-0] INFO it.unibz.inf.mixer_main.execution.MixerThread - Doing query: 01.sql
17:19:38.159 [Thread-0] DEBUG it.unibz.inf.mixer_main.execution.MixerThread - [mixer-debug] Call fillPlaceholders
17:19:38.163 [Thread-0] DEBUG it.unibz.inf.mixer_main.execution.MixerThread - --[48]
--[50]
--[992]

SELECT v1."label" AS "label8m46", v1."nr" AS "product1m2"
FROM "product" v1, "productfeatureproduct" v2, "productfeatureproduct" v3
WHERE ((v1."propertynum1" <= [992]) AND v1."label" IS NOT NULL AND v1."propertynum1" IS NOT NULL AND
v1."nr" = v2."product" AND v1."nr" = v3."product" AND
[48] = v2."productfeature" AND
[50] = v3."productfeature")
```

Figura 3.7: Esempio di file mixer-out-sc1 con sostituzione del placeholder

ster.sh, in maniera tale da non doverle chiamare singolarmente; il secondo file, invece, permette di eseguire le operazioni del file ResultToCSV direttamente su tutti i dataset, altrimenti, ogni volta bisognava cambiare il path, ricompilare il nuovo file ed eseguirlo.

Problematiche di esecuzione sui diversi dataset

I risultati riportati nell'articolo sono stati ottenuti su dataset di diverse dimensioni, che sono uno da 20000, uno da 200000 e uno da 2000000 di elementi.

L'esecuzione degli script sul dataset da 20000 elementi non ha generato problemi.

L'esecuzione degli script sul dataset da 200000 elementi ha generato dei problemi durante l'esecuzione, causati da una capacità di memoria centrale non sufficiente.

Riportiamo di seguito un esempio di questo errore.

3.3.1 Reverse engineering degli schemi

Per poter scrivere delle nuove query è necessario comprendere a pieno lo schema dei dati, quindi, si è proceduto con una fase di reverse engineering del codice sql con il quale sono state implementate le basi di dati sulle singole sorgenti. Leggendo il codice sql delle chiavi esterne si sono potute ricostruire le relazioni che legano tra loro le varie entità.

Riportiamo di seguito le immagini contenenti gli schemi e-r ottenuti durante questa fase, suddivisi per profilo di lavoro.

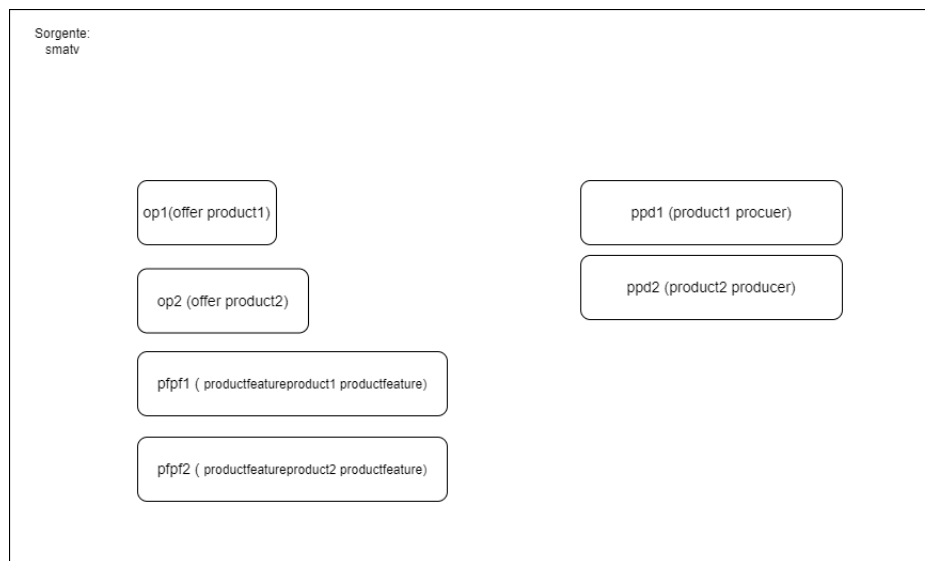


Figura 3.9: Schema e-r delle viste materializzate

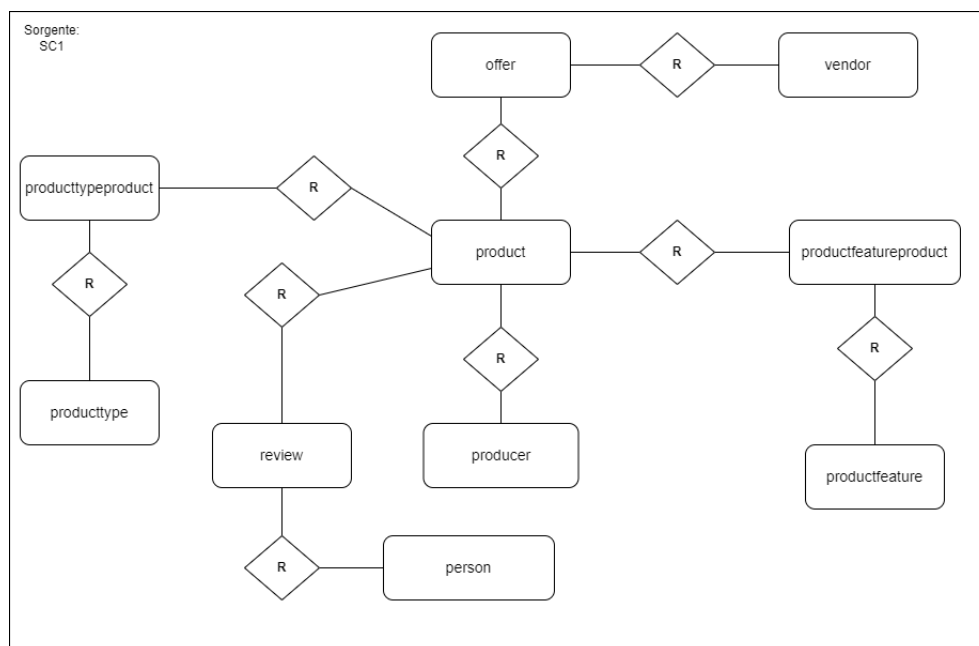


Figura 3.10: Schema e-r del profilo sc1

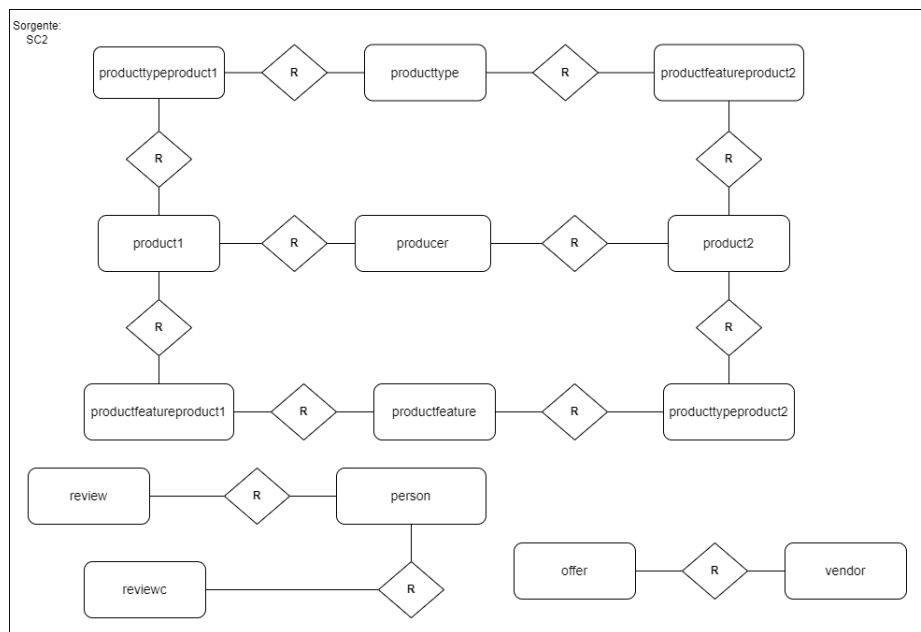


Figura 3.11: Schema e-r del profilo sc2

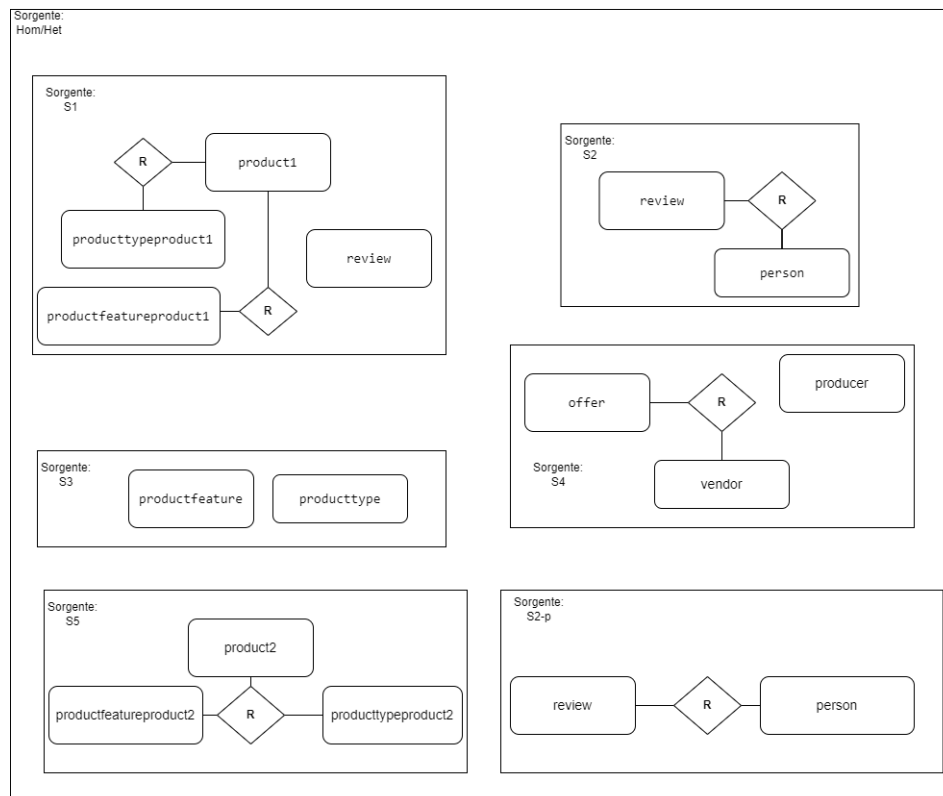


Figura 3.12: Schema e-r del profilo hom/het

3.3.2 Scrittura delle nuove interrogazioni

Durante questa fase del progetto si sarebbero dovute scrivere delle nuove query, per verificare se scrivendo interrogazioni diverse da quelle proposte si osservavano delle differenze nei tempi di esecuzione tra quelle ottimizzate, e ottimizzate con viste materializzate e quelle normali, senza ottimizzazione.

Purtroppo, gli autori dell'articolo non hanno rilasciato i container contenenti gli algoritmi di ottimizzazione. La componente di ottimizzazione non è disponibile in quanto non è funzionante. Gli autori dell'articolo hanno ottimizzato un insieme di query, applicando 'a mano' i loro algoritmi, e la sperimentazione riguarda il mostrare che le ottimizzazioni danno effettivamente risultati migliori in fase di valutazione delle query. Tali informazioni non erano presenti all'interno dell'articolo, ma sono state comunicate, successivamente, dagli autori dell'articolo.

Di conseguenza si potevano tradurre query in sparql solo per i profili sc1, sc2, hom ed het, ma non è possibile tradurre le query per i profili hom/het dove sono presenti le ottimizzazioni e le ottimizzazioni con viste materializzate.

3.4 Esecuzione delle query modificate

3.4.1 Modifica delle query già esistenti

Come ultima fase del progetto si è deciso di verificare se le ottimizzazioni fossero ancora valide dopo aver modificato alcune condizioni delle interrogazioni.

Per alcune query sono stati sostituiti i placeholder all'interno di alcune query con delle costanti dal valore più grande rispetto a quello originale. Modificando solo la condizione all'interno dell'istruzione **where** non andiamo a compromettere le ottimizzazioni fatte sulle query. Sono stati sostituiti i placeholder delle condizioni con simbolo di ' $=$ ', in questo si appesantisce la query andando a considerare un numero maggiore di elementi.

Per altre, invece, si è andati ad aggiungere delle nuove condizioni. In una si è proceduto a calcolare il resto della divisione per tre, tale condizione è vera solo se il resto è zero, quindi, quel valore è un multiplo di tre, e in un'altra interrogazione abbiamo aggiunto una condizione di **LIKE**, quindi, l'attributo come valore doveva contenere una certa sottostringa.

La procedura per l'esecuzione di queste interrogazioni è analoga a quella precedente.

3.4.2 Analisi dei nuovi risultati

Dai risultati ottenuti è possibile osservare come nelle due interrogazioni, dove erano presenti le condizioni aggiuntive nel **where**, ottimizzate con viste materializzate hanno prestazioni peggiori di quelle non ottimizzate, oppure, ottimizzate senza viste materializzate.

Questo è riconducibile al fatto che le informazioni presenti nelle viste materializzate

non sono utili per velocizzare l'esecuzione della query.

Tale comportamento emerge con il dataset da 200000 elementi. Per le altre interrogazioni si è osservato lo stesso risultato ottenuto precedentemente, cioè, le interrogazioni ottimizzate e ottimizzate su viste materializzate hanno un tempo di esecuzione minore rispetto a quelle non ottimizzate.

Capitolo 4

Risultati

Inizialmente si riportano i risultati ottenuti dagli autori del paper.

Successivamente, si riportano i risultati ottenuti durante questo progetto.

I risultati ottenuti confermano le conclusioni formulate dagli autori dell'articolo, infatti, dai risultati ottenuti durante questo progetto è possibile osservare come le query ottimizzate e ottimizzate con viste materializzate siano più veloci di quelle normali.

Infine, si riportano i risultati ottenuti con l'esecuzione delle nuove interrogazioni.

I risultati ottenuti mostrano come le query ottimizzate hanno dei tempi di esecuzione, che in generale, sono minori rispetto a quelle non ottimizzate, anche se lavorano con risultati di dimensioni maggiori.

Le query ottimizzate su viste materializzate e una condizione aggiuntiva nel where hanno, in alcuni casi, un tempo d'esecuzione maggiore rispetto alle altre.

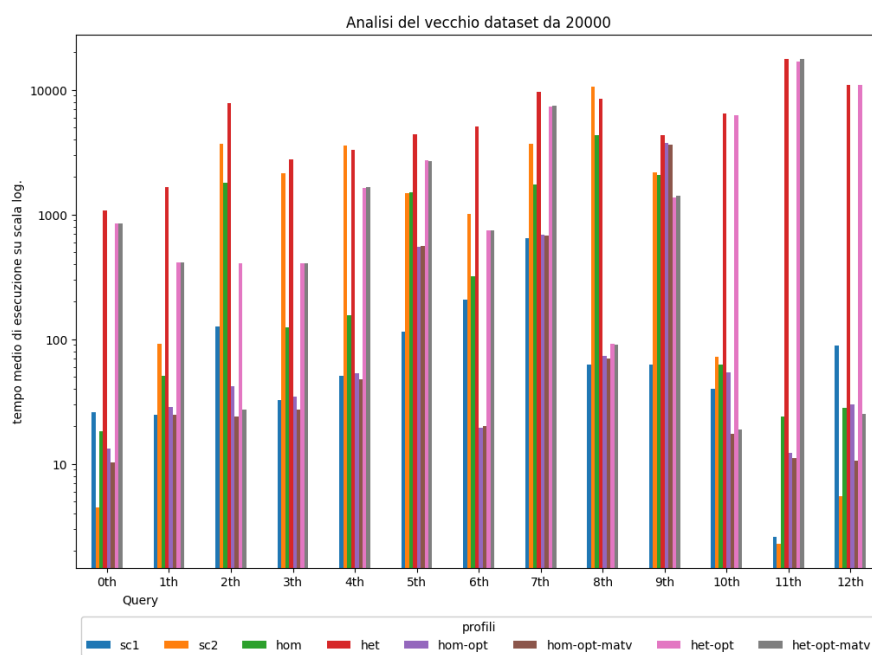


Figura 4.1: Risultati ottenuti dagli autori su dataset da 20000 elementi

query	sc1	sc2	hom	het	hom-opt	hom-opt-matv	het-opt	het-opt-matv
0th	26.1	4.5	18.2	1085.5	13.3	10.3	843.4	846.9
1th	24.7	91.7	51.2	1648.2	28.5	24.6	414.6	413.7
2th	126.8	3713.5	1791.4	7796.1	42.3	24.0	409.4	27.5
3th	32.6	2139.7	125.0	2759.1	34.6	27.4	404.0	405.8
4th	51.3	3602.8	156.8	3315.1	53.8	47.6	1646.1	1662.7
5th	114.3	1488.3	1518.0	4379.6	547.0	559.4	2730.3	2698.3
6th	209.1	1008.8	318.7	5074.8	19.5	20.0	742.1	741.2
7th	643.7	3680.7	1745.8	9654.1	694.1	679.0	7290.3	7473.5
8th	63.1	10608.6	4308.7	8473.6	73.3	70.5	91.3	91.2
9th	62.4	2193.4	2088.5	4344.0	3771.1	3643.1	1379.3	1409.5
10th	40.1	72.1	63.0	6424.5	54.3	17.5	6303.6	18.9
11th	2.6	2.3	24.1	17767.1	12.2	11.1	16793.1	17675.5
12th	89.3	5.5	28.0	10969.2	29.8	10.6	10989.0	25.3

Tabella 4.1: Tabella dei risultati su dataset da 20000 elementi ottenuti dagli autori

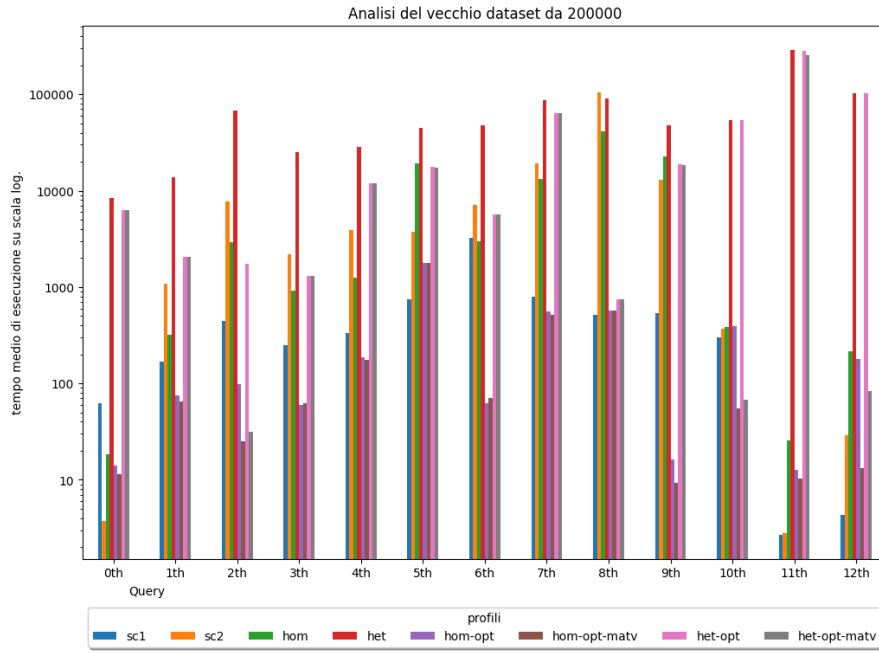


Figura 4.2: Risultati ottenuti dagli autori su dataset da 200000 elementi

query	sc1	sc2	hom	het	hom-opt	hom-opt-matv	het-opt	het-opt-matv
0th	62.9	3.7	18.3	8369.1	14.1	11.4	6224.6	6284.7
1th	167.6	1088.2	320.1	13878.7	75.1	65.0	2058.0	2044.8
2th	443.9	7788.9	2942.2	68289.9	98.0	24.9	1752.1	31.2
3th	247.3	2179.5	916.1	24986.2	59.9	62.1	1307.3	1303.3
4th	336.0	3937.4	1243.2	28563.7	188.1	176.2	12007.8	11841.6
5th	740.3	3747.2	19062.2	44603.3	1768.6	1784.5	17839.2	17426.3
6th	3215.3	7110.7	2978.9	47299.1	62.7	71.2	5659.9	5654.8
7th	799.4	19099.3	13329.2	86729.3	557.6	519.0	63521.3	63541.6
8th	516.2	103837.6	40896.9	90934.7	569.2	567.5	746.9	748.8
9th	539.6	12967.3	22721.9	48039.7	16.3	9.3	18796.9	18515.7
10th	297.5	371.9	384.9	53870.0	396.7	55.3	54287.0	67.9
11th	2.7	2.8	25.7	290374.8	12.7	10.2	283411.9	252541.9
12th	4.3	28.8	214.6	103333.4	180.8	13.3	102843.6	83.1

Tabella 4.2: Tabella dei risultati su dataset da 200000 elementi ottenuti dagli autori

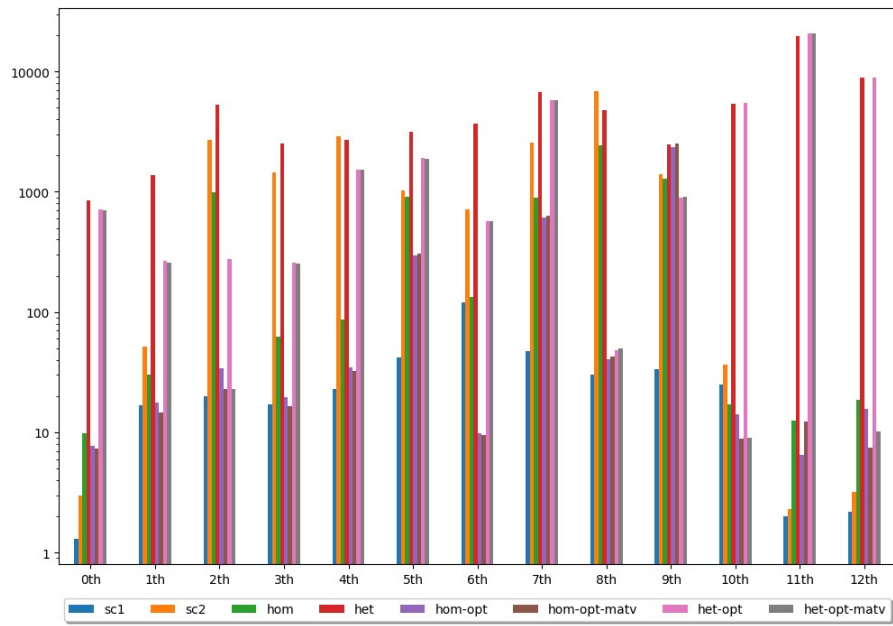


Figura 4.3: Risultati ottenuti durante il progetto su dataset da 20000 elementi

query	sc1	sc2	hom	het	hom-opt	hom-opt-matv	het-opt	het-opt-matv
0th	1.2	2.9	10.1	818.6	7.4	5.8	693.6	696.2
1th	16.4	52.8	32.3	1346.8	17.7	14.0	251.2	262.6
2th	20.1	2845.2	1004.6	4885.9	32.2	21.2	267.0	21.3
3th	16.9	1389.5	54.8	2453.8	19.9	15.5	265.7	266.1
4th	22.6	2483.4	80.7	2594.9	32.4	30.9	1461.7	1445.9
5th	42.1	949.9	926.9	3185.1	293.4	294.1	1852.9	1868.7
6th	122.9	688.8	125.7	3698.5	9.2	8.3	534.5	529.5
7th	49.4	2470.0	885.3	6765.9	615.1	606.1	5578.7	5577.3
8th	32.3	7592.7	2392.5	4860.0	41.6	38.8	46.0	46.2
9th	38.9	1349.1	1315.8	2429.5	2356.0	2343.5	908.3	902.6
10th	26.3	33.8	16.5	5318.3	13.1	7.9	5406.3	8.2
11th	1.8	1.7	12.2	16418.7	6.1	10.6	18742.2	19190.4
12th	2.2	3.1	18.3	8399.7	15.0	6.3	8592.4	9.4

Tabella 4.3: Tabella dei risultati su dataset da 20000 elementi ottenuti durante il progetto

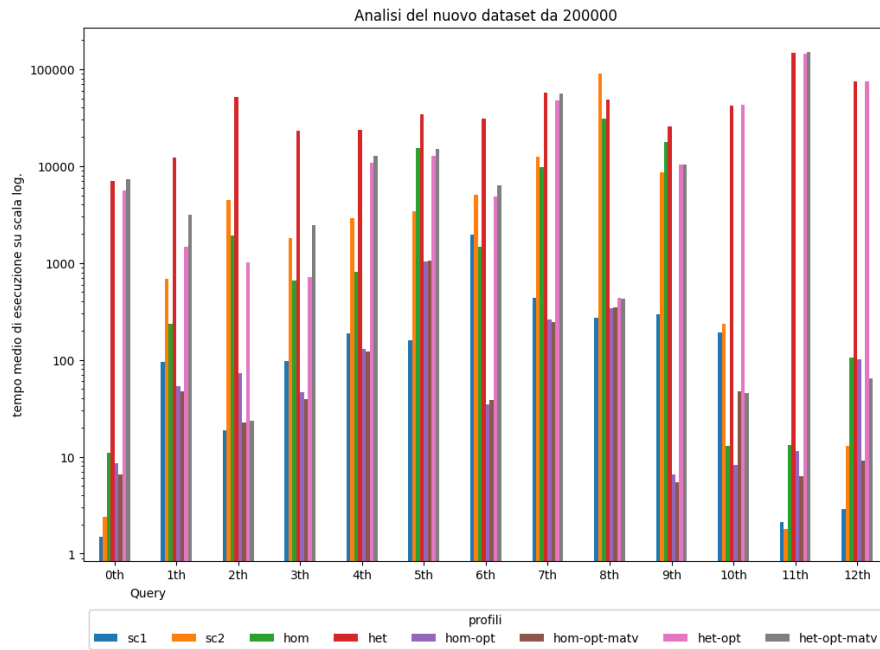


Figura 4.4: Risultati ottenuti durante il progetto su dataset da 200000 elementi

query	sc1	sc2	hom	het	hom-opt	hom-opt-matv	het-opt	het-opt-matv
0th	1.5	2.4	10.9	6996.6	8.5	6.5	5659.0	7385.1
1th	95.6	689.5	237.0	12310.7	53.2	47.6	1462.4	3175.1
2th	18.6	4512.1	1916.6	51735.1	72.2	22.4	1019.2	23.3
3th	98.2	1789.3	653.6	23306.9	46.5	39.1	721.6	2458.5
4th	187.5	2875.5	806.9	23781.2	129.8	122.4	10925.6	12639.8
5th	158.5	3447.7	15404.9	34494.0	1036.9	1048.0	12636.6	14897.9
6th	1948.5	5077.5	1477.1	31028.0	35.1	38.4	4875.4	6377.5
7th	437.5	12503.9	9837.8	57292.8	258.6	247.8	47751.6	55780.6
8th	271.0	90338.3	30859.0	48128.9	340.7	348.4	435.0	426.4
9th	294.4	8634.1	17732.7	25839.4	6.5	5.5	10402.9	10423.3
10th	193.0	237.6	13.0	42432.2	8.3	47.8	43060.8	45.3
11th	2.1	1.8	13.3	148118.3	11.5	6.3	145197.9	151614.1
12th	2.9	13.0	105.1	74422.6	101.9	9.1	74818.9	64.7

Tabella 4.4: Tabella dei risultati su dataset da 200000 elementi ottenuti durante il progetto

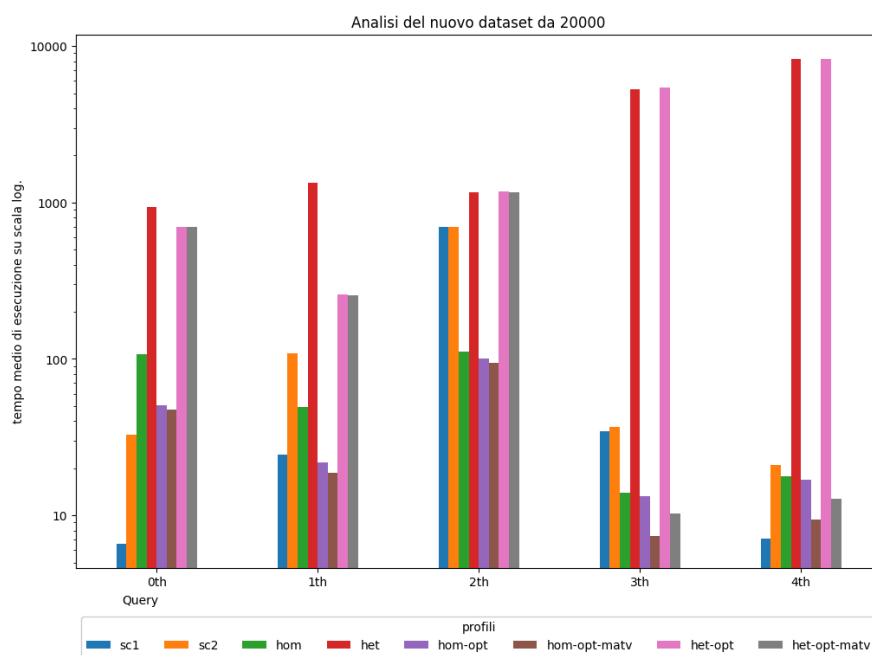


Figura 4.5: Nuove query su dataset da 20000 elementi

query	sc1	sc2	hom	het	hom-opt	hom-opt-matv	het-opt	het-opt-matv	
0th	6.6	32.6	107.3	934.7	50.8	47.6	699.6	695.5	
1th	24.4	109.1	49.0	1334.1	21.9	18.7	257.3	256.5	
2th	697.6	694.8	111.9	1167.2	101.2	94.7	1175.1	1168.1	
3th	34.5	36.7	14.0	5310.6	13.3	7.4	5408.2	10.3	
4th	7.1	20.9	17.7	8291.9	16.8	9.4	8311.5	12.7	

Tabella 4.5: Tabella dei risultati su dataset da 20000 elementi ottenuti con le nuove query

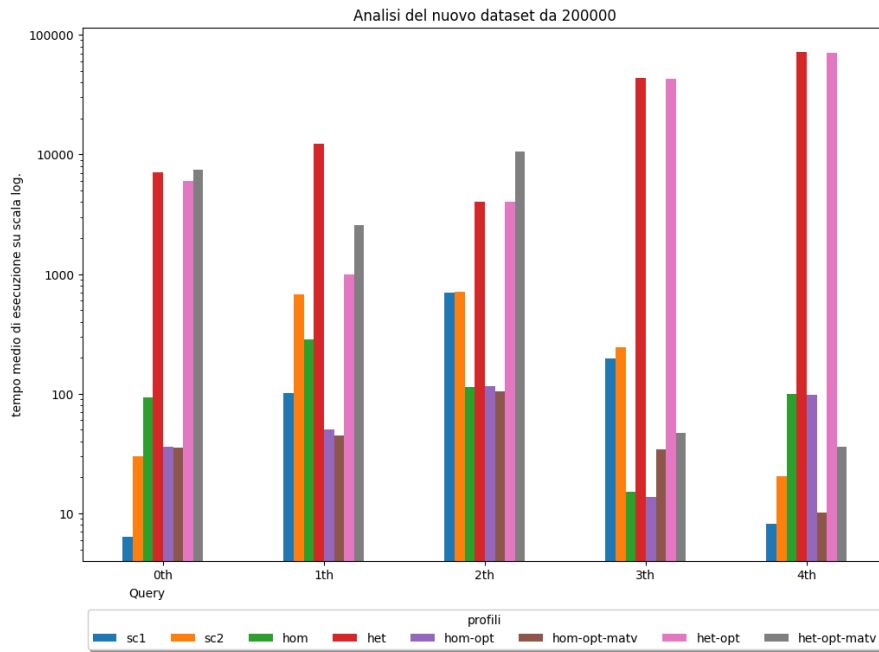


Figura 4.6: Nuove query su dataset da 200000 elementi

query	sc1	sc2	hom	het	hom-opt	hom-opt-matv	het-opt	het-opt-matv
0th	6.4	29.8	93.3	7057.0	36.0	35.5	5953.2	7422.0
1th	100.6	676.2	284.8	12187.6	50.4	45.0	989.2	2554.9
2th	702.6	715.1	114.6	4023.1	115.8	105.1	4040.1	10511.7
3th	198.0	245.8	15.2	43588.0	13.8	34.4	42886.9	46.8
4th	8.2	20.4	99.9	72174.3	98.1	10.2	70376.3	35.9

Tabella 4.6: Tabella dei risultati su dataset da 200000 elementi ottenuti con le nuove query

Capitolo 5

Conclusioni e Sviluppi futuri

5.1 Sviluppi futuri

Quando saranno disponibili le componenti per l'ottimizzazione, si potrà riprendere la fase di questo progetto relativa alla scrittura di nuove query, in maniera tale da testare le capacità di ottimizzazione del sistema in maniera più approfondita e per comprendere al meglio pregi e difetti di questa architettura.

5.2 Conclusioni

In conclusione, i risultati che abbiamo ottenuto sono analoghi a quelli che hanno ottenuto gli autori dell'articolo, quindi, le ottimizzazioni implementate rendono la valutazione delle query effettivamente più veloce.

L'ottimizzazione viene effettuata nel primo livello dell'architettura, quello rappresentato da **obda**. Durante la fase di traduzione applica gli hint per ottimizzare, con e senza viste materializzate, la query. Purtroppo, non si è potuto testare ulteriormente tale meccanismo.

La traduzione delle query sparql è chiara ed è applicabile per ottenere delle query che possono essere mandate in esecuzione sugli altri container.

Bibliografia

- [1] *Python 3.11 documentation*. URL: <https://docs.python.org/3/>.
- [2] *Java documentation*. URL: <https://docs.oracle.com/en/java/>.
- [3] *Windows subsystem Linux*. URL: <https://techcommunity.microsoft.com/t5/windows-11/how-to-install-the-linux-windows-subsystem-in-windows-11/m-p/2701207>.
- [4] *Download docker desktop*. URL: <https://www.docker.com/products/docker-desktop/>.

Elenco delle figure

1.1	framework ODBF e procedura di risposta alle query.	2
3.1	Flowchart	5
3.2	File di configurazione	6
3.3	tester.sh	8
3.4	Traduzione della query 00 per il profilo het	9
3.5	Traduzione della query 00 per il profilo sc1	10
3.6	Esempio di file mixer-stat-sc1	11
3.7	Esempio di file mixer-out-sc1 con sostituzione del placeholder	11
3.8	Esempio di errore out of memory	12
3.9	Schema e-r delle viste materializzate	14
3.10	Schema e-r del profilo sc1	14
3.11	Schema e-r del profilo sc2	15
3.12	Schema e-r del profilo hom/het	15
4.1	Risultati ottenuti dagli autori su dataset da 20000 elementi	19
4.2	Risultati ottenuti dagli autori su dataset da 200000 elementi	20
4.3	Risultati ottenuti durante il progetto su dataset da 20000 elementi	21
4.4	Risultati ottenuti durante il progetto su dataset da 200000 elementi	22
4.5	Nuove query su dataset da 20000 elementi	23
4.6	Nuove query su dataset da 200000 elementi	24

Elenco delle tabelle

4.1	Tabella dei risultati su dataset da 20000 elementi ottenuti dagli autori .	19
4.2	Tabella dei risultati su dataset da 200000 elementi ottenuti dagli autori	20
4.3	Tabella dei risultati su dataset da 20000 elementi ottenuti durante il progetto	21
4.4	Tabella dei risultati su dataset da 200000 elementi ottenuti durante il progetto	22
4.5	Tabella dei risultati su dataset da 20000 elementi ottenuti con le nuove query	23
4.6	Tabella dei risultati su dataset da 200000 elementi ottenuti con le nuove query	24