Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea Specialistica in Ingegneria Informatica

Il componente Query Manager del sistema MOMIS: testing ed analisi delle performance

Relatore:
Chiar.mo Prof. Sonia Bergamaschi

Candidato:
Entela Kazazi

Correlatore:
Ing. Mirko Orsini

Anno Accademico 2008-2009

Ai miei genitori

Parole Chiave:

MOMIS

Query Manager

HSQLDB

JUnit

Indice

Indice delle Tabelle

Indice delle Figure

Introduzione

Nel corso degli anni è diventata sempre più rilevante la necessità di accedere ad informazioni distribuite e contestualmente anche il problema dell'integrazione di informazioni provenienti da sorgenti eterogenee. Le imprese hanno a disposizione moltissime fonti di informazione, che possono contenere dati correlati tra loro ma spesso ridondanti, eterogenei e non sempre consistenti. L'esigenza è quella di poter accedere in modo semplice a tutte le informazioni aziendali distribuite, oppure poter costruire applicazioni che utilizzino in tempo reale tali informazioni. Il diffondersi delle reti di calcolatori e particolarmente del Web ha accelerato la necessità di condivisione e reperimento delle informazioni provenienti da diverse risorse. Il problema dell'integrazione dell'informazione [7,9,10,32] (data integration o information integration) consiste nell'integrare i dati correlati, residenti nelle diversi sorgenti e fornire all'utente una vista integrata di tali dati. I problemi che devono essere affrontati in questo ambito sono principalmente dovuti ad eterogeneità strutturali e applicative (basta pensare alle differenti piattaforme hardware, DBMS, modelli dei dati …), nonché dalla mancanza di una ontologia comune, che porta a differenze semantiche tra le fonti di informazione. A loro volta le differenze semantiche possono dare origine a diversi tipi di conflitti, che vanno dalle semplici incongruenze nell'uso dei nomi (sinonimi: quando nomi differenti sono utilizzati in sorgenti diverse per identificare gli stessi concetti) a conflitti strutturali (quando modelli diversi sono utilizzati per rappresentare le stesse informazioni).

Differenti approcci sono stati proposti per garantire l'integrazione dei dati provenienti da sorgenti eterogenee. I tre approcci più comunemente implementati per l'integrazione dei dati, descritti in modo dettagliato in [6] sono:

- *Federated databases*: Diversi database indipendenti che condividono l'informazione. In questa architettura viene implementato un framework che consente ai database di comportarsi come in una federazione. Ogni sorgente può estendere il suo schema con sottoschemi (e dati) provenienti dagli altri database della federazione.
- *Warehousing*: Copie dei dati provenienti dalle diverse sorgenti vengono memorizzati in un singolo database, chiamato (data) warehouse; i dati prima di essere memorizzati sono sottoposti ad un processo di trasformazione, es. i dati

possono essere filtrati, aggregati …. L'utente può formulare delle query sul data warehouse come se stesse interrogando un unico database relazionale, il risultato conterrà i dati che sono contenuti nel data (warehouse) e non nelle sorgenti. Il data warehouse viene allineato con le sorgenti sottostanti in modo periodico. Le modifiche vengono effettuate sulle sorgenti e periodicamente avviene l'aggiornamento dei dati contenuti nel data warehouse, per allineare questi ultimi con i dati contenuti nelle sorgenti.

- *Mediator systems*: In un sistema a Mediatore l'obiettivo è quello di ottenere una vista virtuale integrata, a sola lettura, dei dati memorizzati nelle diverse sorgenti. Quindi è necessario implementare un componente software, il Mediatore, in grado di costruire uno schema integrato e consentire agli utenti di effettuare delle query e ricevere una risposta unica. Le query vengono tradotte in un insieme di sottoquery che verranno eseguite direttamente sulle sorgenti, i risultati vengono fusi insieme e l'utente riceve un'unica risposta unificata. Quindi, diversamente dall'approccio warehousing, vengono interrogate le sorgenti originarie e lo schema che si ottiene dall'integrazione è uno schema virtuale, cioè non viene materializzato.

Nel seguente lavoro di tesi verrà trattato il sistema a Mediatore *MOMIS* [8,28] *(Mediator EnvirOment for Multiple Information Sources)*, un framework che ha come obiettivo l'estrazione e l'integrazione intelligente delle informazioni provenienti da sorgenti dati strutturate e semi strutturate.

Il sistema *MOMIS* è un sistema a Mediatore, basato sull'architettura wrapper/mediator [1]; il sistema genera una *Global Virtual View* (*GVV*) o *Global Schema* (*GS*) degli schemi delle sorgenti locali da integrare.

Il *GS è* una concettualizzazione (ontologia) che descrive un insieme di sorgenti dati distribuite ed eterogenee, l'utente può formulare delle query sul *GS* e ricevere un'unica risposta ottenuta tramite la fusione dei risultati provenienti dalle sorgenti locali, in modo trasparente rispetto alle sorgenti coinvolte.

Uno degli obiettivi principali è consentire all'utente di formulare query sullo schema globale e il modulo che svolge questa funzione, cioè che si occupa del query processing nel sistema *MOMIS* è il *Query Manager*. Il *Query Manager*, utilizzando tecniche di unfolding, genera la traduzione della query nelle corrispondenti

sottoquery da spedire alle singole sorgenti, elabora i risultati parziali e calcola una risposta unificata da presentare all'utente. Per la fusione dei risultati parziali viene utilizzato un DBMS *(DataBase Management System)* di supporto, in un database di quest'ultimo vengono create delle tabelle globali e locali temporanee nelle quali verranno inseriti i dati provenienti dalle sorgenti, successivamente elaborati questi dati costituiranno la risposta che verrà restituita all'utente. Nella prima implementazione del sistema *MOMIS* è stato scelto come DBMS di supporto *Microsoft SQL Server*, un DBMS relazionale, ad alte prestazioni, prodotto da Microsoft ed utilizzabile solo in una piattaforma Windows.

Essendo *MOMIS* un progetto open source nasce la necessità di utilizzare come DBMS di appoggio non una soluzione proprietaria, ma una soluzione *Open Source*. La scelta è caduta su un RDBMS open source, completamente scritto in Java, *HSQLDB (Hyper Structured Query Language DataBase)*, che verrà descritto in modo dettagliato nel capitolo 3.

L'obiettivo di questa tesi è il testing delle funzionalità del componente Query Manager e l'analisi delle performance e sarà cosi strutturata:

- **Capitolo 1: Il Sistema MOMIS.** In questo capitolo verrà presentata l'architettura del sistema MOMIS ed il processo di generazione del Global Schema (GS).
- **Capitolo 2: Il Query Management nel sistema Momis.** In questo capitolo verrà descritto il componente Query Manager, l'architettura di quest'ultimo e il query processing nel sistema MOMIS.
- **Capitolo 3: HSQLDB e testing correttezza.** In questo capitolo verrà descritto il nuovo DBMS di supporto del Query Manager, HSQLDB e utilizzando un data set di riferimento verranno eseguiti dei test per valutare la correttezza di esecuzione delle query.
- **Capitolo 4: Analisi e testing del Query Unfolding nel sistema MOMIS.** In questo capitolo verrà analizzato il Query Unfolding nel sistema MOMIS e utilizzando un data set di riferimento, verranno eseguiti dei test per valutare la correttezza dell'unfolding delle query.
- **Capitolo 5: Il benchmark THALIA.** In questo capitolo verrà descritto il benchmark THALIA e come sono state affrontate le 12 query del benchmark dal sistema MOMIS.

- **Capitolo 6: SQL Server vs HSQLDB, confronto delle prestazioni.** In questo capitolo si confronteranno i tempi di risposta alle query per i due DBMS, utilizzati come DBMS di supporto al query processing.
- **Capitolo 7: Progettazione e implementazione di un framework di testing.** In questo capitolo verrà descritto un framework progettato ed implementato per il testing del Query Manager.

Capitolo 1

1 Il sistema MOMIS

1.1 Integrazione Intelligente dell'Informazione

Il problema della data integration consiste nell'integrare i dati correlati residenti nelle diverse sorgenti coinvolte nell'integrazione e fornire all'utente una vista integrata di tali dati [7,9,10,32]. L'architettura wrapper/mediator è stata proposta dall'ARPA *(Advanced Research Project Agency)* [1] nel 1992 come un architettura di riferimento per i sistemi di integrazione intelligente dell'informazione. Come viene citato in [2], l'integrazione delle informazioni (I^2) si distingue da quella dei dati e dei database, in quanto non cerca di collegare semplicemente alcune sorgenti, ma risultati opportunamente selezionati da esse. Lo scopo dell'integrazione dell'informazione è quindi quello di ottenere una selezione ragionata dei dati prelevati dalle varie sorgenti e produrre una fusione intelligente ed una seguente sintesi degli stessi. Proprio a questo scopo, è stato sviluppato dall'ARPA, un progetto di ricerca atto a fornire un'architettura di riferimento, che realizzi l'integrazione di risorse eterogenee in maniera automatica; il nome di questo progetto è appunto I^3 *(Intelligent Integration of Information)*. I risultati ottenuti in questo ambito sono molto importanti poiché danno una concreta indicazione su come costruire un sistema a mediatore che sia riusabile, e le cui parti non comportino costi eccessivi di sviluppo. L'ARPA ritiene che un ruolo importante, possa essere giocato dall'utilizzo dell'Intelligenza Artificiale che, essendo in grado di dedurre dagli schemi delle sorgenti informazioni utili, può essere considerata uno strumento prezioso ed in grado di fornire soluzioni flessibili e riusabili. Secondo il progetto I^3 è opportuno costruire architetture modulari, in grado di abbassare i costi di sviluppo e mantenimento, eseguite seguendo uno standard che ponga le basi dei servizi necessari all'integrazione.

Il paradigma impiegato nel progetto I^3 per la suddivisione dei servizi e delle risorse fra i vari moduli, si basa su due partizionamenti fondamentali:

- Il partizio*namento orizzontale* che fornisce le seguenti sezioni: database, sorgenti, basi di conoscenza intermedie, ed applicazioni utente.
- Il *partizionamento verticale* che distingue i domini in cui raggruppare le sorgenti.

I domini non sono strettamente interconnessi tra loro all'interno di un certo livello, ma si scambiano dati e informazioni. Per facilitare la flessibilità e migliorare le prestazioni del sistema, la fase importantissima della combinazione delle informazioni avviene a livello utente. Nel seguito verrà illustrata l'architettura di riferimento per i sistemi I^3.

1.1.1 Architettura dei sistemi I^3

L'architettura dei sistemi I^3 definita dall'ARPA, cerca di far fronte a problemi complessi come:

1. *Eterogeneità delle sorgenti*: differenze tra i tipi di dato, schemi logici, interfacce per accedere ai dati;
2. *Evoluzione delle sorgenti dati*: si possono aggiungere nuove sorgenti o modificare\eliminare quelle già esistenti;
3. *Dimensioni delle fonti*: bisogna far fronte all'aumento dei dati presenti in una singola sorgente e di conseguenza all'aumento dei tempi di risposta;
4. *Semantica nascosta*: bisogna dedurre regole dai differenti schemi, per elaborare ed interpretare i dati da integrare;
5. *Necessità di sistemi modulari e riusabili*: questo punto è fondamentale per ridurre i tempi ed i costi di sviluppo delle varie applicazioni, e far fronte ai mutamenti tecnologici, che inevitabilmente si susseguono nel tempo.

L'architettura del progetto I^3 si propone di evidenziare (separando in più moduli), i vari servizi che devono essere svolti ai fini dell'integrazione intelligente dell'informazione. I servizi evidenziati sono cinque:

- *Coordination Services*
- *Administration Services*
- *Integration and Sematic Trasformation*
- *Wrapping Services*
- *Auxiliary Services*

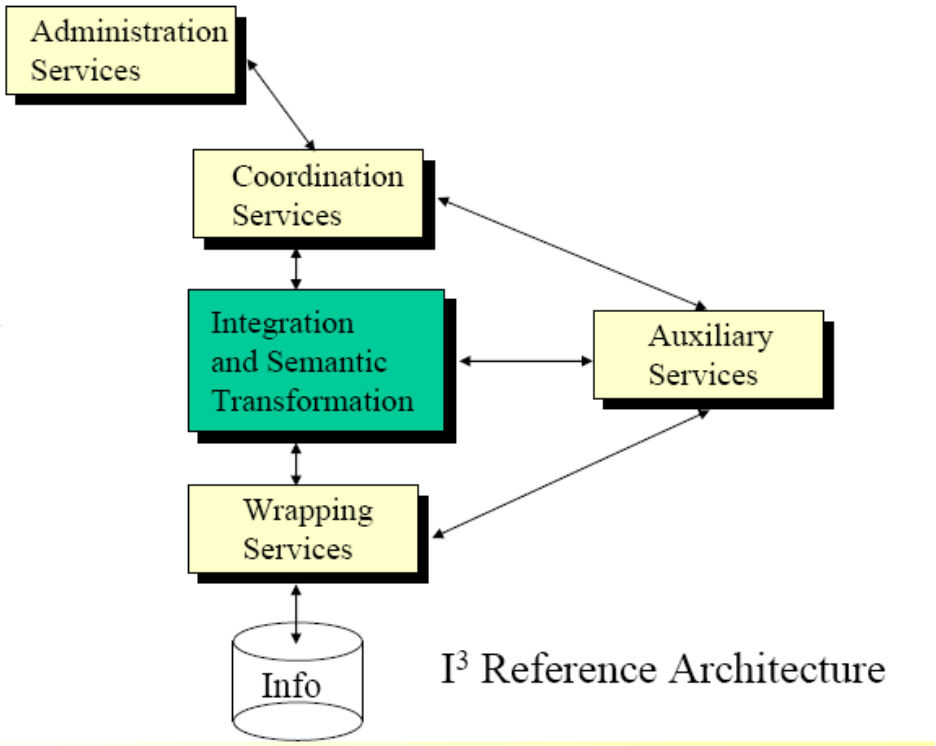

Figura 1. Architettura di riferimento I^3

Di seguito si descriveranno brevemente i servizi precedentemente elencati.

Coordination Services

I servizi di coordinamento hanno un ruolo di supporto, sia in fase di progettazione di nuove configurazioni, che a tempo di esecuzione delle richieste dell'utente. Questi servizi sono di alto livello e oltre ad individuare quali sorgenti possono essere utili per soddisfare una data richiesta, presentano all'utente finale l'intero sistema, diviso fra i suoi vari moduli, come un blocco unico ed omogeneo. Grazie ai servizi di coordinamento, quindi, le divisioni interne di un sistema I^3, sono trasparenti all'utente. I principali moduli appartenenti a questa sezione sono:

- *Broker*: il suo compito è quello di reperire gli strumenti in grado di trattare le richieste dell'utente. Il broker si occupa di contattare un modulo alla volta.
- *Interative query formulation*: si tratta di un modulo di ausilio nella espressione di una query che ha come oggetto lo schema integrato. In particolare è di aiuto se una query già espressa non ha prodotto risultati interessanti.

- *Primitive di costruzione delle configurazioni*: servono a scegliere quali servizi e quali strumenti possono essere utilizzati per la costruzione di una configurazione e come collegarli tra di loro.

Integration and Semantic Transformation

I servizi d'integrazione semantica, hanno come input una o più sorgenti di dati tradotte dai servizi di wrapping e come output, la "*vista*" integrata o trasformata di queste informazioni. Essi vengono indicati spesso come servizi di mediazione.
I principali sono:

- *Servizi d'integrazione degli schemi*: creano il vocabolario e le ontologie condivise dalle sorgenti; integrano gli schemi in una vista globale, mantengono i mapping tra gli schemi globali e le sorgenti;
- *Servizi d'integrazione dell'informazione*: aggregano, riassumono ed estraggono le risposte di più sottoquery per fornire un'unica risposta alla query originale;
- *Servizi di supporto al processo d'integrazione*: sono utilizzati quando la query deve essere scomposta in più sottoquery da inviare a fonti differenti, con la necessità di integrare in seguito i loro risultati.

Wrapping Services

I servizi di wrapping fungono da interfaccia tra il sistema integratore e le singole sorgenti, in particolare, rendendo omogenee le informazioni. Si comportano come dei traduttori dai sistemi locali ai servizi di alto livello dell'integratore. Il loro obiettivo è, quindi, quello di standardizzare il processo di *wrapping* delle sorgenti, permettendo la creazione di una libreria di fonti accessibili; inoltre, il processo di realizzazione di un *wrapper* dovrebbe essere standardizzato, in modo da poter essere riutilizzato per altre fonti.

Auxiliary Services

I servizi ausiliari aumentano le funzionalità degli altri servizi e sono utilizzati prevalentemente dai moduli che operano direttamente sulle informazioni; essi vanno dai semplici servizi di monitoraggio del sistema, ai servizi di propagazione degli aggiornamenti e di ottimizzazione.

1.1.2 Il Mediatore

Il Mediatore è un modulo intermedio che si pone tra l'utente e le sorgenti dati. Secondo la definizione di Wiederhold in [2] *"un Mediatore è un modulo software che sfrutta la conoscenza di livello superiore. Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno o al più, da pochi esperti".*

I compiti del Mediatore sono:

- Assicurare un servizio stabile, anche quando cambiano le risorse;
- Amministrare e risolvere le eterogeneità delle diverse fonti;
- Integrare le informazioni ricavate da più risorse;
- Presentare all'utente le informazioni attraverso un modello scelto dall'utente stesso.

Nell'approccio architetturale I^3 possiamo distinguere principalmente tre livelli:

1. *utente*: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un unica risposta, come se stesse interrogando un'unica sorgente d'informazioni;

2. *mediatore*: il Mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed, eventualmente, arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio interrogatore) comune a tutte le fonti;

3. *wrapper*: ogni wrapper gestisce una sorgente, ed ha una duplice funzione: da un lato converte le richieste del Mediatore in una forma comprensibile alla sorgente, dall'altro traduce le informazioni estratte dalla sorgente nel modello usato dal Mediatore.

Esistono due approcci fondamentali all'architettura precedentemente descritta:

- *Approccio strutturale*: questo approccio è caratterizzato dall'uso di un self-describing model per rappresentare gli oggetti da integrare, limitando così l'uso delle informazioni semantiche a delle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente, bensì è l'oggetto stesso che, attraverso delle etichette, si auto-descrive, specificando tutte le volte, per ogni suo singolo campo, il significato associato.
- *Approccio semantico*: è l'approccio utilizzato in *MOMIS*, ed è caratterizzato dal fatto che il Mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati); le informazioni semantiche sono codificate in questi schemi, inoltre

deve essere disponibile un modello comune per descrivere le informazioni da condividere e i metadati, ed infine, deve essere possibile un'integrazione (parziale o totale) delle sorgenti dati. In questo modo il Mediatore può individuare i concetti comuni a più sorgenti e le relazioni che li legano.

1.2 Il linguaggio ODL_I^3

Object Definition Language *(ODL)* è il linguaggio per la specifica dell'interfaccia di oggetti e di classi nell'ambito della proposta di standard ODMG-93 [16]. Il linguaggio svolge negli *ODBMS* le funzioni di definizione degli schemi che nei sistemi tradizionali sono assegnate al Data Definition Language. Le caratteristiche fondamentali di *ODL*, al pari di altri linguaggi basati sul paradigma ad oggetti, possono essere cosi riassunte:

- ✓ Definizione di tipi classe e tipi valore;
- ✓ Distinzione fra intensione ed estensione di una classe di oggetti;
- ✓ Definizione di attributi semplici e complessi;
- ✓ Definizione di attributi atomici e collezioni (set, list, bag);
- ✓ Definizione di relazioni binarie con relazioni inverse;
- ✓ Dichiarazione della signature dei metodi.

La sintassi di *ODL* estende quella dell'Interface Definition Language (IDL), il linguaggio sviluppato nell'ambito del progetto Common Object Request Broker Architecture (CORBA). Allo scopo di supportare l'integrazione semantica di informazioni in MOMIS viene introdotto il linguaggio object oriented, ODL_I^3 [14] per la rappresentazione di schemi concettuali che descrivono le sorgenti dati strutturate e semistrutturate. ODL_I^3 riprende le specifiche del linguaggio *ODL* introducendo estensioni utili per l'integrazione di informazioni e per modellare dati semistrutturati.

Costruttore Union: indicato come `union`, viene introdotto per esprimere come dominio di un attributo due o più strutture dati.

Costruttore Optional: indicato come "*", viene introdotto per gli attributi per specificare l'opzionalità.

Relazioni terminologiche: esprimono la conoscenza delle relazioni interschema fra le sorgenti integrate. Possono essere espresse per classi ed attributi (che in seguito verranno denominati termini).

- SYN *(SYNonym-of)*: definita tra due termini t_i e t_j (con $t_i \neq t_j$) che sono considerati sinonimi, ovvero che possono essere interscambiati nelle sorgenti, dal fatto che identificano lo stesso concetto del mondo reale;
- BT *(Broader-Term)*: definita tra due termini t_i e t_j tali che t_i ha un significato più generale di t_j. La relazione BT non è simmetrica. La relazione opposta a BT è NT *(Narrower-Term)* e si ha: $(t_i \, BT \, t_j) \Leftrightarrow (t_j \, NT \, t_i)$;
- RT *(Related-Term)*: definita tra due termini t_i e t_j che sono generalmente usati nello stesso contesto, tra i quali esiste un legame meno forte dei precedenti (foreign key, meronimia …).

Regole: Sono introdotte due tipi di regole:

- *if then*, per esprimere in forma dichiarativa i vincoli di integrità intra ed inter schema;
- regole di *mapping* per esprimere le relazioni esistenti tra lo schema integrato e gli schemi sorgenti.

1.3 Il linguaggio OQL_I^3

Il linguaggio *OQL (Object Query Language)* è un linguaggio di interrogazione ad oggetti, definito nell'ambito della proposta di standard ODMG-93 [16] ha una sintassi chiara e potente simile all'SQL. Questo linguaggio è estremamente versatile ed espressivo, perciò, se da un lato è necessario uno sforzo maggiore nello sviluppo di moduli per l'interpretazione e gestione delle interrogazioni dall'altro si ha la possibilità di sfruttare al meglio le informazioni rappresentate nello schema globale. Le principali caratteristiche di questo linguaggio sono:

- è basato sul modello ad oggetti;
- utilizza una sintassi simile al linguaggio SQL92. Rispetto a SQL92 presenta delle estensioni finalizzate alla gestione degli aspetti object-oriented (cammino di navigazione delle gerarchie di aggregazione delle classi);
- fornisce delle primitive ad alto livello per manipolare insiemi di oggetti (Interface), e anche delle primitive per la gestione di altri tipi strutturati come array, liste e strutture.
- non fornisce in modo esplicito operatori per l'aggiornamento del database, lasciando questo compito a operazioni opportune che fanno parte delle caratteristiche di ogni oggetto che popola il database.

Nel sistema *MOMIS* una query globale viene formulata in OQL_I^3 che rappresenta un estensione del linguaggio OQL. In *Appendice A* viene riportata la sintassi del linguaggio OQL_I^3.

1.4 Il sistema MOMIS

MOMIS [8,28] *(Mediator EnvirOnment for Multiple Information Sources)* è un framework per l'estrazione e l'integrazione di informazioni provenienti da sorgenti dati strutturate e semi-strutturate. Allo scopo di presentare le informazioni estratte ed integrate attraverso un linguaggio comune, viene introdotto un linguaggio object-oriented, chiamato ODL_I^3 derivato dallo standard ODMG [16,17], che utilizza la *Description Logics OLCD (Object Language with Complements allowing Descriptive cycles)*. L'integrazione dell'informazione viene compiuta in modo semi-automatico, utilizzando la conoscenza presente in un Common Thesaurus (definito utilizzando il framework), le descrizioni ODL_I^3 degli schemi sorgenti, tecniche di clustering e di Description Logics. Il processo di integrazione genera una vista virtuale integrata (*Global Virtual View* o *Global Schema*) delle sorgenti sottostanti per le quali sono specificate regole di mapping e vincoli di integrità per la gestione delle eterogeneità. Dato un insieme di sorgenti dati relative ad un dominio è possibile sintetizzare un *GS* [30] che è una concettualizzazione del dominio; un ontologia di dominio relativamente alle sorgenti dati. Il sistema *MOMIS*, basato sull'architettura wrapper/mediator [1], fornisce le modalità e tool aperti per il data management.
MOMIS nasce all'interno del progetto MURST INTERDATA, come collaborazione fra le unità operative dell'Università di Modena e Reggio Emilia e dell'Università di Milano.

1.4.1 Architettura del sistema MOMIS

MOMIS è stato progettato per fornire un accesso integrato ad informazioni eterogenee memorizzate in sorgenti dati strutturate (es. database relazionali, database object oriented) e semi-strutturate (es. file XML). *MOMIS* si basa sull'architettura di riferimento I^3 [1,2] e i componenti principali per la fase di integrazione delle sorgenti (Figura 2) sono:

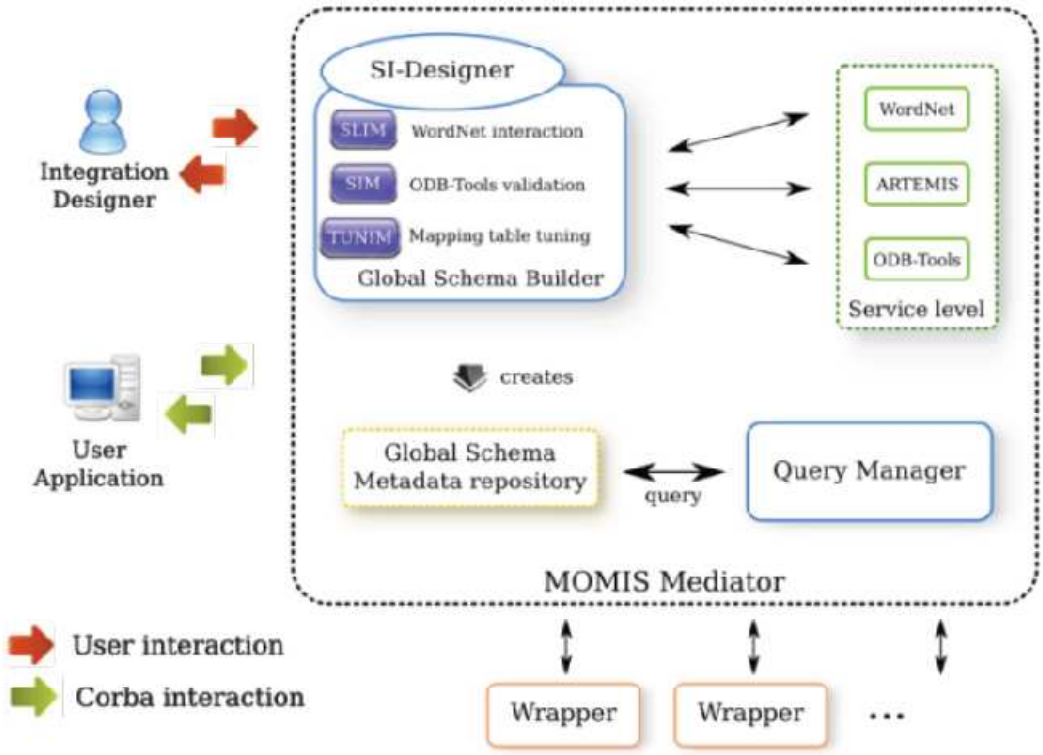

Figura 2. Architettura del sistema MOMIS

1. *Wrapper*: Posti al di sopra di ciascuna sorgente, sono i moduli che rappresentano l'interfaccia fra il Mediatore e le sorgenti dati locali. La loro funzione è duplice:
   - *In fase d'integrazione* forniscono la descrizione dell'informazione in essa contenute. Questa descrizione è fornita attraverso il linguaggio ODL_I^3.
   - *In fase di query processing* traducono la query ricevuta dal Mediatore (espressa quindi nel linguaggio comune d'interrogazione OQL_I^3) in un'interrogazione comprensibile dalla sorgente stessa. Devono inoltre esportare i dati ricevuti come risposta all'interrogazione, presentandoli al mediatore, attraverso il modello comune di dati utilizzato dal sistema.

2. *Mediatore*: Il mediatore rappresenta il cuore del sistema ed è composto da due sottomoduli:
   - ***Global Schema Builder*** (*GSB*): è il modulo che integra gli schemi locali, il quale partendo dalle descrizioni delle sorgenti, espresse attraverso il linguaggio ODL_I^3, genera un unico schema globale da presentare all'utente. Il progettista usa l'interfaccia grafica SI-Designer [18] per integrare le sorgenti e

poi, tutte le informazioni create con SI-Designer vengono salvate in un oggetto CORBA: "*Global Schema Object*". Tale oggetto permetterà al modulo Query Manager di effettuare interrogazioni sullo schema integrato.

- ***Query Manager*** (*QM*): è il modulo di gestione delle interrogazioni. In particolare, utilizzando tecniche di unfolding genera le query in linguaggio OQL_I^3 da inviare ai *wrapper*, partendo dalla singola query formulata dall'utente sullo schema globale. Il *QM* genera automaticamente la traduzione della query nelle corrispondenti sottoquery da spedire alle singole sorgenti, elabora i risultati parziali e restituisce all'utente una risposta unificata.

SI-Designer*: SI-Designer* [18] (*Source Integrator Designer*) è la graphical user interface (*GUI*), per la creazione del global schema, del sistema MOMIS. SI-Designer è un tool di supporto al progettista per le varie fasi dell'integrazione, dall'acquisizione delle sorgenti fino alla messa a punto delle tabelle di Mapping. Si tratta di un contenitore modulare di altri strumenti che permettono la raccolta di informazioni per l'integrazione. *SI-Designer* è composto da quattro moduli:

- ***SIM*** (*Source Integrator Module*): SIM è in grado di estrarre relazioni intensionali dalla conoscenza della struttura degli schemi, validare relazioni inserite dal progettista e di calcolare dalle relazioni esplicite contenute nel Common Thesaurus (grazie al motore inferenziale ODB-Tools [11,12]) tutte le possibili relazioni implicate. *ODB-Tools* è un framework integrato per la validazione degli *OODB (Object Oriented DataBase)* [13,14,15]. ODB-Tools è basato sulla Description Logics OLCD. Il formalismo OLCD *(Object Language with Complements allowing Descriptive cycles)* è una logica descrittiva per basi di dati che estende ODL introducendo nuovi costruttori per l'insieme dei tipi e la possibilità di poter descrivere delle regole d'integrità.
- ***SLIM*** (*Sources Lexical Integrator Module*): estrae le relazioni inter-schema tra nomi di classi e di attributi di differenti sorgenti, sfruttando il sistema lessicale *WordNet*. Il database lessicale *WordNet* è un sistema lessicale di riferimento il cui disegno è ispirato alle teorie psico-linguistiche contemporanee. I termini, infatti, non sono disposti seguendo l'ordine alfabetico, ma per affinità di significato. *WordNet* comprende quattro categorie sintattiche: nomi, verbi, aggettivi e avverbi. Ogni categoria sintattica è suddivisa in diversi insiemi di

sinonimi; ad ognuno di questi insiemi è associato un unico significato, condiviso da tutti i termini ad esso associati. Un termine, può possedere più di un significato ed essere, quindi, presente in molti di questi insiemi, ed anche in più di una categoria sintattica. All'interno del progetto *WordNet*, un insieme di vocaboli che condivide il medesimo significato, prende il nome di synset. Un'altro elemento rilevante, che contraddistingue *WordNet* da un semplice dizionario di vocaboli, è la presenza di relazioni tra i synset. Vi sono diverse tipologie di relazioni che possono collegare due synset, come, ad esempio, l'iperonimia e l'iponimia, tramite cui si è in grado di creare, all'interno dell'intera categoria sintattica, gerarchie di significato. *WordNet* è stato sviluppato presso il Cognitive Science Laboratory della Princeton University sotto la direzione del Professor George A. Miller. Nel framework MOMIS, *WordNet* è utilizzato attraverso l'interfaccia CORBA-2 per l'estrazione delle relazioni inter-schema tra i concetti.

- ***ARTEMIS*** (*Analysis and Reconciliation Tool Environment for Multiple Information Sources*): è un tool che implementa tecniche di clustering basate sull'affinità delle classi. ARTEMIS esegue l'analisi semantica ed il clustering delle classi ODL$_I^3$. L'analisi semantica ha come obiettivo l'identificazione degli elementi nei diversi schemi che hanno relazioni semantiche. Il concetto di affinità viene introdotto per valutare il livello di relazione semantica. I coefficienti di affinità sono calcolati tra coppie di elementi, per esprimere la loro similarità nel rappresentare la stessa informazione, in schemi differenti. Gli elementi con maggiore affinità sono identificati e raggruppati utilizzando un processo di hierarchical clustering. ARTEMIS è stato sviluppato presso l'Università di Milano e Brescia.
- ***TUNIM*** (*Tuning of the Mapping Table*): questo modulo gestisce l'ultima fase del procedimento di integrazione per la creazione dello schema globale. Partendo dalle relazioni del Common Thesaurus, per ciascuno dei cluster individuati da ARTEMIS viene creata una classe globale. Ciascuna classe globale è caratterizzata da un insieme di attributi globali e una Mapping Table; l'insieme di attributi ne definisce la struttura mentre la Mapping Table indica quali informazioni locali sono mappate in ogni attributo globale.

Attualmente i wrapper implementati in MOMIS sono:

- ✓ Wrapper JDBC generico;
- ✓ Wrapper SQLServer;
- ✓ Wrapper XML;
- ✓ Wrapper ODBC - MS_Access;
- ✓ Wrapper MySQL;
- ✓ Wrapper Oracle;
- ✓ Wrapper OWL.

1.4.2 Generazione del GS

Lo schema globale integrato (*GS*) generato con il sistema *MOMIS* [4,5,30] è composto da un insieme di classi globali, e dai mapping tra gli attributi globali delle classi globali e gli attributi delle sorgenti locali. Questi mapping vengono generati in modo semiautomatico. Il *GS* è un ontologia di dominio che rappresenta le sorgenti coinvolte nell'integrazione. Di seguito viene descritto il processo di generazione del *GS* (Figura 3).

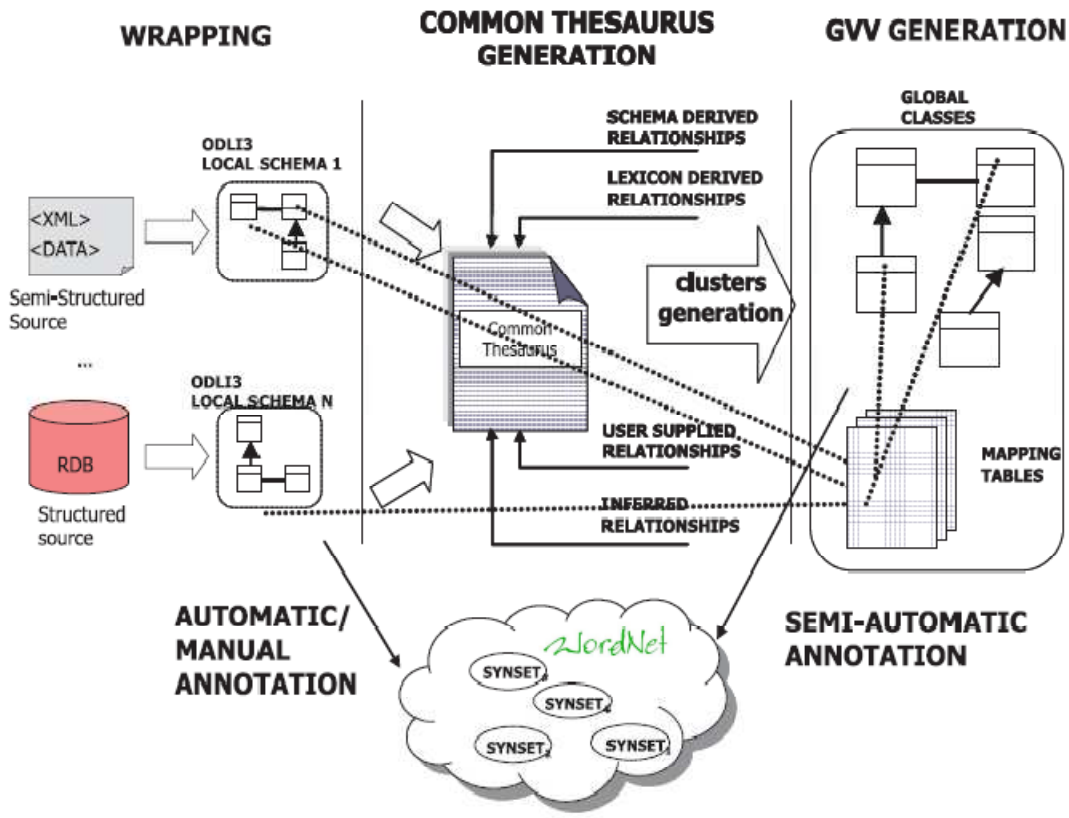

Figura 3. Generazione del GS

Estrazione degli schemi delle sorgenti dati

I wrapper acquisiscono gli schemi delle sorgenti locali e li traducono nel linguaggio ODL_I^3. Per le sorgenti dati strutturate (es. relational database, object oriented database) il processo di acquisizione degli schemi è immediato, essendo che uno schema è sempre disponibile per queste sorgenti, quindi questo schema può essere direttamente tradotto in ODL_I^3, mentre l'estrazione degli schemi da sorgenti dati semistrutturate richiede l'ausilio di ulteriori tecniche di elaborazione. Per eseguire l'estrazione degli schemi da file XML, è stato sviluppato uno specifico wrapper che traduce automaticamente gli schemi XSD in strutture relazionali ed importa i dati in formato relazionale.

Annotazione delle sorgenti dati

Le relazioni inter-schema esprimono un legame semantico tra i nomi delle classi e degli attributi, degli schemi delle diverse sorgenti. In MOMIS si possono annotare i termini (nomi delle classi e nomi degli attributi) degli schemi locali, rispetto all'ontologia lessicale WordNet: ciò consiste nell'associare un significato a ciascun termine. Attraverso degli algoritmi di disambiguazione si possono annotare automaticamente i termini. Comunque il progettista, può rivedere il significato che l'annotazione automatica ha associato a ciascun termine.

La fase di annotazione manuale consiste sostanzialmente in due fasi:

1. *Scelta della forma base*: Consiste nell'inserire il termine di lingua inglese che meglio descrive il concetto rappresentato dalla classe o dall'attributo in esame;
2. *Scelta del significato (sense)*: Il progettista sceglie, in base al dominio, il significato del termine che sta annotando, si può far corrispondere ad un termine zero, uno o più significati.

Generazione del Common Thesaurus

Il Common Thesaurus è costituito da relazioni terminologiche che esprimono la conoscenza intra-schema e inter-schema. Le relazioni possono essere di tre tipi:

- ***SYN*** (*SYNonym-of*, relazioni di sinonimia): definita tra 2 termini che possono essere scambiati in quanto identificano lo stesso concetto;

- ***BT*** (*Broader-Term*, relazioni di specializzazione): definita tra due termini, di cui uno può avere un significato più generale dell'altro;
- ***RT*** (*Related-Term*, relazioni di aggregazione): definita tra due termini tra i quali esiste un legame generico.

Il Common Thesaurus viene costruito attraverso un processo incrementale, durante il quale vengono aggiunte relazioni, seguendo il seguente ordine: schema-derived relationships (relazioni derivanti dagli schemi), lexicon-derived relationships (relazioni lessicali), designer-supplied relationships (relazioni fornite dal progettista) e inferred relationships (relazioni inferite) .

Schema-derived relationships

Analizzando gli schemi delle sorgenti locali, descritte attraverso il linguaggio ODL_I^3 e utilizzando le tecniche di inferenza OLCD, vengono estratte delle relazioni intra-schema, le quali esprimono le gerarchie di generalizzazione (relazioni BT, NT) e di aggregazione (relazioni RT), più altre relazioni intra-schema dalle sorgenti relazionali, dalla definizione di chiavi esterne (relazioni RT, BT).

Lexicon-derived relationships

Le relazioni lessicali derivano dall'annotazione dei termini effettuata rispetto all'ontologia lessicale, WordNet. Queste relazioni interschema esprimono un legame semantico tra i nomi delle classi e degli attributi appartenenti a schemi di sorgenti diverse (possono essere relazioni SYN, BT, NT, RT) e vengono estratte sulla base delle relazioni lessicali tra i nomi delle classi e degli attributi.

Designer-supplied relationships

Essendo a conoscenza del dominio, il progettista può fornire ulteriori relazioni terminologiche. Queste relazioni completano quelle estratte automaticamente.

Inferred relationships

Servendosi delle tecniche di Description Logics di ODB-Tools [11,12] vengono calcolate tutte le possibili relazioni di sussunzione e aggregazione. Tali relazioni sono poi tradotte in relazioni terminologiche che arricchiscono il Common Thesaurus.

Generazione del Global Schema

Figura 4. Generazione Global Schema e Mapping Table

MOMIS utilizza le relazioni contenute nel Common Thesaurus e gli schemi delle sorgenti locali per generare il *GS*. Il *GS* è costituito da un insieme di classi globali; per ogni classe globale viene generata una Mapping Table in cui vengono definite le corrispondenze tra gli attributi globali di quella classe e gli attributi degli schemi delle sorgenti locali.

Questo processo di generazione può essere cosi articolato (Figura 4):

1. *Calcolo delle affinità*: L'obiettivo è l'identificazione delle classi che hanno relazioni semantiche tra di loro. Per tutte le possibili coppie di classi locali ODL_I^3, vengono calcolati i coefficienti di affinità attraverso il componente ARTEMIS. Quest'ultimo determina il grado di matching tra due classi basandosi sui loro nomi *(Name Affinity cofficient)* e sulla loro struttura (attributi) *(Structural Affinity coefficient)*. Poi viene determinato tramite una combinazione lineare delle: Name e Structural Affinity, il *Global Affinity coefficient (GA)*.
2. *Generazione dei cluster*: Un algoritmo di clustering classifica automaticamente le classi ODL_I^3, l'output di questa procedura è un *Affinity Tree* (albero di affinità) dove le foglie rappresentano tutte le classi locali (foglie contigue sono classi caratterizzate da alta affinità, mentre foglie tra loro molto lontane rappresentano invece classi a bassa affinità) ed ogni nodo rappresenta un livello di clusterizzazione ed ha associato il coefficiente di affinità dei due sottoalberi che unisce. Dall'affinity tree, tramite un meccanismo a soglia vengono generati i cluster. Ogni cluster sarà costituito da tutte le classi ODL_I^3 appartenenti ad un sottoalbero, che al nodo radice ha un coefficiente maggiore del valore di soglia. A seconda del contesto di integrazione il progettista sceglie il livello di soglia più appropriato.
3. *Generazione degli attributi globali e delle Mapping Table*: Una classe globale *G* viene generata per ogni cluster.

   G = (*L, GA*) dove:

- *L* è l'insieme delle classi locali mappate nella classe globale *G*;
- *GA* è l'insieme degli attributi globali di *G*.

L'insieme degli attributi globali *GA* è ottenuto tramite l'unione degli attributi di tutte le classi affini appartenenti al cluster dal quale è stata generata la classe globale e dalla fusione degli attributi "simili" (sfruttando le relazioni terminologiche contenute nel Common Thesaurus).

Una *Mapping Table* (*MT*), che contiene i mapping tra gli attributi globali e quelli locali, viene creata per ogni classe globale. La *MT* è una tabella *GA* X *L*; un elemento della tabella, MT[*GA*][*L*], rappresenta l'insieme degli attributi locali di L mappati nell'attributo globale *GA.*

L'attributo globale *GA* può assumere diversi valori:

- Il valore di *GA* può essere uguale al valore *LA*, dell'attributo locale della sorgente *L* su cui è mappato l'attributo globale *GA*. MT[*GA*][*L*] = *LA* (es. attributo globale *dept* mappato nell'attributo locale *dept* della sorgente *QMTestDB2.L2* in Tabella 1).
- Il valore di *GA* può essere ottenuto tramite una composizione dei valori (LA_i) assunti da un insieme di attributi della classe locale *L*. MT[*GA*][*L*] = $LA_1 \ldots LA_n$ (es. due attributi diversi della sorgente *QMTestDB1.L1 (Firstname e Lastname)* vengono mappati nello stesso attributo globale *name*).
- Il valore di *GA* può essere una costante definita dal progettista. MT[*GA*][*L*] = *const*.
- Il valore di *GA* può essere indefinito per quella sorgente locale. MT[*GA*][*L*] = *no mapping* (es. attributo dept non è mappato in nessun attributo della sorgente *QMTestDB3.L3*).

In Tabella 1 viene presentato un esempio di Mapping Table. La classe globale G è mappata in tre classi locali *QMTestDB1.L1, QMTestDB2.L2* e *QMTestDB3.L3).*

G	*QMTestDB1.L1*	*QMTestDB2.L2*	*QMTestDB3.L3*
email	*Email*	*mail*	*Email*
name	*First_Name* *Last_Name*		
dept		*dept*	
section		*section*	
year	*year*		

Tabella 1. Esempio di Mapping Table

La tabella di mapping può essere ulteriormente raffinata dal progettista, il processo di Mapping Refinement viene descritto nel paragrafo 1.4.3.

Annotazione del GS

Il GS viene automaticamente annotata, cioè ad ogni elemento viene associato il significato più generale estratto dall'annotazione delle sorgenti.

1.4.3 Mapping Refinement

Come sopra descritto per ogni classe globale viene generata una Mapping Table (un esempio di Mapping Table viene riportato in Tabella 1) che mantiene i mapping tra gli attributi globali della classe e gli attributi delle classi locali (classi che appartengono al cluster da cui è stata generata la classe globale). In *MOMIS* la *Mapping Table* ottenuta può essere ulteriormente raffinata utilizzando:

- *Data Conversion Functions* (Funzioni di trasformazione dei dati): per ogni elemento MT[*GA*][*L*], non nullo, della tabella di mapping si può definire una Data Conversion Function, denotata con MTF[*GA*][*L*], che stabilisce come gli attributi locali di *L* saranno mappati nell'attributo globale *GA*. MTF[GA][L] è una funzione che deve essere eseguita direttamente sulla sorgente locale e quindi deve essere supportata dal local source wrapper e dalla sorgente stessa.
- *Join Conditions* (Condizioni di Join): vengono definite tra gli attributi delle classi locali mappate nella medesima classe globale, affinché il sistema possa identificare istanze dello stesso oggetto del mondo reale nelle diverse sorgenti. Date 2 classi locali L_1 e L_2 mappate in *G*, la Join Condition definita tra L_1 e L_2, denotata con $JC(L_1, L_2)$, è un espressione su $L_1.A_i$ e $L_2.A_j$ dove A_i (A_j) sono attributi globali con un not null mapping (mapping non nullo) in L_1 (L_2). Fissato un insieme di attributi globali *JA*, Join Attributes, tali che $\forall A_i \in JA$, $1 \leq i \leq n$, l'attributo ha un not null mapping in L_1 (L_2), la join condition è cosi definita:

  $$L_1.A_1 = L_2.A_1 \text{ AND } \ldots L_1.A_n = L_2.A_n$$

  La Join Condition viene utilizzata a tempo di esecuzione delle query per identificare tuple che si riferiscono allo stesso oggetto del mondo reale, quindi

queste tuple possono essere considerate semanticamente equivalenti. La Join Condition è un modo per risolvere il problema del Object Identification nei sistemi di integrazione [3].

- *Resolution Functions* (Funzioni di risoluzione): la fusione di dati provenienti da diverse sorgenti evidenzia il problema dell'inconsistenza delle informazioni. In MOMIS, per risolvere i conflitti tra i dati, vengono adottate le funzioni di risoluzione [29], che si possono definire per ogni attributo globale mappato in più attributi locali provenienti da sorgenti diverse. Un attributo che non presenta un conflitto dati (cioè le istanze dello stesso oggetto reale in differenti classi locali, possiedono lo stesso valore in corrispondenza di un determinato attributo) è chiamato attributo omogeneo; per questo tipo di attributi non c'è bisogno di una funzione di risoluzione. Un attributo globale mappato in un'unica classe locale rappresenta un caso particolare di attributo omogeneo.

1.5 Il progetto MOMIS Open Source

L'Open Source (sorgente aperta) indica un software i cui autori (più precisamente i detentori dei diritti) ne permettono, anzi ne favoriscono il libero studio e l'apporto di modifiche da parte di altri programmatori indipendenti. Questo è realizzato mediante l'applicazione di apposite licenze d'uso, es. licenza GNU GPL *(General Public License)*.

La collaborazione di più parti permette al prodotto finale di raggiungere una complessità maggiore di quanto potrebbe ottenere un singolo gruppo di lavoro. L'Open Source era nato principalmente come un movimento filosofico che consisteva di una nuova concezione della vita, aperta e refrattaria ad ogni oscurantismo, che l'Open Source si proponeva di superare mediante la condivisione della conoscenza; oggi è diventato un fenomeno internazionale di successo all'interno delle aziende.

I software Open Source attualmente più diffusi sono Firefox, OpenOffice… oltre ad un gran numero di progetti rivolti non all'utente finale ma ad altri programmatori, es. *HSQLDB (Hyper Structured Query Language DataBase)* [23] (DBMS di supporto del *Query Manager* del sistema *MOMIS*). Inoltre ci sono i sistemi operativi BSD, GNU e il kernel Linux, i cui autori hanno contribuito in modo fondamentale alla nascita del movimento. La comunità Open Source è molto attiva, comprende decine di migliaia

di progetti, numero che cresce quotidianamente. Nel saggio "La cattedrale e il bazaar" di Eric Steven Raymond, che viene considerato il manifesto del movimento Open Source, vengono descritti i vantaggi dell'adozione del modello di sviluppo Open Source che viene chiamato modello bazaar, a cui viene contrapposto il modello cattedrale, che rappresenta un modello tipico delle aziende commerciali. Infatti l'adozione del modello bazaar, come definito da Eric S. Raymond favorisce la revisione del codice e, soprattutto, offre una base di testing e controllo molto ampia che difficilmente una software house di medie dimensioni si può permettere. La tesi centrale di Eric S. Raymond è: *"Dato un numero sufficiente di occhi, tutti i bug vengono a galla"*. Come viene riportato in [25] gli sviluppatori del software Open Source hanno, forse in modo inconsapevole, creato un nuovo paradigma economico per lo sviluppo del software. La *Gartner* in [22], afferma che il modello del software Open Source sta evolvendo l'industria del software e molte aziende hanno trovato in questo modello rivoluzionario di sviluppo ed economico la chiave per il successo sul mercato. Secondo le previsioni di Gartner*, "nel 2011, all'incirca il 50% delle aziende leader nell'information tecnology, adotteranno il modello Open Source in modo formale nella loro strategia aziendale e definiranno delle policy aziendali per la gestione del modello"*. Infatti, il paradigma Open Source porta dei vantaggi per il cliente il quale avrà a disposizione un software di elevata qualità e affidabilità ad un costo minore, inoltre la disponibilità del codice sorgente consente una personalizzazione e una raffinazione del prodotto, permettendo di modificarne ogni aspetto e poterlo adattare alle proprie esigenze specifiche. Invece per quanto riguarda il produttore l'adozione del modello Open Source consente di diminuire i costi iniziali di sviluppo e di ridurre i tempi, consentendo un time to market minore, però solo una soluzioni di base del software sarà completamente open, e a pagamento versioni con funzionalità aggiuntive e specifiche. In questo caso il software productor avrà diverse fonti di guadagno: vendita del software accessoriato, assistenza, mantenimento, aggiunta di nuove funzionalità ecc. MOMIS è un progetto di data integration Open Source progettato e sviluppato dal DBGroup dell'Università degli studi di Modena e Reggio Emilia. In [24] viene descritto in modo dettagliato il mondo Open Source e il progetto per rendere MOMIS Open Source.

Capitolo 2

2 Il Query Management nel sistema MOMIS

2.1 Query Processing nel sistema MOMIS

Molti sistemi di integrazione dati, come anche *MOMIS*, sono caratterizzati da un architettura wrapper/mediator [1]. Il sistema di integrazione genera un *Global Virtual Schema* (*GS*) degli schemi delle sorgenti locali da integrare. I dati continuano a rimanere nelle sorgenti e il *GS* rappresenta una vista virtuale, integrata e riconciliata delle sorgenti dati.

Quindi in un sistema di integrazione dati, due sono gli aspetti cruciali: la modellazione dei mapping e la generazione del *GS* (vedere 1.4.2). Per quanto riguarda la modellazione dei mapping tra le sorgenti dati, in letteratura sono stati proposti due approcci*: Local-As-View (LAV)* e *Global-As-View (GAV)*. Nell'approccio *LAV* ogni sorgente locale viene descritta in termini dello schema globale. Un aspetto negativo di questo approccio è la necessità di adozione di tecniche di reasoning in fase di query processing, dall'altra parte un aspetto positivo è la semplicità nell'aggiungere nuove risorse da integrare.

In *MOMIS* viene adottato l'approccio *GAV* [28], nel quale le classi dello schema globale sono definite come viste sulle sorgenti locali, cioè ogni classe globale è definita da una query sulle classi locali. Il vantaggio di questo approccio è la gestione del query processing, ma dall'altra parte, estendere lo schema con nuove sorgenti dati risulta più difficile, aggiungere una nuova sorgente può incidere nella definizione di diverse classi del *GS* [31, 33].

2.1.1 Definizione formale del sistema MOMIS

In MOMIS le sorgenti locali e globali sono descritte tramite il linguaggio object-oriented, con una semantica basata su di una Description Logics, chiamato ODL_I^3.

I componenti fondamentali di MOMIS, come sistema di integrazione sono:

- Il *Global Schema (GS)*, cioè lo schema globale espresso in ODL_I^3;

- Un insieme di *N* sorgenti locali; ogni sorgente ha il proprio schema in ODL_I^3;
- Un insieme *M* di *GAV* mapping assertions (asserzioni di mapping) tra il *GS* e le *N* sorgenti locali. Ogni asserzione di mapping associa ad un elemento *G* di *GS* una mapping query (query di mapping) q_G, espressa su un insieme di classi locali. Cioè per ogni classe globale $G \in GS$ viene definito un insieme di classi locali, denotato con *LG*, appartenenti alle sorgenti locali *N*, e una mapping query q_G espressa sugli schemi delle classi locali *LG*, attraverso l'operatore *full outerjoin-merge*.

2.1.2 La Mapping Query q_G

Per definire la mapping query associata ad una classe globale *G* viene utilizzato l'operatore *full outerjoin-merge*. Se la classe globale G è mappata in n classi locali L_1, L_2 … L_n, la mapping query q_G avrà la seguente espressione:

```
(L1 full outer join L2 on JC(L1,L2))
 full outer join L3 on (JC(L1, L3) OR JC(L2, L3))
 ...
 full outer join Ln on (JC(L1, Ln) OR JC(L2, Ln) OR ... OR
 JC(Ln-1, Ln))
```

Quindi viene effettuato il full outer join sulla base delle Join Conditions ottenute nella fase di raffinamento della Mapping Table.

2.1.3 Query Processing

Per quanto riguarda la modellazione dei mapping tra le classi globali e quelle locali, in *MOMIS* viene adottato l'approccio *GAV* [28]. L'adozione di questo approccio determina la modalità con cui una query posta sullo schema globale, viene riscritta in termini delle classi locali su cui è mappata la classe globale. Il query processing nell'approccio *GAV* si basa sul processo di *Query Unfolding*. In MOMIS ad ogni classe globale *G* viene associata una mapping query q_G, espressa sugli schemi delle classi locali *LG* mappate in *G*. Il *Query Unfolding* consiste nel riscrivere la query globale formulata sulla classe globale $G \in GS$, in termini delle classi locali su cui è mappata la classe globale. In *MOMIS* le query vengono formulate in OQL_I^3

(Appendice A). In seguito verrà descritto il Query Unfolding nel caso di una query globale espressa su un'unica classe globale del GS (Single class query) e nel caso di una query globale espressa su più di una classe globale (Multiple class query) [20].

2.1.3.1 Query Unfolding - Single class query

Il processo di *Query Unfolding* viene eseguito per ogni query globale *Q*, formulata su una classe globale $G \in GS$. Nella fase di generazione del GS per ogni classe globale viene generata una Mapping Table, in cui si definiscono i mapping tra la classe globale e le classi locali *LG* delle diverse sorgenti dati.

Il *Query Unfolding* genera:

- un insieme di local query (query locali) *LQ*, che verranno inviate tramite i wrapper alle sorgenti dati corrispondenti, su cui verranno eseguite;
- la mapping query q_G per fondere i risultati parziali;
- la final query (query finale) per applicare le eventuali resolution functions e le clausole residue.

In seguito verranno descritti i passi che costituiscono il *Query Unfolding*.

Data una query globale Q:

```
Q =  SELECT <Q_select_list>
     FROM G
     WHERE <Q_condition>
```

Il processo di query unfolding è costituito dai seguenti passi:

1. *Generazione delle query locali*:

```
LQ = SELECT <L_select_list>
     FROM L
     WHERE <L_condition>
```

Una local query viene generata per ogni classe $L \in LG$, cioè per ogni classe locale su cui la classe globale *G* è mappata. La `<L_select_list>` è calcolata considerando l'unione degli:

- attributi globali in `<Q_select_list>` con un mapping non nullo in *L*;
- attributi globali usati per esprimere la join condition per *L*;
- attributi globali in `<Q_condition>` con un mapping non nullo in *L*.

L'insieme degli attributi globali è trasformato nel corrispondente insieme di attributi locali tramite l'utilizzo della Mapping Table. Mentre, la condizione <L_condition> è calcolata eseguendo un mapping sui predicati. Ogni predicato della <Q_condition> viene riscritto in maniera tale da essere supportato dalla sorgente locale. Il mapping dei predicati viene eseguito prendendo in considerazione eventuali Data Conversion functions e Resolution Functions definite durante la fase di Mapping Reffinement sulla Mapping Table.

Un predicato del tipo (*GA* op value), presente nella <Q_condition> verrà riscritto in <L_condition> nel modo seguente:

(*MTF[*GA*][*L*] *op value)

If *GA* è un attributo omogeneo

and MT[*GA*][*L*] è non nullo

and l'operatore op è supportato da *L*

and MTF data conversion function è supportata da *L*

true altrimenti

Le clausole presenti in <Q_condition> che coinvolgono attributi non omogenei su cui sono state applicate delle Resolution Functions non possono essere tradotte in condizioni locali: queste condizioni verranno considerate come condizioni residue e verranno risolte solo a livello globale.

2. *Generazione della mapping query q_G*:

I dati restituiti dall'esecuzione delle local query *LQ* sulle sorgenti dati locali, vengono fusi attraverso l'esecuzione della mapping query q_G corrispondente a G.

3. *Generazione della final query*:

Attraverso la final query verranno eseguite le eventuali Resolution Functions presenti e le condizioni residue che non è stato possibile eseguire direttamente sulle sorgenti locali.

2.1.3.2 Query Unfolding - Multiple class query

Dato un insieme di n classi globale G_1, G_2,…, G_n consideriamo la seguente query globale *MQ*;

```
MQ = SELECT <MQ_select_list>
     FROM G1, G2,…, Gn
     WHERE <MQ_condition>
     AND <join_condition>
```

Il query unfolding è costituito dai seguenti due passi:

1. La generazione di tante query Q_i con $1 \leq i \leq n$, Q_i una Single class query, cioè query che coinvolge una sola classe globale:

```
Qi = SELECT <Qi_select_list>
     FROM Gi
     where <Qi _condition>
```

La <Qi_select_list> è calcolata considerando l'unione degli:

- attributi in <MQ_select_list> relativi alla classe globale G_i
- attributi in <join_condition> relativi alla classe globale G_i

La <Qi _condition> viene riscritta prendendo in considerazione le condizioni che coinvolgono gli attributi globali di G_i.

2. Viene eseguito il Query Unfolding per ogni Single class query *Qi*.

2.2 Architettura del Query Manager

Il *Query Manager* [20] è costituito da diversi blocchi funzionali che coincidono con i moduli software che sono stati implementati. Il Query Manager è implementato in Java. Di seguito verrà descritto l'architettura del Query Manager (Figura 5) sia nel caso di una Single class query, cioè una query globale formulata su una singola classe globale, sia nel caso di una Multiple class query (Figura 6):

• ***Graphical User Interface***: permette all'utente di inserire la query formulata sul *GS* e il risultato viene visualizzato in una tabella.

• ***Unfolder***: riceve la query globale *Q* e sfruttando i mapping tra gli attributi globali e quelli locali genera il *Query Plan QP*. Prima della generazione del *QP* viene valutata la correttezza sintattica (attraverso un processo di parsing) e semantica della query *Q*. Il *QP* è composto dall'insieme delle local query da eseguire sulle singole sorgenti coinvolte nell'integrazione, dalla mapping query q_G, per unire i risultati parziali (attraverso l'operatore full outerjoin-merge) e dalla final query per applicare le eventuali Resolution Functions e le condizioni residue.

• ***Join Engine***: riceve come input il *QP* e esegue l'insieme delle query per ottenere il risultato finale. Il Join Engine, prima invia le local query ai rispettivi wrapper affinché vengano eseguite sulle sorgenti simultaneamente, poi unisce i risultati parziali eseguendo la mapping query, ed infine ottiene il risultato globale applicando la final query. Ogni query locale è inviata al rispettivo wrapper che la traduce nello specifico linguaggio supportato dalla sorgente. Un DBMS funge da supporto per la fusione dei dati parziali, che vengono memorizzati in tabelle temporanee create in un database del DBMS.

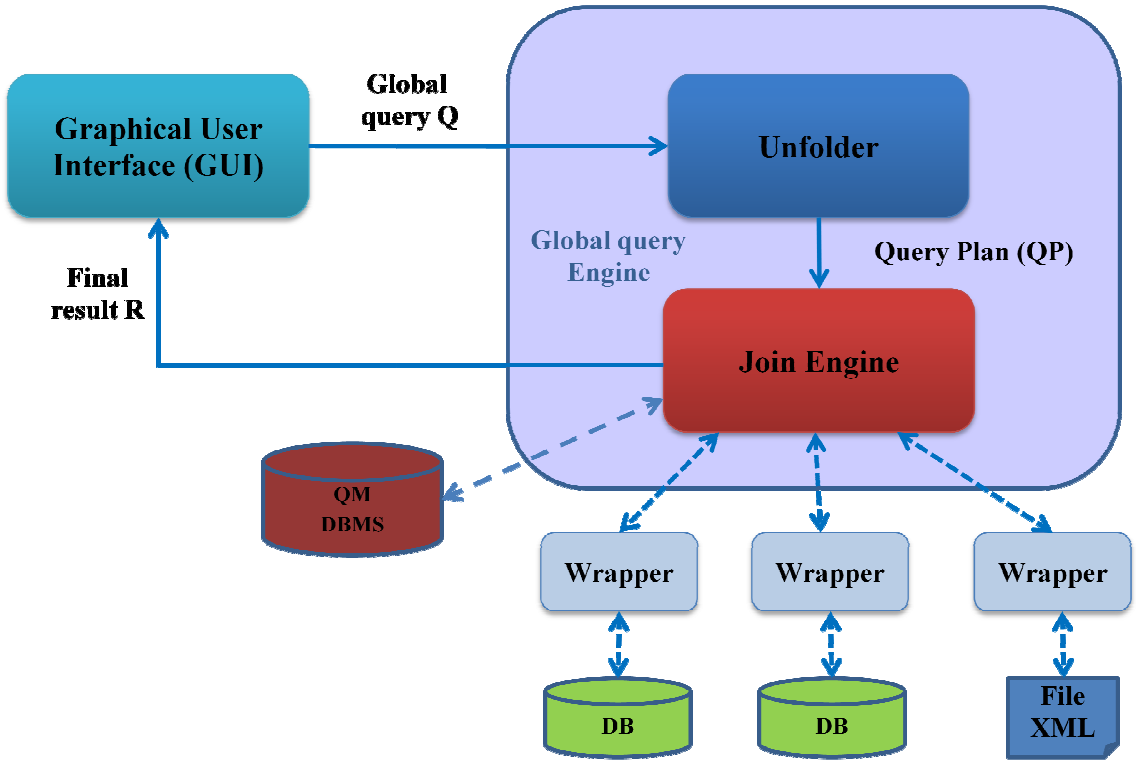

Figura 5. Architettura Query Manager

In Figura 6 viene riportata l'architettura del Query Manager nel caso di una Multiple class query, cioè una query globale formulata su più classi globali.

• ***Multiple class query Unfolder***: riceve la Multiple class query e considerando le clausole di join presenti genera un *Query Plan*. Il *QP* è composto dall'insieme delle global query, ognuna relativa ad una sola classe globale e da una join query per unire i risultati delle varie global query.

• ***Multiple class query Join Engine***: riceve come input il *QP* ed esegue l'insieme delle Single class query per ottenere il risultato finale. Il Join Engine prima esegue l'insieme delle Single class query simultaneamente, in seguito unisce i risultati parziali eseguendo la join query per ottenere il risultato finale. Ogni Single class query è mandata al *Global Query Engine* per essere eseguita sulle sorgenti dati.

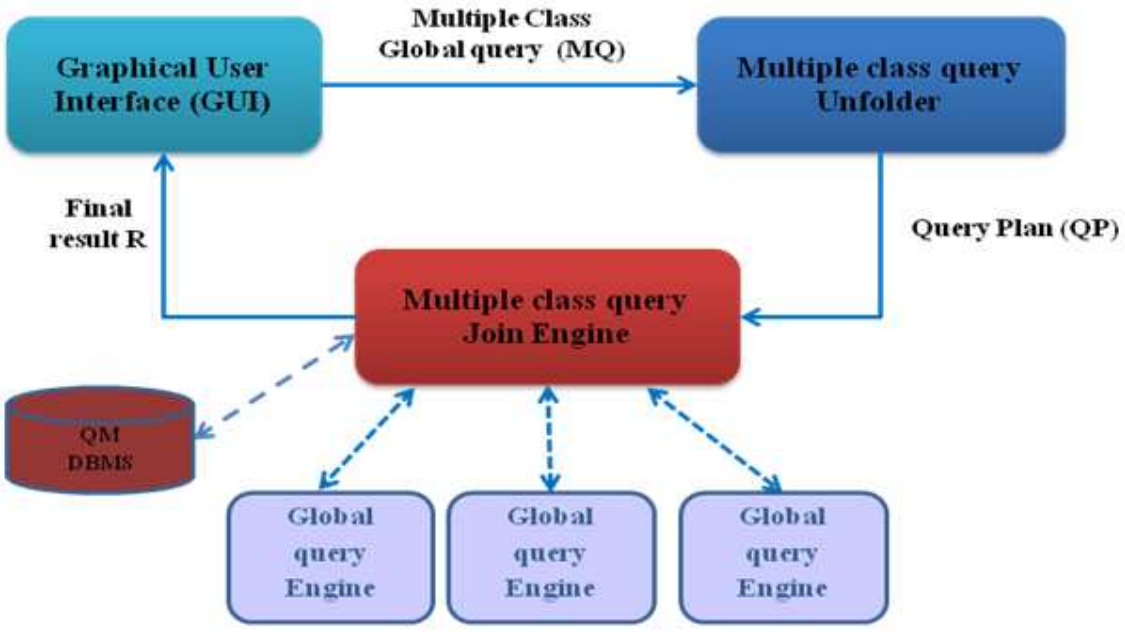

Figura 6. Architettura Query Manager - Multiple class query

Come mostrato in Figura 5 ogni *Global Query Engine* è composto da un modulo *Unfolder* e da un *Join Engine*. Un DBMS funge da supporto per la fusione dei dati parziali, che vengono memorizzati in tabelle temporanee create in un database del DBMS.

Capitolo 3

3 HSQLDB e testing correttezza

3.1 Le classi java del componente Query Manager

Nel secondo capitolo è stato descritto il query processing nel sistema *MOMIS;* il componente software che si occupa dell'esecuzione delle query formulate dall'utente è il *Query Manager* (Figura 5). Per effettuare l'elaborazione della query viene utilizzato un *DBMS* di supporto; in un database di quest'ultimo vengono create delle tabelle temporanee nelle quali verranno inseriti i dati provenienti dalle sorgenti. Questi dati successivamente elaborati costituiranno la risposta che verrà restituita all'utente. Questo componente è implementato in Java (come l'intero progetto *MOMIS*) ed è costituito da 10 package e più di 100 classi. In seguito elenchiamo alcune delle classi principali del Query Manager:

- ✓ Classe *QueryManagerConfiguration.java,*
  (package *it.unimo.datariver.querymanager),* questa classe mette a disposizione dei metodi per impostare i parametri di connessione al DBMS di supporto.
- ✓ Classe *QueryManagerImpl.java,*
  (package *it.unimo.datariver.querymanager*), questa classe mette a disposizione dei metodi per inizializzare il query manager.
- ✓ Classe *DatabaseHandler_base.java,*
  (package *it.unimo.datariver.querymanager.joinEngine.function),* questa classe astratta viene estesa dalle classi *handler* relative ad un particolare DBMS di supporto, contiene un metodo createTable e dei metodi per nominare le tabelle locali e le tabelle globali temporanee create sul DBMS di supporto.
- ✓ Classe *DatabaseHandler_Hsqldb.java,*
  (package *it.unimo.datariver.querymanager.joinEngine.function*), questa classe eredità dalla classe padre *DatabaseHandler_base.java* e ridefinisce il metodo createTable, affinché lo statement SQL `CREATE TABLE` venga eseguito sul DBMS *HSQLDB*.

- ✓ Classe *WrapperThread.java,*
  (package *it.unimo.datariver.querymanager.joinEngine*), questa classe rappresenta il gestore di un Wrapper. Esegue la local query OQL_I^3 e informa un semaforo quando ha terminato di inserire i dati nella tabella locale.
- ✓ Classe *JoinEngine.java,*
  (package *it.unimo.datariver.querymanager.joinEngine*), questa classe esegue la "mapping query" e "la final query" contenute nel *Query Plan*. Gestisce la comunicazione con i Wrapper e tutte le operazioni di reperimento e manipolazione dei dati.
- ✓ Classe *Oql_SelectExpr.java,*
  (package *it.unimo.datariver.oql*), questa classe contiene dei metodi per ottenere le clausole (select, from, where, order by, group by) della query globale. Le clausole di *select* e *from* in una query sono obbligatorie mentre le altre sono opzionali.
- ✓ Classe *Oql_Parser.java,*
  (package *it.unimo.datariver.oql.parser*), questa classe si occupa del parsing delle query.
- ✓ Classe *Oql_QueryPlan.java,*
  (package *it.unimo.datariver.oql.query*), questa classe si occupa della generazione del Query Plan costituito dalle "local query", "mapping query" e "final query".
- ✓ Classe *Oql_ExpandedQueryPlan,*
  (package *it.unimo.datariver.oql.query*), questa classe gestisce le query formulate su più classi globali (Multiple class query), vengono generati tanti query plan, uno per ogni classe globale coinvolta.

In seguito verrà descritto il processo di esecuzione di una query, vedremo come e quando vengono create le tabelle globali e locali sul DBMS di supporto; in tali tabelle verranno inserite le tuple provenienti dalle sorgenti. Verrà formulata una query su un *GS* ottenuto tramite l'integrazione di tre sorgenti relazionali: "prontocomune", "fibre2fashion" e "usawear". La prima fase dell'integrazione consiste nell'acquisizione delle sorgenti, tramite la configurazione del wrapper specifico per connettersi ad una particolare tipologia di sorgente dati. I dati delle sorgenti in questione sono memorizzati all'interno del *RDBMS Microsoft SQL Server.* Attraverso la GUI di MOMIS, SI-Designer [18], viene costruito il Common Thesaurus e lo schema globale,

quest'ultimo è costituito da quattro classi globali: *Address, Category, SubCategory* e *Company* (Tabella 2)**.**

GS	**Local Classes**		
Global Classes	Sorgente relazionale **prontocomune**	Sorgente relazionale **fibre2fashion**	Sorgente relazionale **usawear**
Address	**Indirizzo**		
Code	Codice		
Region	Regione		
Street	Via		
Town	Comune		
ZipCode	CAP		
Category	**Categoria**	**Category**	
CategoryCode	CodiceCategoria	CategoryCode	
Description	Descrizione	Description	
SubCategory		SubCategory	
SubCategory		**SubCategory**	
Description		Description	
SubCategoryCode		SubCategoryCode	
Company	**Azienda**	**Company**	**Company**
Name	Nome	Name	CompanyName
Description		AboutUs	Description
Category	Categoria	Category	
Phone	Telefono	Tel	Phone

Tabella 2. Schema globale e sorgenti locali

Per eseguire una query si deve istanziare un oggetto Query Manager (classe *QueryManagerImpl.java*) a cui viene passato come parametro lo schema globale. In seguito vengono create sul DBMS di supporto le seguenti tabelle globali e le tabelle locali (ID è un identificatore del Join Engine).

Tabelle globali	***Tabelle locali***
`ID_Address`	`ID_Address_prontocomune_Indirizzo`
`ID_Category`	`ID_Category_prontocomune_Categoria`
`ID_SubCategory`	`ID_Category_fibre2fashion_CategorY`
`ID_Company`	`ID_SubCategory_fibre2fashion_SubCategory`
	`ID_prontocomune_Azienda`
	`ID_Company_fibre2fashion_Company`
	`ID_Company_usawear_Company`

Sullo schema globale viene formulata la seguente query (Multiple class query) (Figura 7):

Query: "*Selezionare per ogni Category con codice 1 e descrizione contenente la stringa "Import", la descrizione della SubCategory corrispondente*"

```
SELECT C.Description, C.CategoryCode, S.Description
FROM Category AS C, SubCategory AS S
WHERE  C.Description Like '%Import%'
AND C.CategoryCode = '1'
AND  C.SubCategory = S.SubCategoryCode
```

Viene eseguito il parsing della query, ovvero nella classe *Oql_Parser.java* avviene il parsing delle clausole *select*, *from* e *where*. Essendo che nella clausola *where* sono presenti due classi globali allora vengono generate due global query (Single class query), ciascuna delle quali riferita ad un'unica classe globale:

```
1-Global Class – Category – ScqG1
SELECT C.Description , C.CategoryCode , C.SubCategory
FROM Category AS C
WHERE C.Description like '%Import%'
AND C.CategoryCode =  '1'

2-Global class – SubCategory – ScqG2
SELECT S.SubCategoryCode, S.Description
FROM SubCategory AS S
```

Quindi viene creato un oggetto *Oql_ExpandedQueryPlan* a cui è passato il vettore contenente le query globali. Nella classe *Oql_ExpandedQueryPlan.java* vengono creati tanti *JoinEngine* quante sono le classi globali nella query, con riferimento alla nostra query di esempio vengono quindi create le tabelle mostrate in precedenza, con `ID=je1 e ID=je2.`

Per ogni query globale relativa ad un'unica classe globale verrà generato il *Query Plan* costituito dalle local query, la mapping query e la final query.

1-Query Plan (Single class query - Category)

Local Query:

```
1-Source "prontocomune"
Local Query on "prontocomune.Categoria" – LQ1:
SELECT Categoria.CodiceCategoria, Categoria.Descrizione
FROM Categoria
WHERE (Descrizione) LIKE ('%Import%')
AND (CodiceCategoria) = ('1')
Table Name: je1_Category_prontocomune_Categoria.

2-Source "fibre2fashion"
Local Query on "fibre2fashion.Category" – LQ2:
SELECT Category.SubCategory, Category.CategoryCode,
Category.Description
FROM Category
WHERE (Description) LIKE ('%Import%')
AND (CategoryCode) = ('1')
Table Name: je1_Category_fibre2fashion_Category.
```

Mapping Query

```
SELECT je1_Category_fibre2fashion_Category.SubCategory AS
SubCategory_1,
je1_Category_fibre2fashion_Category.CategoryCode AS
CategoryCode_1,
je1_Category_prontocomune_Categoria.CodiceCategoria AS
CategoryCode_2,
je1_Category_fibre2fashion_Category.Description AS
Description_1, je1_Category_prontocomune_Categoria.Descrizione
AS Description_2 from je1_Category_prontocomune_Categoria full
outer join je1_Category_fibre2fashion_Category on
(((je1_Category_fibre2fashion_Category.CategoryCode) =
(je1_Category_prontocomune_Categoria.CodiceCategoria))),
Table Name: je1_Category.
```

Final Query

```
SELECT SubCategory, Description, CategoryCode
FROM je1_Category
```

Vengono create la seguenti viste:

- ```
  Create View je1_ResultView_TMP
  AS SELECT SubCategory, Description, CategoryCode
  FROM je1_Category
  ```
- ```
  CREATE VIEW je1_ResultView
  AS SELECT DISTINCT * FROM je1_ResultView_TMP
  ```

2-Query Plan (Single class query - SubCategory)

Local Query:

```
1-Source "fibre2fashion"
Local Query on "fibre2fashion.SubCategory" – LQ1:
SELECT SubCategory.SubCategoryCode, SubCategory.Description
FROM SubCategory
Table Name: je2_SubCategory_fibre2fashion_SubCategory.
```

Mapping Query

```
SELECT
je2_SubCategory_fibre2fashion_SubCategory.SubCategoryCode AS
SubCategoryCode_1,
je2_SubCategory_fibre2fashion_SubCategory.Description AS
Description_1
FROM je2_SubCategory_fibre2fashion_SubCategory,
Table Name: je2_SubCategory.
```

Final Query

```
SELECT SubCategoryCode, Description
FROM je2_SubCategory
```

Vengono create la seguenti viste:

- `CREATE VIEW je2_ResultView_TMP`
  `AS SELECT SubCategoryCode, Description`
  `FROM je2_SubCategory`
- `CREATE VIEW je2_ResultView`
  `AS SELECT DISTINCT * FROM je2_ResultView_TMP`

Infine i risultati vengono fusi insieme applicando la clausola di join e vengono create le seguenti viste:

- `CREATE VIEW Final_View_TMP AS SELECT C.Description AS Description_Category , C.CategoryCode AS CategoryCode_Category ,S.Description AS Description_SubCategory FROM je1_ResultView as C , je2_ResultView as S WHERE (C.SubCategory = S.SubCategoryCode )`
- `CREATE VIEW Final_View`
  `AS SELECT DISTINCT *`
  `FROM Final_View_TMP`

All'utente vengono restituite le tuple contenute nella `Final_View`.

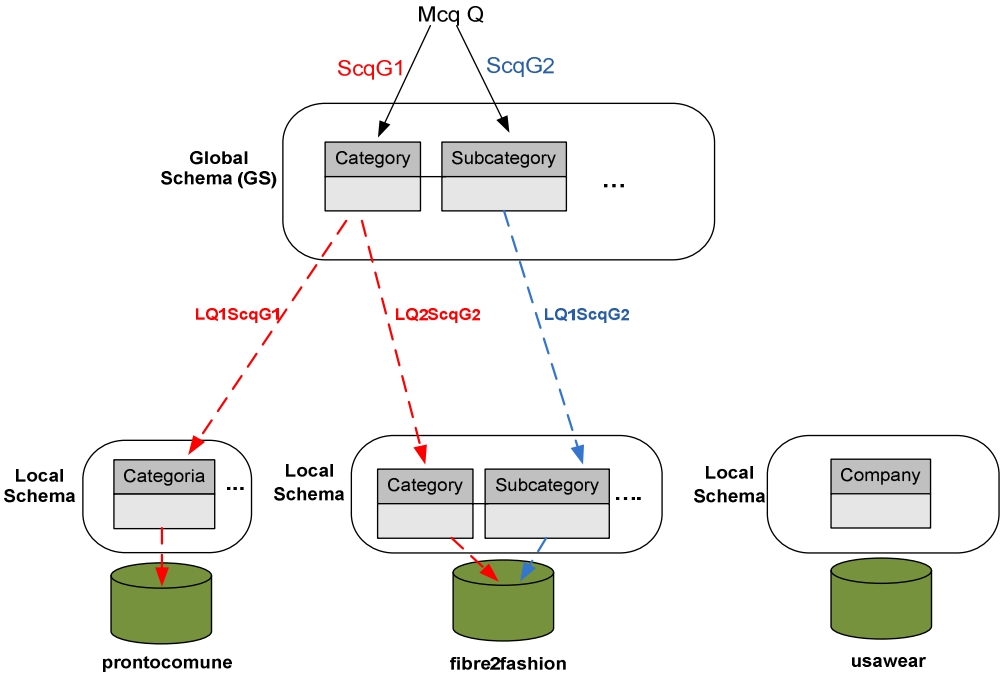

Figura 7. Query Unfolding - Multiple class query

Nella prima implementazione del sistema *MOMIS* è stato scelto come DBMS di supporto *Microsoft SQL Server*, un DBMS relazionale, ad alte prestazioni, prodotto da Microsoft ed utilizzabile solo in una piattaforma Windows.
Essendo *MOMIS* un progetto *Open Source* nasce la necessità di utilizzare come DBMS di appoggio non una soluzione proprietaria, ma una soluzione *Open Source*. La scelta è caduta su un RDBMS Open Source, completamente scritto in Java, *HSQLDB (Hyper Structured Query Language DataBase)* [23], che verrà descritto in modo dettagliato in seguito.

3.2 Descrizione di HSQLDB

HSQLDB (*Hyper Structured Query Language DataBase*) [23] nasce nel 2001 dal Hypersonic SQL Project, è un DBMS relazionale Open Source (viene distribuito con licenza BSD license[1]), completamente scritto in Java (quindi completamente portabile). HSQLDB è conforme allo standard SQL92, inoltre supporta alcuni statement SQL previsti nello standard SQL2008. HSQLDB si può scaricare direttamente dal sito della Sourceforge.net[2] (sito in cui si possono scaricare prodotti Open Source e si può partecipare allo sviluppo di progetti Open Source), si presenta come un jar package (hsqldb.jar); per eseguirlo è necessario avere già installato sulla macchina JRE (Java Runtime Environment) oppure JDK (Java Development Kit).
I componenti del HSQLDB jar package sono:

- RDBMS engine (HSQLDB);
- JDBC Driver;
- Database Manager (GUI di amministrazione (Figura 8));
- SQL tool (command line tool per effettuare interrogazioni in SQL).

L'unico requisito richiesto per l'utilizzo è che il file hsqldb.jar sia nel classpath dell'applicazione chiamante.

1 http://www.opensource.org/licenses/bsd-license.html

2 http://sourceforge.net/projects/hsqldb/files/

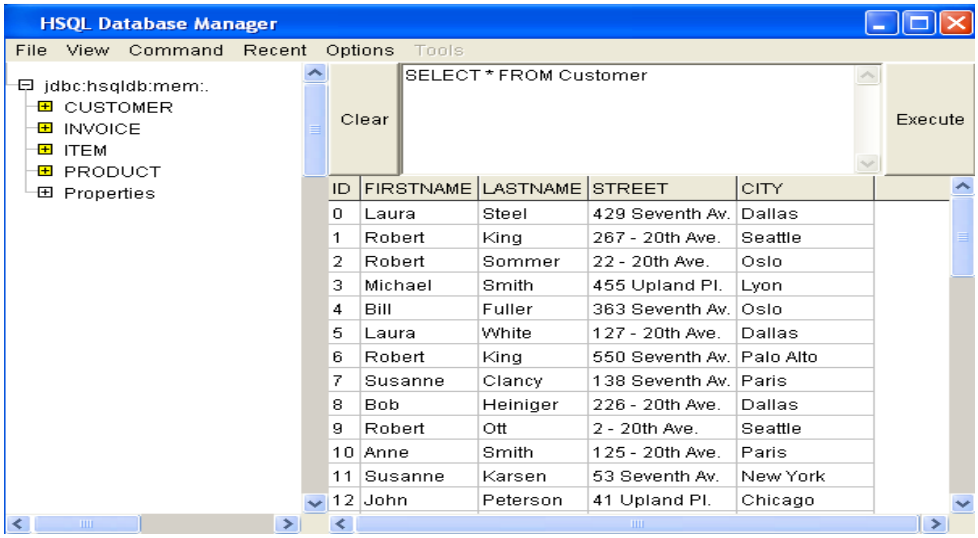

Figura 8. HSQLDB Database Manager

Data Catalog (DataBase)

A seconda della modalità di memorizzazione dei dati (in memoria o su disco), in HSQLDB si possono creare tre tipi di database (data catalog):

- *mem*: i dati vengono salvati nella RAM;
- *file*: i dati vengono salvati nel file system;
- *res*: i dati vengono salvati in una Java resource, es.un file JAR, questi database sono a solo lettura e possono essere distribuiti come parte di un applicazione Java.

In un *mem data catalog* tutti i dati vengono salvati nella memoria volatile quindi non viene creato nessun file. Mentre un *file* data catalog è costituito da un insieme di file (da 2 a 5) che portano tutti lo stesso nome (quello del database), ma differiscono per l'estensione:

- **.properties*, informazioni generali sulla configurazione del database;
- **.script*, contiene le descrizione degli oggetti del database (es. gli statement SQL `Create Table` per la creazione delle tabelle che costituiscono il database), ed eventualmente i dati delle tabelle non in cache;
- **.log*, log delle modifiche recenti del database. In questo file vengono memorizzate le modifiche effettuate sui dati;
- **.data*, contiene i dati delle tabelle in cache;
- **.backup*, file compresso dell'ultimo stato consistente del database.

Un sesto file con estensione .lck è utilizzato per indicare che il database è attualmente "aperto".

Server e In-Process Mode

HSQLDB può essere utilizzato sia in modalità server (Server Mode) e sia come istanza interna di un'applicazione (In-Process (Standalone) Mode). Quest'ultima modalità esegue il database engine come parte dell'applicazione, nella stessa JVM (Java Virtual Machine), lo svantaggio è che non ci si può connettere al database dall'esterno dell'applicazione, di conseguenza non si può controllare il contenuto del database quando l'applicazione è in esecuzione, ma per molte applicazioni questa è la modalità con prestazioni più alte dal fatto che i dati non devono essere convertiti ed inviati attraverso la rete. In generale è preferibile sviluppare un'applicazione utilizzando la modalità server e poi passare alla modalità In-Process solo alla fine dello sviluppo. Nella modalità Server Mode il database engine in esecuzione (su una JVM), rimane in attesa di richieste di connessione che possono pervenire da applicazioni (client) in esecuzione sulla stessa macchina oppure su altre macchine connesse tramite la rete al server. Le applicazioni si connettono al server utilizzando il driver JDBC. Esistono tre modalità server:

- *Hsqldb Server*: questa è la modalità più utilizzata per eseguire il database server ed è la modalità che assicura prestazioni più alte rispetto alle altre modalità server. Il comando per lanciare il server con un database di default (mydb) è riportato in seguito:

  ```
  java -cp ../lib/hsqldb.jar org.hsqldb.Server -database.0
  file:mydb -dbname.0 xdb
  ```

Codice Java per connettersi al server HSQLDB in esecuzione sul localhost:

```
  try {
        Class.forName("org.hsqldb.jdbcDriver" );
    } catch (Exception e) {
        System.out.println("ERROR: failed to load HSQLDB JDBC driver.");
        e.printStackTrace();
        return;
    }
Connection c =
DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "sa", "");
```

- *Hsqldb Web Server*: HSqldb Web Server è un web server che consente ai client JDBC di connettersi attraverso il protocollo HTTP.
  Per lanciare un HSQLDB web server si deve eseguire il seguente comando:
  ```
  java -cp ../lib/hsqldb.jar org.hsqldb.WebServer -database.0
  file:mydb -dbname.0 xdb
  ```
- *Hsqldb Servlet*: Utilizza lo stesso protocollo della modalità Web Server, questa modalità viene usata quando un servlet engine separato e indipendente (o application server) (come ad esempio Tomcat) fornisce accesso al database. Per connettersi al database sia in questa modalità, e sia in quella Web Server bisogna utilizzare il driver JDBC lato client.

Una volta terminato l'utilizzo del database bisogna effettuare lo shutdown del server HSQLDB attraverso l'esecuzione del comando SHUTDOWN.

Tabelle

In HSQLDB, in accordo con lo standard SQL, esistono 2 tipologie di tabelle: Temporary Tables (tabelle temporanee) e Persistent Tables (tabelle persistenti). A seconda di come i dati vengono memorizzati, le Persistent Tables vengono classificate in: Memory Tables, Cached Tables e Text Tables. Memory Tables sono le tabelle di default, vengono create tramite lo statement SQL `CREATE TABLE`, i dati inseriti in queste tabelle vengono mantenuti in memoria, ogni cambiamento della struttura o del contenuto viene scritto nei file *.log e *.script. Al prossimo start up del database avviene la lettura dei file *.log e *.script, le Memory Tables vengono ricreate e viene effettuato l'insert di tutte le tuple che contenevano le tabelle, quindi le Memory Tables diversamente dalle Temporary Tables sono persistenti. Le tabelle temporanee che vengono create in MOMIS durante il query processing sono create con il comando `CREATE TABLE`, quindi sono Memory Tables. Lo svantaggio di questi tipi di tabelle è che il processo di ricreazione e di inserimento delle tuple quando si riaccede al database può richiedere un intervallo di tempo importante, specialmente in caso di database di grandi dimensioni (> 10MB).

Attraverso lo statement `CREATE CACHED TABLE` vengono create le Cached Tables, in queste tabelle solo una parte dei dati e degli indici è mantenuto in memoria, questo consente la creazione di tabelle di grandi dimensioni che altrimenti occuperebbero molta memoria. Lo start up del database in caso di cached tables

richiede meno tempo rispetto alle memory tables ma d'altra parte essendo che solo una parte dei dati è mantenuto in memoria c'è una riduzione nelle performance. Le Text Table utilizzano un CSV (Comma Separated Value) come sorgente dei loro dati, inoltre esiste la possibilità di specificare un file vuoto su cui il database engine inserirà i dati. Nei database di tipo *mem* non si possono creare dei Text Table.

3.3 HSQLDB vs SQL Server

Nella prima implementazione del Query Manager è stato scelto come DBMS di supporto Microsoft SQL Server, un RDBMS prodotto da Microsoft ad alte prestazioni, progettato per gestire altissimi volumi di operazioni transazionali in ambiente multiutente. Però dall'altra parte presenta degli svantaggi: per poterlo utilizzare all'interno di una applicazione è necessaria una licenza commerciale ed inoltre questo RDBMS è compatibile solo con la piattaforma Windows. Dopo avere installato MOMIS, per eseguire delle query su un GS, generato tramite il Global Schema Builder, è necessario installare Microsoft SQL Server e creare un database definito nel file di configurazione di MOMIS, sul quale verranno create le tabelle temporanee necessarie per l'elaborazione delle query. Per questi motivi è stato scelto come nuovo DBMS di supporto al query processing, HSQLDB, un RDBMS Open Source e completamente portabile perché scritto in Java. Il database engine HSQLDB viene integrato nell'applicazione MOMIS quindi non è necessario installarlo a parte. Inoltre, diversamente dagli altri inner database engine (es. Derby, H2) Open Source implementati in Java, HSQLDB supporta l'operatore *Full Outer Join*, operatore che viene utilizzato nella Mapping Query.
Il cambio del DBMS di supporto ha reso necessario il testing del modulo Query Manager.

3.4 Testing

Per testare il funzionamento del Query Manager (con DBMS di supporto HSQLDB) sono state formulate delle query sul *GS*, le cui classi globali e sorgenti locali integrate sono riportate in Tabella 2. Per ciascuna query eseguita viene riportato l'esito dell'esecuzione, l'eventuale errore emerso e le modifiche effettuate

sul codice. Il testing del Query Manager ci ha permesso di definire la sintassi OQL_I[3] accettata dal Query Manager del sistema MOMIS (Appendice A).

Query 1: *"Selezionare gli indirizzi della regione Emilia Romagna con codice maggiore di 20 e zip code pari a 42015 "*

```
SELECT Code, Street, Town, Region, ZipCode
FROM Address
WHERE Region = 'Emilia Romagna'
AND Code > 20
AND ZipCode = '42015'
```

L'esecuzione della prima query evidenzia il primo problema. Infatti la query non viene eseguita e viene restituito il seguente errore `"user lacks privilege or object not found"`. HSQLDB è un DBMS case sensitive ed inoltre all'atto della creazione della tabella converte tutti gli identificatori in maiuscolo. Nel manuale di HSQLDB è riportata la sintassi della `CREATE TABLE` e viene specificato che tutti gli identificatori (nomi di tabelle, colonne, vincoli o indici) se non inseriti tra doppi apici vengono convertiti automaticamente in maiuscolo. Un importante implicazione della conversione è quando si vuole accedere agli attributi della tabella tramite JDBC DatabaseMetaData (bisogna usare i nomi degli attributi in maiuscolo e non come sono stati specificati nella `CREATE TABLE`). Mentre, se nella `CREATE TABLE` gli identificatori vengono inseriti tra doppi apici, non verranno convertiti in maiuscolo ma quando si accederà agli attributi bisognerà usare il nome dell'attributo come è stato specificato nello statement della `CREATE TABLE` incluso i doppi apici es.

```
 Create Table "Studente"(
"Name" varchar(20),
"Matricola" varchar(20),
"Corso" varchar(20))
```

La query sulla seguente tabella deve avere la seguente espressione:

Select "Name", "Corso" FROM "Studente"

Per un esecuzione corretta della query nella classe *JoinEngine.java*, package *it.unimo.datariver.querymanager.joinEngine* , viene modificato il metodo *initJE()*, che inizializza il join engine, vengono convertiti in maiuscolo i nomi degli attributi globali e locali.

SQL Server

```
for (int j=0; j<gaA.length; j++) {
                GlobalSimpleAttribute gattribute =
(GlobalSimpleAttribute)gaA[j];
//colonne è un vettore che conterrà i nomi degli attributi
              colonne.add(gattribute.getName());
                tipi.add(gattribute.getType());
                domains_precision.add(new Integer(-1));
     if (this.debug) out.println("  Global Simple Attribute [" + j + "] = "
                    + gattribute.getName());


if(!t.colonne.contains(nomeattr)){
                        t.colonne.add(nomeattr);
                        t.tipi.add(tipoattr);
                        t.domains_precision.add(domain_precision);
                    }
                    tabelleLocali.put(localClass, t);
```

HSQLDB

```
for (int j=0; j<gaA.length; j++) {
                GlobalSimpleAttribute gattribute =
(GlobalSimpleAttribute)gaA[j];
                  colonne.add(gattribute.getName().toUpperCase());
                tipi.add(gattribute.getType());
                domains_precision.add(new Integer(-1));
     if (this.debug) out.println("  Global Simple Attribute [" + j + "] = "
                    + gattribute.getName());

if(!t.colonne.contains(nomeattr)){
                        t.colonne.add(nomeattr.toUpperCase());
                        t.tipi.add(tipoattr);
                        t.domains_precision.add(domain_precision);
                    }
                    tabelleLocali.put(localClass, t);
```

Dopo la modifica effettuata la query viene eseguita correttamente.

Query 2: *“Selezionare le categorie che hanno nella descrizione la stringa “importer” ”*

```
SELECT B.CategoryCode, B.Description, B.SubCategory
FROM Category AS B
WHERE B.Description LIKE '%importer%'
```

La query viene eseguita correttamente.

Query 3: *"Selezionare le categorie che hanno nella descrizione la stringa "Export" e hanno un codice di SubCategory superiore a 10 "*

```
SELECT C.CategoryCode, C.Description
FROM Category AS C
WHERE C.Description LIKE '%Export% '
AND C.SubCategory > 10
```

La query viene eseguita correttamente.

Query 4: *"Selezionare, per ogni company con nome contenente la stringa "im", descrizione contenente la stringa "stocklot" e attributo EMail non specificato, il nome, la categoria, l'email e la descrizione "*

```
SELECT DISTINCT A.Name, A.Category, A.EMail, A.Description
FROM Company AS A
WHERE A.Description LIKE '%stocklot%'
AND A.Name LIKE '%im%'
AND IS NULL A.Phone
```

L'esecuzione della query non va a buon fine e viene visualizzato il seguente errore `"Atribute not find: Phoneis not present in environment"` infatti l'attributo Phoneis non è un attributo della tabella, è stata effettuata una piccola modifica nella classe *Oql_IsNullExpr.java*, package *it.unimo.datariver.oql*, nel metodo *toStringWithNoAlias()* che riscrive la clausola `is null A.Phone` in `A.Phone is null`. Dopo la modifica la query viene eseguita correttamente.

Query 5: *"Selezionare le company che appartengono ad una categoria maggiore di 100 e che hanno la stringa "RAM" nel loro nome "*

```
SELECT A.Name, A.Category, A.Description, A.Phone, A.EMail
FROM Company AS A
WHERE A.Name LIKE  '%RAM% '
AND A.Category > 100
```

L'esecuzione della query non va a buon fine e viene visualizzato il seguente errore `"incompatible data types in combination"`. Il problema deriva dal fatto che l'attributo globale Category (short) della classe globale Company (Tabella 3) viene mappato in due attributi di tipo diverso prontocomune.Azienda.Categoria (string) e

fibre2fashion.Company.Category (short). HSQLDB diversamente da SQL Server (l'esecuzione della query nel Query Manager che si appoggia a SQL Server non genera errori e la query viene eseguita correttamente) non effettua conversione implicita dei dati. Il problema è generato dall'assenza di una funzione di conversione (Data Conversione Function) StringToInt sull'attributo locale Categoria della sorgente prontocomune.Azienda.

Company	**prontocomune.Azienda**	**fibre2fashion.Company**	**usawear.Company**
Name	Nome	Name	CompanyName
Description		AboutUs	Description
Category (short)	MTF[Category][prontocomune.Azienda] = StrToInt(Categoria)	Category(short)	
Phone	Telefono	Tel	Phone
EMail	Email	EMail	Email

Tabella 3. Mapping Table - Company

Query 6: *"Ordinare in senso discendente rispetto a Description, le categorie che hanno l'attributo Description specificato e codice di SubCategory diverso da 11 "*

```
SELECT C.CategoryCode, C.Description, C.SubCategory
FROM Category AS C
WHERE C.SubCategory != 11
AND NOT IS NULL C.Description
ORDER BY C.Description DESC
```

L'esecuzione della query non va a buon fine e viene visualizzato il seguente errore `"Error during query execution"`. Nel processo di Query Unfolding la clausola *Order By* non viene eseguita direttamente sulle sorgenti ma costituisce una clausola residua che verrà inserita nella "final query". In presenza di clausola *Order By* nella query globale, essendo che in SQL Server non si può includere la clausola Order By nella creazione di una vista, nella select statement della final query viene inclusa la clausola "TOP 100 PERCENT" che rende possibile l'inclusione della clausola di ordinamento nella creazione di una vista. In HSQLDB la clausola "TOP 100 PERCENT" non è supportata e questo genera l'errore. Lo statement SQL per la creazione delle viste in HSQLDB prevede la possibilità di includere la clausola Order By e di conseguenza creare una vista già ordinata:

```
CREATE VIEW <viewname>[(<viewcolumn>,..) AS SELECT ... FROM
... [WHERE Expression]
[ORDER BY orderExpression [, ...]]
[LIMIT <limit> [OFFSET <offset>]]
```

Nella classe *Oql_SelectExpr.java* del package *it.unimo.datariver.oql* vengono modificati i metodi *toString()* e *getFinalQuery()*.

SQL Server

```
if (this.getOrderByClause() != null && !
this.getOrderByClause().equals(""))
    s = s + " TOP 100 PERCENT ";


  String orderBy;
      orderBy = getFinalQuery_OrderBy(localIntefaces, jen);
      if (orderBy != null && ! orderBy.equals("")){
      sql = sql + "TOP 100 PERCENT ";
```

HSQLDB

```
if (this.getOrderByClause() != null && !
this.getOrderByClause().equals("")){
      s = s ;

String orderBy;
      orderBy = getFinalQuery_OrderBy(localIntefaces, jen);
      if (orderBy != null && ! orderBy.equals("")){
      sql = sql;
```

Query 7: *"Selezionare, per ogni categoria con descrizione contenente la stringa "export" e codice di SubCategory minore di 22, la descrizione della SubCategory"*

```
SELECT D.Description
FROM Category AS C, SubCategory AS D
WHERE C.Description LIKE '%export%'
AND C.SubCategory < 22
AND C.SubCategory = D.SubCategoryCode
```

La query viene eseguita correttamente.

Query 8: *"Selezionare, per ogni company con nome contenente la stringa "Importer" e categoria di appartenenza uno, il nome, la descrizione della categoria e il codice di SubCategory corrispondente "*

```
SELECT A.Name, C.Description, C.SubCategory
FROM Company AS A , Category AS C
WHERE A.Name Like '%Importer%'
```

```
AND C.CategoryCode = '1'
AND A.Category = C.CategoryCode
```

L'esecuzione della query non va a buon fine e viene visualizzato il seguente errore "`incompatible data types in combination`". Nella fase di Query Unfolding il predicato di join viene incluso nella final query. L'errore visualizzato deriva dalla differenza dei tipi di dato degli attributi globali Category della classe Company e CategoryCode della classe globale Category (rispettivamente short e string). In SQL Server è possibile eseguire il join tra due tabelle anche se i data type, degli attributi su cui è espresso il predicato di join, sono diversi. SQL Server converte implicitamente il tipo degli attributi di join, mentre in HSQLDB non è possibile eseguire il join tra due tabelle se il predicato di join viene espresso su due attributi che hanno tipo di dato diverso. In MOMIS, durante la fase di Mapping Refinement, è possibile cambiare il tipi di dato di un attributo globale. Il problema si risolve cambiando il data type di uno dei due attributi globali (es. il tipo di dato dell'attributo globale Category della classe Category viene cambiato in short).

Category	**prontocomune.Categoria**	**fibre2fashion.Category**
CategoryCode(string⇒short)	CodiceCategoria	CategoryCode
Description	Descrizione	Description
SubCategory		SubCategory

Tabella 4. Mapping Table - Category

Query 9: *"Selezionare, per ogni company il cui nome contiene la stringa "RAM" e la descrizione della categoria di appartenenza contiene la stringa "Importer", il nome, il telefono, la descrizione della categoria, il codice di SubCategory e la descrizione della SubCategory "*

```
SELECT A.Name, A.Phone, C.Description, B.SubCategory,
C.Description
FROM Company AS A, Category AS B, SubCategory AS C
WHERE A.Name LIKE '%RAM%'
AND B.Description LIKE '%importer%'
AND A.Category = B.CategoryCode
AND B.SubCategory = C.SubCategoryCode
```

La query viene eseguita correttamente.

Query 10: *"Selezionare, per ogni company con nome che inizia con la lettera i, con descrizione della categoria di appartenenza contenente la stringa "Importer", codice di categoria uguale a 71 e codice di SubCategory minore di 10, il nome, il telefono, la pagina web, la descrizione della categoria, il codice di SubCategory e la descrizione della SubCategory"*

```
SELECT A.Name, A.Phone, B.Description, A.Web, A.Category,
C.SubCategoryCode, C.Description
FROM Company AS A, Category AS B, SubCategory AS C
WHERE A.Description like 'i%'
AND B.Description LIKE '%importer%'
AND B.CategoryCode = '71'
AND C.SubCategoryCode < 10
AND A.Category = B.CategoryCode
AND B.SubCategory = C.SubCategoryCode
ORDER BY A.Name DESC
```

La query viene eseguita correttamente.

Query 11: *Selezionare le company che hanno un codice di SubCategory uguale a 2"*

```
SELECT A.Name, A.HomePage, A.Fax, A.EMail, A.Description,
A.Category
FROM Company AS A, Category AS B
WHERE  B.SubCategory = 2
AND A.Category = B.CategoryCode
```

L'esecuzione della query genera il seguente errore `"ParseException"`. La query eseguita sul Query Manager che utilizza come DBMS di supporto SQL Server genera lo stesso errore. Inoltre lo stesso errore viene generato se nel predicato di confronto viene utilizzato un intero compreso tra 1-9. Prima di riportare la soluzione dell'errore diamo una breve descrizione del JavaCC[1] (Java Compiler Compiler) che viene utilizzato, per generare le classi del package it.unimo.datariver.oql.parser, cioè per generare le classi che effettueranno il parsing della query OQL_I[3] inserita.

1 http://javacc.dev.java.net/

JavaCC

Il JavaCC partendo da una specifica di una grammatica (con azioni semantiche), genera le classi Java che realizzano un analizzatore sintattico top-down per tale grammatica. Il JavaCC prende in input un file testuale con estensione .jj dove si definisce la grammatica del parser e genera le seguenti classi:

classi generiche	***classi specifiche***
SimpleCharStream.java	nome_parser.java
Token.java	nome_parserConstants.java
ParseException.java	nome_parserTokenManager.java
TokenMgrError.java	

Il file con estesione .jj è suddiviso in tre sezioni:

1. lista di opzioni per JavaCC;
2. unità di compilazione Java che inizia con PARSER_BEGIN(nome_parser) e termina con PARSER_END(nome_parser);
3. lista di regole di regole lessicali e sintattiche (ovvero produzioni poste essenzialmente in grammatica EBNF)

Inoltre si possono aggiungere azioni semantiche, ovvero, porzioni di codice Java eseguite al momento dell'espansione delle produzioni.

Struttura di un file per JavaCC

Il file della grammatica inizia con una lista di opzioni (questa lista può non essere presente). Poi è seguito dalla Java compilation unit racchiusa tra le stringhe PARSER_BEGIN(nome_parser) e PARSER_END(nome_parser). Alla fine è presente la lista di produzione della grammatica. Il nome che segue PARSER_BEGIN e PARSER_END deve essere lo stesso ed è il nome del parser che verrà generato. Ad es. se la stringa passata a PARSER_BEGIN E PARSER_END è Oql_Parser, verranno generati le seguenti classi java:

- ✓ Oql_Parser.java: il parser, in questa classe sarà incluso un metodo per ciascun simbolo non terminale della grammatica.
- ✓ Oql_ParserTokenManager.java: il token manager (o analizzatore lessicale), analizza il flusso di caratteri in ingresso suddividendolo in porzioni (Token) in base alla specifica lessicale contenuta nel file .jj e collabora con il parser per

creare la sequenza di Token utilizzando le regole di produzione presenti nella specifica JavaCC.

✓ Oql_ParserConstant.java: un insieme di costanti di utilità.

Ci sono quattro tipi produzioni in JavaCC:

- javacode_production e bnf_production sono utilizzate per definire la grammatica a partire dalla quale verrà generato il parser;
- regular_expr_production è usata per definire i token della grammatica. Il token manager è generato a partire da tali informazioni;
- token_manager_decls vengono utilizzate per introdurre dichiarazioni da inserire nel token manager che verrà generato.

Per risolvere il problema del `ParseException` viene eseguita la seguente modifica nel file oql_momis_v.jj che è il file dal quale verranno generati le classi del parser:

```
TOKEN  :
{
   < IDENTIFIER:  ["a"-"z","A"-"Z"] (["a"-"z","A"-"Z","_","0"-"9"])* >
|  < IDENTIFIER_C:      "[" ["a"-"z","A"-"Z"] (["a"-"z","A"-"Z","_","
","0"-"9"])* "]" >
|  < DIGIT: ["0"-"9"] >
| < CHAR_C:       ["a"-"z","A"-"Z","_","*","%"," ","?","0"-"9"] >
|  < STRING_C:          ( "'"  (["a"-"z","A"-"Z","0"-"9","_","*","%","
","?","@","!","£","$","&","/","(",")","=","|","^","ì","\\","è","é","[","]",
"{","}","ò","ç","°","à","#","ù","§","-",",",";",".",":","+","*"])* "'"
| < INTEGER_NUM:  ["0"-"9"] (["0"-"9"])*>
```

Modificato

```
TOKEN  :
{
   < IDENTIFIER:  ["a"-"z","A"-"Z"] (["a"-"z","A"-"Z","_","0"-"9"])* >
|  < IDENTIFIER_C:      "[" ["a"-"z","A"-"Z"] (["a"-"z","A"-"Z","_","
","0"-"9"])* "]" >
|  < STRING_C:          ( "'"  (["a"-"z","A"-"Z","0"-"9","_","*","%","
","?","@","!","£","$","&","/","(",")","=","|","^","ì","\\","è","é","[","]",
"{","}","ò","ç","°","à","#","ù","§","-",",",";",".",":","+","*"])* "'"
| < INTEGER_NUM:  (["0"-"9"])+>
```

Vengono eliminati i token `<DIGIT>, <CHAR_C>` essendo che il digit è un caso particolare di Integer e char è un caso particolare di String. Dopo avere modificato il file, vengono cancellate le classi precedentemente generate e viene utilizzato JavaCC al quale viene passato in input il file modificato. JavaCC genera nuovamente le classi relative al parser. L'esecuzione della query non genera più errori e la query viene eseguita correttamente.

Sul query manager sono state eseguite più di 40 query e questo ha permesso di verificare che la sintassi OQL_I^3 accettata dal Query Manager sia la stessa riportata in Appendice A.

Capitolo 4

4 Analisi e testing del Query Unfolding nel sistema MOMIS

4.1 Il Data Set per il testing del Query Unfolding

In un sistema di integrazione dati, due sono gli aspetti cruciali: la modellazione dei mapping e la generazione del *GS*. Per quanto riguarda la modellazione dei mapping tra le sorgenti dati, in letteratura sono stati proposti due approcci*: Local-As-View (LAV)* e *Global-As-View (GAV).* In *MOMIS* viene adottato l'approccio *GAV* [28], nel quale le classi dello schema globale sono definite come viste sulle sorgenti locali, cioè ogni classe globale è definita da una query sulle classi locali. Il vantaggio di questo approccio è la gestione del query processing, la query formulata sullo schema globale viene riscritta in termini degli schemi delle classi locali coinvolte, *Query Unfolding*. In seguito verrà descritto il data set, cioè l'insieme delle sorgenti dati, utilizzato per testare il query unfolding nel sistema MOMIS.
Vengono create le seguenti tre sorgenti relazionali:

1. QMTestDB1 (database relazionale)

```
CREATE TABLE L1(
First_Name varchar(50),
Last_Name varchar(50),
Email varchar(50),
Year integer);

INSERT INTO L1(First_Name, Last_Name, Email, Year)
values('Marco','Rossi','rossi@unimore.it',34);
values('Luca','Verdi','verdi@unimore.it',22);
values('Franco','Neri','neri@unimore.it',35);
values('Mario','Bianchi','bianchi@unimore.it', null);
```

2. QMTestDB2 (database relazionale)

```
CREATE TABLE L2(
name varchar(50),
mail varchar(50),
section integer,
dept integer);

INSERT INTO L2(name, mail, section, dept)
values('Marco Rossi', 'rossi@unimore.it, 1, 1);
values('Luca Verdi', 'verdi@unimore.it', 2, 2);
values('Maria Verdi', 'verdi@gmail.it', 1, 3 );
values('Joe Black', 'black@unimore.it, 2, 2);
```

3. QMTestDB3 (database relazionale)

```
CREATE TABLE L3(
Name varchar(50),
Email varchar(50));

 INSERT INTO L3(Name,Email)
values('Marco Rossi','rossi@unimore.it');
values('Franco Neri','neri@unimore.it');
values('Maria Verdi','verdi@gmail.it');
values('Jack Red','red@unimore.it');
```

La prima fase dell'integrazione delle sorgenti consiste nell'acquisizione delle stesse tramite la configurazione del wrapper specifico, per connettersi ad una particolare tipologia di sorgente dati. I dati delle sorgenti sopra riportate sono stati memorizzati all'interno del *RDBMS Microsoft SQL Server.* Attraverso la GUI di MOMIS, SI-Designer [18], viene costruito il Common Thesaurus e lo schema globale (*GS*), quest'ultimo è costituito da un unica classe globale: *G1* la cui Mapping Table viene riportata in Tabella 5.

G1	*QMTestDB1.L1*	*QMTestDB2.L2*	*QMTestDB3.L3*
email	Email	mail	Email
name	First_Name Last_Name	name	Name
dept		dept	
section		section	
year	Year		

Tabella 5. Mapping Table - G1

Sulla Mapping Table (MT) della classe globale *G1* (Tabella 5) viene effettuato un Mapping Refinement:

- L'attributo globale "*name"* viene mappato in due attributi "First_Name" e "Last_Name" della classe locale *QMTestDB1.L1*;
- Viene applicata su questi attributi locali una Data Conversion Function disponibile in MOMIS, *String Concatenation* (es. 'Marco' + 'Rossi' sarà convertito a livello globale in 'Marco Rossi');
- Viene cambiato il tipo di dato di tutti gli attributi globali in string;
- Join Attribute: "*name"*. Quindi la mapping query q_G corrispondente alla classe globale *G1* avrà la seguente espressione:

```
QMTestDB2.L2 full outer join QMTestDB1.L1 on
(((QMTestDB1.L1.First_Name + ' ' + QMTestDB1.L1.Last_Name) =
(QMTestDB2.L2.name))) full outer join QMTestDB3.L3 on
(((QMTestDB3.L3.Name) = (QMTestDB2.L2.name)) OR
((QMTestDB3.L3.Name) = (QMTestDB1.L1.First_Name + ' ' +
QMTestDB1.L1.Last_Name)))
```

4.2 Il Query Unfolding

Nel paragrafo 2.1.3 è stato descritto ampiamente il query processing nel sistema MOMIS.

Data una query globale Q:

```
Q =  SELECT <Q_select_list>
     FROM G
     WHERE <Q_condition>
```

Il processo di query unfolding è costituito dai seguenti passi:

1. *Generazione delle query locali*:

```
LQ = SELECT <L_select_list>
     FROM L
     WHERE <L_condition>
```

Viene generata una local query per ogni classe $L \in LG$, cioè per ogni classe locale su cui la classe globale *G* è mappata.

2. *Generazione della mapping query* q_G:

I dati restituiti dall'esecuzione delle local query *LQ* sulle sorgenti dati locali, vengono fusi attraverso l'esecuzione della mapping query q_G corrispondente a *G*.

3. *Generazione della final query*:

Attraverso la final query verranno eseguite le eventuali Resolution Functions presenti e le condizioni residue che non è stato possibile eseguire direttamente sulle sorgenti locali.

In seguito descriveremo in modo dettagliato il processo di generazione delle local query e, in modo particolare, la riscrittura della condizione della query globale in termini degli schemi delle sorgenti locali.

Sia $G \in GS$, una classe globale su cui viene formulata la seguente query:

```
Q =  SELECT <select_list>
     FROM G
     WHERE <condition>
```

Sia *GA* l'insieme degli attributi globali di *G*; gli attributi specificati in `<select_list>` sono un sottoinsieme di *GA*. Indichiamo con ***C*** la condizione della query globale *Q*; la condizione è un espressione booleana di predicati atomici positivi, dove ogni predicato atomico positivo può avere la seguente forma (GA_i op value) oppure (GA_i op GA_j), dove GA_i, GA_j sono attributi globali della classe globale *G*.

Come esempio prendiamo la seguente query (con riferimento al data set sopra riportato):

```
SELECT name, email
FROM G1
WHERE name LIKE 'Marco%'
AND (year = '35' OR section = '2')
```

`<select_list>` = {`name`, `email`}

C = `name LIKE 'Marco%' AND (year = '35' OR section = '2')`

La query globale viene riscritta in termini delle classi locali, cioè viene generata una local query LQ_i, $1 \leq i \leq n$ (n è il numero delle classi locali su cui *G* è mappata), per ogni classe locale su cui *G* è mappata. Riferendoci alla query sopra riportata verranno generate tre query locali, essendo tre le classi locali (*QMTestDB1.L1*, *QMTestDB2.L2*, *QMTestDB3.L3*) su cui è mappata la classe globale *G1*. Ogni query viene mandata tramite il wrapper alla sorgente corrispondente; il wrapper tradurrà la query dal linguggio OQL_l^3, nel linguaggio di interrogazione specifico per la sorgente (*TSQL* in questo caso).

I passi necessari per generare le local query LQ_i , $1 \leq i \leq n$, sono:

1. Query normalization

Viene riportata la condizione della query globale*Q* in Disjunctive Normal Form (DNF), la indichiamo con $\boldsymbol{C_{DNF}}$; la condizione della query *Q* in DNF avrà la seguente forma $\boldsymbol{C_{DNF}} = C_1$ or C_2 or … or C_n. Ogni clausola C_i avrà la seguente forma $C_i = P_1$ and P_2 and …. and P_m , congiunzione di predicati atomici (P_j).

Quindi la condizione $\boldsymbol{C_{DNF}}$ della query di esempio sarà uguale:

$\boldsymbol{C_{DNF}} = C_1$ or C_2 :

C_1 = `name LIKE 'Marco%' AND year = '35'`

C_2 = `name LIKE 'Marco%' AND section = '2'`

2. Local query Condition

Per generare le condizioni delle local query LQ_i, ogni predicato atomico P della query globale, la cui condizione è stata riportata in DNF, viene riscritto in maniera tale da essere supportato dalla sorgente locale *L* corrispondente. Se gli attributi globali che compaiono in P sono mappati nella classe locale *L*, cioè gli attributi globali hanno un not null mapping in *L*, e l'operatore op (es. LIKE) che compare nel predicato è supportato da *L* allora il predicato P può essere riscritto in termini della classe locale *L*, e viene denotato con P_L. Se P è un predicato del tipo (*GA* op value) allora P_L = MTF[*GA*][*L*] op value (MTF, la funzione di Data Conversion applicata sugli attributi locali di L su cui l'attributo globale *GA* è mappato, deve essere supportata da *L*). Altrimenti se il predicato P comprende un attributo non mappato in *L*, P_L = true.

Con riferimento alla query sopra riportata prendiamo in considerazione il predicato atomico P = `name LIKE 'Marco%'`; questo predicato verrà riscritto nella local query relativa alla classe locale L1 (sorgente *QMTestDB1*) nel seguente modo:

P_{L1} = `(First_Name + ' ' + Last_Name) LIKE ('Marco%')`

perché l'attributo globale "*name*" è mappato in due attributi locali "First_Name" e "Last_Name" di L1 e su questi attributi viene applicata una *String Concatenation* (Data Conversion Function presente in MOMIS).

Mentre il predicato P = (`section = '2'`), essendo l'attributo globale "*section"* non mappato nella classe locale L1 (sorgente *QMTestDB1*), verrà riscritto nella local query relativa a L1 nel modo seguente:

P_{L1} = *true*

Con riferimento alla query di esempio nella Tabella 6 viene riportata la riscrittura dei predicati per ciascuna sorgente locale.

	name LIKE 'Marco%'	year = '35'	section = '2'
L1	(First_Name+''+Last_Name)LIKE'Marco%'	Year = '35'	true
L2	name LIKE 'Marco%'	true	section = '2'
L3	Name LIKE 'Marco%'	true	true

Tabella 6. Riscrittura dei predicati

La condizione della local query LQ_1 (classe locale *QMTestDB1.L1*) sarà:

```
WHERE ((First_Name+''+Last_Name) LIKE 'Marco%' AND Year ='35')
OR ((First_Name+''+Last_Name) LIKE 'Marco%' AND true)
```

Che per la proprietà di *neutrality* ((A and true) ≡ A) è equivalente a:

```
WHERE ((First_Name+''+Last_Name) LIKE 'Marco%' AND Year ='35')
OR ((First_Name+''+Last_Name) LIKE 'Marco%')
```

La condizione della local query LQ_2 (classe locale *QMTestDB2.L2*) sarà:

```
WHERE ((name) LIKE 'Marco%' AND true)
OR(name LIKE 'Marco%' AND section = '2')
```

Che per la proprietà di *neutrality* ((A and true) ≡ A) è equivalente a:

```
WHERE (name LIKE 'Marco%')
OR(name LIKE 'Marco%' AND section = '2')
```

La condizione della local query LQ_3 (classe locale *QMTestDB3.L3*) sarà:

```
WHERE (Name LIKE 'Marco%' AND true)
OR (Name LIKE 'Marco%' AND true)
```

Che per la proprietà di *neutrality* ((A and true) ≡ A) è equivalente a:

```
WHERE (Name LIKE 'Marco%') OR (Name LIKE 'Marco%')
```

3. Residual Condition

Il calcolo della condizione residua, $\boldsymbol{C_r}$, avviene attraverso i seguenti tre passi:

a. Viene riportata la condizione della query globale *Q* in Conjunctive Normal Form (CNF), la indichiamo con $\boldsymbol{C_{CNF}}$; la condizione della query *Q* in CNF avrà la seguente forma $\boldsymbol{C_{CNF}} = C_1$ and C_2 and … and C_n. Ogni clausola C_i avrà la seguente forma $C_i = P_1$ or P_2 or …. or P_m , disgiunzione di predicati atomici (P_j).
   Facendo riferimento alla query di esempio abbiamo:
   $\boldsymbol{C_{CNF}}$ = `name LIKE 'Marco%' AND (year = '35' OR section = '2')`
b. Ogni clausola C_i, della condizione della query globale riportata in CNF ($\boldsymbol{C_{CNF}}$), contenente almeno un predicato che non può essere riscritto in termini della classe locale, costituisce una clausola residua per la local query corrispondente.
   Facendo riferimento alla query di esempio abbiamo:
   - Il predicato `section = '2'` non può essere riscritto in termini della classe locale L1, essendo l'attributo globale "*section*" non mappato nella classe locale L1 (sorgente *QMTestDB1*), quindi la clausola `(year = '35' OR section = '2')` è una clausola residua per L1.
   - Il predicato `year = '35'` non può essere riscritto in termini della classe locale L2, essendo l'attributo globale "*year*" non mappato nella classe locale

L2 (sorgente *QMTestDB2*), quindi la clausola `(year = '35' OR section = '2')` è una clausola residua per L2.

- I predicati `year = '35' e section = '2'` non possono essere riscritti in termini della classe locale L3, essendo gli attributo globali "*year*" e "*section*" non mappati nella classe locale L3 (sorgente *QMTestDB3*), quindi la clausola `(year = '35' OR section = '2')` è una clausola residua per L3.

c. La condizione residua è ottenuta tramite l'AND delle clausole residue.
   Facendo riferimento alla query di esempio abbiamo:
   C_r = `(year = '35' OR section = '2') AND (year = '35' OR section = '2') AND (year = '35' OR section = '2')`
   Che per la proprietà di *idempotenza* (A and A ≡ A) è equivalente a:
   C_r = `(year = '35' OR section = '2')`

4. *Local query* <`L_select_list`>

Per ogni query locale la <`L_select_list`> è calcolata considerando l'unione degli:

- attributi globali in <`select_list`> con un mapping non nullo in *L*;
- attributi globali usati per esprimere la join condition per *L*;
- attributi globali in <`condition`> con un mapping non nullo in *L*.

questi attributi vengono trasformati nei corrispondenti attributi locali tramite la Mapping Table.

Con riferimento alla query di esempio avremmo:

<LQ_1_`select_list`> = {First_Name, Last_Name, Email, Year}

<LQ_2_`select_list`> = {name, mail, section}

<LQ_3_`select_list`> = {Name, Email}

In conclusione avremmo le seguenti local query:

LQ_1:

```
SELECT L1.First_Name,L1.Last_Name, L1.Email, L1.Year
FROM L1
WHERE ((First_Name+''+Last_Name) LIKE 'Marco%' AND Year='35')
OR ((First_Name+''+Last_Name) LIKE 'Marco%')
```

LQ_2:

```
SELECT L2.Name, L2.mail, L2.section
FROM L2
```

```
WHERE (name LIKE 'Marco%')
OR(name LIKE 'Marco%' AND section = '2')
```

LQ_3:

```
SELECT L3.Name, L3.Email
FROM L3
WHERE (Name LIKE 'Marco%') OR (Name LIKE 'Marco%')
```

Le tuple che costituiscono il risultato delle local query verranno memorizzate in tabelle temporanee (create in un database del DBMS di supporto). In seguito viene eseguito il full outer join tra le tabelle temporanee ed infine viene applicata la final query (che comprende le eventuali Resolution functions e la condizione residua), per ottenere il risultato finale.

4.3 Testing

Sulla classe globale G1 formuleremo delle query globali in accordo con la sintassi supportata dal Query Manager riportata in Appendice A. In seguito saranno riportate alcune query che sono state eseguite, per ciascuna query riporteremo le local query expected (o corrette) e le local query effettivamente mandate alle sorgenti dai wrapper e analizzeremo la correttezza di quest'ultime.

Query con operatori logici e predicati di confronto

Query 1

```
SELECT name, email, section
FROM G1
WHERE name = 'Luca Verdi'
AND dept = '2'
AND year = '22'
```

	name = 'Luca Verdi'	year = '22'	dept = '2'
L1	(First_Name+''+Last_Name)='Luca Verdi'	Year = '22'	true
L2	name = 'Luca Verdi'	true	dept = '2'
L3	Name = 'Luca Verdi'	true	true

Tabella 7. Query 1 - Riscrittura dei predicati

Expected Local Query

LQ_1:

```
SELECT L1.Last_Name, L1.First_Name, L1.Email, L1.Year
FROM L1
WHERE (First_Name + ' ' + Last_Name) = 'Luca Verdi'
AND Year = '22'
```

LQ_2:

```
SELECT L2.dept, L2.section, L2.name, L2.mail
FROM L2
WHERE name = 'Luca Verdi'
AND dept = '2'
```

LQ_3:

```
SELECT L3.Name, L3.Email
FROM L3
WHERE Name = 'Luca Verdi'
```

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.Last_Name, L1.First_Name, L1.Email, L1.Year
FROM L1
WHERE (First_Name + ' ' + Last_Name) = 'Luca Verdi'
AND Year = '22'
```

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.section, L2.name, L2.mail
FROM L2
WHERE name = 'Luca Verdi'
AND dept = '2'
```

3 - Source "QMTestDB3" – LQ_3

```
SELECT L3.Name, L3.Email
FROM L3
WHERE Name = 'Luca Verdi'
```

Come possiamo constatare le expected local query coincidono con quelle effettivamente eseguite sulle sorgenti locali.

Query 2

```
SELECT name, email
FROM G1
WHERE name = 'Marco Rossi'
AND (section = '1' OR dept = '1')
```

	name = 'Marco Rossi'	sect = '1'	dept = '1'
L1	(First_Name+''+Last_Name)='Marco Rossi'	true	true
L2	name = 'Marco Rossi'	sect = '1'	dept = '2'
L3	Name = 'Marco Rossi'	true	true

Tabella 8. Query 2 - Riscrittura dei predicati

Expected Local Query

LQ_1:

```
SELECT L1.Last_Name, L1.First_Name, L1.Email
FROM L1
WHERE (First_Name + ' ' + Last_Name) = 'Marco Rossi'
```

LQ_2:

```
SELECT L2.section, L2.dept, L2.name, L2.mail
FROM L2
WHERE (name = 'Marco Rossi' AND section = '1')
OR (name = 'Marco Rossi' AND dept = '1')
```

LQ_3:

```
SELECT L3.Name, L3.Email
FROM L3
WHERE Name = 'Marco Rossi'
```

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.Last_Name,L1.First_Name, L1.Email
FROM L1
WHERE (First_Name + ' ' + Last_Name) = ('Marco Rossi')
OR (First_Name + ' ' + Last_Name) = ('Marco Rossi')
```

Come si può constatare la expected local query non coincide “sintatticamente” con quella effettivamente eseguita ma sono equivalenti a meno di trasformazioni logiche.

La local query LQ_1 è “semanticamente” equivalente con la query riportata in seguito per la proprietà di idempotenza ((A or A) ≡ A) (queste trasformazioni/ottimizzazioni logiche vengono effettuate da un database engine):

```
SELECT L1.Last_Name,L1.First_Name, L1.Email
FROM L1
WHERE (First_Name + ' ' + Last_Name) = ('Marco Rossi')
```

I predicati `(section = '1')`e `(dept = '1')`, essendo che gli attributi globali “*section”* e *“dept”* non sono mappati nella sorgente locale *QMTestDB1.L1*, nella local query LQ_1 entrambi saranno riscritti nella seguente forma:P_{LQ1} = *true*. Di conseguenza per l’algoritmo di calcolo delle local query condition, la query LQ_1 avrà la seguente condizione:

```
(((First_Name + ' ' + Last_Name) = ('Marco Rossi') AND true)
OR ((First_Name + ' ' + Last_Name) = ('Marco Rossi') AND
true))
```

Per la proprietà di *neutrality* ((A and true) ≡ A) e per la proprietà di *idempotenza* ((A or A) ≡ A) la condizione della local query LQ_1 è equivalente a:

`(First_Name + ' ' + Last_Name) = ('Marco Rossi')` quindi coincide con la condizione della expected local query.

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.section, L2.name, L2.mail
FROM L2
WHERE ((name) = ('Marco Rossi') AND (section) = ('1'))
OR ((name) = ('Marco Rossi') AND (dept) = ('1'))
```

La expected local query coincide con quella effettivamente eseguita sulla sorgente locale.

3 - Source "QMTestDB3" – LQ_3

```
SELECT L3.Name, L3.Email
FROM L3
WHERE (Name) = ('Marco Rossi') OR (Name) = ('Marco Rossi')
```

La local query LQ_3 è “semanticamente” equivalente con la query riportata in seguito per la proprietà di idempotenza ((A or A) ≡ A):

```
SELECT L3.Name, L3.Email FROM L3
WHERE (Name) = ('Marco Rossi')
```

Query 3

```
SELECT name
FROM G1
WHERE (name = 'Marco Rossi' AND year = '34')
OR (name = 'Marco Rossi' AND dept = '2')
```

	name = 'Marco Rossi'	year = '34'	dept = '2'
L1	(First_Name+''+Last_Name)='Marco Rossi'	year = '34'	true
L2	name = 'Marco Rossi'	true	dept = '2'
L3	Name = 'Marco Rossi'	true	true

Tabella 9. Query 3 - Riscrittura dei predicati

Expected Local Query

LQ_1:

```
SELECT L1.Last_Name, L1.First_Name, L1.Year
FROM L1
WHERE (First_Name + ' ' + Last_Name) = 'Marco Rossi'
```

LQ_2:

```
SELECT L2.dept, L2.name
FROM L2
WHERE name = 'Marco Rossi'
```

LQ_3:

```
SELECT L3.Name
FROM L3
WHERE Name = 'Marco Rossi'
```

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.Last_Name,L1.First_Name, L1.Year
FROM L1
WHERE ((First_Name + ' ' + Last_Name) = ('Marco Rossi') AND
(Year) = ('34'))
OR (First_Name + ' ' + Last_Name) = ('Marco Rossi')
```

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.name
FROM L2
WHERE (name) = ('Marco Rossi')
OR ((name) = ('Marco Rossi') AND (dept) = ('2'))
```

3 - Source "QMTestDB3" – LQ_3

```
SELECT L3.Name
FROM L3
WHERE (Name) = ('Marco Rossi')
OR (Name) = ('Marco Rossi')
```

Le local query effettivamente eseguite sulle sorgenti sono “sintatticamente” diverse da quelle expected ma “semanticamente” equivalenti a meno di ottimizzazioni/trasformazioni logiche, infatti per la proprietà di *absorption* (A or (A and B) ≡ A) le condizioni delle local query LQ_1 e LQ_2 sono equivalenti con le condizioni delle rispettive local query expected.

Mentre la local query LQ_3 è “semanticamente” equivalente con la rispettiva local query expected per la proprietà di idempotenza ((A or A) ≡ A):

```
SELECT L3.Name, L3.Email FROM L3
WHERE (Name) = ('Marco Rossi')
```

Query 4

```
SELECT name, email
FROM G1
WHERE (name = 'Luca Verdi')
OR (year = '22' AND dept ='2')
```

	name = 'Luca Verdi'	year = '22'	dept = '2'
L1	(First_Name+''+Last_Name)='Luca Verdi'	year = '22'	true
L2	name = 'Luca Verdi'	true	dept = '2'
L3	Name = 'Luca Verdi'	true	true

Tabella 10. Query 4 - Riscrittura dei Predicati

Expected Local Query

LQ_1:

```
SELECT L1.Last_Name, L1.First_Name, L1.Email, L1.Year
FROM L1
WHERE (First_Name + ' ' + Last_Name) = 'Luca Verdi'
OR Year = '22'
```

LQ_2:

```
SELECT L2.section, L2.name, L2.mail
FROM L2
WHERE name = 'Luca Verdi'
OR dept = '2'
```

LQ_3:

```
SELECT L3.Name, L3.Email
FROM L3
```

La local query LQ_3 deve avere l'espressione sopra riportata perché se applichiamo l'algoritmo di calcolo delle local query condition, la condizione della query globale (che si trova in DNF) sarà cosi riscritta (Tabella 10):

```
(name = 'Luca Verdi') OR (true AND true) ≡
(name = 'Luca Verdi') OR true ≡ true
```

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.Last_Name,L1.First_Name, L1.Email, L1.Year
FROM L1
WHERE (First_Name + ' ' + Last_Name) = ('Luca Verdi')
OR (Year) = ('22')
```

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.name, L2.mail
FROM L2
WHERE (name) = ('Luca Verdi') or (dept) = ('2')
```

3 - Source "QMTestDB3" – LQ_3

```
SELECT L3.Name, L3.Email
FROM L3
```

Si può constatare che le expected local query coincidono con quelle effettivamente eseguite.

Query con predicato LIKE e predicato NULL

Query 5

```
SELECT name, year, section, dept
FROM G1
WHERE (name LIKE '%Bianchi')
AND IS NULL year
AND NOT IS NULL dept
```

	name LIKE '%Bianchi'	is null year	not is null dept
L1	(First_Name+''+Last_Name)LIKE'%Bianchi'	year is null	true
L2	name LIKE '%Bianchi'	true	Dept is not null
L3	Name LIKE '%Bianchi'	true	true

Tabella 11. Query 5 - Riscrittura dei predicati

Expected Local Query

LQ_1:

```
SELECT L1.Last_Name, L1.First_Name, L1.Year
FROM L1
WHERE (First_Name + ' ' + Last_Name) LIKE '%Bianchi'
AND Year IS NULL
```

LQ_2:

```
SELECT L2.section, L2.name, L2.dept
FROM L2
WHERE name LIKE '%Bianchi'
AND dept IS NOT NULL
```

LQ_3:

```
SELECT L3.Name
FROM L3
WHERE Name LIKE '%Bianchi'
```

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.Last_Name,L1.First_Name, L1.Year
FROM L1
WHERE ((First_Name + ' ' + Last_Name) LIKE ('%Bianchi') AND
L1.Year is null)
```

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.section, L2.name
FROM L2
WHERE ((Name) LIKE ('%Bianchi') AND not (L2.dept is null))
```

3 - Source "QMTestDB3" – LQ_3

```
SELECT L3.Name
FROM L3
WHERE (Name) LIKE ('%Bianchi')
```

Si può constatare che le expected local query coincidono con quelle effettivamente eseguite.

Query con clausola ORDER BY

Query 6

```
SELECT name, dept, section
FROM G
WHERE section = 2
AND not is null year
ORDER BY dept DESC
```

La `<order by clause>` costituisce una clausola residua, quindi non verrà eseguita direttamente sulle sorgenti ma tramite la final query verrà risolta a livello globale.

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.First_Name,L1.Last_Name, L1.year
FROM L1 WHERE not (L1.Year is null)
```

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.section, L2.name
FROM L2
WHERE (section) = ('2')
```

3 - Source "QMTestDB3" – LQ_3

```
SELECT L3.Name
FROM L3
```

Final Query

```
SELECT name, dept, section
FROM je1_G
WHERE (section =  '2' )
AND not year is null
ORDER BY dept DESC
```

Query con clausola GROUP BY e HAVING

Query 7

```
SELECT section
FROM G
WHERE year != '22'
GROUP BY  section, dept
HAVING section != '1'
```

La <group by clause> e la <having clause> costituiscono clausole residue, quindi non verranno eseguite direttamente sulle sorgenti ma tramite la final query verranno risolte a livello globale.

Local Query

1 - Source "QMTestDB1" – LQ_1

```
SELECT L1.year FROM L1
```

2 - Source "QMTestDB2" – LQ_2

```
SELECT L2.dept, L2.section FROM L2
```

Final Query

```
SELECT section
FROM je1_G
WHERE (year <>  '22' )
GROUP BY section, dept
HAVING (section <>  '1' )
```

Capitolo 5

5 Il benchmark THALIA

5.1 Descrizione del benchmark THALIA

THALIA *(Test Harness for the Assessment of Legacy information Integration Approaches)* [26] è un benchmark di dominio pubblico per testare i sistemi di data integration. THALIA fornisce ai ricercatori, un insieme di sorgenti dati (scaricabili dal sito di THALIA[1]) che rappresentano cataloghi di corsi universitari, e un insieme di 12 query. L'obiettivo è determinare come il sistema di integrazione in questione riesce a risolvere le eterogeneità sintattiche e semantiche. Naturalmente, gli ideatori di questo benchmark hanno considerato le eterogeneità di rilevanza maggiore, in quanto queste mettono in luce i problemi principali dell'integrazione che ancora oggi molti sistemi di integrazione non risolvono al meglio. Le 25 sorgenti dati si riferiscono ai cataloghi di altrettanti dipartimenti universitari. I cataloghi delle rispettive università sono accessibili tramite il sito ufficiale di THALIA[1], e permettono di avere una visione completa di come può cambiare la rappresentazione dell'informazione, da un'università all'altra. In Figura 9 è rappresentato il catalogo dei corsi dell'Arizona State University, con le relative informazioni.

Figura 9. Il catalogo dell'ASU

1 http://www.cise.ufl.edu/research/dbintegrate/thalia/

I dati relativi a tutti i cataloghi delle varie università vengono forniti sotto forma di file XML. L'estrazione dell'informazione dal formato originale e la trasformazione in file XML è stata fatta tramite uno specifico wrapper chiamato Telegraph Screen Scraper (TESS). In Figura 10 è rappresentato il file XML relativo al catalogo dell'Arizona State University ed il corrispondente schema del file XML.

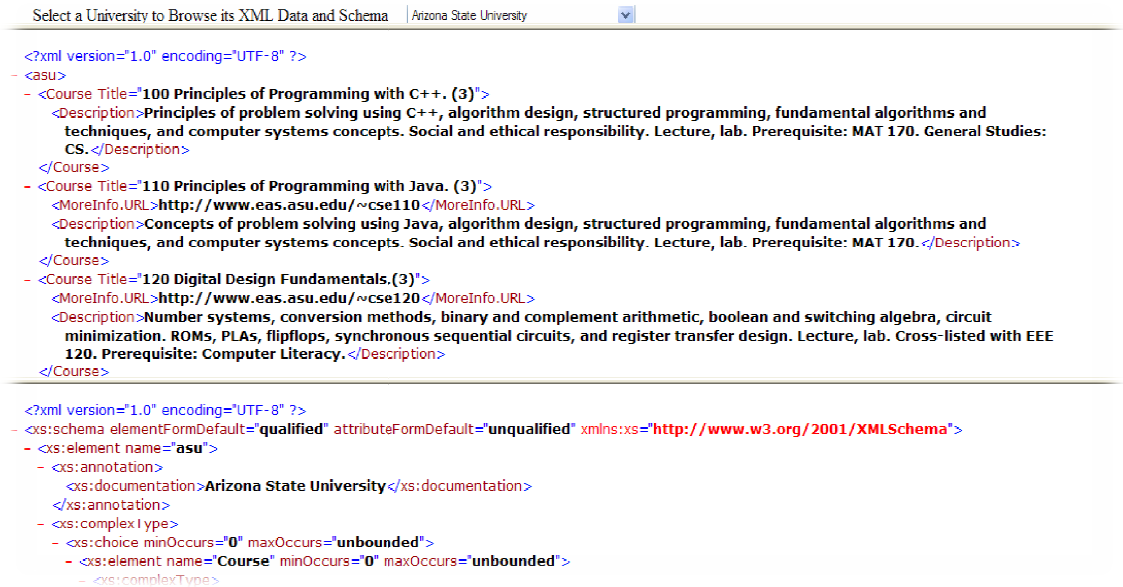

Figura 10. File e schema XML dell'ASU

Oltre ai file XML vengono forniti anche le 12 query in linguaggio XQuery (XML query language). Per ogni tipo di eterogeneità, il benchmark propone una query, ed ognuna di queste si riferisce a due sorgenti dati: un reference schema ed un challenge schema che devono essere integrati tramite il sistema di integrazione che si sta testando.

5.2 MOMIS e le 12 query del benchmark THALIA

Come sopra citato il benchmark THALIA fornisce 25 sorgenti (sotto forma di file XML con il rispettivo schema XSD) e 12 query (in linguaggio XQuery), ognuna delle quali propone un tipo di eterogeneità, che il sistema di integrazione dovrebbe riuscire a risolvere; ogni query viene formulata sullo schema globale ottenuto dall'integrazione di due schemi locali che vengono chiamati: reference schema e challenge schema.

Essendo le sorgenti fornite sotto forma di file XML, verrà utilizzato un wrapper che traduce automaticamente gli schemi XSD in strutture relazionali ed importa i dati in formato relazionale. Per ogni sorgente in XML viene creato un database relazionale e ogni query fornita in XQuery viene tradotta nel linguaggio OQL_I^3 (Appendice A), il linguaggio accettato dal Query Manager del sistema MOMIS per la formulazione delle query. Nel benchmark vengono proposte tre categorie di eterogeneità:

- ***Eterogeneità degli attributi***: è un tipo di eterogeneità che esiste tra due attributi correlati, di schemi differenti. Di questa categoria fanno parte le seguenti eterogeneità: *Synonyms*, *Simple Mapping*, *Union Types*, *Complex Mapping*, *Language Expression*.
- ***Mancanza dei dati***: è un tipo di eterogeneità dovuta alla mancanza di informazione in uno degli schemi integrati. Di questa categoria fanno parte le seguenti eterogeneità: *Nulls*, *Virtual Columns*, *Semantic incompatibility*.
- ***Eterogeneità strutturali***: è un tipo di eterogeneità dovuta alle diverse modalità con cui vengono rappresentate informazioni correlate in schemi differenti, cioè diversi modelli vengono utilizzati in diversi schemi per rappresentare le stesse informazioni. Le eterogeneità che fanno parte in questa categoria sono: *Same attribute in different structure*, *Handling sets*, *Attribute name does not define semantics*, *Attribute composition*.

Di seguito vengono descritte le query del benchmark, l'eterogeneità intrinseca nella query e la modalità con cui *MOMIS* riesce a risolvere i diversi tipi di eterogeneità [27] e fornire una risposta alle query globali target formulate ciascuna su uno schema globale ottenuto dall'integrazione di due particolari sorgenti (specificate nel benchmark).

Query 1

Benchmark Query: “Elenco dei corsi tenuti dal professore Mark ”

Reference Schema: Georgia Tech University (*gatech*)

Challenge Schema: Carnegie Mellon University (*cmu*)

Eterogeneità da risolvere: *Synonyms* (*sinonimi*), attributi con nomi diversi ma con lo stesso significato.

Obiettivo: Determinare che nel catalogo dei corsi di *cmu* l’informazione relativa al professore del corso può essere trovata nell’attributo “Lecturer”. In *gatech* questa informazione è contenuta nell’attributo “Instructor”.

MOMIS:

Come è stato già descritto in precedenza in MOMIS ad ogni classe globale corrisponde una Mapping Table. Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l’attributo “Instructor” della sorgente *gatech* e l’attributo “Lecturer” della sorgente *cmu* nello stesso attributo globale “Lecturer” della classe globale Course.

Course	*gatech.Course*	*cmu.Course*
Code	CRN	Code
Title	Title	CourseTitle
Lecturer	Instructor	Lecturer

Tabella 12. Query 1 - Mapping Table

- ✓ Nessun ulteriore raffinamento.

Query eseguita:
```
SELECT Code, Title, Lecturer
FROM Course
WHERE Lecturer LIKE '%mark%'
```

Query 2

Benchmark Query: “Elenco di tutti i corsi che iniziano alle ore 1:30 pm”

Reference Schema: Carnegie Mellon University (*cmu*)

Challenge Schema: University of Massachusetts (*umb*)

Eterogeneità da risolvere: *Simple Mapping* (*Mapping semplice*), attributi correlati presenti in diversi schemi, differiscono per una trasformazione matematica dei loro valori.

Obiettivo: Convertire l'orario rappresentato in base 12 (es. 1:30 pm) in base 24 (es. 13:30).

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "Time" della sorgente *cmu* e l'attributo "Times" della sorgente *umb* nello stesso attributo globale "Time" della classe globale Course.

Course	*cmu.Course*	*umb.Course*
Code	Code	Code
Title	CourseTitle	TitleCredits
Time	Time	MTF[Time][umb.Course]

Tabella 13. Query 2 - Mapping Table

In MOMIS si possono definire durante la fase di Mapping Refinement delle funzioni chiamate *Data Conversion Functions*. Per ogni elemento non nullo MT[*GA*][*L*], della Mapping Table, il progettista può definire una *Data Conversion Function,* denotata con MTF[*GA*][*L*]; questa funzione determina come l'attributo locale sarà trasformato nel corrispondente attributo globale su cui è mappato. Le MTF[*GA*][*L*] vengono eseguite a livello di sorgente e quindi devono essere supportate dal local source wrapper. Come funzioni di conversione in MOMIS si possono usare delle funzioni *like SQL92*:

- CHAR_LENGTH: ritorna la lunghezza di una stringa;
- POSITION: cerca la posizione di determinati caratteri in una stringa;
- SUBSTRING: ritorna una parte di una stringa;
- CASE … WHEN … THEN: trasforma un record sulla base di determinati valori espressi nella condizione;
- RIGHT and LEFT: ritornano rispettivamente i primi (o gli ultimi) N caratteri di una stringa.

Queste funzioni sono tradotte, a livello wrapper, nel particolare "dialetto SQL" del DBMS che ospita le sorgenti.

- ✓ Raffinamento sull'attributo "Times" della sorgente *umb*:

```
MTF[Time][umb.Course] = TIME12-24[Times,1,12] +
SUBSTRING[Times,6,1] + TIME12-24[Times,7,12]
```

La funzione TIME12-24 è stata cosi definita:

```
CASE WHEN ISNUMERIC[SUBSTRING[Times, 1, 2]] = 1 THEN
CASE WHEN CAST[SUBSTRING[Times, 1, 2] AS int] > 12
THEN CAST[CAST[SUBSTRING[Times, 1, 2] AS integer]-12
AS nvarchar[2]]
ELSE SUBSTRING[Times, 1, 2]
END + SUBSTRING[Times, 3, 4] +
CASE WHEN CAST[SUBSTRING[Times, 7, 2] AS int] > 12
THEN CAST[CAST[SUBSTRING[Times, 7, 2] AS integer]- 12
AS nvarchar[3]] ELSE SUBSTRING[Times, 7, 2]
END + SUBSTRING[Times, 9, 3]
END
```

Query eseguita:

```
SELECT  Title, Time
        FROM Course
        WHERE Time LIKE '1:30%'
```

Query 3

Benchmark Query: “Elenco di tutti i corsi che contengono nel loro nome la stringa Data Structures ”

Reference Schema: University of Maryland (*umd*)

Challenge Schema: Brown University (*brown*)

Eterogeneità da risolvere: *Union Types* (*Tipi unione*), attributi in schemi diversi utilizzano diversi tipi di dato per rappresentare la stessa informazione.

Obiettivo: Determinare che nel catalogo dei corsi di *brown* l’informazione relativa al titolo può essere trovata nell’attributo “Title”, il valore del quale è una stringa composta da stringhe semplici e un link (URL del corso). In *umd* questa informazione è contenuta nell’attributo “CourseName” ed il tipo di dato è una stringa semplice. Inoltre in questa query abbiamo un'altra eterogeneità che deve essere risolta: CourseName (*umd*) è sinonimo di Title (*brown*).

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "CourseName" della sorgente *umd* e l'attributo "Title" della sorgente *brown* nello stesso attributo globale "CorseName" della classe globale Course.

Course	*umd.Course*	*brown.Course*
Code	Code	Code
CourseName	CorseName	MTF[CourseName][brown.Course]

Tabella 14. Query 3 - Mapping Table

- ✓ Raffinamento sull'attributo "Title" della sorgente *brown*:

```
MTF[CourseName][brown.Course] =
SUBSTRING[Title, CHARINDEX['/\"', Title] + 3,
CHARINDEX['hr.', SUBSTRING[Title, CHARINDEX['/\"', Title]
+ 3, 1000]]- 1]
```

Query eseguita:

```
SELECT  Code, CourseName
        FROM Course
        WHERE CourseName LIKE '%Data Structures%'
```

Query 4

Benchmark Query: "Elenco di tutti i corsi con più di 10 crediti"

Reference Schema: Carnegie Mellon University (*cmu*)

Challenge Schema: Swiss Federal Institute of Technology Zurich (*ethz*)

Eterogeneità da risolvere: *Complex Mapping* (*Mapping complesso*), attributi correlati di diversi schemi, differiscono per una trasformazione complessa dei lori valori.

Obiettivo: Oltre alla gestione dei problemi relativi alle diverse lingue di rappresentazione (inglese e tedesco), l'obiettivo è estrarre il valore numerico dei crediti (in ore) da una stringa che descrive il carico di lavoro (tedesco: "Umfang") del corso. In *cmu* il carico di lavoro viene rappresentato dal numero di crediti (Units[integer]), mentre in ethz il carico di lavoro viene rappresentato da una stringa (Umfag[string]).

MOMIS:

Per risolvere la query, nella fase di generazione del GS:

- ✓ Viene mappato l'attributo "Units" della sorgente cmu e l'attributo "Umfag" della sorgente ethz nello stesso attributo globale "Units" della classe globale Course.

Course	*cmu.Course*	*ethz.Unterricht*
Title	CourseTitle	Titel
Units	Units	MTF[Units][ethz.Unterricht]

Tabella 15. Query 4 - Mapping Table

- ✓ Raffinamento sull'attributo Umfag di ethz:

```
MTF[Units][ethz.Unterricht] = CAST[SUBSTRING[Umfang,
CHARINDEX['V', Umfang] - 1, 1] AS int] +
CAST[SUBSTRING[Umfang, CHARINDEX['U', Umfang] - 1, 1]
AS int] + 1
```

Query eseguita:

```
SELECT  Title, Units
        FROM Course
        WHERE Title LIKE TRANSLATE ['%Database%','en']
        AND Units > 10
```

Query 5

Benchmark Query: "Elenco di tutti i corsi che contengono la stringa "database" nel loro nome"

Reference Schema: University of Maryland (*umd*)

Challenge Schema: Swiss Federal Institute of Technology Zurich (*ethz*)

Eterogeneità da risolvere: *Language Expression* (Lingua utilizzata), lo stesso attributo in diversi schemi viene espresso in lingue diverse.

Obiettivo: Convertire il termine inglese 'Database' nel termine tedesco 'Datenbank' o 'Datei' o 'Datenbasis' e restituire i corsi di *ethz* che contengono queste sottostringhe.

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "CourseName" della sorgente *umd* e l'attributo "Titel" della sorgente *ethz* nello stesso attributo globale "CourseName" della classe globale Course.

Course	*umd.Course*	*ethz.Unterricht*
CourseName	CourseName	Titel

Tabella 16. Query 5 - Mapping Table

- ✓ In MOMIS è possibile formulare delle query con condizioni specificate in diverse lingue; in questo caso la condizione sarà cosi espressa:

  GA op TRANSLATE(*term*, *Language*)

  Dove *GA* è un attributo globale e term è una parola in una determinata lingua (es. inglese: 'en'). Durante il processo di Query Unfolding una funzione TRANSLATION, tradurrà le condizioni della query globale, dalla lingua specificata in TRANSLATE(*term*, *Language*) nelle differenti lingue delle sorgenti locali. L'operazione di traduzione viene effettuata sfruttando un vocabolario Open Source sviluppato nell'ambito del Gutenberg Project [1].

Query eseguita:
```
SELECT CourseName
FROM Course
WHERE CourseName LIKE TRANSLATE['%Database%', 'en']
```

La seguente query sarà riscritta per le due sorgenti locali, *umd* e *ethz*, nel seguente modo:

umd local query:

```
SELECT CourseName
FROM Course
WHERE CourseName LIKE '%Database%'
```

ethz local query:

```
SELECT Titel
FROM Unterricht
WHERE Titel LIKE '%Datenbank%'
OR Titel LIKE '%Datei%'
OR Titel LIKE '%Datenbasis%'
```

1 http://www.gutenberg.org

Query 6

Benchmark Query: “Elenco di tutti i corsi che contengono la stringa “verification” nel loro nome”

Reference Schema: University of Toronto (*toronto*)

Challenge Schema: Carnegie Mellon University (*cmu*)

Eterogeneità da risolvere: *Nulls* (*Valori Null*), l’attributo non esiste o il valore dell’attributo non esiste.

Obiettivo: Trattare in modo appropriato i valori NULL, cioè determinare la mancanza dell’attributo text per i corsi della sorgente *cmu*.

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l’attributo “text” della sorgente *toronto* nell’attributo globale “Text” della classe globale Course. Nessun attributo della sorgente *cmu* è mappato in questo attributo globale.

Course	*toronto.course*	*cmu.Course*
Title	title	CourseTitle
Text	text	

Tabella 17. Query 6 - Mapping Table

- ✓ Nessun ulteriore raffinamento.

Query eseguita:
```
SELECT  Title, Text
    FROM Course
   WHERE Title LIKE '%verification%'
```

Per tutte le tuple che provengono dalla sorgente *cmu*, l’attributo globale “Text” avrà come valore: <no mapping>, cioè l’attributo “Text” non è mappato in quella sorgente (vedere query 8).

Query 7

Benchmark Query: “Elenco di tutti i corsi che non prevedono un prerequisito d’ingresso”

Reference Schema: University of Michigan (*umich*)

Challenge Schema: Arizona State University (*asu*)

Eterogeneità da risolvere: *Virtual Columns* (*Attributi virtuali*), l’informazione può essere rappresentata in maniera esplicita in uno schema, e in maniera implicita in un

altro schema; l'informazione in quest'ultimo caso deve essere inferita da uno o più valori.

Obiettivo: Riuscire ad estrarre l'informazione sul prerequisito di un corso dall'attributo "MoreInfoURL" della sorgente *asu*. Nella sorgente *umich* l'informazione riguardante i prerequisiti di un corso è contenuta nell'attributo "prerequisite", mentre è un informazione che si trova sotto forma di commento in un attributo della sorgente *asu*.

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "prerequisite" della sorgente *umich* e l'attributo "MoreInfoURL" della sorgente *asu* nello stesso attributo globale "prerequisite" della classe globale Course.

Course	*umich.course*	*asu.Course*
name	name	Title
description	description	Description
prerequisite	prerequisite	MTF[prerequisite][asu.Course]

Tabella 18. Query 7 - Mapping Table

- ✓ Raffinamento sull'attributo "MoreInfoURL" di *asu*:

```
MTF[prerequisite][asu.Course] =
CASE PATINDEX['%Prerequisite%', Description]
WHEN 0 THEN 'None' ELSE RIGHT[Description,
LEN[Description] - PATINDEX['%Prerequisite%', Description]
+ 1]
END
```

Query eseguita:

```
SELECT  name, description, prerequisite
        FROM Course
        WHERE prerequisite LIKE '%none%'
```

Query 8

Benchmark Query: "Elenco di tutti i corsi aperti agli studenti junior "

Reference Schema: Georgia Tech University (*gatech*)

Challenge Schema: Swiss Federal Institute of Technology Zurich (*ethz*)

Eterogeneità da risolvere: *Semantic incompatibility* (*incompatibilità semantiche*), un concetto del mondo reale può essere modellato in uno schema e non essere modellato in un altro.

Obiettivo: Anche se si può ritornare un valore *NULL* dalla ricerca su *ethz*, questa risposta può essere fuorviante. Per affrontare intelligentemente questo tipo di query, il sistema dovrebbe distinguere i diversi tipi di *NULL*. Nello specifico abbiamo i seguenti casi: "il dato non c'è ma può essere presente", cioè il valore dell'attributo è *null*, "il dato non c'è e non può essere presente", cioè il valore non è presente perché l'attributo non è specificato nella sorgente.

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "Description" della sorgente *gatech* nell'attributo globale "Description" della classe globale Course. Nessun attributo della sorgente *ethz* è mappato in questo attributo globale.

Course	*gatech.Course*	*ethz.Unterricht*
Title	Title	Titel
Description	Description	

Tabella 19. Query 8 - Mapping Table

- ✓ Nessun ulteriore raffinamento.

MOMIS distingue i due casi sopra riportati di *NULL*:

- ✓ Se il dato non c'è ma può essere presente, viene visualizzato <null data>, cioè il valore di quell'attributo nella sorgente è *null.*
- ✓ Se il dato non c'è e non può essere presente, viene visualizzato <no mapping>, cioè quell'attributo globale non è mappato nella sorgente locale.

Query eseguita:
```
SELECT  DISTINCT Title, Description
        FROM Course
        WHERE Description LIKE '%JR%'
```

Query 9

Benchmark Query: "Elenco delle aule nelle quali si tiene il corso di database"

Reference Schema: Brown University (*brown*)

Challenge Schema: University of Maryland (*umd*)

Eterogeneità da risolvere: *Same attribute in different structure* (*Stesso attributo in posizioni diverse*), lo stesso attributo o attributi correlati possono essere localizzati in posizioni diverse nei differenti schemi.

Obiettivo: Determinare che l'informazione sull'aula, nella sorgente *umd*, è disponibile nell'attributo "Time" dello schema "Section", mentre nella sorgente *brown* abbiamo un attributo esplicito "Room".

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "Room" della sorgente *brown* e l'attributo "Time" della sorgente *umd* (schema Section) nello stesso attributo globale "Room" della classe globale Course.

Course	*brown.Course*	*umd.Course*	*umd.Section*
Code	Code	Code	Code
Title	Title	CourseName	
Room	Room		MTF[Room][umd.Section]

Tabella 20. Query 9 - Mapping Table

- ✓ Raffinamento sull'attributo Time di umd:

```
MTF[Room][umd.Section] = SUBSTRING[[Time],
PATINDEX['%[%', [Time]], 30]
```

Query eseguita:

```
SELECT  Code, Title, Room
        FROM Course
        WHERE Title LIKE '%Software Engineering%'
```

Query 10

Benchmark Query: "Elenco dei professori dei corsi di computer"

Reference Schema: Carnegie Mellon University (*cmu*)

Challenge Schema: University of Maryland (*umd*)

Eterogeneità da risolvere: *Handling sets* (*Trattamento dell'insieme dei valori*), un attributo in uno schema può avere come valore un insieme di valori, mentre in altro schema l'informazione viene rappresentata da un insieme di attributi a valore singolo.

Obiettivo: Determinare che l'informazione sul professore è memorizzata nello schema Section della sorgente *umd*, nello specifico, bisogna estrarre l'informazione

relativa al nome del professore dall'attributo "Title" dello schema *Section*. Nella sorgente *cmu*, l'informazione relativa ai professori del corso è contenuta nell'attributo "Lecturer".

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "Lecturer" della sorgente *cmu* e l'attributo "Title" della sorgente *umd* (schema Section) nello stesso attributo globale "Lecturer" della classe globale Course.

Course	*cmu.Course*	*umd.Course*	*umd.Section*
Code	Code	Code	Code
Lecturer	Lecturer		MTF[Lecturer][umd.Section]
Title	CourseTitle	CourseName	

Tabella 21. Query 10 - Mapping Table

- ✓ Raffinamento sull'attributo Title di umd:

```
MTF[Lecturer][umd.Section] = SUBSTRING[SUBSTRING[Title, 1,
PATINDEX['%.%',Title]], PATINDEX['%]%',Title] + 2,
PATINDEX['%.%',Title] + 1]
```

Query eseguita:

```
SELECT  Code, Title, Lecturer
        FROM Course
        WHERE Title LIKE '%Computer%'
```

Query 11

Benchmark Query: "Elenco dei professori dei corsi di database"

Reference Schema: Carnegie Mellon University (*cmu*)

Challenge Schema: University of California San Diego (*ucsd*)

Eterogeneità da risolvere: *Attribute name does not define semantics* (il nome dell'attributo non definisce la sua semantica), il nome dell'attributo non descrive adeguatamente il significato dell'attributo stesso.

Obiettivo: nel catalogo dei corsi dell'università di California San Diego (*ucsd*), l'informazione sul professore del corso è contenuto negli attributi "Fall2003", "Winter2004" e "Spring2004".

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Viene mappato l'attributo "Lecturer" della sorgente *cmu* e gli attributi "Fall2003", "Winter2004" e "Spring2004" della sorgente *ucsd* nello stesso attributo globale "Lecturer" della classe globale Course.

Course	*cmu.Course*	*ucsd.Course*
Code	Code	Number
Lecturer	Lecturer	MTF[Lecturer][ucsd.Course]
Title	CourseTitle	Title

Tabella 22. Query 11 - Mapping Table

- ✓ Raffinamento sul attributo "Fall2003" di *ucsd*:

```
MTF[Lecturer][ucsd.Course] = CASE WHEN [LEN[Fall2003] >
LEN[Winter2004] AND LEN[Fall2003] > LEN[Spring2004]]
THEN Fall2003 WHEN [LEN[Winter2004] > LEN[Fall2003]
AND LEN[Winter2004] > LEN[Spring2004]] THEN Winter2004
WHEN [LEN[Spring2004] > LEN[Fall2003] AND
LEN[Spring2004] > LEN[Winter2004]] THEN Spring2004
END
```

Query eseguita:

```
SELECT Title, Lecturer
FROM Course
WHERE Title LIKE '%Database%'
```

Query 12

Benchmark Query: "Elenco dei nomi e degli orari dei corsi di reti di calcolatori"

Reference Schema: Carnegie Mellon University (*cmu*)

Challenge Schema: Brown University (*brown*)

Eterogeneità da risolvere: *Attribute composition* (*Composizione degli attributi*), la medesima informazione può essere rappresentata tramite un attributo singolo oppure tramite un insieme di attributi, possibilmente organizzati in maniera gerarchica.

Obiettivo: Determinare che le informazioni relative ad un corso (nome, giorno, ora) nella sorgente *brown* sono contenute nell'attributo "Title", mentre nella sorgente *cmu* ogni informazione è contenuta in uno specifico attributo (CorseTitle, Day, Time).

MOMIS:

Per risolvere la query, nella fase di generazione del *GS*:

- ✓ Mapping tra gli attributi “CourseTitle”, “Day” e “Time” della sorgente *cmu* e l’attributo “Title” della sorgente *brown*.

Course	*cmu.Course*	*brown.Course*
Title	CourseTitle	MTF[Title][brown.Course]
Day	Day	MTF[Day][brown.Course]
Time	Time	MTF[Time][brown.Course]

Tabella 23. Query 12 - Mapping Table

- ✓ Raffinamento sugli attributi Title, Day, Time di brown:

```
MTF[Title][brown.Course] = SUBSTRING[Title,
CHARINDEX['/\"', Title] + 3, CHARINDEX['hr.',
SUBSTRING[Title, CHARINDEX['/\"', Title] + 3, 1000]] - 1]

MTF[Day][brown.Course] = SUBSTRING[Title,
CHARINDEX['hr.',
Title] + 4, CHARINDEX[' ', SUBSTRING[Title,
CHARINDEX['hr.', Title] + 4, 10]]]
MDTF[Time][brown.Course] = SUBSTRING[Title, CHARINDEX['',
SUBSTRING[Title, CHARINDEX['hr.', Title] + 4, 10]] +
CHARINDEX['hr.', Title] + 4, 15]
```

Query eseguita:

```
SELECT Title, Day, Time
FROM Course
WHERE Title LIKE '%Computer%Networks%'
```

Capitolo 6

6 SQL Server vs HSQLDB, confronto delle prestazioni

6.1 Il Data Set per il confronto delle prestazioni

L'obiettivo di questo capitolo è valutare le prestazioni, cioè i tempi di risposta (tempi di esecuzione delle query) formulate su uno schema globale ottenuto dall'integrazione di due sorgenti che verranno descritte in seguito, per le due versioni del *Query Manager* del sistema MOMIS: la versione che utilizza come DBMS di supporto al query processing *Microsoft SQL Server* e la versione che utilizza *HSQLDB* come DBMS di supporto. Per il confronto delle prestazioni sono state utilizzate le sorgenti del benchmark TPC-H[1] (*Transaction Processing Performance Council* (*TPC*)). TPC-H è un benchmark pubblico nato per valutare le prestazioni dei DBMS attraverso l'esecuzione di un insieme di query[2] su un database standard (database in cui sono memorizzate informazioni su ordini, fornitori e clienti di un'azienda). Dal sito ufficiale del benchmark[1] si possono scaricare le tabelle del database, i dati delle tabelle vengono forniti sotto forma di file XML, per ogni file XML è stato definito lo schema XSD corrispondente ed è stato utilizzato un wrapper che traduce automaticamente gli schemi XSD in strutture relazionali e importa i dati in formato relazionale. Le tabelle sono state importate in due database relazionali del DBMS Microsoft SQL Server in locale. Di seguito vengono riportati gli schemi dei due database relazionali utilizzati come sorgenti per il testing delle performance.

1. Sorgente Relazionale: TPCH1

region (r_regionkey, r_name, r_comment) 5 tuple

nation (n_nationkey, n_name, n_regionkey, n_comment) 25 tuple

FK: n_regionkey **REFERENCES region**

1 Transaction Processing Performance Council (TPC) http://www.tpc.org/tpch/

2 Non saranno eseguite le query del benchmark perché la loro sintassi non è conforme alla sintassi supportata dal Query Manager del sistema MOMIS.

supplier (s_suppkey, s_name, s_address, s_nationkey, s_phone, s_acctbal, s_comment) 100 tuple

FK: n_nationkey **REFERENCES nation**

part (p_partkey, p_name, p_mfgr, p_brand, p_type, p_size, p_container, p_retailprice, p_comment) 2000 tuple

partsupp (ps_partkey, ps_suppkey, ps_availqty, ps_supplycost, ps_comment) 8000 tuple

FK: ps_partkey **REFERENCES part**

FK: ps_suppkey **REFERENCES supplier**

customer (c_custkey, c_name, c_address, c_nationkey, c_phone, c_acctbal, c_mktsegment, c_comment) 1500 tuple

FK: c_nationkey **REFERENCES nation**

Il database TPCH1 contiene all'incirca 12000 tuple.

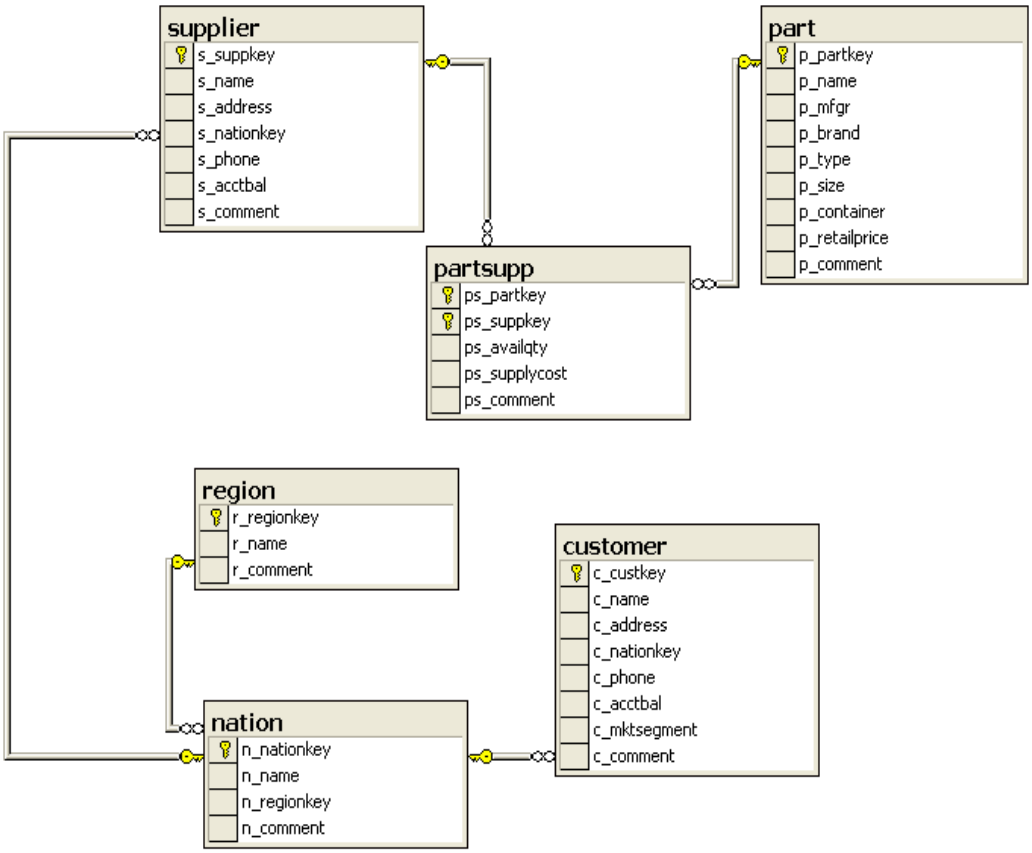

Figura 11. Diagramma database TPCH1

2. Sorgente Relazionale: TPCH2

nation (nationkey, name, regionkey, comment) 25 tuple

partsupp (partkey, suppkey, disponibilità, costofornitura, commento) 8000 tuple

customer (custkey, nome, indirizzo, nationkey, telefono, acctbal, mktsegment, commento) 1500 tuple

FK: nationkey **REFERENCES nation**

orders (o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderpriority, o_clerk, o_comment) 15000 tuple

FK: o_custkey **REFERENCES customer**

lineitem (l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_price, l_discount, l_flag, l_linestatus, l_shipinstruct, l_shipmode, l_comment) 60175

FK: l_orderkey REFERENCES orders

FK: (l_partkey, l_suppkey) **REFERENCES partsupp**

Il database TPCH2 contiene all'incirca 85000 tuple.

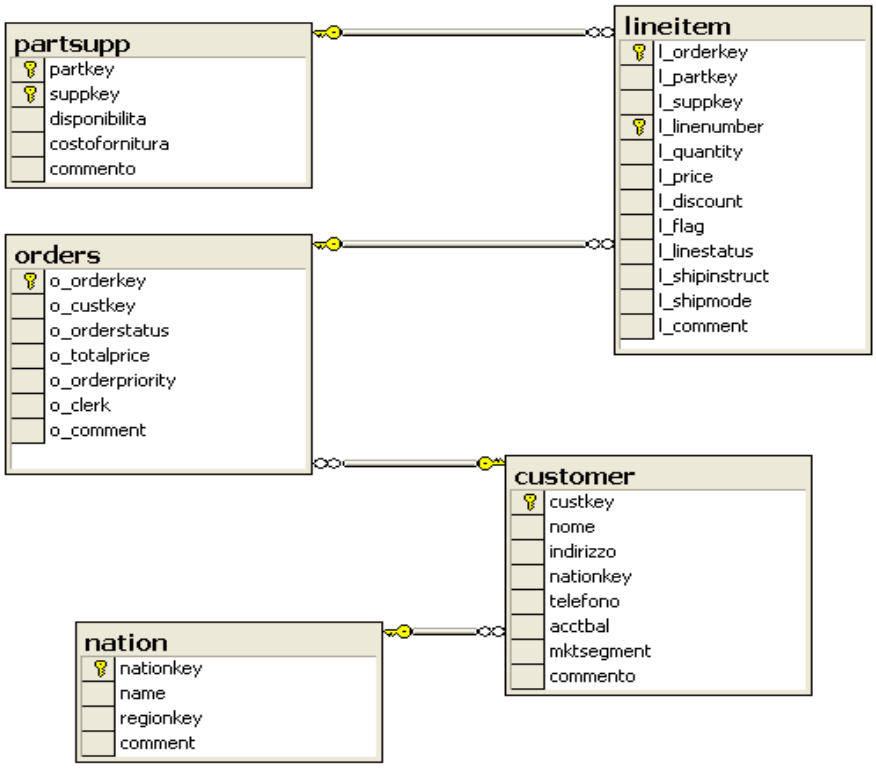

Figura 12. Diagramma database TPCH2

Attraverso la GUI di MOMIS, SI-Designer [18], vengono integrate le due sorgenti relazionali sopra riportate, viene costruito il Common Thesaurus e lo schema globale, su quest'ultimo saranno formulate delle Single class query (Scq) e Multiple class query (Mcq).

6.2 Testing delle performance

Lo schema globale, sul quale saranno formulate le query per valutare le performance delle due versioni del Query Manager è costituito dalle seguenti classi globali (Tabella 24):

Global Classes			
customer	***nation***	***orders***	***part***
c_acctbal	n_comment	o_clerk	p_brand
c_comment	n_name	o_comment	p_comment
c_custkey	n_nationkey	o_custkey	p_container
c_indirizzo	n_regionkey	o_orderkey	p_mfgr
c_mktsegment		o_orderpriority	p_name
c_nationkey		o_orderstatus	p_partkey
c_nome		o_totalprice	p_retailprice
c_telefono			p_size
			p_type
partsupp	***region***	***supplier***	***lineitem***
ps_comment	r_comment	s_acctbal	l_comment
ps_costofornitura	r_name	s_address	l_discount
ps_disponibilità	r_regionkey	s_comment	l_flag
ps_partkey		s_name	l_linenumber
ps_suppkey		s_nationkey	l_linestatus
		s_phone	l_orderkey
		s_suppkey	l_partkey
			l_price
			l_quantity
			l_shipinstruct
			l_shpimode
			l_suppkey

Tabella 24. Global Classes - Testing Performance

6.2.1 Single class query

Sullo schema globale sopra riportato saranno eseguite delle Single class query (query formulate su un'unica classe globale) e delle Multiple class query (query formulate su più classi globali). Per ogni query riporteremo il tempo di risposta, cioè il tempo che intercorre dall'esecuzione della query alla visualizzazione delle tuple che costituiscono il risultato, per le due versioni del Query Manager: versione con DBMS di supporto Microsoft SQL Server e versione con DBMS di supporto HSQLDB (che in seguito denoteremo rispettivamente con $QM_{SQLServer}$ e QM_{HSQLDB}).

Query 1

```
SELECT o_custkey, o_orderkey, o_orderpriority, o_comment
FROM orders
WHERE o_orderkey > 70
AND o_orderpriority = '1-URGENT'
AND o_totalprice > 56000
AND o_totalprice < 100000
```

Query 1	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	1.53	1.25

Query 2

```
SELECT ps_partkey, ps_suppkey, ps_costofornitura, ps_comment
FROM partsupp
WHERE ps_partkey != 4
AND ps_disponibilita > 300
AND ps_comment LIKE '%express%'
AND ps_costofornitura < 747.15
```

Query 2	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	6.15	2.78

Query 3

```
SELECT o_custkey, o_orderkey, o_orderstatus, o_clerk,
o_totalprice
FROM orders
WHERE o_orderstatus = 'O'
AND o_orderpriority = '3-MEDIUM'
```

```
AND o_totalprice > 25000
AND o_custkey > 5
AND o_custkey < 1500
```

Query 3	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	3.53	1.66

Query 4

```
SELECT l_linenumber, l_orderkey, l_price, l_linestatus,
l_quantity, l_shipmode, l_shipinstruct
FROM lineitem
WHERE l_price > 30000
AND l_flag = 'N'
AND l_shipmode = 'AIR'
AND l_quantity > 20
AND l_discount < 0.05
```

Query 4	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	6.20	1.36

Query 5

```
SELECT l_linenumber, l_orderkey, l_partkey, l_price,
l_linestatus, l_quantity, l_shipmode, l_shipinstruct
FROM lineitem
WHERE l_price > 10000
AND l_price < 89000
AND l_shipinstruct LIKE '%TAKE BACK%'
AND l_quantity > 9
AND l_linestatus = 'O'
```

Query 5	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	14.1	3.44

6.2.2 Multiple class query

Multiple class query formulate su 2 classi globali

Query 6

```
SELECT C.c_nome, C.c_telefono, C.c_indirizzo, O.o_orderstatus,
O.o_clerk, O.o_totalprice
FROM orders AS O, customer AS C
WHERE O.o_orderpriority = '2-HIGH'
AND O.o_orderkey > 10
AND C.c_nationkey != 3
AND O.o_custkey = C.c_custkey
```

Query 6	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	8.20	4.69

Query 7

```
SELECT O.o_orderstatus, O.o_orderpriority, O.o_comment,
L.l_discount, L.l_shipinstruct, L.l_shipmode, L.l_comment
FROM orders AS O, lineitem AS L
WHERE L.l_shipinstruct LIKE '%NONE%'
AND L.l_discount < 0.1
AND O.o_totalprice > 11000
AND O.o_comment LIKE '%carefully%'
AND O.o_orderkey = L.l_orderkey
```

Query 7	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	19.51	6.58

Query 8

```
SELECT P.ps_partkey, P.ps_suppkey, P.ps_costofornitura,
P.ps_disponibilita, L.l_shipinstruct, L.l_shipmode,
L.l_comment
FROM partsupp AS P, lineitem AS L
WHERE P.ps_disponibilita > 1000
AND  P.ps_costofornitura < 9870
AND L.l_shipmode = '%RAIL%'
AND P.ps_partkey = L.l_partkey
AND P.ps_suppkey = L.l_suppkey
```

Query 8	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	14.09	24.1

Query 9

```
SELECT PS.ps_partkey, PS.ps_suppkey, PS.ps_disponibilita,
S.s_name, S.s_phone, S.s_address
FROM partsupp AS PS, supplier AS S
WHERE PS.ps_disponibilita < 4000
AND S.s_nationkey != 24
AND S.s_comment LIKE '%express%'
AND PS.ps_suppkey = S.s_suppkey
```

Query 9	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	7.06	7.17

Query 10

```
SELECT C.c_nome, C.c_telefono, C.c_indirizzo, N.n_name,
N.n_comment
FROM customer AS C, nation AS N
WHERE N.n_nationkey != 2
AND C.c_custkey > 50
AND C.c_nationkey = N.n_nationkey
```

Query 10	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	3.81	3.39

Multiple class query formulate su 3 classi globali

Query 11

```
SELECT C.c_nome, C.c_telefono, N.n_name, R.r_name, R.r_comment
FROM customer AS C, nation AS N, region AS R
WHERE N.n_nationkey = 20
AND C.c_custkey < 3000
AND C.c_nationkey = N.n_nationkey
AND N.n_regionkey = R.r_regionkey
```

Query 11	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	3.61	3.75

Query 12

```
SELECT P.ps_partkey, P.ps_suppkey, P.ps_costofornitura,
P.ps_disponibilita, L.l_shipinstruct, L.l_shipmode, R.p_name,
R.p_type
FROM part AS R, partsupp AS P, lineitem AS L
WHERE P.ps_disponibilita > 1000
AND L.l_shipmode = '%AIR%'
AND P.ps_partkey = L.l_partkey
AND P.ps_suppkey = L.l_suppkey
AND R.p_partkey =  P.ps_partkey
```

Query 12	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	17.27	26.72

Query 13

```
SELECT C.c_nome, C.c_telefono, C.c_indirizzo, N.n_name,
N.n_comment, O.o_orderstatus, O.o_clerk, O.o_totalprice
FROM orders AS O, customer AS C, nation AS N
WHERE O.o_orderstatus = 'F'
AND O.o_orderkey > 80
AND C.c_nationkey != 5
AND O.o_custkey = C.c_custkey
AND C.c_nationkey = N.n_nationkey
```

Query 13	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	14.06	6.72

Query 14

```
SELECT C.c_nome, C.c_telefono, C.c_indirizzo,
O.o_orderpriority, O.o_clerk, O.o_totalprice, L.l_linenumber,
L.l_shipmode
FROM orders AS O, customer AS C, lineitem AS L
WHERE O.o_orderstatus = 'O'
AND O.o_totalprice > 10000
AND O.o_totalprice < 85000
AND L.l_partkey > 1000
AND O.o_custkey = C.c_custkey
AND O.o_orderkey = L.l_orderkey
```

Query 14	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	47.05	14.1

Query 15

```
SELECT P.ps_partkey, P.ps_suppkey, P.ps_costofornitura,
P.ps_disponibilita, L.l_shipinstruct, L.l_shipmode, S.s_name,
s.s_phone
FROM supplier AS S, partsupp AS P, lineitem AS L
WHERE P.ps_costofornitura < 8300
AND L.l_linenumber != 1
AND S.s_nationkey != 18
AND P.ps_partkey = L.l_partkey
AND P.ps_suppkey = L.l_suppkey
AND S.s_suppkey =  P.ps_suppkey
```

Query 15	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	97.95	51.41

Multiple class query formulate su 4 classi globali

Query 16

```
SELECT C.c_nome, C.c_telefono, C.c_indirizzo, N.n_name,
N.n_comment, O.o_orderstatus, O.o_clerk, O.o_totalprice
FROM orders AS O, customer AS C, nation AS N, region AS R
WHERE O.o_orderstatus = 'O'
AND O.o_orderkey > 50
AND C.c_comment LIKE '%regular%'
AND R.r_regionkey != 0
AND O.o_custkey = C.c_custkey
AND C.c_nationkey = N.n_nationkey
AND N.n_regionkey = R.r_regionkey
```

Query 16	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	12.45	5.91

Query 17

```
SELECT C.c_nome, C.c_telefono, C.c_indirizzo,
O.o_orderpriority, O.o_clerk, O.o_totalprice, L.l_linenumber,
L.l_shipmode, N.n_name
```

```
FROM orders AS O, customer AS C, lineitem AS L, nation AS N
WHERE O.o_orderpriority = '2-HIGH'
AND O.o_totalprice < 150000
AND L.l_shipmode = 'AIR'
AND O.o_custkey = C.c_custkey
AND O.o_orderkey = L.l_orderkey
AND C.c_nationkey = N.n_nationkey
```

Query 17	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	18.45	7.17

Query 18

```
SELECT P.ps_partkey, P.ps_suppkey, P.ps_costofornitura,
P.ps_disponibilita, L.l_shipinstruct, L.l_shipmode, R.p_name,
R.p_type, S.s_name, S.s_phone
FROM part AS R, partsupp AS P, lineitem AS L, supplier AS S
WHERE P.ps_disponibilita < 5500
AND L.l_flag = 'N'
AND S.s_nationkey != 5
AND P.ps_partkey = L.l_partkey
AND P.ps_suppkey = L.l_suppkey
AND R.p_partkey =  P.ps_partkey
AND S.s_suppkey = P.ps_suppkey
```

Query 18	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	53.16	24.19

Query 19

```
SELECT C.c_nome, C.c_telefono,O.o_orderkey, O.o_orderpriority,
O.o_totalprice, L.l_linenumber, L.l_quantity, N.n_name,
N.n_regionkey
FROM orders AS O, customer AS C, lineitem AS L, nation AS N
WHERE O.o_orderkey > 125
AND L.l_suppkey != 16
AND L.l_quantity < 40
AND O.o_custkey = C.c_custkey
AND O.o_orderkey = L.l_orderkey
AND C.c_nationkey = N.n_nationkey
```

Query 19	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	85.5	25.85

Query 20

```
SELECT C.c_nome, C.c_telefono, N.n_name, R.r_name, S.s_name,
S.s_phone
FROM customer AS C, nation AS N, region AS R, supplier AS S
WHERE N.n_regionkey != 0
AND C.c_custkey > 1000
AND C.c_nationkey = N.n_nationkey
AND N.n_regionkey = R.r_regionkey
AND C.c_nationkey = S.s_nationkey
```

Query 20	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	2.27	2.78

Multiple class query formulate su 5 classi globali

Query 21

```
SELECT P.ps_partkey, P.ps_suppkey, P.ps_costofornitura,
P.ps_disponibilita, L.l_shipinstruct, L.l_shipmode, R.p_name,
R.p_type, S.s_name, S.s_phone, O.o_totalprice
FROM part AS R, partsupp AS P, lineitem AS L, supplier AS S,
orders AS O
WHERE P.ps_costofornitura > 500
AND L.l_quantity > 20
AND S.s_nationkey != 5
AND O.o_orderpriority = '5-LOW'
AND P.ps_partkey = L.l_partkey
AND P.ps_suppkey = L.l_suppkey
AND R.p_partkey =  P.ps_partkey
AND S.s_suppkey = P.ps_suppkey
AND L.l_orderkey = O.o_orderkey
```

Query 21	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	63.77	25.19

Query 22

```
SELECT C.c_nome, C.c_telefono, N.n_name, R.r_name, S.s_name,
S.s_phone, O.o_totalprice
FROM customer AS C, nation AS N, region AS R, supplier AS S,
orders AS O
WHERE N.n_regionkey != 4
AND C.c_custkey < 1550
AND C.c_nationkey = N.n_nationkey
AND N.n_regionkey = R.r_regionkey
AND C.c_nationkey = S.s_nationkey
AND C.c_custkey = O.o_orderkey
```

Query 22	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	24.12	9.25

Query 23

```
SELECT C.c_nome, C.c_telefono,O.o_orderkey, O.o_orderpriority,
O.o_totalprice, L.l_shipinstruct, L.l_quantity, N.n_name,
N.n_regionkey, R.r_name
FROM orders AS O, customer AS C, lineitem AS L, nation AS N,
region AS R
WHERE O.o_totalprice > 70000
AND O.o_totalprice < 150000
AND L.l_suppkey != 16
AND L.l_linestatus = 'F'
AND O.o_custkey = C.c_custkey
AND O.o_orderkey = L.l_orderkey
AND C.c_nationkey = N.n_nationkey
AND N.n_regionkey = R.r_regionkey
```

Query 23	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	51.1	16.03

Query 24

```
SELECT P.ps_partkey, P.ps_suppkey, P.ps_costofornitura,
P.ps_disponibilita, L.l_shipinstruct, L.l_shipmode, R.p_name,
R.p_brand, S.s_name, S.s_phone, O.o_totalprice,
O.o_orderpriority
```

```
FROM part AS R, partsupp AS P, lineitem AS L, supplier AS S,
orders AS O
WHERE L.l_shipinstruct LIKE '%DELIVER%'
AND L.l_quantity > 20
AND R.p_partkey != 7
AND O.o_custkey  > 300
AND P.ps_partkey = L.l_partkey
AND P.ps_suppkey = L.l_suppkey
AND R.p_partkey =  P.ps_partkey
AND S.s_suppkey = P.ps_suppkey
AND L.l_orderkey = O.o_orderkey
```

Query 24	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	58.39	39.25

Query 25

```
SELECT C.c_nome, C.c_telefono,O.o_orderkey, O.o_orderpriority,
O.o_totalprice, L.l_shipinstruct, L.l_quantity, N.n_name,
N.n_regionkey, R.r_name
FROM orders AS O, customer AS C, lineitem AS L, nation AS N,
region AS R
WHERE O.o_totalprice > 50000
AND O.o_orderpriority = '1-URGENT'
AND L.l_partkey != 29
AND L.l_shipmode = 'SHIP'
AND R.r_regionkey = 3
AND O.o_custkey = C.c_custkey
AND O.o_orderkey = L.l_orderkey
AND C.c_nationkey = N.n_nationkey
AND N.n_regionkey = R.r_regionkey
ORDER BY  C.c_nome
```

Query 25	$QM_{SQLServer}$	QM_{HSQLDB}
Tempo di risposta (s)	18.6	8.13

6.3 Considerazioni sul testing delle performance

Analizzando i tempi di risposta sopra riportati possiamo affermare che i tempi di risposta per il QM_{HSQLDB} sono inferiori per l'80% delle query, i tempi di risposta possono essere considerati "uguali" (la differenza è impercettibile per l'utente) per il 12% delle query e solo per l'8% delle query il $QM_{SQLServer}$ ha tempi di risposta inferiori (Figura 13). Nel grafico di Figura 14 riportiamo per ciascuna tipologia di query i tempi di riposta medi di esecuzione per le due versioni del Query Manager. Per tutte le tipologie di query la versione del Query Manager che utilizza come DBMS di supporto HSQLDB presenta tempi di risposta inferiori.

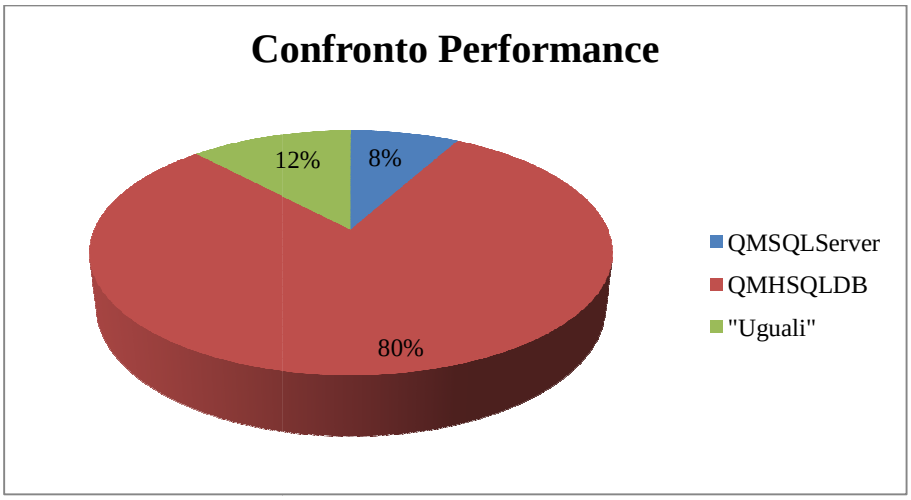

Figura 13. Confronto Performance

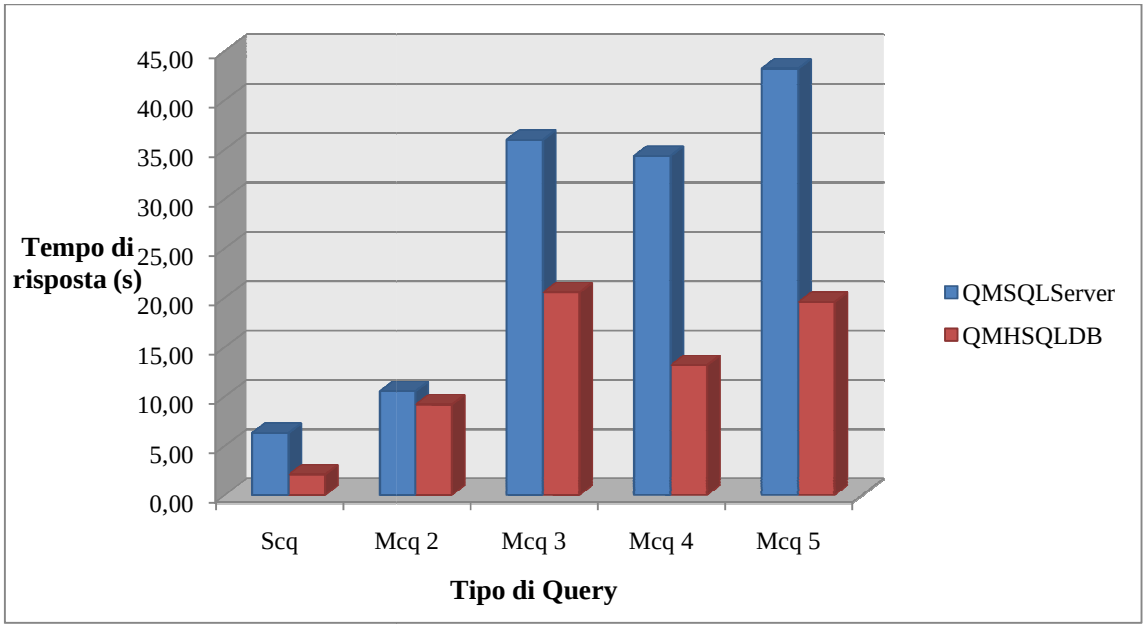

Figura 14. Tempi di Risposta

Capitolo 7

7 Progettazione e implementazione di un framework di testing

7.1 JUnit testing framework

In tutti gli ambienti di sviluppo, qualsiasi sia il processo di produzione del codice adottato, uno degli aspetti considerati poco importanti è la fase di testing. Spesso sottovalutata per mancanza di tempo o per assenza di volontà da parte degli sviluppatori, generalmente avviene subito prima della release, quindi quando ormai le scadenza di consegna sono prossime. Una buona norma che ogni sviluppatore dovrebbe adottare è quello di eseguire test di unità per assicurarsi che la singola unità di sviluppo assolva le sue funzioni e sia conforme ai requisiti. Questo è uno dei più importanti passaggi per poter avere un prodotto software affidabile. Un modo comune adottato dai progettisti Java è quello di definire una classe che contiene al suo interno il metodo main in cui vengono istanziati degli oggetti e viene valutato il comportamento di quest'ultimi, se conforme a quello desiderato. Questa pratica ha delle evidenti limitazioni:

- ✓ Non esiste un concetto esplicito che dimostri che il test ha avuto successo o ha fallito;
- ✓ Dopo l'esecuzione del test bisogna analizzare e valutare i risultati;
- ✓ Non c'è un metodo per aggregare i risultati.

Tutte queste limitazioni sono state superate dal framework di testing, JUnit[1]. JUnit rappresenta lo standard de facto per il testing delle applicazioni Java, è stato sviluppato da Kent Beck e Erich Gamma [34], e consente agli sviluppatori Java di creare in modo molto veloce e semplice dei Java test. Questo framework è stato utilizzato, durante il lavoro di tesi, per testare il Query Manager del sistema MOMIS.

1 http://www.junit,org

Un unit test in Java consiste nel valutare il funzionamento di un metodo, in base ai valori attesi. Un unit test può avere tre differenti esiti:

- successo (pass): il test è stato eseguito correttamente e l'output coincide con quello atteso;
- fallimento (failure): il test è stato eseguito correttamente, ma l'output non coincide con il valore atteso;
- errore (error): si è verificato un errore che può essere stato provocato dal codice del oggetto del test o dal codice del test stesso.

In JUnit per identificare i metodi di test vengono utilizzate le annotazioni, che vengono valutate a tempo di esecuzione. Alcune delle annotazioni disponibili sono riportate in Tabella 25:

Annotazione	***Descrizione***
@Test public void method()	Per annotare i metodi di test.
@Test (expected = java.lang.Exception.class) public void method()	Per annotare un metodo che dovrebbe sollevare un'eccezione specifica (identificata dal parametro java.lang.Exception.class). Il test fallisce se non viene sollevata l'eccezione.
@Test (timeout =) public void method()	Per annotare un metodo la cui esecuzione non dovrebbe superare una certa soglia di tempo specificata nel parametro timeout.
@Before public void method()	Per annotare un metodo la cui esecuzione avviene prima dell'esecuzione di ogni altro test (es. l'apertura di connessione ad un database).
@After public void method()	Per annotare un metodo la cui esecuzione avviene dopo l'esecuzione di tutti i test (es. chiusura di risorse aperte in precedenza).
@BeforeClass public static void method()	Per annotare un metodo che sarà eseguito all'inizio dell'esecuzione del test.
@AfterClass public static void method()	Per annotare un metodo che sarà eseguito dopo la fine dell'esecuzione del test.
@Ignore public void method()	Per annotare un metodo di test che non vogliamo sia eseguito.

Tabella 25. JUnit - Annotazioni

Il framework JUnit mette a disposizione dei metodi assert, metodi statici contenuti nella classe org.junit.Assert che effettuano una semplice comparazione tra il risultato atteso ed il risultato dell'esecuzione. Alcuni metodi messi a disposizione dal framework JUnit sono riportati in Tabella 26:

Metodo	***Descrizione***
assertTrue(boolean condition)	Controlla la boolean condition, se true il test ha successo altrimenti il test fallisce.
assertFalse(boolean condition	Controlla la boolean condition, se false il test ha successo altrimenti il test fallisce.
assertEquals(expected, actual)	Confronta i due parametri expected e actual. Il test ha successo se i due parametri sono uguali.
assertNull(object)	Serve per testare un metodo che dovrebbe ritornare null.
assertNotNull(object)	Serve per testare un metodo che non dovrebbe ritornare null.
assertSame(expected, actual)	Confronta i due parametri expected e actual. Il test ha successo se expected e actual fanno riferimento allo stesso oggetto.
assertNotSame(expected, actual)	Confronta i due parametri expected e actual. Il test ha successo se expected e actual non fanno riferimento allo stesso oggetto.

Tabella 26. JUnit - Metodi Assert

7.2 Sviluppo di un framework per il testing del Query Manager

Durante il lavoro di tesi è stato utilizzato il framework JUnit per eseguire tutti i test descritti nei capitoli precedenti. Per effettuare un test sul Query Manager sono necessari i seguenti tre elementi:

- Le sorgenti: dall'integrazione delle sorgenti si ottiene lo schema globale (*GS*);
- Lo schema globale (*GS*);
- Le query: formulate in linguaggio OQL_I[3] sullo schema globale.

Le sorgenti utilizzate durante il lavoro di tesi per il testing del Query Manager sono state create nel *RDBMS Microsoft SQL Server.* Successivamente integrate tramite l'ausilio della GUI di MOMIS, SI-Designer viene generato lo schema globale (*GS*) sul quale vengono formulate le query riportate nei capitoli precedenti. Per ogni query (l'output viene visualizzato sullo standard output) è necessario analizzare l'output per valutare la correttezza di esecuzione. Questo rende i test molto "statici", cioè non è possibili eseguirli se non si hanno ha disposizione le stesse sorgenti, che in più devono essere create nello stesso DBMS (nel nostro caso). Per poter distribuire i test, in modo che chiunque li possa eseguire indipendentemente se ha disposizione le sorgenti o il DBMS su cui devono essere create le sorgenti dati, è stato progettato e implementato un framework/package per il testing del Query Manager. Il framework di testing si basa sul Java framework JUnit, l'idea è quella di eseguire query su uno schema definito, ottenuto dall'integrazione di sorgenti che verranno generate a runtime. L'output di esecuzione delle query verrà confrontato con l'output atteso, se i valori coincidono il test ha successo, altrimenti fallisce. Quindi per effettuare un test con il framework di testing sono necessari quattro elementi:

- Le sorgenti;
- Lo schema globale (*GS*);
- Le query;
- L'output atteso dall'esecuzione delle query.

Per descrivere il framework facciamo riferimento ad un esempio. Supponiamo di avere due sorgenti unfolding/Test1 e unfolding/Test2, lo schema globale (sotto forma di un file xml) che rappresenta l'integrazione delle due sorgenti, ed un insieme di query q01, q02 … formulate su questo schema. Per ogni sorgente, in una particolare

cartella, deve essere creato un file di testo (SQL - DDL) che deve contenere gli statement `Create Cached Table e Insert Into <table_name> Values` (nel nostro esempio vengono creati due file di testo Test1.sql e Test2.sql). Questi file verranno letti a runtime, per ogni sorgente su un database del database engine H2[1] (Java Database Engine, Open Source) verranno create le tabelle e in queste tabelle saranno inserite le tuple specificate nel file *.sql. In un'altra cartella deve essere creata una classe java (JUnit class) e una sottocartella che deve avere lo stesso nome della classe. In questa sottocartella devono essere creati dei file di testo, due per ogni query da testare (q01.oql e q01.out), il file q*.oql deve contenere la query in linguaggio OQL[3] da eseguire mentre il file q*.out deve contenere l'output atteso dall'esecuzione della query (il file con estensione .out deve avere lo stesso nome del file che contiene la query a cui si riferisce). Quando una sorgente viene inizializzata il framework crea un file *db_qtf_initOk.txt* nel database directory, in questo modo se una sorgente viene utilizzata diverse volte, l'inizializzazione avviene una volta sola. Quindi prima di inizializzare una sorgente viene controllato l'esistenza del file *db_qtf_initOk.txt*, se il file esiste la sorgente è già stata inizializzata in precedenza, se il file non esiste viene cancellato il database e rigenerato tramite l'esecuzione degli statement contenuti nel file *.sql.

Vediamo in seguito i metodi contenuti nella JUnit class:

```
public class Unfolding extends BaseQTFTest {

@Before
        public void beforeTests()throws Exception{
              initSources();
              initQueryManager();
        }

@Test
   public void test1() throws Exception {

          boolean resQuery = runQueryManagerQuery("q01");
         assertTrue(resQuery);

   }

   public void initSources() throws Exception {
            initSource("unfolding/Test1");
            initSource("unfolding/Test2");
         }
    }
```

1 www.h2database.com

Come possiamo vedere il codice del metodo di test (annotato con @Test) è molto semplice, viene eseguito il metodo `runQueryManagerQuery()`(implementato nella classe *BaseQTFTest.java*) al quale viene passato come parametro il nome del file (q01) che contiene la query da testare sul query manager. Questo metodo ritorna un boolean resQuery, se l'output di esecuzione della query q01 coincide con l'output atteso (inserito nel file q01.out) il metodo ritorna true, altrimenti ritorna false. Il valore di ritorno del metodo `runQueryManagerQuery()`viene passato al metodo messo a disposizione dal framework JUnit, `assertTrue()`. Per definizione del metodo `assertTrue`(), se resQuery = `true` il test ha successo (l'output di esecuzione della query coincide con quello expected), mentre se resQuery = `false` il test fallisce.

Ogni JUnit class estende una classe *BaseQTFTest.java* nella quale sono definiti i seguenti metodi: *initSource()*, *initQueryManager()*, *getSchemaAbsolutePath(), runQueryManagerQuery()*, che verranno descritti in seguito.

Il metodo getSchemaAbsolutePath()

Una query viene formulata su uno schema (in formato xml), all'atto di creazione di un oggetto Query Manager, dobbiamo passare come parametro il path assoluto dello schema globale. Il metodo `getSchemaAbsolutePath()` (il codice viene riportato in seguito) ritorna il path assoluto dello schema globale di riferimento per la query di testing.

```
public String getSchemaAbsolutePath(){
String schemaName = getTestDir() + "/schema.xml";
File   schemaFile = new File(schemaName);
String schemaAbsolutePath  = schemaFile.getAbsolutePath();
String schemaNameAbsolutePath =
Tools.stringReplaceInString(schemaAbsolutePath, "\\", "/");
       return schemaNameAbsolutePath;
   }
```

Il metodo initQueryManager()

Il metodo `initQueryManager()`(il codice viene riportato in seguito) inizializza il Query Manager. Viene istanziato un oggetto *configuration*, della classe *QueryManagerConfiguration.java* e vengono invocati i metodi necessari per impostare i parametri di connessione al database engine *HSQLDB*. Viene istanziato

un oggetto *queryManager*, della classe *QueryManagerImpl.java*; il costruttore della classe *QueryManagerImpl.java* prevede che li venga passato come parametro il path assoluto dello schema globale per questo la creazione dell'oggetto *queryManager* deve avere la seguente espressione:

```
QueryManagerImpl queryManager=
new QueryManagerImpl(this.getSchemaAbsolutePath());
```

Su questo oggetto invochiamo:

- ✓ Il metodo `setConfiguration()` al quale viene passato come parametro l'oggetto *configuration*, affinché vengano utilizzati come parametri di connessione quelli precedentemente impostati.
- ✓ Il metodo `setSchemaFileAbsolutePath()`, al quale viene passato come parametro `this.getSchemaAbsolutePath()`, cioè il metodo che ritorna il path assoluto dello schema globale di riferimento descritto in precedenza.
- ✓ Il metodo `setJoinEngineFactory()`, al quale viene passato come parametro un oggetto della classe *JoinEngineFactory.java*. La classe *JoinEngineFactory.java* implementa i metodi per inizializzare il pool di Join Engine necessari per l'esecuzione della query.

Infine sull'oggetto invochiamo il metodo `init()`, per inizializzare il server HSQLDB e creare la cartella che conterrà i file (vedere 3.2) relativi al database querymanager (su cui vengono create le tabelle temporanee necessari per l'elaborazione della query globale).

```
public void  initQueryManager() throws Exception {

   QueryManagerConfiguration configuration=new QueryManagerConfiguration();
       configuration.setInnerBDMS(SupportedDBMS.HSQLDB);

configuration.setInnerDbmsWorkDirectory("hsqldb/data/querymanager/querymana
ger");
         configuration.setInnerDbmsDriver("org.hsqldb.jdbcDriver");

configuration.setInnerDbmsConnectionURL("jdbc:hsqldb:hsql://localhost/xdb")
;
         configuration.setInnerDbmsConnectionUser("sa");
         configuration.setInnerDbmsConnectionPassword("");
         configuration.setInnerDbmsSchema("xdb");

        QueryManagerImpl queryManager= new
QueryManagerImpl(this.getSchemaAbsolutePath());
         System.out.println("Schema absolute Path " +
getSchemaAbsolutePath());
         queryManager.setConfiguration(configuration);
```

```
queryManager.setSchemaFileAbsolutePath(this.getSchemaAbsolutePath()) ;
        joinenginefactory = new JoinEngineFactory();
        queryManager.setJoinEngineFactory(joinenginefactory);
        queryManager.init();

}
```

Il metodo initSource()

Il metodo `initSource()`inizializza le sorgenti a tempo di esecuzione. Le sorgenti vengono create sul database engine H2. Per ogni sorgente viene creato un database e su questo database vengono create le tabelle specificate nel file *.sql relativo a quella sorgente. Il file viene letto a runtime e gli statement SQL `Create Cached Table` e `Insert Into <table_name> Values` vengono eseguiti sul database engine.

Il metodo runQueryManagerQuery()

Il metodo `leggiFile()`, un metodo statico implementato nella classe *Tools.java*, legge il file che li viene passato come parametro e memorizza il contenuto del file in una stringa. A questo metodo li vengono passati come parametri il file che contiene la query OQL_I[3] (q*.oql) e il file che contiene l'output atteso dall'esecuzione della query (q*.out):

```
String oql = Tools.leggiFile(fname);
String expectedOut = Tools.leggiFile(fnameExpectedOut);
```

In output avremmo due stringhe `oql` (query oql) e `expectedOut` (output atteso). Le stringhe `oql` e `expectedOut` e la stringa `testId` (`testId` identifica la query, e coincide con il nome del file che contiene la query) vengono passate in input al metodo `runQueryManagerQuery()`(il codice viene riportato in seguito). Il metodo `runQueryManagerQuery()`esegue la query `oql` (sull'oggetto *queryManager* viene invocato il metodo `executeQuery(oql)`che ritorna in output un oggetto ResultSet) e confronta l'output della query con la stringa expected.

```
boolean resQuery;
resQuery =(expectedOut.equals(result));
return resQuery;
```

Se `result` e `expectedOut` sono uguali il boolean `resQuery` sarà uguale a *true* altrimenti sarà uguale a `false`. Infine sul oggetto queryManager viene invocato il metodo `destroy()`per effettuare lo SHUTDOWN del server HSQLDB.

```
public boolean runQueryManagerQuery(String oql, String expectedOut, String
testId) throws Exception{
      ResultSet rs=queryManager.executeQuery(oql);

      String result="";
      ResultSetMetaData rsmd = rs.getMetaData();
      int numColonne = rsmd.getColumnCount();


      while (rs.next()) {

            for (int i=1; i<=numColonne; i++){
                  result=result+rs.getString(i);
            }
      }
      rs.close();
      queryManager.destroy();
      String s = ""
        + "\nRunning query: ["+testId+"]"
        + "\n oql: ["+oql+"]"
        + "\n expected out:[\n"+expectedOut+"]"
        + "\n result :[ \n" + result + "]"
        ;
     System.out.println(s);
     boolean resQuery;
     resQuery =(expectedOut.equals(result));
     return resQuery;
      }
```

Conclusioni

In tutti gli ambienti di sviluppo, qualsiasi sia il processo di produzione del codice adottato, uno degli aspetti considerati poco importanti è la fase di testing. Invece una buona norma che ogni sviluppatore dovrebbe adottare è quella di eseguire test di unità per assicurarsi che la singola unità di sviluppo assolva le sue funzioni e sia conforme ai requisiti. Questo è uno dei più importanti passaggi per poter avere un prodotto software affidabile. Kent Beck, uno degli sviluppatori del framework JUnit ha definito un nuovo ciclo di sviluppo di un prodotto software, il TDD (Test-Driven Development) [35], che si può riassumere "scrivere i test prima del codice", questo assicura un elevata qualità del codice.

In questo lavoro di tesi è stato trattato il sistema di Data Integration, a Mediatore *MOMIS (Mediator EnvirOment for Multiple Information Sources)* ed è stato descritto in modo dettagliato il modulo Query Manager (Capitolo 2), che si occupa del query processing. E' stata definita la sintassi del linguaggio OQL_I^3 accettata dal *Query Manager* del sistema *MOMIS* (Appendice A) e tramite l'utilizzo del framework JUnit è stato eseguito il testing delle funzionalità del componente Query Manager che utilizza come DBMS di supporto *HSQLDB* (Capitolo 3 e Capitolo 4), sono state effettuate delle modifiche del modulo software per risolvere i diversi problemi emersi durante la fase di testing. E' stato descritto il benchmak pubblico *THALIA* e la modalità con cui *MOMIS* riesce a risolvere i diversi tipi di eterogeneità intrinseche nelle query del benchmark e fornire una risposta alle query (Capitolo 5). Nel Capitolo 6 vengono confrontati i tempi di risposta alle query formulate su uno schema globale, per le due versioni del *Query Manager* del sistema MOMIS: la versione che utilizza come DBMS di supporto al query processing *Microsoft SQL Server* e la versione che utilizza *HSQLDB* come DBMS di supporto. Per consentire la distribuzione dei test effettuati, in modo che chiunque li possa eseguire indipendentemente se ha disposizione le sorgenti o il DBMS su cui devono essere create le sorgenti dati, è stato progettato e implementato un framework/package per il testing del Query Manager. Il framework di testing si basa sul Java framework JUnit, l'idea è quella di eseguire query su uno schema definito, ottenuto dall'integrazione di sorgenti che verranno generate a runtime. L'output di esecuzione delle query verrà confrontato con l'output atteso, se i valori coincidono il test ha successo, altrimenti fallisce.

Concludendo, il progetto svolto è stato molto interessante e particolarmente stimolante in quanto mi ha offerto la possibilità di usare strumenti ed approfondire argomenti del tutto nuovi.

Appendice A

In seguito viene riportata la sintassi del linguaggio OQL_I^3 accettata dal Query Manager del sistema MOMIS:

<asterisk> ::= *

<comma> ::= ,

<period> ::= .

<quote> ::= '

<underscore> ::= _

<equals operator> ::= =

<not equals operator> ::= !=

<less than operator> ::= <

<less than or equals operator> ::= <=

<greater than operator> ::= >

<greater than or equals operator> ::= >=

<left paren> ::= (

<right paren> ::=)

<query> ::=

SELECT [<quantifier>] <select list>

<from clause>

[<where clause>]

[<order by clause>]

[<group by clause>]

[<having clause>]

<quantifier> ::= DISTINCT

<select list> ::=

<asterisk>

| <select sublist> [{ <comma> <select sublist> }...]

<select sublist> ::= <attribute reference>

<attribute reference> ::= [<qualifier> <period>] <global attribute name>

<qualifier> ::= <global interface name> | <correlation name>

<global interface name> ::= <identifier>

<global attribute name> ::= <identifier>

<from clause> ::=

FROM <global interface reference> [{ <comma> <global interface reference> }...]

<global interface reference> ::= <global interface name> [<correlation specification>]

<correlation specification> ::= AS <correlation name>

<correlation name> ::= <identifier>

<where clause> ::= WHERE <search condition>

<search condition> ::=

<boolean term>

| <search condition> OR <boolean term>

<boolean term> ::=

<boolean factor>

| <boolean term> AND <boolean factor>

<boolean factor> ::= <boolean primary>

<boolean primary> ::=

<predicate>

| <left paren> <search condition> <right paren>

<predicate> ::=

<comparison predicate>

| <like predicate>

| <null predicate>

| <join predicate>

<comparison predicate> ::=

<attribute reference> <comp op> <value expression>

<comp op> ::=

<equals operator>

| <not equals operator>

| <less than operator>

| <greater than operator>

| <less than or equals operator>

| <greater than or equals operator>

<null predicate> ::= [NOT] IS NULL <attribute reference>

<like predicate> ::= <attribute reference> LIKE <pattern>

<pattern> ::= <quote> <value expression> <quote>

<join predicate> ::=

<attribute reference> <comp op> <attribute reference>

<order by clause> ::= ORDER BY <sort specification list>

<sort specification list> ::=

<sort specification> [{ <comma> <sort specification> }...]

<sort specification> ::= <sort key> [<ordering specification>]

<sort key> ::= <attribute reference>

<ordering specification> ::= ASC | DESC

<group by clause> ::= GROUP BY <grouping attribute reference list>

<grouping attribute reference list> ::=

<grouping attribute reference> [{ <comma> <grouping attribute reference> }...]

<grouping attribute reference> ::= <attribute reference>

<having clause> ::= HAVING <search condition>

Bibliografia

[1] ARPA: Advanced Research Project Agency
I^3 Project: http://mole.dc.isx.com/I3/

[2] G. Wiederhold: "*Intelligent integration of information*", In P. Buneman and S. Jajodia, editors, SIGMOD Conference, pages 434–437. ACM Press, 1993.

[3] J.Bleiholder, F. Naumann: "*Data fusion*", ACM Computing Surveys (CSUR), Volume 41 , Issue 1 (December 2008) Article No.1.

[4] D. Beneventano, S. Bergamaschi, A. Corni, M. Vincini: "*MOMIS: un sistema di Description Logics per l'integrazione del sistema informativo d'impresa*", XXXVIII Congresso Annuale AICA2000- Taormina, 27-30 Settembre 2000.

[5] S. Bergamaschi, D. Beneventano, A. Corni, M. Vincini: "*Creazione di una vista globale d'impresa con il sistema MOMIS basato su Description Logics*", AI*IA Notizie Journal, Year XIII, N. 2, June 2000.

[6] H. Garcia-Molina, J. Ullman , J. Widom: *"Database Systems: The Complete Book"* , Prentice Hall.

[7] S. Bergamaschi, S. Castano, M. Vincini: "*Semantic Integration of Semistructured and Structured Data Sources*", SIGMOD Record Special Issue on Semantic Interoperability in Global Information, Vol. 28, No. 1, March 1999.

[8] MOMIS Home Page: http://www.dbgroup.unimo.it/Momis/

[9] S. Bergamaschi, S. Castano, D. Beneventano, M. Vincini: "*Semantic Integration of Heterogeneous Information Sources*", Special Issue on Intelligent Information Integration, Data & Knowledge Engineering, Vol. 36, Num. 1, Pages 215-249, Elsevier Science B.V. 2001.

[10] D. Beneventano, S. Bergamaschi, S. Castano, V. De Antonellis, A. Ferrara, F. Guerra, F. Mandreoli, G. C. Ornetti, M. Vincini: "*Semantic Integration and Query Optimization of Heterogeneous Data Sources*". International Conference on Object Oriented Information Systems - Invited paper at OOIS Workshops 2002: 154-165.

[11] D. Beneventano, S. Bergamaschi, Claudio Sartori, Maurizio Vincini: "*ODB-Tools: a description logics based tool for schema validation and semantic query optimization in Object Oriented Databases*", Proceeding of the Fifth Conference of the Italian Association for Artificial Intelligence (AI*IA97): Advance in Artificial Intelligence, LNAI Vol. 1321, Springer. Berlin, September 17-19, 1997.

[12] ODB-Tools Home Page: http://www.dbgroup.unimo.it/ODB-Tools.html

[13] D. Beneventano, S. Bergamaschi, C. Sartori, M. Vincini: "*ODB-QOptimizer: a tool for semantic query optimization in OODB*", Int. Conference on Data Engineering ICDE97, Birmingham, UK, April 1997.

[14] D. Beneventano, S. Bergamaschi, C. Sartori: "*Description Logics for Semantic Query Optimization in Object-Oriented Database Systems*", ACM Transaction on Database Systems, Volume 28: 1-50 (2003).

[15] D. Beneventano, S. Bergamaschi, C. Sartori: *"Semantic Query Optimization by Subsumption in OODB*", Proceedings of the Int. Workshop on Flexible Query-Answering Systems (FQAS96), Roskilde, Denmark, may 1996, pp. 167-185.

[16] Object Data Management Group: http://www.odbms.org/ODMG/

[17] Standard ODMG 3.0: http://www.service architecture.com/database/articles/odmg_3_0.html

[18] D. Beneventano, S. Bergamaschi, A. Corni, R. Guidetti, G. Malvezzi: *"SI-DESIGNER: un tool di ausilio all'integrazione intelligente di sorgenti di informazione"*, Convegno su Sistemi Evoluti per Basi di Dati (SEBD2000), L'Aquila, Giugno 2000.

[19] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini: "*Synthesizing an integrated ontology*", IEEE Internet Computing, 7(5):42–51, 2003.

[20] M. Orsini PHD Thesis: "*Query Management in Data Integration System: the MOMIS approach*".

[21] R. C. Nana Mbinkeu: "*FULL Outer Join Optimization Techniques in Integration Information Systems*", 6e Manisfestation des Jeunes Chercheurs en Sciences et Technologies de l'information et de la Communication, Octobre 2008, Marseille, France.

[22] The Evolving Open-Source Software Model: http://mediaproducts.gartner.com/reprints/actuate/vol2/article1/article1.html

[23] HSQLDB Home Page: www.hsqldb.org

[24] Tesi di laurea di Franceso Nigro: "*DataRiver: un sistema di integrazione dati Open Source"*.

[25] Bruce Perence, Cyber Security Policy Research Institute, George Washington University: "*The Emerging Economic Paradigm of Open Source*".

[26] J. Hammer, M. Stonebraker, and O. Topsakal: *"THALIA : Test Harness for the Assessment of Legacy Information Integration Approaches"*, Technical Report TR05-001, Dept. of Computer Science and Information and Eng., Univ. of Florida, January 2005.

[27] D. Beneventano, S. Bergamaschi, M. Vincini, M. Orsini, R.C. Nana Mbinkeu: "*Getting Through the THALIA Benchmark with MOMIS"*, VLDB 2007 - Third International Workshop on Database Interoperability (InterDB 2007), Vienna, Austria, September 24, 2007.

[28] D. Beneventano, S. Bergamaschi: *"The MOMIS Methodology for Integrating Heterogeneous Data Sources"*, IFIP World Computer Congress. Toulouse France, 22-27 August 2004.

[29] F. Naumann, M. Haussler: "*Declarative data merging with conflict resolution*". In C. Fisher and B. N. Davidson, editors, IQ, pages 212–224. MIT, 2002.

[30] R. Benassi, D. Beneventano, S. Bergamaschi, F. Guerra, M. Vincini: "*Synthesizing an Integrated Ontology with MOMIS*", International Conference on Knowledge Engineering and Decision Support (ICKEDS). Porto, Portugal, 21-23 July 2004.

[31] S. Bergamaschi, F. Guerra, F. Mandreoli, M. Vincini: "*Working in a dynamic enviroment: the NeP4B approach as a MAS*", In proceedings of AP2PC 2009, May 11 2009, Budapest, Hungary.

[32] M. Lenzerini. Data integration: a theoretical perspective. In PODS '02, pages 233–246, 2002.

[33] Tesi di laurea di Stefania Bruschi: "*Dinamica delle Ontologie: inserimento di una nuova sorgente nel sistema MOMIS*".

[34] Elliotte Rusty Harold: "*An early look at JUnit 4*" http://www.ibm.com/developerworks/java

[35] K. Beck: "*Test-Driven Development by Example*", Addison Wesley, 2003

Ringraziamenti

Desidero ringraziare anzitutto la Professoressa Sonia Bergamaschi per l'aiuto fornitomi durante la tesi e per la disponibilità dimostrata.

Un ringraziamento particolare va all'Ing. Mirko Orsini per la sua disponibilità e per il suo prezioso aiuto.

Un ringraziamento molto sentito va agli ingegneri Daniele Miselli, Matteo Di Gioia, Fabrizio Orlandi e Alberto Corni per la loro disponibilità e per il loro costante aiuto fornitomi durante lo svolgimento della tesi.

Esprimo tutta la mia gratitudine ai miei genitori per il loro costante supporto nell'intero arco dei miei studi, per la fiducia e per tutti i sacrifici che hanno fatto per me in questi anni.

Grazie anche al mio fidanzato e a mio fratello per il supporto e per l'affetto che mi hanno sempre dimostrato.

Un ringraziamento va a tutti i miei amici per il sostegno e per la forza che mi hanno dato durante questi anni e per avermi supportato e "sopportato" in questo periodo.

Grazie di cuore a tutti…

Entela Kazazi